

INFORME TÉCNICO-TECHNICAL REPORT

**Acoplo de ruido e integridad de señal:
Herramientas de CAD en el IMSE**

**Noise coupling and signal integrity. CAD
tools at IMSE**

Javier Castro, Pilar Parra y Antonio J. Acosta

Noviembre 2005

IMSE-CNM-CSIC/Universidad de Sevilla

Acoplo de ruido e integridad de señal

Introducción:

- El ruido causado por la actividad de los circuitos integrados está siendo un factor limitante para el desarrollo de los circuitos VLSI del futuro.
- Acoplo de perturbaciones de corriente y voltaje a los circuitos co-integrados (en SoC, más comunes en la actualidad) sobre todo por el sustrato como medio de propagación.
- El efecto del ruido es especialmente importante donde los circuitos digitales de alta-velocidad están integrados junto con secciones analógicas altamente sensitivas.
- Tener en cuenta este ruido en el diseño requiere un modelado eléctrico del sustrato y de la generación del ruido.
- Crosstalk es el término general usado para indicar acoplos parásitos en un sistema electrónico. Los principales crosstalk coupling son 3:
 - 1) Acoplo capacitivo entre las interconexiones
 - 2) Switching noise en las líneas de alimentación
 - 3) Acoplo de ruido por el sustrato de silicio que soporta el circuito (veremos éste, considerado como uno de los factores limitantes de los futuros circuitos integrados de señal-mixta, SoC, y que ha sido razón de fuertes esfuerzos en los últimos años, así como nueva generación de herramientas CAD).
- Dos razones por las que el ruido es un tema importante en el diseño VLSI:
 - 1) El sustrato introduce parásitos y caminos de señal no deseados
 - 2) Por el sustrato se propagan todos los ruidos presentes en el circuito (interconexiones y switching de voltaje y corriente)
- El efecto del acoplo de ruido por el sustrato (a diferencia de otros ruidos) tiene que tener en cuenta muchos aspectos diferentes del diseños (como la tecnología, los detalles del layout, la arquitectura, la red de distribución de potencia, y el empaquetamiento).

Fuentes de ruido:

- Las fuentes de ruido que afectan al sustrato son los transistores (dispositivos), los componentes pasivos en circuitos en circuitos RF, las líneas de interconexión y la red de alimentación (power supply network) formada por las power lines y el packaging.
- Los transistores MOS están difundidos en la superficie del sustrato, y por tanto pertenecen a la estructura metalúrgica del sustrato. La actividad en los nodos del dispositivo es acoplada mediante un acoplo capacitivo de las reverse pn junctions (causadas por un cambio de voltaje, dV/dt).
- Las interconexiones (líneas de metal que unen las diferentes partes del circuito y que van por la parte superior del sustrato) pueden también acoplar ruido debido a su capacidad parásita a sustrato. Interconexiones largas pueden acoplar tanto ruido como cientos de transistores MOS.
- Por último, digital current switching es una importante fuente de ruido que puede ser acoplada directamente al sustrato debido a los biasing contacts. Los circuitos digitales causan funciones de onda de corriente puntiagudas con un inherente alto dI/dt . Impedancias parásitas de carácter inductivo presentes en el packaging, bonding wires y on-chip power-supply distribution causan ruidos de voltaje ($V_{noise} = LdI/dt$, donde L es la inductancia efectiva del power-supply network). Debido a que el sustrato y los pozos están polarizados (biased) con las líneas de Vdd y GND (las cuales están conduciendo ruido LdI/dt) el acoplo de ruido es dirigido al sustrato, inyectado a

través de los contactos óhmicos así como por las capacitancias de deplexión. En un circuito complejo el número de contactos de biasing es elevado, por lo que el acoplo es bastante efectivo. Este tipo de ruido también se llama ground bounce y simultaneous switching noise (SSN), y ha sido mostrado como la fuente de ruido de substrato más importante en mixed-A/D circuits.

Modelado de la fuente de ruido

- El acoplo a través de la capacitancia de deplexión en las uniones pn formada por los dispositivos y el substrato puede ser evaluada analíticamente y puede ser mostrado que el nivel de ruido inyectado es proporcional a $C_j dV/dt$, siendo C_j la capacitancia de unión y dV/dt la pendiente del voltaje swing, y ambos se espera que aumenten en el futuro, por lo que los niveles de ruido generados se espera que suban.
- El acoplo por las interconexiones llegará a ser más importante con el aumento de las frecuencias de reloj digitales con énfasis en las conexiones locales cerca del substrato.
- Mientras que las capacitancias de deplexión MOS y las corrientes de impact ionization están ya incluidas en los modelos de dispositivos, un reto para las CAD de extracción de parásitos será la investigación de modelos óptimos para el acoplo entre las líneas de metal y el substrato (las líneas de metal deben tener en cuenta el comportamiento de línea de transmisión, reflexiones ...).
- La principal fuente de ruido de substrato es el switching noise, el cual debe ser evaluado de forma precisa, y para ello un dato esencial es la inductancia parásita del package.
- Un reto será por tanto un modelo eficiente de los parásitos RLC para CI grandes, junto con nuevas estrategias para reducir el simultaneous switching (como circuitos asíncronos, caminos ópticos o sin cables para la distribución de la señal de reloj).

Propagación por el substrato

- El substrato puede ser considerado como un medio resistivo en 3-D. Una vez que el ruido ha sido acoplado en el substrato se propaga llegando a los puntos sensitivos.
- Dos tipos de substrato pueden ser considerados en las tecnologías CMOS:
 - 1) Substrato digital puro: altamente dopado (baja resistividad) P+ (~ 10 miliOhm*cm), para prevenir efectos latch-up, y con una capa epitaxial resistiva arriba (P- epitaxy ~ 10 Ohm*cm). Aquí el ruido se mueve sin resistencia por el sustrato P+, por lo que no se atenúa con la distancia, y alcanza todas las partes del circuito. Por tanto el substrato conductivo puede ser considerado como un nodo simple (single node).
 - 2) Substrato analógico (preferido para circuitos de señal mixta y RF): ligeramente dopado (alta resistividad) P-. La atenuación con la distancia es apreciable, por lo que los anillos de guardia suelen funcionar (dependiendo de las condiciones de package y layout).
- Se necesitan modelos para evaluar la propagación del ruido de substrato, como una descripción precisa de los modelos de package y los perfiles de dopado verticales.
- El efecto skin no afecta a los substratos P-, para los P+ actualmente tampoco, pero puede que en la siguiente década llegue a afectar, por lo que su influencia deberá ser tenida en cuenta en los modelos de substrato.
- En cuanto al comportamiento dieléctrico del silicio, a bajas frecuencias se comporta como una red puramente resistiva (sólo parte real), pero con el aumento de la frecuencia, la parte imaginaria empieza a ser significativa y el substrato no se puede considerar como una resistencia distribuida pura (el modelar esto será otro reto para las CAD).

Sensibilidad al ruido de sustrato

- La sensibilidad de los componentes activos y pasivos tiene que ser evaluada de forma diferente.
- En los activos (transistores MOS) el ruido puede afectarles de 3 formas diferentes:
 - 1) Acoplo capacitivo (lo opuesto a la generación de ruido por la capacitancia de deplexión, así que la cantidad de ruido acoplado está relacionada con el área de contacto de la unión (C_j) y la pendiente de la señal de ruido (dV/dt))
 - 2) Body effect
 - 3) Acoplo directo (a través de contactos directos (contactos óhmicos))
- En el caso de los pasivos (resistores, capacitores e inductores) es de manera capacitiva como les afecta el ruido. Como suelen tener áreas grandes serán propensos a recibir ruido.

Eliminación del ruido

- La eliminación o al menos la reducción del ruido de sustrato es un tema importante en el diseño VLSI.
- La manera más eficaz es reducir las fuentes de ruido y la eficiencia del camino de propagación.
- Aislar las fuentes y los nodos sensitivos, hundir el ruido en el camino de los nodos sensitivos (guard rings), técnicas de circuito para reducir el ruido, como compensación de ruido (la actividad del nodo es distribuida en tiempo y/o geometría, de manera que el efecto del ruido es compensado parcialmente), realimentación negativa (similar al anterior, pero en este caso nodos específicos y controlados (controlados por un negative feedback loop) causan la cancelación del ruido), differential signaling (la información está no en el voltaje de un nodo sino en la diferencia entre dos nodos) y system decoupling).

Herramientas

- Para el diseño de CI complejos incluyendo la extracción de ruido de sustrato se necesitan herramientas CAD sofisticadas.
- Las herramientas tienen un trade-off exactitud/velocidad de la extracción.

Técnicas para modelado de sustrato

- 5 técnicas desde la más exacta a la menos:
 1. FEM (Finite element methods): usado en los simuladores de dispositivos, basados en la solución numérica de la continuidad de portadores y ecuación de Poisson.
 2. FDM (Finite difference methods): resuelven numéricamente la ecuación de la ley de Ampere, sin campo magnético, lo que reduce el sustrato a una red RC.
 3. BEM (Boundary elements method): resuelve la ecuación de Laplace. Sólo la discretización de ports es necesitada, y sólo las relaciones port-to-port son modeladas.
 4. Modelos usando fórmulas aproximadas semiempíricas con dependencia de la tecnología y la geometría.
 5. Single-node approximation: el sustrato entero es considerado como un nodo eléctrico simple (lo que es razonablemente válido para los P+)

Herramientas comerciales

- Están apareciendo nuevas herramientas para el tratamiento del ruido y la integridad de señal debido a su complejidad.

- 4 herramientas conocidas:

1. Space: desarrollada por la universidad de Delft, puede usar de BEM o fórmulas de interpolación, dependiendo de la exactitud deseada.
2. SCA: basada en BEM fue el primer extractor de parásitos desarrollado por Cadence, pero no se continuó.
3. SeismIC: distribuido por Cadence, usa el BEM como defecto, pero puede pasarse a FDM donde haga falta más exactitud. *About SeismIC: SeismIC performs substrate RC model extraction, noise simulation and advises on techniques to minimize substrate noise. SeismIC's extraction and simulation engines facilitate design verification, trade-off analysis and optimal noise immune designs. SeismIC can presently handle CMOS, epi-CMOS, BiCMOS, and SiGe processes.*
4. SubstrateStorm: de Cadence también, usa FDM, lo que nos da alta exactitud pero una extracción lenta.

The physics challenges at 0.13-micron and beyond demand a new generation of design technology solutions. To be successful, designers need tools that can easily interoperate, and support hierarchical design methods and rapid prototyping. These tools must also take into account critical, deep submicron effects, such as signal integrity, voltage (IR) drop, electromigration and substrate noise.

Cadence has had exclusive distribution rights for Fire & Ice? QX, Simplex's leading 3D interconnect modeling and extraction tool, since January 2002. A roadmap for the incorporation of Simplex's other products is currently under development. This will include: VoltageStorm^(TM) SoC, Simplex's flagship power grid verification product, ElectronStorm^(TM), for electromigration analysis, SignalStorm^(TM) SoC, for digital signal integrity delay calculation and library characterization, and SubstrateStorm^(TM), for substrate noise analysis.

Como medidas indirectas del ruido podemos ver las herramientas de potencia, las que nos dan valores de la intensidad, y por tanto el pico de intensidad lo podemos tomar como referencia de ruido. Como herramientas de este tipo a nivel de puertas tenemos en Synopsys a PrimePower, herramienta que nos da la intensidad promedio por cada celda, y los picos. Para obtener los picos en el tiempo (no sólo el mayor), necesitamos como fichero de actividad el VCD (es el más exacto, y por tanto ocupa un gran tamaño, y en él están todos los eventos de la simulación a nivel de puertas, la cual podemos hacer con Verilog XL, con lo que debemos hacer un testbench verilog con los cambios de las entradas, por lo que será dependiente de los estímulos), y para poder ver la forma de la intensidad en el tiempo nos hará falta un visualizador de ondas. En la herramienta viene TurboWave, pero no lo podemos utilizar por no tener licencia. Sin embargo en Synopsys está CosmoScope que nos deja ver la forma de onda (fichero .fsdb, que se genera con un comando de PrimePower: set_waveform ...). Por ahora no tenemos acceso a la licencia de VCD, para utilizarla, por lo que los resultados no nos sirven en este momento para el ruido (a no ser que sólo haya un pico de ruido). El problema debe solucionarse cuando esté instalado Synopsys 2005. Entonces debemos saber sacar el fichero de actividad VCD (esto se hace con una modificación en el testbench verilog), que por lo visto suele ocupar bastante. Otro problema será cómo son las entradas al sistema (bien aleatorias muchas, todas las combinaciones de las entradas...).

- En Cadence existe la herramienta de síntesis PKS (Physically Knowledgeable Synthesis), la cual nos permite una opción de síntesis low-power (PKS-LPS flow), con la que podemos sintetizar el RTL a puertas minimizando la potencia sin que los tiempos críticos se vean modificados (los

tiempos como principal restricción, y entonces reducir la potencia). La herramienta lee librerías de tiempos y potencia (.tlf o .alf) y físicas (.lef), el verilog (HDL), se genera una implementación hardware en la forma de un netlist genérico (technology independent), se ponen las restricciones de tiempos, se hace una optimización pre-placement (conjunta entre timing y potencia). Luego se genera un netlist verilog a nivel de puertas, que es el que se simula con verilog y da como resultado un fichero .tcf (toggle count format), que tiene la información de la gate switching activity. Esto lo hace con unas rutinas PLI y modificando el testbench, incluyendo dos líneas. También se puede generar el fichero .tcf a partir de uno .vcd. Una vez que tenemos el .tcf, se lee y se calcula la potencia. Si optimizamos el diseño y lo emplazamos, podemos ver la reducción de la potencia calculandola después otra vez. El problema es que no hay ninguna opción (que yo sepa) para poder ver cómo es la potencia (o intensidad) en el tiempo (lo que nos haría falta para información sobre el ruido). Tampoco tiene ningún comando como el de PrimePower (set_waveform) que nos genere un archivo para poder visualizarlo (como el .fsdb). Lo que hay es un comando, `get_dynamic_power` (*get_dynamic_power: Traverses the hierarchy and returns the average cycle-by-cycle power consumed for the specified instances or nets of the current module in the period during which the events are monitored. This period is defined with the read_vcd command. Use the get_power_display_unit command to show the power units. LPS computes the power for each transition in the VCD file that is relevant to the specified instances and nets. Note: You must read a VCD file (using read_vcd -dynamic) before you can run the get_dynamic_power command. If you change any slew or capacitance values for the design, you need to read in the VCD file again to recalculate the dynamic power*).

- Una herramienta para la potencia de Cadence que está en Silicon Ensemble es el Power Analyzer, pero para él hay que llegar al layout, y tener el fichero .DEF (*Design Exchange Format (DEF) files provide information about the physical layout of the design. The power analyzer obtains information about cell placement, net connectivity, and special net (VDD/VSS) physical layout from DEF files.*). Entonces se puede hacer un análisis de potencia dinámico que nos permite ver la intensidad en el tiempo, que es lo que queremos, y la forma en que se ve depende del modelo de intensidad que se elija.

- A diferencia del substrato, también debemos tener en cuenta la integridad de la señal (voltage drop, IR, y electromigration, EM). Para ello debemos buscar herramientas que las tengan en cuenta (...)

