Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Vinicius Bonfim Araujo

Projeto e Simulação do Unified Power Format a partir de um IP-Core de Baixo Consumo

Campina Grande
Outubro de 2021

Vinicius Bonfim Araujo

Projeto e Simulação do Unified Power Format a partir de um IP-Core de Baixo Consumo

Trabalho de Conclusão de Curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Microeletrônica

Orientador:
Marcos Ricardo de Alcântara Morais, D. Sc.

Campina Grande
Outubro de 2021

Vinicius Bonfim Araujo

Projeto e Simulação do Unified Power Format a partir de um IP-Core de Baixo Consumo

Trabalho de Conclusão de Curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Microeletrônica

Aprovado em / /

Marcos Ricardo de Alcântara Morais, D. Sc.
UFCG

Prof. Gutemberg Gonçalves dos Santos Júnior, D. Sc.
UFCG

Campina Grande

Outubro de 2021

Dedico esse trabalho aos meus pais, Roseli Rocha Bonfim e João Francisco de Araújo, que, com toda a ajuda e suporte, me permitiram chegar aonde cheguei.

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus, pois sem Ele nada seria possível.

Em segundo, aos meus pais, Roseli Rocha Bonfim e João Francisco de Araújo, que nunca mediram esforços para me ajudar e apoiar, tornando assim possível que todas as minhas formações se concretizassem, serei eternamente grato a vocês!

Aos meus amigos e colegas da turma do 15.1, que de alguma forma me ajudaram, seja por meio dúvidas sanadas, estudos em grupo realizados ou conselhos dados. Agradeço principalmente àqueles que, por ao menos um dia me acolheram em suas casas quando precisei dormir em Campina por algum motivo, meu muito obrigado. Aqui vai uma uma menção em especial a Cesinha e Saulo que me acolheram, inclusive durante a produção desse trabalho.

Ao laboratório XMEN e em especial a Kaline, que sempre me motivou e ajudou desde que entrei no laboratório, a galera do café, que me proporcionaram bons momentos entre o trabalho e o estudo, a George pelo apoio na produção desse trabalho e suporte oferecido no que compete ao laboratório e fora dele. Agradeço também a Agripino pela oportunidade a mim concedida de poder fazer parte de um dos melhores laboratórios da UFCG.

Ao meu orientador, Marcos Morais, pelos conselhos durante o período de estágio e TCC, e pela capacitação provida a mim na área de microeletrônica. Agradeço também ao professor Gutemberg Júnior pela introdução ao mundo da eletrônica digital e a oportunidade de trabalhar no XMEN, e aos demais professores que contribuíram para a minha formação.

"Our greatest weakness lies in giving up. The most certain way to succeed is always to try just one more time."

Thomas A. Edison

RESUMO

Com o passar dos anos, menores ficaram os chips e assim, melhores desempenhos eles passaram a ter, mas não só isso, como também alguns problemas. O gasto energético se tornou problemático por diversos motivos, sendo assim necessário tornar os chips mais eficientes. A gerência e a eficiência energética puderam ser aprimoradas dentro do design com o formato de intenção de redução de energia (do inglês, *power intent format*). Este trabalho, irá abordar os motivos que levaram a necessidade de técnicas de diminuição de gasto energético, como elas se tornaram possível através do formato *power intent*, e como se dá a aplicação desse formato no fluxo de desenvolvimento.

Palavras-chave: *Power Intent*, UPF, Baixo Consumo, Desenvolvimento de Hardware.

ABSTRACT

Over the years, the chips became smaller and thus, they started to have better performances, but not only that, they also started to have some problems. Energy expenditure has become problematic for several reasons, and it was therefore necessary to make chips more efficient. Energy management and efficiency could be improved within the design with the power intent format, this one provides methods for energy reduction. The present work will address the reasons that led to the need for techniques to reduce energy expenditure, how they became possible through the power intent format, and how this format is applied in the development flow.

Keywords: Power Intent, UPF, Low Power, Hardware Development.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

CPF	*Common Power Intent*
DEE	Departamento de Engenharia Elétrica
DUT	*Design Under Test*
EDA	*Electronic Design Automation*
HDL	*Hardware Description Language*
IEEE	*Institute of Electrical and Electronics Engineers*
IP	*Intellectual Property*
LEC	*Logic Equivalence Chechker*
LPC	*Low Power Coalition*
RTL	*Register Transfer Level*
SI2	*Silicon Integration Initiative*
STA	*Static Timing Analysis*
SoC	*Systems-on-a-Chip*
UPF	*Unified Power Intent*
UFCG	Universidade Federal de Campina Grande

SUMÁRIO

1 INTRODUÇÃO

Impulsionados pela “lei” de Moore, a indústria busca desde 1965 a evolução dos circuitos integrados, dobrando a quantidade de transistores por área a cada dois anos aproximadamente (MOLLICK, 2006). O fato dos transistores serem fabricados cada vez menores, proporcionaram a diminuição no tamanho das litografias com o passar do tempo.

Com uma menor área, vários parâmetros são melhorados, como a velocidade de comunicação entre os componentes, quantidade de transistores e frequências de operação, tornando maior a velocidade de processamento do bloco digital, isso era a base da escala de Dennard e permaneceu válido até 2004, entretanto a diminuição das litografias continuaram.

Em tecnologias de 90 nm e acima disso, não se via a potência de fuga (do inglês, *leakage power*) como um grande problema, pois seu gasto energético era bem menor comparado ao gasto por comutação dos transistores. Contudo para geometrias menores, a potência de fuga se tornou gasto preocupante e dominante, visto que o gasto por comutação poderia ser controlado, mas o de fuga não.

O gerenciamento de energia nessas menores geometrias se tornou essencial no processo de design. Técnicas de redução de energia podem ser usadas para minimizar os dois tipos de consumo de energia. Contudo, não se havia formas de implementar as estratégias tão cedo e eficazmente dentro dos fluxos de desenvolvimento, e com isso surgiu a necessidade do *power intent*.

As intenções de potência (*power intent*), vieram para suprir essas necessidades da indústria e possibilitar que desde às primeiras etapas do fluxo de desenvolvimento de hardware seja possível implementar técnicas de redução de energia. O *power intent* atua de forma separada da descrição de hardware, sendo assim possível que a descrição de hardware não sofra interferência alguma da existência ou não, dessa descrição de intenção de potência.

Em meados de 2006 diversas organizações e companhias se organizaram e deram origem a dois formatos de intenção de potência, o *Unified Power Format* (UPF) e o *Common Power Format* (CPF). Doado ao IEEE, o UPF se tornou, em 2009, oficialmente o formato padrão de intenção de potência e por esse motivo, se tornou o objetivo de estudo desse trabalho.

1.1 OBJETIVOS

1.1.1 OBJETIVO GERAL

Este trabalho tem como objetivo apresentar os conceitos relacionados a descrição de *power intent* em UPF, assim como os motivos que levam a sua necessidade.

Com relação a sua aplicabilidade, é descrito sintaxes do formato, observações em seu uso, assim como, exemplos para facilitar o entendimento.

1.1.2 OBJETIVOS ESPECÍFICOS

O trabalho tem como objetivos específicos para seu desenvolvimento:

- Revisão bibliográfica acerca do formato de *power intent* em UPF;
- Simulação do código RLT de um IP core em companhia da descrição de *power intent* em UPF.

2 FLUXO DE DESENVOLVIMENTO DE HARDWARE

Na construção de uma Propriedade Intelectual (IP), seja ela um IP digital ou um *Systems-on-a-Chip design* (SoC), um fluxo de desenvolvimento é seguido, passando por diversas fases e englobando diversas equipes durante o seu processo. Na área de desenvolvimento de hardware não há um consenso porque o fluxo de desenvolvimento em si é uma propriedade intelectual, porém é conhecido no que toca as principais atividades, como: especificação de hardware, geração de *Register Transfer Level* (RTL), verificação funcional do código RTL, síntese e projeção física do IP digital, produção física do projeto e validação, que o fluxo assim se segue cronologicamente como ordenado na citação das atividades.

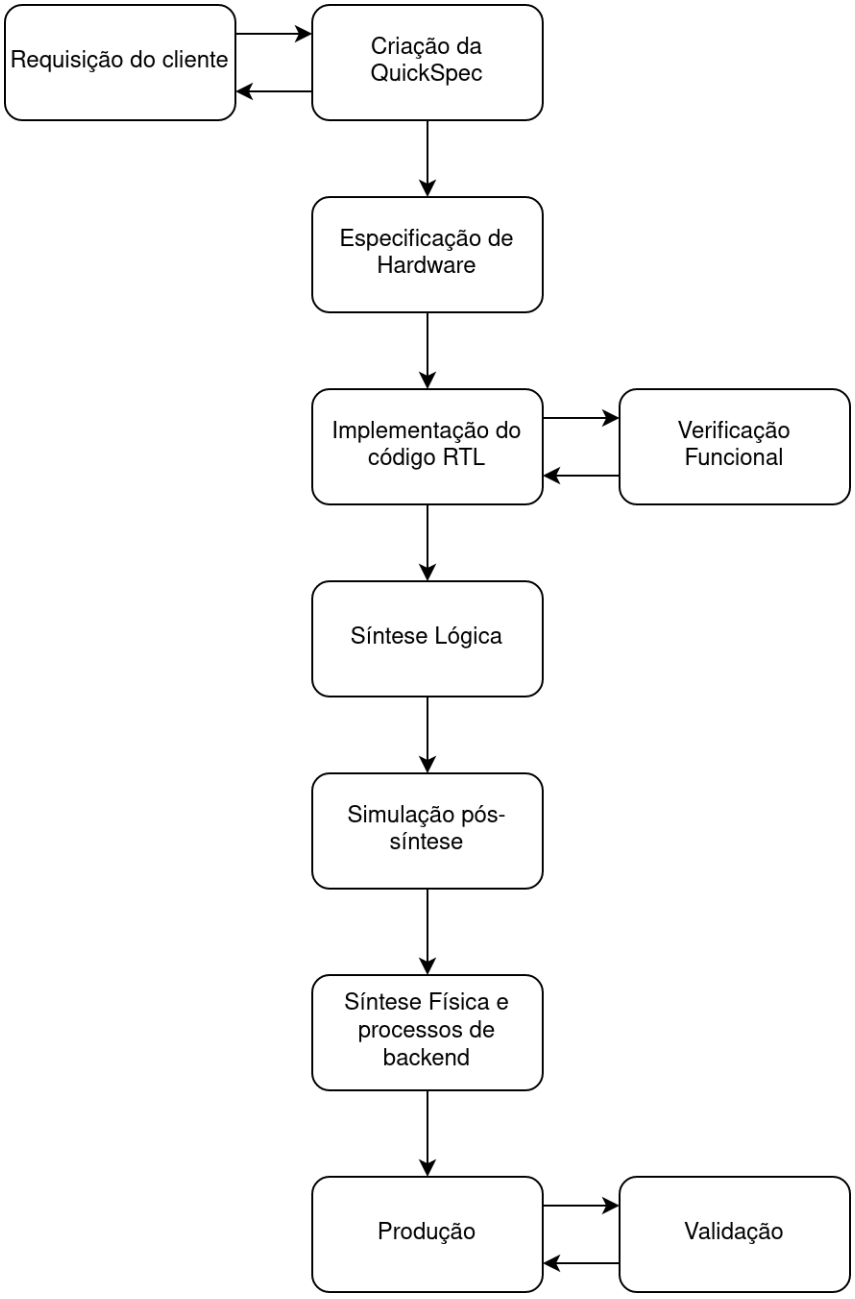
De modo a ilustrar um fluxo completo nesse estudo, aqui será detalhado um fluxo de desenvolvimento que é utilizado dentro de algumas companhias que atuam na área. Esse fluxo conta com a participação das equipes responsáveis pelo desenvolvimento do projeto, assim como, com a colaboração do cliente, que irá propor uma problema e junto com a equipe responsável pelo mesmo, definir as funcionalidades que serão desenvolvidas no IP para a resolução do problema. Na Figura 1 é possível ver o fluxo completo aqui apresentado.

A empresa irá propor uma especificação rápida ao cliente, com algumas funcionalidades acerca do projeto. Esta se molda na cooperação entre os dois até se tornar uma especificação de hardware sólida, que então regirá todo o fluxo de desenvolvimento do IP.

A especificação de hardware define exatamente cada funcionalidade que o hardware, deve executar ao final de sua implementação e tem todas as suas funcionalidades documentadas. Ela deve também apresentar alto nível de abstração e não pode ser definida com base na implementação arquitetural a ser adotada, nem sobre os componentes de software ou hardware a serem utilizados. Além de detalhes em alto nível, conta também com informações de baixo nível, como especificação e descrição de pinos a serem utilizados.

