UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

Departamento de Arquitectura de Computadores y Automática

TESIS DOCTORAL

Mecanismos de compensación de variabilidad en memorias a nivel de sistema

Variability compensation mechanisms for system-level memory management

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Concepción Sanz Pineda

Directores

Manuel Prieto Matías
José Ignacio Gómez Pérez
Christian Tenllado van der Reijden

Madrid, 2012

UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE INFORMÁTICA
Departamento de Arquitectura de Computadores y Automática

Mecanismos de compensación de variabilidad en memorias a nivel de sistema

Variability compensation mechanisms for system-level memory management

Tesis Doctoral

Concepción Sanz Pineda

Madrid, 2012

UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE INFORMÁTICA
Departamento de Arquitectura de Computadores y Automática

Mecanismos de compensación de variabilidad en memorias a nivel de sistema

Variability compensation mechanisms for system-level memory management

MEMORIA PARA OPTAR AL GRADO DE DOCTOR
PRESENTADA POR

Concepción Sanz Pineda

Directores:
Manuel Prieto Matías
José Ignacio Gómez
Christian Tenllado van der Reijden

Madrid, 2012

Mecanismos de compensación de variabilidad en memorias a nivel de sistema

Memoria presentada por Concepción Sanz Pineda para optar al grado de Doctor con Mención Europea por la Universidad Complutense de Madrid, realizada bajo la dirección de los doctores Manuel Prieto Matías, José Ignacio Gómez y Christian Tenllado van der Reijden.

Madrid, 2012

Variability compensation mechanisms for system-level memory management

PhD dissertation presented by Concepción Sanz Pineda in fulfilment of the requirements for the degree of Doctor of Philosophy with European Mention to the Universidad Complutense de Madrid, promoted by Dr. Manuel Prieto Matías, Dr. José Ignacio Gómez and Dr. Christian Tenllado van der Reijden.

Madrid, 2012

Esta investigación ha sido llevada a cabo gracias a la financiación de la Comisiø'n Interministerial de Ciencia y Tecnología, a través de los proyectos TIN2005-5619, CICYT-TIN 2008/00508 e Ingenio 2010 Consolider CSD00C-2007-20811. Además, ha contado con la financiación de la Red Europea de Excelencia HiPEAC y de la Unión Europea por medio de las becas Marie Curie.

This research has been supported by the Spanish goverment through the research contracts TIN2005-5619, CICYT-TIN 2008/00508 and Ingenio 2010 Consolider CSD00C-2007-20811. It has also supported by the HiPEAC European Network of Excellence and the Marie Curie Fellowship of the European Community.

A mi familia

Acknowledgments/Agradecimientos

(English version in the next page)

Aquellos que lean estas líneas no encontrarán muchas diferencias con cualquier otro agradecimiento que hayan podido leer anteriormente, tampoco encontrarán nada innovador, ni siquiera una elevada calidad literaria. Estos agradecimientos son uno más de los muchos que se escriben cada día como final de una etapa, es este caso la tesis. La diferencia está en que éstos son los míos, los primeros y últimos que escribo. Lo cual descarta una segunda tesis jeje.

Ante todo, dar las gracias a mi familia. Aguantar a un doctorando no es tarea fácil, lo reconozco. Todas esas horas dedicadas en cuerpo y alma a una tesis que nunca parece acabar, las jornadas de trabajo que no acaban con los resultados esperados, los deadlines, los nervios de las conferencias y un largo etcétera. Ellos no han hecho una tesis, pero la han '*sufrido*' igualmente. Por todo ello, muchas gracias. En esta categoría también entran los amigos que se han pasado horas escuchando hablar de la tesis jeje.

Dar las gracias a mis directores de tesis, Manuel, Nacho y Christian, por el tiempo, la dedicación y la ayuda brindada en todo momento, y sin la cual esta tesis no habría llegado nunca a término. Ha sido una larga carrera pero nunca he sentido que corriera sola. Todo lo hecho en este tiempo, mucho o poco, no habría sido posible si el profesor Manuel Prieto no me hubiese ofrecido la posibilidad de hacer esta tesis. Al hacerlo me abrió la puerta a una serie de experiencias que de otra forma habrían sido imposibles y que me han enriquecido no sólo profesionalmente sino también personalmente. Las conferencias, las estancias en IMEC, el trabajo en el departamento, y por encima de todo, la gente que he conocido en estos años, me han aportado mucho más de lo que esperaba. En este punto, y aunque suene a tópico, agradecer a todos aquellos que hicieron que mis estancias en Leuven fueran como ir a una segunda casa, y a los que han hecho que el trabajo en DACYA sea algo difícil de olvidar. En este caso no nombraré a nadie porque han sido muchos, los que por unos u otros motivos los que podrían aparecer en esa lista. Todos me han aportado algo durante estos años.

Por último pero no menos importante, gracias también a Francky Catthoor, Antonis Papanikolaou y Miguel Miranda por guiarme y ayudarme durante este tiempo. Agradecer todo el tiempo y esfuerzo dedicados, todas esas horas en IMEC donde

aprendí no sólo sobre mi tema de tesis sino también un poco acerca de la forma de trabajo en IMEC. Ha sido todo un honor formar parte de ese grupo. Y como no todo es trabajo, gracias también por todas las horas fuera de IMEC, participando por ejemplo de excursiones en bici y barbacoas.

Gracias a todos!

Those who read these lines will not find any difference with any other acknowledgments that you can have read previously, neither nothing imaginative, nor high literary quality. This acknowledgment is just one more among all those which are written everyday at the end of a stage, a thesis in this case. What differs from all the others, it is that this is mine, the first and the last one I will ever write.

First of all, I would like to thank my family. Having patience with a PhD candidate is not easy, I must admit it. All those hours devoted to a work which seems never ending, those working days finished without the expected results, the deadlines, conferences, etc. They have not done the thesis, but they have also 'suffered' it. Thanks for everything. At this point, I will also thank my friends, who have being hearing about this for so long.

Special thanks to my promotors, Manuel, Nacho and Christian, for their time, effort, support and help provided at any moment, otherwise this thesis would not have been finished. It has been a long-distance race, but I have never run alone. As there is always a beginning for everything, all the work done, a lot or a little, would not have been possible if Professor Manuel Prieto had not offered me the chance to do this thesis. Doing it, he allowed me to live experiences that otherwise would just have been impossible to live and have improved myself both personally and professionally. The conferences, the stays at IMEC, the work at the department, and over all, the people I have met in these years, have provided me more than expected. At this point, I would like to thank all those who have turned Leuven into a second house and those who have made to work at ACYA department an unforgettable experience.

Last but not least, I would like to thank Professor Francky Catthoor, Dr. Antonis Papanikolaou and Dr. Miguel Miranda for their understanding and help during these years. Thanks for all the time and effort dedicated, I really appreciate it. Thanks for all those hours at IMEC, where I learned not only about my topic but also about the daily work there. And, as life is not only work, thanks also for those hours out of IMEC participating in bike trips and barbecues.

Thanks!

Contents

Summary/Memoria

En 1965 Gordon Moore predijo que la densidad de transistores se doblaría cada 18 meses, estableciendo de esta forma una ley empírica que se ha venido cumpliendo desde entonces con ayuda de una continua reducción en el tamaño de la tecnología. Actualmente, estamos inmersos en la denominada era de la nanotecnología, con tamaños inferiores a los 32nm, muy por debajo de los 100nm alcanzados ya en la década pasada. De acuerdo a las predicciones realizadas por la *SIA* (*Semiconductor Industry Association*) [SIAR10] en 2010, la década actual seguirá caracterizada por una mejora en el escalado de la tecnología, manteniendo en vigor la ley de Moore. Como ejemplo, nos encontramos con estimaciones a corto plazo que sitúan en torno al año 2016 la existencia de tecnologías Flash de 16nm, mientras que a largo plazo con encontramos con estimaciones que indican que esa misma tecnología llegará a las memorias DRAM a finales de la década actual. En ambos casos se estará por debajo de los 10nm hacia el año 2024, reduciéndose más – por debajo de 7.5nm – en MPU/ASIC. Estos últimos tamaños ya se encontrarían cerca de lo que se considera el límite de la tecnología basada en silicio – 7nm –, con elementos que apenas albergan unas decenas de átomos [SGK11b].

Los avances tecnológicos llevados a cabo durante años han hecho posible la integración en un único chip de millones de transistores. En 2009 la densidad de transistores superaba los 2.000 Mtransistores/cm^2 en memorias SRAM y de acuerdo a la *SIA*, este número se incrementará hasta los 40.000 hacia 2020, alcanzado 110.000 Mtransistores/cm^2 tan sólo cuatro años más tarde [SIAR10]. Este enorme incremento en la capacidad de integración ha permitido el rápido desarrollo de soluciones basadas en los denominados *System-on-Chip* (*SoC*) y *System-in-Package* (*SiP*). Soluciones que abarcan desde el ámbito de las telecomunicaciones al de la electrónica de consumo y que suponen unos beneficios que se incrementarán desde los 45 millones de dólares en 2010 hasta los 69 millones hacia 2015 [Res]. Este gran mercado incluye todo tipo de dispositivos, a menudo portátiles, que manejan aplicaciones multimedia o *wireless* y que hacen uso de manera intensiva de algoritmos de procesamiento de imagen, sonido, habla y vídeo.

Por *System-on-Chip* se entiende todo aquel sistema que integra, en un único chip, un conjunto heterogéneo de componentes – procesadores, memorias, interfaces de entrada/salida, estructuras de interconexión, etc – para ofrecer una funcionalidad más compleja de la que pueden presentar por separado. Al estar todo incluido en un único chip se consiguen importantes mejoras en aspectos como el rendimiento, el tamaño, el coste y el consumo. En caso de que uno o varios circuitos integrados, pudiendo ser incluso *System-on-Chips* y no necesariamente de la misma tecnología, se combinen en un mismo paquete con otros componentes pasivos y múltiples interconexiones, lo que obtenemos es conocido como *System-in-Package*. La elección entre uno u otro tipo de sistema dependerá de factores como el tiempo de respuesta del mercado, el volumen de producción esperado o la necesidad de combinar diferentes tecnologías y diseños.

La misma tecnología que proporciona tan altos niveles de integración por medio de tamaños por debajo de la micra, también da lugar a una gran variedad de desafíos. Así, la verificación, el codiseño HW-SW, el incremento de la complejidad y el tiempo de diseño, y la consecuente reducción de productividad son sólo algunos de los problemas a los que se enfrentan los diseñadores. Estos problemas de índole tecnológica se unen además a la creciente demanda del mercado para obtener rápidamente sistemas que ofrezcan mayores funcionalidades con unos costes y consumos cada vez más reducidos.

Para afrontar algunos de estos problemas, nuevas prácticas de diseño para SoC y SiP basadas en el desarrollo independiente de bloques IP (*Intellectual Property*) permiten aumentar el reuso de los diseños, incrementar la fiabilidad de los diseños más complejos y reducir el tiempo de respuesta del mercado. También el hecho de que el tipo de aplicaciones dominantes en SoCs y SiPs compartan una serie de características comunes y generalmente conocidas de antemano, permiten llevar a cabo mejores análisis de los sistemas a desarrollar, contribuyendo así a mejorar el proceso de diseño. Sin embargo, el tipo de aplicaciones – normalmente acompañadas de restricciones de tiempo real y cuyo comportamiento es muy dependiente de los datos que se manejan en cada momento –, y el tipo de dispositivos donde se encuentran empotrados los sistemas – de tipo portátil y dependientes de una batería –, añaden dificultad a un proceso diseño ya de por sí complejo. Las aplicaciones, además, se suelen caracterizar por una gran cantidad de datos involucrados en su funcionamiento, lo que se traduce en un elevado porcentaje del área del dispositivo dedicado al sistema de memoria. Como resultado de esta distribución del área, del almacenamiento de datos y las operaciones de transferencia de datos involucradas, una gran parte de la energía consumida en todo

el sistema recae en la memoria. La importancia del sistema de memoria no se reduce sólo a su consumo, sino que también tiene importantes efectos sobre el rendimiento y la fiabilidad del sistema.

En general, pero especialmente en el ámbito de los sistemas empotrados –debido a los exigentes requerimientos que rigen el mercado –, el desarrollo de diseños globales y eficientes es muy necesario. Con tal fin, en esta tesis nos centramos en algunos de los problemas comentados anteriormente que introducen incertidumbre e ineficiencia durante el proceso de diseño:

- Dinamismo a nivel de aplicación

  La continua reducción tecnológica ha permitido ampliar significativamente el tipo de aplicaciones que nos podemos encontrar actualmente en los sistemas empotrados, dando así respuesta a la gran demanda existente en el mercado para trasladar sistemas complejos, cuyo uso hasta hace poco estaba restringido a los computadores, a entornos empotrados y portátiles, basados en SoC/SiP y con restricciones de energía. La complejidad de estas aplicaciones queda patente en el comportamiento variable, a menudo con restricciones temporales, que exhiben durante su ejecución. Este dinamismo debe ser gestionado adecuadamente, no sólo de cara al cumplimiento de las condiciones temporales sino también para proporcionar un consumo de energía adecuado.

- *Process variation*

  Como mencionamos anteriormente, la reducción en el tamaño de la tecnología saca a la luz problemas que anteriormente habían sido obviados por su escaso impacto. Estos problemas, conocidos como *process variation* se

originan durante el proceso de fabricación producen una falta de control sobre parámetros tecnológicos que resultan claves durante el proceso de diseño, tales como el retardo o la potencia consumida. Esta falta de control se traduce en incertidumbre respecto a los valores de esos parámetros, cuya variabilidad tiene importantes efectos sobre el sistema. En esta tesis, ponemos nuestra atención en los efectos que estas variaciones tienen en las memorias.

- Fiabilidad

  Los problemas de fiabilidad hacen referencia a cuestiones derivadas de la degradación de los dispositivos a lo largo del tiempo. Más allá de las alteraciones que sufren los transistores durante el proceso de fabricación en parámetros tales como el voltaje umbral (V_{th}), a lo largo del tiempo de vida de un dispositivo esos parámetros se ven de nuevo afectados por aspectos ambientales relacionados por ejemplo con caídas de voltaje o fluctuaciones de temperatura. Estas alteraciones pueden llegar a degradar tanto los componentes que circuitos completamente funcionales pueden acabar siendo inservibles.

Dinamismo en plataformas – *Process variation*

La agresiva reducción tecnológica llevada a cabo en las últimas décadas ha permitido una constante mejora de rendimiento en los sistemas empotrados tanto en términos de energía como de rendimiento. Sin embargo, el uso de tamaños por debajo de la micra ha alterado los planes establecidos, dando lugar a una serie de problemas que ya no pueden ser resueltos únicamente desde la tecnología [BKD04].

Problemas que eran considerados secundarios y poco significativos, ahora se han convertido en relevantes y con graves efectos en los sistemas. Estamos hablando de desajustes en los valores de parámetros eléctricos y físicos debido a imperfecciones en las máscaras, variaciones incontroladas durante el proceso de fabricación y factores ambientales que dan lugar a significativas variaciones de temperatura y potencia.

Los principales efectos de estas alteraciones se ven en el rendimiento y el consumo de los circuitos integrados y se pueden clasificar en dos categorías, sistemáticos y aleatorios, dependiendo de si existe o no correlación espacial entre las alteraciones. La correlación aparece principalmente relacionada a parámetros físicos afectados por variaciones por lo que en general, se puede decir que las variaciones sistemáticas tienen su origen en problemas con la litografía – máscaras, lentes, etc –, mientras que los efectos aleatorios son causados principalmente a la irregularidad en la densidad de dopantes.

Aunque la variabilidad se puede dar a distintos niveles, nosotros nos centramos en el tipo de variaciones que actúan dentro de un mismo circuito y que son conocidas como *intra-die process variation*, y en las cuales la SIA (*Semiconductor International Association*) ha puesto mucho énfasis [SIAR10] en los últimos tiempos. Este tipo de variaciones afectan a parámetros físicos y eléctricos del circuito tales como la longitud de puerta o el voltaje umbral de los transistores, comprometiendo el comportamiento, el rendimiento de los dispositivos y el consumo de potencia, especialmente la potencia estática (*leakage*), que incrementa exponencialmente.

A lo largo de los años, una gran variedad de técnicas se han desarrollado con la intención de tolerar las inevitables variaciones, tanto sistemáticas como

aleatorias, pudiéndose encontrar en la literatura técnicas de mitigación centradas en el diseño, centradas en el proceso de fabricación y combinaciones de ambos enfoques [KKK+08]. En este trabajo nos centramos en variaciones de tipo aleatorio, proporcionando un nuevo enfoque a las habituales técnicas centradas en el diseño.

El comportamiento estocástico introducido por la variabilidad ha sido generalmente afrontado mediante metodologías basadas en márgenes de diseño para aumentar la seguridad acerca del comportamiento del circuito [ZHHO04a]. Estos diseños habitualmente establecen como margen hasta tres desviaciones típicas sobre los valores medios ($3\sigma \pm \mu$) para poder capturar la variabilidad. Sin embargo, la variabilidad continúa creciendo. Como muestra podemos mencionar el 30 % de variación que se ha llegado a reportar en la frecuencia de los chips [ABZ03, FP09]. Dada esta situación, lo valores habituales usados como margen de diseño se quedan pequeños, incapaces de recoger las crecientes variaciones. Sin embargo, el incremento de estos márgenes, con diseños que pueden llegar hasta 6σ en tecnologías de 28nm [Gup11], tampoco es sostenible por mucho más tiempo dadas las grandes sobrecargas que recaen en el diseño, tanto en consumo de energía como en pérdida de rendimiento, al resultar demasiado conservadoras [CQS04].

Aunque la variabilidad afecta a todos y cada uno de los componentes presentes en un sistema, lo hace especialmente en las memorias, debido a los reducidos tamaños que se emplean en su fabricación con la finalidad de mejorar el rendimiento y reducir la energía [CDSM04]. Estos tamaños reducidos son los que convierten a las memorias en uno de los componentes más propensos a las variaciones. Estas alteraciones son especialmente relevantes en cuanto a tiempo de acceso y energía, ya que después de su fabricación, los valores reales tienden a ser superiores a los establecidos durante el diseño. Así, además de resultar impredecibles, los módulos

de memoria pueden llegar a comprometer restricciones de tipo temporal y de potencia impuestas en el diseño [HCM+05]. Este comportamiento puede verse en la Figura 1, donde se muestra una memoria bajo los efectos de la variabilidad. La figura muestra el desplazamiento, hacia posiciones más energéticas y lentas de los valores nominales establecidos en durante el diseño. Los valores reales varían de una memoria a otra, por lo que el resultado es una especie de 'nube' en la que el valor concreto de tiempo y energía no se sabrá hasta después de la fabricación.

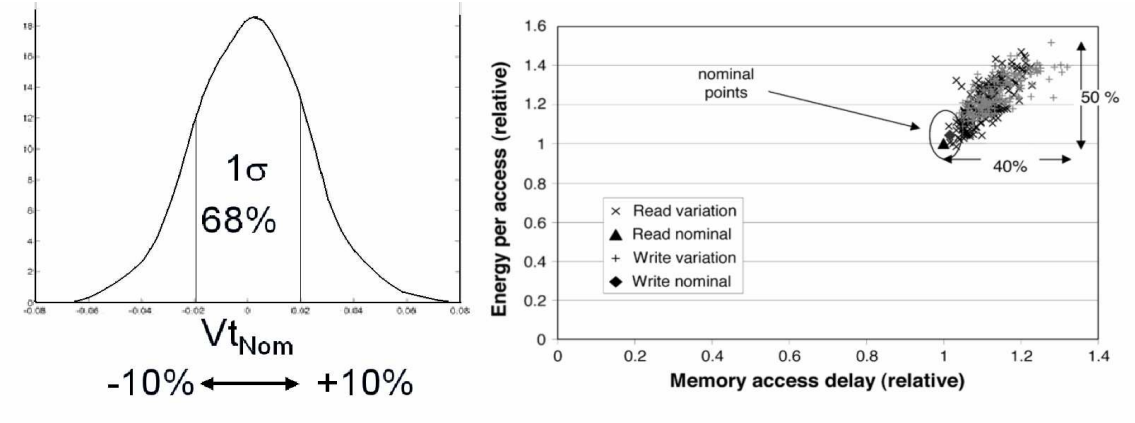


Figura 1: La variabilidad tiene un gran impacto en las memorias, tanto en energía como en tiempo de acceso, donde una variación de 1σ en valores nominales de parámetros como V_{th} conlleva retardos se pueden llegar al 40 % [WMP+05].

El impacto de la variabilidad en las memorias, junto con el hecho de que ocupan una parte muy significativa del área y son responsables de la mayor parte del consumo del sistema debido a su alta actividad, convierten a estos componentes en el centro de atención de esta tesis.

Dinamismo a nivel de aplicación

Hasta hace no mucho tiempo, aplicaciones complejas, con alta demanda de datos y conducidas por eventos, eran sólo posibles en procesadores de propósito general, mientras que las aplicaciones que se usaban en SoC normalmente mostraban un comportamiento mucho más estático. Sin embargo, los avances tecnológicos han ampliado el rango de aplicaciones que se encuentran disponibles en el mercado de los *System-on-Chip*: aplicaciones multimedia, comunicaciones, entornos inteligentes, etc. La diversificación en el tipo de aplicaciones y el incremento del número de usuarios hace necesario desarrollar estándares para tales aplicaciones. Estándares, que a pesar de estar enfocados en diferentes tipos de aplicaciones – imagen, vídeo, audio –, tienen en común significativos incrementos de rendimiento y de complejidad debido a la gran cantidad de datos que se han de manejar en las aplicaciones.

Como ejemplo de estos estándares podemos destacar el que se viene desarrollando por *ISO Motion Picture Experts Group* (MPEG) desde 1992 [Sik97] para representar de forma codificada imágenes en movimiento y/o audio con tasas que alcanzaban inicialmente 1.5 Mbit/s. El estándar, conocido como MPEG-1, fue extendido en 1994 para alcanzar velocidades de hasta 10 Mbit/s (MPEG-2) y ser empleado en aplicaciones para retransmisión por satélite, televisión digital, etc. En 1999 se produjo una nueva extensión, denominada MPEG-4, para afrontar los nuevos desafíos procedentes de aplicaciones multimedia que ofrecían alta interactividad, compresión más eficiente y robustez en entornos propensos a errores.

El estándar inicial ha seguido evolucionado para adaptarse a entornos cada vez más complejos y a situaciones cada vez más dinámicas, dando lugar a

estándares como H.264/MPEG-4 AVC (*Advanced Video Coding*) en 2003 y SVC (*Scalable Video Coding*) en 2007, que resultan de un proyecto conjunto entre *ITU Telecommunication Standardization Sector* (ITU-T) e *ISO/IEC Moving Picture Experts Group* (MPEG). AVC, un estándar de compresión de vídeo diseñado para solventar diversas debilidades existentes en estándares anteriores, reduce significativamente la tasa de bit necesaria para representar un determinado nivel de calidad sin sufrir grandes incrementos en la complejidad de los diseños. AVC permite comunicaciones altamente eficientes y fiables, dando soporte a aplicaciones *streaming* y permaneciendo robusto a los problemas de transmisión de los canales. Además, SVC [Cod], especificado en el Anexo G de H.264/MPEG-4 AVC, está enfocado en la escalabilidad de la codificación de vídeo. Esta escalabilidad puede ser temporal, espacial o referida a la calidad, afectando a la tasa de *frames*, la resolución y la tasa de datos respectivamente. SVC permite de esta forma una fácil adaptación a diferentes características del terminal, así como la difusión de *streaming* a través de redes muy heterogéneas.

Un ejemplo del dinamismo que se puede encontrar en las actuales aplicaciones se muestra en la Figura 2, que ilustra un caso de dinamismo en películas basado en la variación de la tasa de bits entre los diferentes *samples*[1] que la forman. Como se puede ver, hay *samples* que exhiben una alta tasa de bits, mientras que en otros casos ésta es muy reducida. Las tasas altas se corresponden con escenas con una gran cantidad de movimientos de cámara en poco espacio de tiempo, mientras que las tasas más bajas se corresponden con escenas más estáticas donde la cámara apenas se mueve o lo hace de forma muy lenta.

[1]La aplicación *Bitrate Viewer* [Vie] ha sido empleada en este ejemplo para visualizar la variación en la tasa de bit existente en un fichero MPEG.

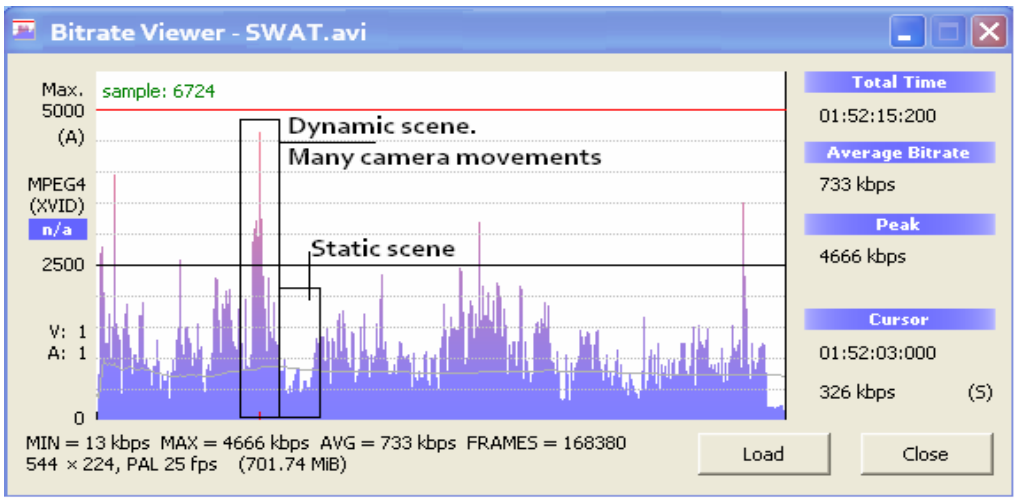


Figura 2: Ejemplo de dinamismo presente en un vídeo de tipo MPEG−4 con tasa de bit variable. Las largas variaciones en la tasa de bit de las muestras diferencia claramente las escenas más estáticas de las más dinámicas.

Actualmente, las aplicaciones multimedia pueden exhibir características muy variables a lo largo de su ejecución, de forma que los recursos usados, la memoria o las comunicaciones pueden variar considerablemente a lo largo de la misma si las comparamos con aplicaciones anteriores en este dominio. La calidad de servicio, los requerimientos temporales que se precisan cumplir o la presencia de otras aplicaciones compitiendo por los recursos también son fuente de dinamismo en tiempo de ejecución.

Este dinamismo a nivel de aplicación generalmente ha sido afrontado de manera estática con metodologías ha nivel de diseño, siendo las más comunes aquellas basadas en la asunción del caso peor para reflejar el comportamiento de la aplicación. Sin embargo, el incremento del dinamismo hace que el comportamiento de las aplicaciones durante su ejecución diste de su estimación estática, aumentando la impredecibilidad de las mismas y llevando a técnicas de diseño cada vez más

conservadoras que derivan en pérdidas de rendimiento y considerables costes energéticos.

Enfoque integrado para afrontar variabilidad y dinamismo

Los efectos de la variabilidad en el rendimiento y la energía, y el comportamiento dinámico existente en las aplicaciones han sido ampliamente considerados a lo largo del proceso de diseño o en tiempo de ejecución. Como resultado, una gran variedad de técnicas se han ido desarrollando para afrontar ambos problemas. Entre las primeras aportaciones se encuentran las técnicas basadas en el caso peor o en márgenes de diseño, siendo especialmente notorias aquellas destinadas a mitigar la variabilidad (*process variation*), con alternativas a muy diversos niveles: circuito, sistema, arquitectura, memoria, etc. A pesar de esto, las técnicas desarrolladas hasta el momento para reducir el impacto de la variabilidad y el dinamismo de las aplicaciones ignoran el efecto combinado de ambos problemas sobre el sistema. Las técnicas de mitigación de variabilidad reducen las aplicaciones a códigos predecibles y estáticos, de forma que no contemplan otro dinamismo a parte del existente debido a la tecnología. De la misma manera, las técnicas que trabajan sobre el dinamismo de las aplicaciones asumen que la información que manejan acerca del rendimiento y el consumo de energía del sistema son completamente fiables en tiempo de ejecución. Ambos enfoques llevan generalmente a posiciones conservadoras, relacionadas por ejemplo con la frecuencia de operación, que permitan garantizar el diseño aún a expensas de sufrir pérdidas de rendimiento.

Dado que se espera que las aplicaciones sean cada vez más dinámicas y el hardware empleado se base en tecnologías cuyos tamaños son cada vez más reducidos, se requieren soluciones que tengan en cuenta ambas fuentes de incertidumbre, de forma que se mejore el rendimiento global de los sistemas y se optimice la energía consumida.

En este trabajo, exploramos un enfoque integrado para manejar la variabilidad presente en los sistemas de memoria diseñados para dominios de aplicaciones dinámicas. Nuestra propuesta combina: (1) técnicas de preprocesado de las aplicaciones para modelar el dinamismo existente en ellas; (2) análisis estadístico para mitigar la variación en memorias; (3) monitorización hardware y calibración en tiempo de ejecución para adaptar el sistema a los efectos tanto de la variabilidad como del dinamismo de las aplicaciones. El objetivo es ofrecer oportunidades de reconfiguración en tiempo de ejecución para maximizar la productividad paramétrica y la eficiencia energética a nivel de memorias, lo que se traduce en un mayor rendimiento y menor consumo en todo el sistema.

Nuestra propuesta está basada en los siguientes conceptos y metodologías ya existentes:

- Memorias configurables [WMP+05]: Memorias on-chip más flexibles debido a la presencia de puntos de trabajo variables y configurables en tiempo de ejecución
- Escenarios [GPH+09]: Tratamiento global del comportamiento dinámico que las aplicaciones exhiben durante su ejecución

Memorias configurables. Mitigando la variabilidad a nivel de memoria

Dado que las características de los componentes presentes en un chip varían durante el proceso de fabricación debido al fenómeno conocido como *process variation*, existen autores que señalan la falta de garantías en sistemas que emplean configuraciones fijas e inmutables [LB06]. Siguiendo esta apreciación, nuestra metodología mitiga el impacto de la variabilidad en el sistema de memoria mediante el uso de memorias configurables que reciben el apoyo de un sistema de realimentación de información a nivel de sistema en tiempo de ejecución [PLW+05]. A diferencia de una memoria habitual, las memorias configurables, como la representada en la Figura 3, presentan al menos dos puntos o modos de trabajo, dando lugar a una memoria adaptable con modos que se diferencian tanto en tiempo de acceso como en energía. En particular, la Figura 3 representa una memoria de dos modos, uno de los cuales es muy económico en términos de energía pero lento en tiempo de acceso, mientras que el segundo es más rápido a costa de consumir más energía.

Justificaremos nuestra elección por las memorias configurables a través del siguiente ejemplo. Consideramos en primer lugar un sistema que hace uso de un único módulo de memoria, con un único punto de trabajo. Supongamos además, que se ha de cumplir un determinado *deadline* en el sistema. En caso de que no tengamos en cuenta la posibilidad de que el sistema presente problemas debido a la variabilidad, en tiempo de diseño podemos vernos tentados a fijar la frecuencia del sistema basándonos en el tiempo de acceso que se espera que tenga la memoria. Sin embargo, en tiempo de ejecución, la variabilidad puede incrementar el tiempo

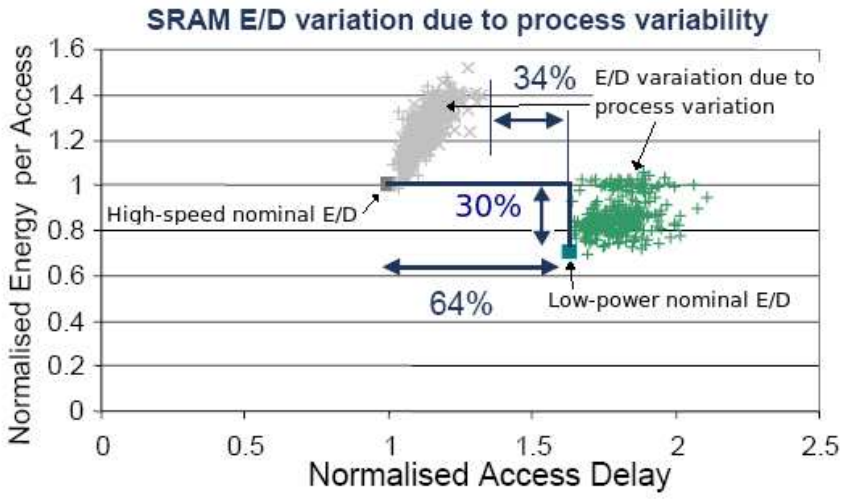


Figura 3: Memoria configurable de dos modos, bajo variabilidad [WMP+05].

de acceso lo suficiente como para hacer que la frecuencia elegida sea demasiado alta y la memoria se vuelva inoperativa. Por otro lado, si tenemos en cuenta la variabilidad durante el diseño del sistema, podemos establecer una frecuencia demasiado conservadora que nos permite cumplir los requerimientos temporales a costa de tiempo y energía.

En presencia de memorias configurables, en tiempo de ejecución y con la metodología adecuada, la existencia de varios puntos de trabajo permite evitar el uso de márgenes de diseño tan drásticos y conseguir una mejor adaptación del sistema frente al impacto de la variabilidad. En nuestro ejemplo, el uso de este tipo de memorias se traduce en no tener que fijar en tiempo de diseño la frecuencia del sistema. Eso ocurrirá después de la fabricación, una vez que el tiempo de acceso real puede ser conocido. Es en ese momento cuando, mediante técnicas de compensación se seleccionaría el punto de trabajo más adecuado para cumplir el *deadline* es seleccionado, es decir, el modo en el que cada memoria debe configurarse. En caso de que el *deadline* varíe a lo largo de la ejecución de

la aplicación, el punto de trabajo puede cambiarse adecuadamente. Esta capacidad para adaptar las memorias a la situación actual del sistema puede aplicarse tanto para mitigar los efectos de la variabilidad debidos a la fabricación como para reducir en cierto modo el impacto de aquellas variaciones que pueden ir apareciendo a lo largo del tiempo.

Investigaciones previas sobre memorias configurables [PLW+05] han mostrado su efectividad lidiando con la variabilidad en contextos de aplicaciones periódicas y estáticas. En este trabajo extendemos la metodología inicial para poder ser aplicada en dominios de aplicaciones dinámicas.

Escenarios. Mitigando el dinamismo a nivel de aplicación

Para ilustrar las ideas que engloban el concepto de escenario, vamos a considerar como ejemplo una sencilla aplicación basada en *frames* para la que asumimos una tasa de *frames* constante durante toda la ejecución – por ejemplo 24 *frames* por segundo –. El núcleo de la aplicación, conforme se muestra en la Figura 4, consiste en un bucle principal controlado por una variable dependiente de las acciones del usuario. Esta variable determina la carga de trabajo a realizar dentro del bucle. Por simplicidad, esta variable admite únicamente dos posibles valores. La carga de trabajo en función del valor de la variable queda reflejada en la Figura 4, donde las unidades que se emplean representan el tiempo de ejecución del bucle asumiendo que el procesador opera a su máxima frecuencia.

Frente a esta situación, una metodología estática tradicional fijaría en tiempo de diseño la frecuencia de operación. En nuestro ejemplo, si el diseño es conservador, esta frecuencia vendría determinada por el valor de la variable de usuario que

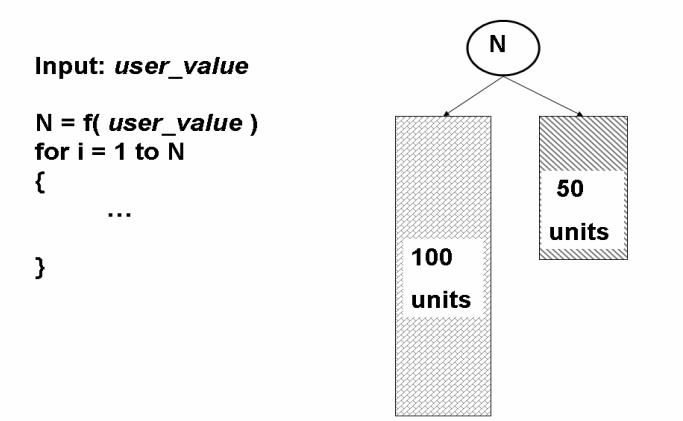


Figura 4: Aplicación basada en *frames* con una carga de trabajo variable a lo largo de su ejecución.

generase el tiempo de ejecución más largo, i.e. el caso peor. De esta forma, los requerimientos temporales podrían estar siempre garantizados, aunque a expensas de emplear constantemente la frecuencia máxima (caso A en la Figura 5). Si se opta por relajar las condiciones temporales, podemos estimar un número medio de iteraciones en función del cual fijar la frecuencia. De esta forma, la frecuencia sería inferior al caso anterior, pero se incumpliría el *deadline* en caso de cargas de trabajo elevadas (caso B en la Figura 5).

Para afrontar el dinamismo existente y evitar el uso de prácticas conservadoras en tiempo de diseño, hacemos uso de una metodología donde las decisiones que se han de aplicar en tiempo de ejecución no se toman por completo durante el diseño del sistema, sino que se van refinando durante la ejecución de la aplicación. Esta metodología permite la optimización de la aplicación a lo largo de todo su tiempo de vida, desde el diseño hasta su ejecución.

La metodología está basada en la existencia de los denominados escenarios, un concepto que permite clasificar el comportamiento de la aplicación en patrones

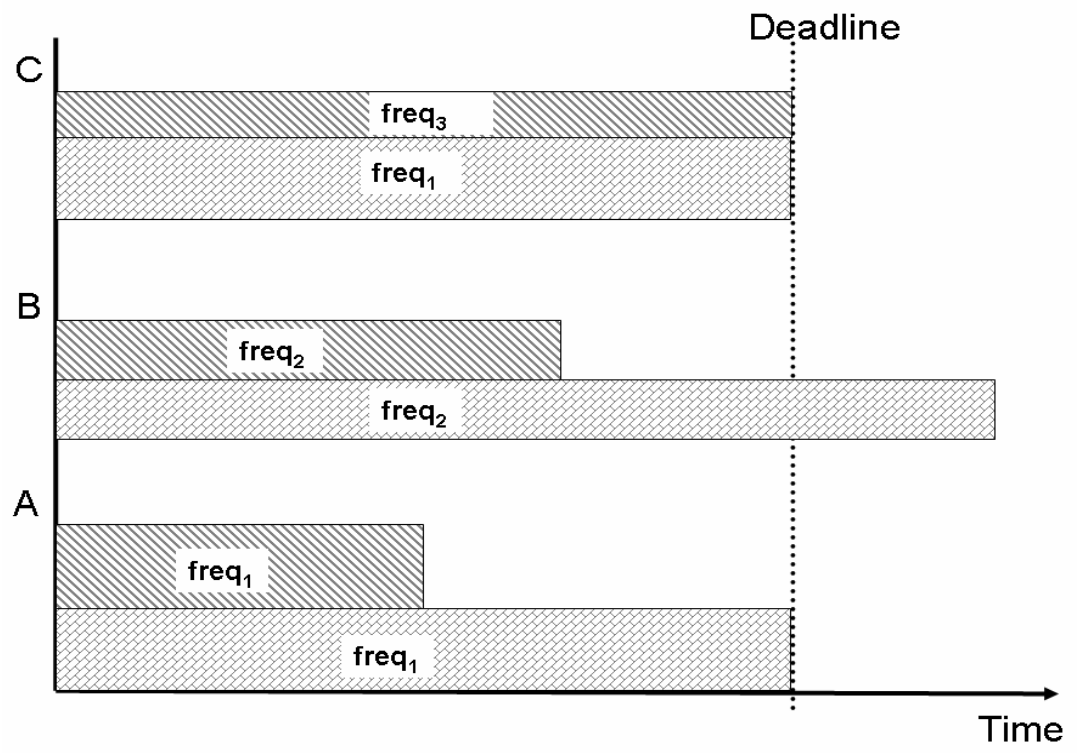


Figura 5: Grado de cumplimiento de requerimientos temporales bajo diversos enfoques.

determinísticos y representativos que son claramente identificables en tiempo de ejecución. Estos patrones, descubiertos y caracterizados en tiempo de diseño pueden ser vistos como flujos de control que serán ejecutados en función de los datos presentes en la aplicación, las entradas del usuario, o las métricas empleadas en el sistema. La información acerca de los escenarios es usada en tiempo de ejecución como forma de prever el comportamiento de la aplicación a corto plazo y proporcionar así una respuesta a nivel de sistema particularizada a las condiciones particulares que presenta la aplicación en un momento dado. En tiempo de ejecución, un mecanismo de detección basado en las entradas y el flujo de la aplicación, permite identificar qué escenario se está produciendo y

así adaptar el sistema a sus condiciones con un mínimo coste y garantizando las condiciones temporales exigidas. En general, todas las técnicas a nivel de sistema necesitan garantizar de alguna forma el rendimiento en aplicaciones dinámicas, ya sea mediante estimaciones o predicciones. De la misma forma, los escenarios ofrecen estas estimaciones de forma que el sistema pueda ser ajustado de manera eficiente mediante los parámetros de configuración que existan para tal propósito [TMQW06, KUM+10].

Para clarificar la idea de escenario, aplicamos el concepto en el ejemplo explicado anteriormente. En tiempo de diseño podríamos considerar dos escenarios, asociados a los dos valores de la variable de entrada ya comentada. Una vez que se han descubierto los escenarios, cada uno de ellos es asociado a una frecuencia única. Posteriormente en tiempo de ejecución, se lleva a cabo la identificación del escenario en cada *frame*, una vez que se conoce el valor de la variable de usuario. De esta forma, en cada *frame* se aplica la frecuencia asignada al escenario detectado (caso C en la Figura 5).

En ocasiones en las que no es posible asociar un determinado comportamiento en tiempo de ejecución con ningún escenario, existe el denominado escenario de *backup*, que aglutina los comportamientos no identificables para asegurar una funcionalidad adecuada en todo momento, incluso ante situaciones no contempladas durante el diseño. En nuestro ejemplo, la activación de este escenario significaría la aplicación de una frecuencia superior a las establecidas para los dos escenarios ya identificados. El escenario de *backup* puede ser considerado como el único momento en la metodología donde se emplearían estimaciones de caso peor. Sin embargo, dada la baja frecuencia de aparición de este escenario, se puede

concluir que los escenarios permiten evitar estimaciones de tiempo demasiado conservadoras.

Metodología

La metodología desarrollada en esta tesis con la finalidad de mitigar los efectos derivados del *process variation* y manejar el dinamismo presente en las aplicaciones actuales, se presenta dividida en tres etapas claramente diferenciadas que abarcan desde el mismo proceso de diseño hasta la ejecución de las aplicaciones en una plataforma específica. Las tres etapas, que gráficamente se muestran en la Figura 6, y sus principales características se indican a continuación.

Diseño

En esta primera etapa se da soporte al desarrollo del sistema de memoria de la plataforma y se adecúa la aplicación o aplicaciones objetivo al concepto de escenarios. Mediante el análisis de cada aplicación se extraen los escenarios. Este análisis permite trazar los diferentes flujos de control en los que se compone la aplicación y caracterizarlos en términos de tiempo, carga de trabajo y uso de memoria. A partir de esta información se determinan las características más representativas de la aplicación y se extraen los patrones que describen su comportamiento. Los patrones que son considerados similares en términos de coste y comportamiento son agrupados, de tal forma que la aplicación es finalmente descrita por estos conjuntos de patrones, cada uno de los cuales recibe el nombre de escenario.

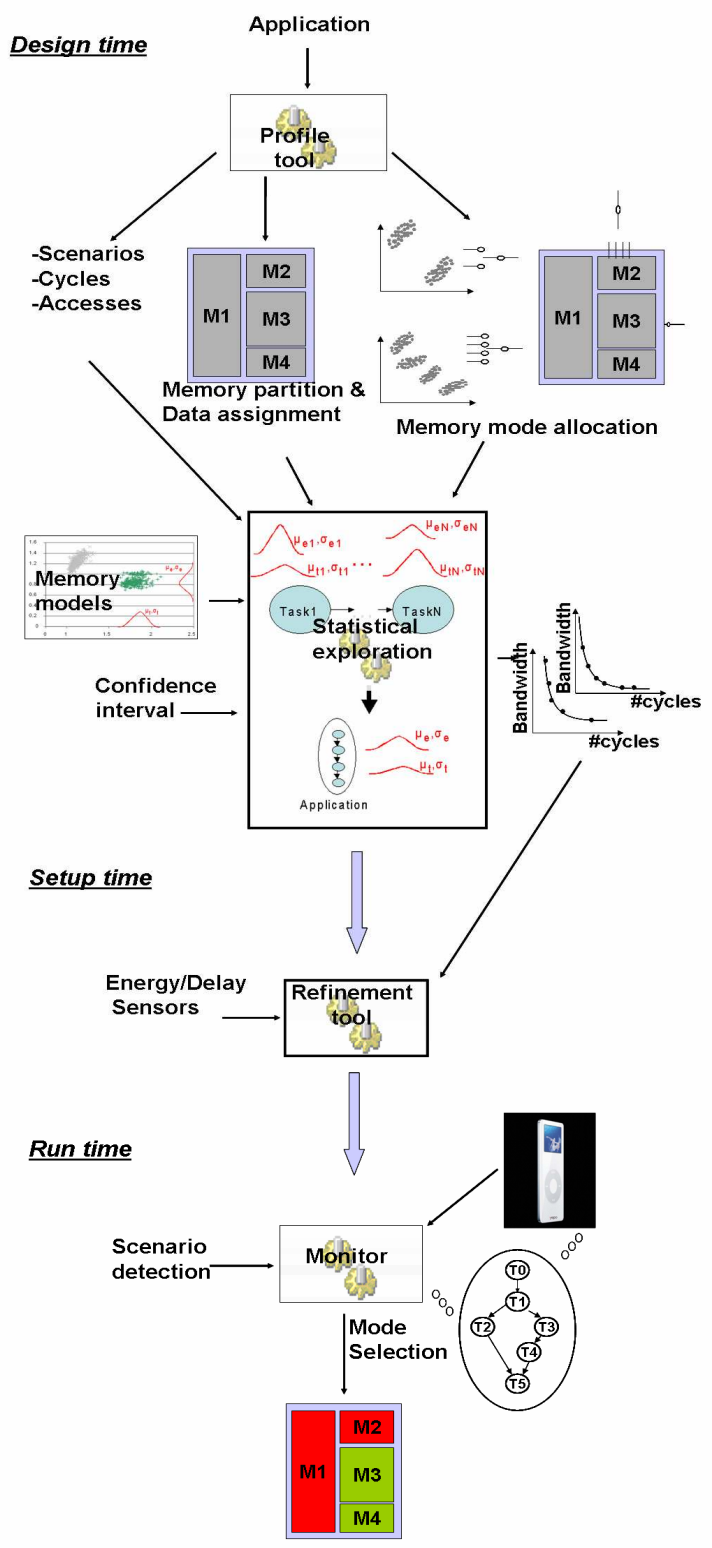


Figura 6: Metodología desarrollada.

Una vez que los escenarios han sido determinados y se establece un sistema de memoria(*memory allocation*) y su correspondiente asignación de datos (emphdata assignment), se lleva a cabo el proceso que determina el número de modos de trabajo que habrá disponibles en cada una de las memorias configurables (***memory mode allocation***). La asignación de modos de trabajo se lleva a cabo mediante un algoritmo de búsqueda por gradiente, estableciendo un compromiso entre los requerimientos de área que se han de cumplir y el ahorro de energía estimado.

A continuación, se establece la forma en la que se ha de configurar cada memoria en tiempo de ejecución para que, dado un escenario y unas restricciones temporales concretas, la aplicación cumpla con ellas de la forma más eficiente posible en términos de energía (***statistical exploration***). Configurar las memorias consiste en determinar qué modo de trabajo se aplicará en cada memoria concreta para cumplir con las restricciones temporales con el menor coste energético. Para ello, la aplicación objetivo es dividida en una secuencia de tareas, de forma que los accesos a memoria y el número de ciclos requerido por la aplicación se reparten entre ellas, resultando más manejable. Cada una de las memorias involucradas en las tareas es configurada de manera independiente, de forma que a lo largo de la ejecución de la aplicación una misma memoria puede ser configurada en diferentes modos. En ausencia de variabilidad la obtención de las configuraciones se podría obtener en tiempo de diseño y aplicarse en ejecución sin perjuicio para el sistema. Sin embargo, la presencia de variabilidad dificulta la obtención de configuraciones precisas en tiempo de diseño debido a la incertidumbre que crea. Por este motivo el establecimiento de las configuraciones se realiza de forma estadística mediante modelos que permiten estimar en tiempo de diseño el impacto de la variabilidad en las memorias, tanto en tiempo como en energía. La técnica estadística desarrollada

maneja unos intervalos de confianza acerca de la energía total y el tiempo asociado a una configuración, de forma que es posible reducir el espacio de búsqueda de forma significativa y asegurar el cumplimiento de las restricciones temporales con la menor energía. Así, dado un conjunto de restricciones temporales en un escenario, cada uno de los tiempos llevará asociado una configuración de cada memoria en cada tarea que permita asegurar su cumplimiento de manera estadística.

Setup

Una vez terminado el proceso de fabricación, y antes de comenzar la ejecución de una aplicación, se lleva a cabo un reconocimiento del sistema en el que se determina el grado de afectación del sistema de memoria por *process variation*. La variabilidad real existente ha de compararse con la estimada en tiempo de diseño, para refinar las soluciones estadísticas obtenidas en la etapa anterior (***setup refinement***). Este proceso da lugar a una etapa de calibración para ajustar las soluciones obtenidas en tiempo de diseño a la situación concreta del sistema y asegurar así los requerimientos de la aplicación. En este proceso las configuraciones iniciales son modificadas ligeramente mediante algoritmos de máxima pendiente con la intención de mejorar: (1) el tiempo de ejecución, en caso de que la variabilidad haya afectado tanto como para que la configuración inicial incumpla las restricciones temporales. Para ello se buscan aquellos cambios de modos que aportan una mayor reducción de tiempo con el menor impacto en energía; (2) la energía, en caso de que no se incumplan las restricciones establecidas. Se buscan aquellos cambios de modo que permitan reducir rápidamente el coste en energía con el menor incremento posible en tiempo.

Este proceso de calibración podría ser considerado como una etapa opcional de la que se podría prescindir, a riesgo de incumplir requerimientos temporales y conseguir menor eficiencia energética.

Ejecución

Una vez que la aplicación comienza a ejecutarse, y hasta que termine, la tercera fase de la metodología permite asistir al sistema de memoria en todo momento para adaptarlo de manera dinámica de acuerdo a los cambios que se reflejen en el comportamiento de la aplicación (*mode selection*). De esta forma, y dado que no se emplean técnicas conservadoras sino adaptativas, se consiguen asegurar las condiciones temporales requeridas por las aplicaciones mientras se reduce el coste en energía.

Así, durante la ejecución de la aplicación el sistema es monitorizado para detectar cambios en el escenario actual o en los requerimientos temporales. En caso de cambio, una de las configuraciones almacenadas se selecciona y aplica al sistema para favorecer el cumplimiento de las nuevas condiciones con el menor coste posible. Este proceso no interfiere en el rendimiento del sistema ya que la reconfiguración es un proceso rápido y ocurre de manera ocasional.

Todo este proceso queda reflejado de manera esquemática en la Figura 7.

Además de mitigar el impacto de la variabilidad en memorias y del dinamismo presente en las aplicaciones, otro de los objetivos de esta metodología es llevar a cabo la mayor parte del trabajo en tiempo de diseño. De esta forma en tiempo de

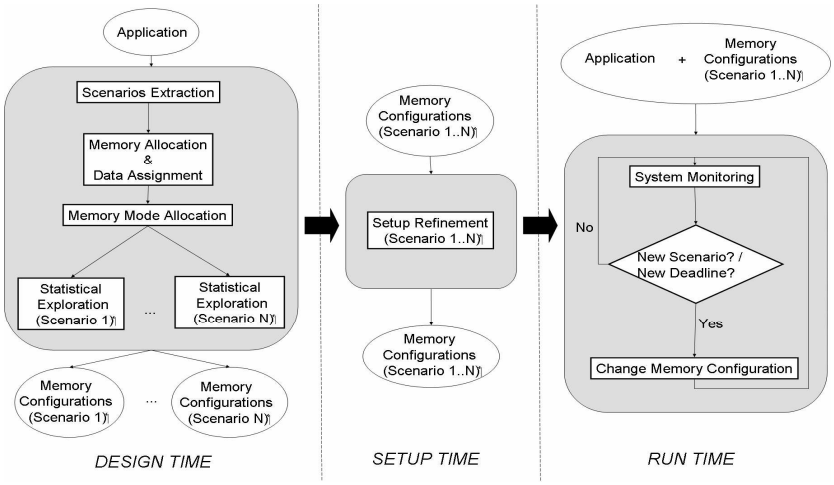


Figura 7: Esquema de funcionamiento de la metodología, donde las técnicas más costosas se aplican en tiempo de diseño.

ejecución ninguna técnica debe suponer una sobrecarga excesiva para el sistema, de forma que su rendimiento no se vea mermado.

En esta investigación nos centramos en el impacto de la variabilidad debida al proceso de fabricación, omitiendo fuentes de variación dinámicas relacionadas con la temperatura o con la degradación temporal de los dispositivos.

Resultados experimentales

En todo momento de la investigación se ha empleado como aplicación objetivo un decodificador de MP3, al que hemos aplicado tanto la metodología presentada en esta tesis, como las técnicas tradicionales de diseño basadas en estimaciones conservadoras con el propósito de compararlas. Nuestras investigaciones parten de una implementación en C para el decodificador de MP3, desarrollado por Krister Lagerström [Lag01], carente de escenarios y que aparecerá nombrado como *Base* de ahora en adelante. Aplicando el concepto de escenario y tras analizar el código de

la aplicación, obtenemos una versión basada en dos escenarios principales – *Long* y *Short* –, además del escenario de *Backup*. Esta nueva versión del decodificador, desarrollada por Martín Palkovic [Pal07] y denominada como *Escenarios* o SA (*Scenario aware*), se basa en el tipo de bloques que son codificados en cada uno de los *frames* que componen un MP3 y que son fácilmente reconocibles en la cabecera de cada *frame*. La extracción de escenarios permite llevar a cabo transformaciones en el código con la finalidad de optimizar la memoria, mejorar la localidad temporal de los datos, reducir el uso de *buffers* auxiliares y así reducir los accesos a los módulos de memoria.

Para simular de forma más clara el dinamismo en nuestra aplicación objetivo, determinamos un conjunto de restricciones temporales que se deberán cumplir a nivel de *frame* en tiempo de ejecución.

En cuanto al sistema de memoria, su determinación y la asignación de datos para ambas versiones de la aplicación se hace a través de ATOMIUM [ATO], tras el análisis correspondiente del código de la aplicación. ATOMIUM considera durante toda su actividad memorias regulares, con un único modo de trabajo, sin embargo, durante el resto de la metodología las memorias son siempre configurables. El sistema de memoria establecido para cada una de las versiones se muestra en la Tabla 1. La asignación de datos muestra una tendencia a almacenar las estructuras más accedidas en memorias pequeñas, ya que su tiempo de acceso es inferior y son más eficientes energéticamente, incluso bajo variabilidad, que las memorias más grandes. Los cambios introducidos en el código por la presencia de escenarios da lugar a sistemas de memoria diferentes, con un total de ocho memorias para el caso *Base* y diez memorias para *Escenarios*

Tamaño memoria	Base	Escenarios (*SA*)
1 KB	1	0
4KB	2	1
8KB	2	2
16KB	2	4
32KB	1	3

Cuadro 1: Sistema de memoria establecido para cada una de las versiones disponibles del decodificador de MP3.

