Universidade de Brasília

Faculdade de Tecnologia

Implementação de técnicas de *Design for Testability* em um processador de arquitetura *RISC-V*

Eduardo Alves Barcelos

TRABALHO DE CONCLUSÃO DE CURSO

ENGENHARIA ELETRÔNICA

Brasília

2023

Universidade de Brasília
Faculdade de Tecnologia

Implementação de técnicas de *Design for Testability* em um processador de arquitetura *RISC-V*

Eduardo Alves Barcelos

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica.

Orientador: Prof. Dr. Gilmar Silva Beserra

Brasília
2023

FICHA CATALOGRÁFICA

Alves Barcelos, Eduardo.

Design for Testability em um processador de arquitetura RISC-VImplementação de técnicas de ; orientador Gilmar Silva Beserra. -- Brasília, 2023.

193 p.

Trabalho de Conclusão de Curso (Engenharia Eletrônica) -- Universidade de Brasília, 2023.

1. RISC-V. 2. DFT. 3. RFID. 4. ASIC. 5. Scan Chain. I. Beserra, Gilmar Silva, orient. II. Título

Universidade de Brasília
Faculdade de Tecnologia

Implementação de técnicas de esign for Testability em um processador de arquitetura ISC-V

Eduardo Alves Barcelos

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica.

Trabalho aprovado. Brasília, 18 de dezembro de 2023:

Prof. Dr. Gilmar Silva Beserra
Orientador

Dr. Prof. Daniel Mauricio Muñoz Arboleda
Examinador interno

Dr. Prof. Wellington Avelino do Amaral
Examinador interno

Brasília
2023

Agradecimentos

Gostaria de expressar minha profunda gratidão a todas as pessoas que desempenharam um papel crucial na realização deste trabalho, uma jornada que reflete a colaboração, o apoio e a dedicação de muitos.

À minha família, um agradecimento especial por serem os pilares constantes ao longo dessa jornada. Mãe, Pai e Irmã seu amor e suporte serviram com uma âncora que me manteve firme durante essa jornada acadêmica. Espero um dia conseguir retornar tudo o que eu recebi.

Ao meu amigo Luis Victor que me acompanhou durante toda a graduação. Sem sua presença constante, a conclusão desta etapa não seria possível. Juntos, enfrentamos cada desafio e celebramos cada conquista, fazendo desta jornada uma experiência inesquecível

Aos professores da graduação, em especial ao meu orientador, Gilmar, e aos professores da banca avaliadora, Daniel Muñoz e Wellington, cuja orientação e ensinamentos foram fundamentais para meu crescimento acadêmico.

Aos meus amigos do colégio Leticia, Paulo, Luis Fábio, Matheus, Pedro, Gabriela, Júlia, Mayra e Marina que me acompanham há anos e continuam fazendo a diferença na minha vida.

A todos os amigos que fiz durante a graduação e me acompanharam não só durante as aulas, mas também as alegrias e as dificuldades da vida universitária. Juntos, formamos uma rede de apoio que enriqueceu esta experiência.

Quero expressar meu sincero agradecimento à EletronJun, pois as experiências e as conexões que fiz na empresa júnior foram vitais para o meu crescimento, tanto pessoal quanto profissional.

Ao final desta etapa, meu agradecimento a todos que contribuíram para este capítulo da minha vida acadêmica. Cada um de vocês é uma parte essencial desta narrativa, e é com profunda gratidão que encerro esta jornada.

Resumo

Encontra-se em desenvolvimento na Universidade de Brasília (UnB) o projeto Cedro, cujo objetivo é a implementação de um SoC (*System-on-Chip)* com capacidade de comunicação *UHF/UWB* (*Ultra High Frequency/Ultra Wideband*) para aplicações de monitoramento na área de saúde. Dentre outros blocos, o SoC possuirá um módulo *DSP* (*Digital Signal Processing*) para processamento de sinais provenientes de sensores. Para aumentar a capacidade de processamento local, foi especificado um processador de baixo consumo baseado na arquitetura *open source* do *RISC-V*. Dada a complexidade da implementação deste projeto em circuito integrado, é crucial garantir a possibilidade de realizar testes que permitam identificar as causas de problemas na funcionalidade do mesmo. Projeto para testabilidade (do inglês, *Design for Testability - DFT*) é um conjunto de técnicas usadas para adicionar características de testabilidade a circuitos integrados tornando-os mais confiáveis, visto que sua funcionalidade pode ser afetada por defeitos de fabricação. Nesse contexto, a proposta do presente trabalho é aplicar técnicas de *DFT* na implementação do processador baseado em *RISC-V* através da utilização das ferramentas de síntese do fluxo de projeto para circuitos integrados digitais da ***Cadence Design Systems***. Assim, com a adição de circuitos como *scan chains* ao processador, espera-se aumentar a controlabilidade e observabilidade e facilitar o processo de teste após a fabricação do mesmo. O trabalho em questão apresenta o fluxo de implementação lógica com técnicas de *DFT*, juntamente com a geração de vetores de teste e a aplicação dos mesmos no circuito pós-síntese. A estratégia escolhida foi a utilização de células *muxed scan-flops*, o que permitiu uma cobertura de falhas de 99,99% do circuito, com um aumento de aproximadamente 16% na área e 18% no consumo de potência.

Palavras-chave: *RISC-V*. *DFT*. *RFID*. *ASIC*. *Scan Chain*.

Abstract

The Cedro project is currently under development at the University of Brasília (UnB), aiming to implement a System-on-Chip (SoC) with UHF/UWB (Ultra High Frequency/Ultra Wideband) communication capabilities for health monitoring applications. Among other components, the SoC will feature a Digital Signal Processing (DSP) module for processing signals from sensors. To enhance local processing capacity, a low-power processor based on the open-source architecture of RISC-V has been specified. Given the complexity of implementing this project in an integrated circuit, ensuring the ability to conduct tests that identify potential functionality issues is crucial. Design for Testability (DFT) is a set of techniques used to incorporate testability features into integrated circuits, making them more reliable as their functionality may be impacted by manufacturing defects. In this context, the proposal of this work is to apply DFT techniques to the implementation of the RISC-V-based processor using synthesis tools from Cadence Design Systems for digital integrated circuit project flow. By adding circuits like scan chains to the processor, the aim is to increase controllability and observability, facilitating the testing process after manufacturing. This work outlines the logical implementation flow with DFT techniques, including the generation of test vectors and their application to the post-synthesis circuit. The chosen strategy involves the use of muxed scan-flops cells, achieving a fault coverage of 99.99% for the circuit, with an increase of approximately 16% in area and 18% in power consumption.

Keywords: RISC-V. DFT. RFID. ASIC. Scan Chain.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

ALU	*Arithmetic Logic Unit*
ASIC	*Application-Specific Integrated Circuit*
ATPG	*Automatic Test Pattern Generation*
BRAM	*Block Random Access Memory*
BSD	*Berkeley Software Distribution*
CMOS	*Complementary Metal-Oxide-semiconductor*
CSR	*Register State Controller*
DFT	*Design for testability*
DSP	*Digital Signal Processing*
DUT	*Devide Under Test*
FPGA	*Field Programmable Gate array*
GDSII	*Graphic Design System*
IO	*Inputs and Outputs*
IP	*Intelectual Property*
ISA	*Instruction Set Architecture*
LSSD	*Level-Sensitive Scan Design*
LUT	*LookUp Table*
PL	*Programmable Logic*
PS	*Processing system*
RFID	*Radio Frequency Identification*
RTL	*Register-Transfer Level*
SoC	*System-on-Chip*
TSMC	*Taiwan Semiconductor Manufacturing Company*
UART	*Universal Asynchronous Receiver/Transmitter*
UHF	*Ultra High Frequency*
ULP	*Ultra-Low Power*
UWB	*Ultra Wideband*
VLSI	*Very Large Scale Integration*
FGA	Faculdade do Gama
RF	Radiofrequência
UnB	Universidade de Brasília

Sumário

1 Introdução

1.1 Contextualização

Encontra-se em desenvolvimento na Universidade de Brasília/Faculdade do Gama (UnB/FGA)- LabMicro o projeto Cedro, que consiste na implementação de um circuito integrado *ULP* (*Ultra-Low Power*) em tecnologia *CMOS* (*Complementary Metal-Oxide-Semiconductor*) 180nm, alimentado por uma unidade *energy harvesting*, excitada por um sinal RF de alta frequência (sub-GHz). O sistema está baseado em um *SoC* (*System on Chip*) que incorpora um sistema híbrido de captação de energia e alimentação por bateria, e permite o monitoramento de sensores e a transmissão de dados por meio da tecnologia *UWB* (*Ultra Wide Band*). A figura 1.1 apresenta a arquitetura do projeto Cedro. O módulo que contém o receptor, o regulador e a referência de tensão é responsável por adquirir a energia da antena *UHF* (*Ultra High Frequency*). Já a unidade de controle de alimentação gerencia a energia do módulo da captação e da bateria. Uma chave de RF determina o modo do sistema (transmissão ou recepção através da antena *UWB*). Os dados recebidos pelos sensores passam pelo conversor analógico-digital e são processados digitalmente pelo bloco *DSP* (*Digital Signal Processing*), escopo deste trabalho. Por fim, o transmissor *UWB* envia os dados processados para uma estação-base.

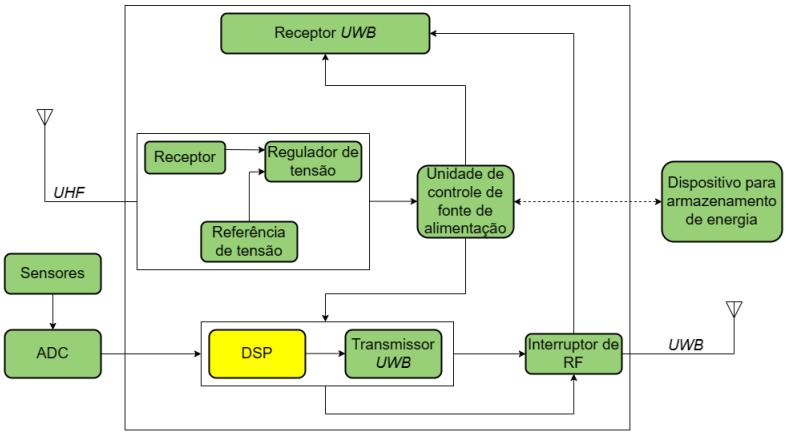

Figura 1.1 – Diagrama de blocos do projeto Cedro

Um objetivo chave para atender os requisitos de *ULP* é reduzir o volume de dados transmitidos, maximizando o processamento local dentro do módulo *DSP*. Para isso, é necessária uma microarquitetura adaptada para processamento digital de sinais com uma boa eficiência energética. A *ISA* (*Instruction Set Architecture*) de código aberto *RISC-V* é uma opção adequada, pois permite a otimização e personalização por meio da seleção de instruções para aplicações específicas.

Alguns blocos isolados da figura 1.1 já foram enviados para fabricação em tecnologia CMOS 0.18 μm e serão caracterizados para validação. Assim, trabalhos futuros consistirão no implementação da arquitetura completa, incluindo um processador RISC-V como módulo DSP.

1.2 Justificativa

Em geral, os chips desenvolvidos precisam apresentar uma faixa de confiabilidade cada vez maior durante sua operação. Tal confiabilidade depende de vários fatores e um dos principais é a sua fabricação, cujo processo está sujeito a erros. Por isso, adicionar estruturas que facilitam o teste dos chips após a fabricação é crucial para garantir a sua funcionalidade. Como a capacidade de trabalhar com sistemas *VLSI* (*Very Large Scale Integration*) aumentou cada vez mais nos últimos anos, os testes desses sistemas também se tornaram cada vez mais complexos, sendo que o custo também aumenta de acordo com a complexidade do *hardware*. A criação de técnicas de projeto para testabilidade permite tornar esse processo menos complexo, adicionando circuitos ao sistema para aumentar a controlabilidade e observabilidade do mesmo. A inserção dessas estruturas em chips pode melhorar a qualidade do produto final, pois elas são responsáveis por identificar o local das variáveis que não obtiveram os valores esperados, assim chegando facilmente à localização do defeito. No caso de circuitos integrados digitais, as ferramentas de síntese automática usadas no fluxo de projeto permitem a inserção de estruturas de teste de forma automatizada.

1.3 Objetivo Geral

Neste trabalho, a proposta é utilizar técnicas de projeto para testabilidade para o processador *RISC-V* especificado para o projeto Cedro utilizando as ferramentas do fluxo de projeto de circuitos integrados e avaliar o impacto que a adição das estruturas terá sobre parâmetros como área, temporização, potência, etc.

1.4 Objetivos Específicos

O trabalho foi dividido em etapas para facilitar a sua execução. Sendo assim, foram definidos os objetivos específicos para cada uma delas:

- Descrição *RTL* (Register-Transfer Level) de um processador RISC-V validada e testada em FPGA;
- Desenvolvimento de *scripts* para ferramenta *Genus* com comandos para inserção de *DFT* (*Design For Testability*) incluídos;
- Obtenção de uma *netlist* mapeada na biblioteca de células padronizadas disponíveis no *design kit* fornecido pelo fabricante;
- Vetores de teste para verificar se cada porta lógica do circuito funciona corretamente;

1.5 Contribuição do Trabalho

A principal contribuição deste trabalho é possibilitar, por meio do aprendizado obtido, a implementação de técnicas de *DFT* em circuitos integrados digitais com as ferramentas ***Cadence***. Em particular, o trabalho em questão visa contribuir com o projeto Cedro, mencionado anteriormente, possibilitando o envio para fabricação de uma versão testável do processador RISC-V. É importante destacar a contribuição inovadora deste trabalho ao grupo, representando o primeiro estudo que aborda a inserção automática de estruturas de teste em um processador RISC-V.

1.6 Aspectos metodológicos

A figura 1.2 mostra o fluxo de projeto de circuitos integrados digitais padronizado pela indústria. O fluxo representa desde o modo mais abstrato do circuito, que é a especificação de *design* até o arquivo *GDSII* (*Graphic Design System*), que é o arquivo enviado para o fabricante. Cada etapa é brevemente descrita a seguir.

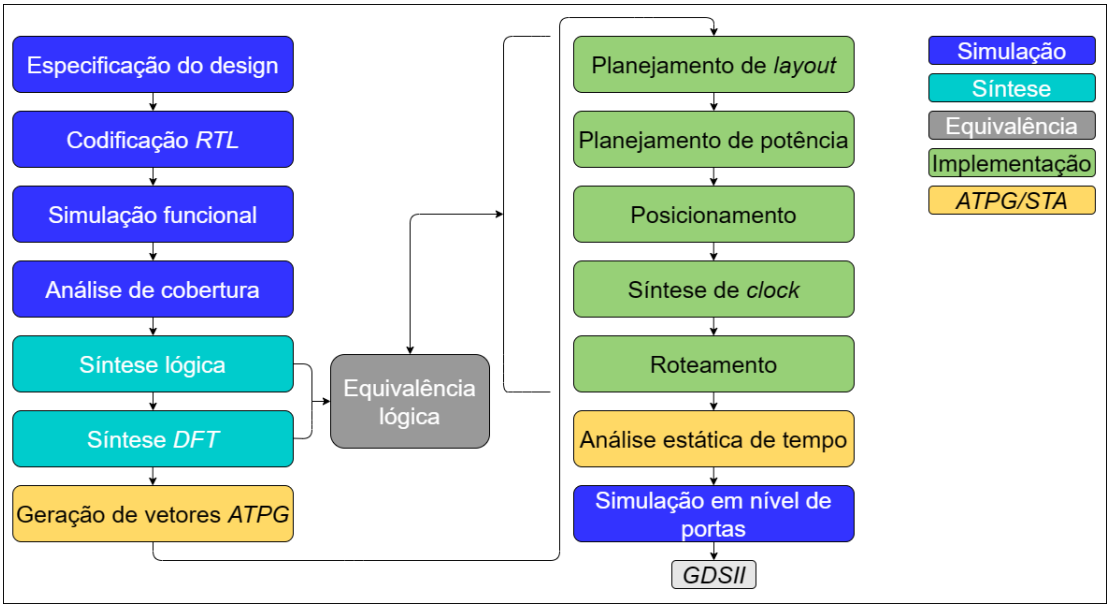

Figura 1.2 – Fluxo de projeto de circuitos integrados digitais, adaptado de Cadence (2023b)

- Especificação do *design*: É a especificação inicial do projeto, onde se define um conjunto explícito de requisitos que precisam ser alcançados no produto final.
- Codificação *RTL*: Nesta etapa, o conjunto de regras e especificações anterior é convertido em um projeto RTL usando uma linguagem de descrição de *hardware*.
- Simulação funcional: Aqui, a ferramenta de simulação compila, elabora e executa o código *RTL*.
- Análise de cobertura: Esta etapa analisa se o *testbench* feito para o sistema cobre todas

as funcionalidades do *RTL*.