Due to shrinking geometries and increasing clock speeds in deep submicron design, power consumption has become a significant design constraint. Signal integrity, electromigration (EM) and voltage (IR) drops also gained importance.

The power analyzer is a tool that performs power calculation and power grid integrity verification, including:

-Calculation of power consumption

-Identification of IR drop issues in power rails and instances

-Identification of EM issues in power rails and vias

You can use this tool to perform both static and dynamic (time-based) analysis on multimillion transistor gate designs at SPICE-level accuracy, and then view the results statically or dynamically in a display window.

Necesidades y retos de las herramientas CAD

- Las herramientas de extracción de substrato existentes básicamente proporcionan un modelo eléctrico para simular la interacción entre un nodo de generación de ruido individual y un nodo sensitivo individual.
- La extracción es obtenida de layout detallado, lo que es claramente ineficiente ya que entonces los problemas de ruido son detectados en la última fase del flujo de diseño, lo que implicaría rediseños costosos y time-consuming.
- Ya que el número de nodos ruidosos y sensitivos en el layout puede ser enorme (todos los contactos simples substrate/well, todos los transistores), el modelo eléctrico que se obtiene es enorme y difícil de tratar (simulaciones a nivel de transistor que pueden llevar una enorme cantidad de tiempo o incluso ser inaccesibles).
- Como principal necesidad para las herramientas de extracción del futuro, es necesario buscar una simplificación de la descripción de las fuentes de ruido y obtener una función de onda de ruido caracterizada desde un alto nivel HDL.
- Una técnica propuesta podría ser usar una librería de modelos de noise-equivalent standard-cells. Desde una descripción HDL y una simulación del circuito y la librería de modelos noise-equivalent, la actividad de ruido global puede ser extraída y reducida a un circuito equivalente simplificado.
- Esta técnica es válida y eficiente para los P+, donde es posible añadir todas las funciones de onda generadas por cada subcircuito para obtener el nivel de ruido de substrate total, incluso ignorando la posición de los dispositivos en el layout.
- Sin embargo, para lo P-, los usados en señal-mixta y RF, esto es un reto, ya que la posición de los elementos ruidosos y sensitivos afecta al resultado. Una solución propuesta es usar macro-modelos de grupos de puertas caracterizados, simplificando el problema de análisis.
- En el futuro, los bloques IP (intellectual property) tendrán una descripción de ruido específica, de manera que las herramientas puedan manejar la estimación de ruido en circuitos SoC con alta exactitud y eficiencia.

Macromodelo de ruido de substrato de digital cores

- Se demuestra que sólo una fuente de ruido por substrato debe ser considerada, el SSN introducido por los contactos de substrato.
- Nos interesa poder ver el efecto del ruido en la fase de diseño más temprana posible, para que no tengamos que llegar al layout del circuito completo. Se necesita una generación de herramientas que permitan evaluar el ruido a nivel de puertas o incluso niveles mayores.
- Herramientas como SubstrateStorm permiten una extracción de parásitos para el substrato y pueden evaluar el acoplo de ruido, pero están lejos de las necesidades de los diseñadores (puesto que se basan en el layout final, y para circuitos grandes el tiempo de simulación es excesivo o inviable).
- Hacen falta otras herramientas, y las aproximaciones que se han llevado a cabo hasta ahora se basan básicamente en la superposición de patrones de ruido (noise signature) generado por cada puerta de una standard cell library. Ahora bien, ¿de qué ruido da cuenta esta “firma” de ruido? En algunos trabajos se debe al ruido generado por los transistores conmutando, ignorando el ruido por el power-supply. Por tanto una cosa importante a saber es qué ruido es el más importante, o cuáles, a la hora de hacer esta firma de ruido. Según los estudios de J. F. Osorio (U.P. Catalunya) en la mayoría de las situaciones, el power-supply noise será el contribuidor dominante al ruido de

substrato, por lo que ignorarlo sería un error. El trabajo de van Heijningen es ahora mismo la aproximación más avanzada, válida para substratos altamente conductivos sólo, no para substratos resistivos (los usados en RF/SoC).

- Otra cosa que hay que saber para hacer el modelos son, además de las fuentes relevantes de ruido, los parásitos que hay que tener en cuenta.

- Y una vez obtenida toda la información para el macromodelo, se obtienen las “firmas” de ruido (aquí no sé muy bien como hacer esto, se supone que desde un nivel bajo de la celda (layout) a la que se le quiere obtener el macromodelo, se realiza una simulación, ¿pero cómo? ¿con patrones de entrada todos los posibles? y con eso se obtendría la firma de ruido para la celda que será la que se use después en un circuito digital largo. El problema en ese momento es si se pueden superponer las firmas (superposición de corrientes) de cada una. Esto sería cierto sólo cuando el substrato sea P+, donde no importa en que parte del layout del circuito grande final estan la celdas.

- Otra cosa a tener en cuenta son los parásitos del package.

- Como resumen de fuentes de ruido de substrato en el dominio digital tenemos 3:

1. Transistores

2. Interconexiones

3. SSN en las líneas de power-supply

- Si todas se tienen en cuenta para un circuito digital complejo el problema resulta ser bastante grande computacionalmente hablando. Por tanto sólo las fuentes relevantes deben ser modeladas si queremos una estimación rápida y eficiente. Por eso debemos saber qué fuente es la relevante y si vale para cada situación.

- Según la experiencia de muchos autores es el SSN introducido por los contactos a substrato la que domina sobre los transistores e interconexiones. Como conclusión se vio que el ruido de substrato observado siempre era originado en las power-supplies, particularmente por las corrientes de switching excitando las resonancias del package.

- Para distinguir las fuentes de ruido (SSN y transistores) se podrían hacer simulaciones con un modelo de package incluido (No package $L=0$ nH, package $L \sim 1, 2$ y 5 nH). Medir el peak-to-peak noise voltage (el cual debe aumentar conforme aumenta L) y comparar el substrate noise waveform con SSN waveform a la línea de tierra V_{gnd} del circuito (esto hay que verlo y entenderlo mejor ...). La cantidad de ruido de substrato producida por los switching transistors solos puede ser observada en la situación de “No package”.

- Se concluye que el SSN domina sobre los transistores como fuente de ruido en los casos reales (incluso se podría calcular, interpolando, cuál es el límite en L para el que ocurre esto para un circuito determinado, y tener en cuenta que para bondwires típicos de ahora se tienen $L \sim 1$ nH/mm).

- Una conclusión importante que se saca del trabajo de Osorio es que pese a que era asumido que sólo los substratos poco-resistivos o epitaxiales se comportaban como un single node, y que el ruido de substrato era atenuado con la distancia en los substratos altamente resistivos, y una red de impedancias compleja era necesitada para modelar estos substratos, esto es cierto si hay distintas fuentes de ruido con diferentes intensidades inyectadas en diferentes localizaciones del substrato (por ejemplo en el caso de las simulaciones de “No package”, donde cada transistor es una fuente de ruido diferente, se observan 4 funciones de onda de ruido diferentes en los 4 puntos de test donde se mide). Pero si hay región donde el ruido es inyectado al substrato como una fuente de ruido simple con una intensidad simple, como en el caso del SSN en el digital core, la aproximación single-node puede ser aplicada a esa región incluso si es un substrato P-.

- Por tanto, el problema de predecir el ruido de substrato producido por un digital core puede ser reducido a la estimación de la cantidad de SSN en su línea de V_{gnd} .

- El ruido en las líneas de power-supply es un producto de los picos de corrientes en las inductancias parásitas, LdI/dt . Pero también existen capacidades entre Vdd y Vgnd que influyen en el SSN, por lo que el SSN puede ser considerado como un pico LdI/dt seguido de unas oscilaciones de amortiguamiento debido a la resonancia de los parásitos del package del circuito. También las resistencias en serie con las capacitancias son importantes.

- Un macromodelo para el ruido podría estar constituido por el package, que puede ser reducido a la inductancia total en los pines de Vdd/Vgnd, una fuente que modele la corriente total demandada por las switching gates, y un circuito equivalente RC de los parásitos entre Vdd y Vgnd.

- ¿Qué contribuye a estas capacitancias y resistencias?:

1. Capacitancia: contribuida por las uniones pozo-substrato, diodos de protección, y la capacitancia de entrada de las puertas non-switching.
2. Resistencia: contribuida por el substrato y los pozos en serie con las capacitancias de unión, y las resistencias del canal de las puertas non-switching.
3. Los elementos relevantes del modelo de ruido podrían ser: la capacitancia de pozo (C_w) y su resistencia de substrato en serie (R_s), la capacitancia de salida de la celda (C_o) y su corriente de switching (I_{sw}). Esto para una celda, para el circuito global se usaría la superposición para obtener los resultados, añadiendo las funciones de onda de ruido en su preciso tiempo. Las intensidades estas (I_{sw_i}) son obtenidas desde una simulación Spice del layout completo de la celda standard digital incluyendo el modelo de substrato pero sin tener en cuenta los parásitos de packaging. La función de onda resultante es entonces guardada en la librería y usada después por el macromodelo durante la simulación. Hay que tener en cuenta que las intensidades dependen del supply voltage, y éste será no constante cuando estén presentes los parásitos de package.
4. La conclusión general es que la superposición de las funciones de onda de corriente será válida en todas las situaciones prácticas, proporcionada una predicción exacta de la capacitancia en el macromodelo.
5. Los valores de la capacitancia del circuito equivalente de ruido se obtiene de las especificaciones de la tecnología dadas por la fundición (foundry). La resistencia de substrato es calculada usando SubstrateStorm para la extracción y simulaciones SPICE para obtener un valor equivalente para la red de substrato de la celda global. Y la corriente de switching de la celda es obtenida de una simulación SPICE ideal (sin modelo de package).

Macromodelo de ruido para la simulación rápida de la generación de switching noise

- Tres características son requeridas para hacer útiles las herramientas CAD para predecir el ruido:

1. exactitud,
2. extracción rápida,
3. y predicción pre-layout.

- El análisis de la generación de ruido de substrato puede ser dividido en dos tareas fundamentales:

1. Modelar el substrato (red resistiva, lo hacen herramientas como SNA)
2. La estimación de la función de onda de ruido de substrato que es inyectada por el circuito digital completo y propagada por a través de la red del substrato (simulaciones a nivel de transistor con SPICE de esto pueden ser impracticables, por lo que hay que buscar otra forma, macromodelos).

- A la hora de hacer el macromodelo podemos hacer dos suposiciones:

1. La alteración inyectada por fuentes de ruido de sustrato que no sean contactos de biasing pueden ser despreciadas para los circuitos digitales de la vida real.
2. El ruido de sustrato es generado de manera dominante por el ruido dI/dt presente en la línea gnd debido a las switching currents del circuito digital.

- Parámetros del macromodelo:

1. Capacitancia de pozo y resistencia de sustrato: C_{well} (well to substrate capacitance) se calcula desde los datos tecnológicos y el layout de la celda, midiendo las dimensiones del well. La resistencia en serie entre el well y los supply nodes (R_{subs}) se obtiene del modelo de sustrato (obtenido de alguna herramienta de extracción de sustrato, como SubstrateStorm) del layout de la celda, forzando a un corto en la well capacitance.
2. Capacitancia y resistencia de la celda: las capacitancias intrínsecas son de difusión de drenador al well (transistores p, C_{outp}) y sustrato (los n, C_{outn}), y se pueden calcular de la geometría del layout de la celda y la información tecnológica. Sus resistencias en serie $R_{\text{outp,n}}$ pueden ser obtenidas desde una simulación del netlist de sustrato, dado que este netlist tendrá puertos de acceso (access ports) a ambos, a los transistor bulk nodes y a los well y substrate taps.
3. Formas de onda de switching current (I_{sw}): se obtiene midiendo la corriente en los nodos Vdd o gnd locales, desde una simulación del netlist extraído de la puerta desde el layout incluyendo el modelo de sustrato. Diferentes formas de onda deben ser obtenidas para cada vector de entrada, y entonces debe ser creada para cada celda una database que incluya todas las combinaciones de las transiciones de las entradas. También debería de tenerse en cuenta el fanout y los input rise/fall times.

Integridad de señal (signal integrity):

Introducción:

Con la era de diseño deep submicron y nanométrica (0.18 micras y por debajo), los circuitos están operando a más altas frecuencias, mayor potencia y los efectos de la capacitancia de acoplo entre las interconexiones están llegando a ser más importantes. Muchos efectos, que no eran importantes en la generación previa de diseños, están llegando a ser un factor importante en el correcto funcionamiento y performance de los chips. Las interconexiones son ahora importantes, su conexión y las geometrías cada vez más pequeñas están siendo un tema de diseño clave. Estamos llegando al paradigma del diseño centrado en las interconexiones. En este paradigma, el timing, power, signal integrity y reliability están llegando a ser tan importantes como el área del dado (lo más importante en la generación previa).

¿Qué es la integridad de señal?:

La integridad de señal incluye todos los efectos de diseño de IC que causan un mal funcionamiento del diseño debido a la distorsión de la forma de onda de la señal. Actualmente la cross-coupling capacitance domina a la inter-layer capacitance con cada nuevo proceso tecnológico, porque los cables están cada vez más juntos, y la cross section de las interconexiones es cada vez más fina. Están la lateral coupling capacitance entre las interconexiones de la misma capa C_c , que es unas 4-5 veces mayor que la inter-layer capacitance C_s debida al solapamiento de las interconexiones entre diferentes capas. Como efectos de signal integrity veremos:

- Crosstalk para el retraso y ruido
- IR drop
- Electromigration
- Inductance

Debemos buscar herramientas que simultáneamente exploren, analicen y optimicen todos los aspectos de la implementación del diseño como el timing, área, power y signal integrity. Además, contra antes podemos ver estos efectos en el ciclo de diseño tanto mejor.

Crosstalk:

Cuando una señal conmuta, la forma de onda de voltaje de una red vecina puede ser afectada debido a la cross-coupling capacitance entre las interconexiones. Este efecto es el crosstalk. Puede dañar tanto el timing como la funcionalidad. Para diseños de 0.5 micras de anchura de línea no era un tema importante, pero conforme la tecnología fue decreciendo (0.25 y menores) la coupling capacitance se vuelve un factor signficante y no pueden ser ignorados los efectos de crosstalk. Los análisis de crosstalk determinan el ruido inducido en una red (la víctima) por su red vecina conmutando (el agresor). Si el agresor causa suficiente variación de voltaje en la red de la víctima como para cambiar su estado digital (de 1 a 0 ó viceversa), el ruido se propaga y puede ser almacenado en un flip-flop produciendo un fallo funcional.

Si las dos redes conmutan a la vez, se pueden producir retrasos debidos al crosstalk. El retraso de la víctima se verá incrementado si conmutan en la dirección opuesta, pudiendo causar problemas de set-up. El efecto de crosstalk reducirá el retraso de la víctima si el agresor conmuta en su misma dirección, lo que podría causar problemas de hold.

Las soluciones que existen hoy para el crosstalk operan principalmente en la fase post-layout. El problema será modificar entonces el layout para el correcto funcionamiento. Como la reparación post-layout no es una aproximación deseable, diferentes aproximaciones usando un router detallado para resolver los temas de crosstalk están siendo implementadas en algunas herramientas de place and route. Una regla del dedo gordo (rule of thumb) para criterio de crosstalk (como por ejemplo la distancia paralela umbral) es proporcionada típicamente al router detallado. El router detallado invoca entonces la función de análisis de crosstalk. Si un problema de crosstalk es identificado, el router intenta de modificar el layout proporcionando más espacio entre las señales afectadas o poniéndolas en capas diferentes.