Uma vez consolidada a especificação, é desenvolvida uma descrição de hardware a nível de sinais, respeitando todos os pontos impostos anteriormente na especificação. A implementação da descrição de hardware implica na geração de um código RTL que por meio de processo automatizado se tornará posteriormente hardware propriamente dito. Com o RTL gerado, já é possível realizar a simulação da descrição de hardware feita e portanto, a realização da verificação e testabilidade das funcionalidades implementadas

Figura 1 – Fluxo de desenvolvimento de um IP em hardware



Fonte – Própria

em baixo nível. Um conceito amplamente conhecido é que, quanto antes um erro for detectado, menos recursos e erros são propagados durante a execução do projeto, por isso, é importante que a verificação seja realizada já na fase de RTL.

A verificação funcional é um dos três tipos de verificação de hardware, esta procura assegurar que as funcionalidades postas na especificação do hardware sejam condizentes com as funcionalidades implementadas no projeto lógico em RTL. A garantia de funcionalidade é dada a partir da comparação entre as respostas do *Design Under Test* (DUT) e do modelo ideal - construído em uma linguagem de abstração maior para ter as respos-

tas de acordo com as especificações - a partir de estímulos capazes de excitar todas as funcionalidades especificadas.

Todas as funcionalidades também estão especificadas no plano de cobertura e a verificação apenas estará finalizada quando todos os critérios de cobertura forem atingidos. É válido ressaltar que durante o processo de verificação o fluxo de desenvolvimento pode regredir a fase de RTL caso seja verificado que funcionalidades não respondam da forma que deveria, sendo assim necessário a correção do RTL e portanto, uma nova verificação do mesmo, até que os critérios de cobertura sejam satisfeitos (SILVA, 2007).

Completamente verificado o IP, com todos os critérios cumpridos, é então iniciada a fase de síntese. A síntese é realizada a partir do código RTL verificado e dá origem a projeção dos componentes eletrônicos e suas conexões, chamado também de *netlist*. Durante esse processo também é realizado a análise temporal por STA, que nada mais é que um método de validação do desempenho de temporização desses resultados obtidos. Caso haja violação de *timing*, a ferramenta de simulação irá gerar um novo *netlist*. Este processo é cíclico até que se obtenha um *netlist* com *timing* o suficiente para não ocasionar erros na funcionalidade final.

Obtido o *netlist*, é então realizada a simulação a nível de portas lógicas (*gate-level*), de modo a verificar se os mesmo realizam as funcionalidades que deveriam. Esse processo é também chamado de pós-síntese e conta também com a execução de LEC (*Logic Equivalence Chechker*) e análise de consumo de energia (*power analysis*, em inglês). Perpassado por todas essas etapas, um *netlist* final é obtido e então enviado para a síntese física realizado pela equipe de *backend* do projeto.

Após a criação de um *netlist* que passou pela verificação funcional e que atingiu, ao menos preliminarmente, aspectos não funcionais de desempenho temporal, área e consumo energético, passa-se para a fase de síntese física. Esta etapa objetiva obter um desenho finalizado do chip, para que possa ser enviado para fabricação. Durante a síntese física verifica-se que alguns parâmetros não podem ser obtidos, necessitando, às vezes, de voltar para as etapas anteriores.

Em posse do chip pronto e em mãos, é então realizado testes das funcionalidades do mesmo, que busca comprovar a ausência de erros de funcionalidades segundo as especificadas no documento de especificação de hardware, assim como verificar se não houve nenhum erro no processo de produção do mesmo. Validado o chip após a execução dos testes, é dado então o aval para a sua produção em massa.

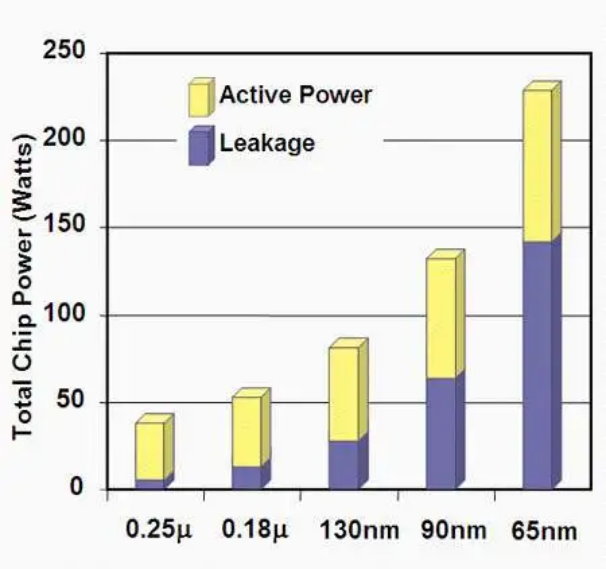
3 *UNIFIED POWER INTENT FORMAT*

3.1 MOTIVAÇÃO

O desenvolvimento de circuitos integrados digitais é constantemente desafiado pelo alto consumo de energia. Os altos níveis de velocidade de *clocks* combinados com a integração de maiores funcionalidades nos circuitos e a diminuição das geometrias no processo de produção dos mesmos, contribuíram significantemente para o crescimento deste gasto energético.

Normalmente, a diminuição do dimensionamento da área do circuito influência na densidade de transistores e contribui para reduzir os tempos de propagação de sinais entre portas lógicas, que por consequência, aumenta a sua frequência de operação. Porém, na prática, em tecnologias com geometrias abaixo de 90 nm, esses benefícios vem com um aumento exponencial da potência estática, também conhecida como *leakage power*. Esse aumento pode ser visualizado na Figura 2, e se dá pelo fato da tensão limiar de funcionamento do transistor ter sido atingida e portanto, não poder mais ser redimensionada.

Figura 2 – Tendências de consumo energético no avanço das tecnologias



Fonte – (JONES, 2004)

A potência estática dissipada por um transistor está relacionada com a corrente de fuga, que por sua vez depende da tensão limiar de operação do transistor.

$$P_{\text{Total}} = P_{\text{Chaveamento}} + P_{\text{Curto-Circuito}} + P_{\text{Leakage}} \tag{3.1}$$

O dispositivo tem a sua potência total dada pela potência estática e a potência dinâmica. Como antes dito, a potência estática é proveniente da corrente de fuga e tensão operação do transistor, já a potência dinâmica é dada por dois fatores: a energia que passa pelo circuito CMOS do transistor e é gasta para a comutação dos estados lógicos de carga e descarga do nó de saída, e a potência dissipada por curto-circuito durante o chaveamento do transistor.

Na Figura 2 é possível visualizar a proporção de gasto energético entre os tipos de potência por tecnologia empregada.

$$P_{\text{Chaveamento}} = \alpha f C {V_{\text{dd}}}^2 \tag{3.2}$$

$$P_{\text{Curto-Circuito}} = I_{\text{cc}} V_{\text{dd}} f \tag{3.3}$$

$$P_{\text{Leakage}} = f(V_{\text{dd}}, V_{\text{th}}, W/L) \tag{3.4}$$

Onde α será um fator de atividade do chaveamento, f a frequência de chaveamento, C a capacitância efetiva, V_{dd} a tensão de alimentação, I_{cc} corrente de curto-circuito, V_{th} tensão limiar do transistor, W/L dimensões do transistor.

É notório que a redução da tensão de alimentação V_{dd} irá diminuir todos os tipos de dissipação de potência existentes no processo. Com isso, técnicas de redução de dissipação relacionadas a tensão foram desenvolvidas. As principais técnicas empregadas hoje para redução de dissipação de energia, são:

- *Power Gating*
- *Power Gating* com retenção de dado/estado
- *Multi-voltage Design*
- Escalonamento dinâmico de tensão (e frequência)
- Escalonamento adaptativo de tensão (e frequência)

As técnicas empregadas possuem nomes autoexplicativos que, de fato, realizam operações e objetivos as quais sugerem. Mesmo embora antes de 2007, já existissem essas técnicas, as mesmas só puderam ser viáveis para adoção e verificação no início do fluxo após a criação dos *power formats*, a exemplo do *Common Power Format* (CPF) que surgiu no citado ano, e que posteriormente contribuiu para o projeto IEEE-1801, padrão de *power format* certificado pelo comitê da associação IEEE.

3.2 IMPORTÂNCIA DO UPF

Devido a todos os tipos de consumo de energia relacionados ao semicondutores serem funções da tensão de alimentação V_{dd}, todas as técnicas de redução do consumo de energia se dão pelo controle do mesmo.

Como antes dito, é conhecido que em alto nível de abstração, como por exemplo, das camadas de execução desde o início do fluxo até antes da síntese, não é possível realizar a manipulação tensões de alimentação e estabelecer a conexão dos mesmos com o código em RTL, ou até mesmo, a nível de portas lógicas. Mesmo sabendo como agir para reduzir o consumo de energia, isso não foi suficiente para ser posto em prática o plano de intenção de redução de energia, devido a ausência de uma padrão que permitisse a realização dessa conexão entre altos níveis de abstração e as intenções de manipulações de grandeza física.

As descrições de hardware são consideradas referências de ouro do design, e assim, como não é comum a aplicação de tensões de alimentação a esse nível de abstração, certamente é impossível a implementação de que certas instâncias de hierarquia do projeto trabalhe dentro do RTL trabalhe em condições específicas de tensão, não ao menos até que o projeto alcance o estágio de *floorplanning* (processo realizado durante a etapa de física de*backend*). A indústria estava enfrentando a ausência de uma metodologia que ajudasse a definir a conectividade do fornecimento de energia, áreas com diversos níveis de tensão de funcionamento e distribuição de energia dentro do design sem interferir na referência de ouro do HDL(KHONDKAR, 2018).

A criação do *Unified Power Format* (UPF) veio para suprir essas necessidades da indústria e possibilitar que, desde às primeiras etapas do fluxo de desenvolvimento de hardware seja possível implementar técnicas de redução de energia. O UPF atua de forma separada a descrição de hardware, sendo assim possível que a referência de ouro (RTL) não sofra interferência da existência do UPF. Durante a simulação do código RTL e o UPF podem se unir (a escolha do designer) por meio do *testbench* criado, e são executados como um projeto só.

3.3 *POWER INTENT*

No início, quando a indústria de automação de designs eletrônicos (EDA) começou a criar padrões para especificar, simular e implementar funcionalidades especificar em circuitos eletrônicos digitais, apenas existiam especificações relacionadas a nível de transistores. Considerações sobre consumo eram simples e fáceis de serem assumidas, visto que consumos de potência não era uma principal preocupação na época e os chips operavam em uma única tensão para todas as funcionalidades.

A presença das linguagens de descrição de hardware (HDLs) como, VHDL e o SystemVerilog no meio EDA, trouxeram um conjunto de recursos necessários para capturar as especificações funcionais de sistemas eletrônicos, porém não tinham suportes relacionados ao controle da arquitetura de distribuição de energia (como cada elemento do sistema seria alimentado) (IEEE..., 2019).

O projeto IEEE 1801, denominado como *Unified Power Format*, possibilita a criação de design em *low power* (baixo custo de utilização de energia) e a verificação do consumo ciente de energia (do inglês, *power aware*) em diferentes etapas do fluxo de desenvolvimento, desde o início antes da criação do código RTL até o projeto físico e sua implementação.

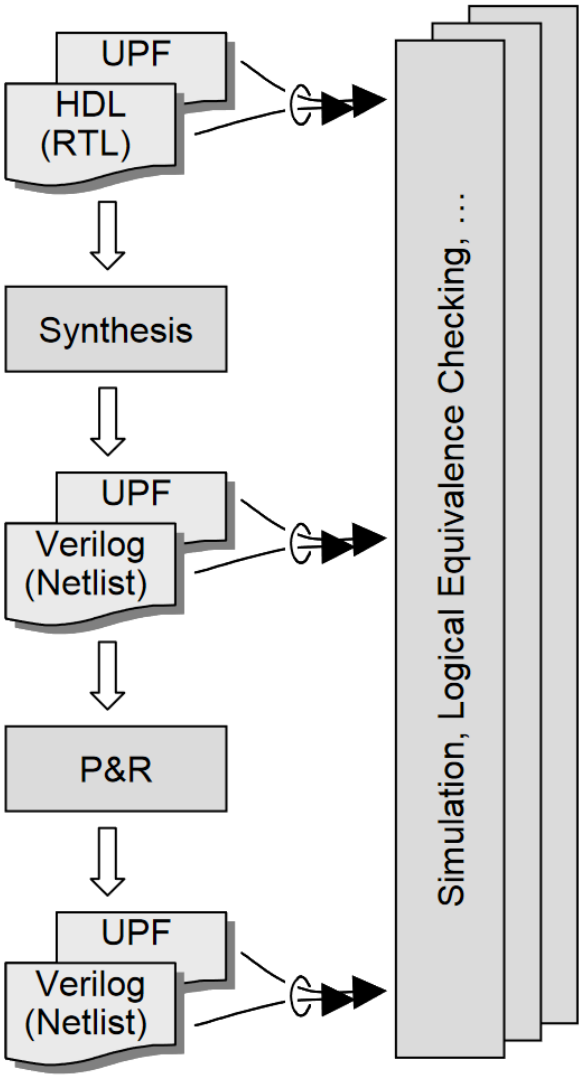
O formato provê a capacidade de sistemas eletrônicos tenham previamente em seu desenvolvimento a projeção de energia como parte chave do processo. A Figura 3 mostra o suporte que o formato provê nas etapas do fluxo de desenvolvimento. UPF viabiliza um formato sólido, que especifica informações de design de *power aware* que não podem ser especificados no código HDL, ou que não são desejáveis serem incorporados diretamente dentro do HDL, pois atrelaria uma especificação lógica diretamente a uma implementação de energia restrita. O UPF também define semântica consistente em toda a verificação e implementação, ou seja, o que é implementado é o mesmo que foi verificado (UNIFIED..., 2007).