- Síntese lógica e *DFT*: Na síntese lógica, a ferramenta traduz o código em um circuito compilado baseado em uma tecnologia específica. A ferramenta *Genus* permite a adição de etapas na síntese lógica para implementar células para testes.
- Verificação de equivalência lógica: Nesta etapa, a ferramenta verifica se a *netlist* gerada pela síntese física é equivalente ao *RTL* gerado pela síntese lógica. A ferramenta *Conformal* permite realizar essa verificação.
- Geração de vetores *ATPG* (*Automatic Test Pattern Generation*): Após a síntese lógica realizada, a ferramenta *Modus* gera os vetores de teste que podem ser simulados na ferramenta ***Xcelium***.
- Implementação física: Engloba as etapas necessárias para a geração do *layout* (*floorplanning*, *powerplanning*, *place and route*, *clock tree synthesis*). É realizada pela ferramenta *Innovus*.
- Análise estática de tempo (*STA - Static Timing Analysis*) e simulação em nível de portas: A STA calcula os atrasos de propagação dos sinais e verifica se há violações dos requisitos de desempenho. A simulação final leva em consideração as informações de atraso obtidas.

Neste trabalho, a especificação inicial é a de uma CPU que implementa o conjunto de instruções do RISC-V RV32IMC chamada PicoRV32. Ela pode ser configurada em diferentes versões e uma comparação da frequência de operação e da utilização de recursos para diferentes versões em FPGAs da Xilinx é feita em (PicoRV32…, 2023).

O código *Verilog* está disponível sob a licença *ISC* (similar à *BSD*). Sendo assim, a etapa de codificação necessita apenas dos ajustes necessários para adaptar a arquitetura à aplicação. Considerando que o *PicoRV32* possui versões com interface nativa de memória, além de interfaces *AXI* e *Wishbone*, é possível expandir a sua arquitetura para acelerar a execução de algoritmos usando *hardware* dedicado. No presente trabalho, as técnicas de *DFT* serão aplicadas em uma versão do *PicoRV32* que permite essa customização. Como estudo de caso, o código do núcleo escolhido será inicialmente validado em *FPGA* através da instanciação de uma rede *MLP* (*Multilayer Perceptron*) descrita em *VHDL*. Após a validação, serão realizadas as etapas de síntese lógica, *DFT* e geração de vetores *ATPG*.

1.7 Organização do trabalho

O presente trabalho contém 7 capítulos organizados da maneira descrita a seguir. No capítulo 2 é apresentada a fundamentação teórica sobre técnicas *DFT*, *FPGAs* e sobre a arquitetura *RISC-V*, bem como uma revisão do estado da arte. No capítulo 3 é apresentado a validação e implementação do *PicoRV32* em *hardware*. O capítulo 4 apresenta o processo de síntese lógica e sua implementação. Já no capítulo 5, é demonstrado a geração de vetores de

teste e como é implementado. Os resultados e as discussões são descritos no capítulo 6 e, por fim, no capítulo 7 é apresentada a conclusão do trabalho, juntamente com os objetivos alcançados e sugestões para trabalhos futuros.

2 Referencial Teórico

2.1 Conceitos de DFT

DFT é um tipo de metodologia que fornece uma forma estrutural de geração de padrões de teste e um facilitador de diagnósticos (Aerts; Marinissen, 1998). O principal objetivo desta metodologia é verificar cada nó, determinando se eles são controláveis e observáveis de forma eficaz, esta metodologia também é capaz de detectar possíveis erros de fabricação (circuitos abertos, metais extras causando curto-circuitos, quebra do dielétrico, impurezas na superfície, etc) causados por problemas no substrato de silício. A detecção de erros de forma física necessita de muito tempo, pois é necessário a execução de todos os testes funcionais em cada um dos milhões de sinais. Logo, houve a necessidade da criação de um método que utilizasse padrões de teste automáticos para verificar o dispositivo físico, então foi criada a técnica *DFT*, onde o fabricante consegue entregar um dispositivo para o seu cliente que seja livre de falhas (Vethamuthu; Sivanantham; Sakthivel, 2018).

2.1.1 Defeitos

O dimensionamento de circuitos de grande escala de integração (*VLSI*) sempre representou desafios para os engenheiros de processos, até recentemente esses projetistas ficavam relativamente protegidos das mudanças qualitativas mais significativas do *CMOS* (Musté, 2021). À medida que o dimensionamento se aproxima dos limites físicos dos dispositivos, variações de processo, ruído e defeitos aparecerão em medidas crescentes (Breuer; Gupta; Mak, 2004). Como já foi dito, um dos principais objetivos da *DFT* é identificar a localização destes defeitos. A figura 2.1, apresenta um exemplo de quebra do dielétrico.

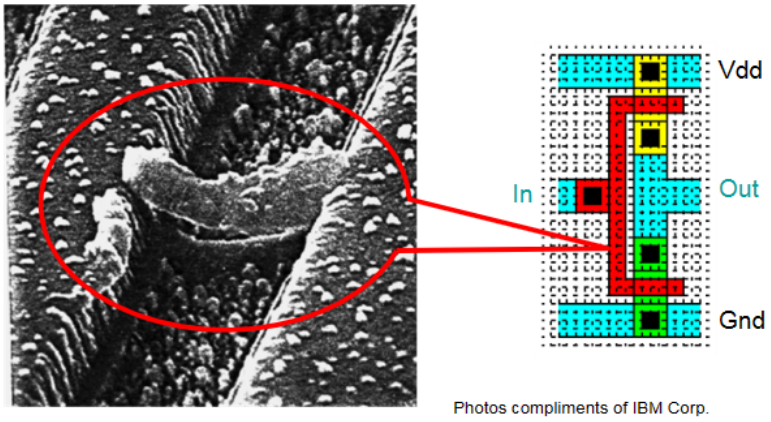

Figura 2.1 – Defeito em um processo de fabricação, imagem retirada de (Cadence, 2023c)

2.1.2 Processo de teste

A etapa de processo de teste visa detectar defeitos de fabricação no chip analisando a lógica da estrutura em busca de falhas de implementação do *RTL*. Este processo é dividido em três modos (Cadence, 2023a):

1. Modo de deslocamento de varredura: É a etapa onde as células de varredura são inseridas. Os dados do vetor de teste são inseridos nessas células por meio de deslocamento e os dados capturados na saída das células.
2. Modo de captura: esta etapa é responsável pela análise lógica combinacional do chip. As células de varredura servem como pseudo-entradas principais (usando dados de vetor de testes gerados por *ATPG*) e depois como pseudo-saídas principais (analisando a saída lógica combinacional).
3. Modo normal: a partir dessa etapa, todas as funcionalidades do chip estão ativas, incluindo qualquer lógica dedicada para *DFT*. O chip executa suas funções regulares e pode ser utilizado em seu ambiente de aplicação previsto.

2.1.3 Células de varredura

Primeiro realiza-se a inserção de varreduras, que substituem *flip-flops* normais por elementos sequenciais escaneáveis (Pandey; Pandey; Hammed, 2017) e, em seguida, estas novas células de varredura são adicionadas em registradores ou cadeias de varredura, estas novas células são conectadas em série para deslocar dados, tanto para dentro quanto para fora da lógica combinacional, durante o modo de teste.

2.1.3.1 *Muxed scan-flops*

O exemplo ao lado esquerdo da figura 2.2 mostra como funcionaria os *muxed scan-flops*, o processo consiste em adicionar *mux* com uma seletora *SE* que selecionaria entre a entrada original *DI* e a entrada desejada *SI*. Quando o sinal *SE* é definido com *"0"*, o *muxed scan-flop* é configurado no modo padrão, ou seja, os dados na entrada *DI* serão observados na saída após uma borda de subida do *CK*. Quando o sinal *SE* é definido com *"1"*, o *muxed scan-flop* é configurado no modo deslocamento, neste modo, a entrada *SI* serve para deslocar novos dados para o *flip-flop* à medida que o valor atual é deslocado para fora (Mannucci, 2018). O modo de operação dos *muxed scan-flop* pode ser observado ao lado direito da figura 2.2.

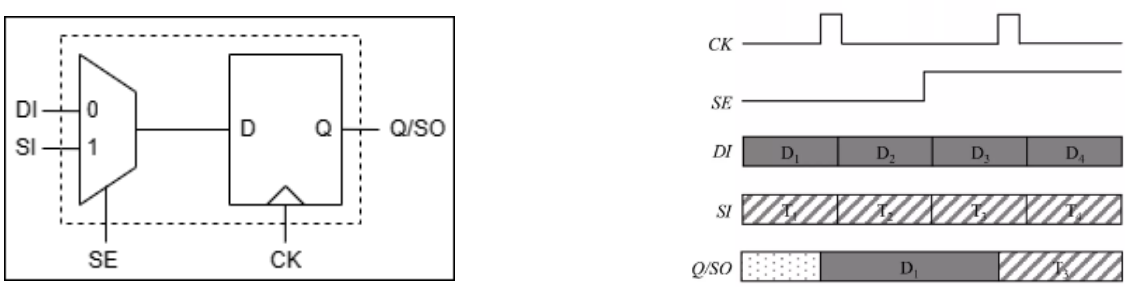

Figura 2.2 – *Muxed scan-flop* e Operação de um *Muxed scan-flop*, disponível em Wang, Wu e Wen (2006)

A figura 2.3 demonstra como funcionaria essa adição, a versão "Antes de inserir os *scan-flops*" tinha três entradas (A, B e DI) e duas saídas (Saída 1 e Saída 2). Esta versão é difícil de inicializar com um estado conhecido, o que torna difícil também o controle interno do circuito e observar seu comportamento usando entradas e saídas iniciais do projeto. Após a adição do circuito de digitalização, obtemos mais duas entradas (SC_IN e SC_EN) e mais uma saída s_out. Os elementos de memória de varredura substituem os elementos de memória originais, pois quando SC_EN estiver ativa, os dados de verificação (lidos a partir da linha SC_IN) e observados em s_out.

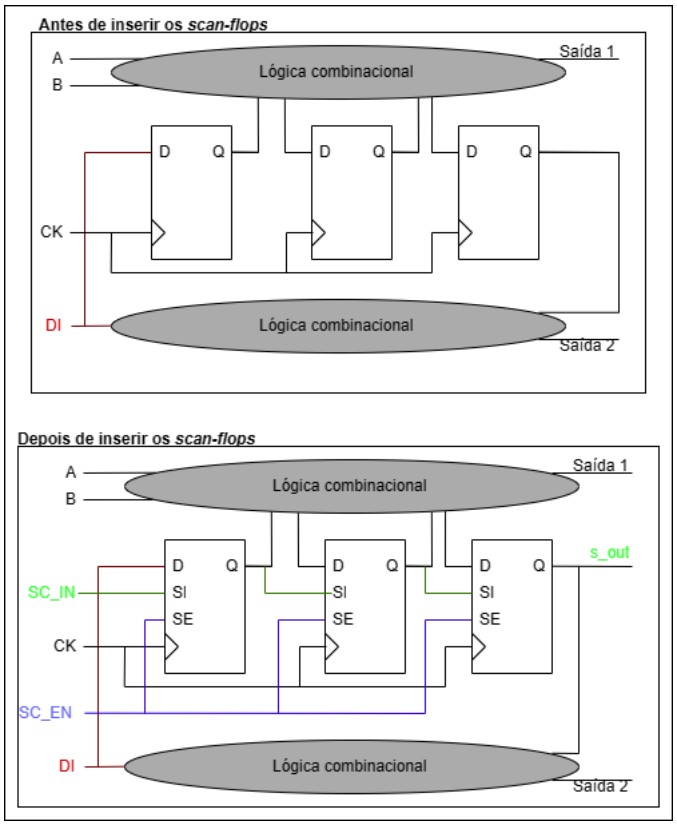

Figura 2.3 – Antes e depois da inserção da varredura

A versão *After Scan* simboliza uma varredura em cadeia, que consiste em transforma

um design sequencial em um design combinacional (Rostami; Karri, 2014). Porém, existe a possibilidade dessa ferramenta de varredura em cadeia seja usada como uma engenharia reversa, obtendo informações sobre a funcionalidade do chip. Isso abre portas para *hackers* comprometerem a segurança de um chip, por isso a importância de ter uma arquitetura segura para o *DFT* (Pandey; Pandey; Hammed, 2017).

2.1.3.2 *Clocked scan-flops*

A diferença do *Clocked scan-flop* é que, ao invés de ser sensível à um sinal seletor, este tipo de célula o dado fica sensível à dois *clocks* independentes *SCK* e *DCK*, como é possível observar na figura 2.4 ao lado esquerdo. Ao lado direito é possívelas formas de onda desta célula. O sinal *DI* é capturado somente na borda de subida de *DCK*, já o sinal de deslocamento *SI* depende da borda de subida de *SCK*..

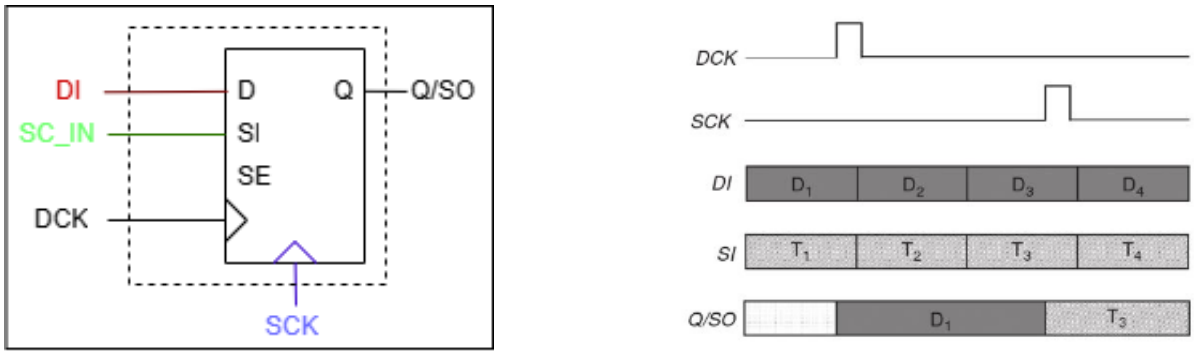

Figura 2.4 – *Clocked scan-flops* e Operação de um *clocked scan-flop*, disponível em Wang, Wu e Wen (2006)

2.1.3.3 Varredura sensível ao nível

Diferentemente dos outros tipos de varreduras que são acionados por uma borda de *clock*, o *LSSD* (*Level-Sensitive Scan Design*, Projeto de Varredura Sensível ao Nível na tradução livre) é baseado em *latchs* acionados por nível, que são circuitos eletrônicos que armazenam e retêm um sinal digital. A figura 2.5 apresenta um exemplo. No lado esquerdo da figura 2.5 observa-se o *latch* principal *L1* e o escravo *L2*, onde os sinais *A*,*B* e *C* são os *clocks*, que são utilizados para selecionar entre as entradas *D* ou *I* para acionar $+L1$ e $+L2$, no exemplo ao lado direito observa-se o comportamento gráfico desta célula.