Asignación de modos a memorias

Una vez determinado el sistema de memoria y la asignación de datos para cada una de las versiones del decodificador MP3 disponibles, se aplica el **algoritmo de asignación de modos** para establecer el número de modos que se implementarán en cada uno de los módulos de memoria. A pesar de que cualquier número de modos es posible, para reducir tanto la complejidad del algoritmo de asignación, así como la del resto de técnicas que componen la metodología, se manejan tres únicos modelos de memoria: memorias consistentes en dos, cuatro u ocho modos. Estos modos se distribuyen en las memorias conforme se muestra en la Figura 8. El área ocupada por las memorias es considerado como factor limitante en este algoritmo, estableciendo como límite el área ocupada por un sistema de memoria basado en módulos que poseen cuatro modos de trabajo cada uno. Los resultados que ofrece el algoritmo muestran una tendencia por asignar un mayor número de modos a las memorias más pequeñas y con mayor frecuencia de accesos. El número de modos se va reduciendo paulatinamente a medida que la memoria aumenta de tamaño y los accesos a ellas disminuyen. Así, la asignación de modos para las memorias de la versión *Escenarios* (SA) queda de la siguiente manera: (1) se asignan 8 modos a las memorias pequeñas (4KB y 8KB); (2) 4 modos a las memorias de 16KB;

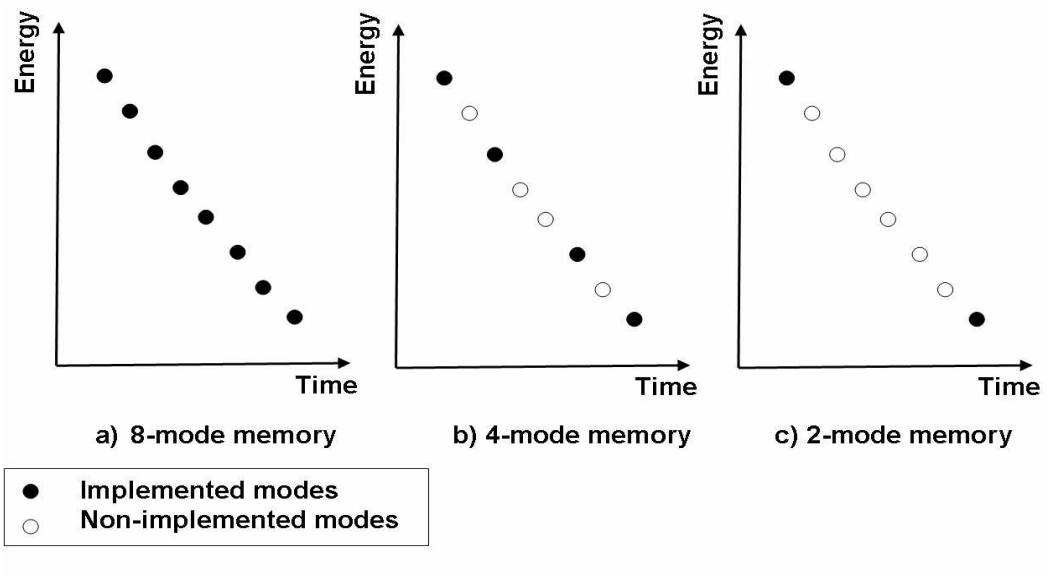


Figura 8: Distribución de modos disponibles en una memoria.

(3) 2 modos asignados a las memorias más grandes (32KB). Gráficamente, esta disposición queda reflejada en la Figura 9 bajo el nombre *Custom mode allocation*.

Comparamos nuestra distribución de modos, *Custom mode allocation*, con otras dos asignaciones de memoria muy significativas: (1) todos los módulos presentan ocho modos, ofreciendo las configuraciones más eficientes en términos de energía y tiempo a costa de requerir el área máxima (*Best energy allocation*); (2) todos los módulos presentan exclusivamente dos modos, ocupando el menor área posible a cambio de configuraciones algo menos óptimas (*Best area allocation*). Tanto *Best energy allocation* como *Best area allocation* aparecen representadas en la Figura 9. Para ver el impacto en tiempo y energía de estas tres distribuciones, simulamos en tiempo de diseño cada una de ellas, empleando valores medios tanto para los parámetros de energía como para el tiempo de acceso por memoria. El resultado de esa simulación queda reflejado en la Figura 10, donde *Custom mode allocation* presenta un consumo de energía muy cercano al realizado por *Best*

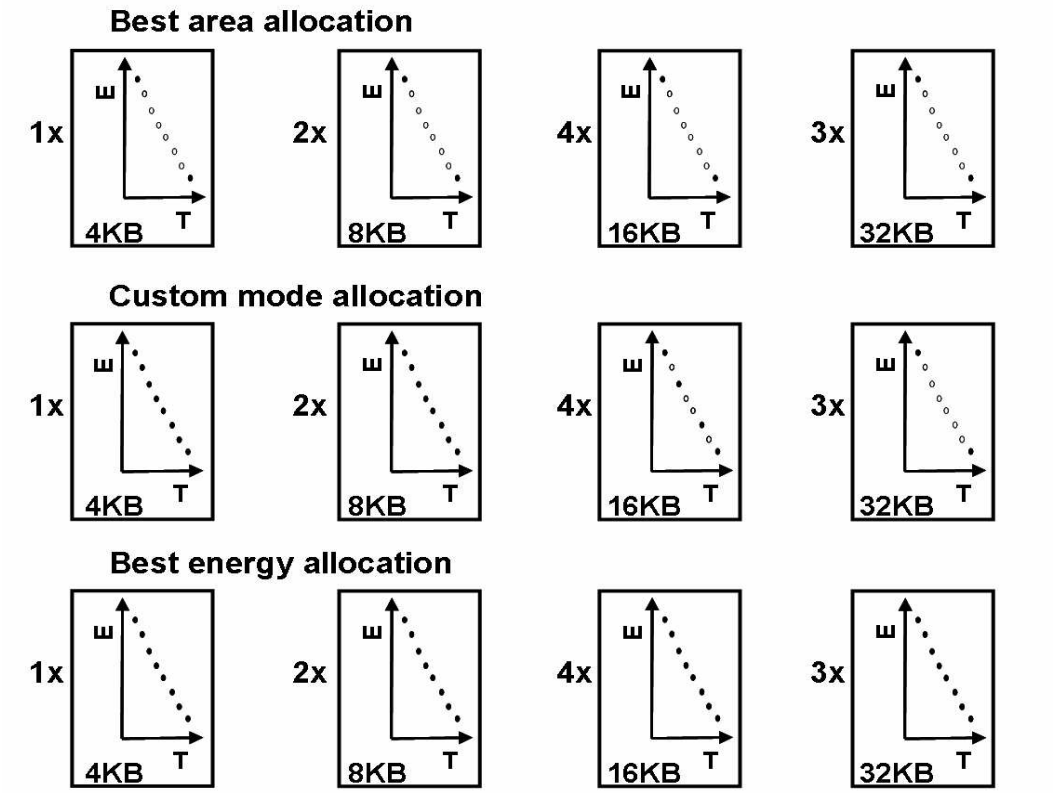


Figura 9: Distribución de los modos de operación en las diferentes asignaciones consideradas para los módulos de memoria.

energy allocation. Aunque los resultados entre *Custom mode allocation* y *Best energy allocation* son muy similares en energía, el área ocupada no es similar. Normalizando el área de las tres distribuciones respecto a *Best energy allocation* obtenemos que la superficie de *Custom mode allocation* es un 25 % menor que la de *Best energy allocation*. El área de *Best area allocation* es apenas un 10 % inferior al de *Custom mode allocation*, lo que refleja el buen comportamiento del algoritmo de asignación de modos.

En cuanto a la versión *base*, la asignación establecida por el algoritmo queda como sigue: (1) 2 modos asignados a la memoria más pequeña (1KB); (2) 8 modos

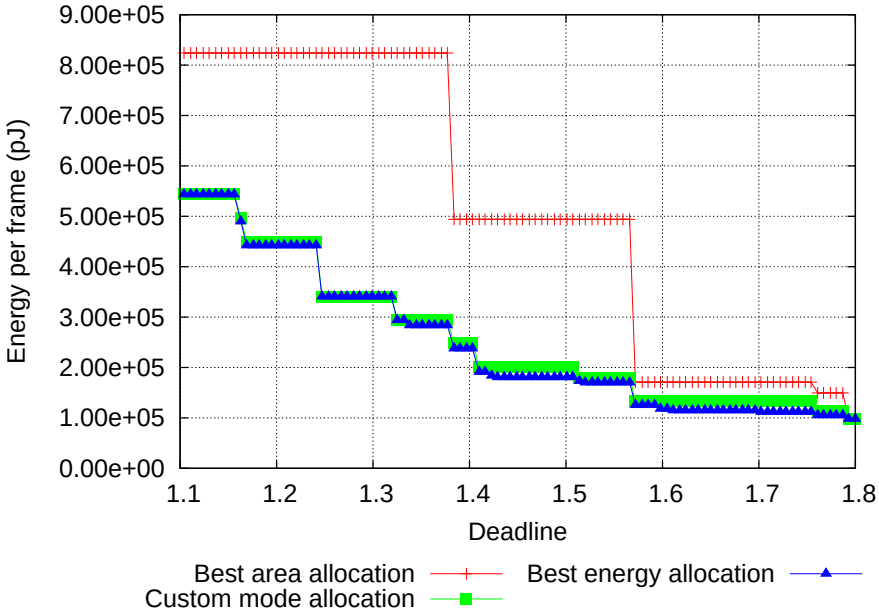


Figura 10: En términos de energía, la asignación proporcionada por nuestro algoritmo es bastante similar a la mejor asignación posible, aunque con un impacto en área mucho menor.

para las memorias de 4KB; (3) 8 modos van a una de las memorias de 8KB, mientras que la otra memoria de 8KB obtiene 4 modos; (4) 4 modos son habilitados en las memorias de 16KB; (5) de nuevo, 2 modos son asignados a la memoria más grande (32KB).

Exploración estadística en tiempo de diseño

Los efectos de la variabilidad en memorias han sido modelados mediante simulaciones de Monte Carlo a nivel de transistor, usando modelos BSIM para 65nm. Estas simulaciones permiten caracterizar, mediante una función de probabilidad, tanto la latencia como la energía por acceso para cada una de los

modos existentes en una memoria. Como se ha mencionado anteriormente, para simular un comportamiento más dinámico en nuestra aplicación objetivo, MP3, asumimos que el tiempo destinado a decodificar un *frame* (*deadline*) puede variar en tiempo de ejecución por la presencia de otras aplicaciones o requerimientos del usuario.

Como resultado de la exploración estadística se obtiene, para cada uno de los *deadlines* considerados, una configuración de los módulos de memoria involucrados en la aplicación, i.e. se indica el modo en que se ha de configurar en cada una de las memorias. El algoritmo elaborado permite asegurar, de acuerdo a un porcentaje determinado, que toda configuración obtenida cumplirá en tiempo de ejecución los requerimientos energéticos y temporales establecidos en diseño. En particular, en nuestras simulaciones hemos trabajado con tres porcentajes diferentes para estudiar el comportamiento del algoritmo bajo variabilidad. Estos porcentajes – 50 %, 80 % y 99 % – indican que toda configuración garantizará en ese porcentaje el cumplimiento de sus *deadlines* con el menor gasto posible en energía. De esta forma, el algoritmo puede generar tanto configuraciones que sólo van a ser correctas en la mitad de las ocasiones, hasta configuraciones que son capaces de cumplir sus requerimientos temporales prácticamente ante cualquier situación de variabilidad (*99 %*). Este último porcentaje nos permite comparara nuestra técnica estadística con enfoques más tradicionales, como los basados en el caso peor.

Como ejemplo, mostraremos los resultados obtenidos con las configuraciones generadas para el escenario *Long* de MP3 y un conjunto de *deadlines*. En base a los porcentajes estudiados, los resultados quedan agrupados bajo la denominación de *statN*, a saber, *stat50*, *stat80* y *stat99* respectivamente. El conjunto de configuraciones obtenidas mediante métodos tradicionales queda denotado como

Base, y corresponde a la versión sin escenarios de MP3 [Lag01] empleando una metodología de diseño en la que se asume que el impacto de la variabilidad será el mayor posible y las configuraciones que permiten garantizar cada *deadline* asumiendo estas condiciones se obtienen mediante búsqueda exhaustiva. El sistema de memoria en cada caso es el obtenido con ATOMIUM. Asumimos siempre memorias configurables, de acuerdo a la asignación de modos establecida por nuestro algoritmo y anteriormente indicada.

Consideramos también las configuraciones óptimas, i.e. aquellas que se obtendrían en tiempo de *setup* una vez conocido el impacto de la variabilidad. Se denotan como *optimum*, y representan la mejor solución en cuanto a tiempo y energía, nuestro límite teórico de optimización. Tenemos en cuenta un cuarto caso, *noStat*, intermedio entre *Base* y *statN* . En él se aplica la metodología conservadora representada por *Base* a la versión de MP3 representada por escenarios. En estos dos últimos supuestos, la arquitectura de memoria es la misma que la empleada para *statN*, la metodología estadística descrita en esta tesis. El conjunto de *deadlines* empleados es común para todos los supuestos.

Generadas las configuraciones, el impacto de la variabilidad bajo ellas se puede ver en la Figura 11, en la que se muestra la energía consumida por cada configuración respecto a su *deadline*. El impacto de la variabilidad se muestra en forma de media - puntos, en la gráfica - y dos desviaciones típicas, tanto en tiempo como en energía. La Figura 11 muestra un buen control del cumplimiento de los dos *deadlines* por parte del algoritmo desarrollado. Fácilmente puede comprobarse con *stat50*, especialmente en las configuraciones para tiempos más restrictivos, que la media queda exactamente encima de su *deadline*. Para *stat99* las simulaciones confirman que en un 99 % de los supuestos de variabilidad, las configuraciones

obtenidas cumplen con los requerimientos temporales. *stat80* no aparece en la gráfica por simplicidad de la figura.

Respecto a la energía, la gráfica muestra una tendencia clara por la cual las configuraciones generadas para ofrecer un mayor cumplimiento de los tiempos, por ejemplo *stat99*, son a su vez las que más energía consumen. A pesar de ello, nuestra versión 'conservadora' (*stat99*) muestra un comportamiento mucho mejor en energía que *noStat*, dada la presencia de la exploración estadística. Comparando *Base* con *noStat* se puede ver además el gran impacto sobre la energía que tiene la existencia de escenarios. Las diferencias en energía van disminuyendo a medida que se relajan las restricciones temporales.

A través de la Figura 11 también podemos ver el excesivo conservadurismo existente en los enfoques tradicionales, ya que nuestro algoritmo estadístico es capaz de ofrecer garantías temporales muy similares a un coste energético mucho mas reducido.

Teniendo en cuenta la diferencia de escala entre los dos ejes de la figura se puede apreciar que el impacto de la variabilidad es mayor en energía que en tiempo, así como el hecho de que éste también se incrementa a medida que los tiempos se hacen menos restrictivos.

Las configuraciones obtenidas mediante nuestra **exploración estadística** en tiempo de diseño no garantizan el cumplimiento de las restricciones temporales al 100 %. Por ejemplo, una configuración generada para *stat50* incumplirá su *deadline* en uno de cada 2 dispositivos. Sin embargo, un pequeño ajuste una vez terminado el proceso de fabricación del dispositivo puede asegurar su correcto funcionamiento para todos los *deadlines* en todos los dispositivos. Para ello, en tiempo de *setup*, el sistema puede ser monitorizado mientras se ejecuta la aplicación. Así, para cada uno

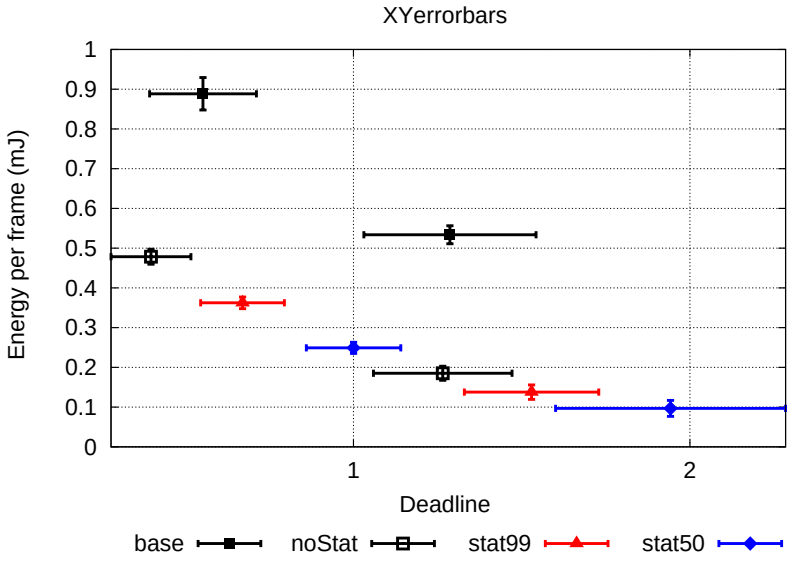


Figura 11: Altos niveles de cumplimiento llevan asociados configuraciones más conservadoras para mantener un sistema altamente fiable.

de los *deadlines* esperados se comprueba el cumplimiento de los mismos. En caso de que una configuración incumpla su deadline, esa configuración será sustituida por otra, correspondiente a alguno de los *deadlines* que le preceden. En la mayor parte de las ocasiones, salvo casos en los que la variabilidad es muy elevada, la nueva configuración será la correspondiente al *deadline* inmediatamente anterior.

Aplicando esta reordenación a las configuraciones presentadas en la Figura 11 obtenemos los resultados presentados en la Figura 12. Ésta muestra la energía media resultante tras la reordenación en un entorno bajo variabilidad, siendo los valores relativos a *Base*. Tras la reordenación, todos los diferentes enfoques considerados cumplen con todos sus requerimientos temporales.

En este caso, nuestra metodología proporciona significativas reducciones de energía en comparación con los diseños basados en el caso peor. Así, en función

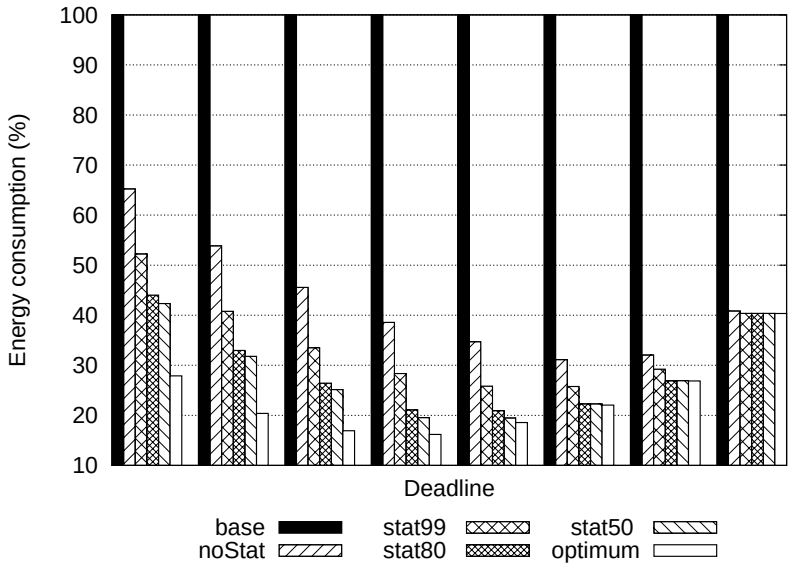


Figura 12: Las configuraciones con bajos niveles de cumplimento temporal son las más eficientes en términos de energía, aunque a expensas de incrementar los dispositivos que requieren una reordenación para cumplir con sus compromisos temporales.

de la restricción temporal, las reducciones van desde un 58 % a un 80 % en *stat50*. *stat80* también presenta reducciones similares con una menor necesidad de llevar a cabo reordenación de las configuraciones. Para *stat99*, la necesidad de reordenar las configuraciones se reduce drásticamente y la energía sigue siendo al menos un 50 % respecto a *Base*. De nuevo, el ahorro debido a los escenarios queda reflejado en el 30 % mínimo de energía que separa a *base* de *noStat*. Nuestra técnica probabilística añade un 10 % a estas reducciones.

Refinamiento en tiempo de *setup*

A pesar de las mejoras obtenidas por nuestra exploración estadística en tiempo de diseño respecto a las técnicas tradicionales, y dado que no buscamos desarrollar un modelo muy preciso de la variabilidad, todavía podemos acercarnos más a los resultados que ofrece *optimum*. Aplicando el algoritmo de **refinamiento en tiempo de *setup***, podemos modificar las configuraciones obtenidas estadísticamente en tiempo de diseño para mejorar sus resultados tanto en tiempo como en energía. Los resultados de este proceso se ofrecen en la Figura 13 y se aplican tanto a las configuraciones *statN* como a *noStat*, normalizando los resultados con *Base*. *Optimum* permanece sin cambios. Como consecuencia del refinamiento se producen nueva y significativas reducciones de energía. En el caso de *noStat*, las nuevas configuraciones consumen hasta un 25 % menos energía que antes del refinamiento. Entre las diferentes configuraciones estadísticas las reducciones son mayores en *stat99* que en *stat50*. Para la mayoría de los *deadlines*, se muestran reducciones de entre 10 % y 15 % en *stat99*, mientras que el porcentaje se reduce a 5 % para *stat80* y *stat50*. A pesar de este proceso de refinamiento se sigue poniendo de manifiesto que las configuraciones que menos energía consumen son las que presentan, por diseño, un bajo cumplimento de las restricciones temporales. Es el caso de *stat50* que muestra resultados en energía muy próximos al óptimo, mejorando los resultados de *base* más de un 60 % en la mayoría de los casos.

Dado que el proceso de refinamiento se lleva a cabo en tiempo de *setup*, el proceso debe ser lo más rápido posible. Esto puede suponer una limitación en el número de modificaciones que se pueden aplicar sobre una configuración dada. La Figura 14 muestra sobre nuestros supuestos los resultados de limitar el proceso de

refinamiento a un número de pasos fijo y reducido, ampliable hasta que se cumpla el *deadline* en aquellos casos que lo requieran. Los conjuntos de configuraciones *noStat* y *stat99* resultan ser los casos más afectados por la limitación, ya que requieren entre tres y cuatro veces más pasos para mejorar sustancialmente las configuraciones iniciales. El resto, *stat50* y *stat80* muestran, a pesar de la limitación de pasos, resultados más cercanos a los que se alcanzan sin ella.

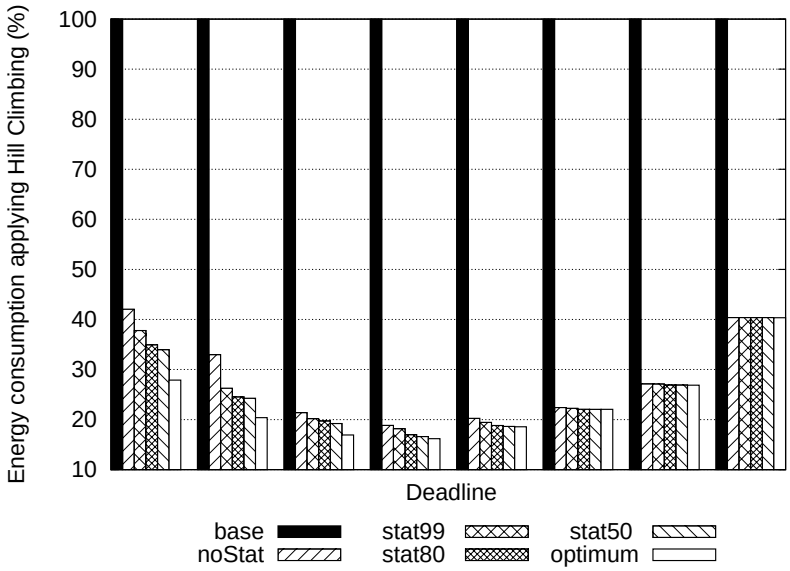


Figura 13: Energía media consumida tras el proceso de refinamiento llevado a cabo entre las configuraciones obtenidas en tiempo de diseño.

Aplicaciones sintéticas

Mediante la creación de aplicaciones sintéticas se ha podido corroborar el comportamiento de la metodología ya reportado para MP3. Así, tomando esta aplicación como patrón se han desarrollado una serie de aplicaciones sobre las que probar la metodología desarrollada. Las simulaciones, considerando las mismas

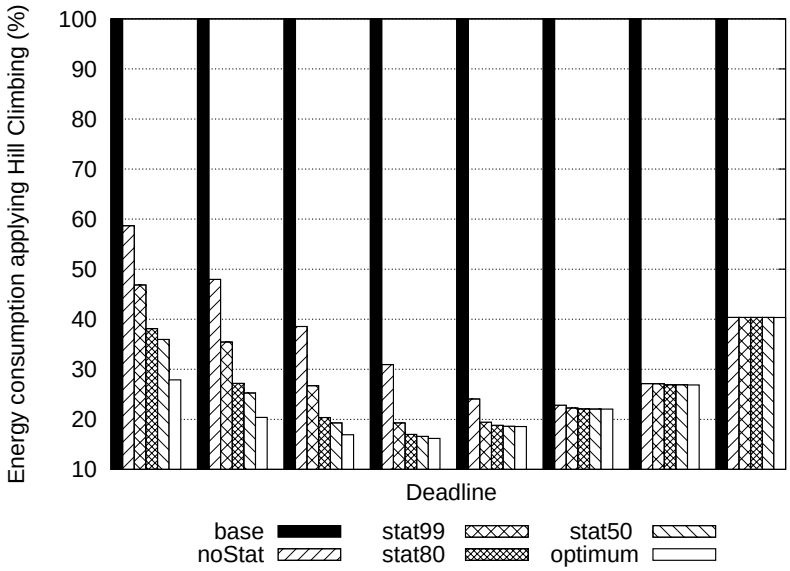


Figura 14: Energía media consumida limitando el proceso de refinamiento.

condiciones que para MP3 en cuanto a variabilidad y cumplimientos temporales, ofrecen las mismas tendencias en cuanto a asignación de modos a memorias, exploración estadística, reducción de energía y refinamiento de las configuraciones.

Conclusiones

A lo largo del tiempo la mejora de la tecnología, especialmente con tecnologías por debajo de la micra, ha permitido una continua mejora de las características presentes tanto en SoC como en SiP, tales como el rendimiento, el área, la capacidad de memoria y la potencia. Sin embargo, a pesar de los claros beneficios del escalado, nuevos desafíos se han hecho presentes. A lo largo de esta tesis nos hemos centrado en dos de ellos, cada uno de diferente naturaleza: variabilidad y dinamismo.

La variabilidad, un fenómeno asociado a la falta de control de los parámetros tecnológicos, viene siendo tratada desde hace tiempo por los diseñadores. Sin embargo, a consecuencia del extremo escalado tecnológico llevado a cabo en la última década, su impacto se ha incrementado de forma muy significativa.

El dinamismo, sin embargo, es una característica reciente de las aplicaciones, que afecta sobre todo a aquellas que se ejecutan en entornos empotrados basados en SoC y SiP. Estos dispositivos, tradicionalmente caracterizados por restricciones relacionadas normalmente con la energía, tienen que hacer frente a aplicaciones cuyo comportamiento en tiempo de ejecución es altamente variable y difícilmente predecible.

Ambos problemas implican un comportamiento incierto en tiempo de ejecución, tanto a nivel de plataforma como de aplicación, que tradicionalmente se ha manejado mediante diferentes técnicas de diseño para asegurar el rendimiento aunque a expensas de un mayor consumo de energía. En esta trabajo hemos desarrollado una metodología para afrontar ambas fuentes de incertidumbre de manera coordinada, de forma que los requerimientos temporales se cumplen con un gasto de energía significativamente inferior a las técnicas tradicionales. Hemos centrado este trabajo en el sistema de memoria, dado que es uno de los componentes más afectados por la variabilidad. Además, nuestra metodología se centra más en dominios específicos, tales como las aplicaciones multimedia, que en sistemas generales o aplicaciones específicas dado que las características de estos dominios están bien definidos y pueden ser explotados de forma satisfactoria.

Los pilares de la metodología presentada son las memorias configurables y el concepto de escenarios. Las memorias configurables permiten mitigar la variabilidad y evitar estimaciones pesimistas mediante el uso de un número variable

de modos de trabajo. Los escenarios nos permiten describir las aplicaciones como colecciones de diferentes comportamientos en tiempo de ejecución. Una vez que la aplicación queda caracterizada de esta forma, se vuelven más predecibles y permiten evitar las soluciones conservadoras que reducen el rendimiento.

Estas dos ideas son combinadas en una misma metodología que se ejecuta durante todo el tiempo de vida de la plataforma, desde el diseño hasta la ejecución de la aplicación. Los principales aspectos que caracterizan esta metodología se muestran a continuación:

- Las aplicaciones son analizadas y los escenarios extraídos
- El sistema de memoria se establece conforme a criterios que tienen en cuenta el consumo de energía y el análisis de las aplicaciones con el objeto de mejorar la asignación de datos.
- Los módulos de memoria son todos configurables. El número adecuado de modos en cada una de ellas es establecido en tiempo de diseño.
- La energía y el rendimiento del sistema dependen en gran medida de la forma en que las memorias son configuradas en cada momento durante el tiempo de ejecución
- El impacto de la variabilidad sobre las memorias es modelado de manera estadística para cada una de las memorias y de los modos existentes en el sistema. Un estudio en tiempo de diseño determina estadísticamente el modo en que las memorias han de ser configuradas para cumplir con los requerimientos temporales con el menor consumo de energía. Dado que las

aplicaciones están caracterizadas en escenarios, este paso se lleva a cabo para cada una de las restricciones temporales en cada uno de los escenarios

- Antes de comenzar con la ejecución de la aplicación, las configuraciones generadas en tiempo de diseño para cada escenario son actualizadas con la información de la variabilidad en ese momento

- En tiempo de ejecución, las aplicaciones son monitorizadas para determinar el escenario en el que se encuentran en cada momento y reconfigurar las memorias de acuerdo a ese escenario y las actuales restricciones temporales

Los resultados de nuestra metodología muestran ahorros en energía significativos respecto a técnica basadas en estimaciones de caso peor. Para MP3, las reducciones alcanzan el 85 %. Tanto la presencia de escenarios, como el método estadístico empleado durante el diseño y la etapa de actualización llevada a cabo en tiempo de *setup* contribuyen a estos significativos ahorros. La metodología desarrollada ha probado reducir el consumo de energía mientras se mitiga de manera coordinada el impacto de la variabilidad y el dinamismo de las aplicaciones, evitando así las técnicas más conservadoras. Además del ahorro en energía, se mejora el rendimiento debido a los escenarios.

1

Introduction

In 1965 Gordon Moore foretold that the transistor density would be doubled every 18 months. His empirical law has been fulfilled since them, helped by a continuous reduction in the technology feature size. Currently, we are in the nanotechnology era with features around 32nm, well lower than the 100nm features reached in the past decade. According to predictions established by the Semiconductor Industry Association (*SIA*) [SIAR10] in 2010, the current decade will be still characterized for an improvement in the technology scaling, which will keep the Moore's law largely effective. As an example, near-term estimations foresee the existence of 16nm Flash technologies by 2016, while DRAM will reach these features by the end of the decade according to long-term predictions. The feature size of these two technologies is believed to be under 10nm by 2024, scaling down to 7.5nm for MPU/ASIC. These feature sizes are close to what it is considered as the limit of silicon technology, 7nm, a size which corresponds to about 30 atoms in a silicon crystal [SGK11b].

The technological advances accomplished for years have made possible the integration on a single chip of millions of transistors. In 2009 the transistor density was over 2,000 Mtransistors/cm^2 in SRAMs, and according to *SIA*, this number will increase up to 40,000 by 2020, reaching 110,000 Mtransistors/cm^2 four years later [SIAR10]. This vast increment in the integration capacity has given rise to the fast development of System-on-Chip (*SoC*) and System-in-Package *SiP* solutions.

1.1. System-on-Chip and System-in-Package

Over the years, the increasing integration capacity has led to the realization of more and more complicated designs either in the same package (SiP) or on the same chip (SoC). Thus, driven by the rapid growth of communication technologies, pervasive computing, and consumer electronics, chips have moved from stand-alone ASICs – medium complexity circuits based on less than 500,000-gate designs and application specific – to integrated and multifeature SoCs and SiPs.

A System-on-Chip integrates onto a single chip a set of complex heterogeneous components, each of them of diverse functionality, to devise a complete system which performs a more complex function. Thus, a SoC design typically consists of a set of components such as programmable processors, dedicated hardware to perform specific tasks, on-chip memories, input/output interfaces, and an on-chip communication architecture to interconnect all the components. Since everything is included on the same chip, SoC technology allows relevant improvements in terms of system performance, die size, unit cost and power.

A System-in-Package is a functional system assembled into a single package. The systems include one or more integrated circuits, including System-on-Chip,

and passive devices with multiple interconnections. This makes possible to combine multiple technologies to form a complete system despite being in different chips.

The same technology that provides these levels of integration, also raises challenges derived from the deep sub-micron technology involved, from verification or HW/SW co-design problems to increasing design complexity which enlarges the design process and reduces its productivity. Designers also have to deal with a higher demand for more functionality, lower cost, tighter time to market requirements and lower power consumption. In order to reduce some of these challenges, the components involved in a SoC are usually developed separately as IP (Intellectual Property) blocks to improve re-use and reliability, and be later combined in new designs to decrease the time to market. These IP blocks can be a library element from a previous design, a parameterizable component to meet design requirements, or a custom-designed computational block to meet performance, power, and other design constraints. The fact that the characteristics of the application domains mapped to SoCs and SiPs are usually known beforehand contributes to the process design, being possible a deep analysis in order to develop the final system.

SoCs are primarily used in computers, communication equipment, consumer electronic devices and automotive applications; giving rise to a business, whose revenues will increase from $45,294M in 2010 to $69,405M by 2015 [Res]. End-use applications, which include smart phones and automotive electronics, are nowadays driving the market growth. Some of the major SoC applications include multimedia applications which involve speech, image, audio or video signal processing algorithms, and data communication such as wireless applications. The leading applications for SiPs are pretty similar to the ones for SoCs: portable consumer products such as cameras or mobile phones. The choice between one kind of integration or

the other depends on several factors such as the compaction in the system, time to market, production volume, or the need for mixing technologies or design methods.

In both cases, *SoC* and *SiP*, the target applications are usually characterized for being embedded on battery-powered handheld devices, having real-time constraints to meet, and being data-dominated. This data dominance translates into devices where most of the area available is devoted to memory modules. As a result, a large part of the power consumed in the system comes from the memory system, and the storage and data transfer operations involved in it. The relevance that the memory system acquires is not only restricted to the power domain, but also affects the performance domain – which is also largely dominated by them –, and the reliability of the system.

In System-on-Chips and System-in-Package, the need for an efficient global design is significant due to all the different requirements driving the market. Among all the challenges affecting the future SoC and SiP design and enumerated previously, in this research we focus on the following ones because of the uncertainty introduced in the design process:

- Application dynamism.

  The continuous technology scaling has broadened the range of applications targeted by SoC. Currently, users demand complex applications that are usually performed on computers to be executed on a SoC, despite the restrictions, mainly related to power, that these systems face. Their complexity lies in the largely variable behaviour that these applications can exhibit at run time. This dynamism requires a proper management not only to cope with the application timing requirements, but also with the system power constraints.

- Process variation.

  As we mentioned above, smaller technology nodes turn technology issues that were negligible in the past as challenges. This problem, known as process variation, is another source of uncertainty. It affects the platform due to the lack of control about technological parameters which are relevant on the design, such as the delay or the power consumption. Particularly, here we focus on the effects that variations have on memories.

- Reliability

  While process variation is mainly used for time-independent problems, reliability is mostly associated with time-dependent degradation issues. Besides the alterations suffered from transistor parameters – such as threshold voltage – during the fabrication process, their values can later be altered again along the lifetime of the component due to environmental aspects related with voltage drops or temperature fluctuations. This situation, related with the stress applied to the circuit, i.e. the current and voltages applied to a transistor, gives rise to the degradation of the component over the time which can turn a functional circuit into a non-functional one.

1.1.1. Process variation

Aggressive technology scaling historically has improved the performance metrics of SoC, both in terms of energy consumption and performance. Unfortunately, scaling the minimum feature sizes in deep-submicron technologies is not business-as-usual anymore, bringing a host of problems which cannot be completely solved at the technology level alone [BKD04]. Effects that were always considered as

second order are becoming very pronounced and severely affect the platform. On a chip, variations in temperature, power supply and switching activity due to environmental factors, and mismatches in electrical and physical parameters due to processing and mask imperfections, are the main issues affecting performance and power consumption on integrated circuits. These alterations are due to the inability to precisely control the fabrication process in small-feature technologies. The effects can be classified as systematic or random, depending on the spatial correlation exhibited. This spatial feature is closely related to the physical parameters affected by variation. In general, it can be said that systematic variations are due to problems with lithography – mask, lens, etc –, while random effects are caused by variable dopant density and alterations in the line edge roughness.

One of the major issues pointed out by the Semiconductor International Association [SIAR10] is intra-die process variability. This kind of variation affects electrical and physical parameters on a circuit, such as the gate length or the threshold voltage at the transistor level, compromising the behaviour and yield of the devices in the die. Also power is severely affected, especially its static component, leakage, which increases exponentially as a consequence of these mismatches in physical parameters.

Along the years, different techniques have been developed to improve tolerance to variations. In the literature we can find pure process mitigation techniques, pure design mitigation techniques and combinations of both approaches to deal with systematic and random variations [KKK+08]. Random variations are mainly coped with process techniques, although design approaches have been also applied. In this dissertation we specially focus on these variations, providing an alternative to the current design techniques.

The stochastic behavior introduced by process variability has been traditionally tackled by corner point design methodologies and the addition of design margins to improve the predictability of the circuits behavior [ZHHO04a]. These designs usually consider 3σ points to capture variations completely. However, variability is increasing. As an example, research focused on the enhancement of the timing yield at the circuit level reports that variation can cause up to 30% of mismatch in chip frequency [ABZ03, FP09]. In this situation, traditional values for corner points are becoming unable to bind the increments, forcing to expand the margins, in some cases without guarantees to cover the whole variation. Recently, it has been reported designs up to 6σ for 28nm technology [Gup11]. The increasing variability turns these techniques no longer sustainable to deal with the problem, becoming too conservative and leading to severe overheads in energy consumption and performance [CQS04].

Variability affects every single component in a system, but specially affects memory system. As we mentioned above, a significant part of the chip area is devoted to memories, which contributes to the majority of the energy consumption. The memory system usually consists of minimum-sized transistors to improve the performance and reduce energy. However, these characteristics also make them more prone to large performance and energy variations, since the variability impact at the cell level is higher in these minimized transistors [CDSM04]. As a result, we show in Section 2.3 how process variability has a significant negative impact on the access time and energy consumption of individual memory modules. After fabrication, the actual values of these two parameters – access time and energy – are always higher than the nominal specifications established at design time. As a consequence, these heavily active memory modules become less predictable in terms

of energy and performance, compromising the fulfilment of a given clock frequency or power budget [HCM+05].

1.1.2. Application dynamism

In the past, complex, highly data demanding and event-driven applications were only feasible on general purpose processors, while applications running on SoC usually showed a more static behaviour. However, the advances in technology have widened the applications available in the SoC market, evolving to multimedia, personal communications, broadband network or ambient intelligent. Also new features and services have been added to these systems, giving rise to a more dynamic behaviour. This diversification in applications and the increasing number of users in the SoC/SiP market made necessary to develop standards for such applications. Despite the different standards existing at the moment focused on different applications – image, video, audio –, all of them increase performance by means of higher compression, although at the expense of a growing complexity to manage the vast amount of data involved in these applications.

As an example of these standards we can find the one developed by the ISO Motion Picture Experts Group (MPEG) since 1992 [Sik97]. The target was a coded representation of moving pictures, audio and their combination at a bit rate up to 1.5 Mbit/s. The standard, known as MPEG-1, was extended in 1994 to cope with bit rates up to 10 Mbit/s (MPEG-2) that applications such as digital cable TV or satellite and terrestrial broadcasting exhibited. The bit rate not only increased, but also could change among the frames in the video or audio. By 1999 the standard was extended to tackle with new challenges coming from multimedia applications,

such as high interactive functionality, high compression efficiency or robustness in error-prone environments (e.g. mobile channels). This standard, known as MPEG-4, is characterized for its interactivity based on content, i.e. it is allowed to access and manipulate audio-video objects at the coded data level. Thus, a sequence containing a number of video objects is coded in different layers, one per object, so the bitstream of each object can be decoded and reconstructed separately, making possible the addition and deletion of objects from the scenes.

The initial standard evolved to cope with more complex and dynamic situations, giving rise to standards such as H.264/MPEG-4 AVC (Advanced Video Coding) in 2003 and SVC (Scalable Video Coding) in 2007, which are the result of a joint project between the ITU-T's Video Coding Experts Group and the ISO/IEC Moving Picture Experts Group (MPEG). AVC, a video compression standard designed to address several weaknesses existing in previous standards, reduces significantly the bit rate necessary to represent a given level of perceptual quality without an increment in the complexity of the design. AVC allows highly efficient and reliable video communications, supports *streaming* and broadcast applications and remains robust to channel transmission problems. SVC [Cod], specified in Annex G of H.264/MPEG-4 AVC, is focused on the scalability of video coding. This scalability can be temporal, spatial or quality scalability, affecting the frame rate, the resolution and the data rate respectively. This standard allows an easy adaptation to different terminal capabilities and multicast streaming through heterogeneous networks.

An example of dynamism can be found in Figure 1.1. This figure illustrates the dynamism present in a movie, based on how the bit rate varies among the different

samples[1]. As can be seen, there are samples which exhibit a high bit rate whereas others have very low rates. A high rate corresponds to scenes with a large amount of camera movements happening in a short period, while the lower rates correspond to static scenes where the camera moves very slowly.

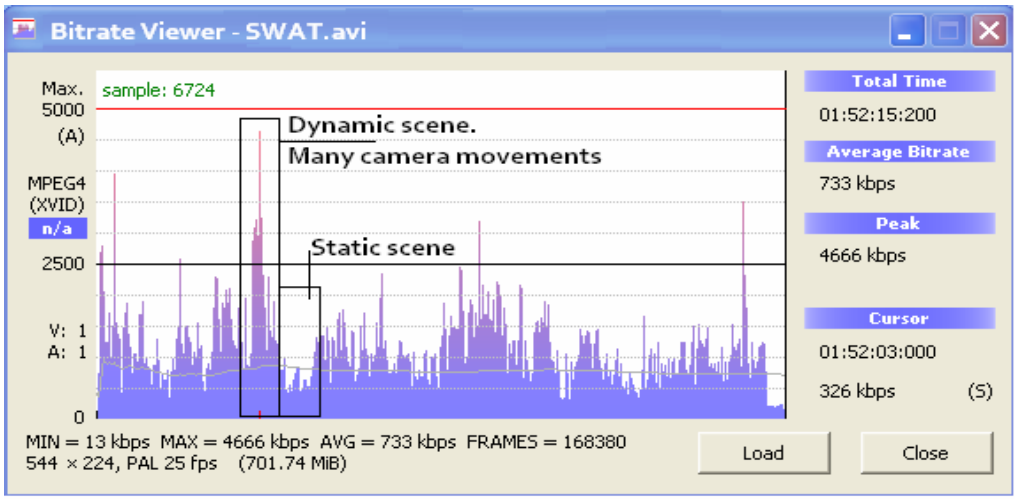


Figure 1.1: Examples of dynamism can be seen on this MPEG−4 variable bit rate video. Large variations in the bit rate among samples differentiate between static and more dynamic scenes.

Currently, multimedia applications can exhibit very dynamic execution time characteristics per frame. Resource usage in terms of computation, memory or communication can also vary largely, compared with former applications in this domain. Multimedia quality of service, timing requirements or other applications competing for common resources are also sources of dynamism at run time.

This application dynamism has usually been dealt statically by means of design time methodologies, being the most common approaches those based on the assumption of the worst-case for the application behaviour. However, as dynamism

[1]The application *Bitrate Viewer* [Vie] has been used to visualize the bit rate variation existing in MPEG files.

increases, the actual application behaviour is situated very far from the static case and not predictable any longer at design-time. This in turn makes these design time techniques too conservative, leading to performance losses and energy overheads.

1.2. Integrated approach to tackle uncertainty

The effects of variability on performance and energy have been largely considered during the design process, as well as the dynamic behaviour existing on the applications. As a result, a large variety of techniques has been developed to face both challenges, from purely design-time techniques to combinations with run-time approaches. Both problems were initially managed by means of design margins and worst-case techniques. Later, other alternatives came up. Specially remarkable it is the case regarding process variation, where the alternatives work at very different levels: circuit, system, architecture, memory system, etc.

Despite all the research done, conventional approaches for mitigating process variation and application dynamism ignore the combined impact of both problems. Variability mitigation techniques reduce the applications to static and predictable codes, so no other variable behaviour apart from the platform is taken into account. On the other hand, application mapping techniques ignore the platform, assuming that reliable information about its performance and energy consumption is available at design time. Both approaches usually lead to conservative assumptions related to the operating frequency for example, in order to make sure that variations, due to process variability or dynamic behaviour, do not endanger the application synchronization. In both cases, a lot of performance is left on the table.

Future systems are expected to run dynamic applications and be built on hardware platforms fabricated using aggressive technology nodes. Solutions dealing with both types of uncertainty are required in order to enable optimal energy efficiency and system performance. In this research, we explore an integrated approach to manage process variability in a domain of dynamic applications. Our proposal combines application pre-processing at the compilation level and a statistical analysis to mitigate variation in memories, with hardware monitoring and calibration at run time to adapt the platform to the effects of variability and application dynamism. The aim is to offer configuration opportunities at run time to maximize the timing parametric yield and the energy efficiency at the memory level which translates into a better performance and lower energy at the system level. This approach is based on the following concepts and methodologies, which are explained along this dissertation:

- Exploiting flexibility adding different operating points to on-chip memories (*Configurable memories*)
- Comprehensive management of the dynamic behaviour exhibited by applications at run time (*System scenarios*)

1.2.1. Configurable memories: dealing with uncertainty at the memory level

Since each fabricated chip and each of its components have different characteristics under process variability, some authors have pointed out it is no longer practical to design a system using a fixed and rigid configuration [LB06]. Following this general design principle, our methodology mitigates the impact of variations on the

memory organization through configurable memories coupled with a system level feedback control loop [PLW+05].

To motivate our choice for configurable memories we use the following example. Consider a regular memory where only a single operating point is available. Consider also, that this memory is the only one in a system with a specific deadline to meet. If we are unaware of variations, at design time we will be tempted to set the operating frequency based on the memory access time expected at this point. Later, at run time we could realize that the chosen frequency is too high, as variation has increased its access time. As a result, the memory turns inoperative. On the other hand, if we are aware of variation, we can set a conservative frequency. In this case, we could be able to meet the access time, but wasting time and energy.

However, configurable memories, which are the focus of Chapter 2, are designed to have two or more operating points, which provide an adaptable energy-delay balance. At run time, using the appropriate methodology, the existence of several operating points are used to avoid worst-case margins and get a better adaptation of the memory system to the current variability impact. Back to our example, this implies that no frequency is set at design time. After fabrication, once the current access time is known, the most suitable operating point to meet the deadline is selected. In case the deadline varies along the execution, the operating point can also be adapted at run time to meet the new timing requirement. This capacity to adapt memories to the current situation allows to apply this technique not only to deal with variations caused during the manufacture process, but also to deal with those time dependant variations which may appear along the lifetime.

Previous research on configurable memories [PLW+05] has already shown that this run-time tuning approach is effective in dealing with parametric memory varia-

tions in the context of periodic static applications. In this dissertation we extend the initial methodology to handle dynamic applications. This technique is not reduced to a particular architecture or application, being applicable to any architecture or domain specific application.

1.2.2. *System scenarios*: dealing with uncertainty at the application level

To illustrate the main ideas behind *system scenarios*, let's consider as motivational example a simple frame-based application. We assume that the frame rate remains constant during the entire execution – for instance, 24 frames per second –. The application core, as Figure 1.2 shows, consists in a main loop controlled by a user-dependent variable which determines the computational workload inside the loop. For the sake of simplicity, the variable admits only two possible values, leading to the computational units illustrated in Figure 1.2. These units represent the execution time of the loop assuming that the processor core operates at its maximum frequency.

A static methodology would set, at design time, the operating frequency that tolerates. In our example, if the design is conservative, the frequency would be determined by the user value which generates the larger execution time, i.e. the worst case. In this way, the timing requirements would always be guaranteed, although at the expense of using always the same high frequency (case A in Figure 1.3). If we opt to relax the constraints, we can estimate the average number of iterations and use it to set the frequency. In this case, the frequency would be lower at the expense of losing the deadline under demanding workloads (case B in Figure 1.3).

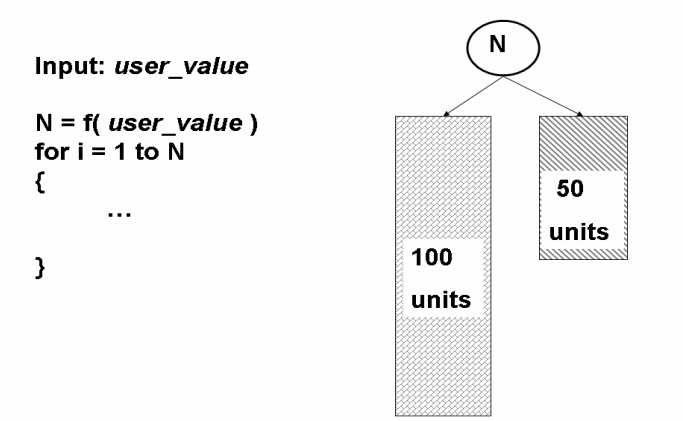


Figure 1.2: Frame-based application with variable workload at execution time.

To tackle with the existing dynamism and avoid the use of conservative design time approaches, we rely on a methodology where the run-time decisions are not completely taken during the system design stage, but also during the application execution. This methodology, which is explained in depth in Chapter 4, consists in the optimization of the application during its whole lifetime, starting at design time until the execution time.

The methodology is based on the existence of *system scenarios*, a concept which allows to cluster the application behaviour in representative and deterministic patterns clearly recognizable at run time. These patterns, discovered and characterized at design time, can be seen as control flows to be executed depending on the application data, the user inputs or the cost metrics used in the system. The scenario information is later used at run time to predict, in a certain way, the short-term application behaviour and provide at the system level an answer customized to the current application conditions. In general, all the system level techniques that need to guarantee some real-time performance metric for dynamic applications, rely on

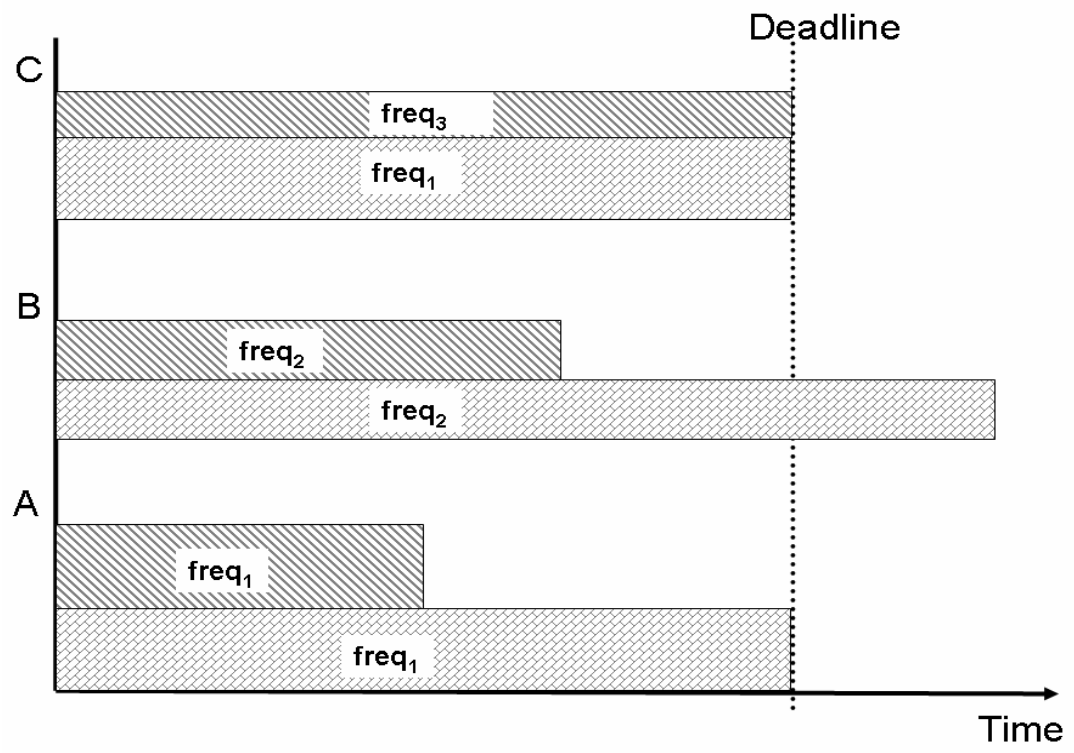


Figure 1.3: Accomplishment of timing requirements under different approaches.

an estimation or prediction – in the same way as *system scenarios* do –, in order to efficiently adjust the respective configuration parameters [TMQW06, KUM+10]. A detection mechanism identifies which *system scenario* is currently present, based on the inputs and the application control flow, and allows to adapt the system to the current conditions with the minimum cost and maximum timing guarantees.

Based on the *system scenario* concept, and going back to the motivational example, at design time we would consider two *system scenarios* based on the input variable. Once the *system scenarios* have been discovered, each one is associated with a single frequency. Later, at run time, the *system scenario* identification is done

in every frame once the user value is known, and its associated frequency is applied (case C in Figure 1.3).

Sometimes it is not possible to identify every single situation which appears at run time and match it with a scenario. In these situations there must be an extra *system scenario*, denoted as *backup scenario*, which collects them to ensure a proper functionality even at those unexpected situations. In our motivational example, the activation of the *backup scenario* would mean applying a faster frequency than the ones established for the two scenarios clearly identified. The *backup scenario* can be considered as the only point in the methodology where worst-case estimations still remain.

Given the low rate of use that a backup scenario exhibits, we can conclude that *system scenarios* avoid conservative timing estimations and enable a better-than-worst-case application mapping, as is shown in Section 4.3.2.

1.2.3. A basic methodology as initial approach

Before introducing the complete methodology developed in this dissertation, we use a simplified version of it to show the benefits of combining *system scenarios* and configurable memories in a collaborative approach. Next, we outline this base approach. The details are discussed later in Chapter 5, showing its benefits in terms of energy savings and performance.

Our simplified methodology to handle dynamic applications and map them onto configurable memories is depicted in Figure 1.4, and it is presented as a three-step process closely related to the platform lifetime. The methodology is initiated at design time supporting the platform development. During the so-called setup time, i.e.

sometime after the fabrication process and before the application starts executing, the platform is calibrated to take into account the current variability impact and fit the system to the application requirements. At run time, the methodology continues assisting the platform to fulfil the timing requirements with a reduced energy consumption.

Going into details in Figure 1.4, we find that at design time the application is profiled and the *system scenarios* extracted. *System scenarios* characterize the application in terms of execution time, energy and memory access patterns. This information allows us to recognize and predict the application behaviour at run time. Still at design time, this profiling information together with in-depth analysis of the source code is used to perform a memory partition and data assignment which provide an energy-efficient memory organization. This last task is carried out following the application mapping methodology known as *DTSE*, which is explained in Section 3.3.1. This methodology allows to build suitable memory architectures for a better data management, which translates into significant energy savings at the memory level. All the memory modules used in the system are configurable.