O UPF visa aplicar o conceito de IP, que busca o reaproveitamento de uma intenção de redução de energia (*power intent*, em inglês) em demais códigos RTL que forem ser utilizados. A utilização é dada em relação as especificações que dão origem ao RTL, contudo ela precisa ser feita se m a intervenção direta e por meio de uma abordagem metodológica para modelar de forma abstrata as conexões de energia, as tensões de alimentação, as áreas que serão alimentadas com determinadas tensões e os estados de energia correspondentes.

As metodologias são baseadas na especificação ou no *power intent* do código RTL e permitem que os designers constantemente realizem refinamentos nas distribuições de energia, nos *power domains*, nas estratégias de isolamento e retenções, e assim em diante, ao longo de todo o fluxo de implementação do projeto, especificamente nas fase de pós-síntese e pós *place and route* (fase realizada na etapa de *backend*). Por consequência, as ferramentas automatizadas de desenvolvimento em *power aware* utilizadas para verificação e implementação são construídas para semânticas e linguagens do UPF (KHONDKAR, 2018).

O UPF é utilizado para particionar o design em domínios de potência, onde cada domínio é energizado por uma rede de distribuição (*supply network*) que obtém energia das portas de alimentação (*supply ports*) e podem ter sua alimentação trocada devidos certas condições pelas chaves de alimentação (*power switches*). Diversas estratégias podem ser

Figura 3 – Contribuição do UPF no fluxo de desenvolvimento



Fonte – (UNIFIED..., 2007)

aplicadas aos *power domains* de forma a controlar os sinais lógicos presentes nos mesmos, quando esses domínios são desligados ou ligados.

Estratégias de isolamento (ISO) buscam evitar a propagação de saídas de sinais indefinidos para dos demais componentes, ao qual seu domínio está conectado, quando este tem sua alimentação cortada. Eles também definem os valores que estas saídas controladas deverão permanecer.

Entre os principais tipos de estratégias, está a retenção, que especifica quais objetos do domínio precisam ter seu valor lógico retido, para quando o domínio seja desligado e após religado ter seu status restaurado. Relacionados a níveis de tensão, estão os deslocadores de nível que tem como objetivo transformar sinais que estão em um determinado nível de tensão no mesmo sinal em outro nível de tensão.

3.4 DIFERENÇAS ENTRE UPF E CPF

No ano de 2007, surgiram não só um formato capaz de realizar essas intenções, mas dois formatos que até os dias atuais possuem suportes ferramentais para serem praticados. As primeiras referências ao *Common Power Format* surgiram, um pouco antes, em 2006 quando a Cadence Design Systems anunciou a iniciativa *Power Forward Initiative*. Mais tarde naquele mesmo ano, a Cadence e a Silicon Integration Initiative (SI2) criaram o *Low Power Coalition* (LPC) sob auspícios de desenvolvimento do formato CPF (ALLEN, 2008).

Como esperado da aliança entre a Cadence e a SI2, no início de 2007, foi lançado o documento oficial contendo a documentação relacionado ao formato CPF. Segundo o documento da LPC, o formato tinha o suporte ferramental da Cadence.

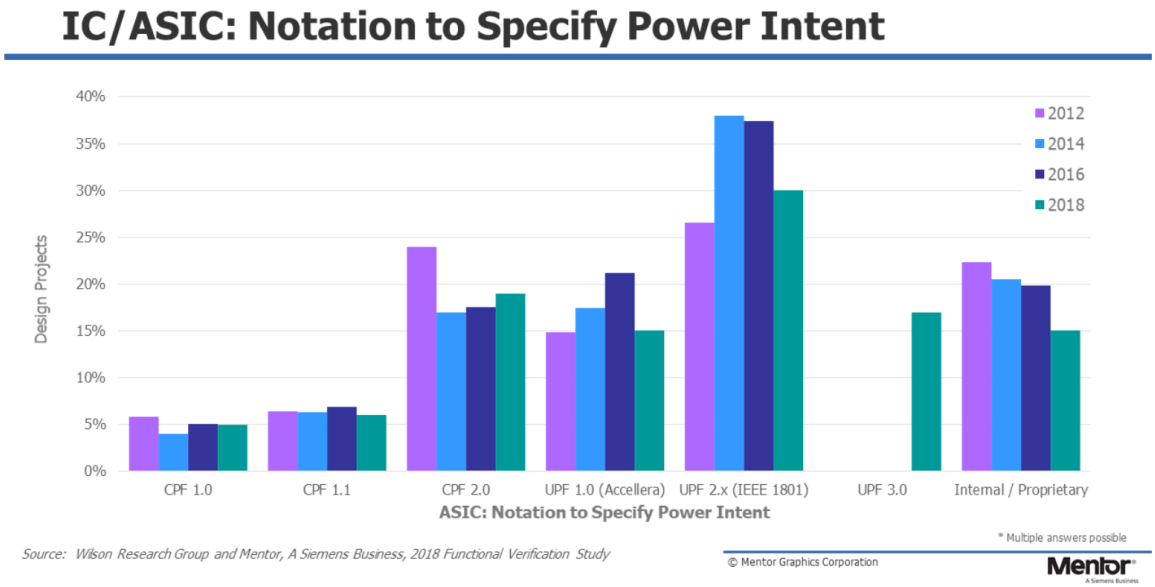
Também no ano de 2006, foi criada a iniciativa de forma a possibilitar a criação do UPF. As empresas Texas Instruments, Nokia, Mentor Graphics e Magma Design Automation mostraram grande cooperação de forma a provê a área de EDA um formato para descrever intenção do projeto de baixo consumo de energia. Com o apoio deles, o UPF se tornou o padrão EDA mais rápido produzido pela Accellera, na época, com o maior número de doações de tecnologia convergindo em um único padrão aberto (HUYGEN; ETTEN, 2007).

Desde a criação dos formatos, ambos foram utilizados, embora no início da utilização dos *power intents* houvesse uma predisposição da indústria de utilizar mais o UPF, visto que esse tinha suporte ferramental das companhias Magma, Mentor, Synopsys e Cadence. Desses três fornecedores de ferramentas, apenas a Cadence disponibilizava suporte para o CPF.

O reflexo desse suporte ferramental pode ser visto na Figura 4. Não só o fato de ter mais suportes ferramentais contribuíram para isso, mas também o fato o formato UPF ter sido adotado pela organização IEEE e ter recebido atualizações em seu documento mais rapidamente desde a criação de ambos os formatos. É possível também perceber na Figura 4, que não só o UPF teve maior emprego em todos os anos e em todas as suas versões, como especificações de *power intent* próprias das instituições que a utilizavam teve utilização próximas as do CPF.

Esses *power formats*, seja o UPF ou o CPF, são reconhecidos em toda a indústria como formatos capazes de realizar considerações relativos as especificações de controle de potência em um projeto. Células de *low power* especificados nos formatos, são inseridos de forma separada no *netlist*. Dessa forma, por meio dos compiladores de potência, é lida a intenção de potência e então são inseridas as células de baixa potência no *netlist*. Nos softwares utilizados no desenvolvimento a exemplo da Cadence, que possui o SimVision, é possível verificar se os domínios de potência estão seguindo a sequência de inicialização

Figura 4 – Tipos de *Power Intent* e uso ao longo dos anos



Fonte – (FOSTER, 2019)

e desligamento corretamente, assim como verificar se as estratégias utilizadas dentro do *power format* estão funcionando como deveriam.

Os formatos CPF e UPF apresentam as mesmas capacidades de abordar as intenções de potência necessárias as técnicas de *low power*, mas apresentando sintaxes diferentes para abordar a mesma intenção. De forma particular a cada formato, eles apresentam capacidade de abordar: áreas do design que trabalhem com diferentes níveis de tensão, por meio do *level shifters*; Domínios de potência, que possuem conceitos diferentes para cada um dos formatos; estratégias de isolação; estratégias de retenção; e *power switches*, capazes de chavear o fornecimento de energia para determinado domínio (ALLEN, 2008).

É válido lembrar que algumas opções dentro dos comandos utilizados para o *power intent* podem acabar coincidindo entre os formatos, mas no geral, as opções vão apresentar notações diferentes, mesmo embora tenham a mesma funcionalidade. Algumas intenções de *power* que em um formato pode ser representado por um só comando, no outro pode ser que se utilize de dois. Assim como em um formato pode ser utilizado uma só opção para realizar uma especifica funcionalidade do comando, que no outro se precisa ser realizado em dois comandos separados.

A principal diferença que há entre o UPF e o CPF, pelo menos no início da construção de ambos os formatos, está relacionado a capacidade de definir atributos de bibliotecas. O UPF não dispõe dessa funcionalidade e portanto, não é capaz de por meio da sua sintaxe definir o tipo de célula que se deseja utilizar no *power intent*, assim como nomes dos pinos de alimentação e o nome dos pinos de entrada e saída de dados. O UPF assume que exista algum outro formato de biblioteca e ele só precise utilizar das

informações contidas no mesmo. Entretanto, o CPF dispõe de sintaxes para definir os atributos dessas bibliotecas.

Visto que este trabalho não tem como escopo abordar diretamente o CPF, é possível verificar mais algumas diferenças que há entre os dois formatos em (ALLEN, 2008).

A ideia principal, é a percepção de que, para se traduzir um formato a outro, caso seja necessário devido alguma particularidade de ferramentas, nenhum comando poderá necessariamente entendido como igual em outro formato apenas pelo fato dos dois terem a mesma sintaxe ou serem parecidos, como é o caso da criação do *power domain*.

3.5 DIFERENÇAS ENTRE UPF 1.0 E 2.0

Com a necessidade de se pôr em práticas as técnicas de baixo consumo de energia, a indústria de EDA se viu impossibilitada de realiza-las, até que no início do ano de 2007 começaram a surgir as descrições de intenção de consumo, ou como corriqueiramente são chamadas na área de EDA, o *power intent design*. Estas, possibilitam que as intenções de *power* advindas das técnicas de baixo consumo sejam postas em prática, já no início do fluxo de desenvolvimento do hardware.

A organização Accellera Systems Initiative foi a responsável pela criação da primeira versão do UPF, conhecido como UPF 1.0. Um outro formato que também visava a implementação da intenção do consumo de energia, foi criado mais cedo naquele mesmo ano de 2007, e tinha como responsável por seu desenvolvimento inicial o grupo Silicon Integration Initiative. Posteriormente, a Accellera doou seu projeto ao IEEE e este por sua vez, realizou melhorias e o tornou o formato padrão de *power intent*. Em 2008, o UPF foi entregue a organização de projeto IEEE P1801, que lançou o padrão IEEE 1801-2009, popularmente conhecido como UPF 2.0, em março de 2009.

Levando em conta a utilização dos formatos e suas versões, como pode ser visto na Figura 4, o padrão mais utilizado nos últimos anos tem sido o UPF 2.0, mesmo após o lançamento do padrão 3.0. Por esse motivo e aliado ao fato do UPF 1.0 ser uma base para melhor entendimento de conceitos do formato e assim, utilizadas para quem está aprendendo a realizar descrição de *power intent*, se torna interessante a comparação entre esses dois formatos. É possível perceber também como o formato vem evoluindo ao longo de suas versões.

Lançado pela Accellera, o UPF 1.0 contava com 32 comandos relacionados a descrição de intenção de potência. Tais comandos englobavam conceitos como *power domain*, especificação de como seria alimentado os domínios, quais a portas e quais tensões seriam utilizadas, assim como aplicabilidade das estratégias de isolamento, retenção, descoladores de níveis, entre outros.