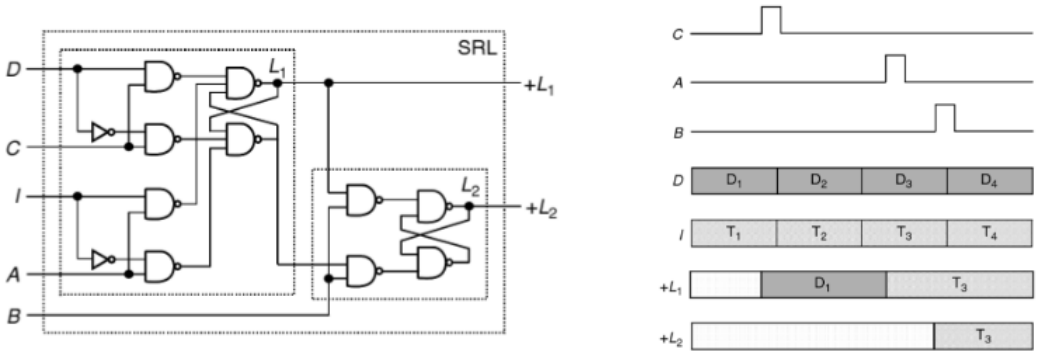

Figura 2.5 – Célula *LSSD* e Operação da célula *LSSD*, disponível em Wang, Wu e Wen (2006)

2.1.3.4 Vantagens e desvantagens das células de varredura

A tabela 2.1 mostra as vantagens e desvantagens de cada tipo de célula de varredura, retirado de (Wang; Wu; Wen, 2006). A ferramenta *Genus* suporta esses três tipos de célula, e a utilizada neste trabalho será a *muxed scan-flops*, pois é o mais utilizado na indústria pela sua compatibilidade com quase todas as ferramentas modernas.

Tabela 2.1 – Vantagens e desvatagens das células de varredura

Tipo de células de varredura	**Vantagens**	**Desvantagens**
Muxed scan-flops	Existe um suporte moderno para as ferramentas de automação existentes e adequado para designs modernos.	A adição de *mux* gera atrasos.
Clocked scan-flops	Não existe uma degradação de performance.	A criação de roteamento de outro *clock*.
Célula LSSD	Baseado em *latchs*.	Aumenta a complexidade de roteamento.

2.1.4 Fluxo de varredura

A implementação das células de varredura ajuda na verificação dos testes, porém necessita de uma atenção durante o processo de implementação de varredura em um projeto *ASIC* (*Application-Specific Integrated Circuit*, Circuito Integrado de Aplicação Específica na tradução livre). Durante o desenvolvimento deste tipo de projeto, há bastante complicações e é necessário modificações no design que o engenheiro de *DFT* precisa considerar. A figura 2.6 apresenta um fluxo para a inserção de varredura.

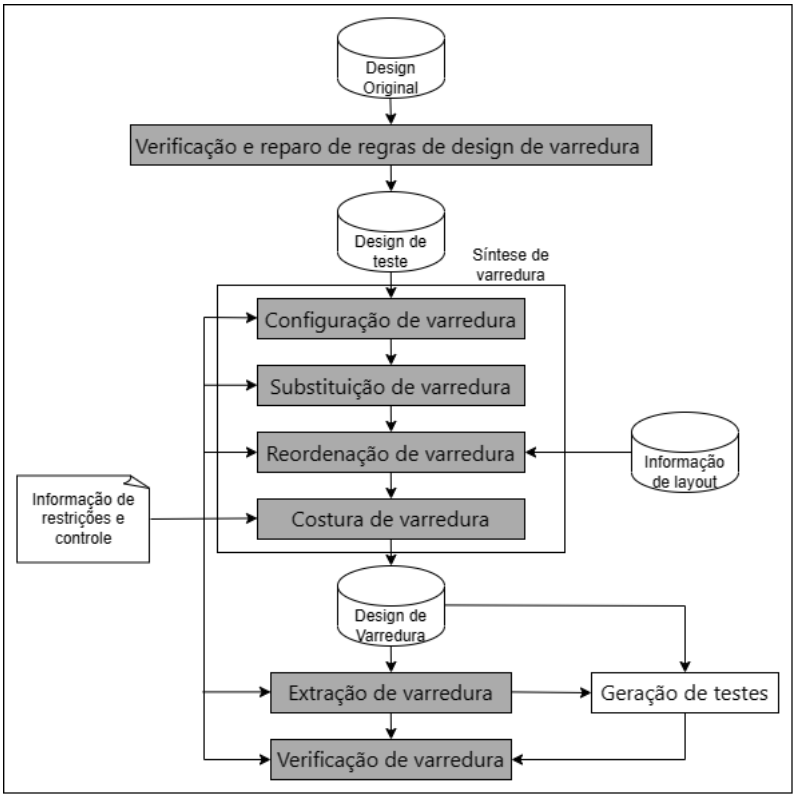

Figura 2.6 – Fluxo de implementação da inserção de varredura, adaptado de Wang, Wu e Wen (2006)

O design pré-síntese, também conhecido como *netlist* passa por uma verificação das restrições implementadas no projeto. A partir do momento que a *netlist* é verificada e as correções são aplicadas, a inserção de varredura é implementada. Após várias verificações realizadas, é feita a extração da arquitetura de varredura final do design para os testes *ATPG*. Por fim, a ferramenta deve implementar as operações de deslocamento e captura na arquitetura de varredura final, confirmando assim, se os resultados previstos correspondem ao comportamento de temporização do circuito.

2.1.4.1 Verificação de restrições

O processo de verificação identifica as violações das restrições instituídas e as corrige. Essa correção permite que o design de varredura funcione corretamente, permitindo que a varredura amplie a capacidade de cobertura de testes. Esta etapa é repetida após o processo de síntese. Uma vez que o processo de verificação de restrições é finalizado e concluído corretamente, podemos concluir que a operação de deslocamento e captura do design de testes irá funcionar corretamente.

2.1.4.2 Síntese de varredura

Após o design de testes é produzido, o fluxo de síntese é iniciado, que pode ser dividido em quatro subetapas:

- Configuração de varredura: A etapa de configuração é a parte onde o projetista espe-

cifica os detalhes da inserção de varredura (Wang; Wu; Wen, 2006). A quantidade de cadeias de varredura inseridas no design está diretamente relacionada a quantidade de pinos de entradas e saídas, que são limitados pelas dimensões do chip (Mannucci, 2018). As células de varredura utilizadas são determinadas pela biblioteca da tecnologia utilizada, geralmente essas bibliotecas possuem vários tipos de células, para abordar projetos com comportamentos variados.

Cada cadeia de varredura depende de um único domínio de *clock*. Se o processo precisar de muitas cadeias de varredura, normalmente elas são divididas em várias cadeias diferentes, então um *latch* de bloqueio é inserido entre as células, deixando o processo independente de desvios de *clocks*(Wang; Wu; Wen, 2006). A figura 2.7 mostra o exemplo de *latch* inserido entre células e o diagrama de tempo dessas células, respectivamente.

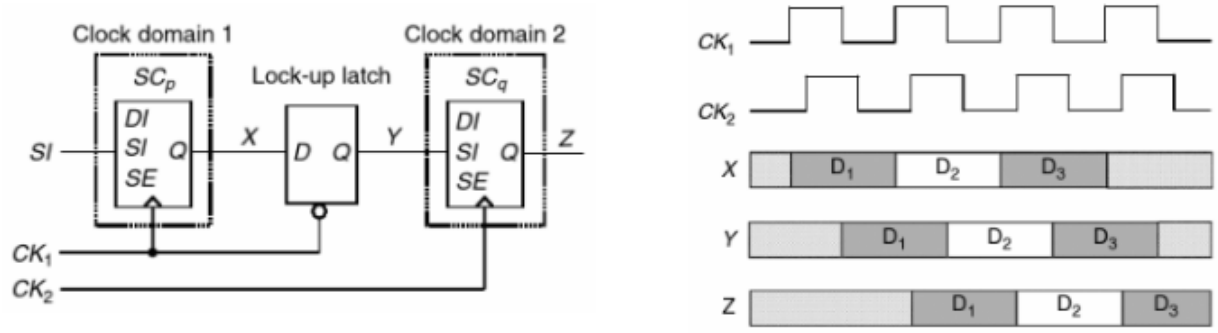

Figura 2.7 – *Latch* de bloqueio inserido entre dois domínios de *clock* e Diagrama de tempo de um *latch* de bloqueio inserido entre dois domínios de *clock*, disponível em Wang, Wu e Wen (2006)

- Substituição de varredura: Nesta etapa os elementos de armazenamento são substituídos por suas respectivas células de varredura. Essas células precisam se assemelhar em termos de velocidade, consumo de energia e dimensões dos elementos originais. Por fim, as entradas de cada célula são conectadas às suas próprias saídas, para evitar pontos flutuantes. Essa configuração é removida na etapa de costura (Wang; Wu; Wen, 2006).
- Reordenação de varredura: Essa etapa tem o objetivo de limitar a extensão dos fios de interconexão. Essa reordenação pode ser realizada em uma única cadeia de varredura ou entre várias separadas.
- Costura de varredura: A última etapa da síntese de varredura é o processo de costura entre as células de varredura para se tornarem cadeias de varreduras. Além de conectar todas as células internas, a etapa de costura conecta a entrada primária de todas as células na primeira célula de varredura, já a última célula da cadeia é conectada à porta de saída primária. A finalização dessa etapa resulta em uma *netlist* final do

design de varredura (Wang; Wu; Wen, 2006).

2.1.4.3 Extração de Varredura

Na etapa de extração, o design original já foi transformado em um design de varredura. A extração é feita para realizar testes e reconhecer a arquitetura de varredura do design (Mannucci, 2018).

2.1.4.4 Verificação de Varredura

Na etapa de verificação, a arquitetura de varredura é validada após a implementação física do design ter sido concluída. Se a costura não for realizada corretamente, a verificação indicará especificamente os erros detectados. Podem ocorrer falhas e erros nas operações de deslocamento, o que pode gerar atrasos entre a saída de uma célula e a entrada de outra célula. Se a violação de temporização entre duas células ocorrer por uma cadeia de varredura com um único domínio de tempo, provavelmente houve uma falha na síntese da árvore de *clock*. Se a violação de temporização ocorrer por uma cadeia de varredura com vários domínios de tempo, indica que pode ter sido uma falha na inserção do *latch* de bloqueio (Wang; Wu; Wen, 2006).

2.1.5 Teste x Verificação

O objetivo do teste funcional é verificar e validar a operação, de acordo com suas especificações, do *DUT* (*Devide Under Test*, Dispositivo Em Teste na tradução livre), já na etapa de verificação funcional assume-se que o chip foi construído corretamente e o objetivo é confirmar que o circuito atende às suas especificações funcionais. O *DFT* verifica se o circuito foi montado corretamente a partir de componentes de baixo nível, seguindo uma lista de conexões estruturais especificada, utilizando um teste de fabricação que compara o comportamento esperado do design em nível de portas com o comportamento do design de silício fabricado. Ou seja, não tem como objetivo de comprovar que o *DUT* funciona corretamente. (Musté, 2021)

2.1.6 Controle e observação

É de extrema importância para a implementação de testes ter o controle e conseguir observar a rede. A partir do momento que é possível observar e controlar um sistema, também é possível interromper o sistema, ler e configurar os seus estados. Em ordem de melhorar a testabilidade de um circuito, é necessário melhorar a observação e o controle do sistema. Para melhorar a observabilidade de um nó, é necessário fornecer meios para acessá-lo e permitir a propagação do sinal para o registrador seguinte, onde um *scan-flop* consegue capturar o valor. Já a controlabilidade de um nó é aprimorada com um hardware adicional que consegue controlar como os valores são sobrepostos no registrador de acionamento. (Musté, 2021)

2.2 *FPGA*

FPGAs (*Field Programmable Gate array*, Arranjo de Portas Programáveis na tradução livre) são circuitos integrados que permitem a criação de circuitos lógicos reconfiguráveis. As *FPGAs* permitem uma grande economia em tempo de desenvolvimento e para custear prototipagens, validações e fabricações, comparando com os *ASICs* e projetos *full-custom*, justamente pela sua capacidade de serem reprogramadas. (ARTHUR, 2021)

Cada fabricante de *FPGAs* possui sua plataforma de desenvolvimento, pois a indústria é extremamente protetiva com os seus *IPs* (*Intelectual Property*, Propriedade Intelectual na tradução livre). Para a *Xilinx*, o *software* utilizado é o ***Vivado***. Os fabricantes disponibilizam kits de desenvolvimento para o auxílio no processo de criação ou de aprendizado de soluções. Na faculdade do Gama da UnB, a placa *Nexys 4* está disponível para os alunos desenvolverem seus projetos.

2.2.1 Arquitetura de uma *FPGA*

De modo geral, existem blocos lógicos, chaves de interconexão, blocos de conexão direta e portas de entradas e saídas em uma *FPGA*. A figura 2.8 apresenta a arquitetura de uma *FPGA*.

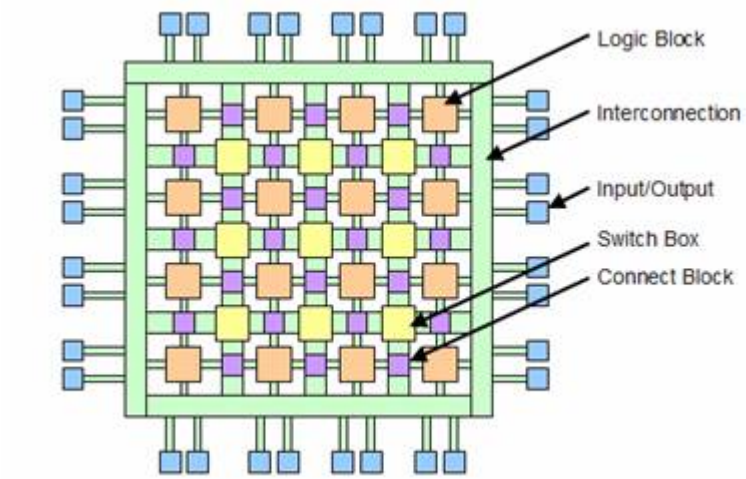

Figura 2.8 – Arquitetura geral de uma *FPGA*, disponível em COLLEGE A.; DOYLE (2023)

Dentro dos blocos lógicos existem *LUTs* (*LookUp Tables*, Tabelas de Consultas na tradução livre), registradores, somadores e multiplexadores. Neste bloco que a lógica reconfigurável é implementada. Existem as chaves de interconexão, que são responsáveis por conectar os diversos blocos lógicos da *FPGA*. Os blocos de conexão direta, onde sua função é ligar os blocos lógicos adjacentes. Além disso, existem as estruturas de *IO* (*Inputs and Outputs*, Entradas e Saídas na tradução livre, blocos de *BRAMs* (*Block Random Access Memory*, Blocos de Memória de Acesso Aleatório na tradução livre) e os blocos *DSPs*.