At setup time, the memory system is measured by means of monitors to know the variability impact on the energy and delay values. After this, compensation techniques explained in Section 5.2 are performed for each *system scenario* and time constraint expected at run time. These techniques establish the best way each memory in the system has to be reconfigured to meet a specific time constraint with the minimum cost in terms of energy. They provide the operating point that each memory uses during the application execution when its *system scenario* is active. For a set of deadlines, we obtain a set of configurations balanced in energy and time, meaning by this that a configuration for a strict deadline never requires less energy

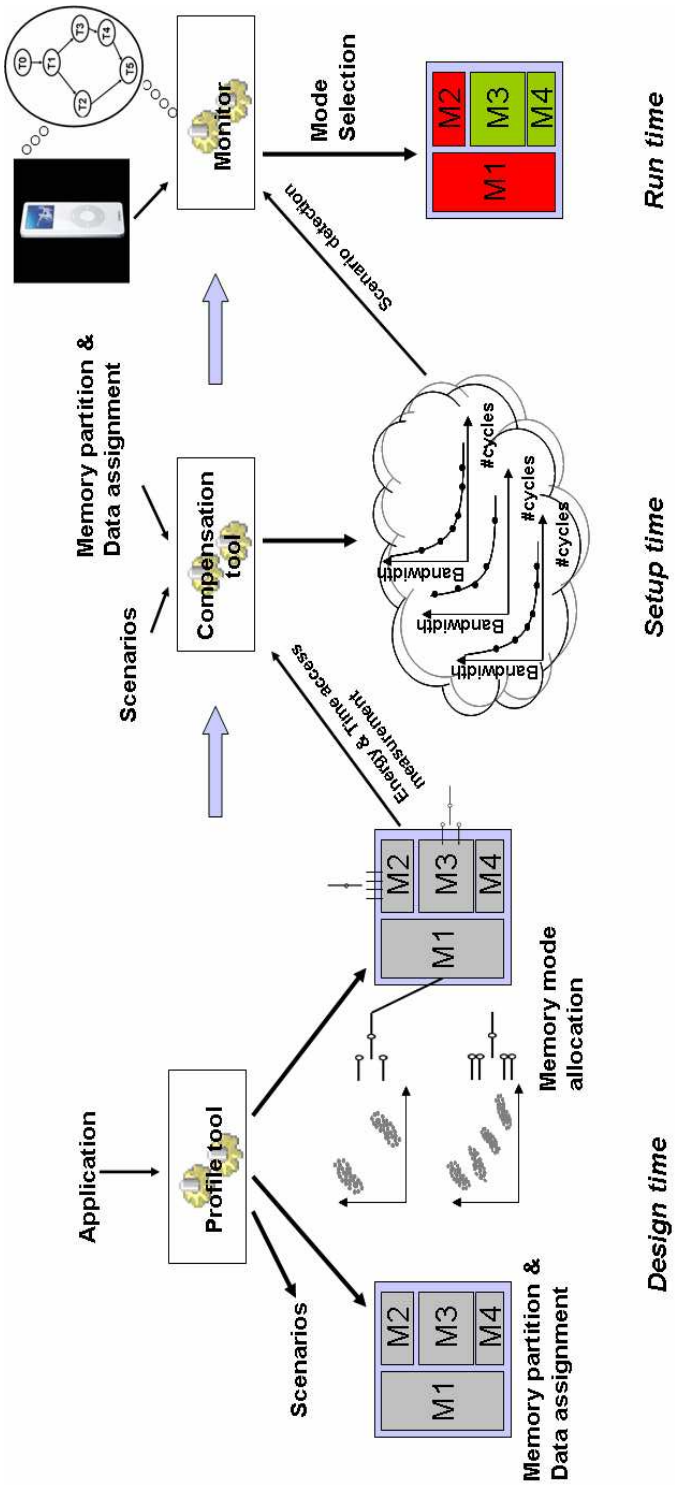


Figure 1.4: System methodology. Several orthogonal steps are applied first at design time and later during setup and run time.

than a configuration for a lenient deadline. In this way, each set of configurations can be seen as a Pareto curve, where each point represents a unique configuration based on its energy and time consuming. From now on we use the term Pareto curve to refer to the set of configurations associated with a *system scenario*.

Later, at run time, the system is monitored looking for changes in the *system scenario* or the timing constraint to meet. If the situation changes, the memory system is updated by selecting from the available configuration sets the one which better allows to meet the timing requirements with the lower cost in energy.

1.3. Holistic methodology

The basic methodology described in the previous section relies on a setup step that performs the heaviest part of the methodology, i.e. the compensation mechanisms that provide the set of configurations to use at run time. However, it would be desirable to perform as much work as possible at design time – off-line mode – to minimize the overhead in the later steps. This overhead reduction, the energy optimization and the dynamism management are the main goals of this dissertation. To achieve these goals, the complete methodology keeps the three-step process illustrated in Figure 1.4 and restructures the design and the setup steps as shown in Figure 1.5.

This comprehensive methodology requires the design time step to be extended with new tasks. Thus, once the memory system has been determined by means of the initial tasks – application profiling, *system scenario* detection, memory allocation and data assignment –, a new step is introduced to the design-time optimization process, the selection of the number of operating points that will be assigned to each

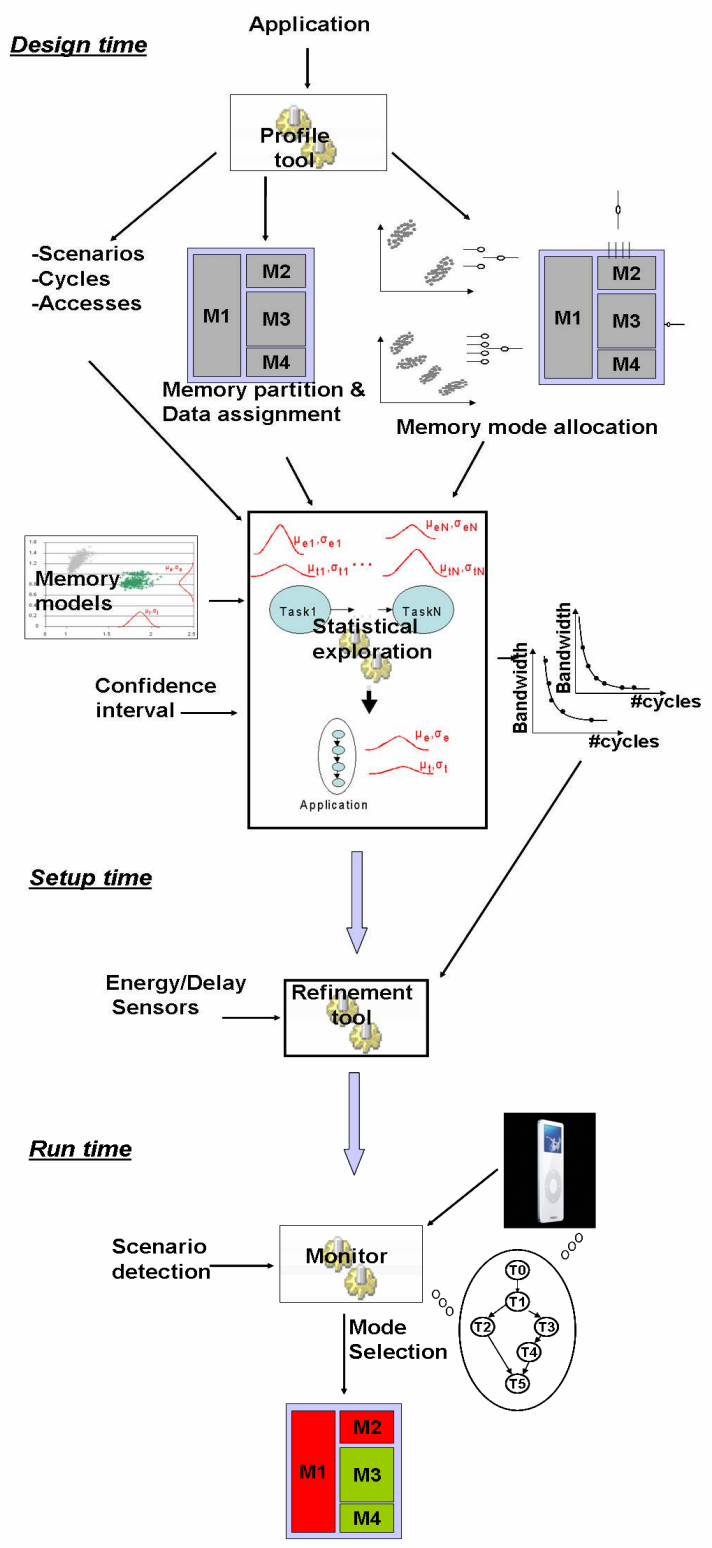


Figure 1.5: Complete system methodology.

configurable memory. Memory modes have to be assigned in such a way that the area requirements are met and the expected energy savings are maximized. This step is explained in depth in Section 6.2.

Our simplified methodology can exhibit a large overhead at setup time. To minimize this overhead, the compensation mechanisms are partially moved at design time. At design time, in the absence of variability, a single optimal memory configuration can be found for each expected timing constraint. Thus, each configuration can be represented as a single point in the energy-time space. However, as variability introduces uncertainty, these single points become clouds in the energy-time space. This means that for a specific configuration affected by process variation, each point in the cloud represents a possible pair of energy-time values. To deal with this uncertainty at design time, the methodology developed in this research includes statistical models to compute different confidence intervals of the overall energy and time associated with each configuration. Based on these intervals, we are able to prune the search space by selecting only those configurations that, with a controlled confidence, meet a given set of deadlines with a minimal energy cost. As a result of this statistical method, we obtain at design time sets of configurations similar to the ones obtained at setup time by means of the basic methodology. We can still represents these sets as Pareto curves, as they keep the same energy-delay balance. However, each point represents only an estimation about the time and energy consumption expected at run time, instead of the final values. This statistical technique, explained in Section 6.3, is the final task to accomplish at design time.

At setup time, the system keeps measuring the memory after fabrication. However, as the configurations (Pareto curves) have already been established at design time, what we have to do at this point is to optimize these configurations according

to the actual impact that process variation has on the current device. This step can either improve the energy consumption or the execution time by performing small changes in the initial system configurations. This optimization process is explained in Section 6.4.

The run-time stage is not changed regarding the basic methodology. We assume in this research that the impact of process variation stays constant during the whole application execution, omitting other sources of dynamic variation such as supply voltage drops, temperature changes or transistor aging degradation.

Including a management process for dynamic sources of variation would require an extension of the current methodology to consider them at run time. Once the methodology would have been adapted, a new step similar to the one mentioned at setup time would be necessary at run time to mitigate its impact on the system.

1.4. System micro-archictecture

In this research, we consider a generic System-on-Chip or System-in-Package architecture to provide support for this methodology. Nowadays, SoC designs integrate multiple heterogeneous processor cores (MPSoC) to fulfil the increasing requirements from embedded applications.

Figure 1.6 outlines a basic MPSoC platform, being the processor cores the components where we focus the attention in this dissertation. The methodology presented here could be included in the design of any of those processor cores, based on an architecture similar to the one shown in Figure 1.6.

The architecture gives an overview of current state-of-the-art designs based on the combination of IP blocks. The system includes among its blocks the process-

ing core, the memory organization and the communication network. The memory system consists of a set of SRAM modules of different sizes organized in a single layer hierarchy. Every single memory in the system is configurable. Monitors are also included in our architecture for testing purposes at setup time.

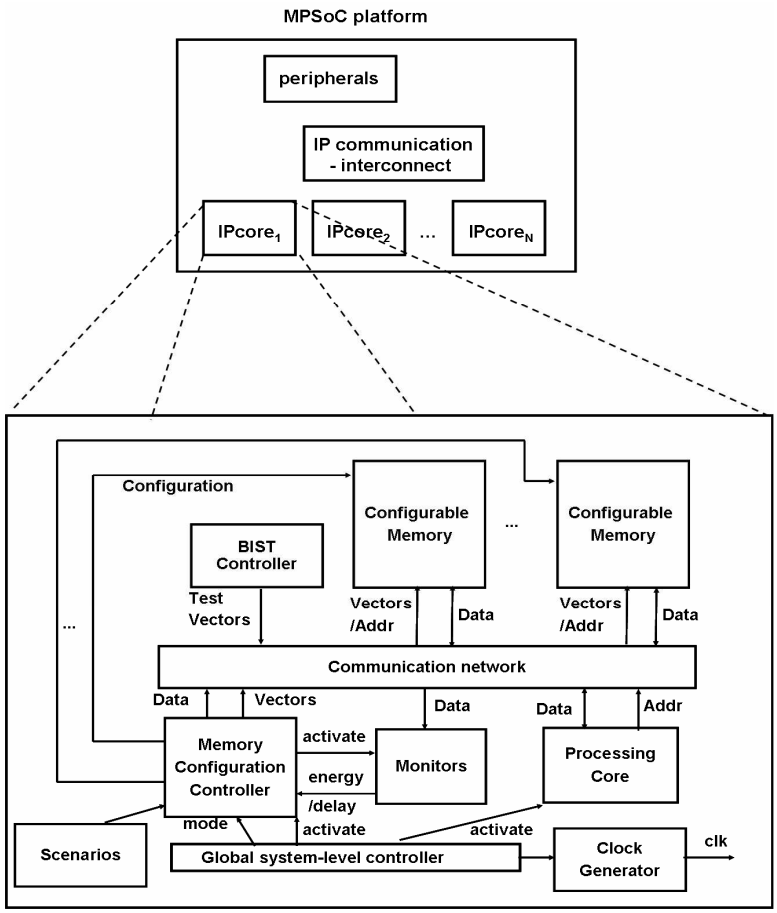


Figure 1.6: Structure of the system architecture for a processor core in a heterogeneous MPSoC.

On top of these blocks we add the block labeled as memory configuration controller, which leads the processes involved at setup time. This component, together with the BIST (Built in Self-Test) controller, sends test vectors to each individual

memory while monitors measure their energy consumption and access time. With this information the controller figures out the actual performance characteristics of the platform. The BIST controller generates the address and data patterns that are loaded into the memory system. Once the current variability impact is known, the memory configuration controller implements the techniques to manipulate the set of configurations mentioned in previous sections. This module, which also has access to the information about the application stored in the form of *system scenarios*, decides on the memory mode and clock configurations for each *system scenario*. At run time, this component is involved in the calibration process. The controller manages the necessary information to chose the optimal configuration and the *system scenario* to be applied at each moment.

For the current implementation we have opted for a hardware controller implemented as a separate block on-chip – labeled as IP component controller in the figure –, since our design is application specific. The controller functionality could also be easily integrated in the operating system of a more programmable device.

We do not consider general purpose computers, nor ad-hoc systems for specific applications. In order to get more efficient solutions, our approach is focused on specific application domains whose characteristics are well-defined and can be better exploited.

1.5. Conclusions

The continuous reduction in technology features has meant an impressive development for the market, specially regarding System-on-Chip or System-in-Package. Nowadays, these systems are widely present in any everyday device and include

a vast variety of characteristics. The technology improvements have increased the complexity of the applications that these systems, mainly portable, perform. Designers have to deal not only with the new challenges related to the scaling, but also with those related to the increasing complexity that applications exhibit.

Regarding technology shrinking, process variability is one of the challenges to cope with, becoming more and more relevant especially on memories due to their minimum feature sizes. Variability affects the memory access time and energy consumption, degrading their values from the nominal specifications used at design time.

On the other hand, as a consequence of the higher number of transistors available, applications have increased their complexity on user demands. This is especially evident regarding media-applications, in which complexity has translated into higher dynamism.

Traditionally, both situations have been tackled by means of conservative methodologies whenever guarantees on real-time behaviour or hard constraints are to be met. However, these approaches are turning inadequate to deal with them due to the high cost in energy and performance involved.

In this research, we opt to avoid conservative methods by means of exploiting *system scenarios* and configurable memories. Focusing on multimedia applications as specific domain, we have developed a complete methodology to deal with both problems in a coordinated way. The methodology presented includes the whole design process, which performs the heaviest and time consuming tasks, but also the setup and run time steps to get a better adaptation of the system to the existing viability and dynamism.

2

Implications of process variation

2.1. A bit of history

The semiconductor industry developed the first commercial integrated circuit chips in 1961. Just a few years later, Gordon Moore published what later on would be largely known as Moore's Law. The future *law* started with the observation that the circuit complexity had increased at a rate of roughly a factor of two per year in a four-year time. Moore pointed out that trend in his renowned paper [Moo65] in 1965, together with the speculation about what industry could expect in terms of complexity in the five to ten-year future. Since that year, the original prediction has been revised twice trying to be more accurate. The first revision happened in 1975, when Moore himself increased in a year the initial one-year period needed to double the complexity [Moo75]. Later, industry would refine it again, establishing an eighteen-month period as a compromise. Nowadays, this last interpretation has become the most quoted version of the law.

Despite the refinement process in which the law has been involved along time to better fit the reality, and the continuous voices predicting the end of the law, Moore's words are still valid for semiconductor industry. The law is considered as the rate that all the industry needs to follow in order not to fall behind technologically.

At the time when Gordon Moore published his observation about component integration, the industry was entirely dominated by the bipolar technology and the TTL logic chips, the only ones which could provide high performance to the large mainframe computers of the day. In those years, integrated circuits based on CMOS technology were starting to show its potential in terms of low power and integration capabilities. However, its low performance, compared to what bipolar technology could get (Figure 2.1) , made the new technology not competitive enough for a large part of the market. Only memories – SRAM, DRAM – were able to be manufactured in large scale, being the first devices where the bipolar technology was successfully replaced. Despite this beginning, the potential of CMOS technology and the high expectations about increasing integration were enough to think of CMOS as the future technology that would replace someday the bipolar transistor in high-performance logic and memory applications too.

The integration capability was an important characteristic due to three important factors:

- Better cost/performance
- Reduction in area per component
- Better reliability

In fact, Moore's predictions in [Moo65] were related to these three aspects. In his article, Moore depicted a future with small systems, such as home comput-

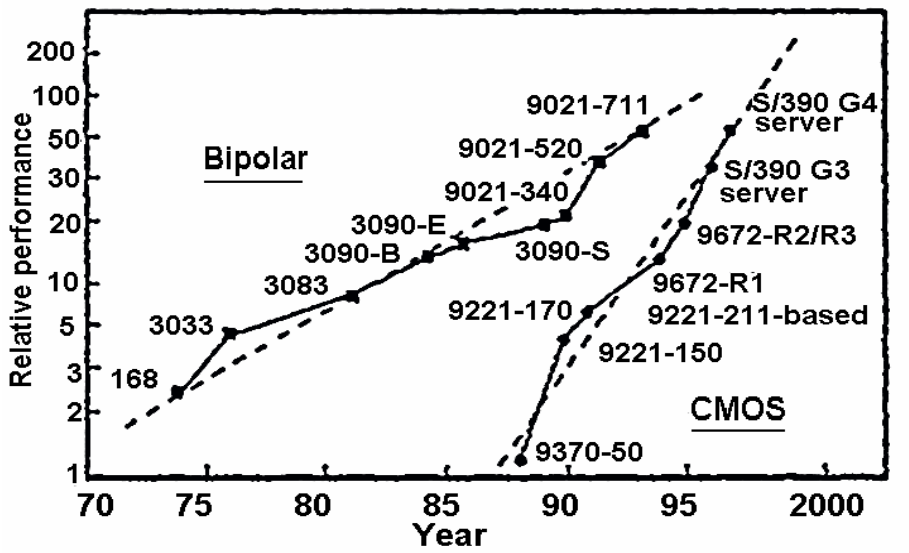


Figure 2.1: Evolution of IBM Mainframes (Bipolar vs CMOS performance) [RGP+97].

ers or portable communication equipments, but also with large systems related to telephone communications and data processing; a future where integrated circuits would be everywhere as integration would make possible to built systems cutting costs in manufacture and design as well as having faster turn-around. Regarding reliability, in the 1960s this issue was already a concern in scopes such as military systems, where integrated electronics had shown their high reliability. In fact, in the military domain it was considered that some demanding systems could only achieve hard requirements in reliability, size and weight with integration.

What Moore observed in the 1960s about component integration was formalized in the following decade by Robert Dennard in his MOSFET scaling theory. Based on a well-defined set of rules, the work published in 1974 [DGH+74] provided a way to rapidly predict how to design MOS devices from one generation to the next. This was the first attempt to establish a methodical approach which coupled geom-

etry shrink with other important factors such as power-delay products, interconnect performance or integration density.

The scaling theory was based on simple rules using Geometrical Scaling, i.e. just reducing physical parameters such as oxide thickness (T_{ox}), channel length (L) and channel wide (W) by a unitless constant factor k was enough to increase transistor density and provide a delay improvement at constant power density. In this way, each new process generation was expected to reduce the feature size by approximately 0.7x, a 30% reduction which was enough to double the transistor density and continue Moore's Law for decades. The device dimensions – L, W, T_{ox} – were not the only characteristics that would be reduced. The scaling rules also affected other aspects related to circuit performance, as Table 2.1 summarizes. What Dennard pointed out was that if voltages were scaled along with physical dimensions it would be possible for each transistor to be cheaper and faster while consuming less energy. At circuit level this would mean the chance to have more gates switching faster but keeping constant the power per unit area, as each gate would consume less than before.

Device or Circuit Parameter	Scaling Factor
Gate length L	$1/k$
Gate insulator T_{ox}	$1/k$
Channel width$W$	$1/k$
Doping Concentration	k
Voltage V	$1/k$
Current	$1/k$
Capacitance	$1/k$
Delay	$1/k$
Power dissipation	$1/k^2$
Power density	1

Table 2.1: Dennard' scaling theory [DGH$^+$74].

The feasibility of a continuous scaling and its benefits on transistor density, performance, power consumption and dissipation where too high to not taking CMOS technology into account. However, for about two decades after the scaling theory was published, CMOS remained as a low-cost technology limited to applications where performance was not important. In that time period, the 1970s, the standard power supply voltage for practically all integrated circuits was set at 5V. As there was little or no market for circuits using non-standard voltages, engineers focused their effort on scaling only the physical size of the CMOS transistors. The supply voltage was kept constant to make the circuits compatible with the dominant bipolar technology. With this purpose, engineers developed all kind of techniques to keep on scaling even at constant voltage. Thus, voltage tolerant device structures such as lightly dope drain (LDD) transistor, silicide clad source drain or hot electron defense emerged to cope with the problem. Many other technical advantages allowed the scaling along the time, and not necessarily related to keep the voltage quasi-constant. Improvements in optical lithography, dry etching, ion implantation, insulators, copper wiring, packaging, testing or design techniques were also necessary to keep scaling.

During the 1970s and 1980s, semiconductor industry was able to introduce a new technology generation every 3 years. However, the constant voltage limited the potential performance in CMOS circuits, making impossible to match the performance provided by bipolar technology. However, the 1980s was also the decade when power dissipation in bipolar technologies appeared as an increasing problem; even more critical than design complexity or yield, which could be coped with the advent of Computer-Aided Design and manufacture improvements respectively. Figure 2.2 shows the trend that bipolar and CMOS technologies followed along

those years, being clear the large difference between them regarding power. At late 1980s, when the power dissipation in bipolar devices was becoming unsustainable, the 5V standard was discarded by the industry as it was the main factor of the power crisis. Also the need for more performance in high-speed applications and the increasing volume of portable devices which required the highest performance at the lowest power made necessary to look for alternatives.

The establishment of voltage standards under 5V benefited CMOS technology, which improved enough to replace bipolar technology in a very short time. Thus, scaling voltage was the key factor to get quickly performance improvements, as well as a less complex fabrication process and hence a cost reduction. Its scaling allowed to maintain constant electric field, which was important for reliability, and lower the transistor power needed to maintain constant power density. These improvements also reduced the time between technology generations. Since the mid-1990s a new generation has appeared every two years, instead of the three year gap which characterized the previous period.

The impressive scaling rate during the 1990s benefited extraordinarily the industry development. The microelectronic dimensions were left behind – being the 130nm generation the last one of that kind –, to enter into the nanoelectronics era (90nm and below) in the early 2000s. For years, the rules set by Dennard [DGH$^+$74] to reduce the transistor size were enough to improve by itself performance, energy consumption and transistor density. However, the nanoelectronics characteristics made the classic scaling theory inadequate to continue the improvement experienced in previous years. Scaling under 100nm made emerge old problems while new challenges came out.

There is a large variety of factors that make difficult the classic scaling in submicron dimensions [Soc06, Soc07], although the most important challenges are power and variability.

Power

In the 2000s, power appeared again on the stage as it had done a decade before, when CMOS replaced bipolar technology due to its lower consumption (Figure 2.2). CMOS technology had been able to postpone the power crisis a decade, until its growing was too high to be tolerable.

In CMOS technology, power consumption is mainly determined by two components, according to Equation 2.1, which is generally accepted to model power[1].

$$P = \alpha C V^2 f + I_{off} V \tag{2.1}$$

The first term of the equation measures the dynamic power consumption, while the second corresponds to the power lost from the leakage current regardless of the state of the gate. A third term related to the short-circuit current – which momentarily flows between the supply voltage and ground when the output of a CMOS logic gate switches – is relatively small and contributes to the dynamic power, so it can be absorbed by it, not appearing in the equation.

For a long time, the dominant component of power dissipation was the dynamic power, resulting from continuous transistor switching and charging and discharging of capacitances. According to Table 2.1 and Equation 2.1, the dynamic power could

[1] In Equation 2.1, α is the switching activity of the gates in the system, C is the total capacitance load of the gates, V is the supply voltage, f is the clock frequency and I_{off} is the leakage current.

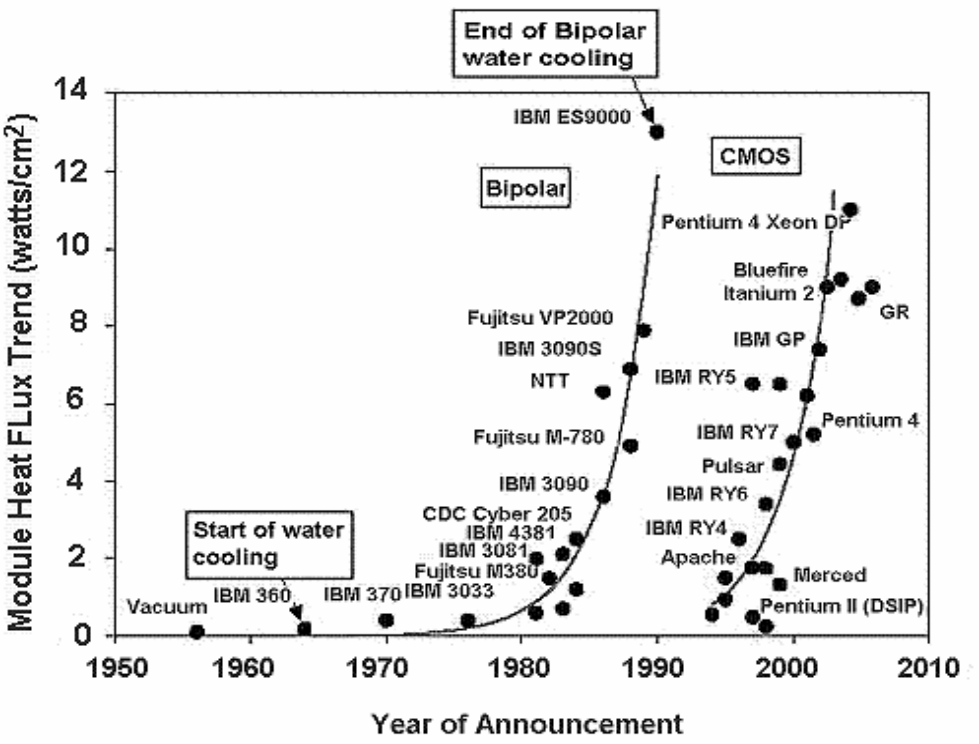


Figure 2.2: Power in CMOS technology shows the same ascending trend observed in Bipolar technology in the past [Isa01].

be reduced by four, scaling voltage a half in each generation. In turn, in order to achieve the expected performance target through scaling, also the threshold voltage had to be reduced along with the supply voltage. However, reductions in threshold voltage are related to exponential increments in leakage current[2], as Relation 2.2 models.

$$I_{leak} \propto exp(-qV_{threshold}^2/kT) \tag{2.2}$$

[2]Leakage current flows between the source and drain terminal when transistors are not in the active mode. Even if the gate voltage of the transistor is lower than the threshold voltage required to turn it on, the transistor is not completely off, causing the subthreshold leakage current.

The scaling rules set in the 1970s assumed the device leakage associated with the static power component as negligible. Nevertheless, three decades of continuous reduction in threshold voltage and the submicron dimensions used since the 2000s in the industry, made the leakage current a significant element to take into account in the total power consumption. In fact, leakage associated with low threshold voltages increased from levels lower than 10^{-10} amp/mm to values higher than 10^{-7} amp/μm. Industry expects leakage to exceed the total dynamic power consumption to be the most significant component in power[NAB+03]. This is due to both active and power-down leakage modes. Although leakage is rapidly increasing due to variability and temperature, currently dynamic power is still higher than leakage in logic and memory [SIAR10].

Leakage increment is not only due to threshold voltage reductions, other factors [EB05] that contribute to the growing static power consumption are:

- Gate insulator thickness (T_{ox}). It is also a key factor in scaling as its reduction allows voltage scaling. Along the different generations, gate oxide – a dielectric layer designed to prevent leakage current between the electrode and the source and drain – has been negligible comparing with subthreshold leakage. However, as thickness has lowered from 2nm to a thickness of only a few atomic layers, quantum-mechanical tunneling has given rise to an increment in gate leakage currents which cannot be neglected anymore, doubling the subthreshold current.

- Short channel effects. Scaling reduces the channel length L to increase both the operation speed and the number of components per chip. However, as this feature decreases the short channel effects arise. The reduction in gate insu-

lator thickness helps in controlling them, so its difficulty to keep on scaling at the current rate will increase faster the short channel effects.

- Channel doping concentration. The higher the doping concentration is, the lower is the resistivity at transistor level and thus, the higher is the conductivity. In Dennard's scaling theory, this parameter was believed to be continually increased to enable shorter channel lengths. However, when doping concentration is too high, performance is degraded and junction leakage increases as a consequence of tunneling effects.

The consequence of the static power raise was the restraint on threshold voltage scaling and thus, on voltage itself from the 130nm technology forward. To keep on scaling with guarantees, new materials and structures such as strained silicon, high-k metal-gate or tri-gate transistors have came up in the last years to deal with power constraints. This is the case of the gate oxide problem, which scaling was initially slowed down to reduce leakage, affecting the scaling of voltage. Currently, a less aggressive gate dielectric thickness reduction can be obtained by replacing the silicon dioxide for materials such as the high-k dielectric, which is able to maintain the required gate overdrive at low supply voltages.

During the last years, all dimensions and doping densities have scaled and the number of gates has also been increasing by a factor three per generation. However, voltage has decreased at a slower rate. This mismatch has affected the energy dissipation, which decreases at a rate lower than expected, and makes power density increase with each generation [BC11]. The high power consumption and device density raise the temperature in circuits and cause more leakage current as Relation

2.2 points out. Furthermore, high temperatures directly affects circuit operation, compromising performance, reliability and lifetime.

Process variability

Transistor behaviour is characterized by physical parameters such as length, width or oxide thickness, as well as by other factors such as supply or threshold voltage. In the past, the manufacturing process was able to produce chips where these parameters behaved as it was expected at design time, or at least, the slight variations could be managed efficiently.

However, the continued technology shrinking has led to levels of variation that cannot be managed as it was used. This phenomenon, known as process variation, increases the difficulty in controlling the uniformity of device parameters and turns the transistor characteristics more uncertain. As a result of this uncertainty, significant aspects on a circuit such as delay or energy can be largely affected, degrading its reliability and performance.

2.2. Introduction to process variation

Variability increases as technology features become smaller, so finding two transistors which share exactly the same characteristics is extremely rare nowadays. This variability can generate failures on the chip, failures which can be transitory due to temporal variations on voltage or temperature, or permanent due to irreversible physical alterations. In this section we briefly summarize the types of variation existing as well as the failures which produces, but firstly we provide the factors that are considered as the main sources of variability on a chip.

2.2.1. Sources of variation

There are three factors which are widely considered as sources of variability [Bor05]: random dopant fluctuations, lithography and power density. Two of these sources are static and happen during the manufacturing process, while the power density – as we point out below – is highly timing and environmental dependant.

- **Random dopant fluctuations**

  The electrical conductivity on a semiconductor material can be altered by adding impurities. On silicon, the impurities are added in the channel of a transistor to increase its conductivity. The reduction in the transistor size, and thus in the channel, reduces also the number of dopant atoms that can be added in each technology generation, as well as their uniformity. This trend can be seen in Figure 2.3. Thousands of atoms were present in the channels of the 1-micron technology. However, this number is reduced to only few tens, non-uniformly distributed, in 32nm to 16nm generation.

  This reduced number of dopants has a greater impact on the transistor electrical properties, altering its behaviour. In fact, a small deviation in the number of atoms or in their position represents a huge change in the device properties. The increasing variability at transistor level makes difficult to characterize parameters such as the threshold voltage or the channel resistivity. At the end of the day, this variation affects the circuits, which are more difficult to control and foresee.

  As stated above, dopants are added to enhance the transistor conductivity, which means that the amount of energy necessary to turn a transistor on is lower. The continuous reduction in the channel length and the non-uniformity

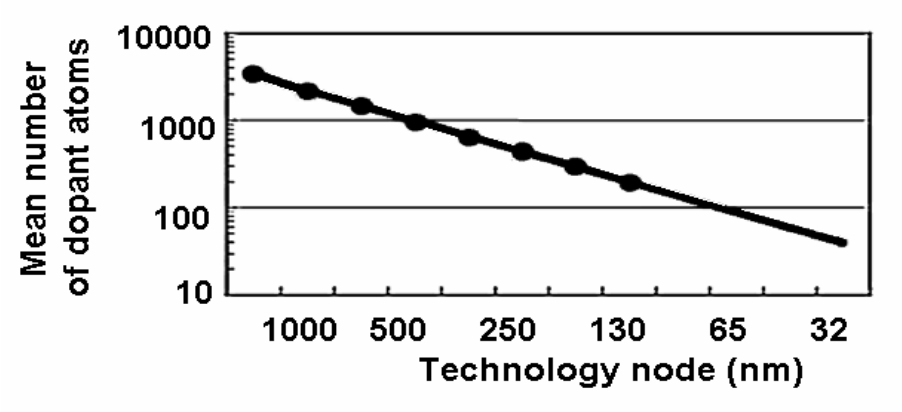


Figure 2.3: The number of dopant atoms decreases as technology shrinks [Bor05].

placement of dopants translate into the variation of this parameter, the threshold voltage. Figure 2.4 illustrates this situation. As technology shrinks down, the same reduction in the channel length does not involve an equivalent reduction in the threshold voltage. The difference is now larger.

This higher variation in the threshold voltage has significant consequences for all the devices on a chip, although in this research we have focused the attention on memories.

- **Lithography**

  Lithography is a concern too. The optical lithography has reduced the wavelength used for patterning transistors for four decades. However, its decrement has been much slower than the minimal geometry size, giving rise to the increasing gap along the years that Figure 2.5 shows.

  In the late 1990s, 248nm wavelength was used to pattern from 250nm to 180nm transistors, to later on decrease to 193nm for 130nm technology. Since then, industry has entered in the subwavelength lithography era. The feature

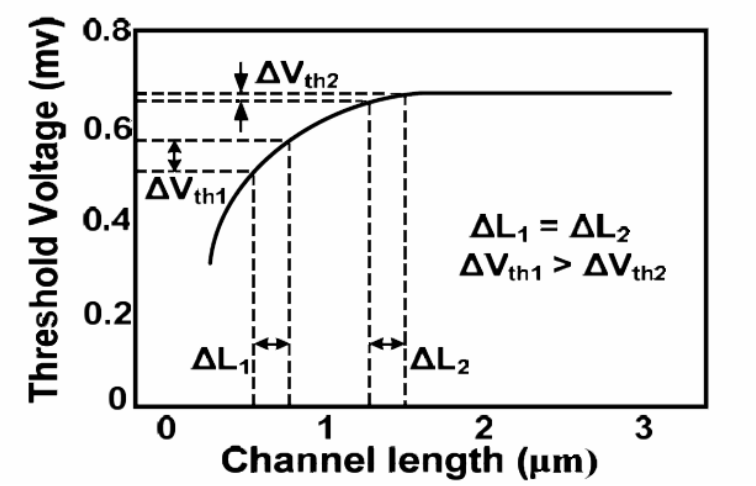


Figure 2.4: Variation in threshold voltage increases in each new process generation.

size has got smaller while resolution at 193nm light lithography has stayed the same. This gap between process generation and lithography affects yield and has introduced significant variations in feature characteristics such as line width or thickness, which finally affects the yield and performance of integrated circuits. As an example of its impact it has been seen that line width roughness results in variations of transistor gate length, which increases leakage of transistors and modifies its speed. The gap between the wavelength and the patterning width, and thus the struggle against variations, will continue until extreme ultra-violet technology (13.5nm) [RBB+06, IME] becomes available, which it will still take some time as new materials and tools are required [SGK11a]. Meanwhile, since many of the variations caused by subwavelength lithography processes are systematic, compensation mechanisms such as optical proximity correction, phase shifts masks or immersion lithography capture them, being able to extend 193nm lithography at least at

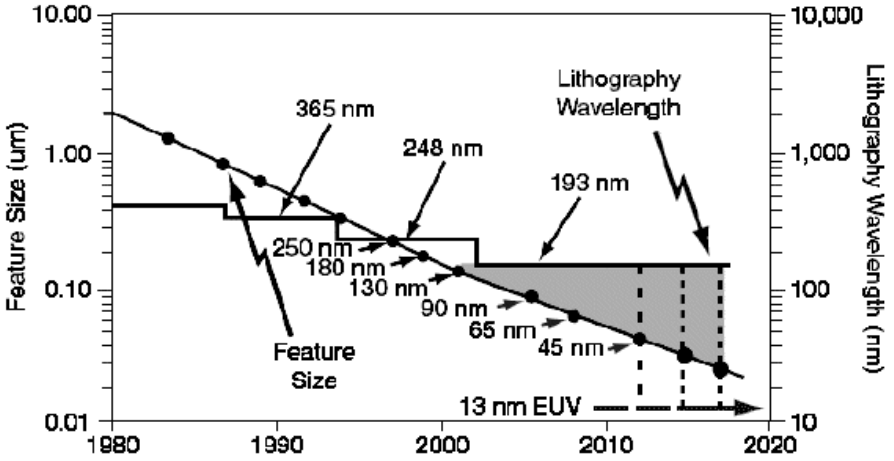


Figure 2.5: The scaling down of the technology is faster than lithography does [Bor04].

the 32nm generation. For technology nodes below 32nm [ITR], other techniques such as dual patterning will do the same.

- **Power density**

  Power density is the only dynamic source, giving rise to spatial and time dependant variations. Firstly, as can be seen in Figure 2.6, not all the components on a chip shows the same power density (also known as heat flux). This is related to the functionality and area of each component. Thus, the power density on a cache is usually lower than the one generated by a functional unit. Regarding time, the power density also depends on the activity and workload existing in a component in a particular moment.

  Variations due to power density affects mainly supply voltage and temperature, generating hot spots which affect the system performance and reliability. The increasing temperature, according to Relation 2.2 involves an increment

in the leakage current, which means that the static power rises. The most immediate consequence of leakage going up is that the power density continues increasing, entering in a vicious circle which leads to significant drops in performance as well as decreasing lifetime reliability.

Among the new approaches to deal with density power at transistor level we can find the use of strained silicon, high-k materials or tri-gate transistors.

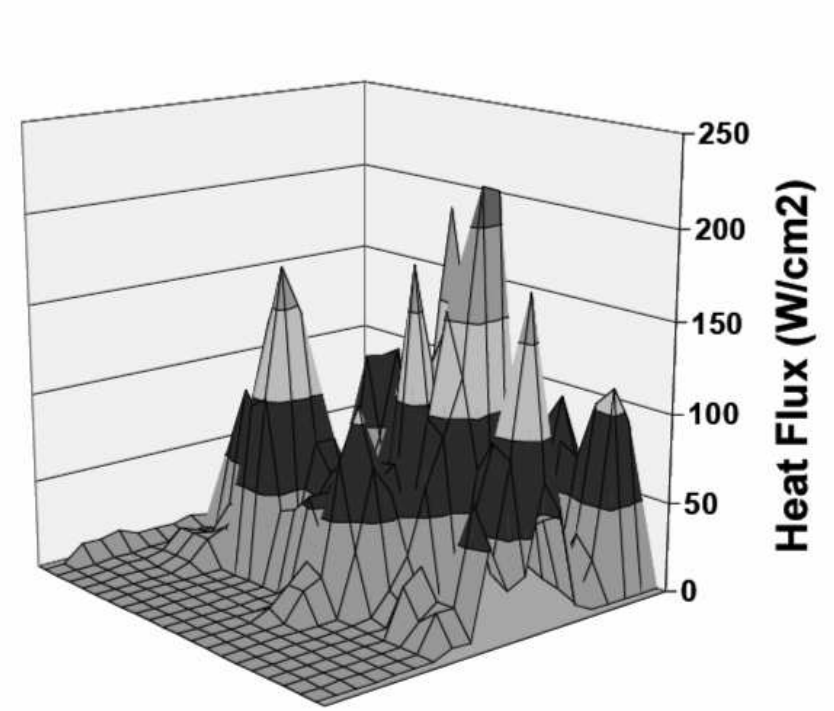


Figure 2.6: Distribution of heat flux on a chip [Bor05].

2.2.2. Taxonomy of process variation

The physical and environmental variations existing in integrated circuits can be classified by a wide variety of criteria [ZHHO04b, CQS04] based on temporal or

spatial considerations, the scope or the random nature of variations. In this section, we outline a classification based on the physical variations. These variations, caused by imperfections in the masking process and manufacturing, translate into changes in the electrical behaviour of the passive (wires) and active components (transistors). Transistor channel length, oxide thickness, threshold voltage or parameters related to the interconnection among components are just some of the technology features affected by physical variations.

Based on the scope, process variations can be classified as *inter-die* or *intra-die* variations. *Inter-die* variations are those which alter, across chips placed in different dice, the mean value expected for technology parameters; however, there is still a correlation across a given die, where all the devices experience the same deviation in a particular parameter. As Figure 2.7 shows, this category includes variations among different lots (*lot-a-lot* variation), different wafers (*within-lot* variation) and variations within the same wafer (within-wafer variation). On the other hand, *intra-die* variations correspond to changes within the same chip.

As Figure 2.7 illustrates, these two types of variations can be classified again, this time based on the randomness exhibited. According to this randomness criteria, variations can be classified as random and uncorrelated, or systematic and correlated. From the point of view of randomness, a systematic variation is characterized by affecting every component in the same level in the same way. However, random variations do not show so clear trend along the circuit, turning the characteristics more unpredictable even for neighboring transistors. A large amount of factors during the fabrication process, such as the dopant fluctuations or the lithography, can cause random and systematic variations. Variations usually exhibit spatial correlations. For instance, in case of systematic intra-die variation, transistors placed

nearby result in similar variations in their characteristics, but very different from the ones placed farther. In case of systematic inter-die variations, characteristics from transistors placed in different dice share a same trend. This trend can show smaller values than the expected ones in all the transistors, or the opposite, exposing values higher than the nominal ones. An example of inter-die random variations is caused by the non-uniformity existing in dopant atoms, which affect the threshold voltage among chips.

Besides physical factors, variations also depend on environmental factors which affect high level aspects of design such as voltage, power or temperature. These variations are usually analyzed at the intra-die level and can be spatial or time-dependent. Aspects such as block placement, power grid design, voltage drops or high temperatures affect circuit parameters such as threshold voltage.

Based on the classification presented above, Table 2.2 summaries some of the main variations existing in relevant parameters at 90nm technology. As can be seen, variability effects can be classified in different categories even for the same parameter. This is the case for threshold voltage, which exhibits systematic and random variability at any level, inter-die and intra-die.

From a temporal point of view, variations can be time-dependent or independent, based on whether reliability and performance are affected or not along the system lifetime. The performance of a circuit can deteriorate over time due to the degradation of individual deep-submicron transistors. This phenomenon, known as aging, causes slower speeds, irregular timing characteristics, higher timing consumption and even functional failures over the time. This degradations is caused by physical mechanisms such as hot carrier injection, negative-bias-temperature instability (NBTI), electromigration or time-dependent-dielectric breakdown. NBTI is

Parameter	Variation
Channel length	Sistematic inter-die, Systematic intra-die, Random intra-die
Threshold voltage	Systematic inter-die, Random intra-die
Voltage and Temperature	Systematic intra-die
Average threshold voltage among different devices	Systematic inter-die
Power density	Systematic intra-die Random intra-die

Table 2.2: Parameters can exhibit more than one type of variability [ZHHO04b].

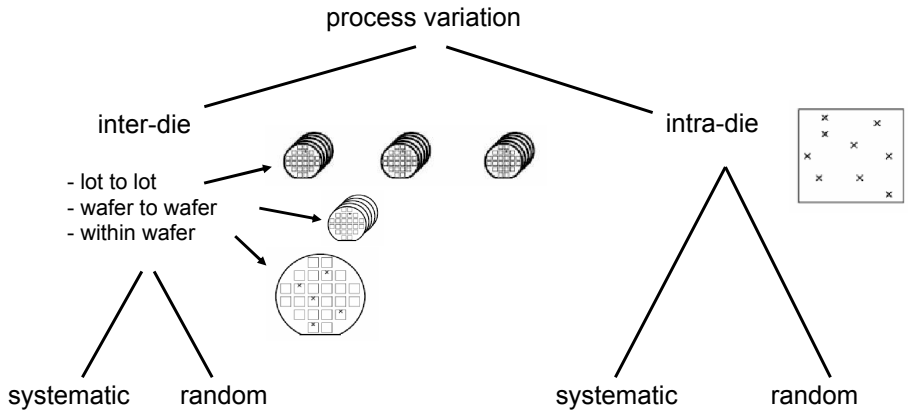


Figure 2.7: Process variation: general classification.

the main responsible for the circuit aging and has a strong dependence of dynamic conditions such as temperature, supply voltage and the input signal. NBTI may result in up to 50mV shifts in threshold voltage through the lifetime, degrading more than 20% the circuit speed [WYB+10].

In this research we focus on intra-die variations, specially on random variations, as we explain in the following section.

2.3. Process variations on memories

Variability affects every single component in a system, turning them less reliable and reducing the functional yield, i.e. the number of components after fabrication which generate the right output for all the possible inputs decreases as variability increases. Although this kind of yield is very important, in this research we focus our attention on another type of yield which is also significant and related to performance, the parametric yield. The parametric yield implies the existence of timing constraints, indicating whether a component or a circuit can generate the right outputs within the required timing. This yield is an important parameter to take into account on sequential circuits or systems designed to meet specific timing considerations from the application running on it. At system level, a component for which the parametric yield is at least as important as the functional yield is the memory system. In the rest of this section we focus our attention on the relevance of the memories on the system and how variability affects them.

Considering a system where variability can be considered as completely negligible, we find that the relevance of the memory system is double. Firstly, memories are specially designed with minimum-sized transistors to increase their performance. Secondly, the area devoted to on-chip memories has largely increased with each new technology node, representing more than 50% of the whole area [Bor07, SIAR10]. As an extreme example of the relevance that the memory has in current designs, we can consider the case of the “Montecito" Itanium processor [Ita], designed by Intel using 90-nm technology. As Figure 2.8 shows, the processor includes in each one of its two cores a three-level memory hierarchy, consisting of 16 Kb data and instruction caches at the first level, a 1 Mb instruction cache and

a 256 Kb data cache at the second level, and finally a 12 Mb unified cache at the third level. In summary, more than 95% of the 1.72 billion transistors existing in the processor are devoted to memories [MB05], as Figure 2.9 shows.

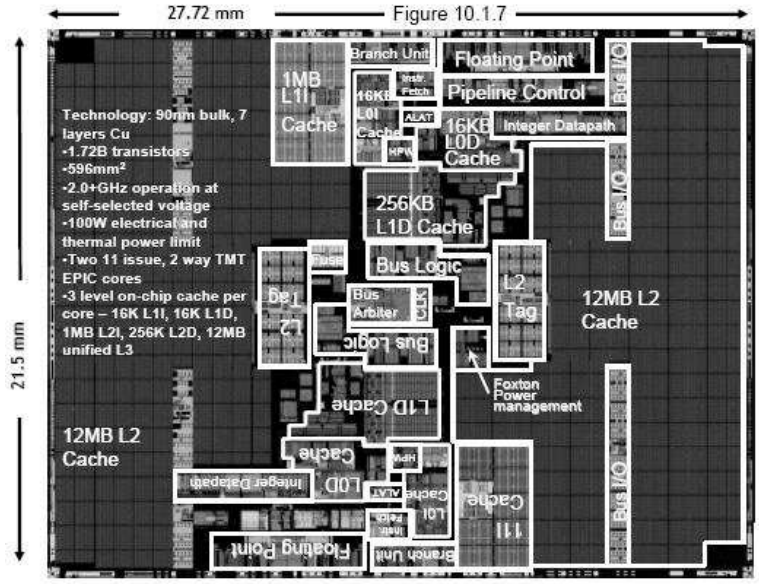


Figure 2.8: “Montecito" Intel Itanium 2 processor. Die overview [MB05].

The extreme scaling and the increasing area density characterize memories in current high-performance SoCs and rise the percentage of the system energy budget referred to them, turning memories into one of the most energy consuming components in the whole system as Figure 2.10 shows [VVWW01].

At system level, it is clear the relevance of the memory system, not only in terms of time but also of energy consumption. Memories behave as any other component under variation, especially in terms of energy and timing. As in any other component, parameters sensitive to variations alter its especifications from the nominal values established at design time to higher or lower values. However, because of the characteristics mentioned before – area density and minimum-sized transistors

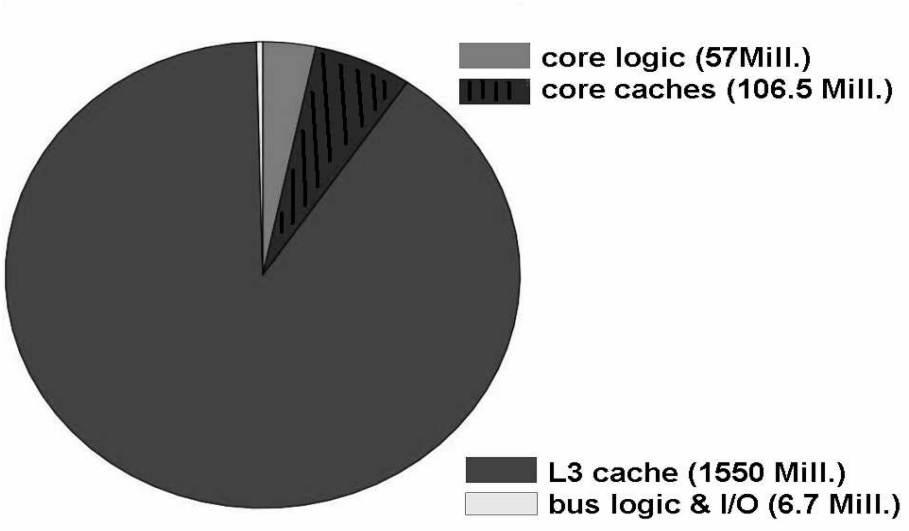


Figure 2.9: Transistor breakdown in “Montecito" Itanium 2 [MB05].

– their effects on memories are significant, becoming one of the most variability sensitive components in a system. This situation endanges not only the parametric yield of memory itself but also the parametric yield of the whole system.

As it was mentioned in Section 2.2, among the variability sources with larger impact on memories, the increasing mismatch on critical components such as the doping concentration is one of the most prominent sources. This mismatch turns electrical properties much less predictable than expected. As an example, we can mention the threshold voltage as one of those electrical properties affected by the lack of uniformity in the dopants. Both parameters are related through the equation below [BESB94], which denotes the mismatch in V_{th} due to doping fluctuations:

$$\delta V_{th} = \frac{q}{2C_{ox}} \sqrt{\frac{\pi N_A}{2}} (W_{eff} L_{eff})^{-3/8} (X_D)^{1/4} \tag{2.3}$$

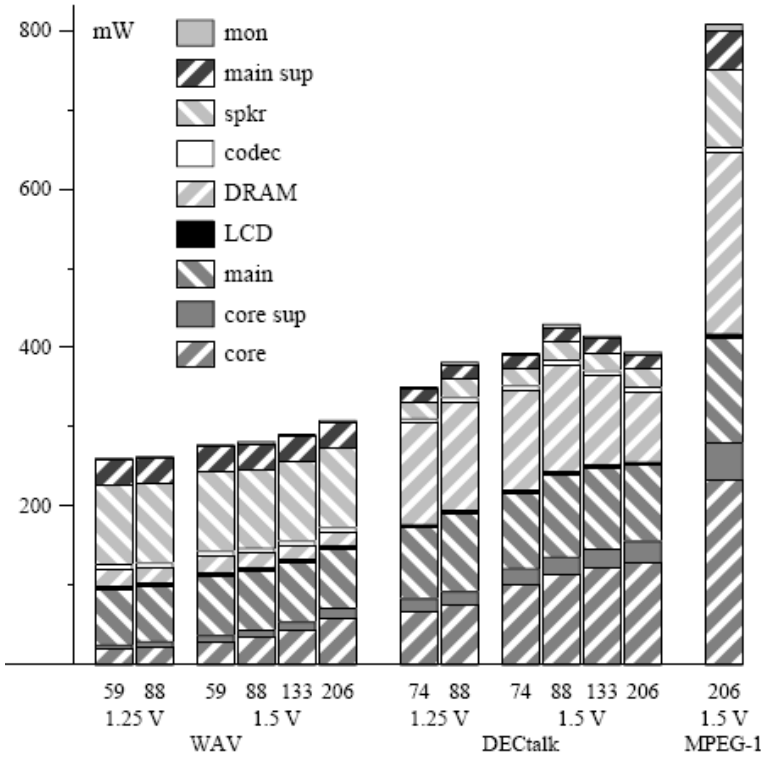


Figure 2.10: Power consumption is expected to be dominated in the future by the memory hierarchies instead of the data-path [VVWW01].

where X_D is the depletion depth, N_A is the doping density, L_{eff} and W_{eff} are the effective channel length and wide respectively and C_{ox} is the gate oxide capacitance per unit area. Amongst all the parameters involved in Equation 2.3, the doping concentration is the most significant regarding V_{th} variations. Such variations occur along the whole chip, giving rise to variations in drive current and propagation delay in any gate of the die. According to [EBSLM97], the gate delay can be modeled as:

$$d_{gate} \propto \frac{C_{load} V_{DD}}{\mu C_{ox} (W_{eff}/L_{eff})(V_{DD} - V_{th})^{\alpha}} \tag{2.4}$$

where μ is the mobility and α is a value between one and two which depends on the short channel effect. This equation shows a clear connection between the existing V_{th} value and the final delay. This link exists at any level, from gates at combinational circuits to memory cells. Together, both equations show how variability, through fluctuations on critical process parameters, increases the circuit delay. Being specially remarkable its impact on memories due to their specific design and small dimensions as it is shown next.

Every memory consists of a set of sub-components, such as cell arrays, decoders and drivers, assembled as Figure 2.11 shows. Each one of those sub-components, not only the cell arrays, has its own particular influence on the delay of the whole structure. In [HCM+05], authors model a typical memory and study the variability impact on it, considering technology parameters for a 65nm technology node. Variability is simulated at transistor level as fluctuations in the nominal values for the threshold voltage (V_{th}) and the current factor (β)[3].

The attention is focused on the decoder and the cell arrays, which constitute the critical path in an SRAM memory. The study [HCM+05], which models a 1KB SRAM, shows that variability has a large impact on the decoder delay because of the large amount of parallel paths existing. The decoder design contributes to make this component much slower than the nominal value expected during the design of the memory. The energy is also affected at the decoder, although so marginally that can be considered negligible comparing with the impact in time. These values for delay and energy are propagated from the decoder to the cell arrays, the next component in the memory critical path. Variability also impacts on the cell arrays, increasing its delay in the same way as on the decoder. In these components, the

[3]The value for β depends on several physical parameters, following the equation $\beta = C_{ox}\mu$W/L.

energy cost is even more significant than at the decoder, being the existing bitlines the ones which dominate the consumption in the cell arrays. This dominance is due to the interaction between their passive and active lines, which is more costly under variation.

Taking into account the energy and delay values from the decoder and the cell arrays, the study shows under variability a shifting from the nominal values to higher rates, which graphically can be seen in Figure 2.12 as a *cloud*. To obtain this figure, as it is mentioned in [HCM+05], each transistor included in the SRAM netlist is injected with its own variation in V_{th} and β parameters. The injection can be completely random and uncorrelated under two assumptions. Firstly, that it is safe to consider the random process variability in transistors as the variability source which currently dominates in current SRAM technologies. Secondly, that within-die process variability is spatially uncorrelated due to their stochastic nature [Cro05]. Thus, transistors in a die have their own variability in electrical parameters and can be considered random. In these conditions, each point in the *cloud* represents the energy and delay of a 1KB SRAM memory under a specific set of variability injections. The main conclusion in [HCM+05] is that process variation on memories increases the delay up to 40% above the nominal value, and up to 50% in the case of the energy consumption. Similar results are reported in [AAA+07]. These percentages rise as memory size increases and technology continues shrinking down, being a challenge to keep the parametric yield.

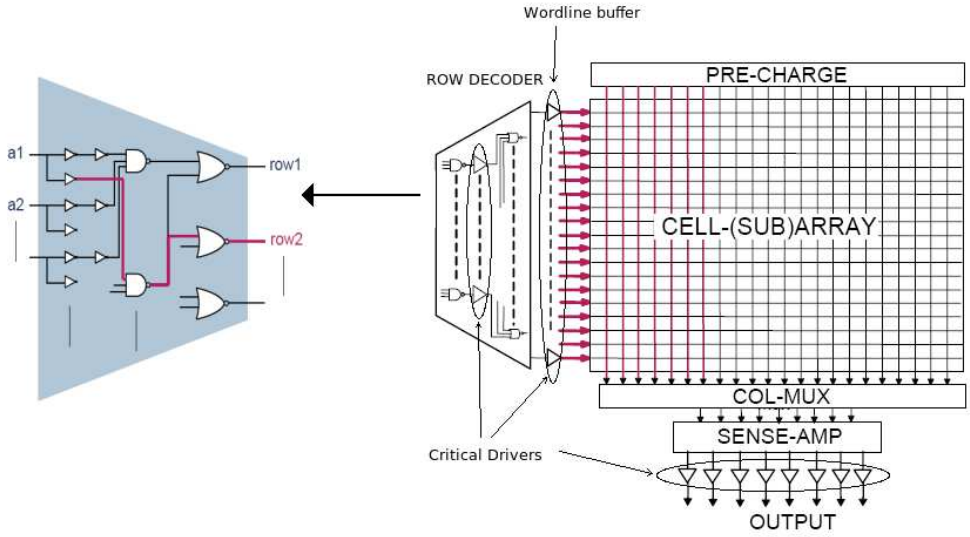


Figure 2.11: Variability has a large impact on memories due to the large amount of parallel paths in the structure.