Com a padronização do formato pelo instituto IEEE, veio também a nova versão do padrão, o *IEEE Standard 1801-2009*, também conhecido como UPF 2.0. A nova versão do formato trouxe diversas melhorias relacionadas aos comandos utilizados para expressar a intenção de *power*. Mas não só isso, a versão também trouxe uma maior abstração ao formato relacionado a distribuição de energia dentro dos domínios.

A versão publicada em 2009, trouxe consigo 57 novos comandos. Somados a antiga versão 1.0, o formato passou a ter 88 comandos, dos quais tem: 49 relacionados a descrição de intenção de energia e 39 destinados ao auxílio do formato (NEUKOM, 2016).

Um conceito bastante importante introduzido ao formato UPF foi o de conjunto de distribuição de energia (do inglês, *supply set*).

Na versão 1.0, para se energizar os domínios dentro do design, era necessário a criação de portas dos quais vinham as alimentações de tensão (estes valores de tensões eram passados especificamente pelo *testbench* utilizados durante a simulação), posteriormente era necessário a criação dos *nets*, ou fios, e conecta-los aos domínios. Porém, fios não podem ser reutilizados para diversos domínios, sendo assim necessário criar conjunto de fios ou realizar a extensão dos já existentes, sempre que fosse realizar a alimentação de um domínio, o mesmo se segue para as células utilizadas para realizar as estratégias de isolação e as demais.

O conceito de *supply set* veio para simplificar as descrições dentro do formato. Dessa forma, os domínios teriam conjuntos de distribuição contando com portas e fios que apenas precisavam ser conectado a alimentação. Assim, diversos domínios poderiam compartilhar do mesmo conjunto, diminuindo assim a quantidade de conexões necessárias para realizar a intenção de consumo. Além disso, possibilitou que domínios que possuíssem mesmos níveis de tensão em sua alimentação e o mesmo estados de energia (momento em que são ligados ou desligados) utilizassem do mesmo chaveamento de tensão.

Relacionados aos comandos, um dos principais pontos adicionados pelo UPF 2.0 foi a inserção da opção **-instance**, que provê a capacidade de utilizar das células de estratégicas já existentes no design. O comando torna possível a detecção dessas células pre-existentes no design. A utilização do comando evita duplicatas caso já existam, e assim previne que a ferramenta adicione as células novamente. Mas não só isso, como torna possível dizer exatamente a ferramenta como essas células precisam ser alimentadas. Esta opção permite a instanciação das células dentro de domínios que serão chaveados, para que assim, mesmo quando estes forem desligados, as células presentes continuarão sendo alimentadas e funcionarão propriamente (KHONDKAR, 2018).

Por fim, a opção **-instance** permite que qualquer ferramenta de verificação certifique se essas células inseridas manualmente, funcionem corretamente, assim como se elas fossem inseridas no design automaticamente pela ferramenta. Além disso, elas utilizam

as células especificadas com a opção como padrão para as demais células que não são especificadas no arquivo UPF.

Outro ponto importante do UPF 2.0 também está relacionado a uma opção de comando. A opção **-update** torna possível a atualização de informações nos comandos, pre-existentes no arquivo devido as fases de design e verificação. Este comando foi adicionado a vários comandos existentes da criação da versão 1.0 e também a diversos comandos que foram adicionados com a nova versão. Portanto, não é mais necessário reescrever um comando ou voltar no comando já existente e editá-lo, sempre que uma nova informação surgir por meio do fluxo de desenvolvimento e esta precisar ser atualizada no arquivo UPF.

A atualização das informações de um comando por meio da opção **-update** pode ser realizada através de um outro arquivo upf, utilizando o comando e o nome da instância anteriormente criada. A opção de atualização se torna muito útil e desejável quando informações acerca do design não são conhecidas tão precocemente no processo. Mais detalhes sobre essas principais opções implementadas na versão 2.0 do UPF, serão melhor abordadas mais a frente no texto.

3.6 PRINCIPAIS COMANDOS USADOS NO UPF

A modelagem do UPF se caracteriza como a criação de um mapeamento de elementos que estão relacionados a energia. As intenções de potência são os resultados da aplicação das técnicas de economia de energia apropriadas para o design projetado. A descrição dessas intenções são os objetivos a serem alcançados com as descrições detalhadas dentro do arquivo criado.

O desenvolvimento do arquivo UPF começa com os pré-requisitos estipulados na especificação do projeto a ser criado. Assim, a modelagem do UPF é regida pelas especificações do projeto e seus objetivos de intenção de potência.

O arquivo é composto da especificação dos nomes das portas e tensões do qual o design será alimentado, dos nomes dos fios que realizará a distribuição de energia entre os domínios de potência e elementos do design, e da quantidade de *power domains* que serão necessários devido as intenções de *power*. Adicionado a estes, pode conter as estratégias aplicadas devido as especificações do projeto, como domínios operando em diferentes níveis de tensão ou diferentes estados de operação dos blocos que compõe o projeto.

Assim como toda linguagem ou descrição, existem componentes que são consideradas essenciais para o desenvolvimento do arquivo. No caso do UPF, os conceitos imprescindíveis ao formato são (KHONDKAR, 2018):

- Escopo de Desenvolvimento no UPF (*Design Scopes*);

- Domínios de potência (*Power Domains*);
- Interfaces e Limites dos Domínios de Potência;
- Fornecedores de Energia (*Power Supply*) e Distribuidores de Energia (*Power Supply Networks*);
- Alimentação de Energia Primária e Nível de Referência de Energia Primária;
- Estados de Potência (*Power States*) e Modos de Operação de Energia;
- Estratégias de Potência (ex: Isolações, Retenções...)

Com o intuito de abordar os conceitos essenciais ao formato afim de tornar possível que qualquer leitor com conhecimento básico de *SystemVerilog* realize a criação de um arquivo UPF para acompanhar a sua descrição de hardware, este capítulo detalhará os comandos da versão 2.0 respectivos a estes conceitos, seus principais parâmetros e opções. É válido lembrar que nem todos os comandos e opções desta versão estão contidos nesse tópico e que todos os comandos aqui representados são uma reprodução do documento oficial do padrão (IEEE. . . , 2009).

3.6.1 SINTAXE DA LINGUAGEM

A descrição do formato é realizado pela utilização de comandos e opções desses comandos, assim como em uma descrição de hardware, por exemplo. Nos tópicos seguintes serão abordados as sintaxes desses comandos e suas principais opções. Também é disposto exemplos de utilização dos comandos, de modo a facilitar o entendimento.

Algumas observações importantes a respeito da linguagem e da extensão do arquivo valem a pena o respaldo. São elas:

- O UPF, assim como o CPF, são formatos com extensões próprias, sendo assim, ".upf" e o ".cpf", respectivamente;
- Não é necessário a presença de um finalizador de sentença (como o ";" no *SystemVerilog*, e portanto para se continuar uma sentença na linha de descrição seguinte se faz necessário colocar um "\" (contra-barra);
- As chaves podem ser utilizadas como os parênteses em outras linguagens (*SystemVerilog* e C++, p. ex.) e são recomendadas para organização de expressões dentro do código;
- Finalizado uma **opção** é possível acrescentar uma outra logo em sequência na mesma linha, contudo separar as opções em várias linhas torna o código mais legível, algo importante devido a futuras etapas do fluxo de desenvolvimento;

- Para o comentário durante a descrição é utilizado o símbolo "#" (cerquilha).

3.6.2 DEFINIÇÃO DE VERSÃO

As ferramentas de simulação e verificação utilizadas para execução do formato juntamente com o código RTL necessitam saber qual a versão que o formato foi descrito. Dessa forma, a ferramenta pode verificar se algum comando foi escrito de forma correta ou não, de acordo com a versão utilizada para descrição do formato.

O comando possui sintaxe simples e geralmente é utilizado no início do arquivo.

Proposta	Definir o tipo de versão do arquivo
Sintaxe	upf_version *versão_utilizada*
Argumentos	*versão_utilizada* Tipo de versão em que o arquivo será considerado quando utilizado dentro do fluxo e ferramentas

Exemplo de sintaxe:

```
1 upf_version 1.0
```

No exemplo acima, é possível ver que foi indicado a versão 1.0 do UPF. Se o caso fosse de utilizar a versão 2.0, 2.1 ou demais, apenas trocaria a numeração de 1.0 pela desejada.

3.6.3 DEFINIÇÃO DE ESCOPO

Proposta	Definir a referência a partir de onde os comandos serão executados
Sintaxe	set_scope *local_do_escopo*
Argumentos	*local_do_escopo* Local ou elemento ao qual será definido o escopo de referência

Normalmente se define como escopo o topo do HDL, ou em outras palavras, o mais alto nível de hierarquia do HDL. Tomando como base o topo do HDL, os demais elementos que compõe o hardware serão subelementos do topo, e terão o topo como referência quando for necessário referenciar os elementos que compõe o bloco.

É válido lembrar que, quando não especificado os elementos de determinado comando, o mesmo será realizado onde o escopo foi definido pela última vez.

Exemplo de sintaxe:

```
1 set_scope PLP_core
2
```

```
3 ..
4
5 set_scope PLP_core/ram
```

Na linha 1, o escopo para trabalho é definido como o elemento definido no código RTL como PLP_core. Na linha 5, o escopo de trabalho é redefinido para a instância PLP_core/ram que foi instanciada dentro do componente PLP_core no código RTL.

3.6.4 *POWER DOMAIN*

Considerado como um dos principais conceitos do UPF, o *power domain* é o alicerce por onde todo o *power intent* acontece. O *power domain* é utilizado para englobar elementos do HDL que utilizam mesma tensão e possuem os mesmo estados de energia (quando sinais permanecem ligados, trabalham em uma diferente tensão ou são desligados ao mesmo tempo).

Proposta	Definir um domínio de potência e suas características	
Sintaxe	create_power_domain *nome_de_domínio* **-elements** *lista_de_elementos* **-exclude_elements** *lista_de_elementos_excluídos* **-supply** {*supply_set_do_domínio* [supply_set_referencia]} **-include_scope** *nome_da_instancia* **-scope** *nome_da_instancia* **-update**	
Argumentos	*nome_de_domínio*	Nome dado a instância de domínio criado
	-elements	Lista de elementos que estarão dentro do domínio
	-exclude_elements	Lista de elementos que serão excluída do domínio, mesmo que seja elemento filho de um elemento incluso
	-supply	Define o conjunto de alimentação do domínio, *supply_set_do_domínio* é o nome dado ao conjunto e se for definido o *supply_set_referencia*, o domínio criado terá como base as conexões da referência
	-include_scope	Cria o *power domain* dentro do escopo atual
	-scope	Cria o *power domain* dentro do escopo *nome_da_instancia*

Argumentos	**-update**	Utilizado para atualizar domínios pre-existentes no arquivo, se faz necessário utilizar o mesmo nome de domínio (nome_de_domínio)

Por meio das opções **-elements** e **-exclude_elements** é possível adicionar quais instância do HDL irão compor o domínio.

Quando utiliza-se a opção **-elements** para a inclusão de um componente no domínio, todos os elementos filhos desse componente também serão inclusos, ao menos que seja realizado a exclusão desses elementos filhos com o comando **-exclude_elements**, ou posteriormente com a criação de um outro *power domain* que englobe especificamente esse elemento filho.