2.3 *RISC-V*

A *ISA* do *RISC-V* é uma arquitetura de código aberto com licença *BSD* (*Berkeley Software Distribution*, Distribuição de Software Berkeley na tradução livre), ou seja, ela pode ser utilizada para quaisquer fins, tanto soluções em código aberto quanto proprietário (Waterman; Asanović, 2019). Isso permite que grandes fabricantes utilizem esta arquitetura para criar seus produtos sem a preocupação com a propriedade intelectual sobre os métodos de implementação. A licença da *ISA RISC-V* permite que qualquer pessoa possa criar, distribuir ou vender sua implementação sem ter que se preocupar em pagar *royalties*, o que facilita para grupos de pesquisas, amadores ou *startups* com poucos investimentos (ARTHUR, 2021).

2.3.1 Registradores

Os registradores são nomeados de "x0" a "x31", porém eles são divididos em grupos para uma visualização melhor. A tabela 2.2 apresenta que somente alguns registradores salvam o seus conteúdos. O objetivo é acelerar o processo, já que o acesso à memória é demorado, assim, ao classificar alguns registradores como "temporários", permite a implementação de uma função sem a necessidade de armazenar os valores dos registradores na pilha de memória.

Tabela 2.2 – Nomenclatura do Registradores

Registradores	Apelido	Descrição	Salvo
x0	zero	zero fixo	não
x1	ra	Endereço de retorno	não
x2	sp	Ponteiro de pilha	sim
x3	gp	Ponteiro global	não
x4	tp	Ponteiro de *thread*	não
x5	t0	Registrador de *link* alternativo/temporário	não
x6-7	t1-2	Temporários	não
x8	s0/fp	Registrador salvo/ponteiro de quadro	sim
x9	s1	Registrador salvo	sim
x10-11	a0-1	Argumentos de funções/valor de retorno	não
x12-17	a2-7	Argumentos de funções	não
x18-27	s2-11	Registradores salvos	sim
x28-31	t3-6	Temporários	não

2.3.2 Arquitetura *RISC-V*

O módulo base de operação da arquitetura do *RISC-V* é feito com inteiros em qualquer implementação, este é o módulo ***I*** (***Integer***, ou Inteiro na tradução livre), com a possibilidade de ser integrado com 32 registradores de 32, 64 ou 128 *bits*, sendo que 31 registradores são de uso geral e um para o valor ZERO. O módulo ***E*** (***Embedded***, ou Integrado na tradução livre) serve para aplicações embarcadas, com 16 registradores (15 de uso geral e 1 como

ZERO) e é limitado em 32 *bits*. Estes são os módulos bases para o *RISC-V*, outras extensões são opcionais, sendo possível ter os seguintes conjuntos: *RV32I*, *RV32E*, *RV64I* e *RV128I*. A tabela 2.3 mostra as extensões padrões da arquitetura.(Waterman; Asanović, 2019).

Essa flexibilidade onde é possível incluir as extensões desejadas, faz com que o *RISC-V* se adeque em aplicações desde pequenos microcontroladores a grandes supercomputadores. Uma boa equipe de projetistas pode adicionar implementações para baixa potência, desempenho, segurança, etc.

Tabela 2.3 – Extensões para a arquitetura *RISC-V*

Extensão	Descrição
M	Contém instruções para multiplicação e divisão de números inteiros.
A	Contém instruções que leem, modificam e escrevem atomicamente na memória.
F	Extensão para aritmética de números em ponto flutuante de precisão simples conforme padrão IEEE 754-200.
D	Extensão para aritmética de números em ponto flutuante de precisão dupla conforme padrão IEE 754-200.
Q	Extensão para aritmética de números em ponto flutuante de precisão quádrupla conforme padrão IEEE 754-2008.
Counters	Define os contadores de hardware da arquitetura.
Zicrs	Contém instruções para manipulação de *CSR* (*Register State Controller*, Registradores de Controle de Estado na tradução livre).
Zifencei	Contém instruções para sincronização de escritas e buscas à memória de instruções da mesma *hardware thread*.
Zam	Expande a extensão A, permitindo operações atômicas desalinhadas.
Ztso	Altera o modelo de consistência de memória para uma versão mais restrita chamada Total *Store Ordering*.

2.3.2.1 Instruções da arquitetura *RISC-V*

As instruções do *RISC-V* são subdivididas em campos (Calçada, 2020), a tabela 2.4 descreve estes campos. A tabela 2.5 descreve os 6 formatos de instrução e a tabela 2.6 apresenta estes formatos. Os tipos **B** e **J** são variantes dos tipos **S** e **U**, é modificado apenas a posição dos bits que formam o imediato. Já os campos **opcode**, **rs1**, **rs2**, **rd**, **funct3** e **funct7** são fixos. O *RISC-V* conta com três modos simples de endereçamento:

1. base-deslocamento: se baseia nas instruções *load/store*, onde um endereço base é somado a um valor imediato que forma o endereço que será acessado.
2. Absoluto: utiliza o valor contido em um registrador como um endereço.
3. PC-relativo: utiliza o valor do contador de programa e soma ao valor imediato.

Tabela 2.4 – Campos das instruções do *RISC-V*

Campo	Nome	Descrição
rs1	*Source Register 1*	Contém o endereço do primeiro operando da operação.
rs2	*Source Register 2*	Contém o endereço do segundo operando da operação.
rd	*Destination Register*	Contém o endereço do registrador destino.
funct3	*3-bit function*	Um campo de 3-*bits* que armazena os códigos de operação para instruções lógicas e aritméticas. Pode ser usado também para codificar o tamanho da palavra das instruções *load/store*.
funct7	*7-bit function*	Usado apenas com instruções lógicas e aritméticas, contém um código de operação adicional para o controle da *ALU* (*Arithmetic Logic Unit*, Unidade Lógica e Aritmética na tradução livre).
opcode	*Operation code*	Contém o código da operação da instrução

Tabela 2.5 – Descrição dos formatos de instrução do *RISC-V*

Formato	Nome	Descrição
R	*Register*	Este formato é destinado para o tipo de instrução que utiliza dois registradores (rs1 e rs2) na operação. O resultado desta operação é armazenado em rd.
I	*Immediate*	Este formato é destinado para o tipo de instrução que opera entre um registrador (rs1) e um valor de imediato de 12 *bits* (imm). O resultado desta operação é armazenado em rd.
S	*Store*	O resultado da soma entre o o valor de rs1 e o valor do imediato imm é armazenado em rs2.
B	*Branch*	É feita a comparação entre rs1 e rs2. Se a comparação for positiva, o contador de programa é incrementado ou decrementado de acordo com o valor de imm.
U	*Upper immediate*	É utilizado somente por instruções que utilizam o imediato de 20 *bits*.
J	*Jump*	O valor de imm é somado ao valor do contador de programa.

Tabela 2.6 – Formato das instruções do *RISC-V*

Formato	**Bits**					
	31 - 25	24 - 20	19 - 15	14 - 12	11 - 7	6 - 0
R	funct7	rs2	rs1	funct3	rd	opcode
I	imm[11:0]		rs1	funct3	rd	opcode
S	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
B	imm[20 | 10:5]	rs2	rs1	funct3	imm[4:1 | 11]	opcode
U	imm[31:12]				rd	opcode
J	imm[20 | 10:1 | 11 | 19:12]				rd	opcode

2.3.3 *PicoRV32*

O *PicoRV32* foi o núcleo escolhido para a implementação do *DFT*, o núcleo em questão implementa as instruções do *RISC-V* de 32 bits com as extensões ***I***, ***M*** e ***C***, formando o *RV32IMC*, mais especificamente o *PicoRV32*, que está em código aberto e disponível no repositório em (PicoRV32…, 2023), a linguagem de descrição de *hardware* (*HDL*) em questão é o *verilog*.

A figura 2.9 apresenta o design que será implementado na seção 3, onde *PicoSOC* representa o *top level* (bloco principal, responsável por instanciar e unir todos os blocos), *SRAM* representa a memória do núcleo, *UART* (*Universal Asynchronous Receiver/Transmitter*, Transmissor/Receptor Assíncrono Universal na tradução livre) representa os receptores e transmissores das instruções, *SPI flash* é o controlador da memória e, por fim, *PicoRV32* é o núcleo que contém as instruções do *RISC-V*. Os desenvolvedores do núcleo deixaram um espaço de desenvolvimento para adicionar blocos.

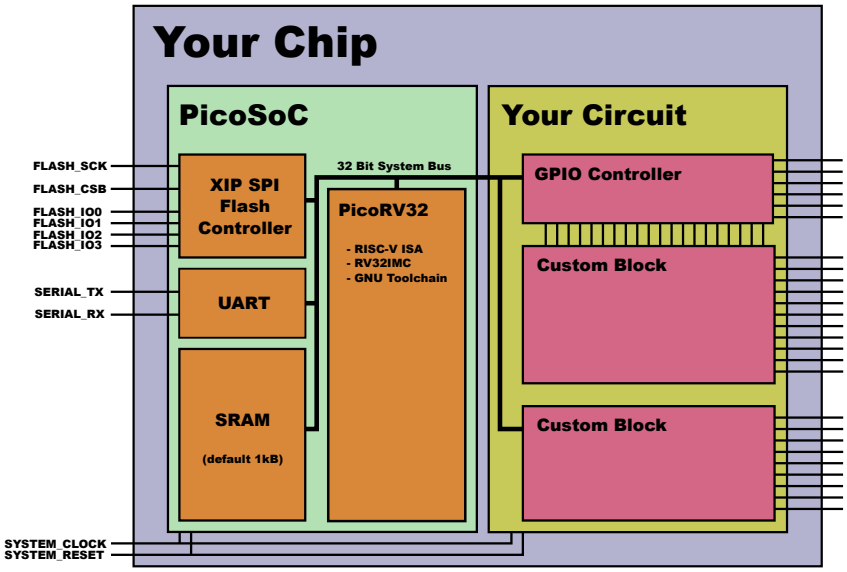

Figura 2.9 – Blocos do *PicoRV32*, disponível em (PicoRV32…, 2023)

2.3.3.1 Máquina de estados

Máquina de estados é um modelo matemático usado para descrever o comportamento de um algoritmo ou circuito lógico. A figura 2.10 apresenta a máquina de estados do *PicoRV32*. Os estados implementados são maiores que os apresentados, porém, somente as mudanças de estados importantes foram apresentadas.

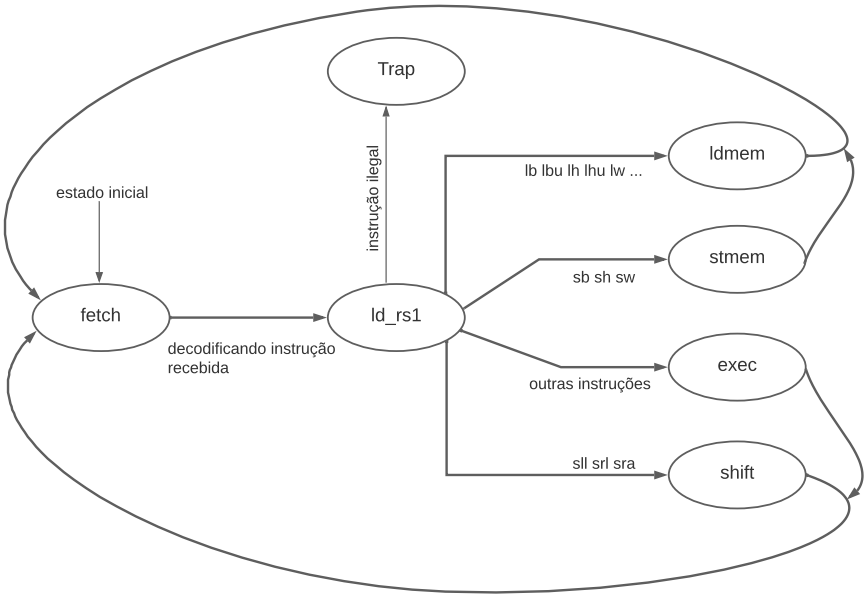

Figura 2.10 – Máquina de Estados do *PicoRV32*

- *fetch*: É o estado inicial do processador. Nesse estado a função é encontrar a próxima

instrução da memória. Assim que a instrução é lida e decodificada, assim a transição para o próximo estado é realizada.

- *ld_rs1*: Neste estado é realizada a leitura do registrador. Uma vez que o valor do registrador é obtido, a máquina avança para o próximo estado, à depender do tipo de instrução decodificada.
- *ldmme*: Neste estado são dadas instruções para carregar a memória.
- *stmem*: Neste estado são realizadas instruções para armazenar a memória.
- *exec*: As operações da *ALU* são realizadas nesse estado. A coleta dos resultados é feita e armazenada em outro registrador.
- *shift*: Este estado é utilizado para reduzir os caminhos críticos do circuito.

2.4 Estado da Arte

O trabalho em questão foi baseado no estudo de (Musté, 2021), a tabela 2.7 apresenta os dados de acréscimo de área e potência causadas pela *DFT* após a implementação física do circuito.

Tabela 2.7 – Dados de área e potência de acordo com (Musté, 2021)

Recursos	**Sem *DFT***	**Com *DFT***
Área (μm^2)	2536060,893	2745630,006
Potência (W)	0,208163	0,216930

3 Validação do *PicoRV32* em *Hardware*

Para a validação do *PicoRV32* em *hardware*, foi necessária a utilização da ferramenta ***Vivado***, na qual é possível realizar a síntese para um determinado chip *FPGA*. A versão do PicoRV32 a ser implementada está descrita na figura 2.9, onde a interface com os blocos customizados é feita diretamente com a memória. A validação será realizada nas seguintes etapas:

- Simular o funcionamento das instruções no *PicoRV32*;
- Simular a rede neural *MLP443*
- Elaborar um *top level* para instanciar o PicoRV32 e a MLP443
- Síntese e teste do *top level* na placa *Nexys4*.

A placa escolhida para implementação foi a *Nexys 4*, uma plataforma de desenvolvimento de circuitos digitais completa, baseada na *FPGA Artix*-7^T *M* da *Xilinx*. A placa pode suportar projetos que vão desde circuitos combinacionais introdutórios até poderosos processadores embarcados. A placa tem a capacidade de 15850 blocos lógicos (cada um com 4 *LookUp Tables* (*LUT*) de 6 entradas e 8 *flip-flops*), 4860K*bits* de RAM, 6 blocos de gerenciamento de clock (cada um com um circuito *Phase-Locked Loop* (*PLL*)), 240 blocos de *DSP*, dentre outros periféricos. Portanto, a placa é capaz de suportar o projeto.

A figura 3.1 apresenta a plataforma utilizada para a validação da arquitetura expandida do PicoRV32.

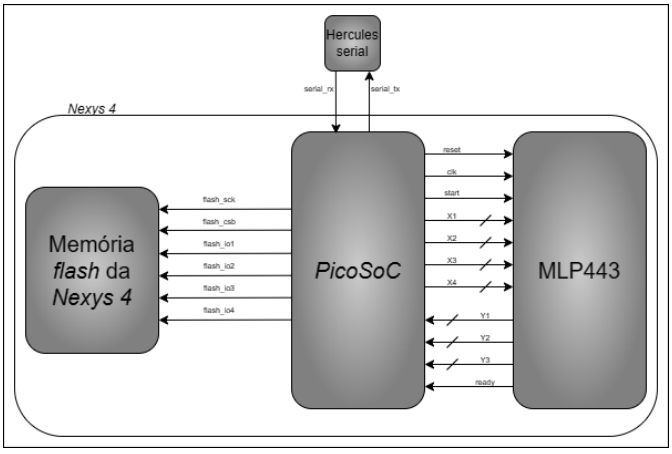

Figura 3.1 – Diagrama de blocos proposto

3.1 Implementação do *PicoRV32* e da *MLP443*

Para a implementação da plataforma, foi necessário realizar a instanciação do arquivo arquivo fonte *VHDL* no *topmodule* do *PICORV32*. Foram necessárias também alterações nos

dados de entrada da rede neural para torná-los compatíveis com o código do *PICORV32*, que está em *verilog*.