2.4. Configurable memories

In the previous section, the significant contribution done by the circuitry around cell arrays to the energy and delay on memories has been highlighted. Previously, similar information regarding small SRAMs was already reported in [AH00], pointing out how the decoder and the wordline drivers consume about half of the total energy and time of the entire memory.

Among all the elements included in the circuitry which surround the cell arrays, there are some elements that can be considered as critical circuits in terms of energy and delay. This is due to their placement on the critical path of every memory to drive large capacitive loads in a reasonable time, as well as to their significant contribution to the energy and delay of the whole memory. These elements are the buffers, placed in the internal decoder, the wordline driver and the output driver,

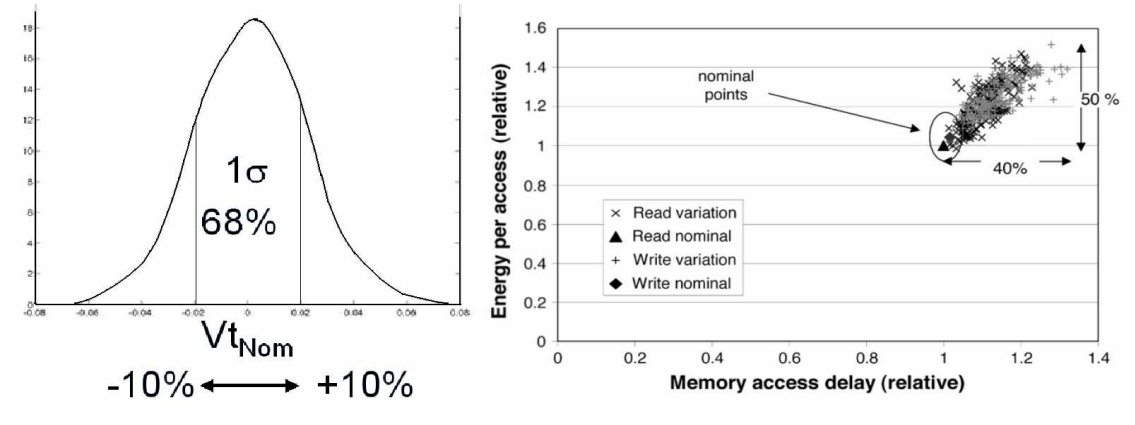


Figure 2.12: One sigma variation on the nominal values for V_{th} and β leads to about 40% variation on delay on memories [WMP$^+$05].

after the sense amplifiers (Figure 2.11). The buffers are designed to achieve a large performance, which is energy costly. However, as any other component in a system, SRAMs do not need always to run at the highest speed. Therefore, and because of their placement, these buffers are considered in [WMP$^+$05] as the ideal components to be turned configurable and work as knobs to provide energy/delay trade-offs.

The general structure of a regular buffer is shown in Figure 2.13. Every buffer is characterized by a number of stages, i.e. number of inverters, and the sizing factor for each stage. As shown in [WMP$^+$05], for a particular load in the circuit there is always a maximum number of stages to get the optimal performance. A higher number would impact negatively in the energy and the delay. For a specific load, the proper design produces what it is called a Pareto buffer, i.e. no other buffer configuration achieves the given delay with less energy cost, and no other can offer a lower energy budget for such delay.

Based on the number of stages and the sizing factors, the energy and delay of an N-stage tapered buffer can be expressed mathematically as follows:

$$Delay = t_{p0} \sum_{i=1}^{N} (1 + \frac{f_{i+1}}{\gamma * f_i}), being\, f_{N+1} = F \tag{2.5}$$

$$Energy = C_{min} V_{dd}^2 (\sum_{i=1}^{N} (1+\gamma) * f_i + F) \tag{2.6}$$

Varying these two parameters, the number of stages N and the sizing factor f_i, the amount of drive current provided by the buffers can be controlled and their energy and delay can be modified.

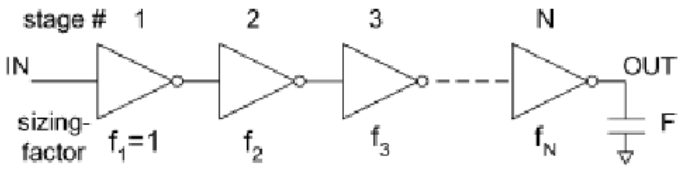


Figure 2.13: General structure of a buffer where the number of stages is variable [WMP+05].

The regular buffers existing in the decoder and wordlines can be substituted for sets of Pareto buffers. Each one of the included buffers provides, to the driver where its placed, a different energy and delay value. Each pair energy-delay is called an option. At transistor level, the general structure shown in Figure 2.13 for a buffer, is implemented as it is shown in Figure 2.14. The circuit represents a configurable buffer composed by two different Pareto buffers which share the input and the output to reduce the final size. The different implementation of each buffer confers them different characteristics in terms of energy and time. Thus, in the figure, one of the buffers is fast but energy costly (labeled as *high speed*), while the other (labeled as *low power*) is slower and consumes less energy. The last stage

in both buffers uses a tri-state inverter. In the high speed buffer, the first stage is also implemented as a tri-state inverter. The remaining stages consists of regular inverters. An existing memory controller is in charge of sending the appropriate signals to manage these buffers. In our example, these signals are *Ctrl* and its complementary *Ctrl#*. The tri-state inverters placed in the last stage of each buffer are used to ensure that buffer outputs are mutually exclusive, so only the selected Pareto buffer is functional at a time. The extra transistor placed at the end of the first stage in the high speed buffer guarantees that the intermediate stages in that buffer always keep a know value ('0' or '1') when this buffer is deactivated, i.e. first stage output is 'Z'.

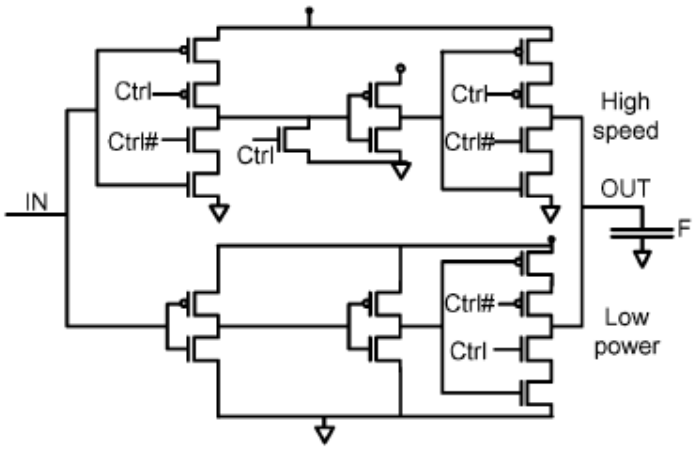


Figure 2.14: Circuit for a two option configurable buffer, where two three-stage Pareto buffers are combined [WMP+05].

From [WMP+05], we reproduce in Figure 2.15 an example of the energy/delay trade-offs that can be achieved with such configurable circuits. In this case, a three-option buffer is validated under a set of variability injections in the V_{th} and β parameters of the transistors. The energy/delay ranges between modes remain

even under the effects of variability, showing the robustness of these buffers. This robust behaviour is due to the larger size of the inverters involved in it. According to [WMP+05], the energy/delay ranges are also largely independent of the technology node used, as they depend mainly on buffer parameters such as the sizing factor. These characteristics turn buffers into excellent tuning knobs to provide energy and delay partitions. The number of available options has to be kept low, as a high number makes it difficult to use and control in an optimal way because of the small sub-ranges among options that would have to be managed. A higher number would also imply a larger penalty in area.

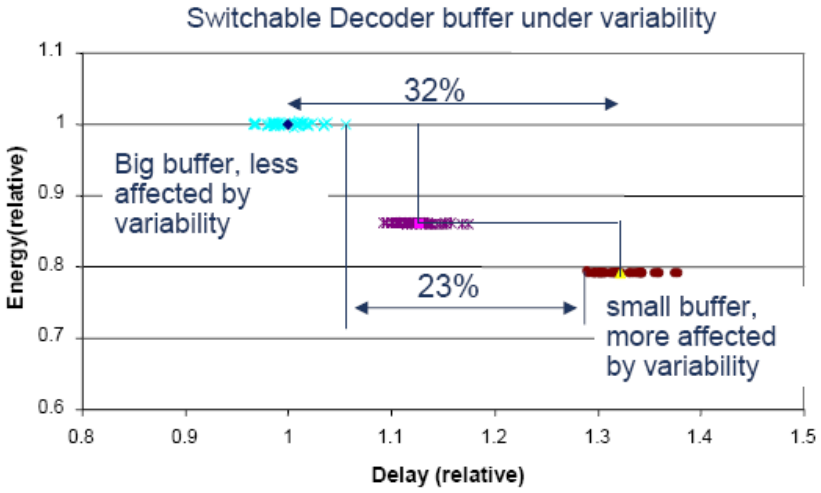


Figure 2.15: Energy and delay trade-offs combining three different Pareto buffers [WMP+05].

The energy/delay trade-off exhibited by the configurable buffer is spread to the whole module, generating a memory with the same energy/delay trade-offs that the configurable buffers provide. In Figure 2.16 we also reproduce the results for a 1KB SRAM designed for a 65nm technology using configurable buffers [WMP+05]. As in previous cases, it has been validated under a set of variability injections in V_{th} and

β for the transistors which constitute the memory netlist. Once again, each transistor has a different injection assuming variability among them is not correlated. The two-option configurable buffers used in the design enable a two mode memory: a high-speed mode and a low-power mode. As in Figure 2.12, the variability impact is seen as a *cloud*, existing one for each mode available in the memory. The trade-off range between the two options is significant. In delay, the difference between the nominal values from both modes reaches 64%, while in energy this difference is about 30%. Under variation, the design must be robust, so every *cloud* must be separated enough to not overlap with any other. In the example, the range in delay is large enough, existing a separation close to 33% in the worst case.

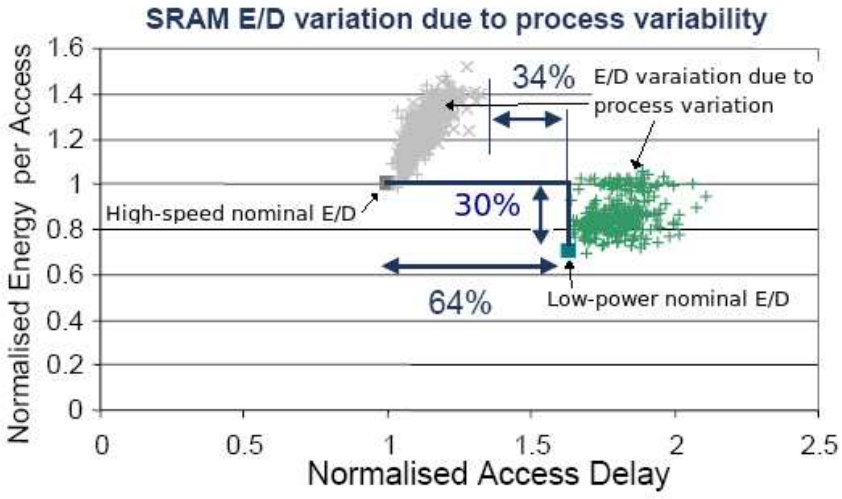


Figure 2.16: A two-mode memory under variability [WMP+05].

The costs of such memory design are related to control and area. In both cases the impact is limited. In area, buffers are only placed in specific locations, i.e. the decoder and wordline drivers, which means less than 5% in area overhead. Regarding control, only a memory control is required, which is a common component in nowadays memories. Furthermore, very few control lines are need, buffers in

the same stage usually share the same control and changes in configuration should not be very frequent. These characteristics turn the cost affordable. On the other hand, the configurable buffers only depends on purely circuits parameters, i.e. the sizing factor and the number of stages, making these component practically independent on the actual technology. Finally, these buffers use the same voltage for all the different energy/delay trade-offs, which makes the circuit design robust to the continuous voltage reduction. The energy/delay trade-offs performed by the Pareto buffers showed in this section depend only on two parameters, the number of stages N and the sizing factor f_i. This is the kind of Pareto buffers that we use along this research, however, it is worth to mention that an extension to research in [WMP+05] involves a third parameter in the generation of energy/delay trade-offs, giving rise to *switchable buffers*. The work, reported in [WMDC09], introduces as additional knob the supply voltage (V_{dd}). Coming back to Equations 2.5 and 2.6, the effect on the energy is clearly seen in Equation 2.6 by means of its direct dependency on V_{dd}. The impact on the delay is done via the parameter t_{p0} in Equation 2.5, the intrinsic delay. The dependency between this parameter and V_{dd} is represented as follows:

$$t_{p0} = K_d \frac{C_{min} V_{dd}}{(V_{dd} - V_{th})^n} \tag{2.7}$$

where K_d and n are process dependent. The methodology explained in [WMDC09] generate for a buffer the values for the options (V_{dd},N_i,f_i) which provide the expected energy/delay ranges. The results carried out on a 8 KB low power embedded SRAM at 65-nm technology shows a four mode memory where the range for energy and delay between the two extreme modes is about 50% as Figure 2.17 shows.

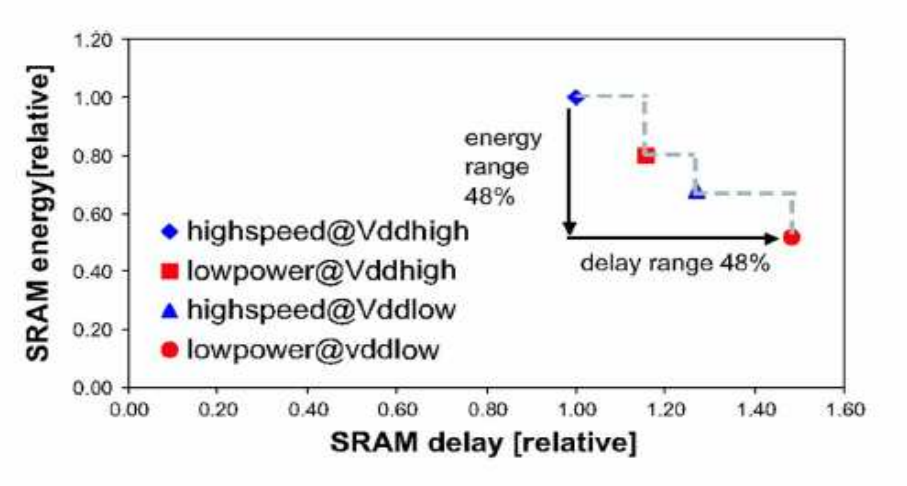


Figure 2.17: Energy/delay tradeoff for a four mode SRAM memory [WMDC09].

Combining the sizing factor knob with the supply voltage knob, V_{dd} may not have to be scaled down too much to achieve the targeted tradeoffs. This help to reduce the implementation and the V_{dd} switching overhead in the real circuit. Similarly to the regular Pareto buffers, the area overhead can be well controlled below 5% and the necessary conversion circuit can be shared with other blocks, also reducing the area overhead.

Although these *switchable buffers* are no applied here, experiments involving them should be considered as future work.

2.5. Conclusions

In this chapter we have reviewed the variability problem, described its sources and its effects on a system. The study has been especially focused on memories, which are one of the most sensitive components on a system regarding variability.

Besides, the memory system is also a very energy consuming component at system level.

The memories can be seen as a collection of sub-components, which can be basically distributed in two groups: the cell arrays and the peripheral components such as the decoders. The research carried out by H. Wang at IMEC, points out the importance that decoders have on the energy and memory delay, especially under variation. Based on this knowledge, they have proposed the design of configurable memories, which provide an interesting energy-delay trade-off. This dissertation is based on this idea.

3

Related work

Manufacturing processes and designers have dealt with variability on platforms as well as in applications for a long time. These problems are not new; what is new is the significant increment that these variations have experimented over the last years. In this section we present a summary about some of the different approaches proposed along the time to deal with each kind of variability.

3.1. Systems under process variation effects

The scenario foreseen for the near future is characterized for:

- Chips composed of billions of transistors, a portion of which are inoperative due to variations happened at fabrication.
- Circuits under the influence of dynamic variations, which affects the supply voltage and temperature, giving rise to different transitory fails.
- Transistors degraded with time, affecting the circuit performance and reliability.

Future designs would not only focus on the usual energy-performance trade-off, but would also have to consider their reliability [Bor05]. Reliability can be defined as the ability of a circuit to fulfil its specifications over a period of time and under specified conditions. Because of technology shrinking, current circuits are not completely reliable, giving rise to failures that can be classified into three categories according to its frequency [Con03, Con02]:

- Permanent failures [Con03, Con02]

  As a result of irreversible physical changes, failures can be permanent after fabrication. In the case of memories, physical failures such as metallization shorts or capacitive coupling can stuck the memory cells at 0 or 1, showing an unchanging behaviour along time.

- Intermittent failures [Con03, Con02]

  These failures are due to unstable hardware, and can be induced by environmental factors such as temperature or voltage variations, or imperfections during the manufacturing process similar to the ones observed for permanent failures. These failures tend to increase due to the higher on-chip densities, as well as temperature and voltage variations. Current solutions are based on error correction, redundancy or error threshold techniques.

- Transient failures [Bor05, Con03, DRV+04]

  These intermittent and non-reproducible failures are not associated with any persistent physical damage on the component. They are caused by cosmic radiation and alpha particles hitting the surface of silicon devices – known as soft errors –, but also by temporal variations due to environmental factors,

such as supply voltage or crosstalk noise. As an example, when high-energy particles or alpha particles strike a silicon component, the energy disturb the distribution of electrons in the semiconductor. It takes about 3.6 eV to generate an electron-hole pair in the substrate and these particles can have energies up to 8 MeV, so one strike can generate about two million electron-hole pairs which are distributed along the semiconductor. In case of a memory cell, whenever the charge generated by the particle exceeds a certain threshold, the stored information will be corrupted, flipping from '0' to '1' or viceversa [Sla10]. The errors appear as short pulses or glitches which flip the correct output in combinational circuits, and can corrupt the state in sequential circuits or flip the bits in the memory cells. Circuit delays are also affected by them. As these errors are not due to physical changes in the circuit, repeating the operation which failed is enough to recover from a transient failure. To deal with these errors, as happens with intermittent failures, error detection and recovery techniques are also used in current designs to mitigate their effects.

As Figure 3.1 illustrates, soft errors are expected to increase with each new technology node. Nowadays, SRAMs, latches and flip-flops are exposed to suffer these failures, especially those on chips for enterprise applications. Nevertheless, low power applications also face this challenge due to single radiation events and erratic shifts in minimum-operating voltages [SIAR10].

The continuous technology scaling increase these failures, reducing the reliability of the devices and, as a consequence, their lifetime. Manufacturing defects due to process variations, which are mainly the ones coped in this research, generate

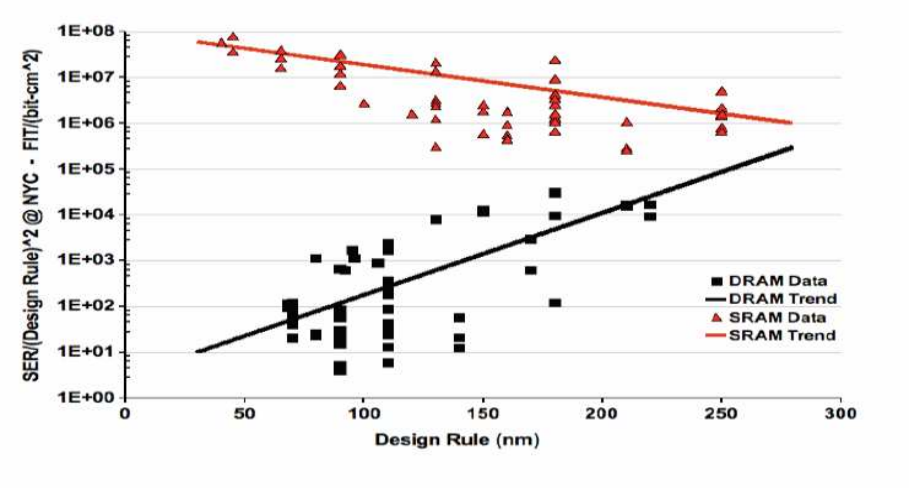


Figure 3.1: Memory soft-error trend per chip [Sla10].

high rates of early life failures when devices start working. Next, the devices enter into a stable period where the failure rate is caused mainly by transient errors. At the end of their lifetime, devices experiments an exponential increment in the failure rate due to aging mechanisms [Cha09]. This behaviour, known as “bathtub" curve, is changing as technology shrinks, being shortened with each new generation.

Reliability poses a new challenge that requires solutions to very different levels: fabrication, design, micro-architecture, testing, software and applications. In general terms, improving reliability is not only a matter of using techniques which reduce the existence of failures, but also error tolerant techniques. Among the mechanisms proposed or developed to deal with failures, it is worth mentioning:

- Addition of new materials and structures
- New designs at the circuit level to increase the system robustness. For instance, single event upsets due to transient and random pulses can be avoided by using tolerant latches which do not alter the signal value

- Conservative designs, based on *worst-case* techniques and static analysis
- Detailed hardware and software tests

However, solely applying this kind of techniques which guarantee the correctness of the system by construction is not enough. Techniques that try to avoid any kind of failure due to manufacturing are extremely difficult to accomplish due to the increasing complexity that systems are achieving, which derives from the continuous scale of integration. Techniques based on the use of worst-case margins or the accomplishment of static analysis are unpractical, due to the generated overhead in terms of energy and time and the inability to tackle the large amount of existing variability sources. On the other hand, the execution of tests to validate the designs are more and more difficult to conduct because of their increasing complexity.

Therefore, techniques which can tolerate the existence of failures based on error detection and correction at run time are essential. For this reason, to check the existence of failures and correct them as much as possible, it is rather common in current designs the inclusion of: (1) detecting and correcting mechanisms based on data redundancy or additional logic; (2) hardware redundancy or replication; (3) temporal redundancy based on repeating an operation several times on the same hardware. These techniques require additional storage or hardware, so they should be applied carefully in order to avoid an inadmissible increment in time and energy. In general, it can be said that, as the reliability problem grows, low-level solutions are not enough to deal with them, while high-level approaches can be really costly. Architecture level solutions can provide a new approach, allowing a more dynamic management and taking advantage of the application behaviour to enhance the system reliability.

Next, it is provided a brief review about some of the solutions mentioned above [INKM05, Bor05].

3.1.1. Overdesigning systems: a large coarse-grain approach

Before variability turned to be a significant problem to deal with, the incipient variations were tackled by means of techniques related to static analysis and design margins, which looked for guaranteeing functional and parametric yield. Robustness and correctness improved with these techniques, while the performance and application timing requirements were also guaranteed [ZHHO04b].

The design margins represent the maximum variability that can be found between any two dice. The aim is to guarantee that a specific design will be able to work under all possible variability conditions. This leads to consider the worst variability situations at design time, assuming that a circuit that performs adequately at the extremes should perform properly at more feasible conditions. The estimation of the highest and lowest values to be used in design margin techniques and static analysis comes from the development of temporal models. These models are based on the process parameters which are considered as the most relevant from the variability point of view, such as threshold voltage or channel length. As Figure 3.2 illustrates, the worst-case assumptions move the parameters to some statistical limit for each of them. The parameters are characterized as distributions, so the limits are usually taken as a multiple of the standard deviation of the nominal value (right-hand side in Figure 3.2). Designs based on this methodology are known as *sigma-based design*. Conservative designs consider $\mu \pm 3\sigma$ points as the extreme values for each parameter, while a lower sigma value relaxes the design conditions.

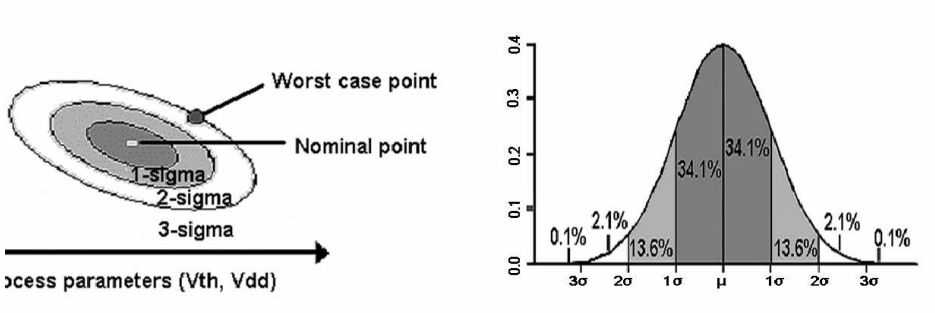


Figure 3.2: The increasing variability forces the increment in the design margins.

As technology has shrunk down, the physical parameters have not been the only ones to be included in these techniques. As the sources of variation and the dimension of their effects have increased with scaling, the set of parameters to consider has been extended with the new sources. Parameters related to environmental variations, such as voltage or temperature, have been also added to the physical characteristics originally considered.

This increment in the number of parameters involves two main drawbacks. On the one hand, it turns the estimation of the margins a hard task, forcing the developing of complex and computationally expensive techniques due to the management of a multidimensional space. On the other hand, the assumption of worst-case values for every single considered parameter, even for parameters that can be independent from each other, generates rather unlikely scenarios. The assumption of these uncommon situations as plausible turns design margin techniques very pessimistic and leads to overdesigns.

Looking for better estimations mainly regarding timing, techniques based on Monte Carlo analysis have emerged as an alternative to the pessimism existing in previous approaches. In this case, the parameters to estimate are randomly var-

ied, following a probability distribution, to analyze the timing circuit and generate statistical results.

The increasing variation makes unfeasible to expect a deterministic behaviour in the system components, which gives rise to abandon the worst-case methodologies and consider these problems from a probability point of view. As an example of such probabilistic behaviour and the increasing challenge that the system characterization based on design margins means, Figure 3.3 shows the current variation exhibited in threshold voltage with the variation expected in future technologies. As the technology scales down, variability on memories increases considerably, which means that designs based on sigma shifts will have to be still more conservative in order to guarantee the yield.

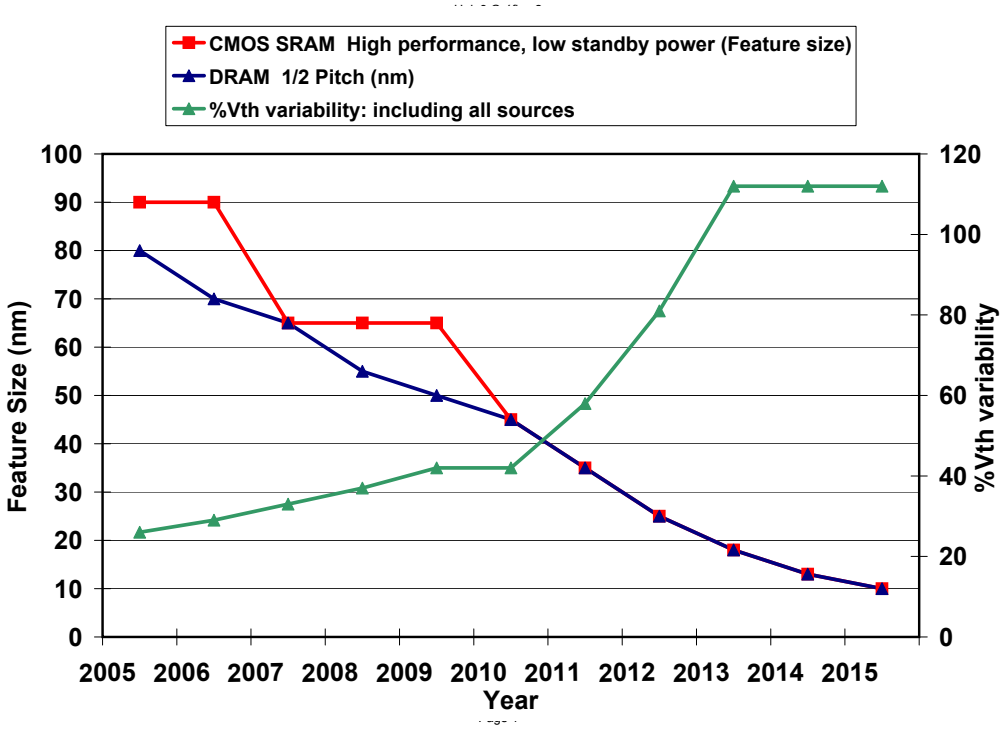


Figure 3.3: Expected threshold voltage variations [SIAR10].

A step beyond worst-case techniques, we find the use of statistical analysis as an alternative to deal with variability, low yields and overheads. Unlike static analysis and design margins, these techniques use statistical information related to the variation of parameters or the correlations among them, enhancing the computational efficiency. Most of the work done in statistical analysis assumes that key parameters affected by variability, such as the threshold voltage or the effective channel length, follow a multivariate normal distribution [HM09]. Gaussian distributions are commonly used to model and foresee the behaviour of combinational and sequential circuits. As technology continues shrinking down, a full guarantee of fulfilment is no longer viable in any of these techniques. The degree of variability found in the new process technologies poses reliability, performance and power concerns, as the margins required to catch the outliers that could appear at run time need to be increased at the expense of getting larger pessimistic estimations.

In this research we are interested in modeling SRAMs memories under variability. Regarding memories, Gaussian distributions have also been reported to model access time variations in SRAM memories [MMMR04] as well as other similar delays at the circuit level.

3.2. Mitigating failures produced by variability

Conservative and statistical techniques commented in the previous section deal with variation oversizing the system to avoid failures and guarantee mainly timing requirements. As we mentioned, failures due to the fabrication process or the environmental factors are increasing as technology scales down and the integration of devices on a chip rises up. These failures, especially the intermittent and tran-

sient ones, lead to a situation where correct implementations cannot ensure correct program executions. In this section we list techniques which deal with variation protecting circuits against failures. The different approaches, from circuit level techniques to architectural techniques, increase the reliability of the components where they are applied.

3.2.1. Circuit level

The continuous reduction in the feature size of each technology node and the higher integration lead to decrement the supply voltage in order to limit the increasing energy and power. This reduction in the supply voltage affects parameters at transistor level such as load capacitance or threshold voltage, making them specially sensitive to the presence of variations. As a consequence, variability at the transistor level becomes evident along the circuit level, affecting its reliability.

Among the different approaches at circuit level to deal with variations and increase reliability, techniques based on transistors with two threshold voltages [Nar05, KKK+08] are rather common. These mechanisms, based on *body bias* schemes, decrease the error rate and reduce the leakage current, which increases exponentially as the threshold voltage falls. At a *body bias* scheme, the transistor threshold voltage is controlled by V_{BS} voltage, which appears in every transistor between the body/bulk and the source. Some of the techniques developed following this scheme appear next.

- *Reverse body bias* (RBB). It consists in increasing the threshold voltage in non-active circuits, which is an effective technique to reduce the leakage cur-

rent in these circuits. As a consequence of the increasing V_{th}, the operation frequency decreases.

- *Forward body bias* (FBB). The threshold voltage is reduced in active circuits to increase their operation frequency and make transistors less sensitive to variability generated in high scale integration components.

- *Bidirectional Adaptive body bias* (BABB). This technique combines the two previous ones to carry out a dynamic adjustment of the transistor behaviour and get a balance between performance and leakage. It is based on the dynamic adaptation of the threshold voltage through applying different V_{BS} voltages to the transistor (Figure 3.4). Depending on this voltage, the threshold voltage will increase to reduce the leakage current, or decrease to enhance the operation frequency. Reverse body bias is used in circuits in standby mode, i.e. less leakage current, while forward body bias is used in the opposite mode to achieve the required performance.

This last technique, BABB, can be used in combination with other mechanisms, such as *Adaptive Supply Voltage*, increasing the effectiveness controlling the leakage currents and the preservation of the circuit performance [Nar05].

3.2.2. System/micro-architecture level

At this level, techniques based on redundancy and self-checking allow to control the output of the circuits. Usually, this control on circuits is done in collaboration with power-aware techniques that vary the frequency of the system. So before explaining the self-checking and redundancy techniques, we focus on a widespread and effective technique to vary the frequency, Dynamic Voltage Scaling (*DVS*).

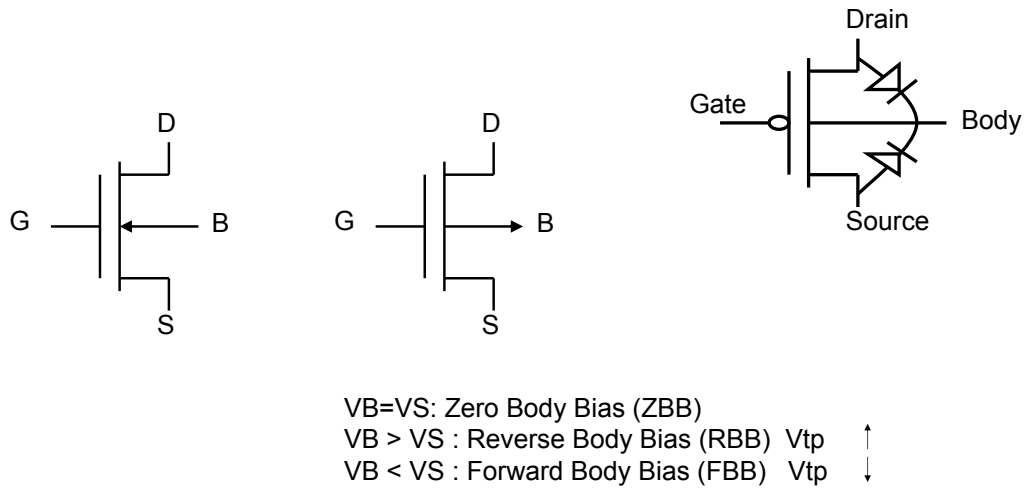


Figure 3.4: Concept of nMOS and pMOS MOSFET transistors under body bias [Nar05].

DVS is based on detecting periods of low resource utilization in the system, so that the supply voltage can be reduced. Decreasing the supply voltage slows down the frequency of the system, and as a result, the energy consumption is also reduce significantly. The increment in the circuit delay due to the reduction in the supply voltage is modeled by the following expression [S. 02]:

$$r_d = \frac{C_L V_{DD}}{\beta (V_{DD} - V_{th})^{\alpha}} \tag{3.1}$$

where C_L is the load capacitance of gate, α is the velocity saturation index and β is the gate transconductance which depends on the transistor aspect ratio (W/L) and other device parameters.

Once the DVS technique has been introduced, we focus on the techniques existing at the system/micro-architectural level to deal with failures derived from variations. At this level it is worth to mention *Razor* [EDL+04, EKD+03], which is

applied for detecting and correcting soft errors in combinational logic, and reducing circuit energy demands in combination with DVS techniques. *Razor* dynamically detects and corrects delay path failures, minimizing the delay and energy consumption of the hardware, not only when a failure happens but also when it works correctly. Otherwise, the energy savings due to the *DVS* techniques would be wasted.

To detect failures, *Razor* adds specific latches, called (*shadow latches*), to the design. These latches are controlled by a delayed clock signal with regards to the main clock, so that the comparison between the value stored in the shadow latch and in a regular one indicates whether a failure has happened or not. Figure 3.5 shows a pipeline stage where these latches are used. In this case, the result from logic stage L1 is firstly stored in a flip-flop, following the main clock signal, and later on stored in the shadow latch, following the delayed clock. If stage L1 finishes its execution within the setup time for the main flip-flop, the shadow latch stores the same value, which indicates the correction of the logic circuit. If the execution of the L1 stage takes longer, the value stored for the main flip-flop will differs from the one stored in the shadow latch, raising the error signal for that pipeline stage. In case a failure happens, the shadow latch always contains the correct output for the L1 stage. After one-cycle penalty, the right value is inserted in the pipeline to be used in the following stage, in case of Figure 3.5 the correct value would be used as input for the L2 stage. This way, *Razor* avoids the complete re-execution of the instructions. A possible implementation of the pipeline error recovery mechanism can be seen in Figure 3.6.

To guarantee the robustness and correction of the shadow latch, this component is designed to work properly even under the worst-case conditions. For this purpose,

the operating voltage is constrained at design time in such a way that the logic delay cannot exceed the setup time of the shadow latch.

One of the *Razor* key features is the existence of an error rate, which is monitored to tune the voltage supplied to the circuit. A low value of this error rate can indicates that the instructions are being executed too quickly, so the voltage can be slowed down. However, a high rate indicates that many time constraints are not being met and the voltage should be increased.

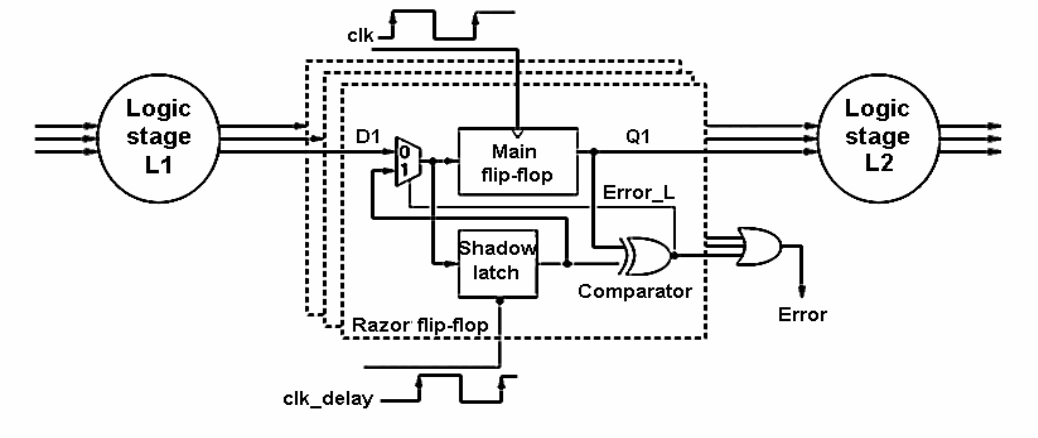


Figure 3.5: *Razor*: Error detection mechanism based on shadow latches [EDL+04].

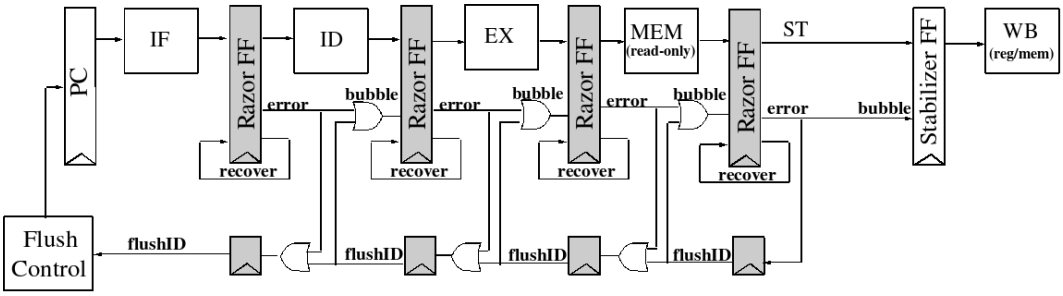


Figure 3.6: Pipeline recovery mechanism based on clock gating [EKD+03].

3.2.3. Architecture level

One of the existing mechanisms to deal with variability at architecture level is *DIVA* [Aus99], a *Dynamic Implementation Verification Architecture*. In this architecture, two processors are designed to deal separately with performance and correctness. The main processor is a sophisticated core, for instance an out-of-order superscalar processor, which pipeline consists in the usual stages in these architectures. This is a regular processor except for the absence of correctness modules. Due to the separation among performance and correctness, the main processor can be simplified to enhance the performance and reduce critical paths by removing those modules that are included in the auxiliary processor.

The auxiliary processor works as a checker of the main processor to verify the instructions executed on it. This checker is designed to ensure the correctness and robustness independently of the device characteristics. The nature of variations, static or dynamic, does not affect the system. For this purpose, it is designed with larger transistors, even with larger timing and voltage margins, increasing in this way the system reliability. Furthermore, as the checker is a very simple processor, the energy overhead is admissible.

DIVA architecture starts working once the main processor finishes the execution of an instruction. At that moment, as can be seen at Figure 3.7, a new stage is inserted in the pipeline, which includes the commit stage. The output generated in the execution stage is stored in a reorder buffer and considered speculative. After this, the output and the input operands are sent, in program order, to the checker processor where a functional checker is performed. In this new pipeline stage, the integrity of the functional units in the main processor is verified, as well as the

communication between the memory and registers. This is done through two independent and parallel verification pipelines existing in the checker as Figure 3.8 shows. In order to test the functional units, the CHKcomp pipeline re-executes the instruction and compares the result with the one obtained from the main processor. The same technique is applied to the other pipeline – CHKcomm –, which verifies the correctness of the memory and register inputs operands. Figure 3.9 summarizes the way the checker works for each instruction type. Firstly, the registers and memory addresses are read in the RD stage; so later on can be checked (CHK). In case no exception is raised, the results stored in the reorder buffer are confirmed and the instruction is completed in the CT stage. If an exception happens, the checker processor contains the right data. The exception forces the flushing of the pipeline to continue with the instruction.

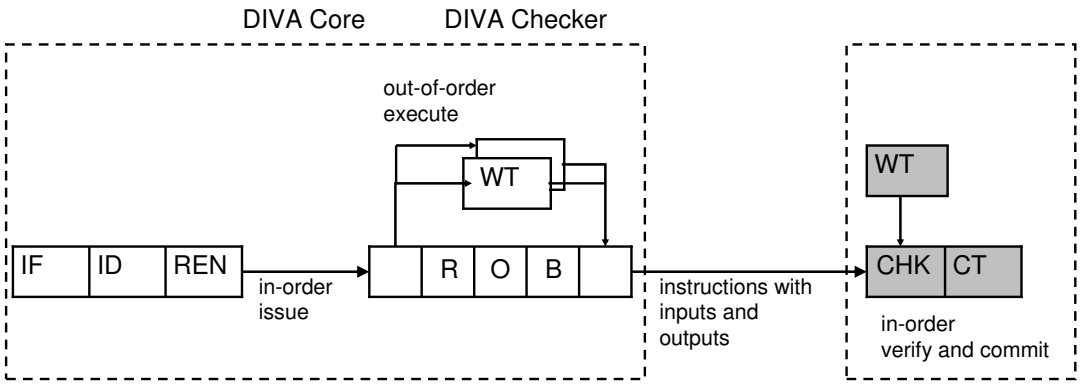


Figure 3.7: *DIVA* structure consisted of a main processor and a checker module [Aus99].

Based on the *DIVA* methodology, multi-processors could be a useful way to enhance the system reliability taking advantage of the large amount of transistors existing in current designs. In this approach, known as *resilient microarchitectures*,

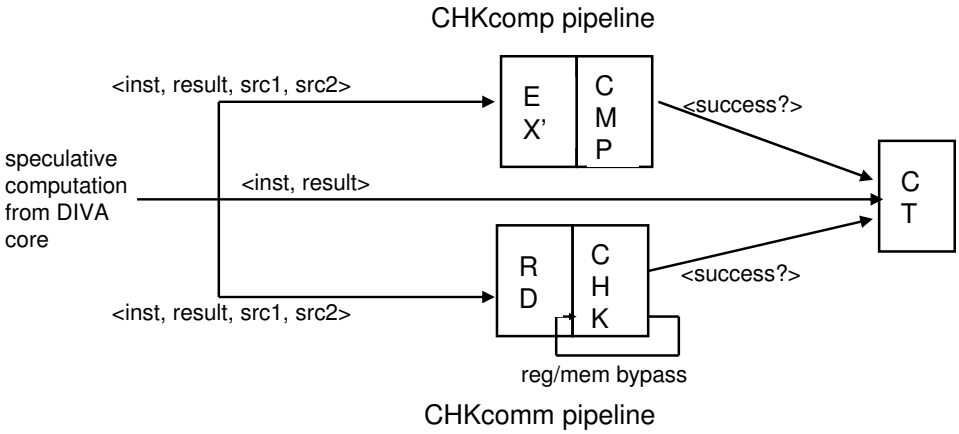


Figure 3.8: Structure of the checker module and interface with the main processor. [Aus99]

a subset of the existing multicores assists in redundancy tasks, being the own system capable of testing and reconfiguring itself along its lifetime.

Nowadays we can find resilient architectures in networked embedded systems. For instance, [LGT09] proposes a technique which takes advantage of the data redundancy present in distributed applications. By using application and architecture graphs together with Boolean functions encoded in Binary Decision Diagrams (BDDs), algorithms are capable of identifying data-redundancy of distributed functions. An appropriate design space exploration exploits the data-redundancy to increase the reliability while optimizing different objectives.

In [MAW08] it is developed a reliability-aware design of a low density parity-check (LDPC) for parallel or partly parallel architectures, where the the amount of resource redundancy – double or triple – is based on the protection priority of the corresponding resource.

stage / inst class	RD	CHK	CT
ALU/ AGEN/ BR	read src1 read src2	check src1 check src2 wait for CHKcomp	WB result or exception
LD	read addr read mem	check addr check mem	WB result or exception
ST	read addr read st data	check addr check st data	ST mean or exception

Figure 3.9: Outline of the CHKcomm pipeline for each instruction class. [Aus99]

3.2.4. Memory level

As technology shrinks down and the density and area devoted to memories increases, the stability of the memory cells is reduced and the chance of an alteration in their values increases. The signal-noise margin (*SNM*) indicates the minimum voltage noise that is necessary to introduce on a memory cell to flip its state. This margin decreases due to the increasing fluctuations among the threshold voltages that constitute the cells. These fluctuations, together with the reduction in the circuit supply voltage, lead to an increment in the number of transient failures caused by random defects. To protect memories from this type of errors and increase the functional yield[1], a wide variety of techniques have been proposed and developed.

One of these techniques is based on the redundancy of rows or columns at memories[KZK+98, CHS+07] to repair single cell faults. Redundancy based on columns or rows depends on the expected fault distribution. Thus, row replacement is more effective repairing faults in word lines, word line drivers and word line

[1]Functional yield is a reflection of the quality of the manufacturing process which estimates the percentage of samples that generate the right output for all the possible inputs.

decoders; while columns are more effective in bit lines, column multiplexer or column line decoder. Considering column redundancy as example, when a faulty cell is detected, a reconfiguration unit repairs the fault by remapping its corresponding column on a free spare one. This redundancy leads to significant hardware overheads. Furthermore, the number of errors that can be detected is reduced. For these reasons, these kind of techniques are not enough to deal with errors.

Another sort of very popular techniques are based on Error-Detecting Codes (*EDC*) or Error-Correcting Codes (*ECC*). *EDC* techniques detect errors through parity checking but cannot correct them, while *ECC* can reconstruct the bits in case of failure. The number of bits that can be reconstructed depends on the algorithm used. Conventional techniques which apply *ECC* can detect and correct 1-bit errors without applications can notice them, as well as detect 2-bit errors. These techniques are based on the existence of auxiliary bits to control the information stored in the memories. In the past, both approaches increased considerably the system reliability, since the frequency of failures in more than one bit was really low. However, as the amount of memory increases, the problem of multibit failures has also risen. This situation turns conventional techniques insufficient to ensure the integrity of the stored data and make necessary the use of more sophisticated techniques, such as Chipkill [Del97].

Chipkill is an advanced form of ECC to deal with multibit errors in DRAM memories. The technique, similar to the one used for RAID systems, is based on checksums which are stored separately from the original data. In case a memory fail occurs, the information can be recovered by means of its corresponding checksum.

3.2.5. Hardware for monitoring system functionality

The system architecture proposed in this research is based on the existence of mechanisms to measure the integrity of the system. These mechanisms, the monitors, are used to measure the system functionality at run time for debugging or engineering reasons, but also for determining operating margins or assisting adaptive mechanisms. These components help to deal with static variations generated in the fabrication process, as well as with the dynamic ones which can appear at run time as a consequence of aging or temperature.

In our case, monitors are used to check the impact that process variation has on the energy and access time of the memory structures involved. This section outlines some of the monitors proposed or implemented in recent years.

Literature regarding monitors which predict or sense timing violations is quite extensive. Well known techniques such as Razor – Section 3.2.2 – include circuits to manage timing violations. These circuits basically consist in comparing signals with a delayed version of themselves, looking for mismatches in that comparison, i.e timing failures. The delayed signals can be used in many different ways. Just for mentioning a couple of examples, Razor is based on the use of a delayed clock signal in the shadow latches to detect and correct errors. On the other hand, the technique illustrated in [EHG+07] to deal with variation, is based on using flip-flops with a setup time larger than regular flip-flops. These flip-flops, called Crystal Ball flip-flops, are placed at the end of critical paths in parallel with a regular flip-flop. In case of a critical timing situation is detected due to a low supply voltage, the Crystal Ball flip-flop notifies the situation before a timing error happens. The detection of a critical situation triggers a compensation system based on the supply

voltage to avoid the failure. In both cases, monitors are used to adjust the system by means of changes at the supply voltage, detecting errors as in Razor or preventing them as in Crystal Ball.

Regarding memories, an example of delay and leakage monitors used to repair failures in memories in combination with adaptive body bias can be found in [MKMR05]. The approach used in this case is based on the online leakage monitoring and delay monitoring of an SRAM array. The leakage monitor measures the current leakage and compares it with reference currents. Based on this comparison the right body bias – forward or reverse – is selected and applied to the memories. The delay monitor has the same purpose, i.e. selecting the proper body bias to reduce failures. In this case a long inverter chain and a counter are used to measure signal delays. The time required for a signal to complete the inverter chain indicates the body bias required in the system.

Voltage and temperature monitors, related to power consumption have been illustrated in [PPV+06a].

Monitors are part of signal-integrity self-test architectures (SIST) [PPV+06b], which include controllers to manage when and how monitors need to be used. To reduce the impact on area and power, state-of-the-art methodologies optimize the placement of these monitors [AAM+09].

3.3. Dealing with dynamic behaviour exhibited at application level

The technology improvements which are taking place nowadays allow to execute, on embedded platforms, applications that were restricted to general purpose processors in the past. These applications usually have time constraints to meet and the devices which host them are generally energy limited. These two characteristics need to be considered during the design process in a wide range of techniques. Thus, techniques related to power management or voltage scaling consider carefully the deadlines while dealing with the energy problem. Other aspects such as the scheduling policies carried out by the operative systems also have deadlines into account. In all cases, these techniques need to collect information, as accurate as possible, about the application behaviour in order to provide energy-efficient solutions guaranteeing the time requirements. This information, related to the application workload and execution time, is based on estimations or predictions which are usually carried out by means of profiling.

Traditionally, static and dynamic profilers have been adopted to estimate the worst-case execution time, the average-case execution time, even the best-case execution time. static profilers rely on static timing analysis for its estimations without running the application at all. This static approach [LPB+05] consists in application flow analyses to calculate the expected execution time. Control-flow analysis methods study the application code to provide information about data dependences, loop bounds, the longest paths, etc. In case the target hardware is known, the object code is also analyzed to determine the application timing behaviour on the hardware. The information provided in these analyses is combined to determine the

final execution time. However, this static approach based on stationary workloads is too conservative. Overestimating analyses do not provide accurate estimations for hardware dependent applications and applications whose workload depends on input data, which can be highly variable.

Dynamic profilers [Mul04] takes measures of the execution time by running the application on the hardware. In this case, the set of data inputs used in the profiling should cover as many representative cases as possible regarding workload and input constraints, otherwise the execution time can be underestimated. Both profilers can be used in a combined way in order to improve the calculation of the worst cases. Recently, statistical methods such as cumulative distribution function (CDF) or probability density function (PDF) have also been proposed to estimate the probabilistic worst-case execution time by offline profiling [XLL08, HYB+08].

These offline approaches always consider the target application in the same way, as a unique and individual scenario. This approach is especially restricted for dynamic applications which exhibit different behaviours at runtime. Considering the plural nature of the application behaviour, we can manage the different behaviours at run time in an individual manner. In this way, execution time estimations are more accurate than those provided by former techniques. We have adopted the system-scenario-based approach proposed by [MWP+00, YC04, MCB+04, PBC+05, GBC06], extended in this research to handle application variability. The *system scenario* approach proposed in these works is integrated in a full methodology to map applications efficiently in terms of energy and area. This methodology, named Data Transfer and Storage Exploration (*DTSE*), is explained next.

3.3.1. DTSE methodology

DTSE [CdGS98] is the acronym for Data Transfer and Storage Exploration, a design time methodology developed at IMEC which assists to map applications onto embedded platforms in a semi-automatic manner. This methodology is focused on the memory usage and the reduction of the memory storage and transfers involved in data dominated applications. Through systematic, ordered and orthogonal steps, the aim is the minimization of the power consumption of a computer platform.

DTSE works at source code level, carrying out modifications in the code to implement the necessary optimizations which help to reduce the power. The application of DTSE allows to develop a memory and transfer organization with the following characteristics:

- It avoids redundant data transfers as much as possible
- It enhances data locality and regularity in the memory access
- It develops a memory architecture where the smaller memories store the most accessed data
- It avoids the use of N-port memories as much as possible

The key issue in the DTSE methodology is the concept of data locality, which means that although large amount of data are stored in memory arrays, computations are centered in a small portion of those data each time. Analyzing the application code and rewriting it adequately is possible to decrease the energy consumption, and even potentially, increase the performance.

Figure 3.10 depicts the main tasks accomplished along this methodology. The first DTSE steps regarding global data-flow, loop and control flow transformations

are focused on improving the data access locality. In these steps, stages 1 and 2 in Figure 3.10, intermediate storage removing between production and consumption or loop transformations are carried out.

The following steps are related to the development of a memory hierarchy. Based on the analysis of data dependencies and their overlapping lifetimes, there is an estimation about the memory size and a study about data reuse (stage 3). At this point, decisions about the distribution of the data along the hierarchy levels will determine the memory access frequency and the memory size. These steps lead to store the most accessed data into small memories, which consume the least power, but there is also a trade-off between the power lost by adding memory levels and the power gained by reducing the access to larger memories in higher memory levels.

The following tasks in the methodology concern the memory allocation and assignment, which depends on the available cycle budget and other timing constraints (stages 4 and 5). To conclude the methodology, techniques to optimize the memory data layout (stage 6), such as inplace mapping, are carried out.

This DTSE methodology can be extended to manage concurrent contexts by what is known as T-DTSE (Task-level Data Transfer and Storage Exploration) [MMS+07]. In this case the application is split in tasks that may be executed concurrently and study the relationship among them and the way the data involved are accessed. This information allows to optimize the memory accesses to those data. To conclude, the effectiveness of this DTSE methodology can be increased by focusing on target application domains rather general purpose computers [CDK+02], as the characteristics of a well-defined domain can be better exploited.

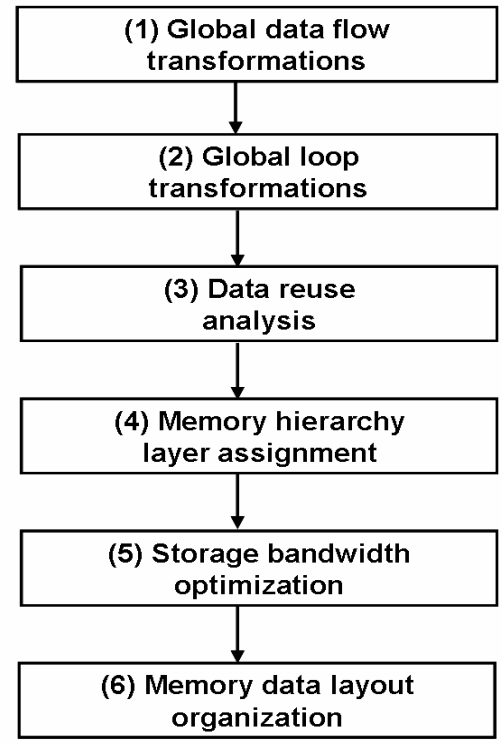


Figure 3.10: DTSE methodology.

3.4. Conclusions

In this chapter we have reviewed the different mechanisms developed to manage the increasing variability in each new technology node, as well as the rising dynamism on current applications.