El tratar el tema del crosstalk en un routing detallado o en la fase post-layout es demasiado tarde en el ciclo de diseño. Usar herramientas de análisis en la fase post-layout no son aceptables para el tiempo de ciclo de diseño.

El ruido de crosstalk que puede llevar a un cambio en el estado lógico puede ser resuelto con:

- spacing (parece ser una solución efectiva para diseños de 0.13 micras)
- shielding
- aumentando la grounded capacitance

Electromigration:

Es el problema de la desintegración de una red (net disintegration) en un chip debido a una densidad de corriente muy alta. Este problema ha surgido con la llegada de los diseños deep submicrométricos. Circuitos más grandes y rápidos, cables más estrechos, mayor corriente pasa por ellos. Si el fallo se produce antes del final de la vida del chip en el sistema (que el metal se desintegre y cause un abierto o un corto en las interconexiones), se deberán retirar los chips, lo que repercutirá en un impacto financiero negativo a la compañía.

Hoy el problema está siendo tratado por el sobre-diseño con wide power buses, los cuales son donde se espera los mayores temas de electromigración. Sin embargo, esto puede causar problemas de congestión para las signal nets y la aproximación del sobre-diseño no puede ser aceptada. También los efectos de electromigración están siendo importantes para las redes de reloj y también signal nets. Una solución que proporcione suficientes anchura de interconexiones sin un excesivo sobre-diseño es necesaria. De nuevo nos vemos en la situación de analizar el problema en la fase post-layout, la cual es muy tarde en el ciclo de diseño. Lo suyo sería proporcionar soluciones al problema de electromigración en el proceso de generación del layout automático calculando las anchuras requeridas de las interconexiones mientras son emplazadas (placed).

IR drop:

Es el problema de la caída de voltaje (voltage drop) del power y el ground debido a una lata corriente fluyendo a través de la red resistiva de power/ground. Cuando la caída de voltaje en la power network o la subida de voltaje en el ground network llega a ser excesiva, los dispositivos correrán a una velocidad menor. El problema de IR drop llega a ser más importante para los diseños submicrométricos con:

- aumento en la corriente debido a más dispositivos en un diseño y mayores corrientes a través de cada dispositivo
- aumento en la resistencia de los cables y contact/via debido a cables más estrechos y menos contacts/vias
- descenso del supply voltage a 1.5 v y menores

Un amperio de corriente fluyendo por un ohmio de interconexión causará una caída de voltaje de 1 voltio, que para 1.5 v de Vdd serían 2/3 de él. IR drop puede causar que el chip falle debido a:

- performance (el circuito yendo más lento que las especificaciones)
- problemas de funcionalidad (violaciones de setup o hold)
- operación poco fiable (menores márgenes de ruido)

El efecto de IR drop es asociado a con el flujo dinámico de corrientes a través de los transistores en cada fase en el tiempo, por tanto es necesario típicamente un simulador de circuitos a nivel de transistor para obtener valores de voltage drop exactos. Con el apoyo de los modelos de potencia a nivel de celdas, el simulador puede ser extendido para correr a nivel de celdas, sin necesitar el transistor netlist (que sería muy engorroso y llevaría mucho tiempo de simulación)

Inductance:

Mientras la frecuencia de reloj del diseño aumenta por encima de los 500 MHz en procesos tecnológicos de 0.13 micras y menores, on-chip inductance effect puede llegar a ser importante para los ASIC y ASSP (Application Specific Standard Product) chips.

Tradicionalmente, la extracción y simulación de la self-inductance y la mutual inductance de todas las interconexiones no ha sido práctica ya que el método de extracción numérico 3D es computacionalmente intenso para ser aplicado. Como el efecto inductivo es sólo importante para redes largas con señales de alta frecuencia en diseños high performance, la extracción y el análisis del efecto de la self-inductance es enfocado a las redes del reloj y el efecto de mutual-inductance es enfocado en high-frequency bus signals.

SiliconSmart for SI

The challenge to support these new signal integrity (SI) flows falls to the library providers. While EDA tool providers offer a variety of gate-level SI analysis, correction, and avoidance capabilities, all these tools require new and unfamiliar constructs in library models. Since glitch waveforms must be considered instead of traditional near-linear signal transitions, the complexity of characterization increases dramatically. In addition, most library providers must support multiple tools and formats with increasingly limited resources and personnel to develop and deploy new libraries.

Meets Real SI Needs for Modern Design Teams

SiliconSmart with the SI option is the proven and practical characterization and modeling solution that addresses the library provider's challenge with the accuracy and throughput required to produce advanced noise libraries. (<http://www.scd.magma-da.com/c/@JU1ZhRi8iq.ko/Pages/SiliconSmartSI.html>)

Pessimism is eliminated by using glitch waveform models that represent the voltage magnitude of the glitch and its energy as a function of duration and magnitude. In addition, accounting for output load dependent effects on noise immunity dramatically reduces reported errors. Noise waveform propagation allows glitches to fan out where they may attenuate. However, small glitches may combine with other glitches to form more significant waveforms. None of this analysis can be done accurately without improved driver models. SiliconSmart provides the characterization and modeling solution to meet the real SI needs of modern design teams.

Reducción de potencia por medio de un circuit/logic design:

- Uso de más estilo estático que dinámico
- Reducir la switching activity mediante optimización lógica
- Optimizar el clock and bus loading
- Técnicas de circuito inteligentes que minimicen el número de transistores e internal swing
- Diseño custom puede reducir la potencia, pero aumenta los costes de diseño
- Reducir Vdd en los caminos no-críticos y dimensiones de los transistores adecuadas
- Usar circuitos de lógica multi-Vt
- Re-codificación de los circuitos secuenciales

Reducción de potencia por medio de un diseño arquitectural:

- Técnicas de gestión de potencia donde los bloques no usados están “cerrados”
- Arquitecturas low-power basadas en paralelismo, pipelining, etc.
- Partición de memoria con bloques permitidos selectivamente
- Reducción del número de busses
- Minimizar la instrucción set para decodificación simple y ejecución

Reducción de potencia a través de una selección de algoritmo:

- Minimizar le número de operaciones y así el número de recursos hardware
- Codificación de datos para una switching activity mínima

Reducción de potencia en integración del sistema:

- Utilizar low system clocks (frecuencias superiores son generadas con un on-chip PLL)
- Alto-nivel de integración. Integrar off-chip memories (ROM, RAM, etc.) y otros ICs como los periféricos digitales y analógicos

Gate-level techniques

Probabilistic power estimation:

La disipación de potencia puede ser analizada usando una aproximación pattern-independent cuando las señales están representadas con probabilidades (también llamadas static techniques). Esta aproximación permite vencer los defectos de las técnicas basadas en simulación. El usuario proporciona las probabilidades de las entradas de la red lógica, y el promedio de disipación de potencia es calculado como:

$$P = V_{dd}^2 f \sum_{i=1}^N \alpha_i C_i$$

donde N es el número de nodos de la red. Con una capacidad física total de Ci. Y α_i es la switching activity, dada por:

$$\alpha_i = P_i(1 - P_i)$$

donde Pi es la probabilidad de que el nodo i esté en nivel alto (high level). En la ecuación se supone que las entradas del circuito y los nodos internos son independientes (spatial independence). También los valores de la misma señal, en dos ciclos de reloj consecutivos, son asumidas independientes (temporal independence).

Las técnicas probabilísticas tienen la ventaja de que el usuario no tiene que suministrar patrones de simulación y tiene un tiempo rápido de computación. Sin embargo no tiene en cuenta la potencia interna de las puertas ni la disipación de potencia estática. Estas técnicas pueden ser usadas como un estimador de potencia rápido para la síntesis lógica, y también apropiado para comparar varias estructuras de subsistemas.

- Event-driven simulation:

Otra aproximación es propuesta para diseños semi-custom, usa una librería de celdas que han sido caracterizadas para disipación de potencia estática y dinámica con el sistema de caracterización de celdas Entice (ENergy and TIming Characterization Environment, de Motorola). La potencia dinámica incluye la potencia debida al short-circuit y la carga de la capacidad. Tiene en cuenta los siguientes parámetros: pendiente de la señal de entrada, carga capacitiva de salida, voltaje de operación, temperatura, y parámetros del proceso. Entice usa SPICE como un simulador de circuitos para modelar cada celda para la potencia. El diseñador crea un diseño de una librería de celdas a nivel de puertas, y ésta es la entrada del estimador de potencia (Aspen). También son especificados los estímulos a cargar por el simulador lógico (Verilog-XL es usado como event-driven simulator). Aspen usa vectores de potencia (los cuales describen todos los posibles eventos donde potencia puede ser disipada por una celda para casos dinámicos y estáticos) de una celda para computar la potencia total. Los resultados reportados por Aspen, como la switching activity de los nodos, pueden ser usados por las herramientas de floorplanning, placement y routing. Aspen es cuatro veces más rápido que SPICE y dentro de un 10% de los resultados de SPICE. Una desventaja de Aspen es que no tiene en cuenta la potencia debida a glitches.

Architecture-level power estimation

La arquitectura de un diseño está representada por bloques funcionales y la complejidad del diseño a este nivel es relativamente pequeña comparada con el nivel de circuito o puertas.

Una técnica aproximada para la potencia total de una parte lógica es el número de puertas multiplicado por la potencia de la puerta usando una switching activity especificada por el usuario. Este factor de actividad es asumido fijo a través de todo el diseño.

Otra técnica está basada en el modelo de bloques funcionales precharacterizados. Usa entradas aleatorias (independent Uniform White Noise, UNW), de manera que los bit de entrada no están correlacionados ni en espacio ni tiempo, y son independientes de la distribución de los datos. Por la tanto la probabilidad de cada bit de entrada será:

$$P(1) = 0.5, \text{ y } P(0 \rightarrow 1) = 0.25$$

Para tener en cuenta la correlación de los datos, se propuso el Dual Bit Type (DBT) data model. El DTB data permite una estimación exacta de la potencia disipada.

Behavioral-level power estimation

Una representación a nivel de comportamiento describe la función de un sistema frente a un conjunto de entradas. El comportamiento puede ser especificado, por ejemplo, por algoritmos (Verilog, VHDL, ...) o por funciones booleanas. La estimación de potencia a este nivel relaciona la energía consumida a la ejecución de un algoritmo. Decisiones a nivel de comportamiento o sistemas puede influenciar en varios órdenes de magnitud la disipación total de potencia del circuito.

Una aproximación para la estimación de potencia está basada en la combinación de modelos de potencia analíticos y estocásticos.

Herramientas de Synopsys

PowerArc

Power Characterization for Cell Libraries

PowerArc[®] is an automated cell library power-characterization tool. PowerArc delivers outstanding accuracy, offers efficient run-times and has the capacity to handle large objects such as memories and megacells. PowerArc generates Synopsys Liberty (.lib) power libraries that are utilized by Synopsys' power tools (Power Compiler and PrimePower) during low power ASIC design and verification.

Power Compiler

Push Button Power Reduction and RTL Power Estimation

Power Compiler automatically minimizes power consumption and enables pre-synthesis power estimation for budget analysis and architectural explorations that result in lower power and shorter design cycle. Power Compiler, built on top of Design Compiler, is the industry's only synthesis tool to simultaneously optimize a design for timing, power and area. The tool's push-button power reduction capabilities use clock-gating, operand isolation (data-gating for arithmetic operators) and gate-level power optimization techniques without violating the design's timing constraint. Power Compiler's fast and accurate RTL Power Estimator allows users to analyze power consumption and make trade-offs for power reduction when it can have the greatest impact-early in the design cycle.

PowerMill

Dynamic Verification and Power Analysis

In the pre- and post-layout cycles, PowerMill[®] provides high-speed, high-capacity circuit simulation and enables low-power design by accurately predicting power consumption and identifying power budget violations. By identifying functional and power problems before tapeout, PowerMill allows users to reduce packaging costs, design in longer battery life and avoid the high cost of re-spins. The Analog Circuit Engine (ACE) enables PowerMill to handle mixed-signal design analysis by providing detailed transistor-level accuracy of the analog and mixed A/D portions of a design.

PrimePower

ASIC Power Analysis and Verification

PrimePower is a dynamic full-chip power analysis tool for complex multimillion-gate ASICs. Its high capacity power analysis supports industry-standard synthesis libraries and provides a comprehensive power diagnosis. PrimePower provides the power signoff details designers need to meet power specs, reduce packaging costs and reduce excess power.

PrimeTime

Gate-Level, Static Timing Analysis

PrimeTime? is a full-chip, gate-level static timing analysis tool targeted for complex multimillion-gate designs. PrimeTime's integration into logical and physical design flows enables quick debugging of complex timing problems and speeds timing closure. PrimeTime is the industry-leading sign-off tool at semiconductor companies throughout the world.

AMPS

Intelligent Design Optimization

AMPS? is the only automatic transistor-level optimization tool that enhances designer productivity by simultaneously optimizing power, speed and area for custom digital CMOS blocks. AMPS eliminates time-consuming manual circuit optimization and aids in timing and power closure by automatically resizing the transistors to meet user-defined power, speed and area goals.

Chip Architect

Full-Featured, Hierarchical Design Planner

Synopsys' Chip Architect Design Planner enables users to perform automatic RTL planning, hierarchical clock tree synthesis, global routing, timing analysis and placement at various design stages. Its architecture is optimized to plan and analyze multimillion-gate designs in a short time. Chip Architect is integrated within a synthesis-based design flow, using the same logical libraries, timing engine and commands as Design Compiler, PrimeTime and Physical Compiler. It provides a seamless flow to the customer's back-end tool suite in a systematic, predictable manner.

PathMillPlus

Integrated Static Timing Verification and IP Characterization

Developed specifically for the exchange and reuse of hard IP between the IP provider and end user, PathMillPlus combines industry-proven static timing analysis with on-demand dynamic simulation to produce SPICE-level accuracy for complex, system-on-a-chip designs. With no capacity limitations, PathMillPlus gives IP developers and integrators the ability to characterize any size design ensuring complete coverage while meeting time- to-market goals.

RailMill

Power Net Analysis

Today's high-performance nanometer IC designs can fail without accurate power net analysis. RailMill? identifies IR drop and electromigration issues before design tape-out and helps to eliminate lengthy prototype debug, and mask and fab re-spins. RailMill maximizes the return on design investment and enhances profitability by providing high quality designs that work the first time, for a long time.

TimeMill

Dynamic Verification and Timing Analysis

In the pre- and post-layout cycles, TimeMill? provides high-speed, high-capacity circuit simulation, comprehensive timing diagnostics and critical path timing diagnostics. By identifying

functional and timing problems before tapeout, TimeMill helps users to achieve timing closure, speed system-on-a-chip integration and avoid costly re-spins. The Analog Circuit Engine (ACE) enables TimeMill to handle mixed-signal design analysis by providing detailed transistor-level accuracy of the analog and mixed A/D portions of a design.