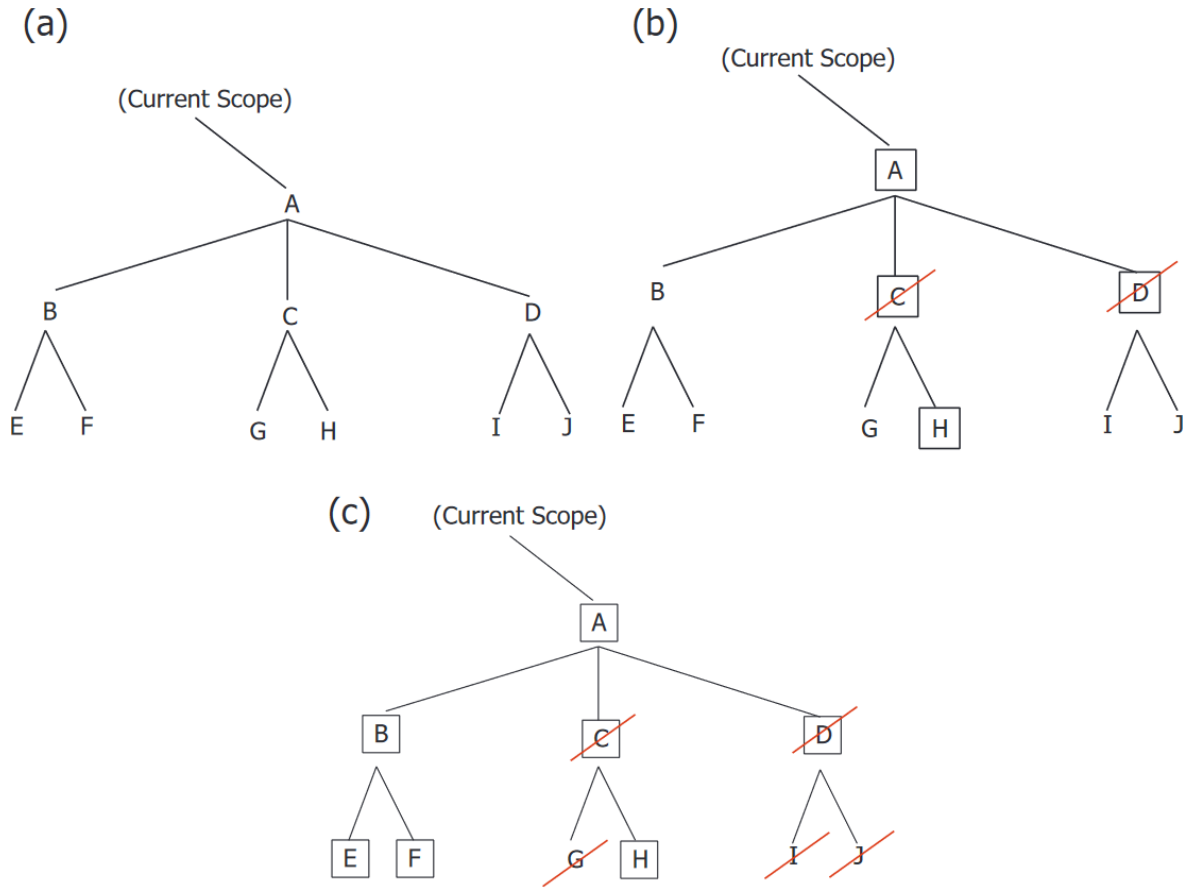


Figura 5 – (a) Exemplo de estrutura de elementos de design. (b) Desenvolvimento da especificação. (c) Resultado do desenvolvimento segundo a especificação

Fonte – (IEEE. . . , 2019)

Uma analogia semelhante pode ser realizada para a inclusão de um elemento que foi excluso pelo comando *-exclude_elements*. Assim como na inclusão, a exclusão também se propaga para toda a hierarquia do componente. Então, se um elemento foi excluso e se

deseja que um elemento filho seja incluso no domínio especificado, é necessário a inclusão do elemento filho pelo comando **-elements**.

Na Figura 5 (a) temos a estrutura dos elementos de um design. Deseja-se a inclusão dos elementos “A”, “B” e seus descendentes, porém não é desejado os elementos “C” e “D” e seus descendentes com exceção do elemento “H” filho do elemento “C”.

De forma a exemplificar o comando e o exemplo da Figura 5, temos:

```
1 create_power_domain PD_core \
2 -elements {A A/C/H}          \
3 -exclude_elements {A/C A/D}
4 -supply PD_core_ss VDD_ss
```

Considerando novamente a Figura 5, o exemplo (b) nos mostra a exclusão do elementos filhos do componente “A” e como consequência da exclusão desses, seus descendentes também serão exclusos, por isso se faz necessário a inclusão do elemento filho de “C”, o elemento “H”, por meio da opção **-elements**.

3.6.5 *SUPPLY SET*

O conjunto de distribuição de energia, conhecido dentro do UPF como *supply set*. É um conjunto de portas e fios, que pode ser utilizado por mais de um domínio ou utilizados como referência para a criação de outros conjuntos. Eles tem como responsabilidade a distribuição de energia dentro do *power intent*. Esse conceito passou a existir após a versão 2.0 e busca simplificar a descrição dessa distribuição.

Proposta	Definir um conjunto de distribuição de energia ou atualiza conjunto pre-existente ou um *supply set handle* (*supply set* criado por meio do *power domain*, por exemplo)	
Sintaxe	create_supply_set *nome_do_conjunto* **-function** {*tipo_func nome_do_fio}* **-update**	
Argumentos	*nome_do_conjunto*	Nome dado a instância do conjunto criado
	-function	A opção define o tipo de função energética que o fio acoplado a ela terá. O *nome_do_fio* é o nome de um fio criado (*supply net*) que será associado a função
	-update	Utilizado para atualizar conjuntos de potência pre-existentes no arquivo, se faz necessário utilizar o mesmo nome de conjunto (nome_de_conjunto)

Dentro do comando, existe a opção **-function** que define o nível energético que o fio conectado ao conjunto terá. Existe sete escolhas para definir a função: *power, ground, nwell, pwell, deepnwell e deeppwell*, ou pode ser criada uma palavra. As palavras reservadas **power** e **ground** são as mais utilizadas, e tem como objetivo ser a alimentação e o terra do conjunto, respectivamente.

Caso ainda não seja conhecido os níveis de energia das funções quando criado a função, é possível utilizar a opção **-update** para posteriormente adicioná-las.

Exemplo de sintaxe:

```
1 create_supply_set VDD_RCVR1_ss \
2 -function {power VDD_rcvr1_out_sw_net} -function {ground VSS}
```

3.6.6 ASSOCIAÇÕES DE *SUPPLY SET*

Após a criação de *supply sets* é possível fazer a associações entre dois ou mais conjuntos, principalmente se parâmetros de funções não tenham sido definidos. A associação copia a configuração de um conjunto para outro.

Proposta	Associar dois ou mais *supply sets*	
Sintaxe	associate_supply_set *lista_de_conjuntos* **-handle** *conjunto_ref*	
Argumentos	*lista_de_conjuntos*	Lista de instâncias de conjuntos a ser associado ao conjunto referência
	-handle *conjunto_ref*	Nome do conjunto de distribuição de *power domain*, *power switch* ou estratégia a ser referenciado para associação

Exemplo de sintaxe:

```
1 create_supply_set PD_axi
2
3 associate_supply_set PD_axi -handle VDD_RCVR1_ss
```

3.6.7 *SUPPLY PORT*

Como a tradução direta já é bastante sugestiva, o *supply port*, é a porta criada para alimentação. A ela é conectado um fio para poder ser efetivamente utilizada. Geralmente são utilizadas para tornar possível as entradas de tensão dentro do *power intent*.

Proposta	Criar uma porta de alimentação para o domínio	
Sintaxe	create_supply_port *nome_da_porta* **-domain** *nome_do_domínio*	
Argumentos	*nome_da_porta*	Nome dado a porta
	-domain *nome_do_domínio*	Domínio onde a porta será criada

Exemplo de sintaxe:

```
1 create_supply_port VDD
```

No exemplo, é possível perceber que não foi especificado um domínio para a porta, portanto a mesma será criada no atual escopo do momento da criação.

3.6.8 *SUPPLY NET*

Utilizado para tornar possível a distribuição de energia dentro do formato. O *supply net* é o fio de conexão entre portas e componentes dentro da descrição.

Proposta	Criar um fio de alimentação	
Sintaxe	create_supply_net *nome_fio* **-domain** *nome_domínio* **-reuse**	
Argumentos	*nome_fio*	Nome da instância de fio criado
	-domain *nome_domínio*	Domínio onde o fio será criado
	-reuse	Utilizado para criar uma extensão do fio de um domínio para o domínio especificado nesse comando

Como fios de um domínio só pertencem a aquele domínio no qual foi criado, a opção **-reuse** torna possível que um fio criado em um domínio possa ser utilizado em outro.

Exemplo de sintaxe:

```
1 create_supply_net AES_net -domain AES
2
3 create_supply_net AES_net -domain AXI \
4 -reuse
```

3.6.9 CONEXÃO DE *SUPPLY NET*

O comando **connect_supply_net** provê a capacidade de conexão entre portas de alimentação e fios de distribuição.

Proposta	Conectar *supply nets* a *supply ports*	
Sintaxe	connect_supply_net *nome_fio* **-ports** *lista_portas*	
Argumentos	*nome_fio*	Nome da instância de fio a ser conectado
	-ports *lista_portas*	Lista de portas ao qual o fio será conectado

Exemplo de sintaxe:

```
1 connect_supply_net AES_net -ports VDD
```

3.6.10 *POWER SWITCH*

O *power switch* funciona como um interruptor e tem como propósito o chaveamento de tensões, conduzindo tensões das portas de entrada para a porta de saída. O *switch* pode funcionar em diferentes estados a depender das expressões que o controlam.

Proposta	Criação de um interruptor de energia	
Sintaxe	create_power_switch *nome_switch* **-domain** *nome_domínio* **-output_supply_port** {*porta_saida nome_fio*} **-input_supply_port** {*porta_entrada* [*nome_fio*]} **-control_port** {*porta_controle* [*sinal_controle*]} **-on_state** {nome_estado porta_entrada {*expressão_booleana*}} **-on_partial_state** {nome_estado porta_entrada {*expressão_booleana*}} **-off_state** {nome_estado {*expressão_booleana*}}	
Argumentos	*nome_fio*	Nome da instância de fio a ser conectado
	-domain *nome_domínio*	Dominio no qual será criado o *switch*
	-output_supply_port	Especifica a porta de saída de energia (*porta_saida*) e opcionalmente o fio conectado a porta (*nome_fio*)

Argumentos	**-input_supply_port**	Especifica a porta de entrada de energia (*porta_entrada*) e opcionalmente o fio conectado a porta (*nome_fio*)
	-control_port	Especifica a porta de controle do *switch* (*porta_controle)* e o sinal ou fio de controle conectado a porta (*sinal_controle*)
	-on_state	Classifica como *on* um estado declarado e funciona de acordo com a *expressão_booleana*
	-on_partial_state	Classifica como parcialmente ligado um estado declarado e funciona de acordo com a *expressão_booleana*
	-off_state	Classifica como *off* um estado declarado e funciona de acordo com a *expressão_booleana*

A opção **-control_port** define uma ou mais portas que serão utilizadas para o controle dos estados. Esses estados são definidos pelas opções **on_state**, **off_state** e **on_partial_state**, e funcionam de acordo com a expressão booleana associada. A expressão booleana é escrita baseada nas portas de controle.

A opção **on_state** define o estado em que o interruptor irá funcionar e qual alimentação ele irá prover como saída, sendo necessário especificar a porta de entrada na sintaxe da opção. A opção **on_partial_state** também provê a saída de uma alimentação, porém com tensão diferente da **on_state** e de acordo com o fio declarado como seu parâmetro.

Exemplo de sintaxe:

```
1 create_power_switch PD_core_sw -domain PD_core      \
2 -output_supply_port {out_core_p VDD_sw_net}         \
3 -input_supply_port {in_core_p VDD_AES_net}          \
4 -control_port {ctrl_core_p CORE_ctrl[0]}            \
5 -on_state {FULL_ON_state in_core_p !ctrl_core_p}
```

No código acima, é criado um *power switch* com controle pelo sinal CORE_ctrl[0] pertencente ao domínio PD_core e alimentação de entrada pelo fio VDD_AES_net.

3.6.11 ESTRATÉGIA DE ISOLAÇÃO

A base do *power intent* é a aplicação das técnicas de economia de energia. A aplicação dessas técnicas muitas vezes se dão nos desligamento de componentes do sistema ou na diminuição da tensão em que eles trabalham. Com o desligamento dos componentes, os seus sinais de saída tem seus valores corrompidos.

De forma a evitar a propagação de sinais corrompidos para o restante do sistema, se é utilizado a estratégia de isolação. Essa estratégia isola um ou mais sinais grampeando esses valores para um valor específico. Dessa forma, quando ocorre o desligamento do componente, é propagado para o restante do sistema os sinais grampeados, invés de sinais corrompido.