Approaches exclusively based on design time mechanisms, such as worst-case margins, have proved to be too conservative and inflexible to deal with variations, so more and more approaches based on dynamic techniques have appeared to replace them. Variation can be better tackled at run time, so the system can be adapted to the particular variability characteristics. At this point, monitors become necessary elements to evaluate the state of the system by measuring the critical parameters

which will be used to adjust it. These dynamic mechanisms enhance the system reliability reduced by the existing variation, but also deal with the increasing complexity in current designs and the unfeasibility to execute detailed hardware and software tests which guarantee its correctness.

Failure tolerant mechanisms have been developed at every level, from circuit to architecture level, mostly focused on performance and correction improvements, but also targeting on power reductions. Most of the techniques mentioned in this chapter are focused on the processor and its improvements in timing yield. Memory architecture has usually been considered apart, developing specific detection and correction techniques to ensure data integrity at memory level. However, the growing area devoted to on-chip memories and the larger variability impact that suffer, increase the relevance of the memories in current and future designs. These two characteristics make necessary to consider memories as components which can play an important role in the increment of performance and the energy reduction of the entire system.

Regarding application dynamism, worst-case techniques are neither a valid solution as they are not dealing with variability. Different system adaptation techniques have been developed based on using information about the actual behavior of the application.

In this research, we propose a system level approach to deal with both sources of uncertainty in a comprehensive manner. At system level is possible to manage a large range of variation effects on platform without explicitly considering the source – leakage, voltage, physical parameters, etc – which generates the intermittent failures. We have more opportunities to get better energy-performance trade-offs at

system level than at any lower level due to all the information available about the application and the system requirements.

The methodology proposed in this research does not replace the techniques shown in this chapter, neither is incompatible. Some of the techniques work on data correction for example, while we look for avoiding the overhead undergone using worst-case techniques.

4

Dealing with application dynamism by means of *system scenarios*

So far we have focused our discussion on the impact of process variation on modern platforms, paying special attention to its effects on the memory system. However, apart from platforms, it is possible to talk about variations or uncertainties at very different levels. For instance, variations can also be found in applications themselves, due to the dynamic characteristics that they currently exhibit.

This additional source of variation is also taken into account in this research due to its impact on the energy and performance at the system level. In the previous chapter, the energy and delay challenges caused by variations in memories were coped with the use of configurable memories. In this chapter, to tackle variations at the application level we make use of the *system scenario* concept, a methodology developed at IMEC [MWP+00]. Later this concept has been extended into a sys-

tematic methodology, mainly in partnership with TU/e[1] [GBC06, GPH+09], which is explained next.

Firstly in this chapter we show that modern applications exhibit a large amount of dynamism in current embedded systems, which impacts on their energy and performance. Next, the concept of *system scenario* and the methodology to automatically detect and exploit them are explained in Section 4.2. Information about the application used as main case study in our research work is provided in Section 4.3, an MP3 decoder. Finally, Section 4.3.2 describes the *system scenario* division applied to the decoder and the code transformations that this division makes possible.

4.1. Application dynamism. Implications in performance and energy

The constant increment in transistor density happened in last decades and the growing percentage of area devoted to on-chip memory have made possible that a huge variety of applications can be executed on current embedded systems. Most of these systems are running multimedia and telecommunication applications which execute complex tasks such as high-quality sound, video processing, wireless communication or 3D graphics. These tasks result in higher demands on digital processing, caused by their large data demanding, user interaction, complexity or real time constraints.

These characteristics lead to a huge dynamism at the application level, with a high degree of data-dependent variability in their executions. An example of such a

[1]Eindhoven University of Technology

dynamic application is the frame-based MPEG-4 decoder. The processor workload can vary by more than an order of magnitude depending on the number of existing objects in the frame or the number of meshes that need to be processed [LCO+05]. This different workload per frame is related to the amount of motion existing in the scenes. Thus, dynamic scenes with a lot of camera movements have a higher number of meshes than more static scenes. In these applications it is possible to find differences as large as a factor of ten between the average load on a processor and its worst-case workload[RvEJ+02]. Other codes, such as the H.264 video decoding standard, exhibit similar behaviour [ASRV05].

Application dynamism can be expressed as variations in the amount of data consumed and produced in each processed sample. Other works, such as [MLCO04], have also linked the application dynamism to the inputs or the quality parameters. Such dynamism at the application level has to be managed efficiently to meet the performance requirements, but also to keep the system energy budget in low rates. Meeting both constraints is challenging as these applications are usually memory intensive and the on-chip memory is responsible for a large part of the energy in the whole processor. Previous research has evidenced the relevance that the memory system has on the energy budget. Results reported in [AJB+05] show energy percentages higher than 30% for data memories, rising to 50% when instruction memories are also taken into account in embedded processors. Similar results are reported in [DBBS+08], [BR08] or [CRL+10].

It is clear that assuming worst-case execution times everywhere ensures the timing constraints. However, this traditional and conservative approach also compromises the performance and the energy of the system. We opt to deal with this dynamism in a more flexible way, using *system scenarios*. If we look at the dynamic

behaviour from the user's point of view, we see the dynamism just in terms of functionality, timing behavior or interaction, what is referenced as use-case scenarios in the literature. But we can also see the dynamism from the designer's point of view, where aspects such as the memory usage, processor cycles, input data or power are much more relevant. The *system scenario* concept developed in this second context has been used in this research to deal more effectively with application dynamism. *System scenarios* in that context can deal with performance, as well as with energy cost, when they are taken into consideration during the design stage, specially during the memory design to get an efficient hierarchy which helps to reduce the energy of the system.

4.2. *System scenarios*: Concept and methodology

The more information known about an application, the better the application can be mapped onto a system. This thought can summarize the idea behind the *system scenario* concept. A deep knowledge about the target application can help to design a more efficient embedded system in terms of energy and performance. All available information about the functional and timing application behavior lead to take more specific decisions during the different steps of the embedded system design as well as at run time.

As a motivational example we can think of a frame-based application. Each frame arrives to the system at a given rate and it may be delivered at a different output rate. In the middle, information contained in frames is manipulated in a specific way depending on the type of frame. The different existing frames are determined by the distinct kernels involved in the data processing, which means that

the resources to be used can be different from frame to frame and that the workload can vary largely among them. This variability will impact on the time and energy needed to manipulate the information, as some frames will require all the available time during the decoding phase while others will only need a portion of it.

If these variations in time, workload or resources can be known at design time and foreseen at run time, they can be used to optimize the design of the system instead of using worst-case assumptions which degrade performance. It would be possible to reduce efficiently the system activity at run time, mainly at the processor level, keeping the performance with a minimum energy cost. However, every single variation in characteristics such as time, workload or memory usage cannot be considered relevant, neither at design time nor at run time. If so, the system complexity due to the exploitation of such amount of information would not be affordable to manage. It is necessary to find a trade-off between the cost of the information to manage and the expected gain. This means it is necessary to determine which are the most representative application characteristics in such a way that the application behavior can be entirely described by them in terms of cost. This clustering of information is what we call *system scenarios*, and it is used to improve the system design as well as the quality of the system at run time. In this way, the application behaviour is represented by a set of *system scenarios* which cover all the run-time situations. Each *system scenario* includes the run-time situations that can be considered similar in terms of cost and behavior. This clustering of information describes the way the application is expected to behave in a near future and how the system has to be adapted at run time to be efficient in case a new *system scenario* is executed.

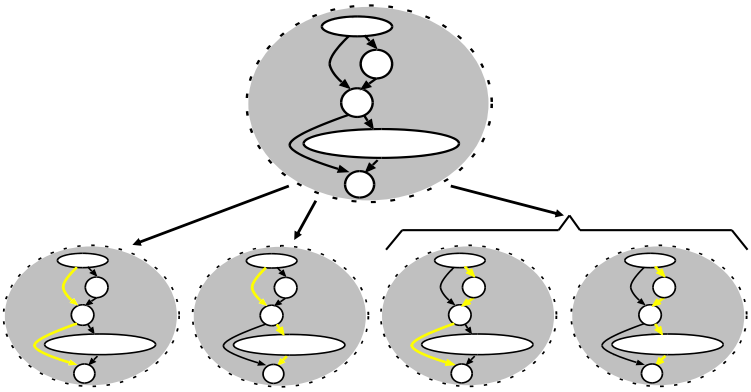


Figure 4.1: *System scenarios*: Depending on the user input and data dependent conditions, different parts of the application are executed. The cost in energy or execution time can vary. The concept of *system scenario* tries to capture the dynamic behavior and avoid the worst-case assumptions.

Figure 4.1 graphically illustrates how a given application can be divided into *system scenarios*. The big bubble on the top represents the original graph of the application control flow. Below, each one of the small bubbles in the figure represents the different paths that the application can follow during its execution. The paths covered in the flow graph depend on input parameters or data-dependent conditions in the source code. This characteristic allow to identify the different run-time situations (RTSs) present in the application behaviour. When several paths have a similar run-time situation, as it happens with the two bubbles on the right-hand side of Figure 4.1, they are clustered together in a single group in order to minimize the number of different application behaviours. Each group of different execution paths is called a *system scenario*. According to this description, the application shown in Figure 4.1 as example is characterized for three *system scenarios*.

To better understand the role that the *system scenarios* can play in the energy and system performance, the concept of *system knob* – or simply *knob* – need to be introduced. The existence of scenarios entails the possibility of accomplishing at

run-time changes in parameters which impact on the system cost. Any parameter that can be tuned at run time and effect the cost can be considered a feasible *system knob*. Any application characterized by *system scenarios* has at least one *knob*, and each *RTS* has associated a specific (set of) knob position(s) to impact on the system cost, while meeting the timing constraints. This cost is usually measured in terms of energy or power and it typically involves a Pareto trade-off that is represented in a Pareto curve for each RTS. For each *system scenario* the worst-case Pareto curve combination (convex hull) is then used to represent that scenario and any run-time situation belonging to it is then linked to the platform mapping (defined by knob positions) associated with that worst-case Pareto boundary.

An example of a widespread *system knob* can be considered the operating voltage used on Dynamic Voltage Scaling techniques, which affects directly the system performance. According to this example, the different voltages that can be adopted by the system represent the knob positions available in the system. Apart from voltage, many other parameters can be considered as knobs. For instance, the processing elements available in a multiprocessor system can also be seen as a *system knob*, or the number of elements available in each particular moment would be a knob position[SHA02] as well. Another example can be the different versions of a specific source code that can be used for an application depending on the required cost[PBC$^+$05]. This last example shows that *system knobs* are not only of hardware nature, but also software can impact on the system cost.

The exploitation of the *system scenario* concept requires to take decisions at design time such as the extraction of the *system scenarios* or the selection of the suitable *knobs*; but also at run time, when the overhead due to detection and the switching between *system scenarios* has to be considered and any unexpected sit-

uation has to be managed. To face up efficiently the dynamism at the application level, these decisions are taken in a specific way giving rise to the *system scenario* methodology, which is described in the following section.

4.2.1. Methodology

The different steps which make up the *system scenario* methodology were presented in [GPH+09] and [GBC08] and its main steps are described in Figure 4.2.

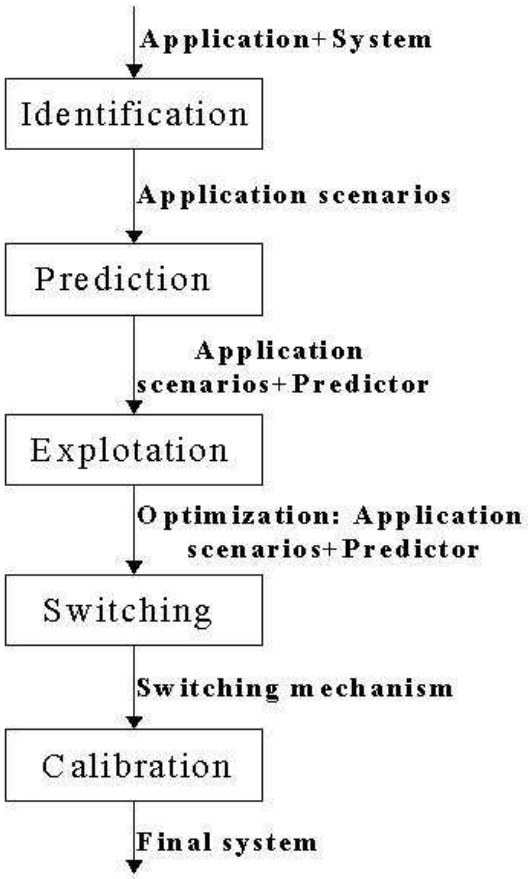


Figure 4.2: *System scenarios* methodology. Summary of the main steps performed at design time [GBC08].

The **identification** step determines a preliminary collection of *system scenarios*. This first step highlights the most characteristic parameters in terms of application behavior and execution cost, among all the different parameters that can be found during the application profiling. The values of the different parameters are used to distinguish among different run-time situations. Once the division has been done, it is assigned a cost to each one of the situations observed. This cost is based on a multi-objective function which depends on the *system knobs* we want to apply on the system. After the discovery of the relevant parameters, the run-time situations with similar cost are clustered, giving rise to the *system scenarios*. The clustering needs to take both, cost and *knob* settings, into account to guarantee that only situations with compatible *knob* settings and similar cost are clustered in a single scenario.

The parameter discovery consists in looking for control variables in the application source code that can have a certain impact on the application resource requirements, such as the number of cycles, the memory usage, etc. First, the variables are statically identified analyzing the source code. The output of this analysis is a list that identifies data and control variables. The first ones represent the data processed by the application and usually appear as arrays, explicitly or implicitly declared. The latter are the ones that influence the execution time of the application and give information about which parts of the code are applied and their activation frequency.

Once the list of variables (parameters from now on) is available, it is necessary to find out which ones are the control parameters which influence the application the most. For this purpose, the application code is instrumented with profile instructions for all the read and write operations corresponding to each parameter in the list. The instrumented code is executed using a training bitstream and the resulting trace is analyzed to identify those parameters which do not have a large influence on the

application behavior, according to different heuristics. The ruled out parameters are deleted from the list and a new execution is done over the instrumented code, using a larger training sequence each time. This step is repeated until the whole training sequence is processed and the list of parameters cannot be reduced anymore. The success in the parameter discovery depends on the training sequence. This should be representative enough to cover most of the situations that can appear at run time. However, finding a good training is usually a hard task and the identification can miss parameters as it cannot be exhaustive.

As this identification step is non-conservative, there is usually a backup *system scenario* to cope with the unexpected situations that can appear at run time. The calibration process is in charge of performing it in case it is necessary.

Once the relevant parameters have been identified through profiling and instrumentation, the different run-time situations which identify these parameters are clustered to create the set of scenarios. The clustering, based on a multi-objective cost function, takes into account run-time aspects such as: (1) how often and for how long a specific situation occurs, (2) cost of the impact that the clustering of some situations have on the system, (3) expected overhead in terms of storage, (4) number of switches among scenarios, (5) cost of the prediction, etc. To cluster situations also the *knob* settings must be compatible among them. The final cost for a *system scenario* is the maximal cost among all the run-time situations involved in it.

Graphically, this clustering process is illustrated in Figure 4.3, where we consider an application whose run-time behaviour is represented by four RTSs. A two-dimensional cost function is applied to each RTSs, characterizing them in terms of the costs (upper part in Figure 4.3). Next, the cost information of the four RTSs,

represented as Pareto curves, are compared in order to decide the appropriate clustering in scenarios (lower part in Figure 4.3). In this case, the similarities in cost exhibited by *RTS3* and *RTS4* turn them into candidates to be considered as a single *system scenario*. The other two RTSs are different enough to represent its own scenario.

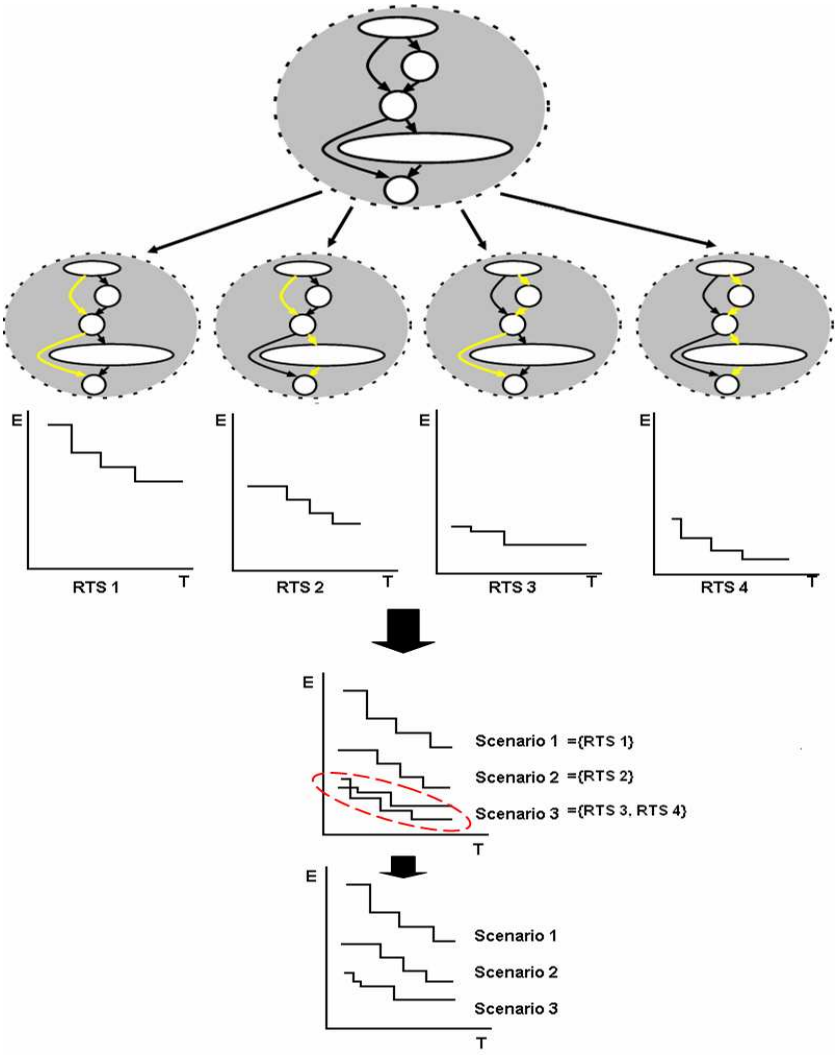


Figure 4.3: *System scenarios*: multi-objective cost functions lead the clustering among the different run-time situations (RTSs).

Once it has been established an initial collection of *system scenarios*, the correspondent predictor is developed, to determine the appropriate *system scenario* which will be executed on the system at a specific moment. This is the aim of the **prediction** step.

As well as the *system scenario* identification is based on the values that specific system parameters have at run time, the predictor uses these same values to decide which *system scenario* is running.

At run time, an important factor to take into account is the moment when the *system scenario* is predicted. The earlier a *system scenario* can be predicted, the highest the profit that can be achieved is. The prediction needs to be done as soon as possible, but it is also necessary to know the values that the system parameters exhibit to get a right prediction. This consideration can lead to change the set of the initial parameters for others that can be known earlier, or even to use probabilistic prediction to select the *system scenario*. Depending on how the prediction mechanism is applied we can talk about centralized or distributed predictors. A centralized predictor is called at a point of the source code shared for all the *system scenarios*. On the contrary, a distributed predictor may not share any prediction point in the code.

As the predictor is used at run time, it is desirable that the evaluation cost is as low as possible. For this purpose, the implementation is based on a decision diagram, a directed acyclic graph whose inner nodes are the system parameters and their edges are the different values for those parameters. The sink nodes correspond to the *system scenarios*.

During the design of the predictor, considerations such as the overhead or the moment when the predictor acts can lead to reduce the system parameters or even the set of *system scenarios*.

The **exploitation** step uses the increasing information about overhead and complexity, derived from the previous steps, to improve the *system scenarios* discovered. This step includes from variations in the number of *system scenarios* to optimizations performed in the source code to enhance computation or energy. These optimizations try to decrease costs such as energy, performance or reduce the size of the to be stored. Nevertheless, this step is highly dependant on the context in which *system scenarios* are applied, so it cannot be fully generalized. Thus, although optimizations can be done individually for each of the *system scenarios*, we take the risk of being suboptimal for the system cost perspective. However, if the whole set of *system scenarios* is taken into account we could find suboptimal optimizations for individual *system scenarios* that lead to optimal system costs instead. In this step, in order to get more efficient optimizations and reduce overheads, the information from the different *system scenarios* need to be considered.

Regarding *system scenarios*, another important issue to consider at run time is the swapping between them, which can incur in overheads in time and energy. In the **switching** step it is developed a mechanism to decide whether to switch to a different *system scenario* or not. Several factors have to be taken into account before deciding if the switching is suitable, and if so, select the appropriate *knob* position. Having a set of *system scenarios*, it must be considered the *system scenario* cost as well as how often a switch between two given *system scenarios* happens at run time and the overhead that each switch entails. The overhead depends on what the switching implies on the system; for instance, changing the voltage or the frequency

do not have the same penalty than loading at run time a different source code. This overhead can affect the system cost, in terms of increasing energy consumption. But it can also impact on the run-time characteristics because of deadline misses due to the timing cost.

There must be a trade-off among the *system scenario* cost, the frequency of switching and the expected gain to avoid spending too much time switching if the gain does not worth it. These trade-offs can lead to refine at this point the set of *system scenarios* established in the previous steps. Thus, two *system scenarios* initially independent but with a high switching frequency and a relatively close cost can be merged into a common scenario assuming the worst-case cost between them to reduce the overhead.

Another potentially useful mechanism to develop is the responsible for monitoring and calibrating the system. The **calibration** step deals with the problems related to the use of *system scenarios* that can appear at run time. The profiling carried out during the identification step to extract the *system scenarios* cannot consider any single possibility that can appear at run time. This can lead to situations at run time which were not considered at all during design, or were underestimated. This is the case, for example, of run-time situations which do not match any *system scenario* or situations which belong to one of the existing *system scenarios* but exhibit a cost higher than predicted. In these cases as well as in others, the system needs to be calibrated in order to keep quality.

Due to the overhead in time and energy that the calibration process can imply, this mechanism is only performed rarely, when the system workload is low, so it not affected by the process, or whenever the system quality has decreased dramatically.

The calibration may imply an adjustment in the value of the parameters which determine the *system scenarios*; an increment in the *system scenario* cost when the real cost is higher than anticipated; a cost reduction in case the only situations appeared at run time related to a specific *system scenario* are the ones with a lower cost, etc. The calibration can also lead to merge *system scenarios* or add new ones to the system if their frequency is relevant enough or their absence is problematic in terms of system quality. As a side-effect of this step, the prediction and maybe the switching processes could be adapted as well.

4.3. An MP3 decoder as case study

4.3.1. Overview of the MPEG-I audio standard

Along this dissertation, we have used as case study example and main benchmark a C-implementation of an MP3 decoder. This stream-based application is based on the MPEG/audio compression algorithm, developed by the Motion Picture Experts Group (MPEG) and adopted in 1992 as ISO-standard for the high fidelity compression of digital audio. The whole standard covers not only the compression of audio (ISO 11172-3), but also the compression of video (ISO 11172-2) as well as the synchronization of both to an aggregate bit rate (ISO 11172-1).

The MPEG/audio compression standard [Pan95, MPE] is lossy, which means that after the encoding process, it is not possible to fully reconstruct the original signal. The standard is based on the psychoacoustic principles about the perceptual limitations of the human auditory system and focus most of the compression on the removal of the perceptually irrelevant parts of the audio signal, i.e. components

which are below the auditory threshold or which are masked by stronger components existing in closer frequencies. This model translates in inaudible distortions, higher compression rates and minimum file sizes.

The amount of data preserved is configurable through the compression process so an optimal balance between the file size and the quality can be achieved. The large variety of compression modes that the standard offers are summarized in four aspects:

- The audio sampling rate can be 32 kHz, 44.1 kHz or 48 kHz.
- The compressed bitstream can support one or two channels and cope with monophonic, dual-monophonic, stereo or joint-stereo modes.
- The bit rates are fixed per channel in a range from 32 to 320 kbits per second, although 128 kbps is widely considered as the minimum rate to generate an optimum quality bistream.
- The standard offers three independent layers of compression depending on the codec (encoder/decoder) complexity and the compressed audio quality required. Thus, Layer I is the simplest one, suited for bit rates above 128 kbits per second per channel, while Layer II targets bit rates around 128 kbits per second. Layer III is the most complex compression but offers the best audio quality, particularly for bit rates around 64 kbits per second per channel.

The MP3 decoder used in this dissertation is based on this third layer of the audio standard.

Generally speaking, the encoder analyses the analog signal, selects the psychoacoustic model to apply as well as the sampling rate and the bit rate to create as output

file a sequence of frames called bitstream. The general structure of an encoder is shown in Figure 4.4. Basically, the analog signal is divided into 32 sub-bands signals with the same bandwidth in the Analysis Polyphase Filter Bank kernel. These sub-bands are windowed using either long or short windows depending on the behaviour within each sub-band. If it shows a stationary behaviour, a long window is used to enhance the spectral resolution. Otherwise short windows are applied to improve the time resolution. Each of these sub-bands is subdivided into 18 spectral lines using the modified discrete cosine transform (MDCT), followed by an alias reduction step. Concurrently, Fast Fourier Transformations (FFT) are performed, followed by the Psycho Acoustic Model to decide which parts of the audio are perceptual relevant. Later, a nonlinear quantization process is applied to each sub-band signal according to the ratio provided by the psychoacoustic model. Finally, data are encoded using Huffman tables.

During the whole encoding process the sampling rate is kept constant, unlike the bit rate, which can be constant or variable. Considering a constant bit rate, the number of bits per second is always the same along the whole bitstream, with independence of the input signal complexity. However, to consider the richness of the signal, a variable bit rate can be applied, varying in this case the number of bits which constitutes each frame. This mismatch regarding the bit rate can generate large differences in workload from one frame to another.

For instance, given a constant sampling rate of 48kHz in two frames, using rates of 320 kbps and 128 kbps respectively, the difference can be close to 200 kbits.

To guarantee a constant rate of playback independent of the bit rate, the number of samples included in each one of the frames is fixed – 1152 audio samples–, as well as the number of frames per second. Thus, a second of audio contains about

38 frames considering the MPEG-I standard, which means that each frame lasts always 26 ms if we are considering an original sampling rate of 44.1 kHz. Each frame, which format can be seen in Figure 4.5 consists of four mandatory sections: header, side information, main data and ancillary data for reservoir bits. The header is 32 bits long and stores all the necessary information to process the data in the frame such as the sampling frequency, the bit rate or the channel mode, which can be single channel, dual channel –two independent mono channels–, stereo or joint stereo in case the channels share bits. In case the mode is joint stereo type, there is also information about the codification used to merge the left and right channels. The available encoding techniques are called middle/side stereo and intensity stereo.

The side information section contains information to reconstruct the original signal as accurately as possible, providing among other data the boundaries for certain data blocks, quantizer size types or the window block types used for MDCT.

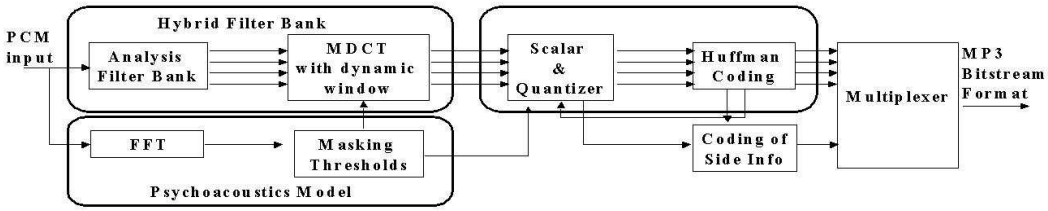


Figure 4.4: General structure of an MP3 encoder.

The main data consists of two granules of 576 samples each one, organized according to the standard. The data stored in the side information section is used to classify each granule as long, short or mixed – in case the granule contains both short and long parts – depending on the blocks built during the windowing step in the encoder. These granules represent the basic unit in the application and can have different workloads, giving rise to different patterns during the decoding process.

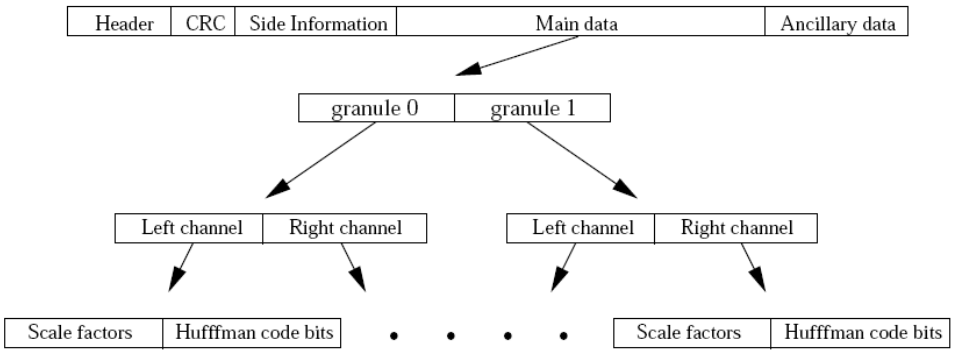


Figure 4.5: Format of an MP3 frame. Every section has a fixed length, expect for the main data, whose length depends on the bit rate. The granule is the smallest acoustic unit.

The decoder takes as input a bitstream, which manages frame by frame to construct a new PCM signal, inverting the steps done by the encoder. The general structure of the decoder is shown in Figure 4.6. The kernels included in the decoder frontend perform preprocessing steps and extract the essential information to reconstruct the signal (bit rate, sampling rate, scale factors, ...). The rest of the kernels represents the decoder backend, which includes the functions with the highest computational workload in the system such as the inversed modified discrete cosine transform (IMDCT) kernel or the syndissertation polyphase filterbank which generates the new PCM signal.

4.3.2. Applying the *system scenario* methodology to an MP3 decoder

The MPEG/audio standard defines the decoding process and the syntax of the coded bitstream, establishing the specifications about how to encode and decode the

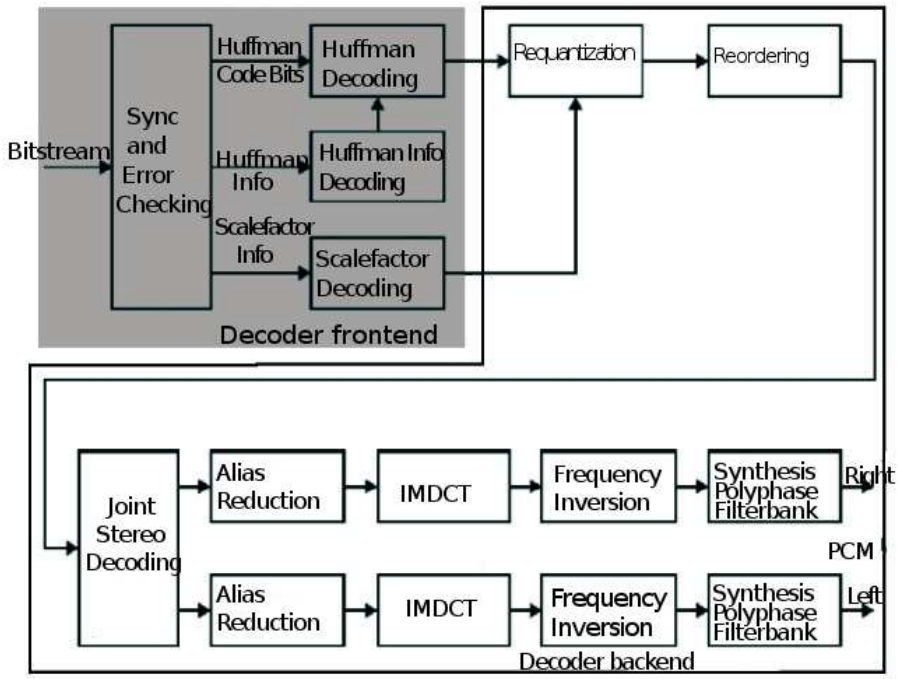


Figure 4.6: General structure of an MP3 decoder which input is a bitstream and as output generates a Linear PCM format.

information. As there is not any standard implementation, everything is allowed to enhance the compression process.

In our research we have used the MP3 decoder as main case study, choosing the C-implementation developed by Lagerström in [Lag01] for our purposes. This implementation is close to the initial specifications, without complex optimizations. Nevertheless, it serves to analyse the impact of *system scenarios* in the predictability of applications with dynamic behaviour such as the MP3 decoder. We use this implementation as baseline for our comparison purposes.

Lagerström's original implementation does not include the *system scenario* methodology. *System scenarios* were added to the decoder by Martin Palkovic [Pal07]. The discovery of *system scenarios* is based on the different types of blocks created in

the windowing step performed during the encoding process. The type of window determines on how many samples the IMDCT is performed and what is the weight of the individual samples in the decoder. The windowing step provides two kind of blocks – *Long* and *Short* – which give rises to three sort of granules. A granule can be considered as *Long* if the whole 32 sub-bands contained where encoded as long blocks by the MDCT kernel. It is considered as *Short* if they were always encoded as short blocks. In some cases, a granule can contain blocks of both types, being called a *Mixed* granule. The three existing types of granules, together with the way the channels are coded – mono, dual, stereo decoding or joint stereo decoding – give rise to different decoding modes. These modes are stated by data dependent conditions in the application code, as well as different workload and memory usage at run time. Any of the existing modes do not show the same frequency at run time. The profiling performed in [PBC+05] for MP3 shows the frequency of the most probable ones. Its results, reproduced in Table 4.1, indicate that the 99.9% on the execution time is covered by just 6 out of the 24 possible modes.

Mode	*Frequency*	*RTS*
Long block (type 0)/Joint Middle Side Stereo Decoding	90.1%	1
Long block (type 0)/Stereo Decoding	3.2%	1
Long block (type 1&3)/Joint Middle Side Stereo Decoding	3.1%	1
Long block (type 1&3)/Stereo Decoding	0.9%	1
Short block (type 2)/Joint Middle Side Stereo Decoding	1.4%	2
Short block (type 2)/Stereo Decoding	1.2%	2
Others	$<$0.1%	3

Table 4.1:

Based on similarity measurements, these six modes are clustered in two main *system scenarios*: a *Long* scenario, which groups the long blocks coded with stereo or middle side coding and represent around 98% of the decoded granules; and a

Short scenario, which groups the short blocks present in about 2% of the granules. The rest of the modes, up to complete 24 modes, are grouped in a backup scenario due to their low relevance.

The creation of *system scenarios* provides enough knowledge about the application behaviour and the kernels involved to adapt the application data flow to the set of *system scenarios*. Thus, the general flow represented in Figure 4.6 derives to Figure 4.7, where kernels are associated to their *system scenarios*. This split encourages code transformations focused on reducing the number of main memory accesses. Regarding the case study used in this research, the MP3 decoder, the changes involve global loop transformations which improve the temporal locality and eliminate large intermediate buffers due to the fusion between kernels. Although the split of the data flow initially increases the code size, the new opportunities for loop transformations generate a code size just 2% larger than the original Lagerström's version. As *system scenarios* enhance the temporal locality and reduce the auxiliary buffers, they optimize the number of memory access performed by the application. The reduction reported in [PBC+05] improves the memory accesses about 50% regarding the state-of-the-art work.

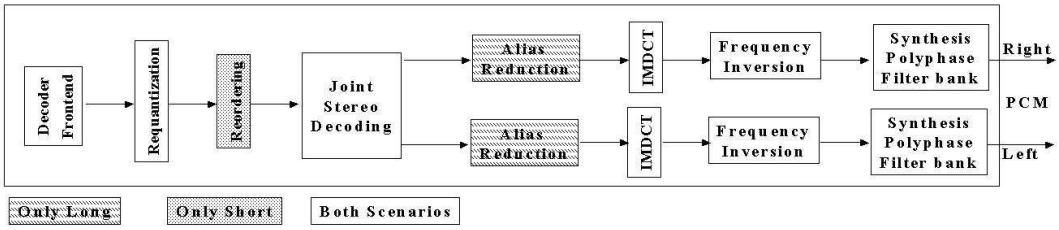


Figure 4.7: Kernels involved in each of the two main *system scenarios* discovered in the MP3 decoder.

The characteristics to differentiate among the possible scenarios are present in each frame. As we mentioned before, the header and side information sections, which are placed at the beginning of the frames, contain all the necessary information to interpret and decode correctly the rest, the audio data. As a result, the scenario predictor necessary to determine the current scenario at run time is focused on reading the information included in these two sections. The answer offered by the predictor triggers all the remaining run-time steps in the scenario methodology: switching, exploitation and calibration. These steps are dependent on the context where the scenario is detected, which includes application constraints such as time and energy, and process variation impact, specially those which are time-dependent. For instance, the knob settings applied to one particular scenario could change from one frame to another in order to mitigate the degradation existing in the system. This connection between scenarios and process variation is explained in Section 5.

4.4. Conclusions

The *system scenario* concept reduces to a large extent the application unpredictability and enables the system to accurately predict the short-term future application behaviour. For instance, in a frame-based application, this prediction could help the following frames in terms of memory accesses and execution time. Inside each pre-characterized piece of code, the behavior is deterministic and hence, its characteristics can be foreseen. This enables the application to still meet real time constraints while reducing energy consumption due to the more realistic use of worst-case margins.

5

Scenarios and Configurable memories. A methodology to work in partnership

In this chapter we explore the potential benefits of combining scenarios and configurable memories to deal with the uncertainty caused by application dynamism and platform variations in a coordinated way. Section 5.1 describes the baseline framework that has been used in this dissertation to design custom memory architectures, along with some details about the output of this framework for an MP3 decoder, our driver application. The algorithms used to implement our compensation mechanisms are described in detail in Section 5.2. The chapter concludes analyzing the additional benefits that can be achieved by increasing the number of operating modes in the memory modules that integrate the custom memory architecture.

5.1. Designing domain-specific memory architectures

Improving the energy efficiency of computer systems is currently one of the most important challenges in computing. In embedded systems, this problem is exacerbated since they impose even more demanding power and energy efficiency constraints than general-purpose systems. Active cooling mechanisms and packages with superior thermal characteristics, which are used nowadays in desktops and servers, cannot be integrated on most embedded systems since the system costs would increase beyond selling price targets. Furthermore, in mobile devices, energy efficiency is a paramount design constraint because devices rely on batteries for energy.

How to improve energy efficiency? In domains such as multimedia and wireless communications, embedded applications are clearly data-dominated. On-chip memories contribute to a large fraction of the area and storage and data transfers dominate energy consumption. This explains the increasing relevance that the memory architecture has on these systems, as well as highlights the significant contribution that memory optimizations may have in energy and area efficiency.

Based on these remarks, IMEC developed along the last decade a comprehensive methodology for the optimization of memory architectures for embedded systems. The methodology, called DTSE (*Data Transfer and Storage Exploration*)[CdGS98] is supported by ATOMIUM[ATO], the tool suite used as a baseline in this dissertation to design custom memory architectures. Next, we give a brief overview of ATOMIUM, presenting the custom memory architecture proposed by this tool for an MP3 decoder.

5.1.1. ATOMIUM

ATOMIUM, the acronym of *A Toolbox for Optimizing Memory I/O Using geometrical Models*, is a tool suite that addresses data transfer and storage aspects for data-intensive multimedia applications. It is the link between the specifications for a prospective system and their implementation as an optimized embedded system, since its outputs can be used as inputs for other compilation and high-level architecture syndissertation tools.

The DTSE methodology is basically focused on four aspects:

- Global data flow transformations
- Loop transformations and control flow optimization
- Memory hierarchy: which involves data reuse decisions, allocation or assignment, among others
- In-place optimization

Each of these features has been developed and included in the ATOMIUM tool suite as part of different cooperating modules. The ATOMIUM's flow and its correspondence with DTSE is illustrated graphically in Figure 5.1. These modules are the following:

- ATOMIUM/Analysis: Memory access count analysis

  Based on the instrumentation of the target application and profiling analysis, this module provides a quickly identification of memory-related bottlenecks and hotspots. This includes information about which are the most accessed structures and its location in the code. This information is used to carry out

transformations in the data flow and control flow to optimize the code in order to successfully map it onto the platform in subsequent stages.

- ATOMIUM/MH: Memory Hierarchy

  ATOMIUM/MH optimizes the memory hierarchy in terms of energy cost and cycle budget. This stage is focused on data reuse and the placement of the memory structures along the different levels of the hierarchy. Its inputs are the description of the available memories, which are provided by the designers, as well as the profiling information obtained by the previous ATOMIUM/Analysis module.

- ATOMIUM/MA: Memory Architect

  ATOMIUM/MA takes into account relevant real-time cost factors such as energy, performance or area. It explores the effects that the timing constraints have on the memory architecture, showing the most efficient ones in terms of energy and area which meet the requirements. This module carries out two of the most significant step in the DTSE methodology, the storage bandwidth optimization (SBO) and the memory allocation and assignment (MAA).

  It takes as inputs the application timing constraints, together with the profiling information from the ATOMIUM/Analysis module and the memory libraries supplied by the designers. Overall, the output of this tool is a custom memory architecture, which includes the corresponding memory allocation and data assignment.

- ATOMIUM/MC: Memory Compaction

Based on data lifetime information, the aim of this step is to further reduce the energy consumption and optimize the reuse of memory locations.

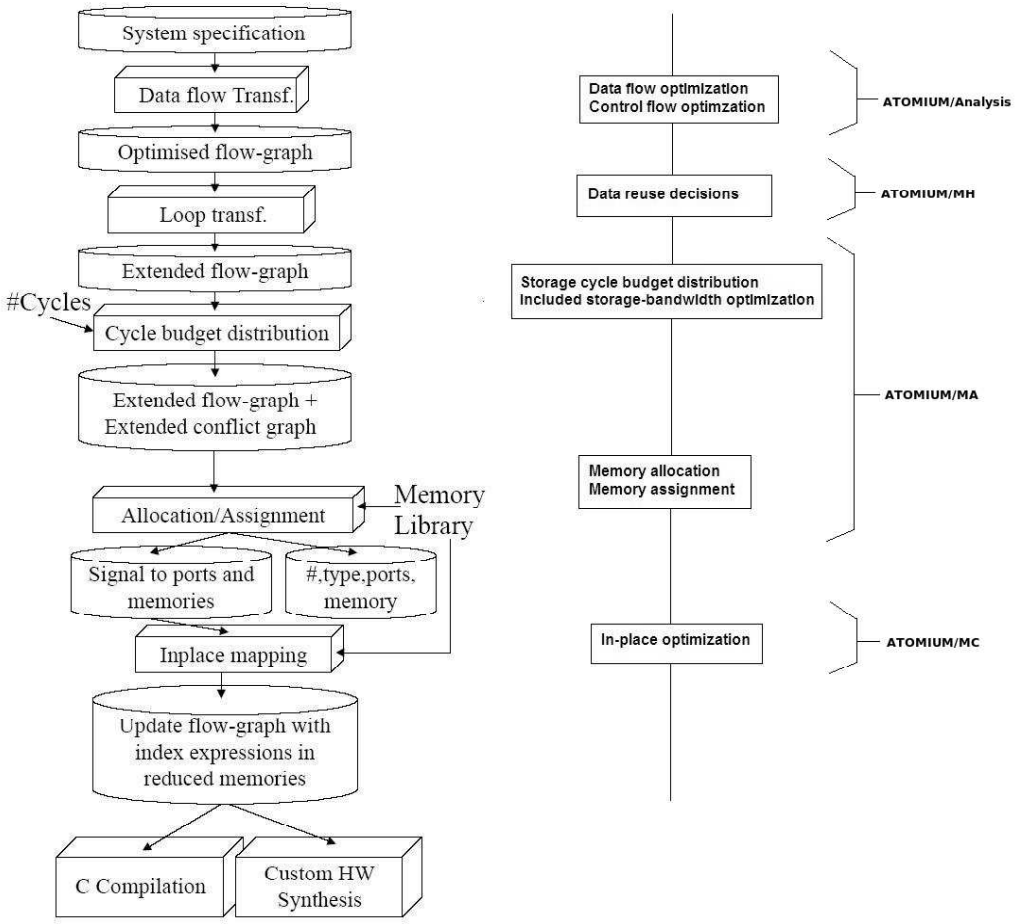


Figure 5.1: General overview of the ATOMIUM toolset.

5.1.2. Custom memory architecture for an MP3 decoder

We have used the ATOMIUM tool suite to build the custom memory architecture of an MP3 decoder. As inputs, we have use two different high-level descriptions of this application. Our baseline implementation is based on the MP3 algorithm de-

veloped by Lagerström's [Lag01]. We have also explored a scenario-aware version based on a code developed by Martin Palkovic [Pal07]. Both of them have been profiled and instrumented by ATOMIUM for comparison purposes. The memory libraries required by some of the ATOMIUM modules are based on CACTI models and have been provided by IMEC.

The custom memory organization generated by ATOMIUM for the baseline code, denoted as *Base* in this chapter, consists of eight memory modules, whereas the scenario-aware version (abbreviated as *SA*) consists of ten modules. The *Long* scenario fully exploits the architecture, while the *Short* scenario only uses seven of them as it operates on a smaller set of data.

Table 5.1 shows the memory architecture for both versions, breaking down the number and size of each of the memory modules involved. The difference in the number of modules between both implementations is due to the extra arrays added to the *SA* version to implement the scenarios.

Memory size	Base	Scenarios (*SA*)
1 KB	1	0
4KB	2	1
8KB	2	2
16KB	2	4
32KB	1	3

Table 5.1: Custom memory architectures used for each version of the MP3 Decoder.

ATOMIUM also provides an estimation about the average number of accesses performed to the memory modules and the cycles estimated for the application execution. As our driver application is frame-based, this information is reported at frame level, i.e. ATOMIUM estimates the average number of memory accesses and cycles expected per frame.

5.2. Exploratory compensation methodology

This section outlines the exploratory methodology that serves us to illustrate how scenarios and configurable memories can be combined to tackle the uncertainty coming from application dynamism and process variability. The methodology uses a mixed design-time/runtime approach that is graphically illustrated in Figure 5.2. As shown, it spans different steps along the platform lifetime.

At design time the application is thoroughly profiled with two main objectives: to identify the most likely system scenarios of the application and to determine a custom memory architecture and a subsequent energy-aware data assignment. For every scenario, we collect at this stage the number of accesses to each memory module, together with an estimation of the associated time and energy cost. For this estimation, the methodology assumes at design time that memory modules are not configurable and work in the nominal point of their most energy efficient mode.

Figure 5.3 illustrates the output of this first stage using as example a fictitious application. The profiling of its task graph allows us to extract two system scenarios. This same profiling also reveals that according to the memory library supplied by the designer, the optimal memory architecture consists of three memory modules. The corresponding data assignment is also shown in the figure. The three memory modules in this example have two operating modes but as mentioned above, this information is not used at design time.

Once the platform has been manufactured following the proposed design, it is necessary to perform a measurement/calibration phase at setup time. Its aim is to get information about the actual performance of the memory modules, i.e. to assess the impact that platform variations caused by the fabrication process have had on the

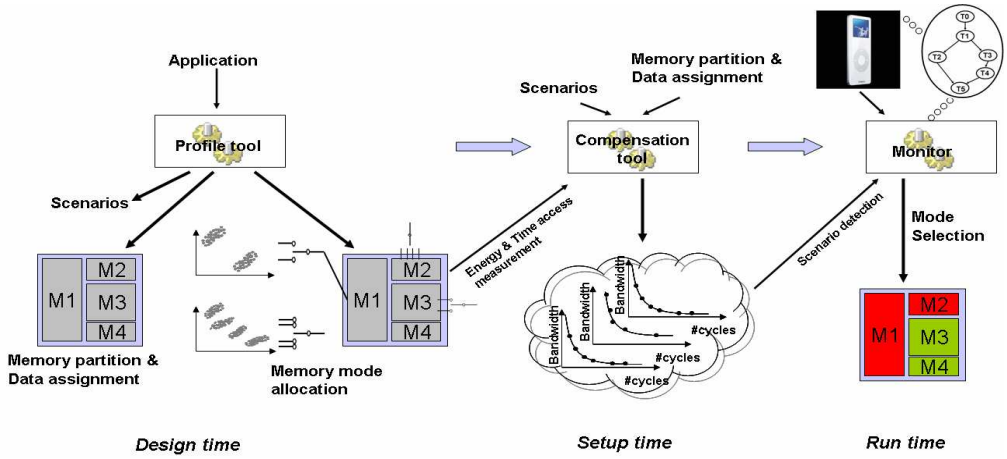
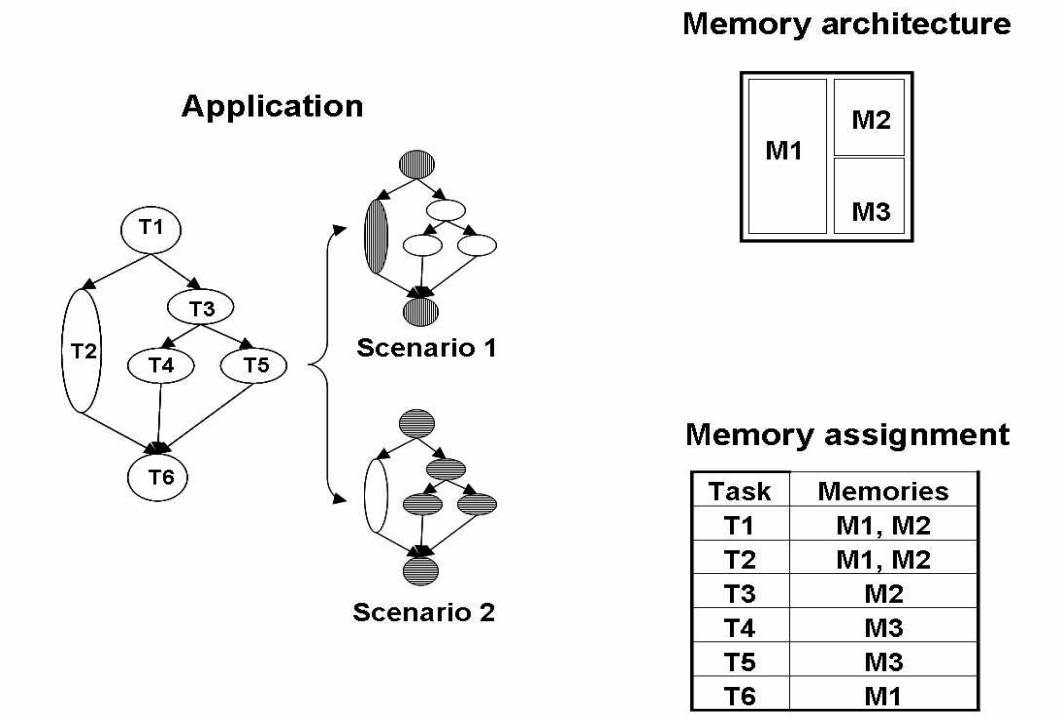


Figure 5.2: System methodology.

memory system. At this point, the different operating points existing in each memory module are considered. For every memory module in the system, and for each of their operating modes, the actual energy and delay per access is quantified, which may differ substantially from the nominal parameters expected at design time. This information is combined with all the existing knowledge extracted at design time – system scenarios, memory organization and data assignment, application time constraints, ... – to generate sets of optimal configurations for the different memory modules that integrate the system. Each set of these configurations is a trade-off between energy and performance, in such a way that the configuration that is able to meet a strict deadline consumes more energy than those configurations that only meet less demanding time constraints. A set of configurations which follow this characteristic constitutes a Pareto curve, term which will be used from now on to describe these sets. Thus, each scenario is described by its specific Pareto curve, as shown in Figure 5.4 for the simplified example introduced above.



Task	Memories
T1	M1, M2
T2	M1, M2
T3	M2
T4	M3
T5	M3
T6	M1

Figure 5.3: Outputs generated at design time.

Finally, at runtime, the application is monitored by means of software controllers included in the system architecture, to detect the current scenario, as well as the timing constraints that must be fulfilled. Known the scenario, its associated Pareto curve is consulted and the memory configuration which best fits the current requirements is selected. Once the configuration has been chosen, the memory is adapted properly to continue the execution. Given that all the necessary information is pre-calculated during the setup step, the selection of the appropriated configuration for each memory is performed very quickly.

The precise moment when the configuration actually happens depends on the application we are working on. For instance, in a frame-based application this process could be done at the beginning of the frames, although not necessarily on ev-

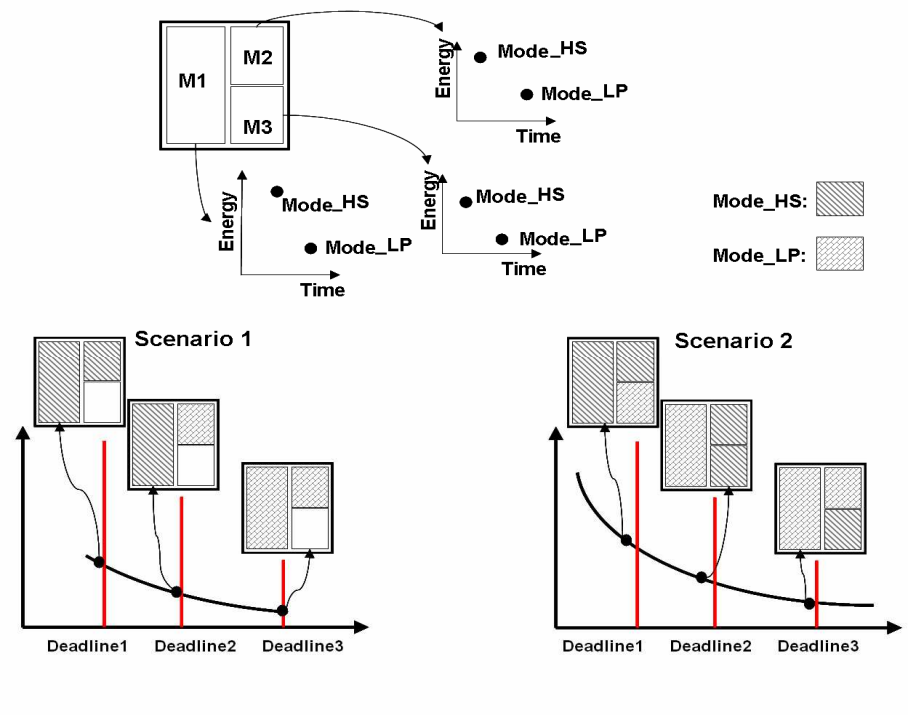


Figure 5.4: A Pareto curve per scenario is the output generated at setup time by means of compensation techniques.

ery frame. An immediate switching can be rejected, among other reasons, by an excessive cost associated to the switching between two particular scenarios or a demanding run-time situation existing at the moment of the switching.

Overall, the heaviest part of this basic methodology is carried out at setup time, during the system calibration. Algorithm 5.1 summarizes the methodology for frame-based applications.

5.2.1. Compensation techniques

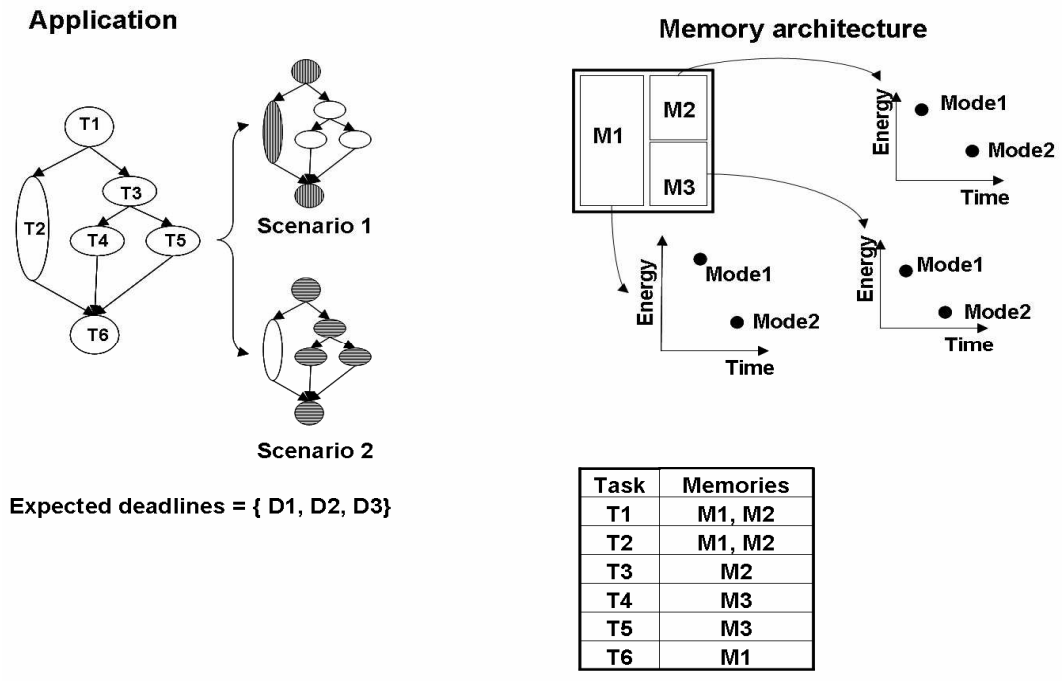
The calibration process carried out at setup time accomplishes the measurement phase of the memory modules and the generation of the Pareto curves for each of the scenarios obtained.