VCS

High-Performance Verilog Simulation

The VCS Verilog simulator, integrated with RoadRunner and Radiant technology, offers significantly faster simulation performance without requiring any design or methodology changes for all phases of design. It is the only high-performance simulator supported by over 150 of the most advanced technology libraries currently available from every major semiconductor vendor worldwide. VCS is the only simulator offering Direct Kernel Integration to maximize performance with key verification technologies such as CoverMeter, VERA and Semiformal verification tools. VCS is available on all UNIX, Linux and Windows platforms.

Herramientas de Cadence para digital IC design

ENCOUNTER DIGITAL IC DESIGN PLATFORM products

First Encounter

First Encounter? pioneered virtual prototyping for digital designs, bringing physical knowledge to front-end logic designers

SoC Encounter

Nanometer integrated circuits (ICs) demand. SoC Encounter supports a wire-centric methodology that accounts for the effects of interconnect across the entire chip from the very beginning of the implementation cycle by combining RTL synthesis, silicon virtual prototyping, and full-chip implementation into a single system

Encounter RTL Compiler

New-generation synthesis technology with a unique global logic structure for timing closure

Encounter Conformal

Encounter Conformal offers the most comprehensive solution for equivalence checking, design-constraint management, and low-power design verification

Encounter Test

Cadence? Encounter? Test delivers the industry's most advanced test solution from RTL to silicon

CeltIC NDC

CeltIC? Nanometer Delay Calculator (NDC) is an SI-aware delay calculator that provides you with a unified timing solution that accurately accounts for the impact of crosstalk and IR drop on both delay and functionality

NanoRoute Ultra

NanoRoute^(TM) is a separate routing, optimization, verification, and chip-finishing technology that also is at the core of the Cadence SoC Encounter^(TM) RTL-to-GDSII system

VoltageStorm

VoltageStorm? power-grid verification is the leading power-grid verification solution, and the first transistor-accurate power-grid analysis product fast enough for use during physical design.

SignalStorm NDC

SignalStorm? NDC (nanometer delay calculator) is a new-generation, fast hierarchical delay calculator for cell-based designs

Fire & Ice QXC

Fire & Ice QXC has redefined the accuracy requirements for cell-based digital designs -- it is 2X more accurate as other extraction technologies and handles in-die process variations that often occur in advanced processes as 130nm and below

Otras herramientas son PKS-LPS, para la síntesis de baja potencia, y Power Analyzer, dentro de Silicon Ensemble (hace falta el layout).

Nomenclatura:

- **LEF (Library Exchange Format):** proporciona información sobre las master cells.
- **DEF (Design Exchange Format):** proporciona información sobre el layout físico del diseño.
- **GCF (General Constraint Format):** proporciona información sobre el supply voltage. Tiene que tener un “pointer” a un fichero TLF.
- **TLF (Timing Library Format):** contiene información de la librería de potencia.
- **RSPF (Reduced Standard Parasitics Format):** da una información de los parásitos más exacta para análisis de potencia.
- **TWF (Timing Window Format):** da información sobre el input slew rate del diseño. Puede ser generado por un “timing analyzer”, como Pearl Timing Analyzer.
- **VCD (Value Change Dump):** para hacer análisis de potencia dinámicos donde se tenga en cuenta el tiempo. Contiene información sobre el cambio de valores en las redes seleccionadas del diseño, y es generado por el simulador Verilog-XL. Este fichero es muy grande, ya que contiene todos los eventos, así que las simulaciones con él serán muy lentas.
- **SAF (Static Activity Format):** para correr un análisis de potencia estático si no tenemos archi VCD ni TCF (Toggle Count Format). Contiene la net switching activity en términos de un porcentaje que es relativo a una frecuencia de reloj especificada.
- **DAF (Dynamic Activity Format):** es una alternativa de Cadence al VCD para hacer análisis de potencia dinámicos. Es similar al VCD, también describe la switching activity a cualquier tiempo dado. Sin embargo no describe toda la switching activity, sólo patrones o vectores predefinidos de switching descritos en el fichero PSF (Power Specification Format) o en la librería TLF. Se puede hacer la simulación con el VCD y para de manera opcional guardar la infirmación hacerlo con el DAF (que ocupa menos). También podemos hacer la simulación con el DAF si queremos que dure menos tiempo.
- **TCF (Toggle Count Format):** da la toggle count information (eventos) de cada net del diseño para el power analyzer.

- **PSF (Power Specification Format)**: nos permite añadir información que no está descrita en la librería TLF, en el fichero LEF, o en otras restricciones del diseño. Por ejemplo, si el TLF no proporciona información de potencia sobre algunas celdas o si hemos precomputado datos de caracterización de celdas, podemos suministrar un fichero PSF que contenga la información de potencia para esas celdas. El power analyzer crea un fichero .psf cuando hacemos un análisis estático de potencia.

Low Power for BuildGates Synthesis and Cadence PKS:

- Es una solución completamente automatizada para conseguir una síntesis low-power partiendo de un alto nivel RTL.

- Flujo resumido:

4. Crear código RTL
5. Usar LPS para explorar el código RTL, identificando los candidatos para conseguir baja potencia y marcarlos en vez de cambiarlos en el diseño
6. Optimizar el diseño usando LPS
7. Analizar los resultados del power saving

- LPS explora el diseño para determinar dónde puede hacer ahorros de potencia insertando clock-gating logic para los bancos de registros y sleep-mode logic para los bloques funcionales.

- Un análisis detallado de la potencia es hecho después de simular el gate-level netlist. La simulación proporciona gate-level switching activity en un formato TCF (Toggle Count Format), fichero .tcf que ayuda para la optimización de potencia gate-level.

- Durante la optimización que sigue la optimización, LPS finaliza la inserción de la lógica de clock-gating y sleep-mode que salvaría potencia sin tener impacto en el timing del camino crítico.

- Al nivel RTL, LPS no inserta realmente el sleep-mode logic, pero marca dónde pueden ser añadidos los candidatos en el diseño.

- Después del place and route, LPS hace gate-level power optimization que incluyen clock-gating logic y sleep-mode logic, y otras transformaciones como gate sizing, pin swapping, buffer removal, gate merging, slew optimization y power-based logic restructuring.

- Para power reporting, LPS permite:

1. Generar un power report detallado
2. Cuestionar la potencia de instances y nets en el esquemático
3. Realizar análisis estadísticos
4. Anotar la potencia en module browser y esquemático
5. Usar colorización en el module browser, esquemático y layout

- Pasos del flujo PKS-LPS:

1) Leer las librerías:

Las librerías de tiempos, síntesis y físicas. LPS calcula la potencia de las instances basándose en la información de potencia o energía disponibles en las librerías (.tlf ,cadence timing library format, o .alf, ambit library format). Las librerías físicas son las .lef (vendrán dadas por la tecnología)

Comandos: **read_tlf** < ... >, **read_lef** < ... >

2) Sintetizar el netlist RTL:

Se lee el Verilog o VHDL y se genera la implementación hardware en la forma de un netlist genérico (technology independent).

Comandos: **read_verilog < ... >, read_vhdl < ... >, do_build_generic -sleep mode**

3) Leer las restricciones temporales:

Después de leer el netlist RTL y hacer la exploración del sleep-mode, debemos leer las restricciones temporales y de diseño.

Comandos: **set_clock < ... >, set_clock_root < ... >, set_clock_tree_constraints < ... >, set_clock_arrival_time < ... >, set_clock_gating_options < ... >**

4) Explorando el clock-gating:

La exploración puede ser invocada con el mismo comando que se usa para invocar la optimización pre-placement. Durante este paso, mapeando el circuito a las celdas de librería de tecnología es seguido por la optimización de timing. Para explorar áreas del diseño donde la potencia podría ser salvada insertando clock-gating logic, usamos el comando:

do_optimize -power -stop_for_power_simulation

5) Escribir el netlist:

Para crear el gate-level netlist en verilog necesario para la simulación:

write_verilog -hier < ... >

6) Simular el diseño:

Se simula el gate-level netlist para crear la información de la gate-switching activity. Durante este paso se crea un fichero TCF (o varios, si se cambia algo). Para generar el fichero TCF se usan rutinas PLI, y debemos añadir dos líneas al fichero de test_bench (ver pág. 113 del manual). También se puede generar el TCF a partir de un VCD, y si no, leemos un fichero donde está lo de las rutinas pli e invocando al simulador verilog (antes debemos haber inicializado verilog en la ventanita naranja antes de arrancar pks_shell -gui, debemos poner init ldv51) con el comando:

verilog -f ../sim/sim.opt

Fichero sim.out:

```
-----  
-v ../libs/demo25.v  
+libext+.v  
+loadpli1=lps_pli:lps_bootstrap  
../sim/test_bench.v  
test.v  
-----
```

7) Leer el TCF:

Una vez generado el fichero TCF, se lee:

read_tcf < ... >

Si no se lee el fichero TCF, el power optimization engine no será disparado a no ser que pongamos el power_opt_no_tcf global a true. En este caso el LPS asigna densidades de transición (y probabilidades) por defecto a todas las nets del diseño causando muchos mensajes en un log file. Llegados a este punto (con el TCF leído), podemos obtener un primer valor de la potencia sin optimizar: **get_power**

8) Optimizar la potencia:

Ahora ya optimizamos y emplazamos el diseño (place):

do_optimize -insert_clock_tree -power

Calculamos otra vez la potencia después de insertar los elementos para reducir la potencia y así ver cuánto se ha reducido: **get_power**

9) Rutado del diseño:

Después del emplazamiento y optimización, se ruta el diseño

10) Temporización post-routing y optimización de la potencia:

Después de la optimización de routing y timing, LPS puede además hacer un gate-resizing para ahorrar potencia. Para hacer la optimización post-routing:

do_xform_optimize_power

Después del routing ninguna transformación de potencia puede ser disparada salvo gate resizing

Flujo realizado para un sumador:

Vamos a seguir el flujo de la herramienta como en el tutorial pero con el ejemplo del sumador.

Partiendo del código verilog del sumador, debemos sintetizarlo, generar el fichero tcf para obtener resultados realistas de potencia, y optimizar para reducir el consumo.

Para empezar debemos leer las librerías necesarias:

```
read_tlf /mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/tlf/c35_CORELIB.tlf
read_lef /mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/lef/cmos35.lef
```

Leemos el código verilog:

```
read_verilog sumador.v
```

Se genera la implementación hardware en la forma de un netlist genérico (technology independent):

```
do_build_generic (ahora mismo tenemos puertas nand, nor, ... pero no son de ninguna librería)
```

Mapeado de las celdas de librería de tecnología seguida de optimización:

```
do_optimize -power -stop_for_power_simulation (de esta forma ya coge las celdas de la librería CORELIB)
```

Se escribe el sumador sintetizado en verilog:

```
write_verilog -hier sumador_syn6.v (en nuestro caso se llamó así porque se estuvieron practicando distintas formas de sintetización y así ver los resultados)
```

Llegados a este punto, tenemos que generar un fichero .tcf con los eventos para la obtención real del cálculo de potencia. Esto lo podemos hacer de dos formas distintas: 1) simulando el circuito a nivel de puertas con Verilog-XL, obteniendo un fichero .vcd, y pasando después ese fichero a tcf en pks otra vez, 2) haciendo unas rutinas PLI, para lo cual nos generaríamos un ficherito sim.opt, y desde pks se llamaría a verilog para que hiciese la simulación (por lo que antes de arrancar pks se debe inicializar verilog => init ldv51, y después ya pks_shell -gui &).

Vemos los resultados obtenidos usando la primera forma, esto es, generando el fichero vcd. Se lleva el verilog sintetizado a Cadence, y se sigue el flujo de diseño digital, simulación con Verilog-XL. De esta forma se genera un fichero testfixture.new que usamos para meter estímulo como test_bench. Además de introducir los cambios de las entradas en el tiempo, debemos introducir unos comandos para que haga el fichero vcd. El test_bench queda de la siguiente forma:

```
// Verilog stimulus file.
// Please do not create a module in this file.

// Default verilog stimulus.

integer counta,countb;
initial
begin

    A[7:0] = 8'b00000000;

    B[7:0] = 8'b00000000;

    Cin = 1'b0;

begin
$dumpfile ("sumador.vcd");
$dumpvars;
end

for (counta=0;counta<=256;counta=counta+1)
begin
    A=counta;
    for(countb=0;countb<=256;countb=countb+1)
        #20 B=countb;
end

Cin=1'b1;

for(counta=0;counta<=256;counta=counta+1)
begin
    A=counta;
    for(countb=0;countb<=256;countb=countb+1)
```

```
#20 B=countb;
end
end
```

En negrita está lo que hemos introducido nuevo para que guarde el fichero vcd con los eventos. De esta forma se genera el fichero sumador.vcd que tiene casi dos millones de líneas. Las primeras líneas tienen la siguiente pinta:

```
$date
  Feb 23, 2006 11:15:37
$end
$version
  Tool:    VERILOG-XL    05.10.003-s
$end
$timescale
  1ps
$end
```

```
$scope module test $end
$var wire    1 !  Cout $end
$var reg     1 "  Cin  $end
$var wire    1 #  S [7] $end
$var wire    1 $  S [6] $end
$var wire    1 %  S [5] $end
$var wire    1 &  S [4] $end
$var wire    1 '  S [3] $end
$var wire    1 (  S [2] $end
$var wire    1 )  S [1] $end
$var wire    1 *  S [0] $end
$var reg     8 +  B [7:0] $end
$var reg     8 ,  A [7:0] $end
```

```
$scope module top $end
$var wire    1 !  Cout $end
$var wire    1 -  Cin  $end
$var wire    1 #  S [7] $end
$var wire    1 $  S [6] $end
$var wire    1 %  S [5] $end
$var wire    1 &  S [4] $end
```

...

Tenemos pues el fichero vcd, volvemos a pks para pasarlo allí a tcf. Para ello tenemos que escribir el siguiente comando:

```
lpsvcd2tcf.exe test /mnt/cnm2/casram/cadence_03/ams_3.60/sumador2/sumador.run1/sumador.vcd -toggle tcf_flat -output_file sumador.tcf_flat
```

Se tiene que escribir de esta forma para que no se genere el tcf de forma jerárquica (si no al leerlo daría un montón de warnings porque no encuentra las instances y demás de esa forma), debe ser flat. El nombre test es el nombre del top module usado por el vcd parser (lo vemos en el fichero vcd). Se genera el fichero sumador.tcf_flat, pero no es igual que el que viene en el tutorial, le faltan comillas, dos puntos y puntos y comas en las líneas. Entonces al leer el fichero este, la herramienta se cierra. Tiene la siguiente pinta:

```
tcffile () {
  tcfversion   :   "1.0";
  generator    :   "BGPower Verilog PLI";
  genversion   :   "1.0";
  date         :   "Fri Feb 24 11:40:29 2006";
  duration     :   "2.641962e+06";
  unit         :   "ns";
  instance () {
    pin () {
      top/i_0/i_4/A 0.498054 32
      top/i_0/i_4/B 0.498054 8224
      top/i_0/i_4/S 0.499992 41088
      top/A[0] 0.498054 512
      top/A[1] 0.498054 256
      top/S[0] 0.500000 131071
      top/A[2] 0.498054 128
      top/S[1] 0.499992 131328
    }
  }
}
```

...