Proposta	Definir uma estratégia de isolação	
Sintaxe	set_isolation *nome_isolação* **-domain** *domínio_ref* **-elements** *lista_elementos* **-applies_to** <inputs | outputs | both> **-clamp_value** {*value*} **-isolation_supply_set** *nome_conjunto* **-isolation_signal** {*sinal_isolação*} **-isolation_sense** {*high ou low*} **-update**	
Argumentos	*nome_isolação*	Nome da instância de isolação criada
	-domain *domínio_ref*	Define que a isolação será criada no domínio referenciado em *domínio_ref*
	-elements *lista_elementos*	Lista elementos/sinais nos quais as isolações irá atuar
	-applies_to <inputs | outputs | both>	A opção **-applies_to** define se a isolações irão atuar nas portas de entrada (*inputs*), saída (*outputs*) ou ambas (*both*)
	-clamp_value *value*	A opção **-clamp_value** grampeia os sinais da *lista_elementos* ou sinais especificados na lista para um determinado valor definido por *value*

Argumentos	**-isolation_supply_set** *domínio_ref*	Define o conjunto de potência que será utilizado para alimentação da estratégia
	-isolation_signal *sinal_isolação*	A opção define qual sinal do HDL (*sinal_isolação*) será utilizo para controlar o grampeamento do valor de isolação
	-isolation_sense	Define o nível lógico (*high ou low*) que o *sinal_isolação* deverá ser para ativar a isolação
	-update	Utilizado para atualizar informações da estratégia

Na lista *lista_elementos* é possível não só colocar elementos (blocos) do HDL como também sinais desses elementos. Os valores grampeados com a opção **-clamp_value** podem ser: *low, high, z, latch* (que é o valor atual daquele momento) ou qualquer valor.

É normal, que durante as etapas do fluxo de desenvolvimento não se tenha todos os dados de forma preencher todas as configurações da estratégia e por isso é utilizada a opção **-update** para futuras atualizações a medida que se obtém as informações durante o fluxo.

No comando de isolação, se teve uma mudança significativa da versão 1.0 para a versão 2.0, no que toca ao ponto de controle da isolação. Antes era necessário um comando separado para realizar o controle de quando a isolação iria atuar, porém na versão aqui abordada esse controle é realizado com a opção **isolation_signal**.

Exemplo de sintaxe:

```
1 set_isolation ISO_XMTR0_low -domain PD_XMTR0          \
2 -isolation_supply_set VDD_ss                          \
3 -clamp_value 0                                        \
4 -isolation_sense high                                 \
5 -isolation_signal UART0_XMTR_PV[3]                    \
6 -applies_to outputs -exclude_elements {               \
7 uart0/xmtr/sout}
```

3.6.12 ESTRATÉGIA DE RETENÇÃO

Como antes dito, o sistema costuma desligar seus componentes de forma a realizar as técnicas de economia de energia e com isso, os seus componentes internos, como registradores, também são desligados perdendo assim os seus estados e valores.

A estratégia de retenção atua na retenção de dados e estados dos registradores e sinais que estão a eles relacionados. As células de retenção iniciam a retenção a partir de uma condição mantém os dados retidos até que a condição de restauração seja verdadeira. Diferente da isolação, na retenção não é possível escolher um valor para ser retido.

Proposta	Definir uma estratégia de retenção	
Sintaxe	set_retention *nome_retenção* **-domain** *domínio_ref* **-elements** *lista_elementos* **-exclude_elements** *lista_exclusão_elementos* **-save_signal** *sinal_retenção* <high | low | posedge | negedge> **-restore_signal** *sinal_retenção* <high | low | posedge | negedge> **-save_condition** {*expressão_booleana*} **-restore_condition** {*expressão_booleana*} **-retention_supply_set** *domínio_ref* **-update**	
Argumentos	*nome_retenção*	Nome da instância de retenção criada
	-domain *domínio_ref*	Define que a retenção será criada no domínio referenciado em *domínio_ref*
	-elements *lista_elementos*	Define uma lista de um ou mais elementos ao qual será aplicado a estratégia de retenção
	-exclude_elements *lista_exclusão_elementos*	Define uma lista de um ou mais elementos ao qual NÃO será aplicado a estratégia de retenção
	-save_signal *sinal_retenção*	A opção define qual sinal do HDL (*sinal_retenção*) será utilizado para controlar o momento em que será salvo/-retido o valor dos *lista_elementos*. Se não definido **-elements** a retenção se aplica a todo o domínio no qual foi criado
	-restore_signal *sinal_retenção*	A opção define qual sinal do HDL (*sinal_retenção*) será utilizado para controlar o momento em que será restaurado o valor dos *lista_elementos*, se não definido **-elements** a retenção se aplica a todo o domínio no qual foi criado

Argumentos	**-save_condition** {*expressão_booleana*{	A opção define uma expressão booleana (*expressão_booleana*) que definirá o momento em que será salvo a *lista_elementos*
	-restore_condition {*expressão_booleana*{	A opção define uma expressão booleana (*expressão_booleana*) que definirá o momento em que será restaurado a *lista_elementos*
	-retention_supply_set *domínio_ref*	Define o conjunto de potência que será utilizado para alimentação da estratégia
	-update	Utilizado para atualizar informações da estratégia

Basicamente, as opções que possuem a mesma sintaxe na isolação, aqui exercem a mesma função.

Na opção **-elements**, a *lista_elementos* pode ser uma instância do HDL, registradores ou sinais de instâncias. Contrária a opção anterior, o **-exclude_elements**, semelhante ao caso do *power domain*, é geralmente utilizada quando especificado um domínio ao qual se quer realizar retenção, mas não se deseja a retenção de certos registradores, componentes ou sinais, e então é adicionado os elementos exclusos no campo *lista_exclusão_elementos*.

Exemplo de sintaxe:

```
1 set_retention RET_XMTR0 -domain PD_XMTR0          \
2 -save_signal { UART0_XMTR_PV[4] posedge}          \
3 -restore_signal { UART0_XMTR_PV[4] negedge}       \
4 -retention_supply_set VDD_ss
```

3.6.13 ESTRATÉGIA DE DESLOCAMENTO DE NÍVEL

Assim como as demais estratégias, a estratégia de deslocamento de nível (*level shifter*) também está relacionado as técnicas de economia de energia. Sabe-se que a redução da tensão de alimentação é uma das principais formas de economia de energia, e por isso, alguns elementos do sistema podem operar a uma tensão menor do que outras.

Devido ao fato do sistema trabalha em diferentes níveis de tensão, a estratégia de deslocamento de nível se torna crucial dentro do *power intent*. Ela tem como objetivo converter o nível de tensão de um sinal para outro nível de tensão preservando o valor de dado que esta atrelado ao sinal.

Proposta	Especifica uma estratégia de *level shifter*	
Sintaxe	set_level_shifter *nome_ls* **-domain** *domínio_ref* **-elements** *lista_elementos* **-rule** <high_to_low ou low_to_high> **-applies_to** <inputs ou outputs> **-input_supply_set** *entrada_supply_set* **-output_supply_set** *saída_supply_set* **-update**	
Argumentos	*nome_ls*	Nome da instância de *level_shifter* criada
	-domain *domínio_ref*	Define que o *level_shifter* será criado no domínio referenciado em *domínio_ref*
	-elements *lista_elementos*	Define uma lista de um ou mais elementos ao qual será aplicado a estratégia de retenção, *lista_elementos* pode ser uma instância do HDL ou sinais de instâncias
	-rule	Define o tipo de *level_shifter* que será necessário
	-applies_to	Define qual porta (entrada ou saída) do domínio será deslocada
	-input_supply_set	Define qual *supply set* será usado para alimentar a entrada do *level shifter*
	-output_supply_set	Define qual *supply set* será usado para alimentar a saída do *level shifter*
	-update	Utilizado para atualizar informações da estratégia

Basicamente, as opções **-domain**, **-elements** e **-update** funcionam semelhante a comandos anteriores. A opção **-rule** especifica se a transformação da tensão vai ser de uma tensão menor para maior - *low_to_high*, utiliza-se esse comando reservado logo após a opção - ou se vai ser de maior tensão para menor tensão - *high_to_low* também necessário após a opção.

Na estratégia de deslocamento é necessário a alimentação por dois conjuntos, que serão os dois níveis de tensão que irão ser trabalhados.

Exemplo de sintaxe:

```
1 set_level_shifter LS_RAM_in  \
```

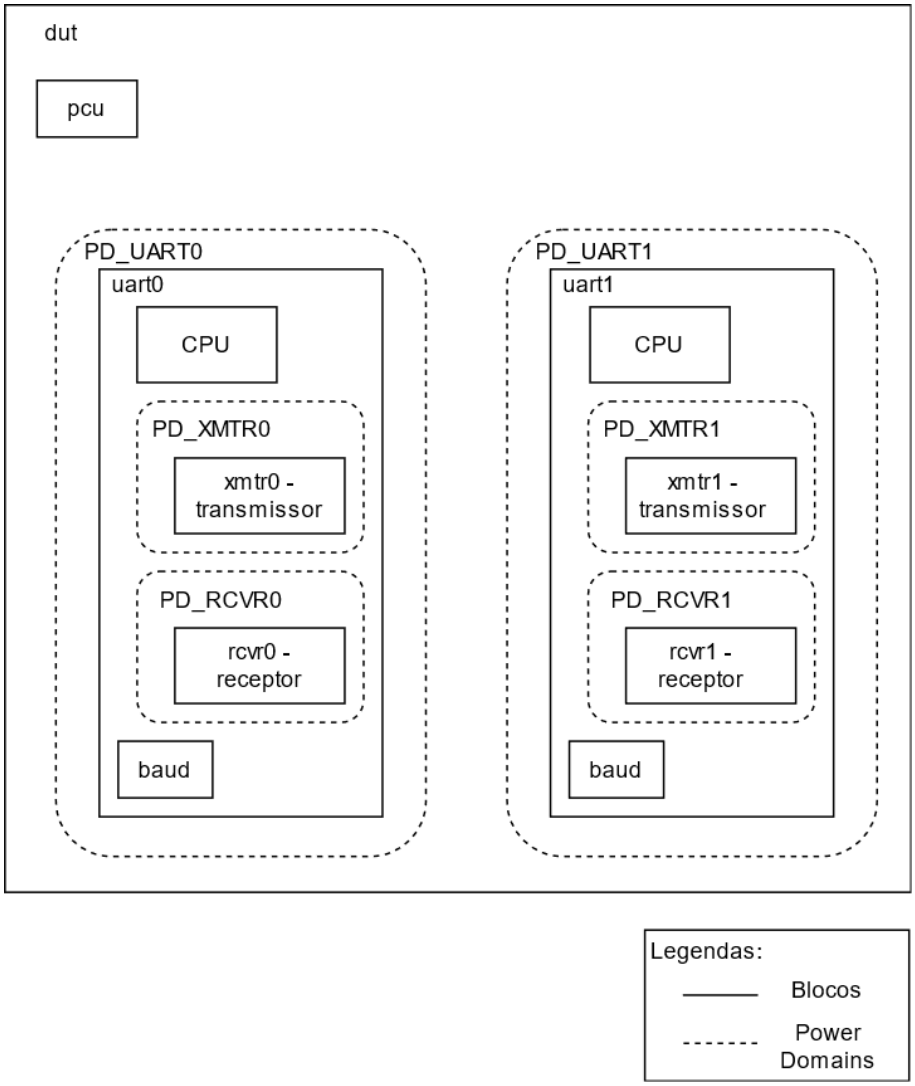
```
2 -domain PD_sram                \
3 -applies_to inputs             \
4 -rule high_to_low              \
5
6
7 set_level_shifter LS_RAM_out   \
8 -domain PD_sram                \
9 -applies_to outputs            \
10 -rule low_to_high             \
```

4 *POWER INTENT* DE UM IP DE BAIXO CONSUMO

De forma a colocar em prática os conhecimentos adquiridos com a revisão bibliográfica do UPF, foi realizado a criação de um arquivo .upf para um IP de baixo consumo.

O IP foi descrito em HDL na linguagem de *SystemVerilog* e tem como macroarquitetura a Figura 6. É composto de duas uarts e uma unidade de controle de energia.

Figura 6 – Macroarquitetura do IP



Fonte – Própria

Para a realização do *power intent* foi utilizado especificações acerca do IP e o projeto em si. As especificações podem ser divididas em quatro categorias: distribuição de energia, controle de energia, requisitos de isolação e requisitos de retenção.

Distribuição de energia:

- O *testbench* provê portas de alimentação de VDD (1.2v), VLL (1.0v) e VSS (0.0v);
- O *power domain* padrão é sempre ligado e alimentado por VDD e contém a instância do DUT;
- Dois *power domains* chaveáveis sempre ligados em que cada domínio contém uma instância da UART;
- Dois *power domains* chaveáveis sempre ligados alimentados pelo domínio da UART e contém em cada: uma instância de transmissor e receptor.

Controle de energia:

- O DUT contém seis vetores de controle de energia para cada domínio. Eles são nomeados como: *UARTn_BASE_PV, UARTn_XMTR_PV, UARTn_RCVR_PV* (onde *n* significa a instância, sendo ou 0 ou 1);
- Cada bit do vetor de controle é responsável por controlar um modo de operação:
  - Bit 0: Modo de alta potência;
  - Bit 1: Modo de baixa potência;
  - Bit 2: Modo de *standby*;
  - Bit 3: Sinal de controle de isolação (*isolation signal* - ativo alto);
  - Bit 4: Sinal de controle da retenção (*save/restore signal - posedge/negedge*);
  - Bit 5: Sinal de desligamento do domínio (ativo alto).