Algorithm 5.1 Exploratory compensation methodology for a frame-based application

```
/*design time*/
 profiling_and_scenarios_detection();
/*system set-up*/
measurement();
compensation_technique();
/* execution_and_configuration(); */
for each frame f
    select_scenario(f);
    find_suitable_configuration_point(f, time_constraint);
    apply_configuration(f);
    execute(f);
```

Figure 5.5 summarizes the information used to generate such curves: the application task graph and its partitioning into system scenarios, the expected deadlines and a detailed description of the actual memory architecture and the data assignment.

Firstly, as each of the memory configurations which constitute the curves is attached to a specific time constraint, a required input is the set of expected deadlines considered at design time. Since we are considering frame-based applications, these time constraints can be expressed as different frame rates. Secondly, it is also necessary to take into account the energy/delay information obtained during the measurement phase to characterize the memory architecture under variation. Finally, the application task graph split in scenarios is also required. The compensation mechanisms manage for each scenario its specific task graph, which contains information about memory accesses, cycles, the memory allocation and data assignment.



Task	Memories
T1	M1, M2
T2	M1, M2
T3	M2
T4	M3
T5	M3
T6	M1

Figure 5.5: Summary of the inputs required by the compensation techniques proposed in this chapter.

The first one of the compensation mechanisms developed in this research works at frame level. This mechanism assumes that the processor frequency, determined by the current deadline, remains constant during the whole frame, setting up the way the memory modes need to be configured to meet that frequency, i.e. memories need to work at least at the same frequency as the processor. This configuration is set during the whole frame. From now on, we also refer to this compensation mechanism based on memory modes at frame level as *frame-level compensation* (*FLC*).

The second mechanism, denoted as *task-level compensation TLC*, works at two different levels. It works at frame level to meet the same deadlines than the previous mechanism, but also works at task level establishing a different memory configuration and processor frequency for each of the tasks involved in the application.

5.2.1.1. Compensation at frame level

This technique adapts the memory system to the processor frequency that is established for the current frame. This adaptation is carried out selecting among the different operating points existing in each memory module.

The starting point of this technique is the deadline which the processor and the memory system have to meet at run time. From this deadline and the number of cycles estimated for the current frame, it is derived the frequency that will be used for the system. Once the compensation mechanism establishes the frequency, the next step is to determine the mode in which each memory module has to be configured. This step is carried out for every memory comparing the clock cycle, which is derived from the frequency, with the access time of each of its existing modes. Thus, the clock cycle establishes the maximum delay admissible for the memory modules in order to consider a mode as suitable. It could happen for a memory module that more than one of its modes have an access time lower than the clock cycle, in this case, it is selected the mode which consumes less energy.

The association of every memory to a single mode constitutes the output of this technique, i.e. the memory configuration which ensures the fulfilment of a deadline under process variation. This is the simplest way to compensate variability effects and make the memory system operative.

A general overview of this compensation mechanism, based on a greedy approach, is sketched in Algorithm 5.2. It takes as initial configuration the slowest mode with the lowest energy existing in each memory. In each step, the algorithm select the slowest memory to be speeded up to its next slowest mode. This process continues until one of the following situations forces the end of the algorithm:

- The access time of the slowest memory is already lower than the required cycle time. In this case, the final configuration for each memory allows us to meet the time constraint.

- The delay is longer than the required cycle time but the slowest memory cannot be speeded up anymore. There is not any configuration which allows us to meet the time constraint.

This compensation technique has to be performed for each of the time constraints expected at run time and for each of the scenarios considered at design time. The final set of configurations constitutes the Pareto curves that are consulted at run time to adapt the system to a new scenario and/or a new time constraint.

Algorithm 5.2 FLC: Greedy implementation to configure the memory architecture

```
input: scenario
output: Pareto_curve
initialize_all_memories(low_mode);
for every clock_cycle_constraint cycle
    for every memory_module mem
        for every memory_mode mode  /*sorted from
            lower to higher access time*/
            if (cycle < access_delay(mem, mode))
               switch_mode(mem, mode);
               break;
```

5.2.1.2. Compensation at task level (TLC)

This scheme exploits mode adjustment at a finer granularity than the previous one, and takes advantage of the multimedia context we are working on, where applications are usually characterized in terms of individual tasks. The overall idea is

that each task, and the memories involved in them, can be configured independently of other tasks. According to this, it is not mandatory for all the tasks to run at the same clock frequency. This means that, in contrast to the previous technique, the frequency can change from one task to another along the same frame.

For a better illustration of this technique we consider as example an application modeled by two tasks, denoted as *Task A* and *Task B* in Figure 5.6. The memory architecture consists of two memory modules, denoted as *Module A* and *Module B* respectively. For the sake of clarity, each of these memory modules stores the data structures of only one of the tasks, i.e. *Module A* stores data from *Task A* while data from *Task B* stays in *Module B.*

Applying to this example the previous technique, *FLC*, it could happen that the access time for the lowest mode in each memory module is larger than the clock cycle required to meet the final deadline (left-hand side in Figure 5.6). Under these circumstances, the only solution that the *FLC* compensation mechanism could offer to keep the memory system operative would be to switch both memories to their high mode, with the consequent increment in energy.

However, it is possible to meet the application timing requirements with a lower impact on the energy cost. This can be achieved considering the clock frequency as a new knob to be used together with the memory modes. In this way, the different tasks can operate at different clock frequencies, allowing new chances to make the memory system operative.

Coming back to the example, if we take into account the application deadline, without derived a global cycle time for the whole frame, different feasible solutions can be achieved. From the profiling step carried out at design time it is possible to know the number of cycles estimated for each task in the application. This infor-

mation, together with the access time of each mode for the memories involved in the task, is enough to establish deadlines at task level. Considering the deadlines of all the tasks we can know if the frame deadline can be met or not. For each of the tasks, its own deadline implies a way to configure the memories inside the task.

As is shown in the right-hand side in Figure 5.6, we can configure *Module B* to use its high speed mode, generating a time slack between the clock cycle initially estimated for the whole frame and the access time selected at *Module B*. The reason to use the high speed mode of *Module B* instead of its counterpart in *Module A* is its energy cost, as the energy penalty due to using the high speed mode is larger for *Module A*. The time slack created can be used to reduce the execution time at *Task B* speeding up the clock frequency for this task. The time saved in *Task B* can be used to slow down the clock frequency at *Task A*, enabling *Module A* to operate in its low power mode. Thus, the output generated by *TLC* for this example sets the low power mode for *Module A*, being executed *Task A* to such frequency, while *Module B* and *Task B* are managed by its high speed mode.

The final memory configuration illustrated in the right-hand side in Figure 5.6 is not the only way to configure the tasks. Considering the two-task application, two-mode configurable memories and the memory allocation and data assignment already described, Figure 5.7 shows all the options that the *TLC* compensation technique would have to consider in order to find the configuration for each task which allows to meet the frame deadline with the lowest energy cost.

Projecting this example to a more complex situation it is clear that the number of configurations to check increases as either the memory modes, the memory modules or the tasks increase. As the options increase, finding the most efficient configuration and frequency for each task is not obvious. Algorithm 5.3 illustrates

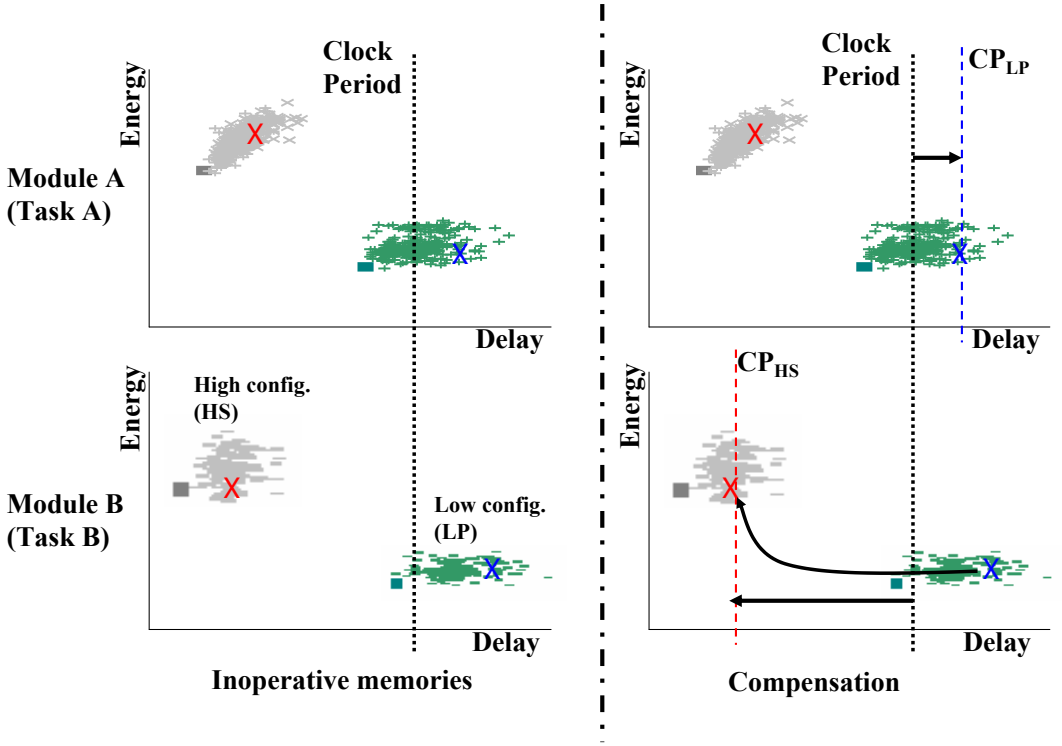


Figure 5.6: Task-level compensation based on mode and frequency. General overview.

the implementation of this technique assuming an exhaustive search[1]. The algorithm considers every feasible configuration for each of the tasks, computing the energy and delay for each one. Among all the existing configurations, the algorithm selects one per task so that the global deadline is met with the lowest energy cost. This global configuration will be Pareto optimum regarding configurations for other deadlines, indicating the mode in which each memory in each task has to be configured. The frequency for each task is determined by the delay of the slowest memory used in the task.

[1]This exhaustive algorithm becomes a branch and bound algorithm in the next chapter by means of heuristics to reduce the search space. For simplicity reasons, as this is a chapter to motivate our methodology, the search algorithm is presented as exhaustive.

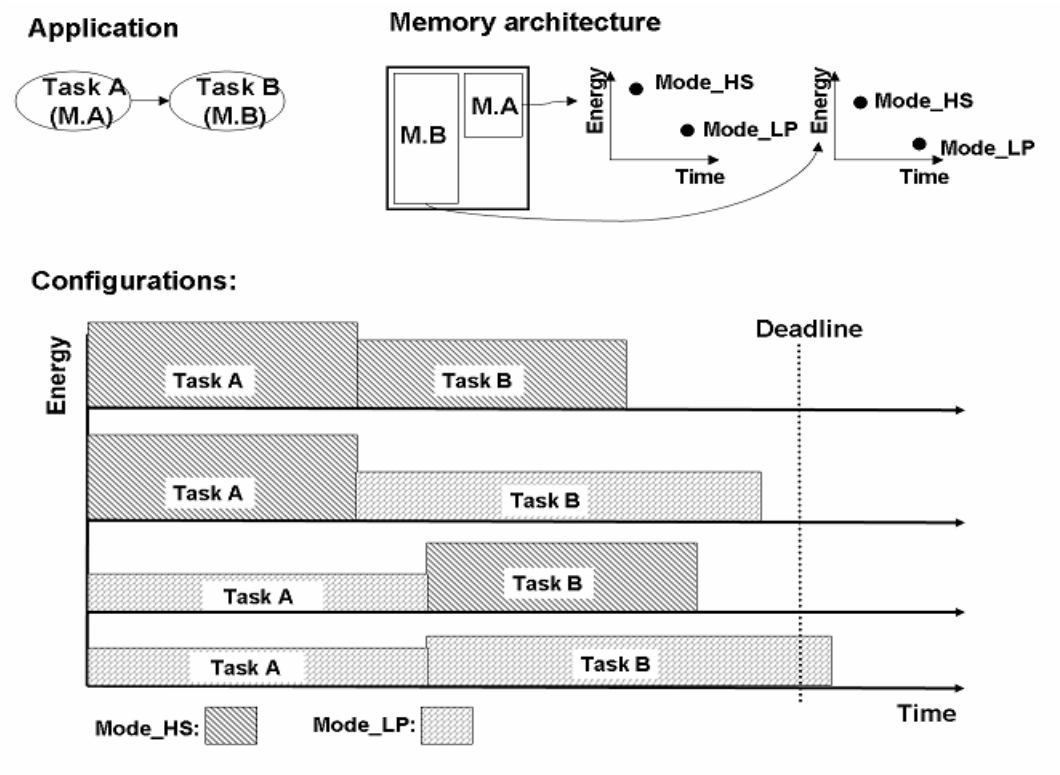


Figure 5.7: Memory configurations included in the search space for our example. The complexity increases with the number of tasks, memories, and memory modes.

In summary, taking as point of reference a global deadline and different frequencies per task, it is possible: (1) a better timing distribution, (2) the fulfilment of the time constraint and (3) the reduction of the total energy consumption. Traditional *Dynamic Voltage Scaling* (DVS) or *Dynamic Frequency Scaling* (DFS) techniques may be used to change the frequency of the system.

In this mechanism, as well as in *FLC*, the time and energy savings are only referred to the memory system. Along this dissertation we do not include information or results about savings at the whole core.

Algorithm 5.3 TLC: Exhaustive implementation to configure each of the memories in each of the tasks involved in the application

```
for each task t
    for each configuration c
        task[t]=c;
        if (all_tasks_are_configurated())
           calculate_energy_delay();
           if (delay < deadline)
              store_possible_solution();
```

5.3. Experimental results

To simulate the variability impact in our memory architecture, we have used information about the variability effects on the energy and delay of 1 KB SRAM memories. The characterization of a two-mode configurable memory under variability, provided by IMEC, was achieved by performing Monte Carlo transistor level simulations using a 65nm BSIM model. In those simulations, V_{th} and β parameters were altered following normal distributions to inject variation at transistor level. Further information about this process can be found at Section 2.3 and [HCM+05].

Taking as reference the energy and delay values on a 1 KB configurable memory under variation, we have simulated the variability impact on the two memory architectures we are working on (Table 5.1 in Section 5.1) by extrapolation of the 1 KB SRAM memory. Every memory existing in the system has been simulated as a two-mode memory.

To illustrate the effects of our compensation techniques on the application dynamism, we compare the two versions developed for the MP3 Decoder (*BASE* and Scenarios) and their associated memory architectures. As we mentioned in Section

5.1, the version called *BASE* represents an implementation of the application where the scenario methodology has not been taken into account. On the other hand, the version denoted as Scenarios or *SA* implements the scenario methodology, giving rise to two main scenarios (*Long* and *Short*).

Regarding variability on the memory architecture, we have considered three different situations to compare how our methodology deals with variability depending on the approach selected. These three situations are related to the way variability is considered at design time. These are the three situations considered:

- Nominal case

  For comparison purposes, this case represents the ideal situation where memories do not suffer from any source of variation and operating points for each of the two-mode configurable memories are set at design time.

- Worst case

  In this case, the design process is aware of the variability impact on memories. The uncertainty generated is tackled by means of conservative design margins applied to the energy and delay of the memories. Design margins are applied to maximize functional yield at the expense of additional delay and energy consumption. We simulate this case characterizing the energy and access time with the highest values expected after fabrication for each memory. As in the nominal case, the memory architecture consists of two-mode configurable memories.

- Compensated case

This case represents our methodology. The design is aware of the variability impact but instead of applying worst-case margins, our methodology copes with this variability at setup time.

To study the effects of scenarios and variability compensation, separately and in combination, these three cases are applied to both implementations of MP3. Figure 5.8 summarizes the experiments carried out in this chapter: (1) represents the traditional and conservative approach regarding variability and dynamism, scenarios are not applied and worst-case margins are used to deal with variation. (2) represents a conservative approach regarding variation, adding scenarios to cope with dynamism. (3) deals with variation by means of compensation, being conservative with dynamism (no scenarios). (4) and (5) are unaware of variability, showing ideal situations. Finally, (6) represents the methodology introduced in this chapter, in which scenarios and variability compensation at setup time are considered in a coordinated way.

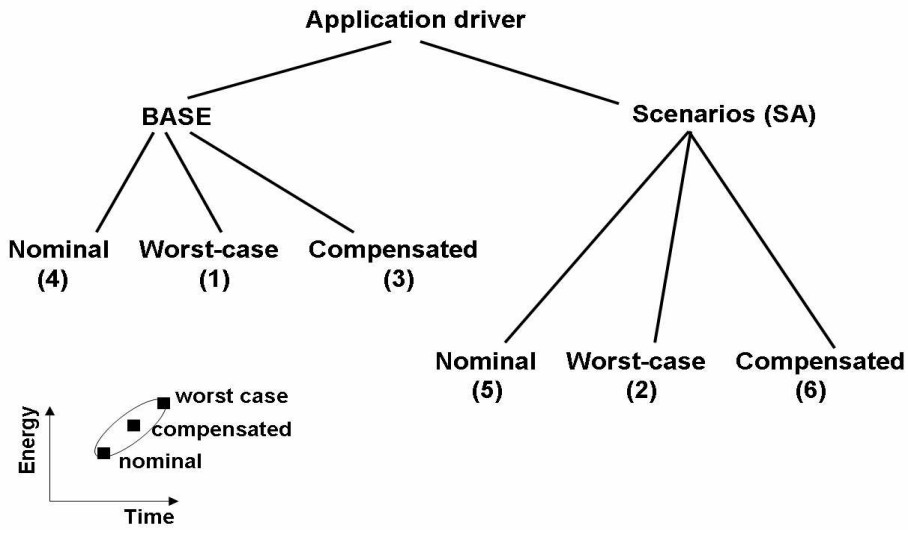


Figure 5.8: Cases taken into account in this chapter.

The compensation mechanisms (*FLC* anf *TLC*) are applied at setup time to the three cases presented: nominal, worst-case and compensated. Regarding the nominal and worst-case approaches, the compensation mechanisms could also be applied at design time in these cases since the current impact of variability is not known and the energy/delay values considered are already fixed from design.

Although the MP3 Decoder uses a fixed time constraint along a regular execution, we have simulated variations in its timing requirements. We have considered a wide range of timing requirements – cycle times or deadlines depending on the compensation technique applied – to explore how the compensation system works under a dynamic environment.

In our framework, a deadline will be the maximum time allowed to decode a frame, and determines the operating clock-frequency of the system. The range of explored deadlines forces clock periods close to the access time of both the fastest and slowest memory modules. This allows us to check how multi-mode memories can be adapted to any constraint, giving flexibility to the system.

FLC results

The execution profile of the MP3 Decoder under this technique is shown in Figure 5.9. At the beginning of a new frame, the system determines the scenario which better fits the current characteristics. If the scenario or the time constraint changes from the previous one, the system selects the configuration which will be applied to the memories of the granule which is starting to be decoded. This configuration has to be chosen among the ones which constitute the Pareto curve, built at setup for its scenario.

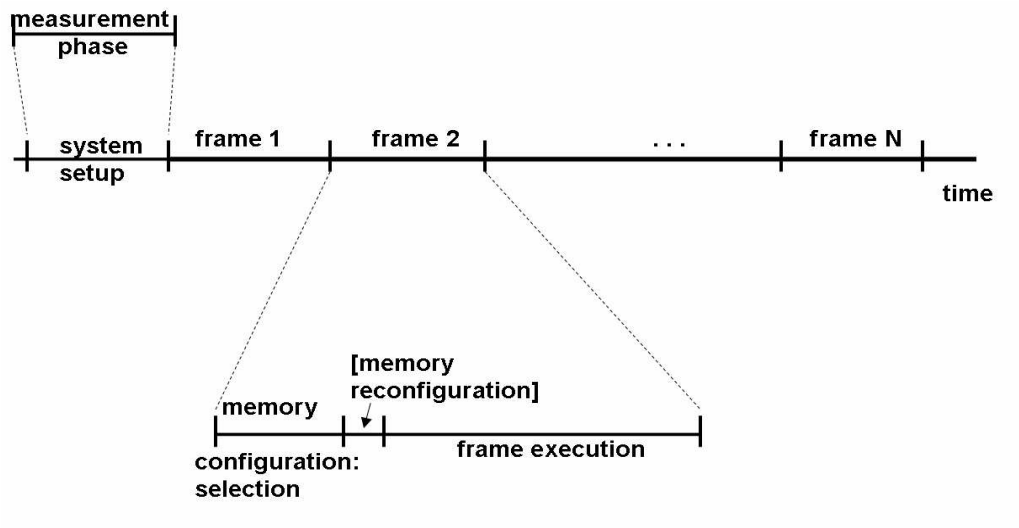


Figure 5.9: MP3 Decoder execution profile under the variability compensation based on mode.

Figure 5.10 illustrates the energy consumption (y-axis) in the full memory organization and the execution time per frame (x-axis) associated with this compensation system for different clock cycle constraints. The curve labeled as *compensated* in the Scenario version (*SA*) refers to this approach. Its counterpart curve in the *BASE* version corresponds to the result that could be obtained by assuming a scenario unaware implementation and a configurable memory organization. The nominal and worst-case curves are deterministic, since each memory module assumes single nominal and worst-case operating points under variability for each of the two implemented modes. The compensated curves, however, are stochastic. Each of the points in the compensated curves is actually a cloud of points, due to the impact of variability on the memory-level energy and timing. In Figure 5.10 it is only illustrated the mean point of each configuration cloud.

Applying scenarios leads to a reduction in the total number of accesses. This property has a clear consequence in the cost as shown in Figure 5.10. There is

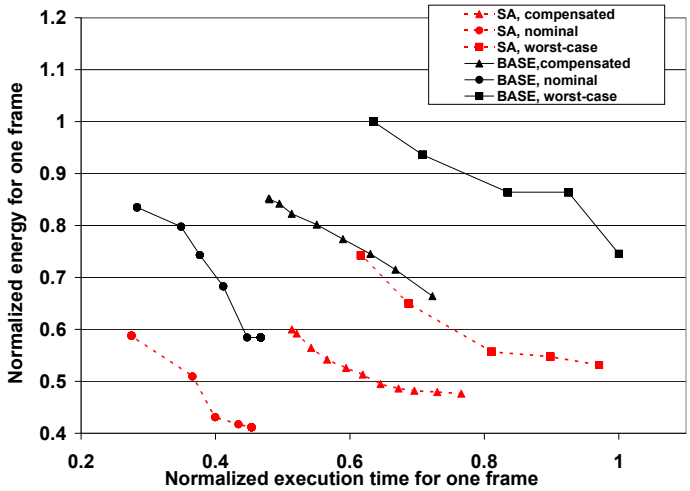


Figure 5.10: The original application version (BASE) is compared with the version split in scenarios (*SA*). Both versions have been simulated under the effect of process variability using the memory mode adjustment approach.

a significant reduction in energy just by considering the scenario concept in the implementation of the application. Comparing the worst-case curve in the *BASE* version with its counterpart in *Scenario* it is clear the significant reduction in energy exhibited. The reduction fluctuates between 20% and 30%. A similar behaviour is shown in the nominal and *compensated* cases.

Another significant result from Figure 5.10 is related to the application of compensation mechanisms at setup time. Being in the *BASE* version, the addition of the compensation process at setup time (*compensated* curve) also reduces significantly the energy cost regarding its worst-case counterpart.

Finally, combining scenarios with our variability mitigation techniques improves not only the energy but also the performance. The combination of both techniques in a single methodology (*compensated* curve in the Scenario version) shows large energy savings regarding the traditional conservative approaches represented by the worst-case curve in the *BASE* version. The fastest *SA compensated* configuration

burns 40% less energy while performs 25% faster than the *BASE*worst-case solution.

Figure 5.11 summarizes the results shown in Figure 5.10, depicting the average energy consumed for different clock cycles. The breakdown of the energy gains due to the application and the platform dynamism compensation is illustrated. As in Figure 5.10, the difference between *BASE* and *SA* version in their worst-case bars shows the energy savings due to scenarios. The energy impact due to the variability compensation is seen in the *BASE* or the *SA* versions as the difference between the worst-case and the *compensated* bars. Our proposal, *SA* compensated, provides the smallest energy overheads over the ideal but practically infeasible to implement nominal case, while still guarantees the application timing constraints. It is clear that using the worst-case assumptions incurs significant energy overhead (*SA* worst-case). The combination of the two optimization techniques can provide energy gains of up to 50% compared to the conventional approaches.

As the cycle clock becomes larger, i.e. the deadline becomes more lenient, the energy savings decrease. Despite this, our compensation methodology still improves the conservative approaches.

The energy gains illustrated in Figure 5.11 can only be achieved by solutions that employ run-time components. Pure design time solutions, like Statistical Static Timing Analysis for the platform side and intelligent workload prediction techniques for the application side, can only accurately estimate the bounds of the dynamic behaviour. By estimating the maximum requirements on bandwidth or execution time for instance, they can aid the designer to avoid redundant design margins. But they cannot exploit the dynamic properties of the platform and application behaviour. The proposed approach can actually monitor the working situation of the

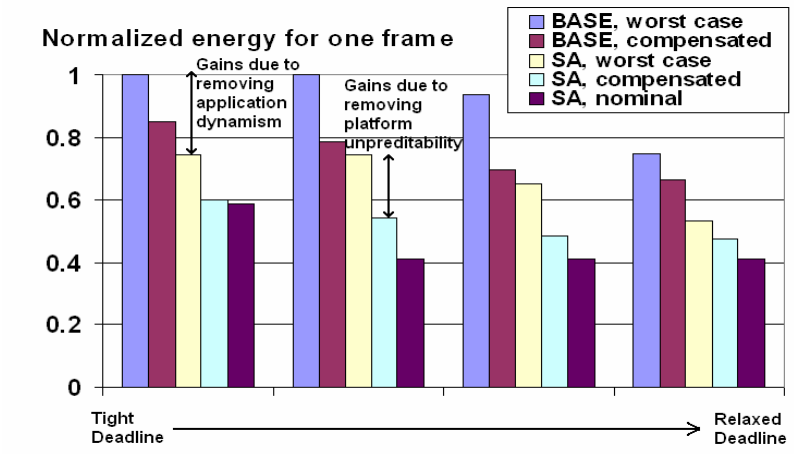


Figure 5.11: Compensation technique *FLC*: Energy consumed for different timing requirements by each version of MP3 Decoder (*BASE*version and *SA* version). The nominal bar is the theoretical minimum assuming no process variability exists.

platform and the application at regular time intervals and make the necessary modifications at run time to make sure that timing constraints are met without excessive energy overheads. Instead of just exploiting the maximum workload information for example, it configures the application and the platform to follow the specific requirements for the next short time interval. This finer grain configuration capability is the enabler for the presented energy gains.

TLC results

The execution profile of the MP3 Decoder under this technique is shown in Figure 5.12. This compensation technique requires the split of the target application in a set of tasks in order to provide to each one its own memory configuration and frequency. The division is performed for both versions of the MP3 Decoder – *BASE* and Scenarios – following the modular structure shown in Figure 4.7 for the driver

application. Regarding the Scenario version, the division has been carried out for each of the scenarios.

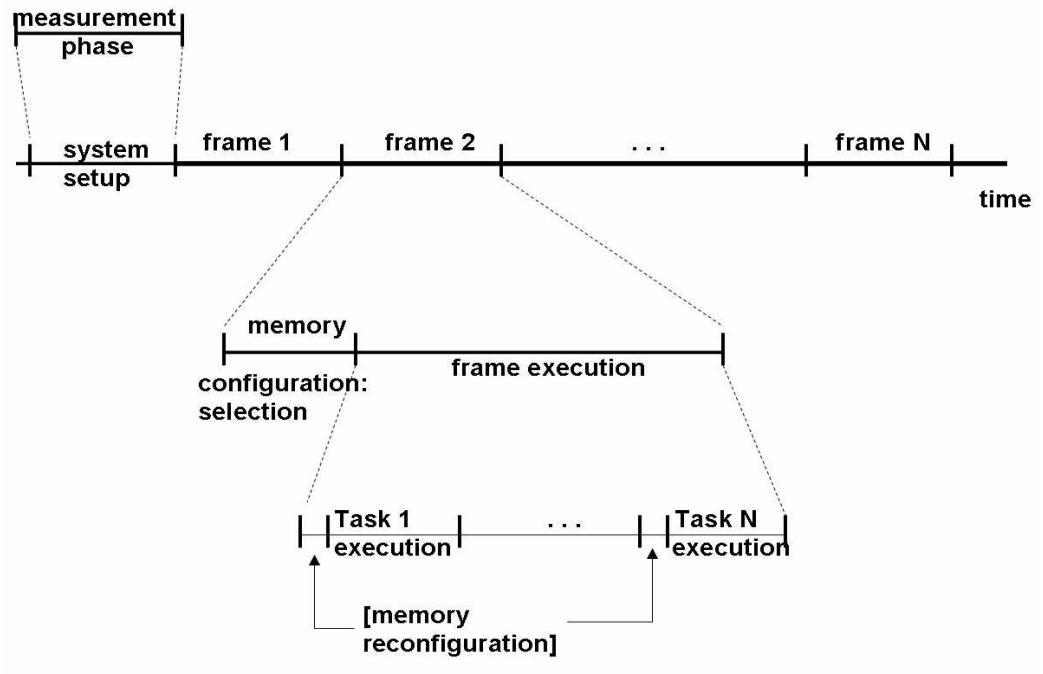


Figure 5.12: MP3 Decoder execution profile under the task-level variability compensation based on mode and frequency.

Using the profiling information provided by ATOMIUM for the applications, the split in tasks carried out in this dissertation for each of the scenarios and the *BASE* version is shown in Table 5.2. Since ATOMIUM provides an estimation about the number of cycles required for an entire frame and for each task, the weight of each of the tasks in the application is also represented. The first two columns corresponds to the main scenarios existing in the *SA* version of our driver application. The third column shows the task split carried out in the *BASE* version.

This method is able to further reduce the energy consumption by exploiting slacks in the execution time, which comes closer to the target deadline. Regarding energy, the algorithm which supports the *TLC* technique allow higher energy

Task division		
Long Scenario	*Short* Scenario	*Base* version
Requantize (4.97%)	*Requantize* (3.26%)	*Requantize* (3.07%)
Stereo (1.91%)	*Reordering* (2.05%)	*Reorder* (0.032%)
Antialias (3.06%)	*Stereo* (0.87%)	*Stereo* (0.54%)
IMDCT (15.32%)	*IMDCT* (14.3%)	*Antialias* (1.50%)
Inversion (0.76%)	*Inversion* (0.20%)	*Hybrid_Syndissertation* (17.34%)
Polyphase_A1 (5.73%)	*Polyphase* (79.3%)	*Frequency_Inversion* (0.18%)
Polyphase_B1 (5.73%)		*Subband_Syndissertation* (77.3%)
Polyphase_C1 (12.9%)		
Polyphase_D1 (12.6%)		
Polyphase_A2 (5.73%)		
Polyphase_B2 (5.73%)		
Polyphase_C2 (12.9%)		
Polyphase_D2 (12.6%)		

Table 5.2: Task division and its weight regarding in the whole frame execution.

improvements even assuming worst-case approaches. Despite this, energy savings assuming compensated configurable memories are significant. Figure 5.13 illustrates the energy reductions achieved by *TLC* when applied to the *SA* version of our driver application. The graph shows energy savings up to 20% between the worst-case approach, where variability is not compensated in the configurable memories, and the compensated case. The scenario version for MP3 and the memory architecture based on two-mode configurable memories configure the system simulated for the three cases presented in this figure.

Regarding performance improvements, both compensation techniques – *FLC* and *TLC* – are compared in Figure 5.14. It graphically represents the energy and time cost of the configurations generated for different deadlines by each technique. The framework for both techniques is the same: the scenario version for MP3 and the memory architecture based on two-mode configurable memories.

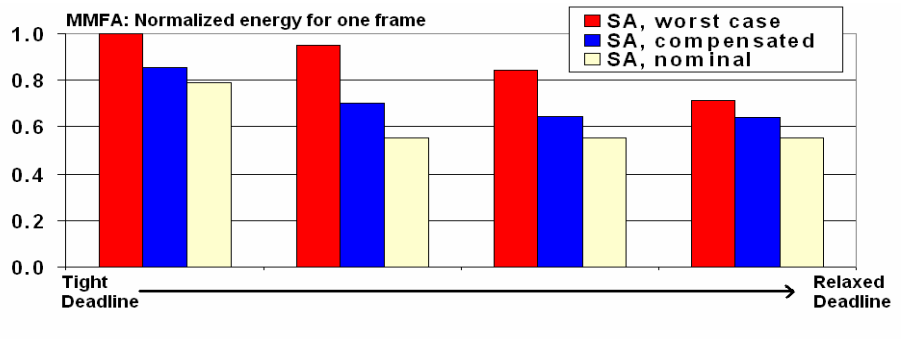


Figure 5.13: Compensation technique *TLC*: Energy consumed by the scenario version of MP3 Decoder for different timing requirements and memory considerations. The nominal bar is the theoretical minimum assuming no process variability exists.

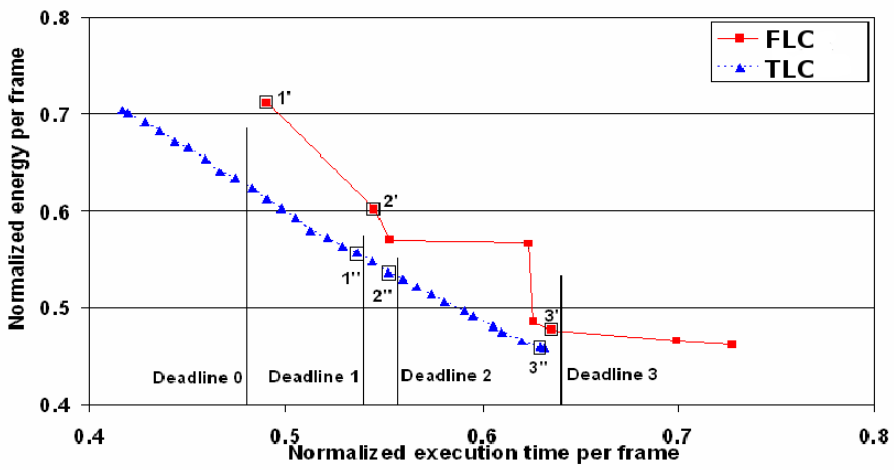


Figure 5.14: Performance behaviour between *FLC* and *TLC* for different deadlines.

Taking as example *Deadline1*, *FLC* outputs the operating point $1'$. However, adjusting the mode and the frequency in a coordinating way, *TLC* provides another solution point (denoted as $1''$) much closer to the deadline. This reduction in the slack to the deadline translates into an additional 20% energy saving over the *FLC* counterpart. As deadlines are more lenient, the differences between techniques diminish. Thus, for *Deadline2* the energy difference between both techniques is only around 7% and becomes insignificant for *Deadline3*.

As a result of working at task level varying memory modes and frequency, the *TLC* compensation technique allows to meet deadlines that the compensation based exclusively in memory modes (*FLC*) is unable to meet. Thus, point $1'$ in Figure 5.14 corresponds with the fastest configuration: all the memories will be switched to the fast mode during the whole execution of the frame. Therefore, the cycle time is limited by the access time of the slowest memory (in its fast mode). *TLC* may increase performance further: certain tasks will run at a faster clock cycle if they do not access slow memories. This way the overall execution time may be reduced. All the points on the left of *Deadline0* are only achievable with this second technique.

Figure 5.14[2] indicates a general trend: the tighter the deadline, the larger the energy savings of the *TLC* technique. Figure 5.15 highlights this behavior further. It is shown the percentage of energy saved by the second technique over the memory mode compensation. In the extreme case (first bar), using *FLC* requires all the memories to be switched to the fast and energy expensive mode. Frequency tuning allows *TLC* to switch them to the slow mode during the execution of certain tasks. These accesses to slow mode memories account for a significant overall energy reduction.

As the deadline is relaxed, more and more memories may be permanently set to the slow mode, even for *FLC*. Thus, energy consumption differences tend to decrease, until both techniques reach the same slowest configuration: all the memories working in its slow mode for an entire frame[3].

[2]Energy and time have been normalized to the same worst-case points already applied in Figure 5.10.

[3]It should be highlighted that the slowest configuration, i.e. all the memories working in its slow mode, does not provide the same performance in both techniques. Since *TLC* fixes the processor frequently on a task basis, it usually provides faster solutions (those tasks that do not use the slowest memory module run faster than the *FLC* counterpart).

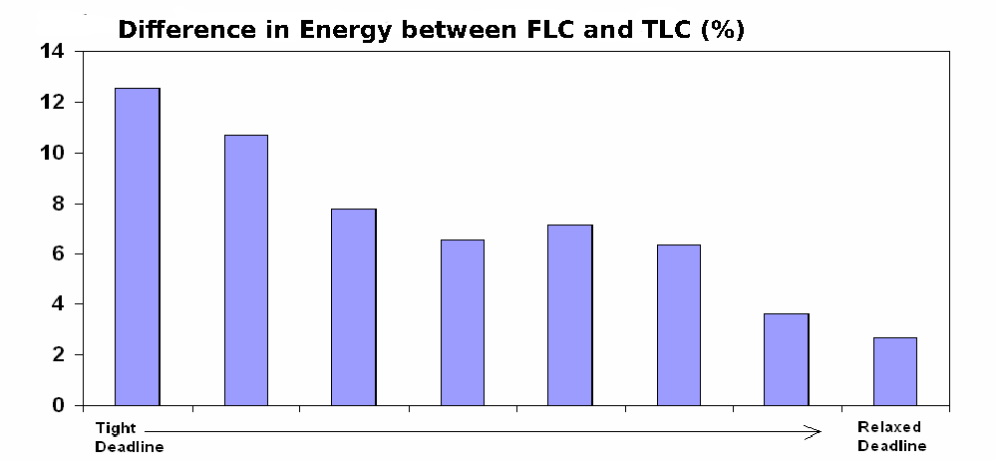


Figure 5.15: Energy savings of frequency and mode compensation (TLC) when compared to only mode compensation (FLC). From left to right, the x-axis is ordered from low to high deadlines.

5.4. Configurable memories: exploring the impact of a variable number of modes

A significant part of the methodology described in this chapter is based on the use of configurable modules on the entire memory architecture. At circuit level, the basis of such modules have been presented in Section 2.4, referring to the original research done in [WMP+05]. There, results from memory implementations with two modes were shown. The experiments carried out so far in this chapter have been limited to two-mode configurable memories. However, implementations with more modes are also feasible from a technological point of view. In this section we explore the effects that a variable number of modes can have in the memory system. As we show next, the most significant consequence of using configurable memories with a larger number of modes is the reduction in the energy consumed by the

memory architecture. Our experimental results show that these energy savings are highly related to the way the mode allocation is performed in the memory system.

5.4.1. Experimental results

The architectures considering two-mode configurable memories are the starting point for our tests, where we explore the impact of a higher number of modes per memory. For each memory module involved in the architecture, the extra modes are added as intermediate options between the two modes already evaluated. Since the extreme modes remain the same, memories do not become faster after introducing intermediate modes. We just increase the chances to adapt efficiently, in terms of energy, the memory system to the run-time application requirements.

There is no a-priori answer about the memories which get the most benefit from introducing additional modes. In absolute terms, large memories might benefit more, since the absolute difference of energy-per-access between two consecutive modes is proportional to the memory size, i.e. the difference is lower in small memories than in large memories. On the other hand, small memories tend to be heavily accessed and hence the overall impact might be more important. The experiments developed in this section will provide us a better knowledge about the memory behaviour in order to answer this question.

The tests performed to study the influence of the number of modes on the memory system are based on the MP3 decoder version which implements scenarios (*SA version*). In particular, we focus on the *Long* scenario. The memory architecture involved consists of ten memories: one 4KB memory, two 8KB memories, four 16KB memories and three 32KB memories. This is the architecture explained in Section

5.1.2. For comparison purposes, we consider as baseline the energy consumed by the initial memory system consisting of two-mode configurable memories.

To start with, we have studied the way the energy savings scale by adding extra intermediate memory modes in the memory platform considered as baseline. Exploring a variable number of modes can generate a wide amount of simulations, as many as different mode assignment per memory you can think of. In order to simplify this process and obtain meaningful information we have opted for simulating only two memory platforms. These platforms are equivalent to the one considered as baseline, but with a larger number of modes. The former consists of four-mode memory modules, while the latter implements eight modes in all its modules. This first exploration based on an equal mode assignment is called *homogeneous mode allocation*.

Graphically, the mode distribution used in this exploration can be seen in Figure 5.16. Assuming a memory consisting of eight modes, as the one shown in Figure 5.16.a, its four-mode version would use the modes highlighted in Figure 5.16.b. A basic memory, implementing only two modes, would use the extreme modes pointed out in Figure 5.16.c.

Results from simulation of the two equivalent platforms and the baseline are plotted in Figure 5.17, which illustrates the energy consumed by the MP3 decoder *Long* scenario under each assumption. The *FLC* compensation technique explained in previous sections has been applied in each case to a common set of deadlines. The energy of the solutions provided for each deadline in each memory platform, i.e. the configuration that meets the application constraints with less energy, is shown in the figure. As Figure 5.17 shows, scaling from two to four modes achieves an impressive improvement which reaches up to 50% in demanding time constraints. As the

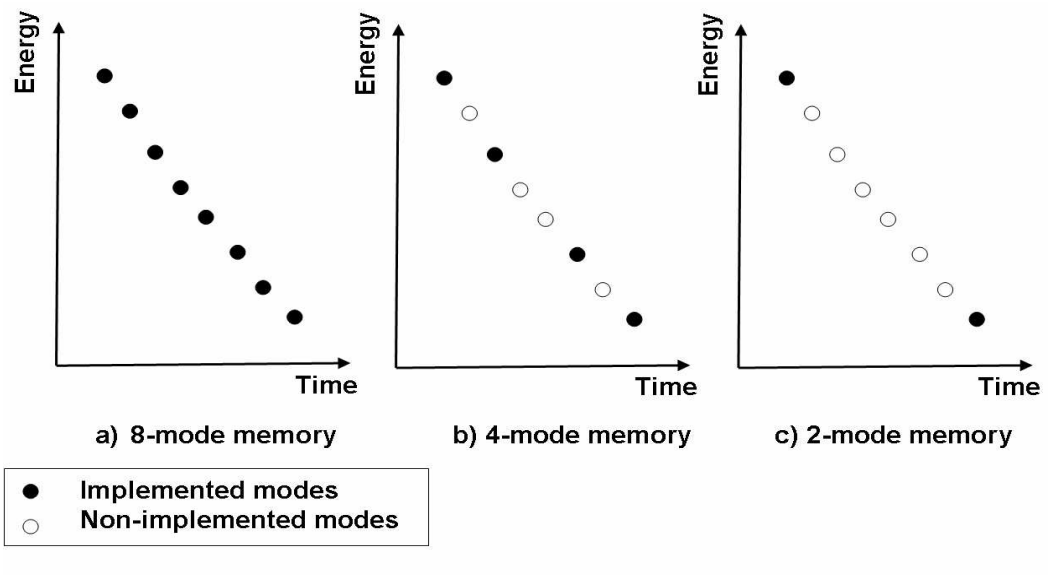


Figure 5.16: Mode distribution applied to memory modules.

number of modes grows from four to eight, the energy improvements are clearly lower, pointing out that beyond four modes the energy savings are inversely proportional to the number of modes. Another remarkable aspect from Figure 5.17 concerns lenient deadlines. As time constraints are less strict, memories tend to be configured in the slowest modes. This behaviour makes unnecessary a large number of modes in these cases, since, as can be seen in the figure, *FLC* compensation technique provides the same solutions for the three mode allocations tested. Besides, adding more modes cannot be done costless since area is also affected. For small and medium size memories this area overhead – less than 5% according to estimations in [WMP+05] – can be tolerable, but for large memories it might compromise the available area budget.

Results in Figure 5.17 and area considerations seem to indicate that homogeneous mode allocation cannot scale well beyond four modes in all memories. An alternative approach can be to consider an *heterogeneous mode allocation*, where

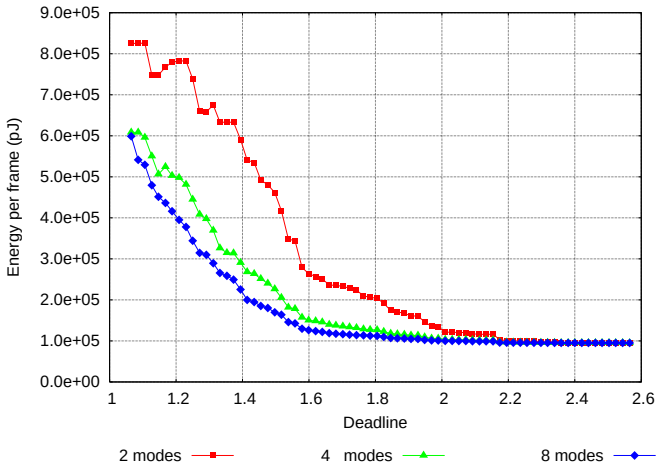


Figure 5.17: The most significant reductions in energy happens extending the number of modes from two to four. Increasing modes from four to eight achieves lower savings.

memories do not need to have the same number of modes, i.e. the mode assignment can be different from one memory to another. Homogeneous and heterogeneous memories only differ on the number of modes distributed among memories at design time. At run-time, the way the memories are configured or switched is common for both types of memory architectures.

Applying this heterogeneous approach requires information about the convenience of using more than four modes or the memories which can get a larger benefit from such increment. In order to obtain this information, we find out the usage frequency of the modes existing in memories. Considering our eight-mode homogeneous allocation and a range of deadlinesto apply *FLC*, the information collected about usage is summarized in Figure 5.18. The graphic shows how often each memory module stays in each particular mode along the different deadlines, which include from strict to more lenient time requirements. It is specially remarkable that smallest memories – 4KB and 8KB – are almost exclusively configured in the lowest power mode. Regarding larger memories, although they can be kept at the

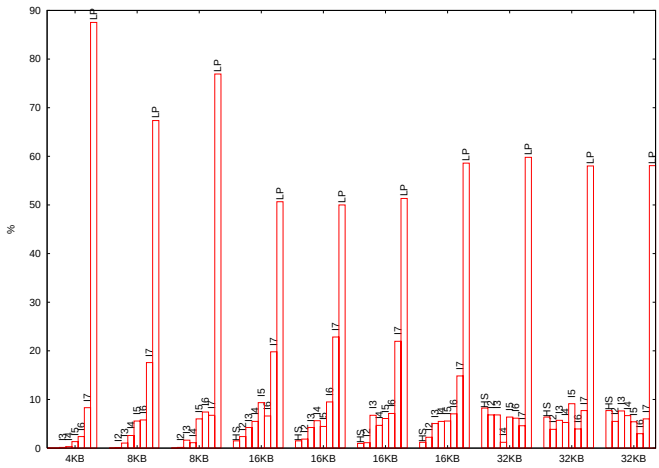


Figure 5.18: The smallest memories from the ten-memory system considered as baseline are configured most of the time in their slowest mode.

slowest mode half of the time, intermediate modes are more often used if we compare with small memories. In the light of these results, it seems obvious that large memories should incorporate more modes, as they use them more often and each extra mode reports to these memories larger energy savings per access.

In these first results, the memory size has been the only factor considered. However, when data allocation is also taken into account, results are quite the opposite, as Figure 5.19 shows. Figure 5.19 illustrates the energy per frame for different homogeneous and heterogeneous allocations. Scaling up the number of modes only in large memories does not translate into significant savings. Starting from the two-mode memory system established as baseline, the assignment of eight modes only to 16KB and 32KB memories – seven out of ten memories – obtains an energy consumption close to the baseline system. Although the benefits on energy per access provided by the additional modes are important in large memories, the savings are negligible since they store the less frequently accessed data structures. On the contrary, adding more configuration options only to the small memories – 4KB and

8KB memories – provides energy savings comparable to the eight-mode homogeneous case, outperforming the four-mode homogeneous allocation at a lower area cost.

The main benefits from this heterogeneous organization focused on small memories happen with very demanding deadlines. In this situation, this approach saves up to 20% more energy than its four-mode homogeneous counterpart because the small memories are fast enough to stay in their slowest mode most of the time. When the deadline becomes very hard to meet, even small memories must switch to a faster mode. Since small memories account for the majority of accesses, extra intermediate modes help to reduce the energy impact while meeting the time constraints.

Regarding area penalty, Table 5.3 represents the area required for each of the memory organizations plotted in Figure 5.19. The information, normalized to the area occupied by the two-mode memory organization, shows that memory size is relevant when the number of modes is chosen in heterogeneous mode allocations. Large memories with a large number of modes, such as *8 modes only in large memories*, occupy an area similar to the homogeneous allocation based on eight modes, although its benefits in energy are far away from this eight-mode homogeneous version. However, restricting higher modes only to small memories achieves energy reductions similar to the most expensive allocation in an area similar to the two-mode homogeneous organization. We only scale three memories out of ten while energy savings are really significant.

According to these first results, we can conclude that data assignment is a relevant factor to take into account in a process of mode allocation. Energy reductions

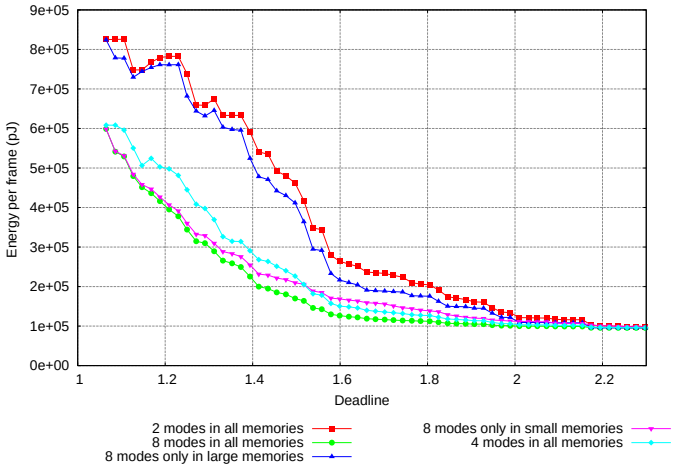


Figure 5.19: Heterogeneous memory organizations allows intelligent distributions of memory modes, adding more where more power is consumed.

are more significant when additional modes are provided to the most accessed memories in the platform, which are the smaller memories in our driver application.

Memory allocation	Normalized area
2 modes in all memories	1
8 modes in all memories	1.54
8 modes only in large memories	1.46
8 modes only in small memories	1.08
4 modes in all memories	1.15

Table 5.3:

5.5. Conclusions

The methodology explained in this chapter illustrates the potential of turning a dynamic application and an uncertain memory system into an adaptive system, which is able to meet the required time constraints with a minimal cost in energy. Application dynamism can be handled using scenarios and platform unpredictability can be managed using configurable memories and compensation techniques.

The compensation mechanism based on mode adjustment is fast due to its greedy approach, at the expense of a high energy cost compared to a more complex compensation technique as the one based on mode and frequency. This second technique is more effective for tight timing constraints but requires a larger setup step. Comparing with conservative approaches here represented by the worst-case *BASE* version (case (1) at Figure 5.8), simulation results reveal energy savings up to 50% in the memory organization by combining scenarios and configurable memories in a coordinated way.

The methodology, distributed along three steps, removes the need of design margins to deal with dynamism or variability. The design process is characterized by the application profiling, the scenario characterization and the search of an optimum memory architecture. At setup the calibration process measures the impact of variation on the memories and use compensation techniques to build Pareto curves to deal with variability and application dynamism at run time. During the application execution, the status of the system is monitored and the memory adapted, if necessary, to ensure the optimum fulfilment of the required timing requirements.

Performing the compensation techniques at setup time ensures that no better configurations can be generated, as the variability impact is already known at that point. However, this guarantee produces a large penalty in time, since the building of the Pareto curves can be costly depending on the application complexity, the scenarios or the compensation technique applied.

The methodology shown here has proved its benefits on energy saving and fulfilment of time constraints. But it is necessary to lighten the setup step to make the methodology really practical. This can be achieved moving as much work as possible from setup time to design time, although at the expense of loosing accuracy in

the process. The methodology developed to face this uncertainty will be explained in next chapter.

Regarding configurable memories, this chapter shows the close relationship existing between energy consumption and number of modes operative in each memory module. As we have also pointed out, memory mode allocation is specially affected by data assignment. Thus, since small memories store the most accessed data in the application, these are the memories which benefit more from an increment in the number of modes per module.

6

Uncertainty mitigation methodology

In previous chapters we have motivated the challenges to tackle in this research: process variation at memory system and application dynamism. We have also explained the individual procedures selected to deal with them: configurable memories and system scenarios respectively. Although both mechanisms are completely independent from each other, we have probed the benefits in terms of energy consumption and timing constraint fulfilment that can be achieved when both work in a coordinated way.

The methodology, based on three steps – design time, setup and run time –, deals with both sources of uncertainty. At design time, scenarios are extracted and the memory architecture is established. At setup, compensation mechanisms are applied to mitigate variation on memories and application dynamism. These compensation techniques generate configurations to enable the memory system adaptation at run time.

The compensation mechanisms applied during the exploration step involve tasks which may become too demanding to be performed at setup time. The main con-

cerns are related to the large timing penalties that can appear when the application is complex enough. This aspect turns the methodology explained previously into an ideal reference, since it is unlikely to be applied in certain environments.

Taking as starting point this first methodology focused on the memory architecture, in this chapter we have developed our final variation-aware design methodology, where complex tasks are moved from setup to design time to make the setup process lighter.

6.1. Global overview

As in our first approach, this final methodology consists of three main steps: design time, setup and run time. The processes and activities carried out in each step are outlined next and summarized in Figure 6.1, where the flow and the main mechanisms involved in the methodology are highlighted.

6.1.1. Design time

This step becomes the most significant in the methodology since all the relevant decisions are taken at this stage. It is also the most timing consuming. In this step, the design of the entire memory system is specifically focused on the target application to be executed at run time and the fulfilment of its time and energy requirements.

Firstly, the target application is profiled using the ATOMIUM tool suite. During this step, the application behaviour is characterized in terms of energy, execution time or memory access patterns to be clustered in *system scenarios*. Next, being aware of the application memory requirements, memory partition and data assign-

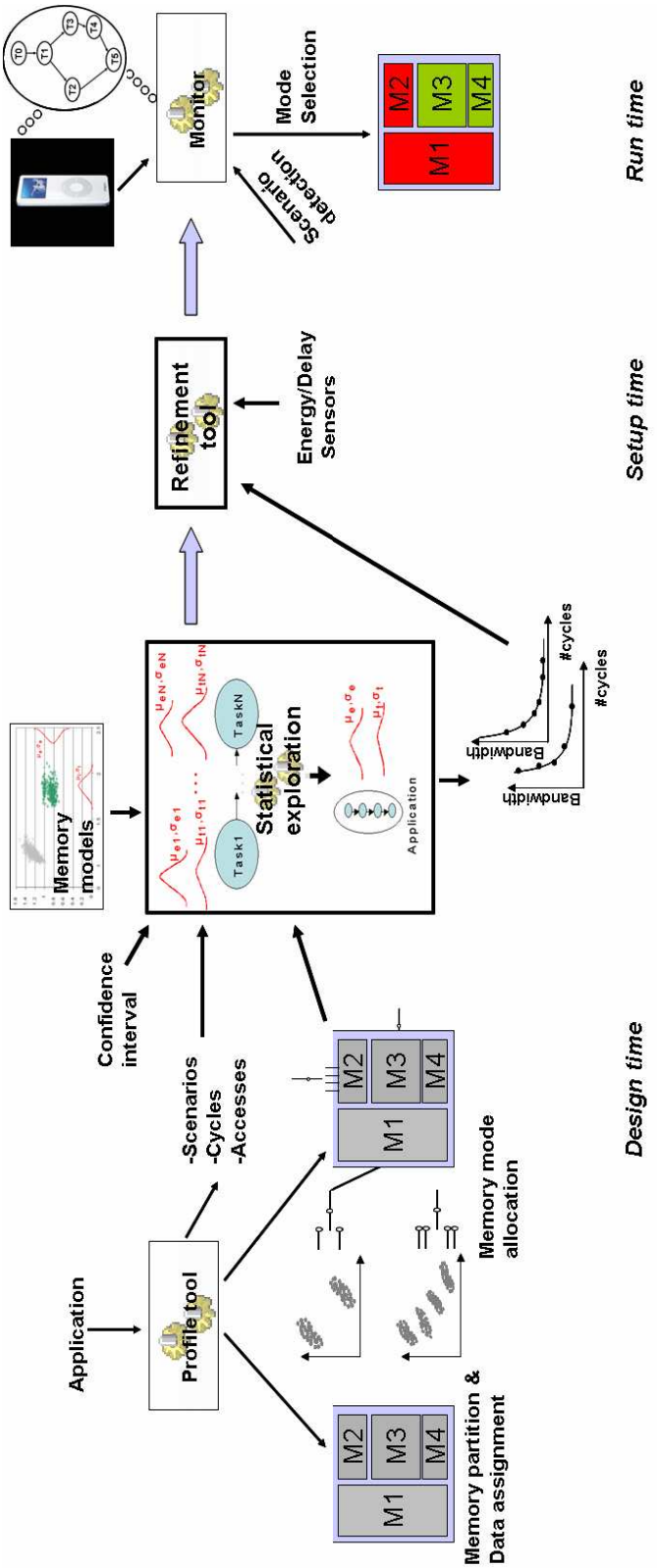


Figure 6.1: Full system methodology. Orthogonal steps are applied at design time, where the heaviest part of the methodology is carried out.

ment are performed. The memory partition allocates a heterogeneous set of memories to provide an energy-efficient memory organization. These steps have been already explained in Chapter 5 and applied to the MP3 decoder. In this chapter we use the same memory architecture and scenario division already explained in previous chapters, assuming, once again, the MP3 decoder as target application.