Estas líneas deberían tener una pinta como ésta:

```
"i_0/i_0/S" : "0.500000 131070";
```

Como no nos lo genera bien, pasamos a hacerlo de la segunda forma, ya sin vcd, sino con lo de las rutinas PLI. Para ello hacemos un fichero llamado sim.opt, de la siguiente forma:

```
-v /cad/KITS/CADENCE_04/ams_cdk_3.60/verilog/c35b4/c35_CORELIB.v
-v /cad/KITS/CADENCE_04/ams_cdk_3.60/verilog/c35b4/c35_UDP.v
+libext+.v
+loadpli1=lp_s_pli:lp_s_bootstrap
testfixture.template
sumador_syn6.v
```

Éste contiene los modelos verilog de las celdas usadas (librerías de AMS 3.60), y de las primitivas que hacen falta porque las celdas usadas llaman a éstas (por eso debemos poner la segunda línea). La 3ª y 4ª (ésta es para que entienda lo nuevo que pondremos en el test_bench) están igual, la 5ª es el test_bench, el cual debe tener unas modificaciones, y la última es el verilog sintetizado. El test_bench lo hicimos modificando el testfixture.template, de manera que queda:

```
‘timescale 1ns / 1ps
```

```

module test;

integer counta,countb;
wire Cout;

reg Cin;

wire [7:0] S;

reg [7:0] B;
reg [7:0] A;

sumador top(A, B, Cin, S, Cout);

initial
begin

    A[7:0] = 8'b00000000;

    B[7:0] = 8'b00000000;

    Cin = 1'b0;

end

initial
    $toggle_count(test.top);

initial
begin

for (counta=0;counta<=256;counta=counta+1)
begin
    A=counta;
    for(countb=0;countb<=256;countb=countb+1)
        #20 B=countb;
end

Cin=1'b1;

for(counta=0;counta<=256;counta=counta+1)
begin
    A=counta;
    for(countb=0;countb<=256;countb=countb+1)

```

```
#20 B=countb;  
end
```

```
$toggle_count_report_flat("test.tcf","test.top");
```

```
$finish;  
end
```

```
endmodule
```

Otra vez en negrita está lo nuevo para que genere el tcf.
Hecho esto se genera un fichero llamado test.tcf, de la forma:

```
tcffile () {  
  tcfversion   :   "1.0";  
  generator    :   "BGPower Verilog PLI";  
  genversion   :   "1.0";  
  date         :   "Mon Mar  6 09:03:46 2006";  
  duration     :   "2.641960e+06";  
  unit         :   "ns";  
  instance () {  
    pin () {  
      "i_0/i_0/S" :   "0.500000 131070";  
      "i_0/i_1/S" :   "0.499992 131326";  
      "i_0/i_2/S" :   "0.499992 98558";  
      "i_0/i_3/S" :   "0.499992 65726";  
      "i_0/i_4/S" :   "0.499992 41086";  
      "i_0/i_5/S" :   "0.499992 24654";  
      "i_0/i_6/S" :   "0.499992 14382";  
      "i_0/i_7/S" :   "0.499992 8218";  
      "i_0/i_7/CO" :  "0.496131 4111";  
      "i_0/i_0/B" :   "0.500000 1";  
      "i_0/i_0/CI" :  "0.498054 512";  
      "i_0/i_1/A" :   "0.498054 256";  
      "i_0/i_0/A" :   "0.498054 131583";  
      "i_0/i_2/A" :   "0.498054 128";  
      "i_0/i_1/B" :   "0.498054 65791";  
      ...  
    }  
  }  
}
```

Por tanto lo leemos con read_tcf, vemos la potencia consumida con get_power. Obtenemos 0.1411 mW (en el top), y 0.1141 en el esquemático.



Module	Instance	Library Cell	Internal Power (mW)	Leakage Power (mW)	Net Power (mW)	Total Power (mW)
sumador			0.0848	1.122e-06	0.0563	0.1411

AWDP_ADD_0						
Module	Instance	Library Cell	Internal Power (mW)	Leakage Power (mW)	Net Power (mW)	Total Power (mW)
AWDP_ADD_0			0.0848	1.122e-06	0.0293	0.1141

Como vemos, la diferencia está en la net power.

Optimizamos con `do_optimize -power`, se crea un nuevo esquemático, y si calculamos otra vez la potencia obtenemos 0.0783 mW, aunque en el esquemático. En el top sale 0.1008. Si hacemos un `report power` podemos ver los resultados (potencia interna, de leakage, de net, de cada instance, etc.)

Report	report_power
Options	-module
Date	20060306.124303
Tool	pks_shell64
Release	v5.14-s078
Version	May 29 2004 11:13:17
Module	AWDP_ADD_0
Power Operating Library	Typical
Process	1.000000
Voltage	3.300000
Temperature	25.000000
time unit	1.00 ns
capacitance unit	1.00 pF
power unit	1.00mW

		Library	Internal	Leakage	Net	Total	
		Cell					

Module	Instance	Cell	Power (mW)	Power (mW)	Power (mW)	Power (mW)	Power (mW)

	i_0/i_13	XOR30	1.086e-03	1.163e-07	1.420e-04	1.228e-03	
	i_0/i_8	MAJ31	2.415e-04	7.649e-08	1.291e-04	3.707e-04	
	i_0/i_7	XOR30	5.684e-04	1.163e-07	2.088e-04	7.773e-04	
	i_0/i_12	MAJ31	7.181e-04	7.649e-08	7.466e-04	1.465e-03	
	i_0/i_11	XOR30	1.686e-03	1.163e-07	3.223e-04	2.008e-03	
	i_0/i_10	MAJ31	4.155e-04	7.649e-08	4.809e-04	8.965e-04	
	i_0/i_4	ADD31	4.003e-03	1.396e-07	2.096e-03	6.099e-03	
	i_0/i_3	ADD31	6.215e-03	1.396e-07	5.013e-03	0.0112	
	i_0/i_2	ADD31	9.337e-03	1.396e-07	6.140e-03	0.0155	
	i_0/i_1	ADD31	0.0126	1.396e-07	6.749e-03	0.0193	
	i_0/i_0	ADD31	0.0127	1.396e-07	6.729e-03	0.0194	

AWDP_ADD_0							
		Library	Internal	Leakage	Net	Total	
		Cell					

Module	Instance	Cell	Power (mW)	Power (mW)	Power (mW)	Power (mW)	Power (mW)

AWDP_ADD_0			0.0495	1.276e-06	0.0288	0.0783	

Estos cálculos de potencia están hechos con el tcf anterior, lo que debemos hacer es escribir otro netlist y re-simular. Ahora sale como potencia 0.1008 mW, pero era porque estábamos en el top, no en el esquemático, y en el top sale un poco más debido a la net power, pero vemos que en este caso tras re-simular vuelven a salir los mismos valores de potencia.

sumador							
		Library	Internal	Leakage	Net	Total	
		Cell					

Module	Instance	Cell	Power (mW)	Power (mW)	Power (mW)	Power (mW)	Power (mW)

sumador			0.0495	1.276e-06	0.0513	0.1008	

+-----+

O sea pasamos en el top de 0.1411mW => 0.1008 mW, y en el esquemático de 0.1141 mW => 0.0783 mW.

Flujo realizado para circuito del ITC'99 : b01

Vamos a intentar hacer lo mismo con otro circuito, de manera que lo tendremos que hacer todo, sintetizar, realizar el test_bench ...

En este caso se parte de un circuito en VHDL, pero la única diferencia sería poner read_vhdl en vez de read_verilog:

1. Leer librerías tlf y lef
2. Leer VHDL
3. do_build_generic
4. do_optimize -power -stop_for_power_simulation
5. write_verilog -hier b01_syn.v

Hasta aquí sin problemas. Ahora debemos simular este verilog a nivel de puertas, y para ello nos hace falta generar el test_bench en verilog. O bien lo escribimos desde cero, todo nosotros, para lo cual hay que saber cómo es la estructura de dicho fichero, o podemos importar el diseño a Cadence y allí al arrancar Verilog-XL y estímulos, se generan unos templates, que tienen el esqueleto, y las entradas inicializadas. De esta forma sólo tenemos que escribir cómo cambian las entradas en el tiempo, así como las líneas necesarias para que se genere el fichero tcf. Para importar a Cadence, nos vamos al directorio /cadence_03/ams_3.60/sumador2, donde ya habíamos arrancado antes Cadence: tools => cadence 03/04 => 3.60 (Europractice). Nos hacemos una librería llamada b01 atacheada a c35b4, y para importar en verilog sintetizado ponemos en Reference libraries: sample basic CORELIB, y en las -f options: /cad/KITS/CADENCE_03/ams_cdk_3.60/verilog/c35b4/verilogin.inc. Una vez importado correctamente nos vamos a simulación, Verilog-XL, y estímulos, de manera que se crean los templates:

“testfixture.template”

‘timescale 1ns / 1ps

module test;

 wire outp, overflow;

 reg clock, line1, line2, reset;

b01 top(line1, line2, reset, outp, overflow, clock);

(aquí debemos poner otro testfixture, testfixture.verilog)

endmodule

Podemos ver como usa *reg* para las entradas y *wire* para las salidas.

“testfixture.verilog”

initial

```
begin
  clock = 1'b0;
  line1 = 1'b0;
  line2 = 1'b0;
  reset = 1'b0;
end
```

Después de este testfixture.verilog debemos introducir nuestros estímulos, pero antes escribir:

```
initial
  $toggle_count(test.top);
```

Estímulos y para terminar:

```
$toggle_count_report_flat("b01.tcf","test.top");
```

Nos hacemos un fichero llamado b01_tb.v, que tiene la siguiente forma:

```
“b01_tb.v”
`timescale 1ns / 1ps

module test;
  wire outp, overflow;
  reg clock, line1, line2, reset;

  b01 top(line1, line2, reset, outp, overflow, clock);

  //inicializacion de las entradas

  initial
  begin
    clock = 1'b0;
    line1 = 1'b0;
    line2 = 1'b0;
    reset = 1'b0;
  end

  initial
    $toggle_count(test.top);

  //estimulos

  $toggle_count_report_flat("b01.tcf","test.top");
  $finish;
end (de los estímulos)
```

```
endmodule
```

De esta forma lo que queda por hacer es introducir dichos estímulos. Lo hacemos de la siguiente forma:

```
always  
#10 clock = !clock;
```

```
initial begin  
#30 line1 = 1;  
#30 line2 = 1;  
#30 line1 = 0;  
#30 line2 = 0;
```

Entonces queda el test_bench tal como:
'timescale 1ns / 1ps

```
module test;  
    wire outp, overflow;  
    reg clock, line1, line2, reset;
```

```
b01 top(line1, line2, reset, outp, overflow, clock);
```

```
//inicializacion de las entradas
```

```
initial  
begin  
    clock = 1'b0;  
    line1 = 1'b0;  
    line2 = 1'b0;  
    reset = 1'b0;  
end
```

```
initial  
    $toggle_count(test.top);
```

```
//estimulos
```

```
always  
#10 clock = !clock;
```

```
initial begin  
#30 line1 = 1;  
#30 line2 = 1;  
#30 line1 = 0;
```

```

#30 line2 = 0;
//end

$stoggle_count_report_flat("b01.tcf","test.top");

//initial
    #200 $finish;

end

endmodule

```

Hacemos la simulación, que tal y como hemos escrito el test_bench dura 320 ns (30+30+30+30+200), y se genera el archivo b01.tcf, que si lo vemos tiene muchos ceros porque no habrá muchos cambios. Debemos hacer algo para que se produzcan más cambios, y así tener un resultado lo más realista posible. De esta forma, si leemos el tcf y calculamos la potencia, nos sale 0.0874 mW. Si optimizamos y volvemos a calcular la potencia, sale 0.0833 mW. Pero no habíamos metido nada de reloj (set_clock ...), ningunas restricciones (Clock constraints are not complete. Not performing do_xform_optimize_clock_gate. Please source clock-tree constraints before invoking this command). Tenemos que ver qué debemos introducir como información del reloj (skew, propagation delay, ...). Tenemos que definir un reloj ideal, como una señal de referencia global para todas las señales de datos en el diseño. Esto se hace con el comando set_clock:

```

set clock clock_name {[-period period] | [-waveform {lead_time trail_time}]}
set_clock ideal_clock -period 10 -waveform {3 8}

```

Como vemos es un reloj de periodo 10, con el primer cambio en 3, y el segundo en 8 (si no tuviese ese retraso inicial de 3, sería {0 5}). Si hay varios relojes se pueden escribir varios set_clock. Para indicar el primer cambio cómo es, usamos el comando:

```

set_clock_root -clock ideal_clock_name [-pos | -neg] list_of_pins
set_clock_root -clock ideal_clock neg clk

```

Otra cosa que ponemos son las restricciones del reloj, retraso y skew:

```

set_clock_tree_constraints -pin clock -min_delay .0 -max_delay .5 -max_skew .0

```

Escrito esto, haríamos ahora el primer optimize, el de antes de la simulación, para tener el circuito mapeado a la tecnología, y poder hacer la simulación a nivel de puertas. Después haríamos otro optimize, donde se generará un circuito que optimice al anterior. El orden de comandos sería una cosa tal como:

```

read_tlf /mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/tlf/c35_CORELIB.tlf
read_lef /mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/lef/cmos35.lef
read_vhdl b01.vhd

```

```

do_build_generic
set_clock CLOCK -waveform {0 5} -period 10
set_clock_root -clock CLOCK -pos clock
set_clock_tree_constraints -pin clock -min_delay .0 -max_delay .5 -max_skew .0
//set_clock_arrival_time -clock CLOCK -rise 0.2 -fall 5.2 clock
do_optimize -power -stop_for_power_simulation
write_verilog -hier b01_syn.v
verilog -f b01.opt
read_tcf b01.tcf
get_power
do_optimize -power -insert_clock_tree
get_power
get_clock_tree_power clock

```

Nos sale de potencia, tras el “do_optimize -power -insert_clock_tree”, 0.3293 mW, siendo del reloj 0.2948 mW.