Os blocos foram implementados de maneira hierárquica, conforme mostram os itens a seguir:

- icebreaker.v (anexo B): representa um exemplo de implementação baseada na FPGA icebreaker. Foi adaptado como *top level* para instanciar todos blocos.
- picosoc.v (apêndice D): como na figura 2.9, é onde foram instanciadas as memórias, a interface *UART* e o *PicoRV32*.
- picorv32.v (anexo C): representa a versão *RISCV32IMC*. Contém as máquinas de estados, decodificadores de instruções, etc.
- spimemio.v (anexo D): interface SPI responsável por se comunicar com a memória flash externa.
- simpleuart.v (anexo E): interface que conecta as linhas de transmissão e recepção com o PC.

O esquemático *RTL* pode ser observado nas figuras 3.2, 3.3 e 3.4, a figura 3.5 mostra o bloco *picosoc* expandido, a figura 3.6 apresenta os blocos *uart* e *neuronio*, a figura 3.7 mostra o bloco *memory* e, por fim, a figura 3.8 apresenta os blocos *spimemio* e *picorv32*, todos instanciados dentro do bloco *picosoc*.

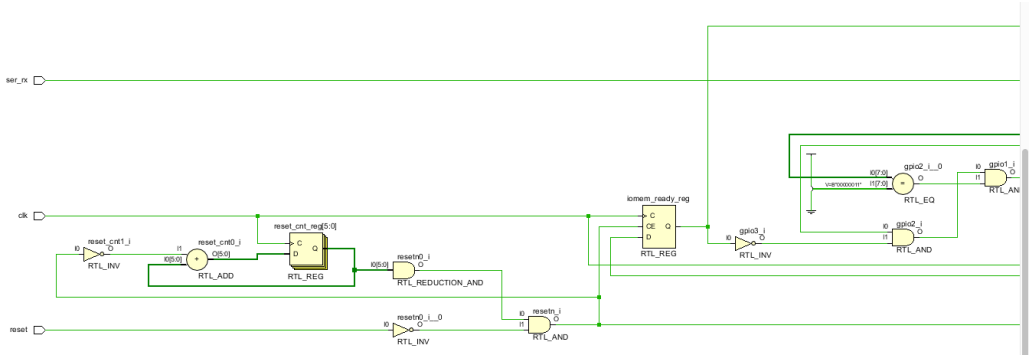

Figura 3.2 – Esquemático *RTL* parte 1

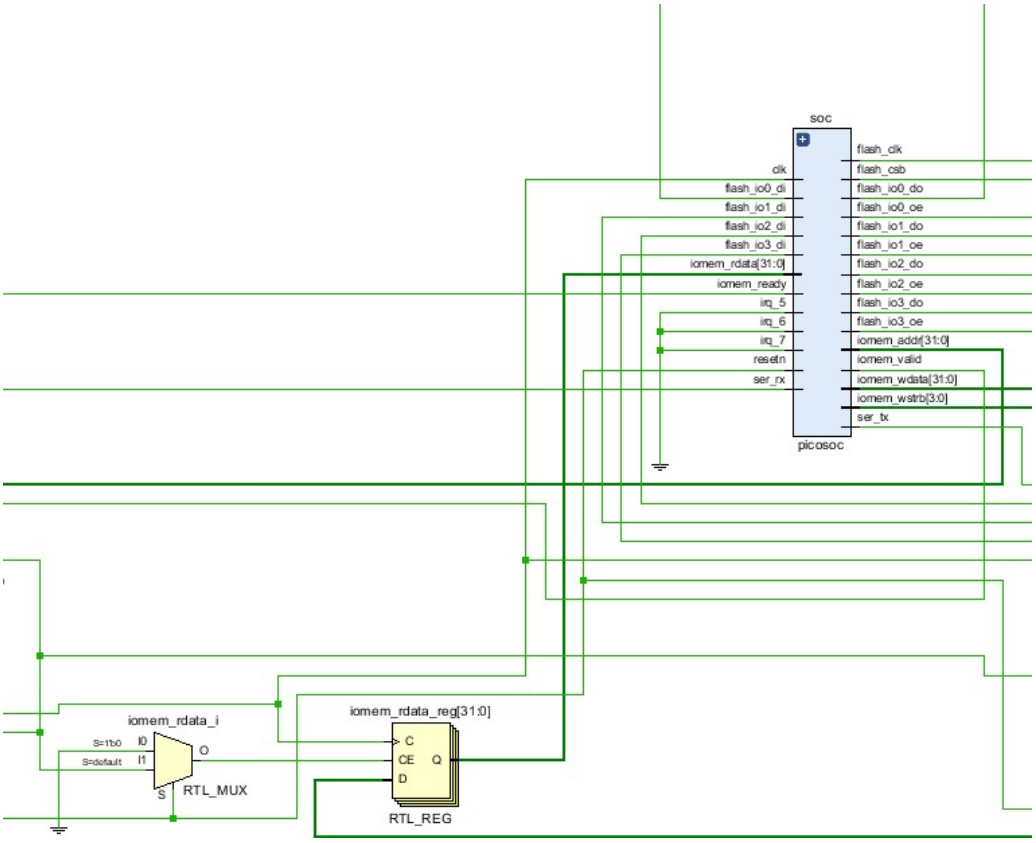

Figura 3.3 – Esquemático *RTL* parte 2

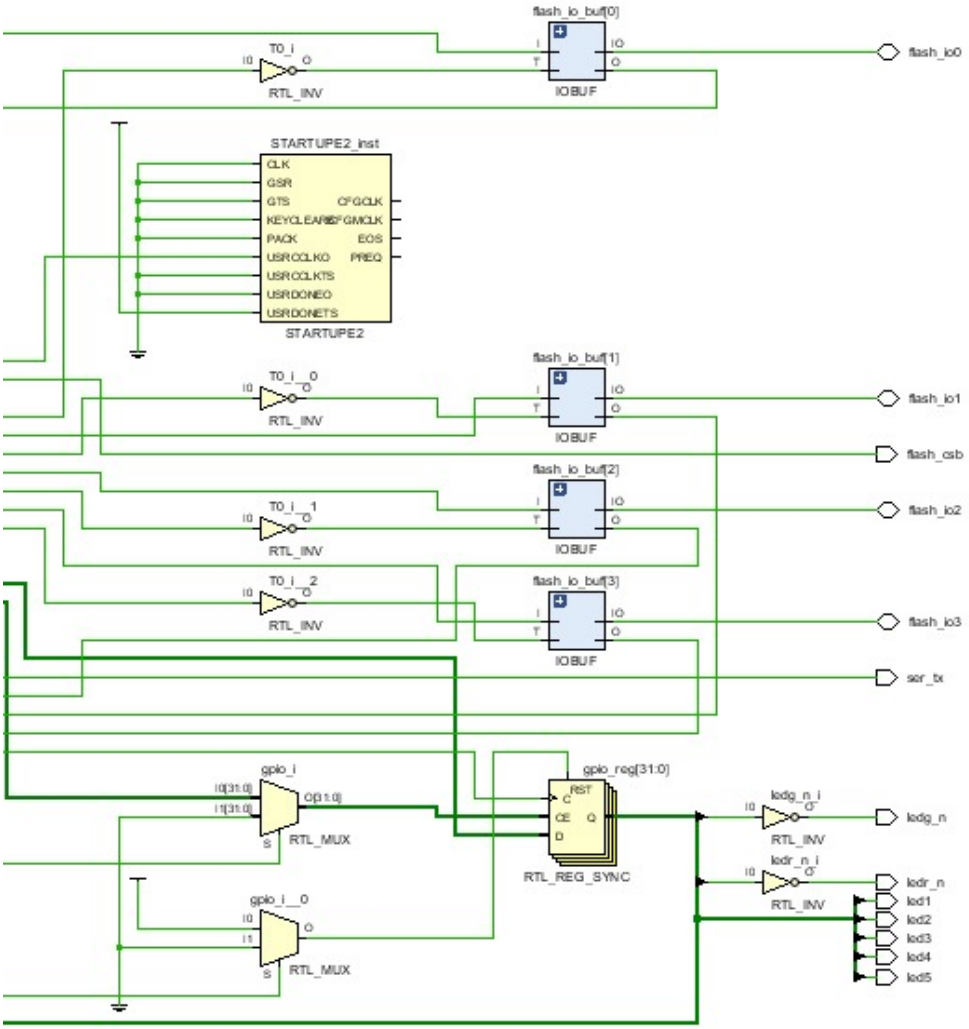

Figura 3.4 – Esquemático *RTL* parte 3

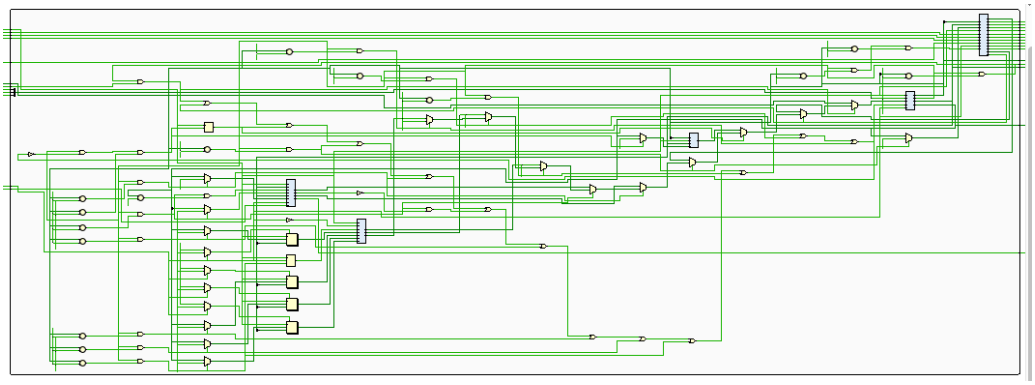

Figura 3.5 – Módulo *picosoc* expandido

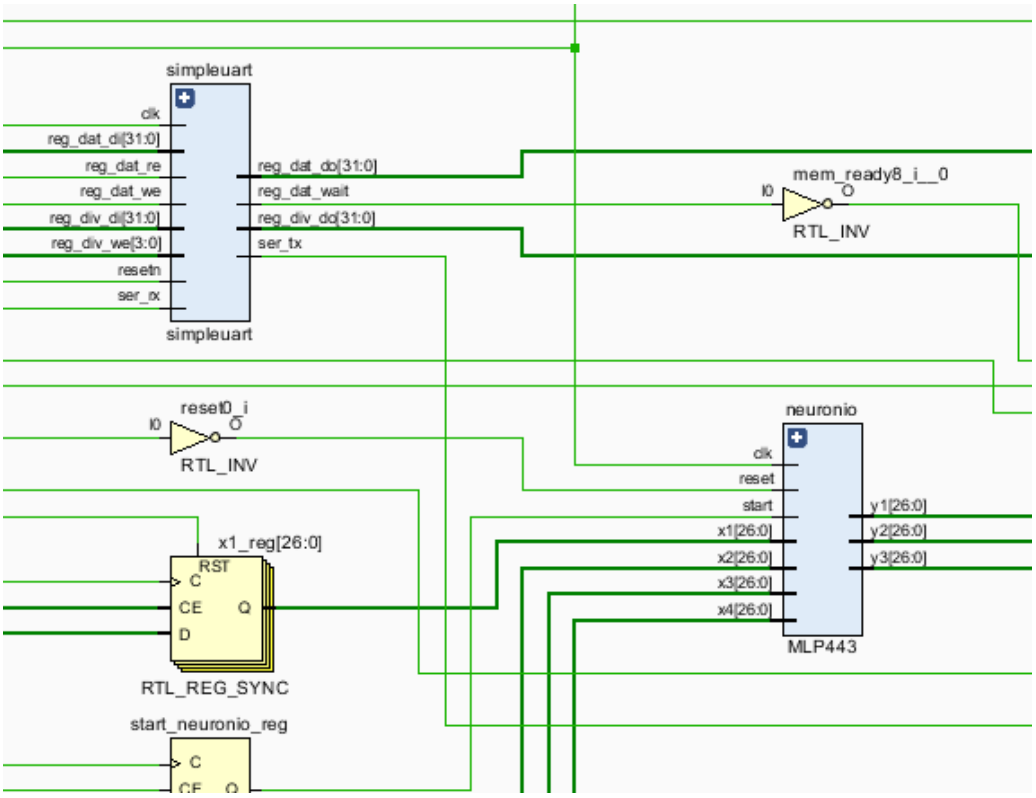

Figura 3.6 – Blocos *uart* e neuronio

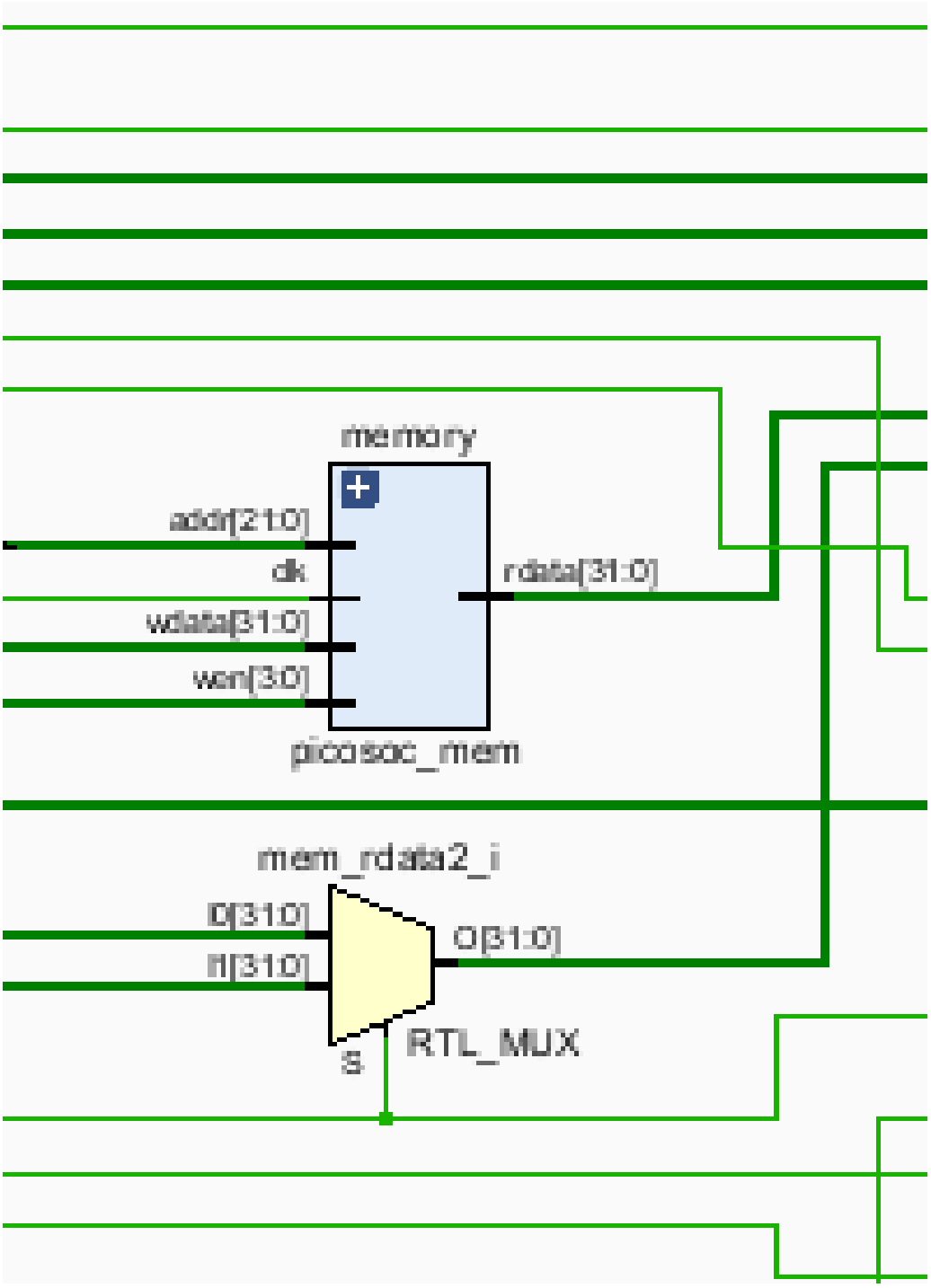

Figura 3.7 – Bloco *memory*

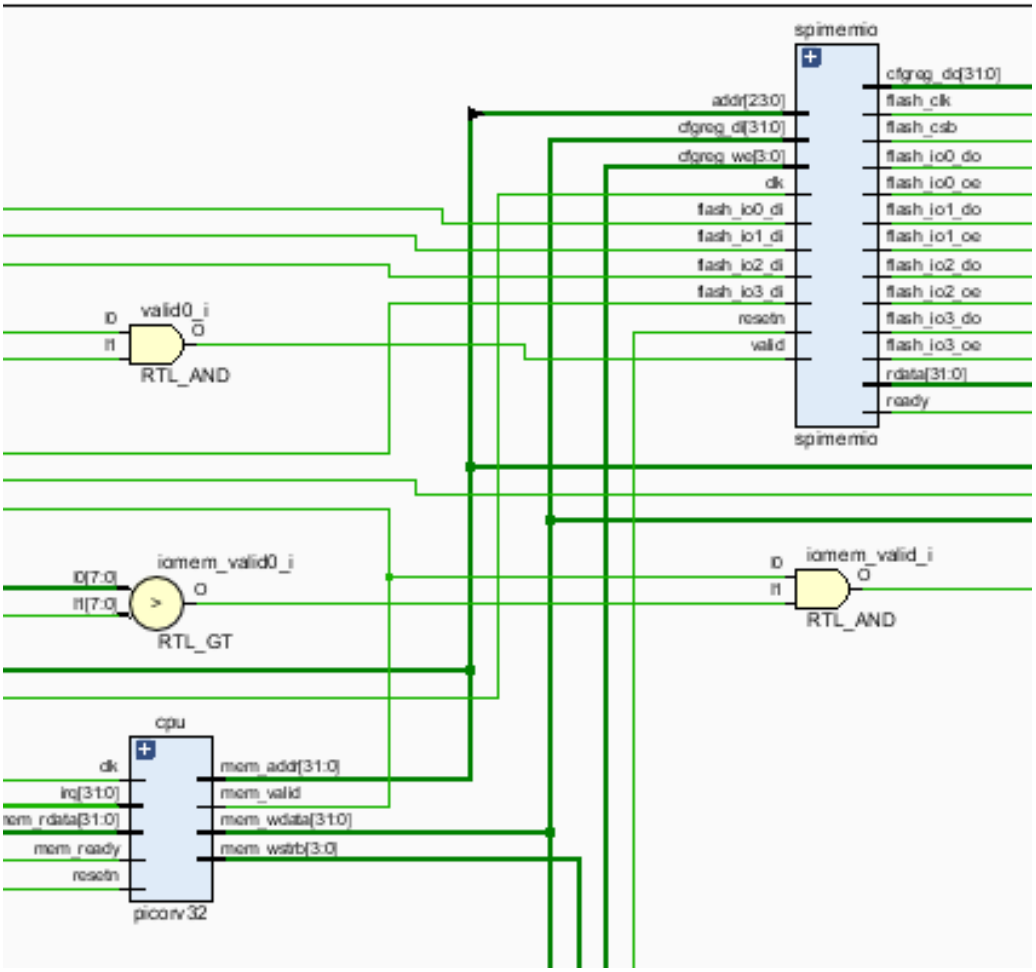

Figura 3.8 – Bloco *spimemio* e *picorv32*

A rede neural utilizada é a *MLP* com quatro neurônios de entrada, quatro na camada escondida e três de saída. Ela foi implementada utilizando *IP-cores* de soma e multiplicação em ponto flutuante, disponibilizados em VHDL pelo Professor Doutor Daniel Mauricio Muñoz Arboleda na disciplina Projeto com Circuitos Reconfiguráveis durante a graduação.

3.2 Integração dos módulos

Para a integração dos módulos, primeiro é necessário adaptar o *firmware* criado em linguagem "C", disponível no apêndice E, que posteriormente será convertido em um arquivo *.bin* (utilizado para configuração da memória flash) e *.hex* (utilizado para simulação). O algoritmo 1 explicitado a seguir é responsável por definir os endereços das entradas e saídas da rede neural na memória.

Algoritmo 1 – Definição de entradas e saídas

```
1  #define reg_spictrl (*(volatile uint32_t*)0x02000000)
2  #define reg_uart_clkdiv (*(volatile uint32_t*)0x02000004)
3  #define reg_uart_data (*(volatile uint32_t*)0x02000008)
```

```
4   #define reg_leds (*(volatile uint32_t*)0x03000000)
5   #define x1 (*(volatile uint32_t*)0x0200000C)
6   #define x2 (*(volatile uint32_t*)0x02000010)
7   #define x3 (*(volatile uint32_t*)0x02000014)
8   #define x4 (*(volatile uint32_t*)0x02000018)
9   #define y1 (*(volatile uint32_t*)0x0200001C)
10  #define y2 (*(volatile uint32_t*)0x02000020)
11  #define y3 (*(volatile uint32_t*)0x02000024)
```

Na função *main* (2), são utilizadas duas funções para exibir a saída no leitor serial do programa Hercules, a função padrão *print()* e a função criada para converter inteiro para hexadecimal, chamada *print_hex()*. Com essas funções é possível observar diretamente o funcionamento da rede neural implementada em *hardware*.

Algoritmo 2 – Função principal

```
1   void main()
2   {
3     reg_leds = 31;
4     reg_uart_clkdiv = 868;
5     x1 = 0b01000000101100110011001;
6     x2 = 0b01000000010000000000000;
7     x3 = 0b01000000011110011001100;
8     x4 = 0b00111111100011001100110;
9
10    int y1_aux;
11    int y2_aux;
12    int y3_aux;
13
14    y1_aux = y1;
15    y2_aux = y2;
16    y3_aux = y3;
17
18    print ("y1= ");
19    print_hex (y1_aux,8);
20    print("\n");
21
22    print ("y2 = ");
23    print_hex (y2_aux,8);
24    print("\n");
25
26    print ("y3 = ");
27    print_hex (y3_aux,8);
28    print("\n");
29
30  }
```

Após a análise do *firmware*, as instanciações são feitas. A primeira modificação é realizada no módulo do *MLP443* (apêndice F), pois as entradas e saídas estão configuradas para receber um tipo de *array* especificado em uma biblioteca em *VHDL*. Já o módulo

picosoc está em *verilog*, que não reconhece este tipo de *array*. A adaptação realizada pode ser observada a seguir. A entrada do tipo *array1D_in* e a saída do tipo *array1D_h* foram substituídas por 4 entradas e 3 saídas do tipo *std_logic_vector*, que o módulo em *verilog* consegue reconhecer. Em seguida é realizada a instanciação do módulo *MLP443* no módulo *picosoc*. Os algoritmos 3 e 4 mostram a instanciação de entradas e saídas do módulo *MLP443* e em seguida, a instanciação deste módulo no *picosoc*, respectivamente.

Algoritmo 3 – Entradas e saídas do módulo *MLP443*

```
1  entity MLP443 is
2  Port (reset : in STD_LOGIC;
3  clk : in STD_LOGIC;