Secondly, once the memory system has been determined, we establish the number of modes that will be integrated in each configurable memory existing in the system. Results shown in the previous chapter suggest that factors such as memory size, data assignment and frequency access need to be considered in this step. In Section 6.2 we provide an automatic mechanism to determine the suitable number of modes to implement in each memory module.

At this point, according to the former methodology, the design time step finishes, waiting until the setup step generates the memory configurations to deal properly with process variation and application dynamism. The existence of monitors, which measure the variability impact on the memory modules after fabrication, allows to explore and provide an optimal set of memory configurations for each application scenario. However, the time penalty associated to this exploration could be not affordable. This penalty turns the generation of memory configurations into a candidate to be moved at design time. Nonetheless, this movement cannot be done without a penalty in accuracy, as at design time is not possible to measure the variability impact that will be present in the memories at the end of the fabrication process. To overcome this challenge we have developed a statistical technique to generate memory configurations at design time. The technique proposed takes as input memory models where variability is included, the description of the application in terms of accesses performed to each memory module, the set of expected

deadlines and a confidence interval to meet them. By means of statistical models we can estimate the variability impact on memories. We have chosen as compensation technique to generate the memory configurations *TLC*. As in the former version where the setup was the main step, here the statistical mechanism generates as output a set of global configurations for the tasks existing in each scenario. For each specific deadline and scenario, its associated configuration indicates the mode in which each memory has to work. Details are explained in Section 6.3.

6.1.2. Setup time

At the time the methodology enters into the setup step, the memory configurations have been already generated at design time, reducing this step significantly. However, these configurations obtained statistically to deal with variation are not necessarily the optimal solutions. A high accurate model about variability effects on the energy and delay of memories is not necessary at design time since the actual characteristics, together with the design-time configurations, are used at a *Setup Refinement* for a better adaptation of the system to the actual variability effects. This refinement step can be made by knowing the actual energy and delay characteristics existing in the memory modules, which allows us to figure out how accurate our model was at design time. Supported by the existence of suitable monitors, the actual energy and delay characteristics from the memory modules can be estimated at setup time. A detailed explanation of this step is given in Section 6.4.

6.1.3. Run time

At this point, the hardest part of the methodology has been already accomplished. During the execution, the system is monitored to detect changes in the current scenario or in the timing requirements (due to, for example, QoS reasons). If there is a change in any of these parameters, a new optimal pre-stored configuration is selected and applied to fulfil the new conditions. Realtime considerations are fulfil since: (1) the monitoring system is a light process which does not affect the application performance; (2) the reconfiguration step takes short time and happens occasionally. Besides, the number of stored configurations is large enough and uniformly distributed to cover the expecting time constraints and fits any situation properly.

6.2. Mode selection: Establishing modes for an entire memory system

A variable number of modes per memory is technologically feasible and our experiments have pointed out its impact on the energy of the memory architecture. However, as its implementation is not costless, it is necessary to develop a mechanism to establish the most suitable number of modes in each memory module. As we already mentioned, this mechanism has to take into consideration the memory allocation, data assignment and application profile generated in the first steps at design time. But also, it has to consider limiting factors, such as the energy/delay trade-off or the area overhead, which can restrict the number of modes in a memory. The energy/delay trade-off is related to the distance in energy and delay

between each two consecutive modes. It must be clear the frontier between two modes so the selection between them is not ambiguous. The second limiting factor comes from the area overhead that modes introduce in the circuit. In [WMP+05], this overhead was estimated around 5% for a two-mode memory.

Considering these points, we present an algorithm to determine the optimal mode allocation for every single memory in the system as a result of a trade-off between energy improvements and area. The current algorithm takes into account the area as main limiting factor and the expecting energy savings. The energy-delay trade-off is guaranteed by setting a maximum number of modes and establishing the adequate energy/delay distance between each two modes.

Our exploratory study has highlighted the need for a proper mechanism to determine automatically the most suitable number of modes to implement in a particular memory system. Next, we explain the algorithm developed for such purpose.

As Figure 6.1 illustrates, the mode allocation happens right after the memory system is allocated and data are assigned. The output provided by these steps is used as input for our mode allocation technique.

From the experimental tests accomplished and explained in the previous chapter, it can be inferred a close connection among memory allocation, data assignment and mode allocation, which impacts on the energy consumption. The technique developed takes these factors into account, as well as the area penalty as main limiting factor associated to the mode allocation process.

To model the area of our memory architecture we initially use CACTI [CAC], an analytical model for features such as access, cycle time, leakage, dynamic power and area at on-chip caches. CACTI estimates the area for each one of the memory modules involved in the system, considered as regular memories with a single op-

erating point. It is applied then a penalty factor to each module, similar to the one reported in [WMP+05], to include the modes in the area information.

Algorithm 6.1 summarizes the general mode allocation technique developed. The method has been developed as a gradient search, since exhaustive explorations are usually unfeasible due to the huge space search resulting from the memories included in the system and the number of modes allowed in each one. Every memory in the system is considered individually, i.e. memories with the same size are analyzed independently from each other, so they can end up having different number of modes despite their equal size. Next the algorithm is explained in detail.

Firstly, the area limit is set up, so mode allocations too expensive in terms of area are not considered (line 1). Next, the algorithm, which is performed once for each deadline expected at run time, takes as starting point the energy and area estimated for a memory system where all the configurable memories consist of two modes (lines 3 to 8). This homogeneous mode allocation based on two modes is considered as the initial solution for the algorithm. The energy estimation is obtained by applying one of the compensation techniques previously described: *FLC* and *TLC*. In this case we have used the *TLC* compensation technique to determine the memory configuration which allows to meet the deadline with the lowest cost in energy. The estimation is performed assuming mean values for the energy and access time of each memory in the system. These mean values are obtained by taking into account the variability impact on the mentioned parameters. Once we have established the starting point, the algorithm enters in a loop (line 11) which covers all the memories in the system and increases the number of modes available for each memory. Loop at line 16 tests all the memories with the same number of modes, generating different mode allocations along this process. The one which provides

Algorithm 6.1: Memory mode allocation

Input: Memory allocation *memories*, application profile $profile$, time constraint *deadline* and area information $info$
Output: Memory mode allocation *bestModeAllocation*

```
1  limitArea = CalculateLimitArea (memories,info);
2  mode = 2;
3  foreach m ∈ memories do
4  |  bestModeAllocation[m] = oldModeAllocation[m] = mode;
5  end
6  ratio_0 = 0;
7  a_0 = CalculateArea (memories, bestModeAllocation, info);
8  e_0 = GetConfiguration (memories, deadline, profile,
   bestModeAllocation);
9  (e_tmp, a_tmp, continue ) = (e_0, a_0, 1);
10 mode = NextMode (mode);
11 while continue do
12 |  ratio_tmp = 0;
13 |  foreach mem ∈ memories do
14 |  |  modeAllocation[mem] = bestModeAllocation[mem];
15 |  end
16 |  foreach m ∈ memories do
17 |  |  modeAllocation[m] = mode;
18 |  |  a = CalculateArea (memories,info_a);
19 |  |  e = GetConfiguration (memories, deadline, profile);
20 |  |  ratio = (e_0 - e)/(a - a_0);
21 |  |  if (e < e_tmp)and(a < limitArea)and(ratio > ratio_tmp) then
22 |  |  |  (e_tmp,a_tmp,ratio_tmp) = (e,a,ratio);
23 |  |  |  foreach mem ∈ memories do
24 |  |  |  |  auxModeAllocation[mem] = modeAllocation[mem];
25 |  |  |  end
26 |  |  end
27 |  |  modeAllocation[m] = bestModeAllocation[m];
28 |  end
29 |  if (e_tmp < e_0) then
30 |  |  (e_0,a_0,ratio_0,continue) = (e_tmp,a_tmp,ratio_tmp,1);
31 |  |  foreach mem ∈ memories do
32 |  |  |  oldModeAllocation[mem] = bestModeAllocation[mem];
33 |  |  |  bestModeAllocation[mem] = auxModeAllocation[mem];
34 |  |  end
35 |  else
36 |  |  if (bestModeAllocation ==
   |  |  oldModeAllocation)and(LimitMode(mode)) then
37 |  |  |  continue = 0;
38 |  |  end
39 |  |  mode = NextMode (mode);
40 |  end
41 end
42 return bestModeAllocation
```

the highest ratio between area and energy is stored in *auxModeAllocation*. This ratio is calculated at line 20, looking for the mode allocation which consumes the lowest energy compared with the current best mode allocation, while its area is as close as possible to that one. From lines 29 to 34, the best mode allocation so far is compared with the temporal solution found in the previous loop, updating the best solution if necessary.

The number of modes to check is given by the function *NextMode*, which can increase the number of modes or reduce it again to two modes in order to test all the options available for the memory system. This function increases the number of available modes when the loop at line 16 is not able to provide a better mode allocation. Otherwise, the number of modes is kept stable. Once the number of modes achieves its upper bound, the value is initialized again to two modes. The algorithm comes to an end when the mode allocation has not changed between two consecutive rounds of the outer loop (line 11) and all number of modes have been checked.

For each deadline, the algorithm finds out the *local* mode allocation which exhibits the highest energy reduction with the lowest area overhead, compared with the memory system established as initial baseline (two-mode allocation). Once we have collected for each deadline its best mode allocation, a final global step analyses the allocations of all the deadlines and outputs the one that, on average, provides the best energy/delay ratio. This solution is mathematically described as:

$$modeAlloc = min\{\frac{\sum_{i=0}^{N-1} \frac{\left(energy_{i,j} - energy_{i,best}\right)}{\left(area_{i,best} - area_{i,j}\right)}}{N} : j = 0,\ldots,M-1\} \tag{6.1}$$

being M the number of feasible mode allocations to compare, N the number of deadlines considered, and *best* the mode allocation where every memory implements eight modes.

6.3. Statistical Design-Time Exploration

Once the memory architecture is completely determined, the last step at design time concerns the estimation of the memory configurations that will be used at run time. Generating these configurations at design time requires to take into account the future impact of variability on memories after fabrication. In this section we explain the statistical technique developed to determine these memory configurations.

In our methodology, *system scenarios* are modelled as a set of sequential tasks which have to meet a set of possible deadlines $T_j : j = 0, \ldots, L-1$ at run time. For each deadline, every task k has a configuration vector $c^k = [c_0^k, \ldots, c_{N-1}^k]$, whose elements c_i^k indicate the mode selected for each of the N memories existing in the architecture – including a special *off* mode if the memory is not accessed by the task – for each deadline. Thus, a *global configuration*, one per deadline, is composed of the configuration vectors of all the tasks involved in a particular *system scenario*. The goal of this design phase is to find the configuration vectors for each task. Therefore we need to model the execution time and energy consumption of the scenario as functions of the configuration vectors and the parameters of the N memories.

To model the execution time of task k, we use the number of cycles needed to complete it and the clock frequency used, which is constrained by the latency of the slowest memory accessed by the task. Formally, we denote by $\tau[i, c_i^k]$ the delay per

access to memory i configured for task k on mode c_i^k. Then the slowest access time involved in the execution of the task is:

$$\tau_k = max_i\{\tau[i, c_i^k] : i = 0, \ldots, N-1 \wedge c_i^k \neq off\} \tag{6.2}$$

and the execution time of the task can be modeled as:

$$t_k = C_k \tau_k \tag{6.3}$$

where C_k is the number of cycles needed to execute the task. The global execution time for the M tasks is:

$$T = \sum_{k=0}^{M-1} t_k \tag{6.4}$$

Regarding energy, the contribution of the j mode of the i memory to the global energy consumption is modeled as:

$$e_{i,j} = \varepsilon[i, j] \sum_{k=0}^{M-1} n_i^k \delta_{j,c_i^k} \tag{6.5}$$

where M is the number of tasks in the scenario, n_i^k is the number of accesses performed by task k to memory i, $\varepsilon[i, j]$ is the energy consumed per access to memory i configured on mode j, and $\delta_{i,j}$ is 1 if $i = j$ and 0 otherwise. The global energy consumed by the scenario is:

$$E = \sum_{i=0}^{N-1} \sum_{j=0}^{P_i-1} e_{i,j} \tag{6.6}$$

where P_i is the number of modes of memory i.

Due to process variation, $\tau[i, c_i^k]$ and $\varepsilon[i, j]$ are random variables with certain probability distribution, and thus T and E become also random variables. The goal of this phase is to obtain the configuration vectors $c^k : k = 0, \ldots, M-1$ for each deadline $T_j : j = 0, \ldots, L-1$, so that the probability of the execution time T being lower than T_j is controlled. Among all the candidates which fulfil this condition, we will keep those that, with a controlled probability, have a lower energy consumption.

To select these solutions we use a modification of the exhaustive algorithm we explained in Section 5.2. It consists in a branch and bound algorithm which updates a list of valid solutions and a *best* candidate[1]. A new solution is added to the list if and only if it satisfies the following conditions:

1. The probability of its execution time being lower or equal than a given deadline is larger than a certain threshold P_T: $P\left[T \leq T_j\right] > P_T$

2. The probability of its energy consumption being larger than the *best* candidate counterpart is smaller than a certain threshold P_E: $P\left[E_1 > E_{best}\right] \leq P_E$.

where the thresholds P_T and P_E are design parameters. After inserting a new valid solution into the list, we check if the *best* candidate needs to be replaced. In that case, we process the list of valid solutions to discard those which do not satisfy condition 2 with the new *best* candidate.

To find the expressions for these probabilities, it is convenient to define the total number of accesses to memory i when configured on mode j as:

$$N_{i,j} = \sum_{k=0}^{M-1} n_i^k \delta_{j,c_i^k} \tag{6.7}$$

[1]The branch and bound algorithm applied here is explained in Appendix A.

and rewrite Equation 6.5 as:

$$e_{i,j} = \varepsilon[i,j] N_{i,j} \tag{6.8}$$

Notice that, for a given global configuration, T and E are random variables expressed as summations of the random variables t_k and $e_{i,j}$. We denote by $f_{\tau_k}(x)$ and $f_{\varepsilon[i,j]}(x)$ the probability density functions (pdfs) of τ_k and $\varepsilon[i,j]$ respectively. Then we can express the pdfs of t_k and $e_{i,j}$ as:

$$f_{t_k}(x) = \frac{1}{C_k} f_{\tau_k}\left(\frac{x}{C_k}\right) \tag{6.9}$$

$$f_{e_{i,j}}(x) = \frac{1}{N_{i,j}} f_{\varepsilon[i,j]}\left(\frac{x}{N_{i,j}}\right) \tag{6.10}$$

In this work we assume that process variation affects the delay and energy per access of each memory and each mode independently, making $\varepsilon[i,j]$ and its derivatives $e_{i,j}$ independent random variables.

Since the probability density function of the summation of two independent random variables X and Y can be expressed as the convolution of their probability density functions:

$$f_{X+Y}(z) = f_X \otimes f_Y = \int_{-\infty}^{\infty} f_X(\lambda) f_Y(z-\lambda) d\lambda \tag{6.11}$$

we can express the probability density function of E as:

$$f_E = f_{e_0} \otimes f_{e_1} \otimes \ldots \otimes f_{e_{N-1}} \tag{6.12}$$

The expression of the pdf of τ_k can be computed from the pdfs and the cumulative density functions (cdfs) of each memory as:

$$f_{\tau_k}(x) = \sum_{i=0}^{N-1} \left(f_{\tau[i,c_i^k]}(x) \prod_{0 \leq j \leq N-1 j \neq i} F_{\tau[j,c_j^k]}(x) \right) \tag{6.13}$$

where $f_{\tau[i,c_i^k]}$ is the pdf of the access time to memory i on mode c_i^k and $F_{\tau[i,c_i^k]}$ is the corresponding cdf.

However, the *max* function in Equation 6.2 introduces dependencies among the t_k variables that cannot be handled by any equation reordering. These dependencies come from tasks, two or more, which share memories. As a result, the execution times of those tasks are not independent since common memories can be configured in the same mode in different tasks and may lead to tasks which share the same cycle time (τ). Unfortunately, these correlations between the execution times of the different tasks are hard to model. To tackle this difficulty in this research we propose the following work around. First, we consider the $\{t_k : k = 0, \ldots, M-1\}$ as independent random variables (knowing that this is not mathematically correct), and compute the pdf of T by:

$$f_T = f_{t_0} \otimes f_{t_1} \otimes \ldots \otimes f_{t_{M-1}} \tag{6.14}$$

This f_T is not necessarily a Gaussian distribution. However, empirically it is close to it. Therefore, we replace f_T by a Gaussian distribution with the same expected value ($\mu_T = E[f_T]$) and a variance that is c times larger than the variance of f_T ($\sigma_T^2 = c \cdot var[f_T]$). This empirical parameter c serves us to tackle the correlation between the execution times of the different tasks.

Using this Gaussian model for f_T we can perform the selection of the valid configurations as explained above, computing the target probabilities as:

$$P\left[T \leq T_j\right] = \int_{-\infty}^{T_j} f_T(x)dx \tag{6.15}$$

and

$$\begin{aligned} P\left[E_1 > E_{best}\right] &= 1 - P\left(E_1 \leq E_{best}\right) \\ &= 1 - P\left(E_1 - E_{best} \leq 0\right) \\ &= 1 - \int_{-\infty}^{0} \left(f_{E_1} \otimes f_{-E_{best}}\right)(x)dx \end{aligned} \tag{6.16}$$

where $f_{-E_{best}}(x) = f_{E_{best}}(-x)$.

In this research we assume that process variation turns all the memory parameters, i.e. $\tau[i,j]$ and $\varepsilon[i,j]$, into random variables with Gaussian distribution, making E also Gaussian with the following parameters:

$$\mu_E = \sum_{i=0}^{M-1} \mu_{e_i} \quad \sigma_E^2 = \textstyle\sum_{i=0}^{M-1} \sigma_{e_i}^2 \tag{6.17}$$

As explained, T is also modeled as a Gaussian random variable. This way, Equation 6.15 and Equation 6.16 translate to:

$$\begin{aligned} P\left[T \leq T_j\right] &= \frac{1}{\sigma_T\sqrt{2\pi}} \int_{-\infty}^{T_j} \exp\left(-\frac{(x-\mu_T)^2}{2\sigma_T^2}\right) dx \\ &= \frac{1}{2} + \frac{1}{2} erf\left(\frac{T_j-\mu_T}{\sqrt{2}\sigma_T^2}\right) \end{aligned} \tag{6.18}$$

$$\begin{aligned} P\left[E_1 > E_{best}\right] &= 1 - \int_{-\infty}^{0} \left(f_{E_1} \otimes f_{-E_{best}}\right)(x) dx \\ &= \frac{1}{2} - \frac{1}{2} erf\left(\frac{-(\mu_{E_1}-\mu_{E_{best}})}{\sqrt{2}(\sigma_{E_1}^2+\sigma_{E_{best}}^2)}\right) \end{aligned} \tag{6.19}$$

where erf is the error function.

6.3.1. Candidate list pruning

The number of candidate solutions obtained by means of the *Statistical Exploration* is usually small, compared with the huge exploration space managed. Although the model normally offers correct solutions, in order to check whether our Gaussian model was accurate enough to estimate the pdf of the execution time of our *best* solution we can perform a Monte Carlo simulation. Every solution is tested, removing from the list the ones which fail. Notice that performing the whole exploration using Monte Carlo simulations to estimate f_T for each configuration would make the exploration too heavy. However, a good estimation of c (we used $c = 2$ in our work) serves us to drastically reduce the number of candidate configurations and makes the statistical simulation affordable.

Once Monte Carlo simulations have been applied to each of the candidate solutions provided by our statistical exploration, it is possible to have a small list of

configurations. As we are looking for a unique solution for each deadline, we need to reduce the candidate list to a single element. The candidates which remain in the list exhibit a probability of fulfilling the deadline higher than threshold P_T. Besides, all of them have passed successfully the Monte Carlo simulations explained previously. To reduce the list to a unique configuration, we sort the configurations looking for the one which have the highest probability of consuming less energy at run time. This process, described in Algorithm 6.2, estimates for pairs of configurations (m, n) the probability of the following events:

- Event A: *Configuration m* is slower than *Configuration n*
- Event B: *Configuration m* consumes less energy than *Configuration n*

Equations 6.20 and 6.21 are used to establish these probabilities between two configurations.

$$\begin{aligned} P[T_m > T_n] &= 1 - \int_{-\infty}^{0} (f_{T_m} \otimes f_{-T_n})(x)dx \\ &= \frac{1}{2} - \frac{1}{2} erf\left(\frac{-(\mu_{T_m} - \mu_{T_n})}{\sqrt{2}(\sigma_{T_m}^2 + \sigma_{T_n}^2)}\right) \qquad (6.20) \\ P[E_n > E_m] &= 1 - \int_{-\infty}^{0} (f_{E_n} \otimes f_{-E_m})(x)dx \\ &= \frac{1}{2} - \frac{1}{2} erf\left(\frac{-(\mu_{E_n} - \mu_{E_m})}{\sqrt{2}(\sigma_{E_n}^2 + \sigma_{E_m}^2)}\right) \qquad (6.21) \end{aligned}$$

The final probability is given by the probability of both events ($A \cup B$):

$$P[A \cup B] = P[A] + P[B] - P[A \cap B] \qquad (6.22)$$

Considering events A and B as independent, the final probability is given by:

$$P[A \cup B] = P[A] + P[B] - P[A] * P[B] \tag{6.23}$$

As Algorithm 6.2 shows, every configuration m is compared with all the remaining configurations n in the list, checking if configuration m is statistically the most energy efficient in the list. Otherwise, configuration m would be deleted, repeating this process with the rest of the elements. In case configuration m seems to consume less energy and be slower than a configuration n, n is now evaluated regarding m. Configuration n will be deleted from the list if statistically consumes more energy and is faster than m. Given two configurations - m and n -, it can happen that we can not decide which configuration is statistically better. In this case, both are kept in the list, increasing the cumulative probability for both configurations. Once all the pairs have been checked, the initial list has been reduced to those configurations which can not be sorted. Whether the list contains more than one element, we will choose the one with the highest cumulative probability.

The process described in this section requires a statistical model of memory parameters, i.e. $\tau[i, j]$ and $\varepsilon[i, j]$, that should be provided by the manufacturers. In this dissertation, we rely on the Monte Carlo simulations performed in [HCM+05] to model the effect of process variability on these parameters. As it was mentioned in Section 5.3, the simulation is performed at the transistor level using a 65nm BSIM model, where the impact of variability reflects in a drift in the drain current (I_D) from its nominal value. As drain current in MOS transistors changes linearly with the threshold voltage (V_t) and the current factor (β), process variability has been modeled by adjusting these two sensitive electrical parameters [PDW89, HCM+05].

Algorithm 6.2: Configuration extraction

```
Input: list of candidate configurations, deadline
Output: global configuration
foreach m ∈ List do
    foreach n ∈ List do
        prob_m = Prob (m,n);
        if (prob_m < threshold) then
            Delete (m);
        else
            prob_n = Prob (n,m);
            if (prob_n < threshold) then
                Delete (n);
            else
                List[m].cumProb = List[m].cumProb + prob_m_pareto;
                List[n].cumProb = List[n].cumProb + prob_n_pareto;
                List[m].num = List[m].num + 1;
                List[n].num = List[n].num + 1;
            end
        end
    end
end
bestProb = 0;
foreach m ∈ List do
    aux = m.cumProb/m.num;
    if (aux > bestProb) then
        bestProb = aux;
        conf = m;
    end
end
return conf
```

The Gaussian parameters for modes in the memories (means and variances) are estimated from these simulation results. Figure 2.16 shows the result of a Monte Carlo simulation for a two operating mode memory. The Gaussian parameters of the memories (means and variances) are estimated from the simulation results.

As mentioned previously, our *Statistical Exploration* does not build a completely accurate model to characterize the effects of process variation. The step explained next adapts the solutions provided at design time based on the actual variability effects on energy and access time.

6.4. Setup Refinement

The global configurations obtained from the *Statistical Design-Time Exploration* phase have a controlled probability of being *good* configurations for the target deadlines. However, under a particular injection of variability they will not necessarily be optimal in terms of energy, even they might miss their deadlines. As we showed in the previous chapter, if logic is available to check the actual values of $\tau[i, c_i^k]$ and $\varepsilon[i, c_i^k]$ for a specific platform, we could potentially find the optimal configurations to use. However, this search would only be available at setup time, just after system boots, making a full exploration not affordable. In this section we propose a gradient search method to obtain a local optimum starting from the configurations obtained from *Design Time Exploration*.

Algorithm 6.3 describes the gradient search. If the original configuration does not meet its corresponding deadline for the actual variability, the algorithm modifies the configuration in the direction that minimizes the ratio $\Delta T/\Delta E$ (reducing T implies $\Delta T < 0$ and $\Delta E > 0$) by iteratively performing *delay reduction steps*.

Once the modified configuration meets the deadline, the algorithm proceeds in the direction that minimizes the ratio $\Delta E/\Delta T$ constrained by $T \leq$ deadline. This movement is performed by means of *energy reduction steps*. This step continues until no additional movements preserve the deadline.

The *delay reduction step* is described in Algorithm 6.4. Each time this function is called, only one task changes its configuration, the one which allows to reduce time with the lowest cost in energy. To select the best option the algorithm considers each task in the scenario individually, selecting the slowest memory in each case. This memory, which determines the execution time of the task, is set on to its next mode (line 7), which reduces its delay and increases the energy it consumes. This step involves an increment in the energy consumed by the task and establishes a new global configuration whose cost the algorithm needs to measure. Any configuration which minimizes the ratio $\Delta T/\Delta E$ in the whole application is kept as a prospective change. Before testing a new task, the initial configuration – stored in line 6 – is retrieved again in line 16.

Likewise, the *energy reduction step* is described in Algorithm 6.5. In this case, the aim is to reduce the energy consumption once the time constraint is guaranteed. It takes each task in the scenario and for each memory accessed by the task, the memory is set on its previous mode, which reduces the energy it consumes, and increases its delay. In this case, the new global configuration is set up by selecting the change that, meeting the target deadline, minimizes the ratio $\Delta E/\Delta T$ (line 15). The solution provided by this algorithm establishes a new global configuration which still meets the deadline but at a lower energy cost.

As this refinement process is performed at setup time, this step should be fast, which emphasizes the relevance of obtaining from *Design Time Exploration* con-

Algorithm 6.3: Setup Refinement

```
Input: initial global configuration
Output: new global configuration (local optimum)
new = initial_configuration;
repeat
    old = new;
    time = ComputeTime(old);
    if (time > deadline) then
        new = DelayReduction(old);
    else
        new = EnergyReduction(old,deadline);
    end
until new = old;
return new
```

figurations close to the local optima. Once the refinement process has been accomplished at setup, the platform is ready to execute the target application at run time and deal with variation.

6.4.0.1. Timing degradation

Once the setup refinement is accomplished, the memory configurations obtained are ready to face the specific variability impact existing in the memory system. Nevertheless, variability effects can vary along time due to aging-related alterations. By aging it is denoted the degradation that a system suffers along its lifetime. As we mentioned in previous chapters, aging is related to phenomena such as hot carrier injection, negative-bias-temperature instability (NBTI) or time-dependent-dielectric breakdown, and increases with high temperatures and threshold voltage shifts. Its effects have a negative impact on reliability, degrading access time in memory cells.

Although timing degradation may introduce smooth changes in the memory system which can alter the application performance and lifetime, the gradient search

Algorithm 6.4: DelayReduction

Input: A global configuration *old*
Output: A *new* global configuration, moved one step down the time gradient direction

```
1  mingrad = 0;
2  t0 = ComputeTime(old);
3  e0 = ComputeEnergy(old);
4  new = old;
5  foreach T ∈ old.Tasks do
6  |  c = GetConfigurationVector(T);
7  |  cnew = FasterModeForSlowestMemory(c);
8  |  SetConfigurationVector(T,cnew);
9  |  tf = ComputeTime(old);
10 |  ef = ComputeEnergy(old);
11 |  grad = (tf − t0)/(ef − e0);
12 |  if (grad < mingrad) then
13 |  |  mingrad = grad;
14 |  |  new = old
15 |  end
16 |  SetConfigurationVector(T,c);
17 end
18 return new
```

technique proposed here could be re-scheduled again at run time as a first attempt to mitigate its effects. This would only require to monitor the system periodically at *run time*, or at time boot, and trigger the *Setup Refinement* whenever substantial changes in the memory parameters are detected. For this purpose would not be necessary to have support for an on-chip monitor system, as its application would not be frequent.

Algorithm 6.5: EnergyReduction

Input: A global configuration *old* and a target *deadline*
Output: A *new* global configuration, moved one step down the energy gradient direction

```
1  mingrad = 0;
2  t0 = ComputeTime(old);
3  e0 = ComputeEnergy(old);
4  new = old;
5  foreach T ∈ old.Tasks do
6      c = GetConfigurationVector(T);
7      foreach m ∈ c.memories do
8          if (m is off) then
9              continue
10         end
11         cnew = PreviousModeForMemory(c,m);
12         SetConfigurationVector(T,cnew);
13         tf = ComputeTime(old);
14         ef = ComputeEnergy(old);
15         grad = (ef − e0)/(tf − t0);
16         if (grad < mingrad)and(tf ≤ deadline) then
17             mingrad = grad;
18             new = old
19         end
20         SetConfigurationVector(T,c);
21     end
22 end
23 return new
```

6.5. Experimental results for MP3 Decoder

Results provided in this section regarding the statistical technique developed, as well as the later refinement process, refer to the MP3 decoder. The version used in this case is the one based on scenarios, whose characteristics have been already commented. From the two scenarios described in this version, we focus this research on the *Long* scenario, as it is more relevant in terms of workload.

The memory system consists of configurable memories, implementing a variable number of modes in each one according to the results provided by the mode exploration described in Algorithm 6.1.

Our methodology is evaluated in terms of percentage of fulfilled deadlines and average energy consumption. We first consider the case in which only *Design Time Exploration* is performed, extending the analysis to the results obtained after *Setup Refinement*.

6.5.1. Memory Mode Allocation

The mode allocation algorithm has been applied to the ten-memory architecture provided by ATOMIUM and the scenario version of the MP3 decoder, focusing on its *Long* scenario. The number of modes considered in our experiments with MP3, for every single memory in the system, is: two, four and eight. Therefore, the upper bound managed by function *NextMode* is eight. A higher number would be difficult to manage by the compensation techniques, as well as the energy-delay trade-off required between modes, which would be too reduced. Although any other number of modes between two and eight would have been feasible, for the sake of simplicity we have reduced the options to these three. For this specific case, the gradient search performed in this algorithm is able to manage successfully a search space which includes $modes^{memories}$ (3^{10}) potential mode allocations. Taking area as limiting factor, the upper bound managed by the allocation algorithm corresponds to the surface occupied for the ten-memory system where all the modules enable four operating points.

Under these initial assumptions, our mode allocation algorithm provides as solution a memory architecture consisting of: (1) eight modes assigned to small memories (4KB and 8KB); (2) four modes enabled in the 16KB memories existing in the system; (3) two modes assigned to large memories (32KB). This is the mode allocation we will consider from now on in this chapter, being named *Custom mode allocation*. Notice that, as we reported in the previous chapter, the small memories (4KB and 8KB), where the most accessed data are assigned, are also the ones which require more modes.

Table 6.1 compares the area penalty estimated for some significant mode allocations used in this dissertation. Among these, the mode allocations which represent the energy-delay boundaries in the search space. The lower limit is represented by the fully eight-mode allocation (*Best energy allocation*), while the upper limit is characterized by only two-mode allocation modules (*Best area allocation*). *Best energy allocation* allows to consume the lowest energy at the expense of occupying the largest area. On the contrary, *Best area allocation* requires a minimum area, but its energy solutions are just a subset from the ones available at *Best energy allocation*. *Custom mode allocation* represents the final solution provided by our algorithm, whereas *Tentative mode allocation* is the mode allocation used in the previous chapter as experimental case. This last mode allocation appears as one of the eligible solutions provided by Algorithm 6.1, really close to the final solution, as we will see. The area appears normalized to the area estimated for *Best energy allocation*. Our solution is close to *Best area allocation*, which occupies 35% less space than *Best energy allocation*, while our approach is 25% smaller.

Graphically, the memory architecture for each of these four mode allocations is shown in Figure 6.2, and their simulation at design time, assuming mean energy-

delay values, reports the energy-delay results reported in Figure 6.3. Our proposal is plotted together with the other three mode allocations already presented in Table 6.1. The solution provided by our mode allocation algorithm (*Custom mode allocation*) is very close, in terms of energy, to *Best energy allocation*. As can be noticed, configurations represented by *Custom mode allocation* and *Tentative mode allocation* are very similar in area and energy.

Mode allocation	Normalized area	Description
Best area allocation	0.65	Every memory module is implemented with 2 single modes
Custom mode allocation	0.74	One 4KB memory with 8 modes Two 8KB memories with 8 modes Four 16KB memories with 4 modes Three 32KB memories with 2 modes
Tentative mode allocation	0.70	One 4KB memory with 8 modes Two 8KB memories with 8 modes Four 16KB memories with 2 modes Three 32KB memories with 2 modes
Best energy allocation	1	Every memory module implements 8 single modes

Table 6.1: Area information

For comparison purpose, our memory mode allocation algorithm has also analyzed the memory architecture obtained for the original version of the MP3 Decoder, the one without scenarios [Lag01]. The algorithm provides as solution the following memory mode allocation: (1) two modes are assigned to the smallest memories (1KB); (2) eight modes for the 4KB memories; (3) eight modes are assigned to one of the 8KB memories, while the other 8KB memory gets four modes; (4) four modes are enabled in the 16KB memories existing in the system; (5) two modes assigned to the largest memory (32KB).

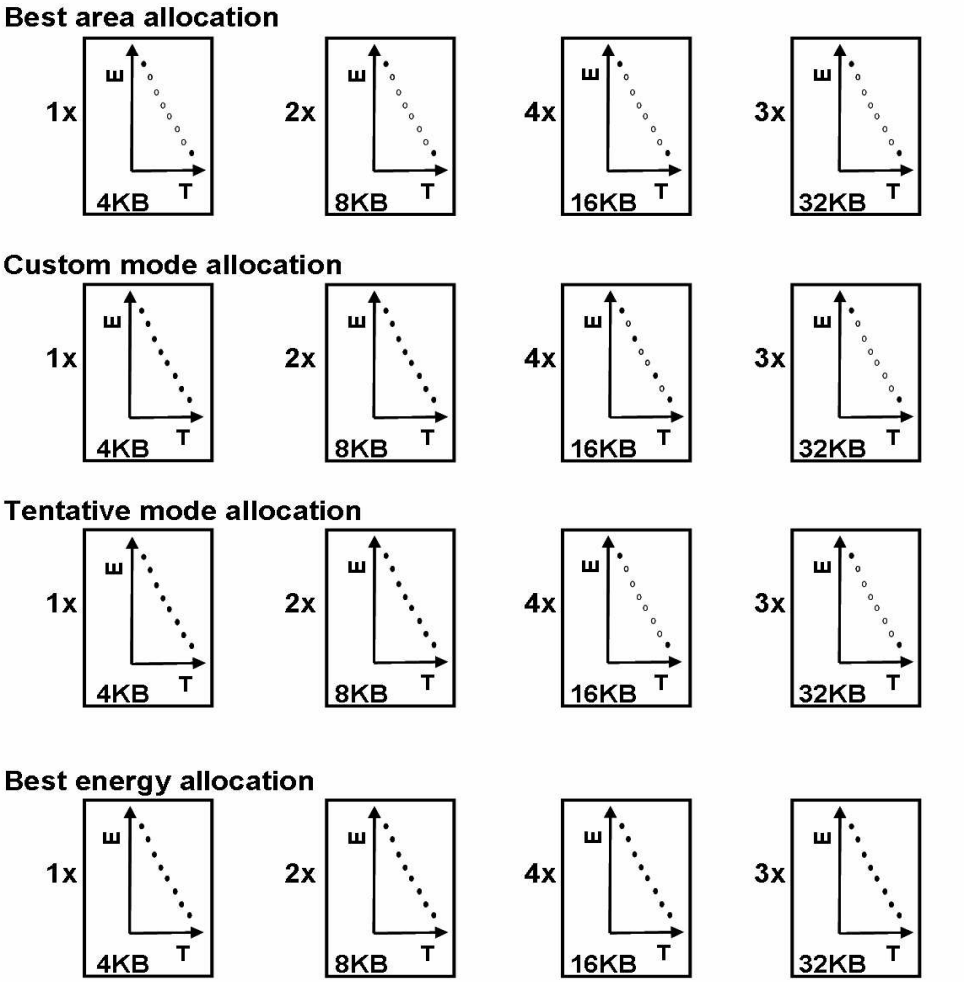


Figure 6.2: Distribution of memory modes in the different allocations tested.

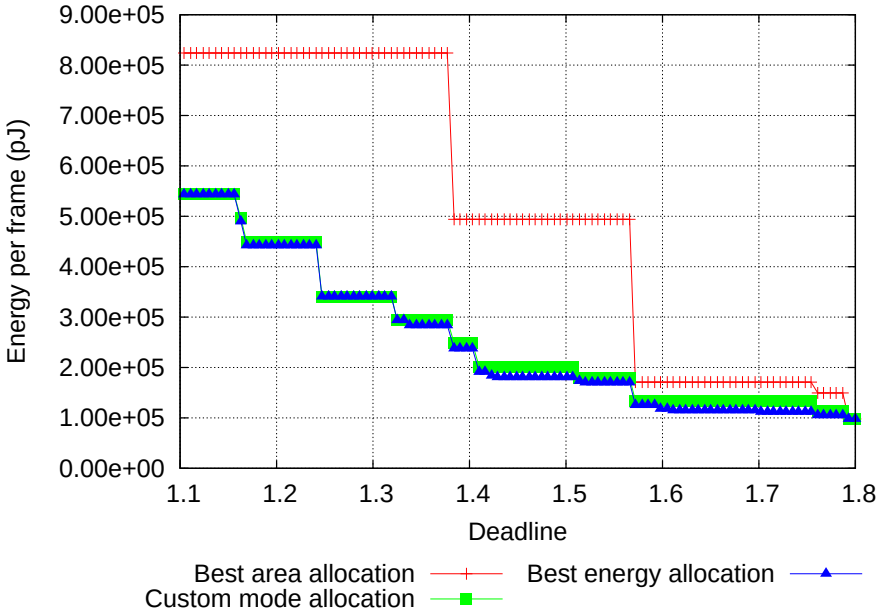


Figure 6.3: In terms of energy, the solution provided for Algorithm 6.1 is quite similar to the best allocation tested with a lower impact on area.

The results obtained with the mode allocation algorithm presented here, for MP3 decoder in any of its versions – original version, scenarios –, confirm the ideas extracted from the previous chapter: a large number of modes is more convenient in small memories, since they support the largest workload in the memory system. The number of modes implemented in large memories is minimum instead.

6.5.2. Statistical Design-Time Exploration

As mentioned in previous sections, variability effects on memories have been modeled by transistor-level Monte Carlo simulations using a 65nm BSIM model. These simulations allow us to characterize latency and energy per access of each memory mode with a certain probability distribution. In order to simulate a more

dynamic behavior in our benchmark application, we assume that the time to decode a frame may change at run time, being shorter than its original deadline establishes.

The output of the design time exploration is a global memory configuration (potentially, a set of configurations) for every expected deadline. As these configurations are used on a specific device affected by variability, it is possible that in some cases the configurations do not fulfil their timing constraints at run time. In our statistical methodology, a configuration can only be considered as a feasible solution if its probability of fulfilling the deadline under variation is at least equal to the design parameter denoted as P_T (Equation 6.15). Three different values for this parameter have been studied in this research:

- $P_T = 0.5$

  The resulting configuration guarantees statistically that under variation, it will be able to fulfil its time requirement half of the times

- $P_T = 0.8$

  The configuration will be operative in 80% of the occasions

- $P_T = 0.99$

  This last value for P_T represents a highly conservative approach for our statistical technique, since every configuration fulfils its deadline in 99% of the injections simulated. This demanding version is used to compare our technique with other traditional worst-case approaches.

In all cases, parameter P_E has been set at 0.5 (Equation 6.16). The sets of global configurations obtained for these three P_T values are named *stat50, stat80* and *stat99* respectively.

The configurations obtained by means of our *Statistical Design-Time Exploration* are compared with configurations from traditional approaches. Denoted as *base*, we represent the configurations obtained at design time following traditional worst-case estimations, i.e. *system scenarios* are not considered but the original version of the MP3 Decoder [Lag01], neither statistical estimations at design time, and the energy and delay values for each mode are assumed to be the largest that can be expected under variations. Regarding the memory architecture, we assume configurable modules for *base*, being also applied our memory mode allocation algorithm with the result mentioned above. An exhaustive search is performed at design time to establish the configurations which meet the deadlines with such energy/delay values. It is the most conservative technique to guarantee performance although at the expense of a high energy cost. Likewise, we consider the optimal configurations obtained at setup time by exhaustive search. This result, denoted as *optimum*, represents the ideal energy-delay solution for each variability injection, as well as the theoretical upper bound for optimization. This approach corresponds with the one presented in Chapter 5. We also consider an intermediate case, resulting from adding to *base* the scenario characterization, *noStat*. The memory architecture simulated for *noStat* and *optimum* is exactly the same – in number of memories, sizes and modes – to the one used in our statistical technique, as we are considering the scenario-aware implementation in these cases. As we mentioned in the previous chapter, the lack of scenarios in *base* gives rise to a different memory allocation and data assignment. All these cases have been performed under a common set of deadlines, to later on applying the resulting global configurations on a memory architecture affected by a large set of variability injections. The impact of variability can be seen in Figure 6.4, which plots the energy consumed by each

global configuration versus its execution time, considering all the variability injections simulated. Mean values are represented by points while the total length of the error bars is two times the standard deviation. As we see, by means of parameter P_T the methodology proposed takes well under control the number of cases where a configuration do not fulfil its deadline. Thus, *stat99* limits the failures to less than 1% of the cases, while *stat50* ($P_T = 0.5$) configurations guarantee deadlines in 50% of the occasions. Although, for the sake of clarity, results for *stat80* are not plotted, a similar behaviour is exhibited, missing less than 20% of the deadlines. For *Stat50* this is graphically represented by the fact that mean values in time appear over the deadlines. Regarding energy consumption, the trend follows the intuition since higher performance guarantees, as the ones offered by *stat99*, come at extra energy expenses if we compare them with more lenient configurations as the ones represented by *stat50*. Despite this, our conservative solution represented by *stat99* improves the energy of *noStat* approach, which becomes extremely conservative in time due to the absence of statistical knowledge. Comparing *Base* with *noStat* we can see the huge impact on energy due to the presence of *system scenarios*. Regarding time, *base* is in most of the cases as conservative as *noStat*.

As expected from the methodology, the number of cases where configurations do not meet their deadlines is well controlled by P_T. Regarding energy consumption, the trend follows the intuition since higher performance guarantees come at extra energy expenses. Thus, configurations labeled *stat99* always consume more energy than those labeled *stat80*. Likewise, *stat80* configurations also consume more than *stat50*. This trend is kept independently of the deadline, although the energy differences are reduced as the deadlines are more relaxed.

Another significant aspect shown in Figure 6.4 is related to the excessive conservatism that traditional approaches provide to their solutions at design time. In order to avoid any risk of missing deadlines, configurations guarantee the performance at a high cost in energy. Our statistical method can offer a similar guarantee at a much lower energy cost. As we show in the figure, unless conservative solutions are required, less energy demanding configurations can be obtained with our *Statistical Design-Time Exploration*, as *stat50* illustrates.

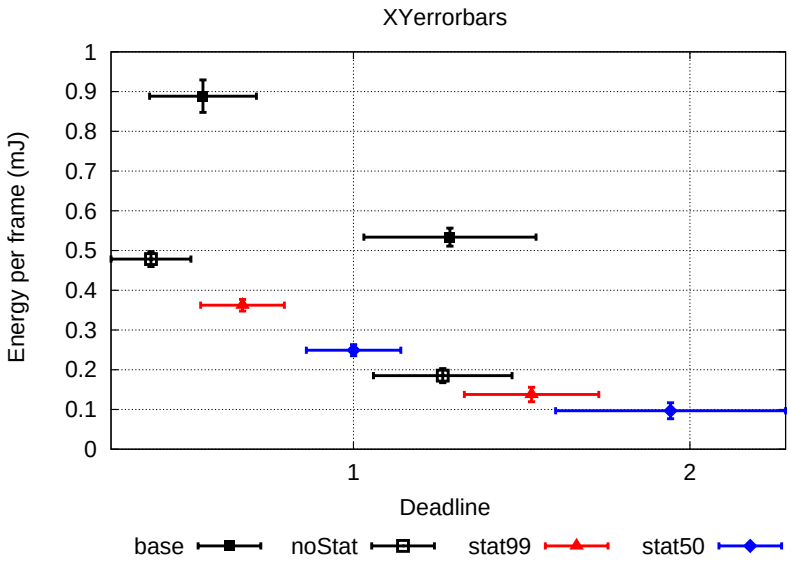


Figure 6.4: High confidence intervals generate more conservative configurations to keep the system reliable in time.

Another trend, not so intuitive, can be observed in Figure 6.4 regarding the effects of process variation on energy and time. Its impact is much larger in energy than in time, as well as larger on the *slow* configurations than on their *fast* counterparts. Consider for instance the *stat50* configurations set. For the shortest deadline shown in the figure, the energy error bar size is about 12% of the corresponding

mean value, while for the less strict deadline it grows to about 26%. Although the figure suggests that this trend is more relevant for time, this is actually a result of the different scales used on the energy and time axis. For the shortest deadline, the size of the time error bar is about 2% of the mean value, while for the largest it only grows to about 3%. The variation in time is kept in a similar range in the other configurations plotted. However, the range of variation in energy depends on the approach used, being higher for the conservative *base* case. Ranges similar to *stat50* can be found in *stat80*, whilst *stat99* and *noStat* exhibit a energy variation lower than 10% in any deadline.

As a result of variations during the manufacture process and our statistical approach to cope with them, a controlled amount of devices will not be able to fulfil some of their deadlines by directly using the configurations generated at design time. In these cases, in order to guarantee the fulfilment of every single deadline in a particular device, the system can be easily checked by monitoring the application execution once and applying a simple post-process adjustment as follows: for each target deadline, in case the corresponding configuration obtained at design time does not meet it, the system must assign a new global memory configuration. The new solution is among the configurations already obtained at design time. Most of the time, this configuration corresponds to the next *tighter* deadline. Just in case variability is too large, the configuration selected will be one established for an earlier deadline. As an example, considering *stat50* in Figure 6.4, half of the simulations for the second deadline plotted would use the configuration found by the *Statistical Design-Time Exploration*, while almost the other half would use the configuration obtained for a previous deadline. The cases where it is necessary to use even more conservative configurations would be almost negligible. The percentage of situa-

tions where it is necessary to change the initial configuration is reduced to 80% for devices using *stat80* configurations.

The impact in energy of this adjustment to guarantee the deadlines is summarized in Figure 6.5. For each of the approaches already described on Figure 6.4 and an extended set of deadlines, we show for each case the mean energy consumed by hundreds of different simulated devices, each one attached to a different injection of variability. The energy values shown in this figure are all relative to the *base* set. Note that all the strategies lead to 100% of meet deadlines, since the post-manufacturing step update the configurations when needed as explained above.

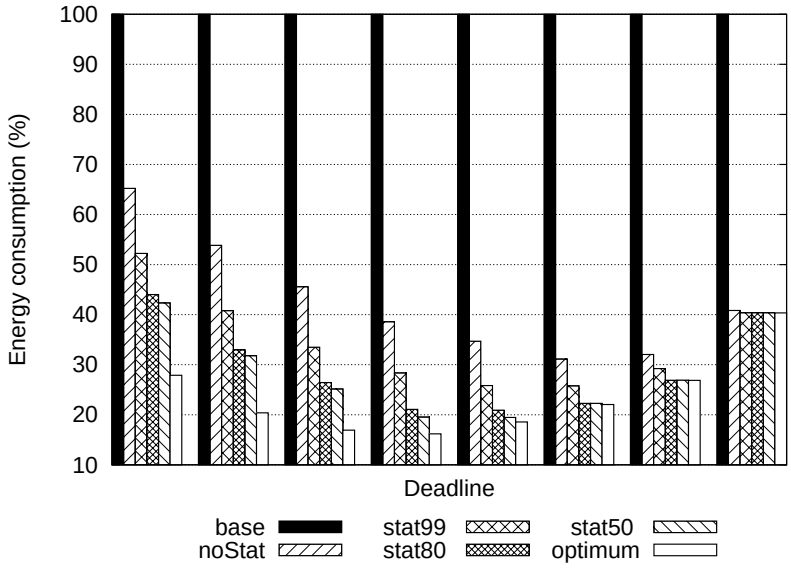


Figure 6.5: Average energy seems to benefit lower confidence intervals, although at the expense of a necessary re-arrangement at setup to meet each deadline.

In this case, our methodology provides large energy savings compared to traditional worst-case designs. For instance, energy consumption is reduced in *stat50* in a range from 58% to 80% depending on the deadline. Similar savings are achieved

by *stat80* with a smaller adjustment process at setup. If design is performed using $P_T = 0.99$ (*stat99*), the adjustment process is seldom carried out and savings are larger than 50% in most of the cases. Again, we see the significant energy savings due to scenarios (*noStat*), more than 30% compared to *base*. These savings increase up to 10% more by using our probabilistic approach. For very relaxed deadlines, all approaches based on scenarios, including the static design *noStat*, tend to configure memories in the same way (the slow mode) and little differences in savings are available. The same happens for very tight deadlines, not shown in the figure. As a consequence of the different memory architecture derived from an unaware scenario version, energy costs in *base* are larger even for relaxed deadlines. Scenario exploitation improves the energy efficiency on memories.

6.5.3. Setup refinement results

Even when we have significantly improved worst-case design results, for tight deadlines or conservative P_T values, we are still far from optimal configurations. This is reasonable for a design time technique, since the parametric variations in energy per access may be as large as 40%. Thus, a run time (or at least, setup time) step is needed to bridge this gap and avoid building an accurate model for process variation at design time.

Applying the *Setup Refinement* directly to the configurations obtained from the *Statistical Design-Time Exploration* we can improve the results, as Figure 6.6 shows. This refinement step is also applied to *noStat*). Results are normalized using as baseline the results from *base* already seen in Figure 6.5, since the refinement is not applied there. *Optimum* configurations are the same. It can be noticed a sig-

nificant energy reduction in *noStat*. In this case, solutions consume up to 25% less energy than before the refinement. Among the different statistical configurations, energy reductions are larger for *stat99* than for *stat50*. Most of the deadlines simulated exhibit reductions between 10% and 15% in *stat99*, while these percentages decrease to 5% for the other two statistical configurations (*stat80*, *stat50*). After the refinement process, the trend already seen in Figure 6.5 remains, i.e. statistical solutions are more energy efficient and *stat50* shows the closest results to the optimal ones, improving *base* results above 60% in most of the cases. The number of steps involved in the refinement process varies from one approach to another. This number is higher for *noStat*, whilst it is shortened as the *Statistical Design-Time Exploration* is applied and generates configurations less time demanding. As this technique is carried out at setup time, it can be necessary to limit its duration. Figure 6.6 illustrates results without applying any limitation to the refinement process. If we limit all the cases to a reduced number of steps, which is only extended in case this number is too reduced to fulfil the deadline, we obtain the results plotted in Figure 6.7. In this situation, *noStat* and *stat99* are the most affected by the limit as they require more steps (between 3 and 4 times more steps) to improve the initial configurations. The others are able to achieve similar results to the ones obtained without limitations as the total number of steps is slightly higher than the limit imposed.

6.6. Extended experimental results for MP3 Decoder

In this section we provide a final comparison, using once more the MP3 Decoder, where the energy savings reported in the previous can be improved a bit more. Here we compare our methodology with a more rigid worst case. As new

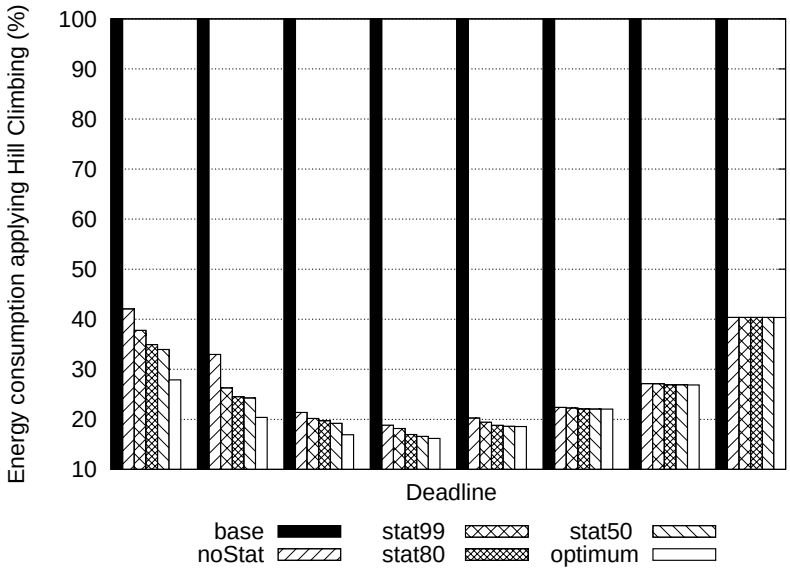


Figure 6.6: Average energy consumption after the refinement is applied to the *candidates*.

base we consider the scenario-unaware implementation [Lag01] already used in the previous section, assuming in this occasion that only two modes are available in each configurable memories. Once more, conservative estimations in energy and delay on memories are considered to deal with process variation, and the memory configurations to use at run time are generated at design time, without further post-process.

Figure 6.8 summarizes the results obtained in this case. Energy results are normalized to results from the new *base*. The scenario version is simulated again for the three P_T values used along this research, so results keep the notation already known: *stat50*, *stat80* and *stat100*. The same set of deadlines already shown for this application has been simulated here for both versions. As a result, we can see

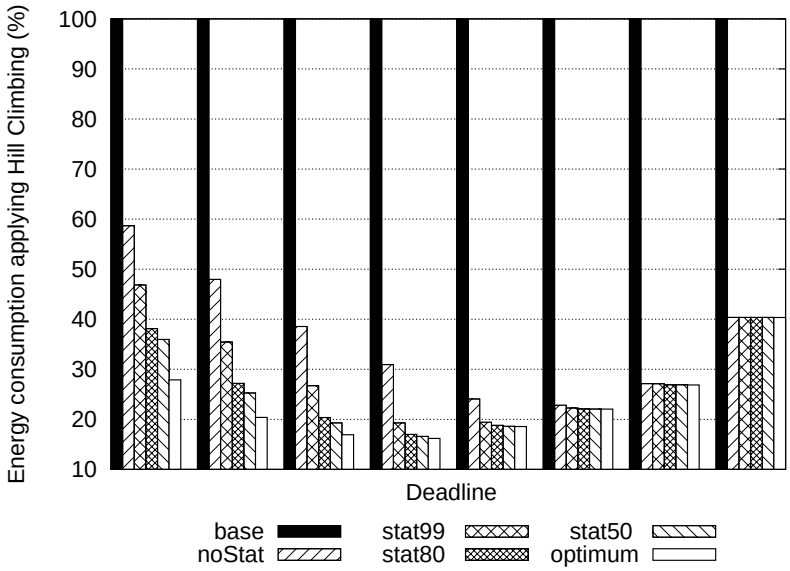


Figure 6.7: Average energy consumption after limiting the number of steps performed by the refinement process.

in Figure 6.8 energy savings higher than 70% comparing the new *base* with any other solution.