Flujo realizado para UMC 0.13

Vamos a hacer lo mismo, pero para UMC 0.13, de manera que debemos coger las librerías adecuadas. Nos hacemos un script para no tener que estar escribiendo los mismos comandos siempre que arranquemos. En este caso tiene la forma:

b01_UMC13.tcl

```

read_tlf /cad/KITS/UMC/UMC_13/VST/UMCE13H210D3_1.2/tlf/
umce13h210t3_tc_120V_25C_4.3.tlf
read_lef /mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/lef/umc13.lef
read_vhdl b01.vhd
do_build_generic
set_clock CLOCK -waveform {0 5} -period 10
set_clock_root -clock CLOCK -pos clock
set_clock_tree_constraints -pin clock -min_delay .0 -max_delay .5 -max_skew .0
do_optimize -power -stop_for_power_simulation
write_verilog -hier b01_syn.v
verilog -f b01_UMC13.opt
read_tcf b01.tcf
get_power
do_optimize -power -insert_clock_tree
get_power
get_clock_tree_power clock

```

Como podemos ver, se lee la librería tlf y lef. Para la lef tuvimos que concatenar dos ficheros lef, y añadir unas líneas al nuevo fichero. Para concatenarlos, en el directorio donde están escribimos:

```
cat header_8m2t_5.4.lef umce13h210t3_5.4.lef > /mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/lef/umc13.lef
```

y dentro del fichero debemos escribir en la línea 1201, justo antes de la definición de la primera macro, lo siguiente:

```
VERSION 5.4 ;
NAMESCASESENSITIVE ON ;
BUSBITCHARS "[]" ;
DIVIDERCHAR "/" ;
UNITS
  DATABASE MICRONS 1000 ;
END UNITS
```

```
MACRO HDADFULD1
```

```
...
```

Con esto ya sí se cargan bien las librerías. Luego se lee el código vhdl, se crea la construcción genérica independiente de la tecnología, se define el reloj y sus restricciones. Se mapea ya a puertas de la librería, con lo que tendríamos ya el circuito a nivel de puertas, lo que guardamos como verilog sintetizado, y que será el que usemos en la simulación para obtener el fichero tcf. Para la simulación nos hicimos un fichero b01_UMC13.opt:

```
-v /cad/KITS/UMC/UMC_13/VST/UMCE13H210D3_1.2/cadence/verilog/UMCE13H210T3/UMCE13H210T3.v
-v /cad/KITS/UMC/UMC_13/VST/UMCE13H210D3_1.2/cadence/verilog/UMCE13H210T3/UDPS.v
+libext+.v
+loadpli1=lps_pli:lps_bootstrap
b01_tb.v
b01_syn.v
```

que lee la librerías verilog, el test_bench (que también nos hacemos), el verilog sintetizado a nivel de puertas de nuestro circuito, así como el comando para lo de las rutinas pli y los nuevos comandos que se deben introducir en el test_bench. El test_bench tiene la misma forma que con AMS, pero en este caso le hemos cambiado los estímulos, que han quedado de la siguiente forma:

```
initial
  $toggle_count(test.top);
```

```
//estimulos
```

```
always
#10 clock = !clock;
```

```

initial begin
#15 reset = 0;
#30 line1 = 1;
#35 line2 = 1;
//end

$toggle_count_report_flat("b01.tcf","test.top");

//initial
    #500 $finish;

end

endmodule

```

Con esto ejecutamos el comando verilog -f b01_UMC13.opt, y se crea el fichero b01.tcf, el cual leemos para obtener un valor de la potencia basada en los estímulos. Si ahora calculamos la potencia (get_power), obtenemos: 0.0271 mW, y si vemos la del reloj (get_clock_tree_power clock), la de este es: 0.0191 mW. Vemos que el reloj consume el 70 % de la potencia. Con la herramienta podemos ver la potencia consumida por cada net y por cada instance. Por ejemplo:

```
Instance power (stato_reg_0): cell(3.430000 uW), leakage(0.010000 uW), net(0.880000 uW), total(4.320000 uW)
```

Para el circuito completo:

```

+-----+
| Report      | report_power  |
+-----+-----+
| Options     |                |
+-----+-----+
| Date        | 20060316.144635 |
| Tool        | pks_shell64    |
| Release     | v5.14-s078     |
| Version     | May 29 2004 11:13:17 |
+-----+-----+
| Module      | b01            |
| Power Operating Library | Typical      |
| Process     | 1.000000      |
| Voltage     | 1.200000      |
| Temperature  | 25.000000     |
| time unit   | 1.00 ns       |
| capacitance unit | 1.00 pF      |
| power unit  | 1.00mW        |
+-----+-----+

```

b01						
Library	Internal	Leakage	Net	Total		
Cell						
Module	Instance	Cell	Power (mW)	Power (mW)	Power (mW)	Power (mW)
b01			0.0177	1.151e-04	9.301e-03	0.0271

También en el esquemático podemos hacer que nos salga el valor de la potencia consumida por cada celda y cada net, así que pinte el esquemático de colores según el consumo. Otra cosa que se puede ver son los tiempos:

Instance	Arc	Cell	Delay	Arrival	Required
			Time	Time	
	clock ^		0.00	9.30	
stato_reg_1	CK ^ -> Q v	HDDFFRPQ1	0.14	0.14	9.44
i_108	A v -> Z ^	HDINVD1	0.12	0.26	9.55
i_5	A1 ^ -> Z v	HDNOR2D1	0.08	0.34	9.63
i_18	A2 v -> Z ^	HDOAI21M10D1	0.10	0.44	9.73
i_36	B ^ -> Z v	HDOAI31M10D1	0.03	0.47	9.77
i_31	C v -> Z v	HDAO211DL	0.16	0.63	9.93
stato_reg_0	D v	HDDFFRPQ1	0.00	0.63	9.93

Así en el report, y también lo podemos ver de manera gráfica, algo parecido al pearl. En conclusión, podemos ver bastantes cosillas, como el área, la potencia, los tiempos, sin más que darle a report ...

En este punto tenemos 28 instancias.

Ahora debemos hacer la optimización. Si ejecutamos el comando: `do_optimize -power -insert_clock_tree`, se realizan las siguientes operaciones:

```
# PKS-WLM timing optimization
# sleep-mode commitment
# gate-level power optimization
# placement
# cloning/de-cloning/root gating
# CTPKS
# timing optimization
```

```
# clock-gating decommitment
# sleep-mode decommitment
# gate-level power optimization
```

Si ejecutamos este do_optimize de esta forma, nos sale un error en el placement, porque sale un row utilization de 379.67% :

```
--> WARNING: Standard cell row utilization of 379.67% is more than the specified maximum
row utilization of 95.00%
```

```
<PLC-167>.
```

```
--> WARNING: Grow not allowed for cluster. Will continue placement anyway. . <PLC-170>.
```

```
==> ERROR: Failure to place. Row utilization of 379.67% prevents placement. To continue
placement, set the global
```

```
variable place_over_utilized to true or insure that the floorplan is not fixed. <PLC-182>.
```

```
Command do_place -eco finished at Thu Mar 16 17:37:10 2006
```

```
using 0:0:0 Real time. Current peak memory: 196.375MB <COMMON-076>.
```

```
==> ERROR: Failure during qplace ECO. <CTPKS-213>.
```

Y si ahora miramos la potencia, sale: 0.4829 mW

```

+-----+
|                                     b01                                     |
+-----+
|                                     |                                     | | | | | | |
|      |      | Library | Internal | Leakage | Net | Total |      |
|      |      | Cell   |         |         |    |      |      |
+-----+-----+-----+-----+-----+-----+-----+
| Module | Instance | Cell | Power (mW ) | Power (mW ) | Power (mW ) | Power |
(mW ) |
+-----+-----+-----+-----+-----+-----+-----+
| b01    |          |      | 0.3847 | 1.486e-03 | 0.0967 | 0.4829 |
+-----+

```

Siendo la del reloj: 0.4784 mW (un 99% del total !!). Ahora hay 37 instancias, 9 más, debido al reloj, son distintos inversores y buffers a lo largo del circuito por la rama del reloj a los distintos elementos secuenciales. Si vemos la potencia consumida por la parte del circuito que no es reloj antes y después del optimize, se pasa de 0.0080 mW a 0.0045 mW:

Table 1: Potencia antes de do_optimize -power - insert_clock_tree, y después para UMC 0.13

	P_total	P_reloj	P_No_reloj
Antes	0.0271	0.0191	0.0080
Después	0.4829	0.4784	0.0045

Si guardamos este verilog sintetizado de ahora, hacemos otra vez la simulación verilog, leemos el nuevo tcf, y miramos la potencia, nos sale: 0.2265 mW, y para le reloj: 0.2243 mW. Hacemos el do_optimize -power -insert_clock_tree otra vez, y volvemos a mirar la potencia, vemos que sale igual que antes, 0.2265 mW y 0.2243 mW, por lo que la potencia de la parte que no es reloj sale ahora 0.0022 mW.

Con AMS 0.35:

Table 2: Potencia antes de do_optimize -power -insert_clock_tree, y después para AMS 0.35

	P_total	P_reloj	P_No_reloj
Antes (32)	0.2226	0.1513	0.0713
Después (38)	0.4108	0.3472	0.0636

Si volvemos a guardar el verilog sintetizado, simulamos, y leemos el nuevo tcf, sale como potencia total 0.4080 mW, y del reloj 0.3507 mW, la de no reloj será 0.0573 mW.

Componentes de disipación de potencia:

En el mundo de los circuitos integrados digitales, la disipación de potencia para un circuito incluye los dos tipos generales siguientes:

- Disipación de potencia estática
- Disipación de potencia dinámica

Disipación de potencia estática:

Es generalmente dependiente del estado de la celda. Puede ser definida como la potencia que es perdida mientras las señales del circuito no están conmutando activamente. Tiene principalmente dos componentes:

- *Leakage power*: causada por las junction leakages y las sub-threshold leakages. Cuando el voltaje de gate-to-source del transistor es menor que el voltaje umbral, el transistor está inversamente polarizado (reverse biases) y entonces conduce una pequeña corriente subumbral, típicamente medida en picoamperios por micra de anchura de puerta. Se forman como “diodos” entre la fuente y el drenador del transistor y la unión de pozo o substrato. Esto no sólo depende de la tecnología, sino también de la temperatura. La potencia de leakage depende principalmente de la tecnología del dispositivo, el voltaje umbral y el power supply voltage. Aunque está siendo una componente importante en la tecnología submicrométrica, su contribución es todavía insignificante comparada con otras componentes de potencia. Los valores de leakage power son obtenidos de una librería tecnológica.
- *Standby power*: disipación causada por la corriente de DC que está continuamente fluyendo desde power to ground. Esta componente es insignificante para circuitos CMOS, por lo que los algoritmos de LPS la ignoran.

Disipación de potencia dinámica:

La disipación de potencia dinámica global de una celda está basada en los efectos del voltaje y temperatura, carga, input slew rate, así como el estado de la celda. Es definida como la potencia perdida mientras un circuito está activamente conmutando a una frecuencia dada. Incluye dos tipos:

- *Switching power (or capacitive power) dissipation*: la que se consume en la carga y descarga de la capacitancia de carga. La carga capacitiva incluye la capacitancia de la red y la capacitancia de los pines de entrada. La potencia disipada por la capacidad interna es modelada como una parte de la potencia disipada por la celda. (El flujo de diseño LPS no tiene en cuenta la disipación de potencia causada por una transición parcial)
- *Short-circuit power dissipation*: la consumida cuando el circuito está ON (conduciendo ambos el PMOS y el NMOS) y una corriente fluye desde el supply voltage a los ground rails. La pendiente de la entrada, la velocidad de switching, el drive strength de los transistores, y la carga driven (cargada) determinan la magnitud y duración del short-circuit. Las reglas de diseño severas seguidas por la mayoría de los diseños, especialmente en los diseños standard cell, aseguran que esta componente es pequeña seleccionando cuidadosamente la pendiente de la señal, transistor size, y valores de carga.

Cómo el LPS modela la disipación de potencia:

LPS obtiene la potencia de las instancias de las look-up tables en la librería de síntesis o de la información de la supply current (esto es lo que nos interesa saber para intentar hacer algo parecido para el ruido) para los siguientes factores:

- Input slew rate (SR)
- Output load capacitance (C_L)

LPS tiene en cuenta los siguientes componentes para la instance power:

- Disipación de potencia de internal load capacitive y short-circuit, que son obtenidas de una look-up table en la librería de síntesis.
- Disipación de potencia leakage, que es obtenida de la anotación de potencia de leakage en la librería de síntesis.

LPS tiene en cuenta los siguientes componentes para la net power:

- Disipación de potencia en la capacitancia de la red
- Disipación de potencia en las capacitancias de los pines driven por la red.

LPS-PKS puede realizar análisis de potencia tanto estático como dinámico.

En el estático LPS estima la disipación de potencia estática a nivel de puertas en el diseño, dando la capacidad de caracterizar y evaluar varias alternativas de diseño. Esto permite hacer tradeoffs de diseño entre performance y disipación de potencia. Se tienen tres formas de obtener las net switching activities:

- 1) Si proporcionamos un fichero TCF o VCD, LPS lee este fichero
- 2) Para saltarnos la simulación podemos meter la switching activity con el comando `set_switching_activity` para algunas critical nets
- 3) Si no queremos especificar ninguna switching activity, LPS asume la switching activity por defecto en las entradas primarias y las salidas de los elementos secuenciales. La signal probability por defecto es 0.5 y la transition density por defecto es $1.0e-4$

Otra forma de realizar un análisis de potencia es estimar la potencia en un cycle-by-cycle basis. Esta información es crucial para la fiabilidad del circuito, análisis de ruido AC y DC, y optimización del diseño. El k-cycle average power puede proporcionar información del consumo de potencia para cualquier ventana de tiempo dada. Realizar estas tareas requiere el conocimiento del valor del consumo de potencia en cada ciclo de reloj. El estimador de potencia devuelve la potencia consumida en cada ciclo de reloj y la total en el periodo de simulación completo. También podemos obtener el pico de potencia consumida (éste es el problema, que sólo da un pico para toda la simulación (eso es lo que se entiende, no lo hemos probado), el mayor, pero claro no sabemos nada de los demás, por lo que para tema de ruido no nos valdría). Para realizar la estimación de potencia dinámica, necesitamos simular el diseño y generar el fichero VCD. Este fichero contiene la lista de eventos ocurriendo en las nets a diferentes sellos de tiempos. El estimador de potencia dinámica lee este fichero y calcula la potencia consumida por cada evento que ocurre en el periodo de tiempo especificado. Si cambiamos algún slew o capacitancia del circuito, debemos volver a leer el VCD y recalcular la potencia dinámica.

Con un análisis estadístico podemos ver un histograma de la toggle count, que nos da cuántas redes (number of nets) tienen una determinada toggles / s. Lo mismo se puede hacer con la probabilidad en vez del toggle.

(.....)

Calculando potencia dentro de XILINX

The Xilinx Power Tools offer the world's most full featured Web-based pre-implementation power estimator for programmable logic devices.

Web Power Tools and Spreadsheet Tools are pre implementation tools that estimate a design's power consumption based on the expected utilization of device resources, operating frequencies and toggle rates.

Once the design has been implemented in the Xilinx software tools, XPower can be used to estimate with more precise accuracy the power consumption of your design. Actual power consumption must be determined in circuit under the appropriate operating conditions.

Web Power Tool Quick Start Guide

Xilinx provides Web Power Tools for pre-implementation power estimates and XPower tools for post implementation power analysis. We believe that both our Web Power and XPower tools deliver results that correlate well to actual silicon measurements when used correctly. The best power estimates are those done on a completed and routed design that has been loaded into the XPower tool and stimulated with a functionally accurate set of stimulus vectors.