Requisitos de isolação:

- Em cada domínio de UART, grampear o sinal de saída INTR da uart para nível lógico BAIXO enquanto o sinal de controle (*isolation signal*) estiver ativo. Esta isolação deve ser sempre alimentada pelo *supply set* do DUT;
- Em cada domínio de UART, grampear os sinais de saída SOUT, DDIS, OUT2_, OUT1_, RTS_ e DTR_ da uart para nível lógico ALTO enquanto o sinal de controle (*isolation signal*) estiver ativo. Esta isolação deve ser sempre alimentada pelo *supply set* do DUT;
- Em cada domínio de UART, grampear o sinal de saída BAUDOUT_ da uart para nível lógico atual (*latch*) enquanto o sinal de controle (*isolation signal*) estiver ativo. Esta isolação deve ser sempre alimentada pelo *supply set* do DUT;

- Em cada domínio de transmissor, grampear os sinais de saída do transmissor para nível lógico BAIXO enquanto o sinal de controle (*isolation signal*) estiver ativo. Esta isolação deve ser sempre alimentada pelo *supply set* da UART em que está instanciada;

- Em cada domínio de transmissor, grampear apenas o sinal de saída *sout* do transmissor para nível lógico ALTO enquanto o sinal de controle (*isolation signal*) estiver ativo (este controle sobrepõe o controle anterior). Esta isolação deve ser sempre alimentada pelo *supply set* da UART em que está instanciada;

- Em cada domínio de receptor, grampear os sinais de saída do receptor para nível lógico BAIXO enquanto o sinal de controle (*isolation signal*) estiver ativo. Esta isolação deve ser sempre alimentada pelo *supply set* da UART em que está instanciada.

Requisitos de retenção:

- Reter e restaurar todos os registradores dentro de cada *power domain*;

- Todas as retenções deve ser sempre ligadas e alimentadas pelo *supply set* do domínio do DUT.

4.1 RESULTADOS

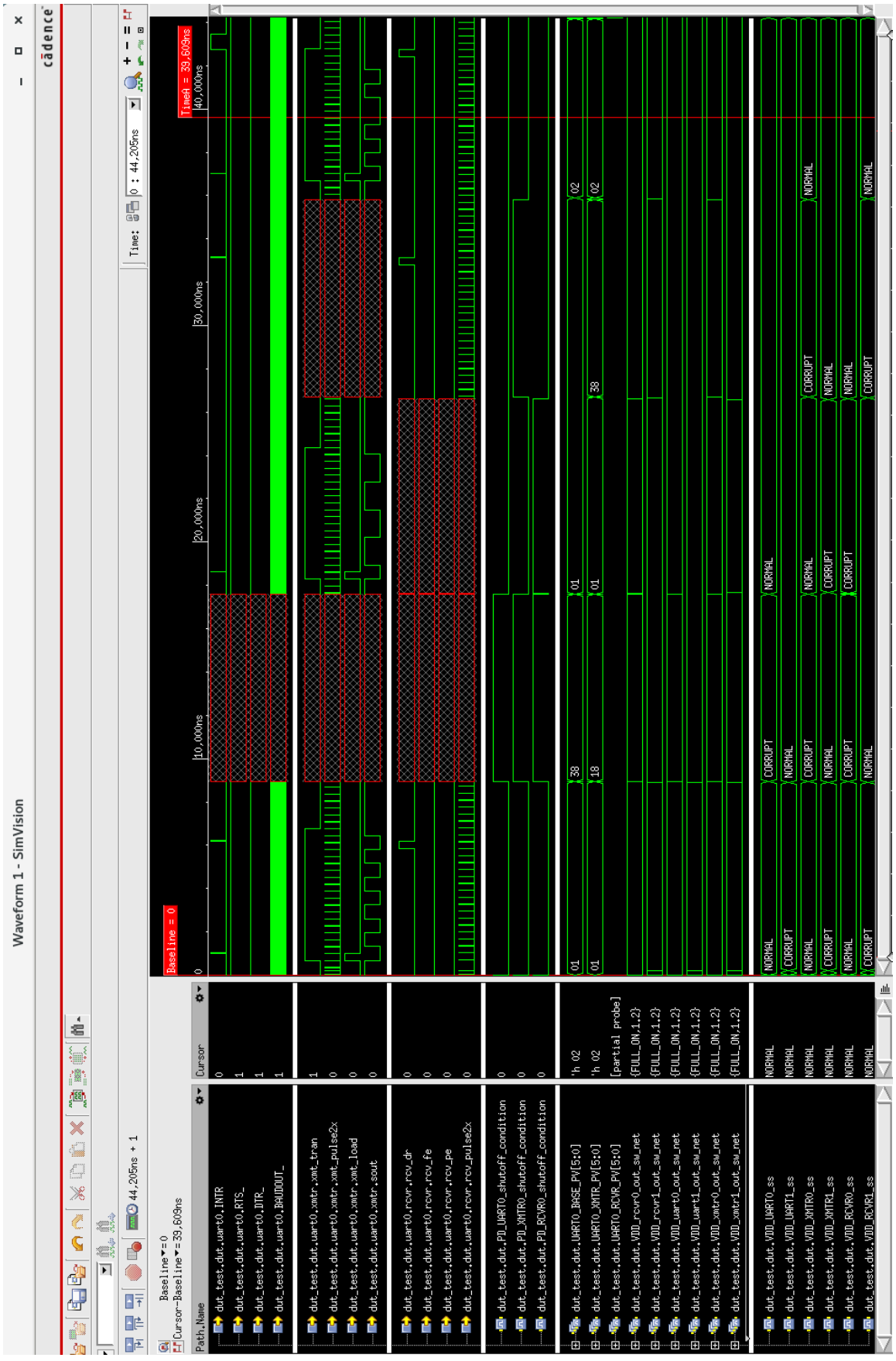
O desenvolvimento do arquivo dut.upf que foi criado para a descrição do *power intent* do IP pode ser visto no Anexo A e teve como base as especificações citadas no tópico anterior.

Para a realização da simulação foi utilizado das seguintes ferramentas distribuídas pela Cadence: Xcelium Single-Core e Cadence Simulation Analysis Environment (SimVision).

A ferramenta SimVision dispõe de recursos visuais em formas de *waveforms* e diagramas. Na Figura 7, se encontra as *waveforms* de alguns sinais utilizados dentro do código RTL, assim como estados de energia de alguns fios. E na Figura 8, é possível visualizar como resultado, o diagrama gerado pela ferramenta a partir da descrição dut.upf em formato UPF, versão 2.1.

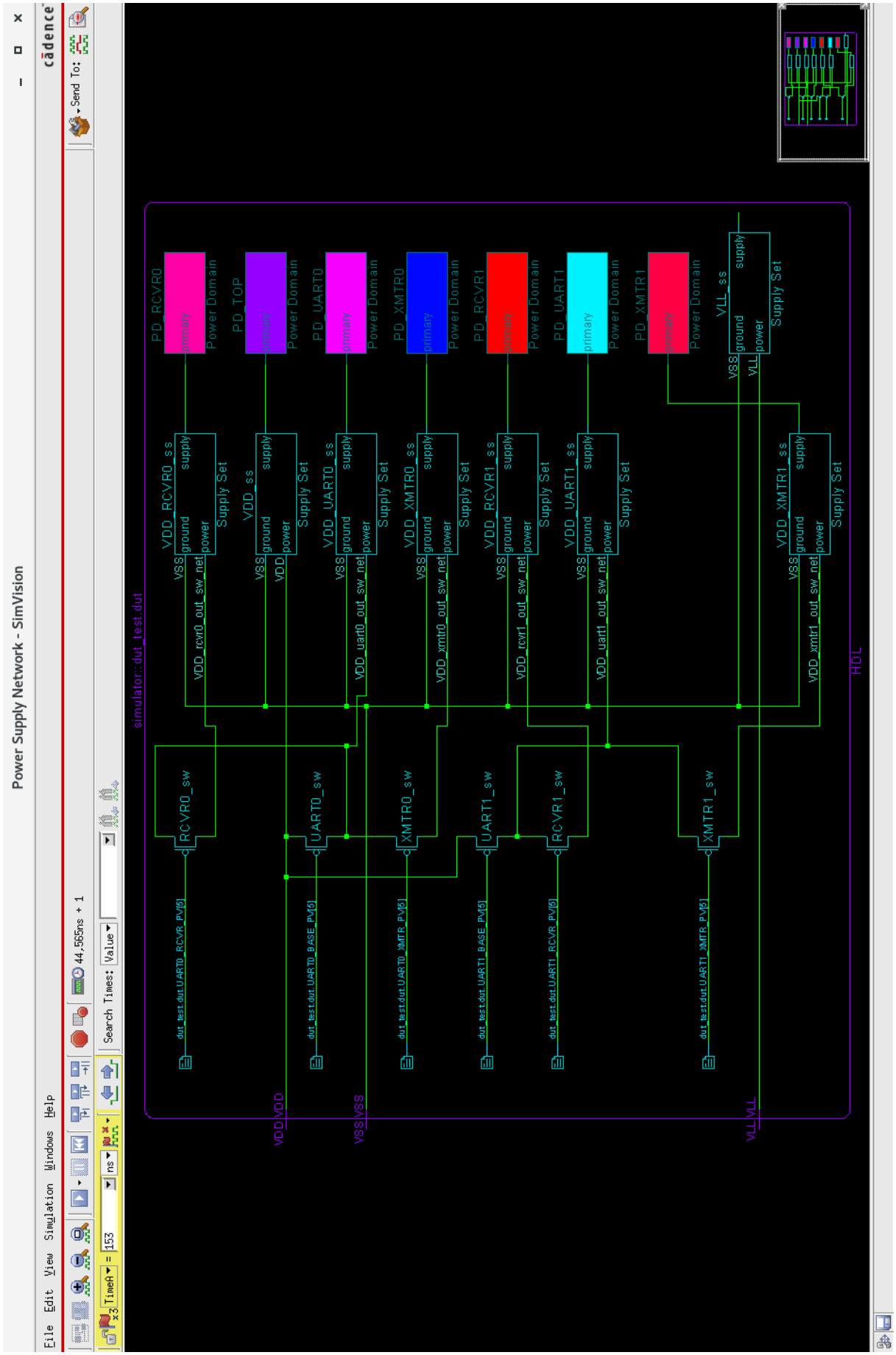
Como etapas futuras do trabalho, seria a realização de *Power Aware Dynamic Simulation*, *Power Aware Dynamic Simulation Coverage* e *Power Aware Static Verification*. Tais desenvolvimentos são realizado após a etapa de implementação do código RTL, no fluxo de desenvolvimento de hardware.

Figura 7 – Simulação do código RTL em conjunto com o formato UPF na ferramenta SimVision



Fonte – Própria

Figura 8 – Diagrama de *Power Supply Network* da ferramenta SimVision



Fonte – Própria

5 CONSIDERAÇÕES FINAIS

Com o desenvolvimento desse trabalho foi possível verificar a importância a intenção de gasto energético no desenvolvimento de hardware. Pois sem ele, técnicas de baixo consumo de energia não poderiam ser aplicados desde as primeiras etapas do fluxo, não sendo assim possível, a maior diminuição possível de energia em um design.

Este trabalho foi importante pelo fato de dispor de forma didática muitos conceitos relacionados ao *power intent*, além do fato de tornar acessível tais assuntos, visto que não se há conteúdo em português que aborde o tema.

Em relação a execução e aplicação do formato, foi possível mostrar através das ferramentas, resultados visuais de descrições que ainda não se concretizaram de forma física, mas que são necessários para futuras etapas do processo de desenvolvimento em baixo consumo.

Como sugestão para trabalhos futuros, poderia ser realizado o prosseguimento do fluxo de desenvolvimento de hardware de baixo consumo, com a execução da verificação com a metodologia *Power Aware Verification* utilizando a descrição de *power intent* desenvolvida nesse trabalho.