4  start : in STD_LOGIC;
5  x1 : in std_logic_vector(26 downto 0);
6  x2 : in std_logic_vector(26 downto 0);
7  x3 : in std_logic_vector(26 downto 0);
8  x4 : in std_logic_vector(26 downto 0);
9  y1 : out std_logic_vector(26 downto 0);
10 y2 : out std_logic_vector(26 downto 0);
11 y3 : out std_logic_vector(26 downto 0);
12 ready : out STD_LOGIC);
13 end MLP443;
```

Algoritmo 4 – Instanciação do módulo *MLP443* no *picosoc*

```
1  MLP443 neuronio(
2  .reset   (~resetn),
3  .clk     (clk),
4  .start   (start_neuronio),
5  .x1      (x1),
6  .x2      (x2),
7  .x3      (x3),
8  .x4      (x4),
9  .y1      (y1),
10 .y2      (y2),
11 .y3      (y3),
12 .ready   (ready_neuronio)
13 );
```

A segunda modificação foi a lógica para entregar os valores definidos no segundo algoritmo apresentado para os registradores definidos no primeiro algoritmo. O algoritmo 5 a seguir mostra como é realizada a definição dos registradores de entrada do módulo *MLP443*. É possível observar que cada entrada é definida somente quando o seu endereço, definido no primeiro algoritmo, é válido. Assim, a entrada serial entrega de 8 em 8 *bits*.

Algoritmo 5 – Definição dos registradores de entrada do módulo *MLP443*

```
1  always @(posedge clk) begin
2
```

```
3   if (~resetn) begin
4   x1<=0;
5   x2<=0;
6   x3<=0;
7   x4<=0;
8   end else begin
9   start_neuronio = 0;
10  if (mem_valid && mem_addr == 32'h 0200_000C) begin
11  if (mem_wstrb[0]) x1[ 7: 0] <= mem_wdata[ 7: 0];
12  if (mem_wstrb[1]) x1[15: 8] <= mem_wdata[15: 8];
13  if (mem_wstrb[2]) x1[23:16] <= mem_wdata[23:16];
14  if (mem_wstrb[3]) x1[26:24] <= mem_wdata[26:24];
15  end
16
17  if (mem_valid && mem_addr == 32'h 0200_0010) begin
18  if (mem_wstrb[0]) x2[ 7: 0] <= mem_wdata[ 7: 0];
19  if (mem_wstrb[1]) x2[15: 8] <= mem_wdata[15: 8];
20  if (mem_wstrb[2]) x2[23:16] <= mem_wdata[23:16];
21  if (mem_wstrb[3]) x2[26:24] <= mem_wdata[26:24];
22  end
23
24  if (mem_valid && mem_addr == 32'h 0200_0014) begin
25  if (mem_wstrb[0]) x3[ 7: 0] <= mem_wdata[ 7: 0];
26  if (mem_wstrb[1]) x3[15: 8] <= mem_wdata[15: 8];
27  if (mem_wstrb[2]) x3[23:16] <= mem_wdata[23:16];
28  if (mem_wstrb[3]) x3[26:24] <= mem_wdata[26:24];
29  end
30
31  if (mem_valid && mem_addr == 32'h 0200_0018) begin
32  if (mem_wstrb[0]) x4[ 7: 0] <= mem_wdata[ 7: 0];
33  if (mem_wstrb[1]) x4[15: 8] <= mem_wdata[15: 8];
34  if (mem_wstrb[2]) x4[23:16] <= mem_wdata[23:16];
35  if (mem_wstrb[3]) x4[26:24] <= mem_wdata[26:24];
36  start_neuronio = 1;
37  end
38  end
39  end
```

A última modificação realizada foi a validação de cada registrador, pois a próxima instrução só será inicializada após um sinal de validação. A modificação realizada pode ser observada no algoritmo 6.

Algoritmo 6 – Validação das instruções

```
1   wire saida_y1 = mem_valid && (mem_addr == 32'h 0200_001C);
2   wire saida_y2 = mem_valid && (mem_addr == 32'h 0200_0020);
3   wire saida_y3 = mem_valid && (mem_addr == 32'h 0200_0024);
4
5   wire x1_valid = mem_valid && (mem_addr == 32'h 0200_000C);
6   wire x2_valid = mem_valid && (mem_addr == 32'h 0200_0010);
7   wire x3_valid = mem_valid && (mem_addr == 32'h 0200_0014);
8   wire x4_valid = mem_valid && (mem_addr == 32'h 0200_0018);
```

```
9
10  assign mem_ready = (iomem_valid && iomem_ready) || spimem_ready ||
        ram_ready || spimemio_cfgreg_sel || simpleuart_reg_div_sel ||
        (simpleuart_reg_dat_sel && !simpleuart_reg_dat_wait) || x1_valid ||
        x2_valid || x3_valid || x4_valid || saida_y1 || saida_y2 ||
        saida_y3;
11
12  assign mem_rdata = (iomem_valid && iomem_ready) ? iomem_rdata :
        spimem_ready ? spimem_rdata : ram_ready ? ram_rdata :
        spimemio_cfgreg_sel ? spimemio_cfgreg_do : simpleuart_reg_div_sel ?
        simpleuart_reg_div_do : simpleuart_reg_dat_sel ?
        simpleuart_reg_dat_do : saida_y1 ? {5'h0,y1}: saida_y2 ? {5'h0,y2}:
        saida_y3 ? {5'h0,y3}: 32'h 0000_0000;
```

4 Síntese Lógica

Após a validação do projeto *RTL*, o processo de síntese será realizado apenas para o núcleo PicoRV32. Nesta etapa, juntamente com as opções de *DFT*, será obtida uma *netlist*, mapeando a descrição em *RTL* em nível de portas lógicas utilizando bibliotecas de células padrão de uma determinada tecnologia. Neste caso, foi escolhida a tecnologia *CMOS TSMC* 180nm, juntamente com as restrições de temporização do projeto. Os resultados gerados serão utilizados em simulações com foco em temporização e também na etapa de síntese física, onde são realizados o posicionamento e roteamento das células. É importante ressaltar que, apesar da validação em *hardware* descrita na seção 3 ter sido realizada com o *PicoRV32* sem encapsulamento *axi*, para os testes nas ferramentas ***Cadence***, foi utilizado o arquivo *picoRV32_wrapper*, disponível no anexo I, que contém suporte para o barramento *axi*, mas não foi utlizado.