Integrated Power Tools

XPower is the first power-analysis software available for programmable logic design allowing analysis of total device power, power per-net, routed, partially routed or unrouted designs.
(XPower)

Spreadsheet Power Tools

The Spartan-IIIE Spreadsheet Power Tool is designed to generate accurate power estimates based on the expected utilization of the various device resources, operating frequencies and toggle rates and can still be power estimated using the spreadsheet link below.
(Spartan-IIIE)

LPS-PKS (CADENCE)

b01_UMC13.tcl

```
read_tlf /cad/KITS/UMC/UMC_13/VST/UMCE13H210D3_1.2/tlf/  
umce13h210t3_tc_120V_25C_4.3.tlf  
read_lef /mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/lef/umc13.lef  
read_vhdl b01.vhd  
do_build_generic  
set_clock CLOCK -waveform {0 5} -period 10  
set_clock_root -clock CLOCK -pos clock  
set_clock_tree_constraints -pin clock -min_delay .0 -max_delay .5 -max_skew .0  
set_clock_arrival_time -clock CLOCK -rise 0.2 -fall 5.2 clock  
do_optimize -power -stop_for_power_simulation  
write_verilog -hier b01_syn.v  
verilog -f b01_UMC13.opt  
read_tcf b01.tcf  
get_power  
do_optimize -power -insert_clock_tree  
get_power  
get_clock_tree_power clock
```

b01_UMC13.opt

```
-v /cad/KITS/UMC/UMC_13/VST/UMCE13H210D3_1.2/cadence/verilog/UMCE13H210T3/  
UMCE13H210T3.v  
-v /cad/KITS/UMC/UMC_13/VST/UMCE13H210D3_1.2/cadence/verilog/UMCE13H210T3/  
UDPS.v  
+libext+.v  
+loadpli1=lps_pli:lps_bootstrap  
b01_tb.v  
b01_syn.v
```

b01_tb.v

```
`timescale 1ns / 1ps  
  
module test;  
    wire outp, overflow;  
    reg clock, line1, line2, reset;  
  
    b01 top(line1, line2, reset, outp, overflow, clock);  
  
    //inicializacion de las entradas
```

```

initial
begin
    clock = 1'b0;
    line1 = 1'b0;
    line2 = 1'b0;
    reset = 1'b1;
end

initial
    $toggle_count(test.top);

//estimulos

always
#10 clock = !clock;

initial begin
#15 reset = 0;
#30 line1 = 1;
#35 line2 = 1;
//end

$toggle_count_report_flat("b01.tcf","test.top");

//initial
    #500
$finish;

end

endmodule

```

b01_UMC13_vcd.opt

```

-v /cad/KITS/UMC/UMC_13/VST/UMCE13H210D3_1.2/cadence/verilog/UMCE13H210T3/
UMCE13H210T3.v
-v /cad/KITS/UMC/UMC_13/VST/UMCE13H210D3_1.2/cadence/verilog/UMCE13H210T3/
UDPS.v
+libext+.v
+loadpli1=lps_pli:lps_bootstrap
b01_tb_vcd.v
b01_syn.v

```

b01_tb_vcd.v

```

`timescale 1ns / 1ps

```

```

module test;
  wire outp, overflow;
  reg clock, line1, line2, reset;

b01 top(line1, line2, reset, outp, overflow, clock);

//inicializacion de las entradas

initial
begin
  clock = 1'b0;
  line1 = 1'b0;
  line2 = 1'b0;
  reset = 1'b1;
end

always
#10 clock = !clock;

initial begin
  $dumpfile ("b01.vcd");
  $dumpvars;
end

initial
#100 $finish;

endmodule

```

b01.tcl

```

#####
#   link design
#####
set search_path      "/mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador /cad/KITS/
UMC/UMC_13/VST/Tapeout_kit
s/UMCE13H210T3_1.2/design_compiler ."
set link_library     "* umce13h210t3_tc_120V_25C.db"

read_verilog        b01_syn.v
current_design      b01
link

#####

```

```
# set transition time / annotate parasitics
#####
set_input_transition 0.1 [all_inputs]
create_clock -period 10 -waveform {0 5} {clock}

#####
# power analysis
#####ed switching activity file
#####
read_vcd -strip_path test/top /mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/
sumador/b01.vcd

#####
set_waveform_options -file b01_vcd -format out
calculate_power -waveform -statistics
report_power -file b01_vcd

#quit
```

Se arranca la herramienta PKS en el directorio: cadence_03/spr_5.0_usr4/ams_3.60/sumador, y tenemos otro directorio para el b04: cadence_03/spr_5.0_usr4/ams_3.60/b04, donde están los específicos del b04 (b04.vhd, b04_tb.v, b04_UMC13.opt, b04_tb_vcd.v, b04_UMC13_vcd.opt) y se generan todos los ficheros (b04_syn.v, b04.tcf, b04.vcd).

Luego para PrimePower lo arrancamos desde el directorio: primepower/tutorial/pp_vcd, y guardamos los ficheros necesarios (b04.tcl) y las salidas (b04_vcd.fsdb, b04_vcd.rpt, b04_vcd.out) en primepower/tutorial/pp_vcd/b04

b04_UMC13.tcl

```
read_tlf /cad/KITS/UMC/UMC_13/VST/UMCE13H210D3_1.2/tlf/
umce13h210t3_tc_120V_25C_4.3.tlf
read_lef /mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/lef/umc13.lef
read_vhdl ../b04/b04.vhd
do_build_generic -sleep_mode
set_clock CLOCK -waveform {0 5} -period 10
set_clock_root -clock CLOCK -pos clock
set_clock_tree_constraints -pin clock -min_delay .0 -max_delay .5 -max_skew .0
set_clock_arrival_time -clock CLOCK -rise 0.2 -fall 5.2 clock
do_optimize -power -stop_for_power_simulation
write_verilog -hier ../b04/b04_syn.v
verilog -f ../b04/b04_UMC13.opt
read_tcf ../b04/b04.tcf
get_power
do_optimize -power -insert_clock_tree
get_power
get_clock_tree_power clock
write_verilog -hier ../b04/b04_syn.v
verilog -f ../b04/b04_UMC13.opt
read_tcf ../b04/b04.tcf
get_power
get_clock_tree_power clock
write_verilog -hier ../b04/b04_syn.v
verilog -f ../b04/b04_UMC13_vcd.opt
```

b04_UMC13.opt

```
-v /cad/KITS/UMC/UMC_13/VST/UMCE13H210D3_1.2/cadence/verilog/UMCE13H210T3/
UMCE13H210T3.v
-v /cad/KITS/UMC/UMC_13/VST/UMCE13H210D3_1.2/cadence/verilog/UMCE13H210T3/
UDPS.v
+libext+.v
+loadplil=lps_pli:lps_bootstrap
../b04/b04_tb.v
../b04/b04_syn.v
```

b04_tb.v

```
'timescale 1ns / 1ps
```

```
module test;
```

```
    wire [7:0] DATA_OUT;
```

```
    reg RESTART, AVERAGE, ENABLE, RESET,CLOCK ;
```

```
    reg [7:0] DATA_IN;
```

```
b04 top(
```

```
.DATA_OUT (DATA_OUT),
```

```
.RESTART (RESTART),
```

```
.AVERAGE (AVERAGE),
```

```
.ENABLE (ENABLE),
```

```
.RESET (RESET),
```

```
.CLOCK (CLOCK),
```

```
.DATA_IN (DATA_IN)
```

```
);
```

```
//inicializacion de las entradas
```

```
initial
```

```
begin
```

```
    CLOCK = 1'b0;
```

```
    RESTART = 1'b0;
```

```
    AVERAGE = 1'b0;
```

```
    ENABLE = 1'b0;
```

```
    RESET = 1'b0;
```

```
    DATA_IN = 8'b00000000;
```

```
end
```

```
always
```

```
#5 CLOCK =!CLOCK;
```

```
initial
```

```
    $toggle_count(test.top);
```

```
initial
```

```
begin
```

```
#500
```

```
$toggle_count_report_flat("../b04/b04.tcf","test.top");
```

```
$finish;
```

```
end
```

```
endmodule
```

b04_UMC13_vcd.opt

```
-v /cad/KITS/UMC/UMC_13/VST/UMCE13H210D3_1.2/cadence/verilog/UMCE13H210T3/  
UMCE13H210T3.v
```

```
-v /cad/KITS/UMC/UMC_13/VST/UMCE13H210D3_1.2/cadence/verilog/UMCE13H210T3/  
UDPS.v
```

```
+libext+.v
```

```
+loadpli1=lps_pli:lps_bootstrap
```

```
../b04/b04_tb_vcd.v
```

```
../b04/b04_syn.v
```

b04_tb_vcd.v

```
`timescale 1ns / 1ps
```

```
module test;
```

```
  wire [7:0] DATA_OUT;
```

```
  reg RESTART, AVERAGE, ENABLE, RESET,CLOCK ;
```

```
  reg [7:0] DATA_IN;
```

```
b04 top(
```

```
  .DATA_OUT (DATA_OUT),
```

```
  .RESTART (RESTART),
```

```
  .AVERAGE (AVERAGE),
```

```
  .ENABLE (ENABLE),
```

```
  .RESET (RESET),
```

```
  .CLOCK (CLOCK),
```

```
  .DATA_IN (DATA_IN)
```

```
);
```

```
//inicializacion de las entradas
```

```
initial
```

```
begin
```

```
  CLOCK = 1'b0;
```

```
  RESTART = 1'b0;
```

```
  AVERAGE = 1'b0;
```

```
  ENABLE = 1'b0;
```

```
  RESET = 1'b0;
```

```
  DATA_IN = 8'b00000000;
```

```
end
```

```
always
```

```
#5 CLOCK = !CLOCK;
```

```
initial begin
  $dumpfile ("../b04/b04.vcd");
  $dumpvars;
end
```

```
initial
#500 $finish;
```

```
endmodule
```

b04.tcl

```
#####
```

```
# link design
```

```
#####
```

```
set search_path "/mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/b04 /cad/KITS/UMC/UMC_13/VST/Tapeout_kits/UMCE13H210T3_1.2/design_compiler . "
```

```
set link_library " * umce13h210t3_tc_120V_25C.db"
```

```
read_verilog b04_syn.v
```

```
current_design b04
```

```
link
```

```
#####
```

```
# set transition time / annotate parasitics
```

```
#####
```

```
set_input_transition 0.1 [all_inputs]
create_clock -period 10 -waveform {0 5} {CLOCK}
```

```
#####
```

```
# read switching activity file
```

```
#####
```

```
read_vcd -strip_path test/top /mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/b04/b04.vcd
```

```
#####
```

```
# power analysis
```

```
#####
```

```
set_waveform_options -file b04/b04_vcd
calculate_power -waveform -statistics
report_power -file b04/b04_vcd
```

```
#quit
```

PrimePower

- PrimePower es una herramienta de análisis de potencia gate-level que analiza con exactitud la disipación de potencia de diseños basados en celdas.

- Tipos de análisis:

6. Average analysis:

Para análisis de potencia promedios, soporta la propagación de la switching activity basada en defaults, user-defined switching or switching derivado de una simulación HDL (RTL o gate-level).

7. Dynamic (peak power) analysis:

Para análisis extremadamente más exactos de la potencia con respecto al tiempo, soporta análisis basado en gate-level simulation activity a través del tiempo.

- PrimePower construye un perfil de potencia detallado del diseño basado en la conectividad del circuito, la switching activity, la net capacitance y el cell-level power behavior data en la librería .db de Synopsys.

- Reporta la potencia consumida a los niveles de chip, bloque y celdas.

- El flujo de simulación de PrimePower comprende dos fases:

1. Generación de la switching activity

2. Generación del perfil de potencia

- Realiza los siguientes pasos para analizar la potencia del diseño:

1. Basado en la conectividad del diseño y la wire-capacitance, determina los tiempos de transición para todos los pines dentro del circuito.

2. Para análisis de potencia promedios, determina el state- and path- dependent switching para todos los nudos del diseño.

3. Accede a las tablas de potencia de las librerías de Synopsys y determina la potencia total del circuito. Para los análisis dinámicos, procesa cada actividad en el fichero de actividad para construir un perfil de potencia exacto en el tiempo y para determinar el peak power consumption.

- Se puede arrancar en modo gráfico o modo comando.

- Desde el directorio /primepower/tutorial/pp_vcd arrancamos las tools, synopsys, version 2004.06, sin tecnologia, y en la ventanita naranja escribimos: primepower. Así se arranca la herramienta, y para seguir el flujo nos hacemos un script con los comandos a seguir, que serán más o menos como siempre: leer librerías (en este caso un fichero .db), el código verilog a nivel de puertas, restricciones del reloj, ..., y como salidas, un report de la potencia (.rpt), y un fichero para visualizar la forma de onda de la potencia en el tiempo (.fsdb). Éste último o veremos después con la herramienta también de Synopsys CosmosScope, que se arranca desde la misma ventanita naranja, escribiendo: scope.

- Para empezar hemos cogido la librería de UMC_0.13 y el netlist verilog a nivel de puertas del b01_syn.v, para el cual no tenemos todavía fichero .vcd, pero lo probamos así para ver que funciona. El script (b01.tcl) seguido fue:

```
#####  
# link design  
#####
```

```
set search_path      "/mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador /cad/KITS/
UMC/UMC_13/VST/Tapeout_kits/UMCE13H210T3_1.2/design_compiler ."
set link_library     " * umce13h210t3_tc_120V_25C.db"

read_verilog        b01_syn.v
current_design       b01
link
```

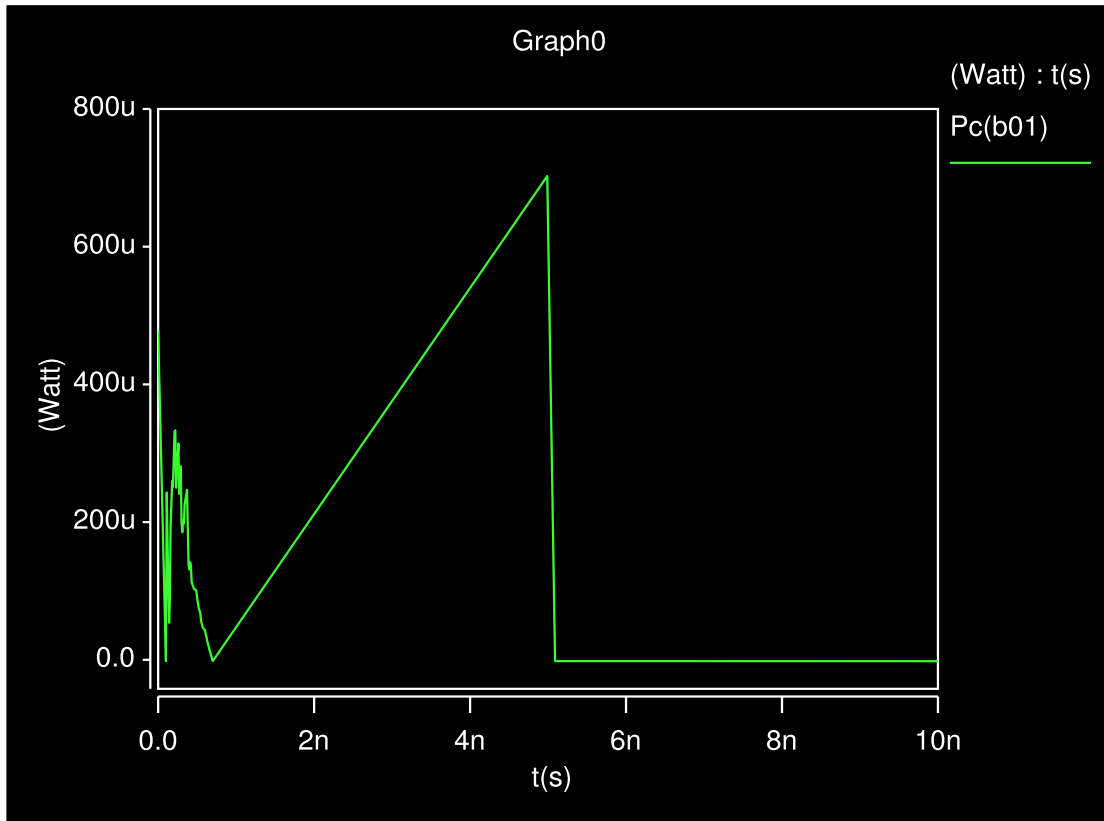
```
#####
#   set transition time / annotate parasitics
#####
set_input_transition 0.1 [all_inputs]
create_clock -period 10 -waveform {0 5} {clock}

#####
#   power analysis
#####
set_waveform_options -file b01_vcd
calculate_power      -waveform -statistics
report_power         -file b01_vcd

#quit
#####
```

En set search_path se pone primero la ruta donde va a buscar los ficheros verilog (el b01_syn.v) y después la ruta donde están las librerías, y en set link_library el nombre de la librería. Tuvimos que definir un reloj: create_clock -period 10 -waveform {0 5} {clock}, se generan los ficheros b01_vcd.fsdb de set_waveform_options -file b01_vcd, y b01_vcd.rpt de report_power-file

b01_vcd. Estos dos ficheros se generan bien sin problemas, aunque como no le dimos ningún fichero .vcd habrá asignado sus probabilidades, y en el .fsdb solo se ve un ciclo del reloj.