Regarding time constraint fulfilment, the MP3 implementation represented by *base* is characterized by a larger number of cycles, so under variation some of the deadlines cannot be fulfilled at run time by *base*. However, the scenario version can fulfil them sucessfully. The use of scenarios and the improvement in the code implementation associated to them not only help to reduce energy consumption but also to increase timing requirement fulfillment.

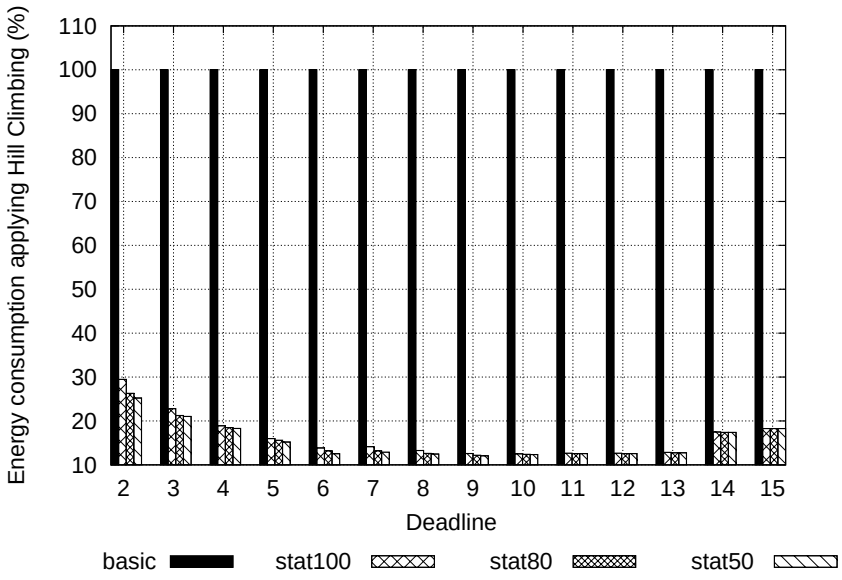


Figure 6.8: Energy difference comparing our full methodology with an MP3 version lacking of *system scenarios* and a reduced number of modes per memory.

6.7. Beyond MP3. Extended results

The methodology developed along the chapters of this dissertation has been invariably applied to an MP3 decoder as illustrative application driver. However, this methodology is not focused on particular applications, but on specific domains, such as multimedia applications.

In this section, once the entire methodology has been already explained, we extend its application to other benchmarks. In order to shorten the very early steps accomplished at design time, such as application profiling, scenario clustering, memory allocation or data assignment, we have developed a collection of synthetic benchmarks. To generate these benchmarks we have considered the MP3 application as reference. Each synthetic benchmark consists of a set of sequential tasks

whose information comprises the cycles required for each task, the memories involved in the tasks as well as the accesses performed to each memory in each task. Memory architecture and data allocation are included in the benchmark description.

The number of tasks and memories existing in each benchmark is randomly generated using as pattern the profile extracted from MP3. The distribution of memory accesses as well as the number of cycles per task has been carried out by considering these characteristics as normally distributed at the MP3 decoder. Similarly to MP3, memory architectures consist entirely of 4KB, 8KB, 16KB and 32KB modules. As a result of these assumptions, the most accessed memories in the system are also the small ones.

The memory mode allocation algorithm (Section 6.2) has been applied to each of the synthetic benchmarks developed, providing the same results already reported for MP3: memories of 4KB and 8KB, considered as small, include eight modes; intermediate modules comprise four modes, whereas the largest memories – 32KB – consist of two modes only.

Once the benchmarks and their associated memory architectures are fully defined, the statistical model is applied to each one. Global configurations are obtained for the same three values used in MP3 for parameter P_T: 0.5 (*stat50*), P_T: 0.8 (*stat80*) and 0.99 (*stat99*). Once more, value $P_T = 0.99$ allows us to compare our statistical method with the design time conservative approaches where deadline fulfilment must be always guaranteed under any variability situation. Regarding P_E, its value is always kept at 0.5. Similarly to the MP3 decoder, we denote as *noStat* the configurations obtained by means of worst case estimations on energy and access time for the modes of each memory module in the architecture. Unlike MP3, *noStat* configurations are used as baseline in these benchmarks. For comparison purposes,

for each deadline we consider as *best* solution the configurations obtained at setup time once the effects of variability are known, representing the theoretical upper bound for optimization. From the collection of synthetic applications developed, we show below the results for five of them, named from *Benchmark 1* to *Benchmark 5*, analyzing in detail the results for one of them, *Benchmark 1*. The results, plotted from Figure 6.9 to Figure 6.13, are shown in a similar way to MP3. Thus, Taking *Benchmark 1* as example, Figure 6.9(a) represents the execution time and energy consumed under variations by each one of the global configurations developed at design time. As we are considering a wide range of variability conditions at run time, these results are plotted as pairs of time-energy mean values, with error bars two times the standard deviation. For the sake of clarity, we show a reduced number of deadlines and its corresponding configurations. As observed, configurations generated at design time accomplish the confidence interval fixed by parameter P_T, clearly illustrated in Figure 6.9(a) for *stat50*, specially for tight deadlines, where configurations fulfil the deadline in 50% of the cases. After considering a wide set of variability injections, mean values for time fall right over the deadlines. Similarly to MP3, as the need for guaranteeing performance increases, more conservative solutions are generated and higher costs in energy appear. The savings in the figure are remarkable in terms of energy obtained for $P_T = 0.99$, comparing with the conservative approach represented by *noStat*. Energy reductions in *Benchmark 1* are over 10% for demanding timings, but savings can reach up to 50% for less strict solutions, such as *stat50*. These differences show how pessimistic conservative approaches are. As we obtain a global configuration for each possible deadline, in case of not being able to meet a deadline by means of its correspondent configuration, which happens mainly at *stat50*, configurations are redistributed at setup time

to ensure performance. To be closer to *best* results, specially for strict deadlines, we can improve results in Figure 6.9(b) by applying the *Setup Refinement* in all cases (*noStat* and *stat*).

In all these figures, we can see the same trend regarding the statistical configurations. After any kind of refinement to guarantee performance, *stat50* provides better results than the other approaches.

Analyses similar to the one detailed here for *Benchmark 1* can be done for benchmarks 2 to 5.

6.8. Conclusions

It has been presented in this chapter a complete methodology to deal with process variation at memory level and dynamism at application level, paying special attention to the methods developed at design time and setup to deal with variability.

Firstly, it has been shown a technique which determines the suitable number of modes that each memory in the system should have. This technique performs a trade-off between the area penalty and the energy gains expected by adding new modes. The best ratio energy-area found with this technique establishes the number of modes to implement in each memory of the system. Our technique provides more modes to small memories, decreasing its number as the memory size increases. This is a consequence of the memory allocation and data assignment steps carried out at design time, which turn these small memories into the most accessed modules in the system.

Secondly, it has been presented a general statistical approach to face variability on memory systems. This approach is composed of two main steps: *Statistical*

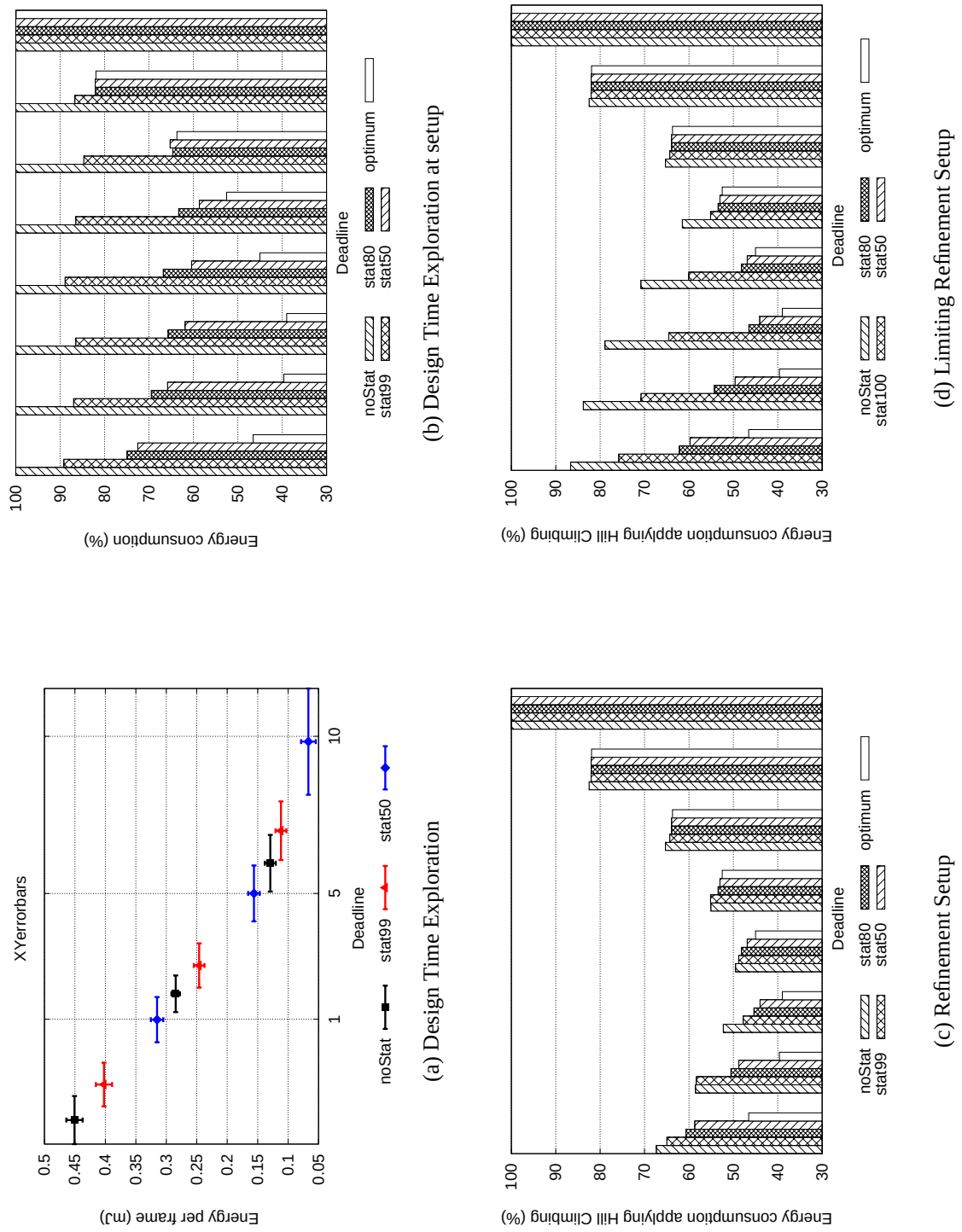
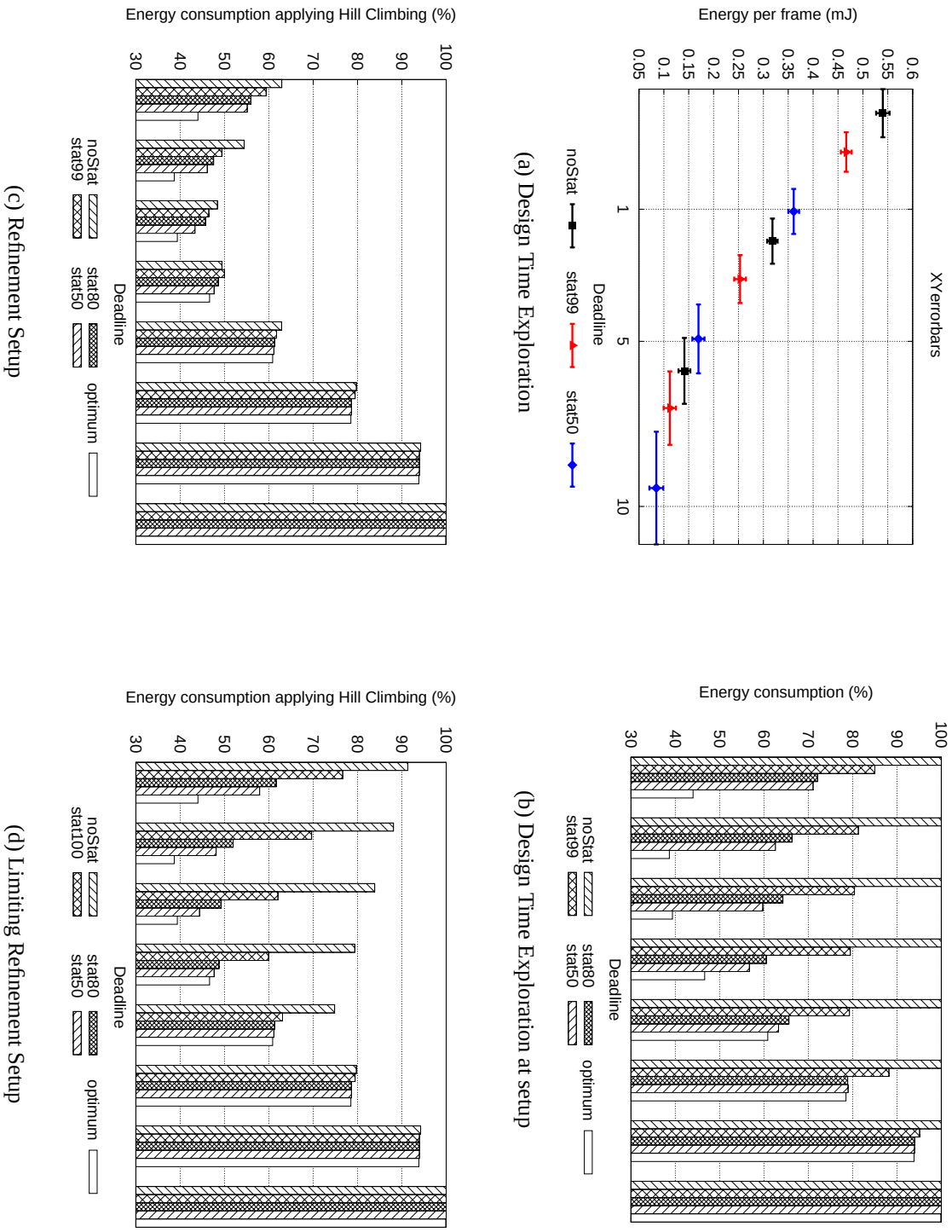


Figure 6.9: Benchmark 1



(a) Design Time Exploration

(b) Design Time Exploration at setup

(c) Refinement Setup

(d) Limiting Refinement Setup

Figure 6.10: Benchmark 2

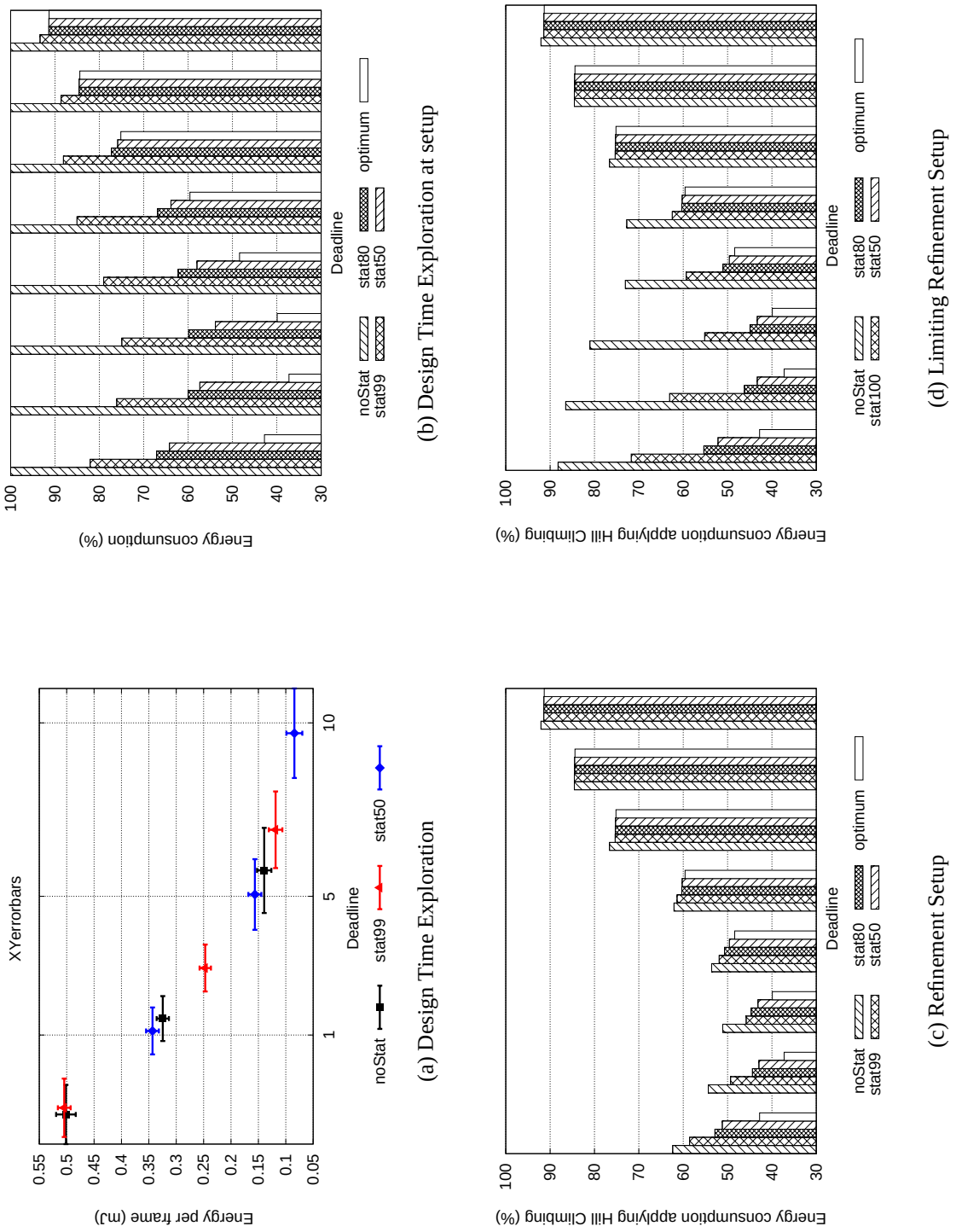
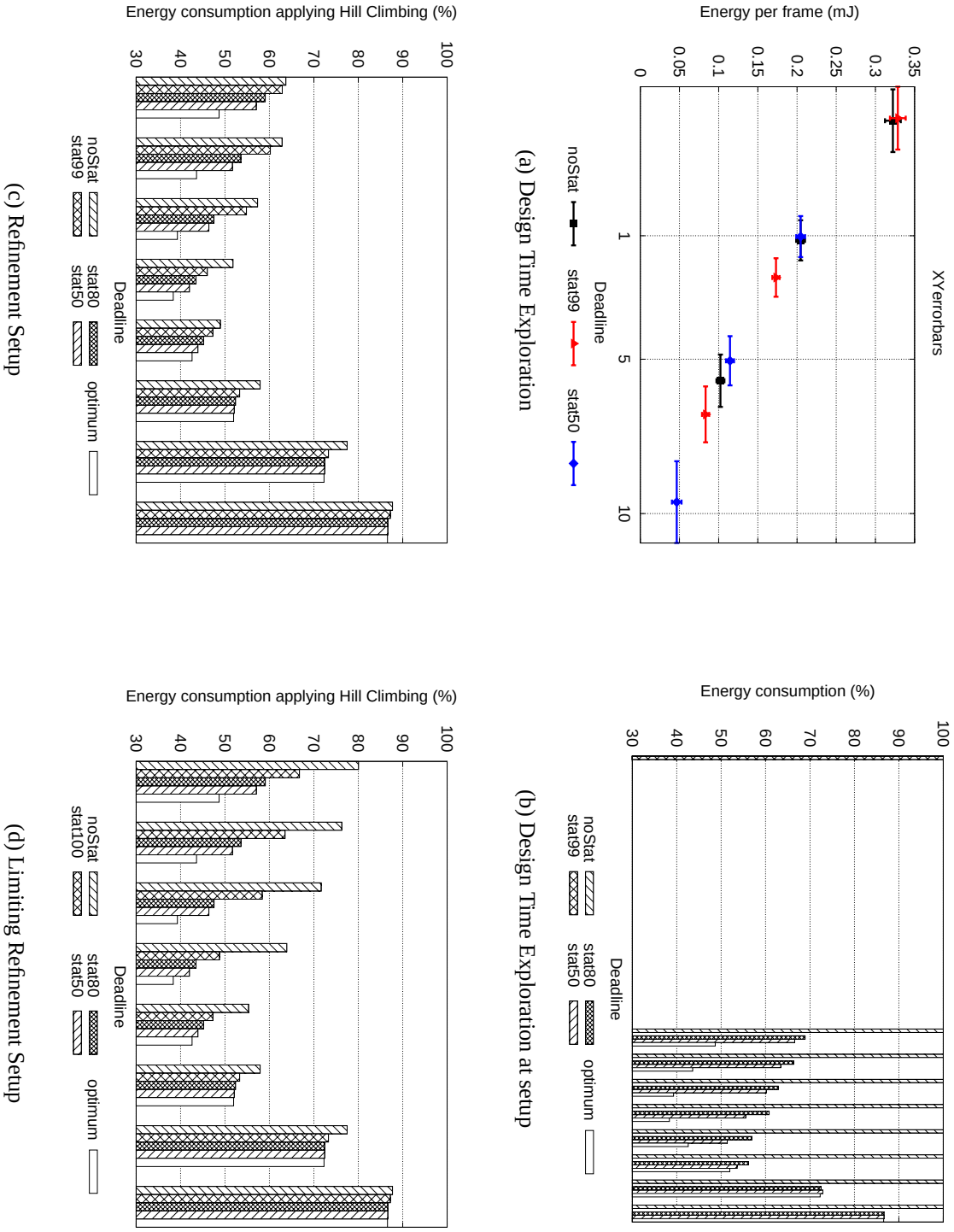


Figure 6.11: Benchmark 3



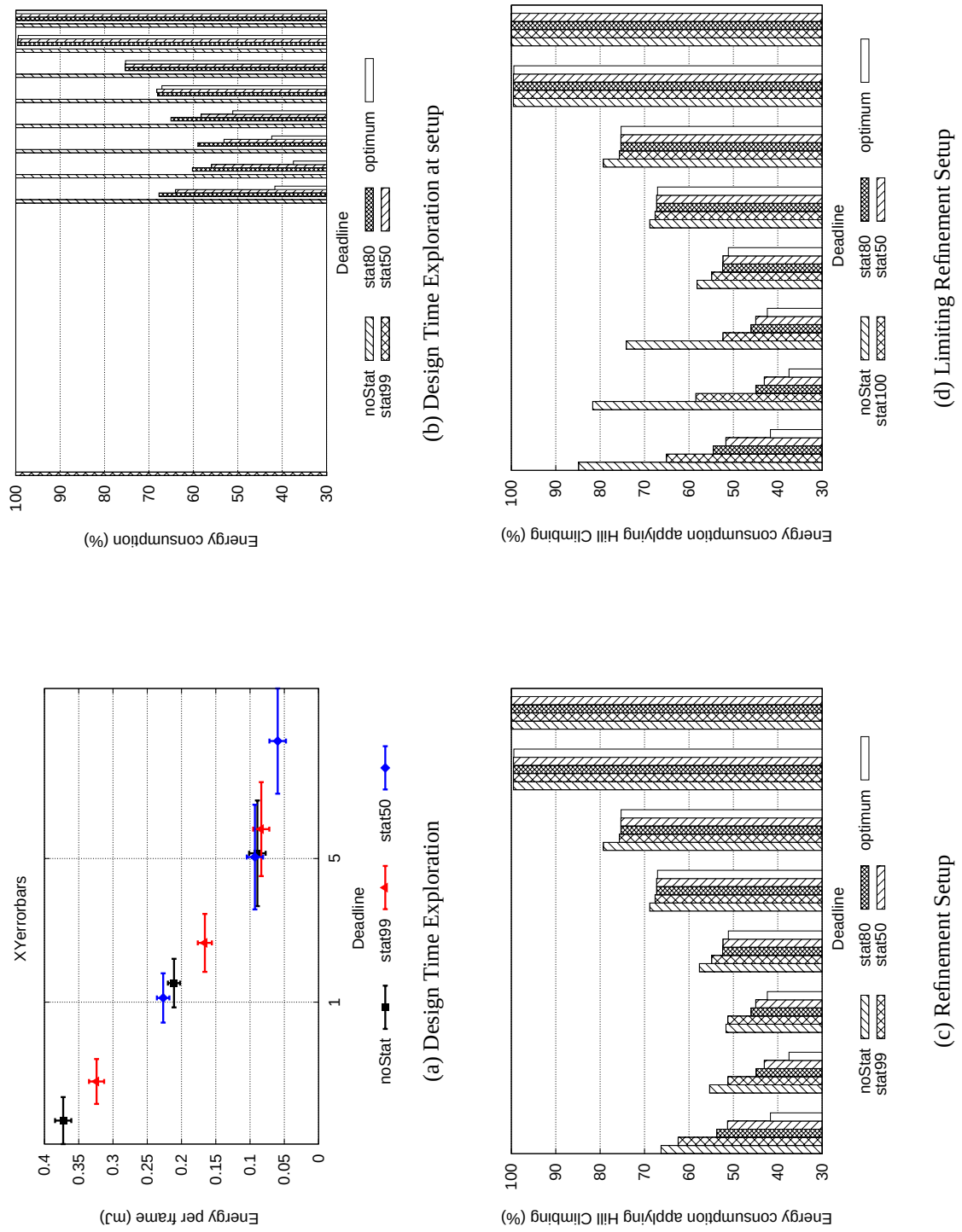
(a) Design Time Exploration

(b) Design Time Exploration at setup

(c) Refinement Setup

(d) Limiting Refinement Setup

Figure 6.12: Benchmark 4



(a) Design Time Exploration

(b) Design Time Exploration at setup

(c) Refinement Setup

(d) Limiting Refinement Setup

Figure 6.13: Benchmark 5

Design-Time Exploration and *Setup Refinement*. The former step performs a design space exploration to exploit the statistical knowledge about how variability affects design parameters in multi-modal memories. The outcome of this exploratory stage is a set of global configurations which establish for each particular task the mode each memory has to be set. This technique allows a reduction in the number of devices that does not meet their time constraints because of variability effects. Furthermore, this improvement is carried out at design time without the energy overheads associated to traditional worst-case techniques. Being conservative with the statistical estimations, our experimental evaluation on MP3 shows energy reduction benefits over conservative designs higher than 30%. Less strict performance requirements lead to reductions larger than 50%.

The latter step, *Setup Refinement*, provides a fast and efficient mechanism to improve the system when actual memory parameters are already established at setup time. It is based on a gradient search algorithm to improve the execution time and energy consumption of the configurations obtained at *Statistical Design-Time Exploration*. This technique adapts the system to the actual injection of variability, ensuring that final configurations meet the timing constraints. Results show the effectiveness of this algorithm, moving the original solutions closer to the theoretical optima.

Finally, at *run time* the system is monitored, so that the final pre-stored configurations can be dynamically selected when conditions change. Furthermore, when substantial changes are detected, for instance in case of ageing, the *Setup Refinement* phase could be triggered to perform a fast re-adaptation of the system.

Results show that it is possible to avoid, at design time, overestimations which would afterwards lead to energy and performance overheads. Among other factors,

these penalties are caused by the way application dynamism and variability are managed. Our statistical model, developed to consider variability effects at design time, provides interesting energy-performance tradeoffs, guaranteeing performance at a lower energy than traditional solutions. Similarly, scenarios delimit the dynamism exhibited by applications at run time, enhancing its management. The presented methodology has been also tested with a set of synthetic benchmarks. These applications reproduce similar results to the ones obtained for our driver application, the MP3 decoder.

We can conclude this chapter by highlighting the fact that configurations generated under not very demanding performance requirements, such as *stat50*, are finally more energy efficient in case the cost associated to the refinement process carried out at setup time is affordable. Otherwise, a more feasible trade-off between energy savings and refinement can be also statistically found, such as *stat80*.

7

Conclusions

Over the years, technology enhancement has made possible a continuous improvement in the characteristics present in SoC and SiP. Performance, area, memory and power are just some of the characteristics which have taken advantage of such enhancement. Technological scaling, specially under sub-micron dimensions has strongly contributed to this improving.

Despite the clear benefits provided by scaling, new challenges have also emerged. In this dissertation, we have dealt with two of those problems, each one of a different nature: process variation and dynamism.

Process variation, a phenomenon associated to a lack of control over technological parameters at design time, was not new for designers, which were able to deal with it in the past. However, as a consequence of the extreme technology scaling carried out in the last decade, it has quickly emerged as a key issue to deal with.

If process variation is a technological issue, dynamism is a more recent characteristic exhibited at application level. The continuous technology improvement and the market requirements have promoted the development of complex and de-

manding applications to be executed on devises based on SoC/SiP platforms. Such devices, traditionally characterized by restrictions, mainly related to power, have to deal with applications whose behaviour at run time is largely variable and hardly predictable.

These two issues are the challenges faced along the chapters of this dissertation. Both topics lead to an uncertain behaviour at run time, at platform level and application level, which has been traditionally managed by means of different design time methodologies in order to guarantee performance, although at the expense of a high energy cost. In this research we have developed a methodology to deal with both sources of uncertainty in a coordinated way, so the timing requirements are fulfilled and the energy consumption is drastically reduced. We have focused the research on the memory architecture, since it is one of the most variability affected components of the system. This methodology is focused on specific domains, such as multimedia applications, rather than general purpose systems or specific applications since the characteristics of a well-defined domain can be exploited in a more successful way.

The pillars of the methodology proposed are the use of configurable memories and the concept of *system scenarios*. Configurable memories allow us to deal with process variation by implementing a variable number of operating modes which generate an energy-delay trade-off, avoiding pessimistic estimations at design time over mono-mode memories. *System scenarios* allow to describe an application as a collection of different run-time behaviours. Once these behaviours are characterized, the applications become more predictable, reducing also the use of conservative solutions to keep performance. These two ideas are combined in a methodol-

ogy which is performed along the whole platform lifetime: from its design until run time.

The main aspects which characterize the methodology developed are listed next:

- Applications are profiled and *system scenarios* are extracted
- The memory architecture is energy-aware and the application profile is used to improve the memory allocation and data assignment
- Memory modules are all configurable. The suitable number of modes per memory is set at design time
- Energy and performance depend on the way the memory architecture configures its modules, i.e. the mode in which each memory will operate at run time
- The impact of process variation on memories is statistically modeled for each memory and each mode existing in the system. A study at design time determines statistically the way the memories must be reconfigured to meet application timing constraints with the lowest energy penalty. As application is split in scenarios, this step is carried out for each scenario
- Before the application begins its execution at run time, design-time memory configurations for each scenario can be upgraded and refined with the current information about variation
- At run time, the application is monitored to determine the scenario which is currently running and reconfigure the memories according to this scenario

Results show significant savings in energy regarding worst case estimations. For MP3, cumulative savings up to 85% have been reported. The scenario awareness, the *Statistical Design-Time Exploration* and the *Setup Refinement* contribute to reach remarkable energy savings.

The methodology developed along this research has proved significant advantages regarding conservative designs. We deal with process variation and application dynamism in a coordinated way. This cooperation reports meaningful savings in energy. Besides, the use of *system scenarios* allows to improve performance.

7.1. Future work

Along the different chapters we have mentioned the two main issues to be considered in future work as an extension to the methodology presented:

- The introduction of switchable buffers [WMDC09] instead of the current configurable buffers
- The adaptation of the methodology to consider time-dependent variations at run time

The use of switchable buffers would add a new system knob, the supply voltage, to provide larger energy/delay trade-offs. The operating points in current buffers are based on two parameters – the number of stages N and the sizing factor f_i – which are largely independent of the technology node used. However, the supply voltage is process dependent, being necessary to study the impact of the continuous technology scaling on this parameter.

Regarding time-dependent variations, our current methodology provides limited setup/run-time adaptation possibilities to calibrate the design-time configurations and cope with these variations. In order to do it, it would be necessary to develop a model about the way degradation affects memories and its implications on the configurations obtained at design time. The overhead introduced by considering temporal degradation at run time would also have to be taken into account.

Appendix A

Methodology compensation based on a branch and bound algorithm

The exhaustive algorithm 5.3 shown in Section 5.2.1.2 as a compensation mechanism at task level based on memory mode and frequency adjustment (named as TLC) can be too expensive in time just with a small increment in the number of tasks or modes per memory, as the exploration space grows exponentially. Taking as example the scenario version of the MP3 decoder consisting of thirteen tasks and the memory architecture which involves ten memory modules, the number of different Pareto configurations that the exhaustuve algorithm should consider is $(modes^{memories})^{tasks}$. This means 'slightly over' 8K options assuming that each memory module has two modes, but over 5e+11 combinations if each memory consists of eight modes.

Instead, a branch-and-bound algorithm can be designed. At every step, we pick one task and we try all the possible memory modes. However, to reduce the time complexity we introduce a bound function (*is_admissible()*) which determines when

a given branch should not be further explored. We estimate the bounds of a potential solution by combining the cost of the partial solution already built with an optimistic estimate of the energy and delay for the unset tasks. This bound function reduces the exploration time around 70% over exhaustive search.

Algorithm A.1 shows the pseudo-code for the alternative implementation of the compensation technique denoted as TLC.

Algorithm A.1 Task-level compensation (TLC)

```
input: scenario, tasks_list
output: Pareto_curve
for each deadline
    TLC_point(tasks_list , deadline );


TLC_point(tasks_list , deadline )
    for each task t in tasks_list
        for each memory_mode mode
            task[t]=mode ;
            if (all_tasks_are_set())
                  calculate_energy_delay();
                  if (delay < deadline )
                     update_possible_solution();
            else
                if (is_admissible())
                   TLC_point(tasks_list , deadline );
    add_TLC_point();
```

Appendix B

Publications

- *Statistical approach in a system level methodology to deal with process variation*

  Concepción Sanz Pineda, Manuel Prieto, José Ignacio Gómez, Christian Tenllado and Francky Catthoor, International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010, pp.115-124

- *System-level process variability compensation on memory organizations: on the scalability of multi-mode memories*

  Concepción Sanz Pineda, Manuel Prieto, José Ignacio Gómez, Antonis Papanikolaou and Francky Catthoor, Asia and South Pacific Design Automation Conference (ASP-DAC), 2009, pp.254-259

- *Dealing with unpredictability through a system-level methodology*

  Concepción Sanz Pineda, Manuel Prieto, José Ignacio Gómez, Antonis Papanikolaou and Francky Catthoor, Design of Circuits and Integrated Systems Conference (DCIS), 2008, pp.1-6

- *Combining system scenarios and configurable memories to tolerate unpredictability*

  Concepción Sanz Pineda, Antonis Papanikolaou, Manuel Prieto, José Ignacio Gómez, Miguel Miranda and Francky Catthoor, ACM Transactions on Design Automation of Electronic Systems (TODAES), 2008, pp.1-7

- *System-level process variability compensation on memory organizations of dynamic applications: a case study*

  Concepción Sanz Pineda, Antonis Papanikolaou, Manuel Prieto, José Ignacio Gómez, Miguel Miranda and Francky Catthoor, Proceedings of the Seventh International Symposium on Quality of Electronic Design (ISQED), 2006, pp.376-382

Under review:

- *System-level memory management based on statistical variability compensation*

  Concepción Sanz Pineda, Manuel Prieto, José Ignacio Gómez, Christian Tenllado and Francky Catthoor, ACM Transactions on Embedded Computing Systems (TECS)

Bibliography

[AAA+07] P. Marchal A, B. Bougard A, A. Papanikolaou A, M. Mir, F. Catthoor A B, and W. Dehaene B. A designer's perspective on future memory architectures for software defined radios. In *Proceedings 2nd International Conference on Memory Technology and Design - ICMTD*, pages 25–28, 2007.

[AAM+09] A.M. AbdelHamid, A. Anchlia, S. Mamagkakis, M. Corbalan Miranda, B. Dierickx, and M. Kuijk. A standardized knobs and monitors rtl2rtl insertion methodology for fine grain soc tuning. *Digital Systems Design, Euromicro Symposium on*, 0:401–408, 2009.

[ABZ03] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical timing analysis for intra-die process variations with spatial correlations. In *ICCAD '03: Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 900, Washington, DC, USA, 2003. IEEE Computer Society.

[AH00] B.S. Amrutur and M.A. Horowitz. Speed and power scaling of sram's. *Solid-State Circuits, IEEE Journal of*, 35(2):175 –185, February 2000.

[AJB+05] T. Vander Aa, M. Jayapala, F. Barat, G. Deconinck, R. Lauwereins, H. Corporaal, and F. Catthoor. Instruction buffering exploration for low energy embedded processors. *J. Embedded Comput.*, 1:341–351, August 2005.

[ASRV05] M. Alvarez, E. Salami, A. Ramirez, and M. Valero. A performance characterization of high definition digital video decoding using h.264/avc. In *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*, pages 24 – 33, 2005.

[ATO] ATOMIUM. http://www.imec.be/design/atomium/.

[Aus99] T.M. Austin. Diva: a reliable substrate for deep submicron microarchitecture design. In *Microarchitecture, 1999. MICRO-32. Proceedings. 32nd Annual International Symposium on*, pages 196 –207, 1999.

[BC11] S. Borkar and A.A. Chien. The future of microprocessors. *Communications of the ACM*, 54:67–77, May 2011.

[BESB94] D. Burnett, K. Erington, C. Subramanian, and K. Baker. Implications of fundamental threshold voltage variations for high-density sram and logic circuits. In *VLSI Technology, 1994. Digest of Technical Papers. 1994 Symposium on*, pages 15 –16, June 1994.

[BKD04] S. Borkar, T. Karnik, and V. De. Design and reliability challenges in nanometer technologies. In *DAC'04: Proceedings of the 41st annual conference on Design automation*, page 75, New York, NY, USA, 2004. ACM Press.

[Bor04] S. Borkar. Microarchitecture and design challenges for gigascale integration. In *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 37, pages 3–3, Washington, DC, USA, 2004. IEEE Computer Society.

[Bor05] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Micro, IEEE*, 25(6):10 – 16, 2005.

[Bor07] S. Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, pages 746–749, New York, NY, USA, 2007. ACM.

[BR08] P.J. Bourke and R.A. Rutenbar. A low-power hardware search architecture for speech recognition. In *INTERSPEECH-08: Proceedings of the 9th Conference of the International Speech Communication Association*, pages 2102–2105, 2008.

[CAC] CACTI. http://www.hpl.hp.com/research/cacti/.

[CdGS98] F. Catthoor, E. de Greef, and S. Suytack. *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[CDK+02] F. Catthoor, K. Danckaert, K.K. Kulkarni, E. Brockmeyer, P.G. Kjeldsberg, Achteren, and T. T. van, Omnes. *Data Access and Storage Management for Embedded Programmable Processors*. Kluwer Academic Publishers, 2002.

[CDSM04] J. Croon, S. Decoutere, S. Sansen, and W. Maes. Physical modeling and prediction of the matching properties of MOSFETs. In *ESSDERC'04: Proceedings of the 34th European Solid-State Device Research Conference*, pages 193–196. IEEE Computer Society, 2004.

[Cha09] V. Chandra. Dependable design in nanoscale cmos technologies : Challenges and solutions. *Design*, pages 1–48, 2009.

[CHS+07] J. Chang, M. Huang, J. Shoemaker, J. Benoit, S. Chen, W. Chen, S. Chiu, R. Ganesan, G. Leong, V. Lukka, S. Rusu, and D. Srivastava. The 65-nm 16-mb shared on-die l3 cache for the dual-core intel xeon processor 7100 series. *Solid-State Circuits, IEEE Journal of*, 42(4):846 –852, 2007.

[Cod] Scalable Video Coding. http://www.hhi.fraunhofer.de/en/departments/image-processing/image-communication/video-coding/svc-scalable-extension-of-h264avc/.

[Con02] C. Constantinescu. Impact of deep submicron technology on dependability of vlsi circuits. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, DSN '02, pages 205–209, Washington, DC, USA, 2002. IEEE Computer Society.

[Con03] C. Constantinescu. Trends and challenges in vlsi circuit reliability. *IEEE Micro*, 23:14–19, July 2003.

[CQS04] H. Chang, H. Qian, and S.S. Sapatnekar. The certainty of uncertainty: randomness in nanometer design. In *Integrated Circuit and System Design, Power and Timing, Modeling, Optimization and Simulation. 14th Int. Workshop. PATMOS*, pages 36–47. Springer Verlag, 2004.

[CRL+10] F. Catthoor, P. Raghavan, A. Lambrechts, M. Jayapala, A. Kritikakou, and J. Absar. In *Ultra-Low Energy Domain-Specific Instruction-Set Processors*. Springer, 2010.

[Cro05] Croon, J.A and Sansen, W. and Maes, H.E. *Matching Properties of Deep Sub-Micron MOS Transistors*. Springer, New York, 2005.

[DBBS+08] W.J. Dally, J. Balfour, D. Black-Shaffer, J. Chen, R.C. Harting, V. Parikh, J. Park, and D. Sheffield. Efficient embedded computing. *Computer*, 41(7):27 –32, 2008.

[Del97] T. J. Dell. http://www.ece.umd.edu/courses/enee759h.s2003/references/chipkill_white_paper.pdf, 1997. IBM Microelectronics Division.

[DGH+74] R.H. Dennard, F.H. Gaensslen, H.Yu, V. Leo Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted mosfets with very small physical dimensions. In *IEEE Journal of Solid-State Circuits*, volume 9. IEEE Computer Society, 1974.

[DRV+04] V. Degalahal, R. Ramanarayanan, N. Vijaykrishnan, Y. Xie, and M. J. Irwin. The effect of threshold voltages on the soft error rate. In *Proceedings of the 5th International Symposium on Quality Electronic Design*, ISQED '04, pages 503–508, Washington, DC, USA, 2004. IEEE Computer Society.

[EB05] W.M. Elgharbawy and M.A. Bayoumi. Leakage sources and possible solutions in nanometer cmos technologies. *Circuits and Systems Magazine, IEEE*, 5(4):6 – 17, 2005.

[EBSLM97] M. Eisele, J. Berthold, D. Schmitt-Landsiedel, and R. Mahnkopf. The impact of intra-die device parameter variations on path delays and on the design for yield of low voltage digital circuits. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 5(4):360 – 368, December 1997.

[EDL+04] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, Nam Sung Kim, and K. Flautner. Razor: circuit-level correction of timing errors for low-power operation. *Micro, IEEE*, 24(6):10 –20, 2004.

[EHG+07] M. Eireiner, S. Henzler, G. Georgakos, J. Berthold, and D. Schmitt-Landsiedel. In-situ delay characterization and local supply voltage adjustment for compensation of local parametric variations. *Solid-State Circuits, IEEE Journal of*, 42(7):1583 –1592, 2007.

[EKD+03] D. Ernst, Nam Sung Kim, S. Das, S. Pant, R. Rao, Toan Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 7 – 18, 2003.

[FP09] C. Forzan and D. Pandini. Statistical static timing analysis: A survey. *Integr. VLSI J.*, 42(3):409–435, 2009.

[GBC06] S. Gheorghita, T. Baasten, and H. Corporaal. Application scenarios in streaming-oriented embedded system design. In *Proceedings of the Intl. Symposium on System-on-Chip*, pages 175–178. IEEE, 2006.

[GBC08] S.V. Gheorghita, T. Basten, and H. Corporaal. Scenario selection and prediction for dvs-aware scheduling of multimedia applications. *J. Signal Process. Syst.*, 50(2):137–161, 2008.

[GPH+09] S.V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, and K. de Bosschere. System-scenario-based design of dynamic embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 14(1):1–45, 2009.

[Gup11] A. Gupta. http://www.solidodesign.com/press_releases/globalfoundries-selects-solido-variation-designer-for-high-sigma-monte-carlo-and-pvt-design-in-its-ams-reference-flow/, 2011. Solido Design Automation Inc.

[HCM+05] H.Wang, F. Catthoor, K. Maex, M. Miranda, and W. Dehaene. Systematic analysis of energy and delay impact of very deep submicron process variability effects in embedded SRAM modules. In *DATE'05: Proceedings of the conference on Design, automation and test in Europe*, pages 914–919, Washington, DC, USA, 2005. IEEE Computer Society.

[HM09] S. Herbert and D. Marculescu. Variation-aware dynamic voltage/frequency scaling. In *HPCA*, pages 301–312, 2009.

[HYB+08] S. Hong, S. Yoo, B. Bin, K. Choi, S. Eo, and T. Kim. Dynamic voltage scaling of supply and body bias exploiting software runtime distribution. In *Design, Automation and Test in Europe, 2008. DATE '08*, pages 242 –247, march 2008.

[IME] IMEC. http://www2.imec.be/been/research/scaling-driven-nanoelectronics/lithography.html.

[INKM05] R.K. Iyer, N.M. Nakka, Z.T. Kalbarczyk, and S. Mitra. Recent advances and new avenues in hardware-level reliability support. *Micro, IEEE*, 25(6):18 – 29, 2005.

[Isa01] R. Isaac. Keynote: Influence of technoogy directions on system architecture, October 2001. http://research.ac.upc.edu/pact01/keynotes/isaac.pdf.

[Ita] Intel Itanium2. http://www.intel.com/products/processor/itanium2.

[ITR] ITRS. http://www.itrs.net/links/2009itrs/2009chapters_2009tables/2009_litho.pdf.

[KKK+08] K. Kuhn, C. Kenyon, A. Kornfeld, M. Liu, A. Maheshwari, W. Shih, S. Sivakumar, G. Taylor, and P. VanDerVoorn K. Zawadzki. Managing process variation in intel's 45 nm cmos technology, 2008. Intel Technology Journal.

[KUM+10] F. Kluge, S. Uhrig, J. Mische, B. Satzger, and T. Ungerer. Optimisation of energy consumption of soft real-time applications by workload prediction. In *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2010 13th IEEE International Symposium on*, pages 63 –72, May 2010.

[KZK+98] I. Kim, Y. Zorian, G. Komoriya, H. Pham, F.P. Higgins, and J.L. Lewandowski. Built in self repair for embedded high density sram. *Test Conference, International*, 0:1112, 1998.

[Lag01] K. Lagerström. Design and implementation of an MP3 decoder. Master's thesis, Chalmers University of Technology, Sweden, 2001.

[LB06] X. Liang and D. Brooks. Mitigating the impact of process variations on CPU register file and execution units. In *MICRO-39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 504–514. IEEE Computer Society, 2006.

[LCO+05] Y. Liu, S. Chakraborty, W.T. Ooi, A. Gupta, and S. Mohan. Workload characterization and cost-quality tradeoffs in mpeg-4 decoding on resource-constrained devices. In *Embedded Systems for Real-Time Multimedia, 2005. 3rd Workshop on*, pages 129 – 134, 2005.

[LGT09] M. Lukasiewycz, M. Glaß, and J. Teich. Exploiting data-redundancy in reliability-aware networked embedded system design. In *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '09, pages 229–238, New York, NY, USA, 2009. ACM.

[LPB+05] J. Lee, S. Park, H. Bang, T. Kim, and S. Cha. A hybrid framework of worst-case execution time analysis for real-time embedded system software. In *Aerospace Conference, 2005 IEEE*, pages 1 –10, march 2005.

[MAW08] M. May, M. Alles, and N. Wehn. A case study in reliability-aware design: A resilient ldpc code decoder. In *Design, Automation and Test in Europe, 2008. DATE '08*, pages 456 –461, march 2008.

[MB05] C. McNairy and R. Bhatia. Montecito: a dual-core, dual-thread itanium processor. *Micro, IEEE*, 25(2):10 – 20, 2005.

[MCB+04] P. Marchal, F. Catthoor, D. Bruni, L. Benini, J.I. Gomez, and L. Pinuel. Integrated task scheduling and data assignment for SDRAMs in dynamic applications. *IEEE Design and Test of Computers*, 21(5):378–387, 2004.

[MKMR05] S. Mukhopadhyay, K. Kang, H. Mahmoodi, and K. Roy. Design of reliable and self-repairing sram in nano-scale technologies using leakage and delay monitoring. In *ITC '05: Proceedings of the 2005 IEEE International Test Conference*, pages 1135–1145. IEEE Computer Society, 2005.

[MLCO04] A. Maxiaguine, Yanhong Liu, S. Chakraborty, and Wei Tsang Ooi. Identifying "representative" workloads in designing mpsoc platforms for media processing. In *Embedded Systems for Real-Time Multimedia, 2004. ESTImedia 2004. 2nd Workshop on*, pages 41 – 46, 2004.

[MMMR04] S. Mukhopadhyay, H. Mahmoodi-Meimand, and K. Roy. Modeling and estimation of failure probability due to parameter variations in nano-scale SRAMs for yield enhancement. In *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on*, pages 64–67, June 2004.

[MMS+07] Z. Ma, P. Marchal, D.P. Scarpazza, P. Yang, C. Wong, J.I. Gomez, S. Himpe, C. Ykman-Couvreur, and F. Catthoor. *Systematic Methodology for Real-Time Cost-Effective Mapping of Dynamic Concurrent Task-Based Systems on Heterogenous Platforms*. Springer Publishing Company, Incorporated, 1st edition, 2007.

[Moo65] G.E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[Moo75] G.E. Moore. Progress in digital integrated electronics. volume 21, pages 11 – 13, 1975.

[MPE] MPEG. http://www.digital-audio.net/technical_ref.shtml.

[Mul04] F. Muller. Timing analysis: in search of multiple paradigms. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 126, april 2004.

[MWP+00] P. Marchal, C. Wong, A.S. Prayati, N. Cossement, F. Catthoor, R. Lauwereins, D. Verkest, and H. De Man. Dynamic memory oriented transformations in the mpeg4 im1-player on a low power platform. In *Proceedings of the First International Workshop on Power-Aware Computer Systems - PACS*, pages 40–50, 2000.

[NAB+03] N.S.Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36:68–75, 2003.

[Nar05] S.G. Narendra. Challenges and design choices in nanoscale cmos. *J. Emerg. Technol. Comput. Syst.*, 1:7–49, March 2005.

[Pal07] Martin Palkovic. *Enhanced applicability of loop transformations*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 2007.

[Pan95] D. Pan. A tutorial on mpeg/audio compression. *IEEE MultiMedia*, 2:60–74, June 1995.

[PBC+05] M. Palkovic, E. Brockmeyer, F. Catthoor, H. Corporaal, and P. Vanbroekhoven. Systematic preprocessing of data dependent constructs for embedded systems. In *Integrated Circuit and System Design, Power and Timing, Modeling, Optimization and Simulation. 15th Int. Workshop. PATMOS*, pages 89–99. Springer Verlag, 2005.

[PDW89] M.J.M. Pelgrom, A.C.J. Duinmaijer, and A.P.G. Welbers. Matching properties of mos transistors. volume 24, pages 1433–1440, 1989.

[PLW+05] A. Papanikolaou, F. Lobmaier, H. Wang, M. Miranda, and F. Catthoor. A system-level methodology for fully compensating process variability impact of memory organizations in periodic applications. In *CODES+ISSS'05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 117–122, New York, NY, USA, 2005. ACM Press.

[PPV+06a] V. Petrescu, M. Pelgrom, H. Veendrick, P. Pavithran, and J. Wieling. Monitors for a signal integrity measurement system. In *ESSCIR '06: Proceedings of the 2006 IEEE European Solid-State Circuits Conference*, pages 122–125. IEEE Computer Society, 2006.

[PPV+06b] V. Petrescu, M. Pelgrom, H. Veendrick, P. Pavithran, and J. Wieling. A signal-integrity self-test concept for debugging nanometer cmos ics. In *Solid-State Circuits Conference, 2006. ISSCC 2006. Digest of Technical Papers. IEEE International*, pages 2220 –2229, 2006.

[RBB+06] J. Roberts, T. Bacuita, R.L. Bristol, H. Cao, M. Chandhok, S.H. Lee, M. Leeson, T. Liang, E. Panning, B.J. Rice, U. Shah, M. Shell, W. Yueh, and G. Zhang. Exposing extreme ultraviolet lithography at intel. *Microelectron. Eng.*, 83(4-9):672–675, 2006.

[Res] Semico Research. http://www.semico.com/studies/category.asp?id=4.

[RGP+97] G. S. Rao, T. A. Gregg, C. A. Price, C. L. Rao, and S. J. Repka. Ibm s/390 parallel enterprise servers g3 and g4. *IBM J. Res. Dev.*, 41:397–403, July 1997.

[RvEJ+02] M.J. Rutten, J.T.J. van Eijndhoven, E.G.T. Jaspers, P. van der Wolf, O.P. Gangwal, A. Timmer, and E.-J.D. Pol. A heterogeneous multiprocessor architecture for flexible media processing. *Design Test of Computers, IEEE*, 19(4):39 –50, 2002.

[S. 02] S. Dimitrios and C. Piguet and G. Costas. *Designing CMOS Circuits For Low Power*. Springer, Boston MA, 2002.

[SGK11a] M. Snir, W. Gropp, and P. Kogge. http://spectrum.ieee.org/semiconductors/nanotechnology/euv-light-makers-playing-catchup, 2011. IEEE Spectrum.

[SGK11b] M. Snir, W. Gropp, and P. Kogge. https://www.ideals.illinois.edu/handle/2142/25469, 2011. Computer Science Whitepapers.

[SHA02] R. Sasanka, C.J. Hughes, and S.V. Adve. Joint local and global hardware adaptations for energy. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 144–155, New York, NY, USA, 2002. ACM.

[SIAR10] Semiconductor Industries Association Roadmap. International Technology Roadmap for Semiconductors, Technical Reports. 2010 update, 2010.

[Sik97] T. Sikora. Mpeg digital video-coding standards. *Signal Processing Magazine, IEEE*, 14(5):82 –100, September 1997.

[Sla10] C. Slayman. Impact and mitigation of dram and sram soft errors, 2010. IEEE - SCV - Reliability.

[Soc06] IEEE Solid-State Circuits Society. The technical impact of moore's law, 2006.

[Soc07] IEEE Solid-State Circuits Society. The impact of dennard's scaling theory, 2007.

[TMQW06] Y. Tan, P. Malani, Q. Qiu, and Q. Wu. Workload prediction and dynamic voltage scaling for MPEG decoding. In *ASP-DAC'06: Proceedings of the 2006 conference on Asia South Pacific design automation*, pages 911–916, New York, NY, USA, 2006. ACM Press.

[Vie] Bit Rate Viewer. http://www.winhoros.de/docs/bitrate-viewer/.

[VVWW01] Marc A. Viredaz, Marc A. Viredaz, Deborah A. Wallach, and Deborah A. Wallach. Power evaluation of a handheld computer: A case study. Technical report, Compaq Western Research Laboratory, 2001.

[WMDC09] H. Wang, M. Miranda, W. Dehaene, and F. Catthoor. Design and synthesis of pareto buffers offering large range runtime energy/delay tradeoffs via combined buffer size and supply voltage tuning. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(1):117 –127, jan. 2009.

[WMP+05] H. Wang, M. Miranda, A. Papanikolaou, F. Catthoor, and W. Dehaene. Variable tapered pareto buffer design and implementation allowing run-time configuration for low-power embedded srams. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 13(10):1127 – 1135, 2005.

[WYB+10] W. Wang, S. Yang, S. Bhardwaj, S. Vrudhula, F. Liu, and Yu Cao. The impact of nbti effect on combinational circuit: Modeling, simulation, and analysis. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 18(2):173 –183, 2010.

[XLL08] C. Xian, Y. Lu, and Z. Li. Dynamic voltage scaling for multitasking real-time systems with uncertain execution time. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(8):1467 –1478, aug. 2008.

[YC04] P. Yang and F. Catthoor. Pareto optimization based run-time task scheduling for embedded systems. In *Proc. Wsh. on Hardware/Software Co-Design and Intnl. System-level Synthesis Symposium (CodesISSS)*, pages 120–125, 2004.

[ZHHO04a] P.S. Zuchowski, P.A. Habitz, J.D. Hayes, and J.H. Oppold. Process and environmental variation impacts on ASIC timing. In *ICCAD'04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pages 336–342, Washington, DC, USA, 2004. IEEE Computer Society.

[ZHHO04b] P.S. Zuchowski, P.A. Habitz, J.D. Hayes, and J.H. Oppold. Process and environmental variation impacts on asic timing. In *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pages 336 – 342, 2004.

List of Figures

List of Tables