4.1 Solução de síntese *Genus*

Genus é uma ferramenta de nova geração da ***Cadence*** que realiza a síntese do projeto *RTL*. Ela permite a implementação de estruturas *DFT* sem comprometer o desempenho, consumo de energia, tempo de execução, área ou precisão. Para funcionar corretamente, a ferramenta precisa dos itens específicos listados a seguir:

- Arquivos da biblioteca no formato *liberty* (*.lib*)
- Arquivos da tecnologia no formato *library exchange format* (*.lef*)
- Arquivos *HDL*
- Arquivos de restrições *synopsys design constraints* (*.sdc*)
- *Scripts* de execução (*.tcl*)
- Executável e licença do ***Genus***

Arquivos opcionais:

- Arquivos de tecnologia *QRC*
- Arquivo de tabela de capacitância (*.captbl*)
- Arquivo de formato *design exchange format* (*.def*) para o mapeamento.
- Arquivos de comutação: *.saif*, *.tcf*, *.vcd*
- Arquivo de potência (*.cpf*)

A figura 4.1 mostra o processo geral de síntese da ferramenta ***Cadence Genus***, que se baseia em três passos:

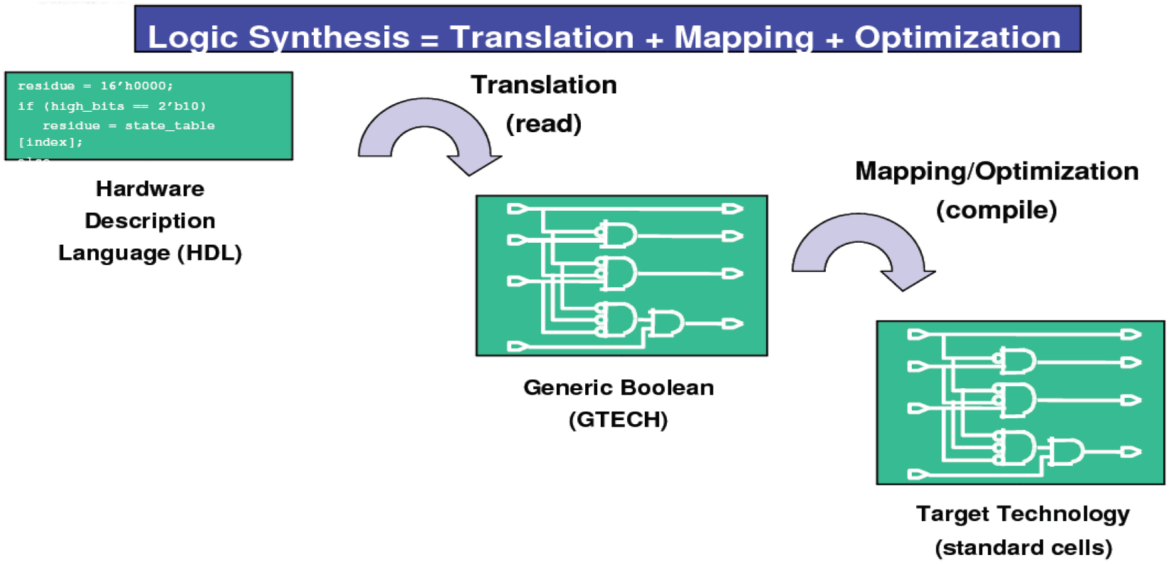

Figura 4.1 – Processo de síntese, disponível em Cadence (2023b)

1. Tradução: Neste passo, a ferramenta realiza a leitura da a linguagem de descrição de *hardware*.
2. Mapeamento: Em seguida, a ferramenta realiza a tradução do código para um circuito lógico baseado em uma biblioteca específica.
3. Otimização: Por último, a ferramenta otimiza o projeto de acordo com as restrições de atraso, área e potência.

4.2 Fluxo de síntese

O fluxo de síntese da ferramenta ***Genus*** pode ser observado na figura 4.2. A implementação será realizada na ordem apresentada, começando com a leitura das bibliotecas-alvo e terminando com a execução da otimização incremental. Cada uma das etapas é brevemente descrita a seguir. O *script* completo sem *DFT* está disponível no apêndice A, enquanto a versão com *DFT* está no apêndice B.

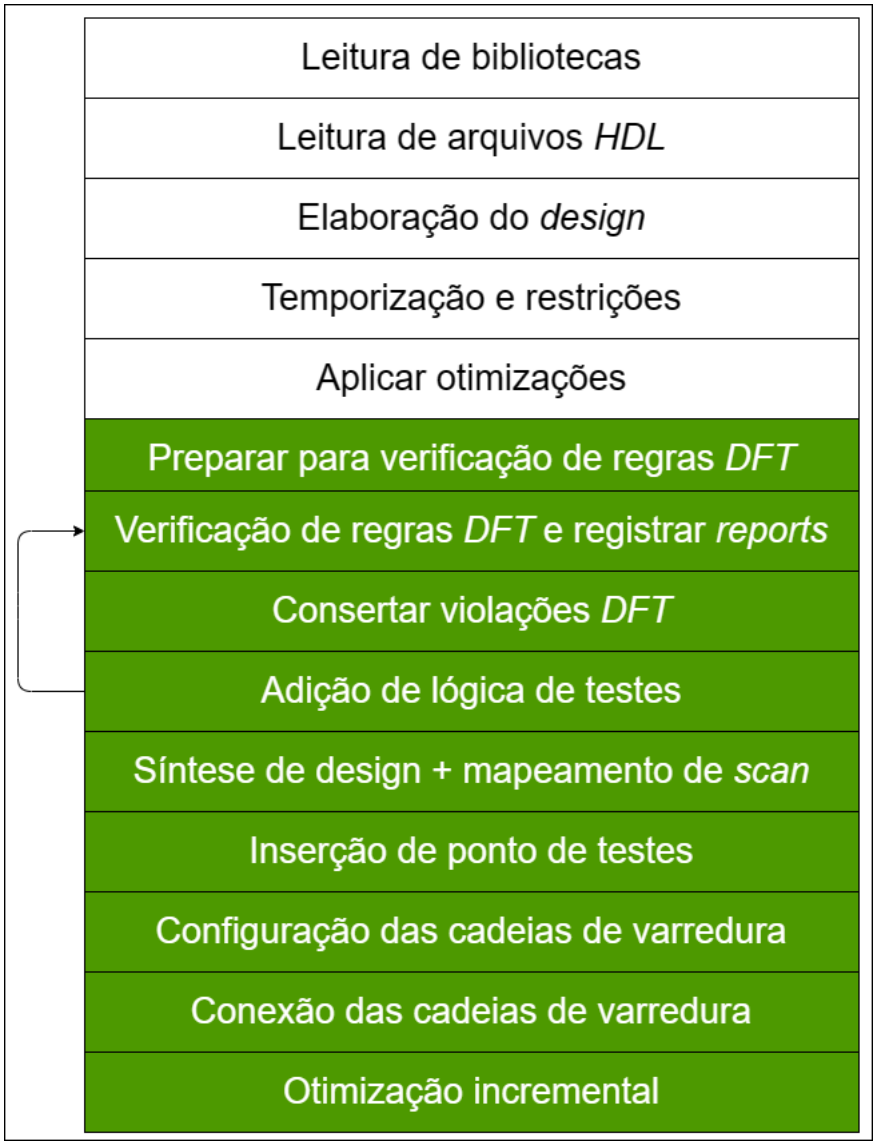

Figura 4.2 – Fluxo de síntese, adaptado de Mannucci (2018)

As duas opções abaixo podem ser usadas para abrir a ferramenta de síntese, sendo a segunda a mais recomendada:

```
1    Genus
2    Genus -leagcy_ui
```

A ferramenta pode gerar modelos de *scripts* de execução em *tcl* para a síntese que garantem que todas as configurações de variáveis e atributos estejam adequados, bem como o fluxo da síntese. O comando a seguir serve como modelo. É importante ressaltar que opção "$-dft$" permite gerar os comandos *DFT* no *script*, que será gerado no mesmo diretório que o ***Genus*** foi aberto e pode ser observado no trecho a seguir:

```
1    legacy_genus: /> write_template -dft -outfile <sintese_dft.tcl>
```

A etapas do fluxo são detalhadas a seguir.

- Configuração inicial: indica o nome do arquivo principal, o nome das pastas de destino e o esforço da ferramenta para certos comando.

```
1     set DESIGN "picorv32_wrapper"
2     set GEN_EFF medium
3     set MAP_OPT_EFF high
4     set _OUTPUTS_PATH outputs
5     set _REPORTS_PATH reports
6     set _LOG_PATH logs
```

- Leitura de bibliotecas: Nesta etapa, os caminhos para os arquivos *verilog*, de biblioteca e variáveis são definidos. A biblioteca de destino é a parte mais importante, pois sem ela a síntese não pode ser realizada.

```
1     set_db init_lib_search_path
2     read_libs
```

  O primeiro comando indica o local onde a biblioteca pode ser encontrada, o segundo especifica o nome exato do arquivo de biblioteca (*.lib*) a ser utilizado.

- Leitura de arquivos de projeto: Após a definição do caminho das bibliotecas, o próximo passo é indicar o caminho dos arquivos de projeto. O primeiro comando indica o caminho onde os arquivos estão localizados e o segundo indica todos os arquivos usados no projeto já o terceiro comando realiza a leitura dos arquivos.

```
1     set_db init_hdl_search_path
2     set rtl_files
3     read_hdl
```

- Elaboração do projeto: O arquivo principal, mais conhecido como "*top_design*", é o que instancia e realiza a comunicação entre os outros blocos. Ao realizar a elaboração deste arquivo, o projeto global é criado. Um exemplo pode ser observado a seguir:

```
1     elaborate ${DESIGN}
```

- Temporização, restrições e otimizações: Nesta etapa, a ferramenta ***Genus*** realiza a leitura do arquivo *.sdc*, que é um formato usado para declarar as restrições de potência, atraso e área do projeto. O comando a seguir demonstra um exemplo que pode ser usado para a ferramenta realizar a leitura:

```
1     read_sdc
```

  No apêndice C, é possível observar o conteúdo de um arquivo *.sdc* onde são especificadas as características do *clock*, bem como as suas restrições de transição de subida e descida, incerteza e atraso máximo de entrada assim como a capacitância de entrada máxima e mínima.

- Preparar para a verificação de regras *DFT*: O *DFT Rule Checker* fornece informações sobre possíveis problemas na inserção de teste de acordo com a configuração especificada. As características específicas de *DFT* necessárias para o projeto são configuradas, e as portas relevantes usadas durante a operação de varredura, que dependem do estilo de varredura implementado, são definidas. Como já foi informado anteriormente, a célula escolhida foi a *muxed scan-flop*. Esta etapa é onde a síntese comum e a síntese com *DFT* se diferem. Os comandos as seguir podem ser utilizados na etapa de verificação:

```
1     set_db dft_scan_style muxed_scan
2     set_db dft_prefix dft_
3     define_shift_enable -name se -active high -create_port se
```

  O atributo "dft_scan_style" define o estilo de célula a ser utilizada. A configuração de verificação também necessita que de uma especificação dos pinos utilizados no estilo da célula. Os *muxed scan-flops* utilizam um sinal de "*shift enable*" (habilitação de deslocamento na tradução livre).

- Verificação de regras *DFT* e registro de *reports*: Na etapa de verificação, a ferramenta observa as regras e a consistência do *DFT*. O comando "*report*" é utilizado para fornecer informações sobre a arquitetrua do *DFT*. Para executar a verificação, o seguinte comando é utilizado:

```
1     check_dft_rules
```

  Este comando avalia o projeto para determinar se ele está pronto para receber os *scan-flops*. Os *flip-flops* que passarem por essa verificação são trocados por *scan-flops* na síntese e serão incluídos nas cadeias de varredura. Os comandos de *report* devem ser usados após a etapa de verificação. O comando "report dft_registers" gera como saída o *status* de todos os *scan-flops*. Já o comando "dft_setup" fornece informações de configuração.

```
1     report dft_registers
2     report dft_setup
3     report dft_registers
4     check_design -all
5     report dft_violations
```

- Consertar violações *DFT*: O ***Genus*** possui um comando para corrigir as violações identificadas na etapa de verificação. Os seguintes comandos servem para corrigir as violações observadas e registrar as violações observadas, respectivamente:

```
1     fix_dft_violations -design ${DESIGN} -test_control
2     report dft_violations
```

- Síntese de projeto + mapeamento de *scan*: Nesta etapa, o projeto é sintetizado para as portas genéricas. A ferramenta pega o arquivo principal do projeto e sintetiza o *RTL*. A ferramenta também permite que o projetista defina o esforço para tal mapeamento, juntamente com reports sobre o processo com os seguintes comandos:

```
1    set_db syn_generic_effort $GEN_EFF
2    syn_generic
3    write_snapshot -outdir $_REPORTS_PATH/ -tag generic
4    report datapath
5    report_summary -directory
```

  Este comando utiliza as restrições do arquivo *.sdc*. Em seguida, a ferramenta realiza o mapeamento do projeto para células descritas pela biblioteca anteriormente definida. A ferramenta permite com que o projetista defina o esforço para tal mapeamento com os seguintes comandos:

```
1    set_db syn_map_effort $MAP_OPT_EFF
2    syn_map
3    write_snapshot -outdir $_REPORTS_PATH/ -tag map
4    report_summary -directory
5    report datapath
```

- Configuração das cadeias de varredura: Existe um padrão que a ferramenta segue para a inserção de varredura. Porém, essa configuração pode ser modificada, assim como o limite de comprimento da cadeia de varredura. O projetista fica responsável por realizar essas modificações. Essa etapa é opcional para o projeto.
- Conexão das cadeias de varredura: Após a configuração, os *scan-flops* são conectados entre si e formam as cadeias de varredura com o seguinte comando:

```
1    set_db design:$DESIGN .dft_min_number_of_scan_chains 1
2    define_scan_chain -name top_chain -sdi scan_in -sdo
3      scan_out -create_ports
4    connect_scan_chains -auto_create_chains
5    check_dft_rules
```

  Existe a opção do comando "*–auto_create_chains*", que adiciona novas cadeias que não foram definidas. Essa opção entrega uma segurança a mais para o projetista, pois, se ela não estiver habilitada, a ferramenta pode indicar um erro se precisar de cadeias de varreduras adicionais. Os seguintes comandos permitem a observação da configuração e verificação das cadeias de varredura:

```
1    report dft_registers
2    report dft_chains
```

- Otimização incremental: A última etapa do fluxo de síntese é a otimização incremental. Os comandos utilizados podem ser observados a seguir:

```
set_db syn_map_effort $MAP_OPT_EFF
syn_opt
write_snapshot -outdir $_REPORTS_PATH/ -tag opt
report_summary -directory
```

- Escrita dos arquivos de saída e *Reports* da síntese: Após a síntese concluída sem violações, os arquivos relevantes para a próxima etapa podem ser gerados pelos seguintes comandos:

```
write_dft_atpg
write_hdl
write_do_lec
write_sdc
write_sdf
write_scande
```

O primeiro comando é responsável por especificar os arquivos de biblioteca *verilog* que são necessários para a execução da ferramenta ***Modus***. Esse comando gera a seguinte lista de arquivos necessários para executar o *ATPG*:

1. top_design.test_netlist.v
2. top_design.FULLSACN.pinassign
3. run_fullscan_sim e run_fullscan_sim_sdf
4. runmodus.atpg.tcl

O primeiro arquivo representa a *netlist* final e otimizada. Já o segundo arquivo é utilizado para aplicar estímulos necessários quando o modo de teste de varredura é construído. O arquivo seguinte serve como modelo que pode ser editado para verificar os padrões do *ATPG* de varredura usando o simulador. Por fim, o último arquivo é um modelo que pode ser editado para executar a ferramenta ***Modus***. É recomendado gerar os *reports* de toda a síntese ao final do fluxo para uma análise do circuito como um todo. O exemplo a seguir demonstra os *reports* extraídos.

```
report dft_setup
report_timing
report_power
report_area
report_qor
report_scan_chains
```

5 Geração de vetores de Teste

Após a síntese lógica gerar todas as saídas desejadas, os testes podem ser executados pelo *ATPG*. Esta etapa permite, através das cadeias de varreduras, a verificação de falhas nos dados do *design*. Todo este processo é necessário para a identificação dos comportamentos que exigem a seleção dos modelos de falha, modo de teste e as características do *design* para criar os vetores de teste. A verificação dos padrões de teste nas ferramentas disponíveis evita muitos custos, pois não é necessário um testador de *hardware* e garante que os testes detectem o maior número possível de falhas. A figura 5.1 demonstra como funciona este processo.