PrimePower (TM)

Version V-2004.06 for sparc64 -- May 21, 2004
Copyright (c) 2000-2004 by Synopsys, Inc.
ALL RIGHTS RESERVED

Report: average power
Design: b01
Library File(s): /cad/KITS/UMC/UMC_13/VST/Tapeout_kits/UMCE13H210T3_1.2/
design_compiler/umce13h210t3_tc_120V_25C.db
Sampling interval: 1 ns
Reporting threshold: 0 Watt
Date: Fri Apr 7 12:10:03 2006

operating_condition_max_name: <lib_default> Library: umce13h210t3_tc_120V_25C

Wire Loading Model Mode: enclosed

Cell	Design	Wire_model	Library	Selection_type
	b01	suggested_10K	umce13h210t3_tc_120V_25C	automatic-by-area

Library Power-specific unit information :

Time Unit : 1 ns
 Capacitance Unit : 1 pF
 Voltage Unit : 1 V
 Dynamic Power Unit : 0.001 W
 Leakage Power Unit : 1e-06 W

Definitions:

Total Power = Dynamic + Leakage
 Dynamic Power = Switching + Internal
 Switching Power = load capacitance charge or discharge power
 Internal Power = power dissipated within a cell
 X-tran Power = component of dynamic power-dissipated into x-transitions
 Glitch Power = component of dynamic power-dissipated into detectable
 glitches at the nets
 Leakage Power = reverse-biased junction leakage + subthreshold leakage

Level	Instance_Name (Cell_Name) #_of_States					
Total Power in Watt	Dynamic Power in Watt	Leakage Power in Watt	Switching Power in Watt	Internal Power in Watt	X-tran Power in Watt	Glitch Power in Watt
	(% of Tot)	(% of Tot)	(% of Dyn)	(% of Dyn)	(% of Dyn)	(% of Dyn)
Peak Power in Watt	Peak Time Interval in ns					

----- Instances -----

*----- 0	pp_root (.)	33814					
2.927e-05	2.915e-05	1.147e-07	8.465e-06	2.069e-05	0.000e+00	0.000e+00	
(99.61%)	(0.39%)	(29.04%)	(70.96%)	(0.00%)	(0.00%)		
7.028e-04	5-6						
-*----- 1	b01 (b01)	33814					
2.927e-05	2.915e-05	1.147e-07	8.465e-06	2.069e-05	0.000e+00	0.000e+00	
(99.61%)	(0.39%)	(29.04%)	(70.96%)	(0.00%)	(0.00%)		

7.028e-04 5-6

----- End of Instances -----

----- End of report -----

The following are primepower Run Data Record for GUI:

GUI_version: V-2004.06
GUI_report: power
GUI_current_design: b01
GUI_instance_count: 29
GUI_lib: /cad/KITS/UMC/UMC_13/VST/Tapeout_kits/UMCE13H210T3_1.2/design_compiler/
umce13h210t3_tc_120V_25C.db
GUI_input_ramp: 0
GUI_hier_sep: /
GUI_insts: b01
GUI_delta_time: -1e+08
GUI_waveform: 1
GUI_threshold: 0

time: 1144404607 Fri Apr 7 12:10:07 2006

Dentro del programa de síntesis de cadence PKS usamos el comando verilog -f para que haga la simulación con verilogXL, leyendo un fichero .opt donde llama a las librerías y a los ficheros de test_bench y netlist de puertas. El test_bench lo hemos modificado, le hemos introducido las líneas para que nos genere el fichero .vcd, de manera que nos queda:

b01_tb_vcd.v

```
'timescale 1ns / 1ps
```

```
module test;  
    wire outp, overflow;  
    reg clock, line1, line2, reset;
```

```
b01 top(line1, line2, reset, outp, overflow, clock);
```

```
//inicializacion de las entradas
```

```
initial  
begin  
    clock = 1'b0;  
    line1 = 1'b0;  
    line2 = 1'b0;  
    reset = 1'b1;
```

```

end

always
#10 clock = !clock;

initial begin
  $dumpfile ("b01.vcd");
  $dumpvars;
end

initial
#100 $finish;

//initial begin
//#15 reset = 0;
//#30 line1 = 1;
//#35 line2 = 1;
//end

//$toggle_count_report_flat("b01.tcf","test.top");

//initial
//  #500
//$finish;

endmodule

```

#####

Con esto se genera el fichero b01.vcd, que es de la forma:

```

$date
  Apr 11, 2006 12:12:59
$end
$version
  Tool:  VERILOG-XL  05.10.003-s
$end
$timescale
  1ps
$end

$scope module test $end
$var wire  1 !  outp $end
$var wire  1 "  overflow $end
$var reg   1 #  clock $end

```

```
$var reg 1 $ line1 $end
$var reg 1 % line2 $end
$var reg 1 & reset $end
```

```
$scope module top $end
$var wire 1 ' n_73 $end
$var wire 1 ( n_41 $end
$var wire 1 ) n_47 $end
$var wire 1 * n_34 $end
$var wire 1 + n_72 $end
$var wire 1 , n_33 $end
$var wire 1 - n_44 $end
$var wire 1 . n_71 $end
$var wire 1 / n_32 $end
...
```

Lo intentamos pasar a .tcf, con el comando: `lpsvcd2tcf.exe test /mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/b01.vcd -toggle tcf_flat -output_file b01.tcf_flat`, y se genera el fichero, pero pasa como antes, que no es como debería ser, faltan las comillas, puntos y comas, ... Entonces deberemos de hacer en PKS es hacerlo todo con el tcf, y una vez se haya optimizado obtener el .vcd, que será el que llevemos a primepower, donde corremos el script b01.tcl, con el comando para leer el vcd:

```
read_vcd -strip_path b01_tb_vcd/top /mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/b01.vcd
```

Pero dan dos errores y un montón de warnings:

```
Warning: The net "line1" is not covered by VCD file "/mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/b01.vcd". (SIM-220)
Warning: The net "line2" is not covered by VCD file "/mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/b01.vcd". (SIM-220)
Warning: The net "reset" is not covered by VCD file "/mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/b01.vcd". (SIM-220)
Warning: The net "outp" is not covered by VCD file "/mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/b01.vcd". (SIM-220)
Warning: The net "overflow" is not covered by VCD file "/mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/b01.vcd". (SIM-220)
Warning: The net "clock" is not covered by VCD file ...
```

```
Error: (read_vcd) Strip path "b01_tb_vcd/top" can not be matched in the VCD file. (SIM-218)
Error: Power is not calculated. Please run calculate_power before report_power. (RPT-201)
```

El problema es que teníamos que haber puesto el comando para leer el vcd de la forma:

```
read_vcd -strip_path test/top /mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/b01.vcd
```

Lo que tenemos que poner es lo que venga en el test_bench:

```
module test;  
    wire outp, overflow;  
    reg clock, line1, line2, reset;  
  
b01 top(line1, line2, reset, outp, overflow, clock);
```

Así ahora se generan el b01_vcd.rpt y el b01_vcd.fsdb.

PrimePower (TM)

Version V-2004.06 for sparc64 -- May 21, 2004
Copyright (c) 2000-2004 by Synopsys, Inc.
ALL RIGHTS RESERVED

Report: average power
Design: b01
Library File(s): /cad/KITS/UMC/UMC_13/VST/Tapeout_kits/UMCE13H210T3_1.2/
design_compiler/umce13h210t3_tc_120V_25C.d
b
Simulation period[1]: 0 ns - 90 ns
Sampling interval: 1 ns
Reporting threshold: 0 Watt
Date: Tue Apr 11 13:11:47 2006

operating_condition_max_name: <lib_default> Library: umce13h210t3_tc_120V_25C

Wire Loading Model Mode: enclosed

Cell	Design	Wire_model	Library	Selection_type
	b01	suggested_10K	umce13h210t3_tc_120V_25C	automatic-by-area

Library Power-specific unit information :

- Time Unit : 1 ns
- Capacitance Unit : 1 pF
- Voltage Unit : 1 V
- Dynamic Power Unit : 0.001 W
- Leakage Power Unit : 1e-06 W

Definitions:

- Total Power = Dynamic + Leakage
- Dynamic Power = Switching + Internal
- Switching Power = load capacitance charge or discharge power
- Internal Power = power dissipated within a cell
- X-tran Power = component of dynamic power-dissipated into x-transitions
- Glitch Power = component of dynamic power-dissipated into detectable glitches at the nets
- Leakage Power = reverse-biased junction leakage + subthreshold leakage

```

-----
Level      Instance_Name (Cell_Name) #_of_States
-----
Total      Dynamic  Leakage  Switching Internal  X-tran  Glitch
Power      Power     Power    Power    Power    Power    Power
in Watt   in Watt   in Watt  in Watt  in Watt  in Watt  in Watt
          (% of Tot) (% of Tot) (% of Dyn) (% of Dyn) (% of Dyn) (% of Dyn)
-----
          Peak      Peak
          Power     Time Interval
          in Watt    in ns
-----

```

```

----- Instances -----
*----- 0 pp_root (.) 45
5.632e-06 5.515e-06 1.171e-07 0.000e+00 5.515e-06 0.000e+00 0.000e+00
          ( 97.92%) (  2.08%) (  0.00%) (100.00%) (  0.00%) (  0.00%)
          6.411e-05  20-21

-*----- 1 b01 (b01) 45
5.632e-06 5.515e-06 1.171e-07 0.000e+00 5.515e-06 0.000e+00 0.000e+00
          ( 97.92%) (  2.08%) (  0.00%) (100.00%) (  0.00%) (  0.00%)
          6.411e-05  20-21
----- End of Instances -----

```

----- End of report -----

The following are primepower Run Data Record for GUI:

```

GUI_version: V-2004.06
GUI_report: power
GUI_current_design: b01
GUI_activity: /mnt/cnm2/casram/cadence_03/spr_5.0_usr4/ams_3.60/sumador/b01.vcd
GUI_instance_count: 29

```

GUI_lib: /cad/KITS/UMC/UMC_13/VST/Tapeout_kits/UMCE13H210T3_1.2/design_compiler/
umce13h210t3_tc_120V_25C.db

GUI_input_ramp: 0

GUI_hier_sep: /

GUI_insts: b01

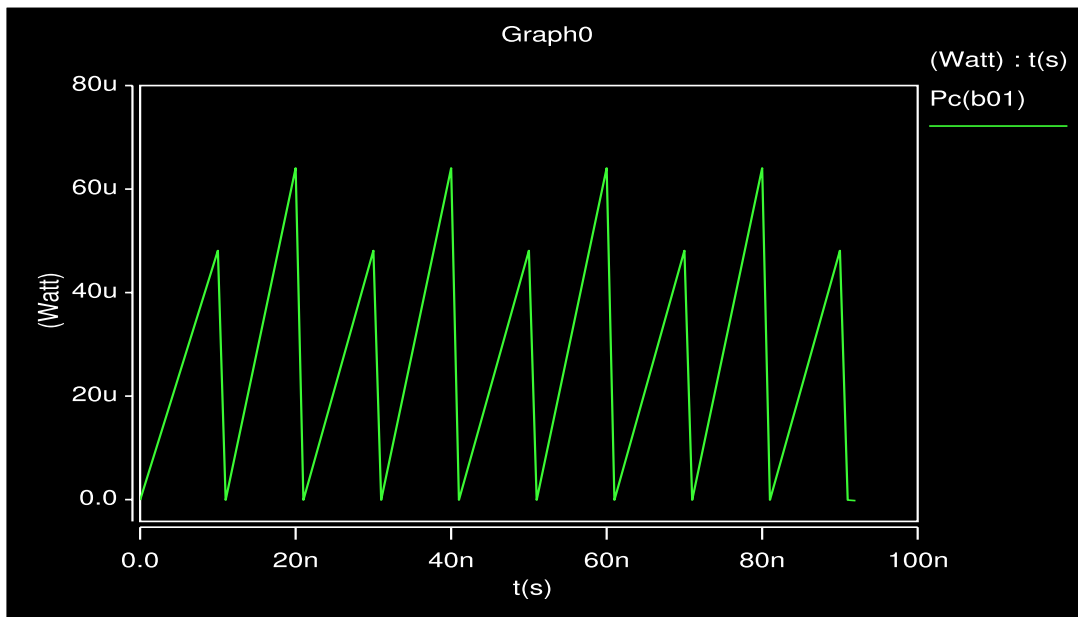
GUI_time: 0 90

GUI_delta_time: 10

GUI_waveform: 1

GUI_threshold: 0

time: 1144753909 Tue Apr 11 13:11:49 2006



Lo que debemos hacer es:

- Partimos del código HDL, el cual sintetizamos en PKS (con las restricciones temporales ...)
- Una vez sintetizado, realizamos una simulación verilog que nos generará un fichero de actividad .tcf, que leeremos a continuación
- Una vez leído este fichero se vuelve a sintetizar con las optimizaciones pertinentes, con lo que tenemos un netlist verilog a nivel de puertas ya optimizado
- A este netlist le realizamos una simulación verilog para obtener el fichero de actividad .vcd
- Nos vamos a PrimePower para poder visualizar la potencia en el tiempo, con sus picos.

Para tener los valores generamos un fichero .out, con el comando: `set_waveform_options -file b01_vcd -format out`. Esto nos genera un fichero llamado `b01_vcd.out` que tiene los valores representados en la gráfica, y tiene la forma:

```
;! output_format 5.3  
; header 116145095650
```

;
;
; PrimePower (TM)
;
; Version V-2004.06 for sparc64 -- May 21, 2004
; Copyright (c) 2000-2004 by Synopsys, Inc.
; ALL RIGHTS RESERVED
;

;
.time_resolution 1
.hier_separator /
.index Pc(pp_root) 1 Pc
.index Pc(b01) 2 Pc
0
2 1.172e-07
1 1.172e-07
10
2 4.820e-05
1 4.820e-05
11
2 1.169e-07
1 1.169e-07
20
2 6.411e-05
1 6.411e-05
21
2 1.172e-07
1 1.172e-07
30
2 4.820e-05
1 4.820e-05