REFERÊNCIAS

ALLEN, D. Power formats: You can have it your way. 2008. Disponível em: <https://www.electronicdesign.com/news/products/article/21762765/power-formats-you-can-have-it-your-way>. 21, 22, 23

FOSTER, H. Part 11: The 2018 wilson research group functional verification study. *Siemens: Verification Horizons*, 2019. Disponível em: <https://blogs.sw.siemens.com/verificationhorizons/2019/03/05/part-11-the-2018-wilson-research-group-functional-verification-study/>. 22

HUYGEN, Y.; ETTEN, E. V. Synopsys honors unified power format collaborators with seventh annual tenzing norgay interoperability achievement award. 2007. Disponível em: <https://news.synopsys.com/index.php?s=20295&item=122827>. 21

IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems. *IEEE Std 1801-2018*, p. 1–548, 2019. 19, 29

IEEE Standard for Design and Verification of Low Power Integrated Circuits. *IEEE Std 1801-2009*, p. 1–234, 2009. 26

JONES, R. Modeling and design techniques reduce 90 nm power. 2004. Disponível em: <https://www.eetimes.com/modeling-and-design-techniques-reduce-90-nm-power/>. 16

KHONDKAR, P. *Low-Power Design and Power-Aware Verification*. [S.l.: s.n.], 2018. ISBN 978-3-319-66618-1. 18, 19, 24, 25

MOLLICK, E. Establishing moore's law. *Annals of the History of Computing, IEEE*, v. 28, p. 62 – 75, 08 2006. 11

NEUKOM, L. Upf 3.0 is now official. 2016. Disponível em: <https://www.eetimes.com/upf-3-0-is-now-official/>. 24

SILVA, K. R. G. da. *Uma Metodologia de Verificação Funcional para Circuitos Digitais*. Tese (Doutorado) — Universidade Federal de Campina Grande, 2007. 15

UNIFIED Power Format (UPF) Standard. *Accellera*, v. 1.0, p. 1–104, 2007. 19, 20

A ANEXO - dut.upf

```
1
2  upf_version 2.1
3
4  set_design_top dut_test -testbench
5  set_scope dut_test/dut
6
7  ################## Create Supply Ports and Nets ###################
8
9  create_supply_port VDD
10 create_supply_net VDD
11 connect_supply_net VDD -ports VDD
12
13 create_supply_port VLL
14 create_supply_net VLL
15 connect_supply_net VLL -ports VLL
16
17 create_supply_port VSS
18 create_supply_net VSS
19 connect_supply_net VSS -ports VSS
20
21
22 create_supply_net VDD_uart0_out_sw_net
23 create_supply_net VDD_uart1_out_sw_net
24 create_supply_net VDD_xmtr0_out_sw_net
25 create_supply_net VDD_rcvr0_out_sw_net
26 create_supply_net VDD_xmtr1_out_sw_net
27 create_supply_net VDD_rcvr1_out_sw_net
28
29
30 ################## Create Supply Sets ##############################
31
32 create_supply_set VDD_ss    \
33 -function {power VDD} -function {ground VSS}
34
35 create_supply_set VLL_ss    \
36 -function {power VLL} -function {ground VSS}
37
38 create_supply_set VDD_UART0_ss \
39 -function {power VDD_uart0_out_sw_net} -function {ground VSS}
40
41 create_supply_set VDD_UART1_ss \
42 -function {power VDD_uart1_out_sw_net} -function {ground VSS}
43
```

```
44 create_supply_set VDD_XMTR0_ss \
45 -function {power VDD_xmtr0_out_sw_net} -function {ground VSS}
46
47 create_supply_set VDD_XMTR1_ss \
48 -function {power VDD_xmtr1_out_sw_net} -function {ground VSS}
49
50 create_supply_set VDD_RCVR0_ss \
51 -function {power VDD_rcvr0_out_sw_net} -function {ground VSS}
52
53 create_supply_set VDD_RCVR1_ss \
54 -function {power VDD_rcvr1_out_sw_net} -function {ground VSS}
55
56
57 ################## Create Power Domains ############################
58
59 create_power_domain PD_TOP -include_scope          \
60 -supply {primary VDD_ss}
61
62 create_power_domain PD_UART0 -elements uart0       \
63 -supply {primary VDD_UART0_ss}
64
65 create_power_domain PD_UART1 -elements uart1       \
66 -supply {primary VDD_UART1_ss}
67
68 create_power_domain PD_XMTR0 -elements uart0/xmtr \
69 -supply {primary VDD_XMTR0_ss}
70
71 create_power_domain PD_RCVR0 -elements uart0/rcvr \
72 -supply {primary VDD_RCVR0_ss}
73
74 create_power_domain PD_XMTR1 -elements uart1/xmtr \
75 -supply {primary VDD_XMTR1_ss}
76
77 create_power_domain PD_RCVR1 -elements uart1/rcvr \
78 -supply {primary VDD_RCVR1_ss}
79
80
81 ################## Create Power Switchs ############################
82
83 create_power_switch UART0_sw -domain PD_UART0            \
84 -output_supply_port {out_uart0_p VDD_uart0_out_sw_net}   \
85 -input_supply_port {in_uart0_p VDD}                      \
86 -control_port {ctrl_uart0_p UART0_BASE_PV[5]}            \
87 -on_state {FULL_ON_state in_uart0_p !ctrl_uart0_p}
88
89 create_power_switch UART1_sw -domain PD_UART1            \
90 -output_supply_port {out_uart1_p VDD_uart1_out_sw_net}   \
```

```
91 -input_supply_port {in_uart1_p VDD}                          \
92 -control_port {ctrl_uart1_p UART1_BASE_PV[5]}                \
93 -on_state {FULL_ON_state in_uart1_p !ctrl_uart1_p}
94
95 create_power_switch XMTR0_sw -domain PD_XMTR0                 \
96 -output_supply_port {out_xmtr0_p VDD_xmtr0_out_sw_net}       \
97 -input_supply_port {in_xmtr0_p VDD_uart0_out_sw_net}         \
98 -control_port {ctrl_xmtr0_p UART0_XMTR_PV[5]}                \
99 -on_state {FULL_ON_state in_xmtr0_p !ctrl_xmtr0_p}
100
101 create_power_switch RCVR0_sw -domain PD_RCVR0                \
102 -output_supply_port {out_rcvr0_p VDD_rcvr0_out_sw_net}       \
103 -input_supply_port {in_rcvr0_p VDD_uart0_out_sw_net}         \
104 -control_port {ctrl_rcvr0_p UART0_RCVR_PV[5]}                \
105 -on_state {FULL_ON_state in_rcvr0_p !ctrl_rcvr0_p}
106
107 create_power_switch XMTR1_sw -domain PD_XMTR1                \
108 -output_supply_port {out_xmtr1_p VDD_xmtr1_out_sw_net}       \
109 -input_supply_port {in_xmtr1_p VDD_uart1_out_sw_net}         \
110 -control_port {ctrl_xmtr1_p UART1_XMTR_PV[5]}                \
111 -on_state {FULL_ON_state in_xmtr1_p !ctrl_xmtr1_p}
112
113 create_power_switch RCVR1_sw -domain PD_RCVR1                \
114 -output_supply_port {out_rcvr1_p VDD_rcvr1_out_sw_net}       \
115 -input_supply_port {in_rcvr1_p VDD_uart1_out_sw_net}         \
116 -control_port {ctrl_rcvr1_p UART1_RCVR_PV[5]}                \
117 -on_state {FULL_ON_state in_rcvr1_p !ctrl_rcvr1_p}
118
119
120 ############# Create Isolations and Retentions ##################
121
122 set_isolation ISO_UART0_low -domain PD_UART0                  \
123 -isolation_supply_set VDD_ss                                  \
124 -clamp_value 0                                               \
125 -isolation_sense high                                        \
126 -isolation_signal UART0_BASE_PV[3]                           \
127 -applies_to outputs -elements uart0/INTR
128
129 set_isolation ISO_UART0_high -domain PD_UART0                 \
130 -isolation_supply_set VDD_ss                                  \
131 -clamp_value 1                                               \
132 -isolation_sense high                                        \
133 -isolation_signal UART0_BASE_PV[3]                           \
134 -applies_to outputs -elements { uart0/SOUT                   \
135 uart0/DDIS uart0/OUT2_ uart0/OUT1_ uart0/RTS_                \
136 uart0/DTR_ }
137
```

```
138 set_isolation ISO_UART0_latch -domain PD_UART0      \
139 -isolation_supply_set VDD_ss                        \
140 -clamp_value latch                                  \
141 -isolation_sense high                               \
142 -isolation_signal UART0_BASE_PV[3]                  \
143 -applies_to outputs -elements uart0/BAUDOUT_
144
145 set_isolation ISO_XMTR0_low -domain PD_XMTR0        \
146 -isolation_supply_set VDD_ss                        \
147 -clamp_value 0                                     \
148 -isolation_sense high                               \
149 -isolation_signal UART0_XMTR_PV[3]                  \
150 -applies_to outputs -exclude_elements {             \
151 uart0/xmtr/sout}
152
153 set_isolation ISO_XMTR0_high -domain PD_XMTR0       \
154 -isolation_supply_set VDD_ss                        \
155 -clamp_value 1                                     \
156 -isolation_sense high                               \
157 -isolation_signal UART0_XMTR_PV[3]                  \
158 -applies_to outputs -elements uart0/xmtr/sout
159
160 set_isolation ISO_RCVR0_low -domain PD_RCVR0        \
161 -isolation_supply_set VDD_ss                        \
162 -clamp_value 0                                     \
163 -isolation_sense high                               \
164 -isolation_signal UART0_RCVR_PV[3]                  \
165 -applies_to outputs
166
167
168
169 set_isolation ISO_UART1_low -domain PD_UART1        \
170 -isolation_supply_set VDD_ss                        \
171 -clamp_value 0                                     \
172 -isolation_sense high                               \
173 -isolation_signal UART1_BASE_PV[3]                  \
174 -applies_to outputs -elements uart1/INTR
175
176 set_isolation ISO_UART1_high -domain PD_UART1       \
177 -isolation_supply_set VDD_ss                        \
178 -clamp_value 1                                     \
179 -isolation_sense high                               \
180 -isolation_signal UART1_BASE_PV[3]                  \
181 -applies_to outputs -elements { uart1/SOUT         \
182 uart1/DDIS uart1/OUT2_ uart1/OUT1_ uart1/RTS_      \
183 uart1/DTR_ }
184
```

```
185 set_isolation ISO_UART1_latch -domain PD_UART1       \
186 -isolation_supply_set VDD_ss                         \
187 -clamp_value latch                                   \
188 -isolation_sense high                                \
189 -isolation_signal UART1_BASE_PV[3]                   \
190 -applies_to outputs -elements uart1/BAUDOUT_
191
192 set_isolation ISO_XMTR1_low -domain PD_XMTR1         \
193 -isolation_supply_set VDD_ss                         \
194 -clamp_value 0                                       \
195 -isolation_sense high                                \
196 -isolation_signal UART1_XMTR_PV[3]                   \
197 -applies_to outputs -exclude_elements {              \
198 uart1/xmtr/sout }
199
200 set_isolation ISO_XMTR1_high -domain PD_XMTR1        \
201 -isolation_supply_set VDD_ss                         \
202 -clamp_value 1                                       \
203 -isolation_sense high                                \
204 -isolation_signal UART1_XMTR_PV[3]                   \
205 -applies_to outputs -elements uart1/xmtr/sout
206
207 set_isolation ISO_RCVR_high -domain PD_RCVR1         \
208 -isolation_supply_set VDD_ss                         \
209 -clamp_value 0                                       \
210 -isolation_sense high                                \
211 -isolation_signal UART1_RCVR_PV[3]                   \
212 -applies_to outputs
213
214
215 set_retention RET_UART0 -domain PD_UART0             \
216 -save_signal { UART0_BASE_PV[4] posedge}             \
217 -restore_signal { UART0_BASE_PV[4] negedge}          \
218 -retention_supply_set VDD_ss
219
220 set_retention RET_XMTR0 -domain PD_XMTR0             \
221 -save_signal { UART0_XMTR_PV[4] posedge}             \
222 -restore_signal { UART0_XMTR_PV[4] negedge}          \
223 -retention_supply_set VDD_ss
224
225 set_retention RET_RCVR0 -domain PD_RCVR0             \
226 -save_signal { UART0_RCVR_PV[4] posedge}             \
227 -restore_signal { UART0_RCVR_PV[4] negedge}          \
228 -retention_supply_set VDD_ss
229
230
231
```

```
232 set_retention RET_UART1 -domain PD_UART1                \
233 -save_signal { UART1_BASE_PV[4] posedge}                \
234 -restore_signal { UART1_BASE_PV[4] negedge}             \
235 -retention_supply_set VDD_ss
236
237 set_retention RET_XMTR1 -domain PD_XMTR1                \
238 -save_signal { UART1_XMTR_PV[4] posedge}                \
239 -restore_signal { UART1_XMTR_PV[4] negedge}             \
240 -retention_supply_set VDD_ss
241
242 set_retention RET_RCVR1 -domain PD_RCVR1                \
243 -save_signal { UART1_RCVR_PV[4] posedge}                \
244 -restore_signal { UART1_RCVR_PV[4] negedge}             \
245 -retention_supply_set VDD_ss
```