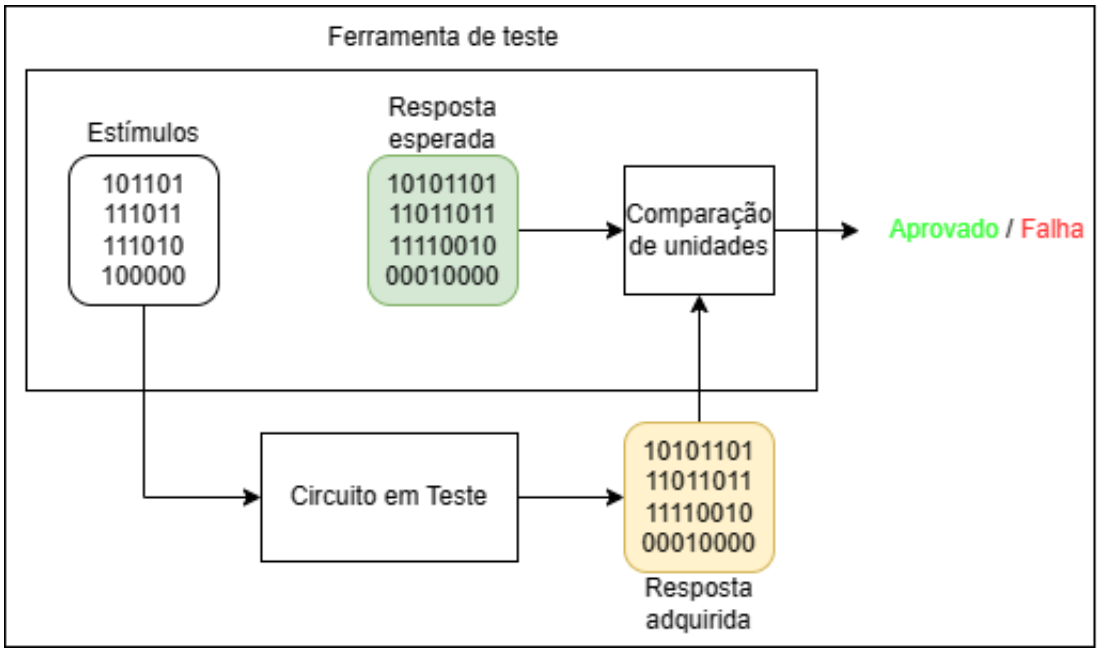

Figura 5.1 – Processo de teste de vetores gerados pelo *ATPG*

5.1 Solução de teste *DFT Modus*

Modus é uma ferramenta de nova geração da ***Cadence*** que foi escolhida para criar os padrões de teste do *design*. Após a geração dos padrões de teste, os vetores poderão ser simulados e compilados para a verificação. Para funcionar corretamente, a ferramenta precisa dos itens específicos (dois gerados pela ferramenta ***Genus***) listados a seguir:

- Arquivo da biblioteca no formato *verilog*.
- Arquivo que lista as portas principais e suas funções de teste (entrada de varredura, *clock* de teste do sistema, habilitação de varredura, etc.).
- *Netlist* do *design* gerado pela ferramenta ***Genus***.

5.2 Fluxo de Teste

A figura 5.2 ilustra o modelo básico *ATPG* para uma *netlist* inserida com varreduras.A ferramenta ***Genus*** gera um *script* ("*runmodus.atpg.tcl*") para o uso da ferramenta *Modus*. O *script* completo está disponível no anexo A e as etapas serão brevemente descritas a seguir.

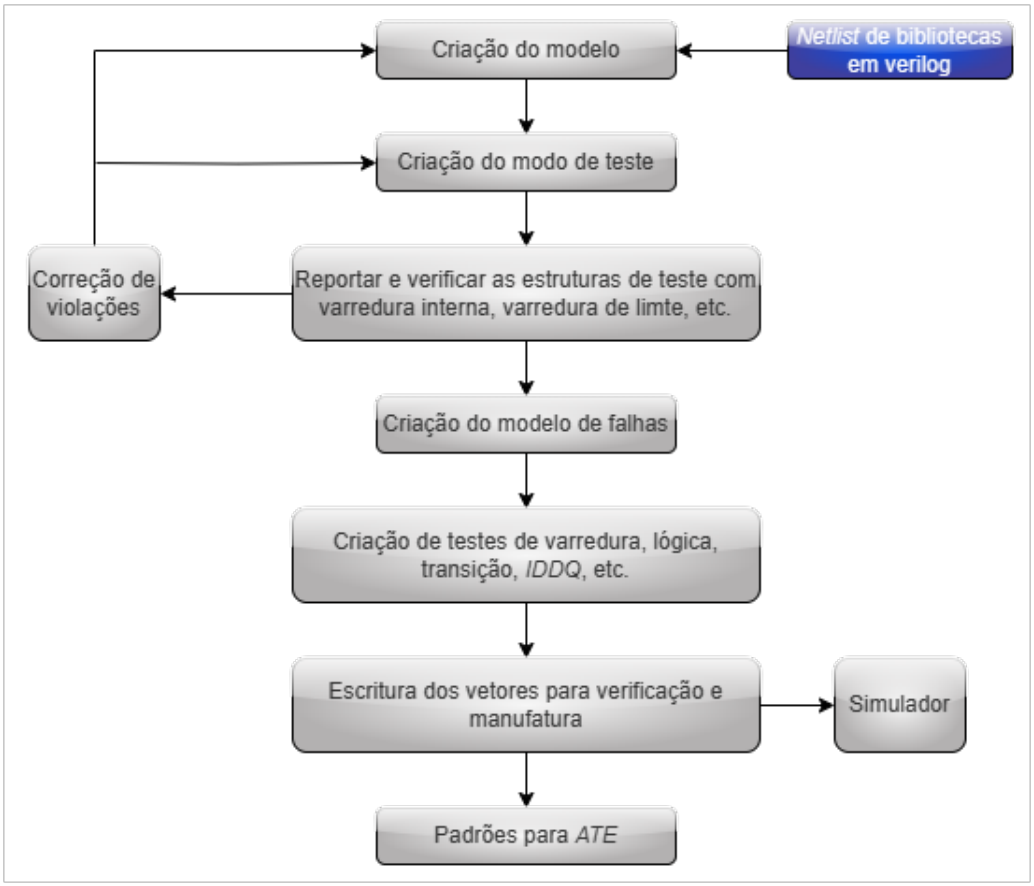

Figura 5.2 – Modelo para *ATPG*, adaptado de Mannucci (2018)

- Criação do modelo: nesta etapa, o modelo de teste ***Modus*** é construído a partir da *netlist* gerada pela ferramenta. Os seguintes comandos e opções são necessários:

```
1    build_model
2        -cell
3        -techlib
4        -designsource
```

- Criação do modo de teste: Nesta etapa, os modos de teste do Modus são construídos. Os modos de teste definem a estrutura de varredura e a lógica para os testes. Os seguintes comandos e opções são necessários:

```
1    build_testmode
2        -testmode
3        -assignfile
```

O arquivo de atribuição (*assignfile*) contém a lista de portas principais e suas respectivas funções de teste. Essas devem ser configuradas com base nos atributos do *design*.

- Reportar e verificar as estruturas: Nesta etapa o projetista pode coletar dados sobre a criação dos testes, em seguida a ferramente verifica possíveis problemas no design como problemas na inserção de varredura, temporização, etc. O seguintes comandos são utilizados:

```
1    report_test_structures
2        -testmode  FULLSCAN
3    verify_test_structures
4        -messagecount
5        -testmode
```

- Criação do modelo de falhas: O conjunto de defeitos abstratos que exibe um tipo comum de mudança de comportamento é chamado de modelo de falha. A ferramenta desenvolve um modelo que descreve diferentes tipos de falhas que podem ocorrer no circuito. Os comandos a seguir são utilizados, percebe-se que na opção *"includedynamic"* está com a opção "*no*", pois nessa etapa somente as falhas estáticas interessam.

```
1    build_faultmodel
2        -includedynamic  no
```

- Criação de testes de varredura: Nessa etapa, padrões são criados para testar a células de varreduras que a ferramenta identifica. O comando a seguir é utilizado:

```
1    create_scanchain_tests
2        -experiment
3        -testmode
```

- Criação de testes de lógica: Nessa etapa, padrões de testes são gerados para verificar falhas estáticas no circuito. O comando a seguir é utilizado:

```
1    create_logic_tests
2        -experiment
3        -testmode
```

- Escritura de vetores para verificação: Os arquivos gerados nessa etapa são padrões de testes e um arquivo *testbench* na linguagem especificada (*verilog*). O comando a seguir é utilizado:

```
1    write_vectors
2        -inexperiment
3        -testmode
4        -language
5        -scanformat
```

6 Resultados e Discussões

Neste capítulo, são apresentados os resutados obtidos na execução das etapas de validação, síntese e teste.

6.1 Validação em *Hardware*

6.1.1 Simulação funcional *RISC-V* no ***Vivado***

Para a simulação automática, utilizamos o testbench disponível no código do PicoRV32 (icebreaker_tb.v, disponível no anexo F), o código busca um arquivo de memória (*firmware_file*), que no caso é o arquivo icebreaker_fw.hex, que contém as instruções em hexadecimal. A figura 6.1 mostra as instruções simuladas.

N°	Tipo de instrução	Instruções	Codificação em Hexadecimal
1	Tipo I	addi t0, zero, 0	00000293
2	Tipo I	addi t1, zero, 5	00500313
3	Tipo I	addi t2, zero, 10	00A00393
4	Tipo I	addi t3, zero, 0	00000E13
5	Tipo I	addi t4, zero, 0	00000E93
6	Tipo R	sub t2, t2, t1	406383B3
7	Tipo B	verificar: beq t1, t2, caso_1	00730463
8	Tipo J	jal t4, caso_2	00C00EEF
9	Tipo I	caso_1: addi t1, zero, 6	00600313
10	Tipo J	jal t4, verificar	FF5FFEEF
11	Tipo I	caso_2: addi t1, zero, 5	00500313
12	Tipo U	lui t3, 1000000	F4240E37
13	Tipo J	jal t4, verificar	FE9FFEEF

Figura 6.1 – Instruções em hexadecimal

Na figura 6.2 é possível observar a primeira instrução em hexadecimal, o qual se inicia no endereço de memória "00100000", a partir da segunda instrução o endereço de memória é acrescido em 4 unidades por instrução, a figura 6.3 apresenta a leitura da sexta instrução, onde já podemos observar os valores dos registradores "t1" e "t2" correspondentes aos sinais [6] e [7], respectivamente. É importante ressaltar que o núcleo faz leituras de 32 bits, ou seja, para uma instrução é necessário um número de 8 dígitos em hexadecimal que é lido de forma inversa, a cada dois dígitos (os dois primeiros dígitos são os menos significativos da instrução e assim por diante.

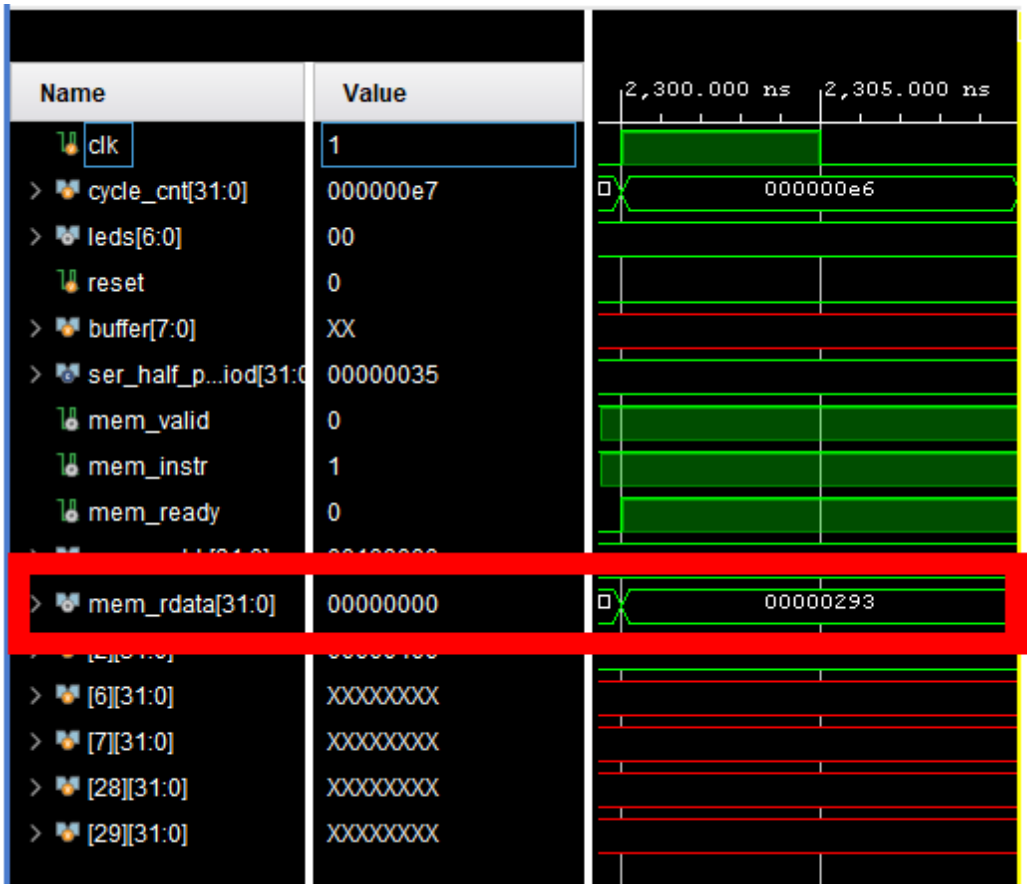

Figura 6.2 – Leitura da primeira instrução

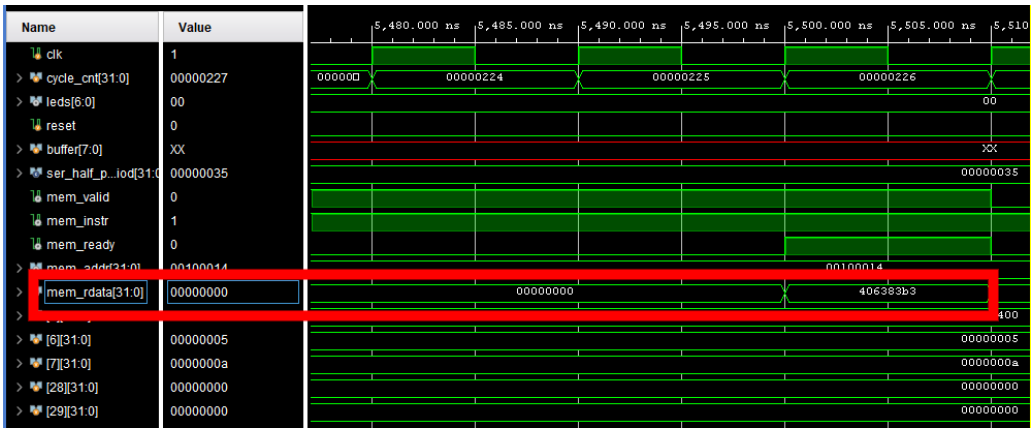

Figura 6.3 – Leitura da sexta instrução

6.1.2 Simulação funcional do *MLP443*

Para a simulação, foi utilizado um *testbench* que possui 4 entradas em binário ("01000000101100110011001100110011", "01000000001000000000000000000000", "01000000011110011001100110011000", "00111111100011001100110011001100"). O arquivo está disponível no anexo G. A simulação pode ser observada na figura 6.4, onde o tempo de execução é de 211 [ns].

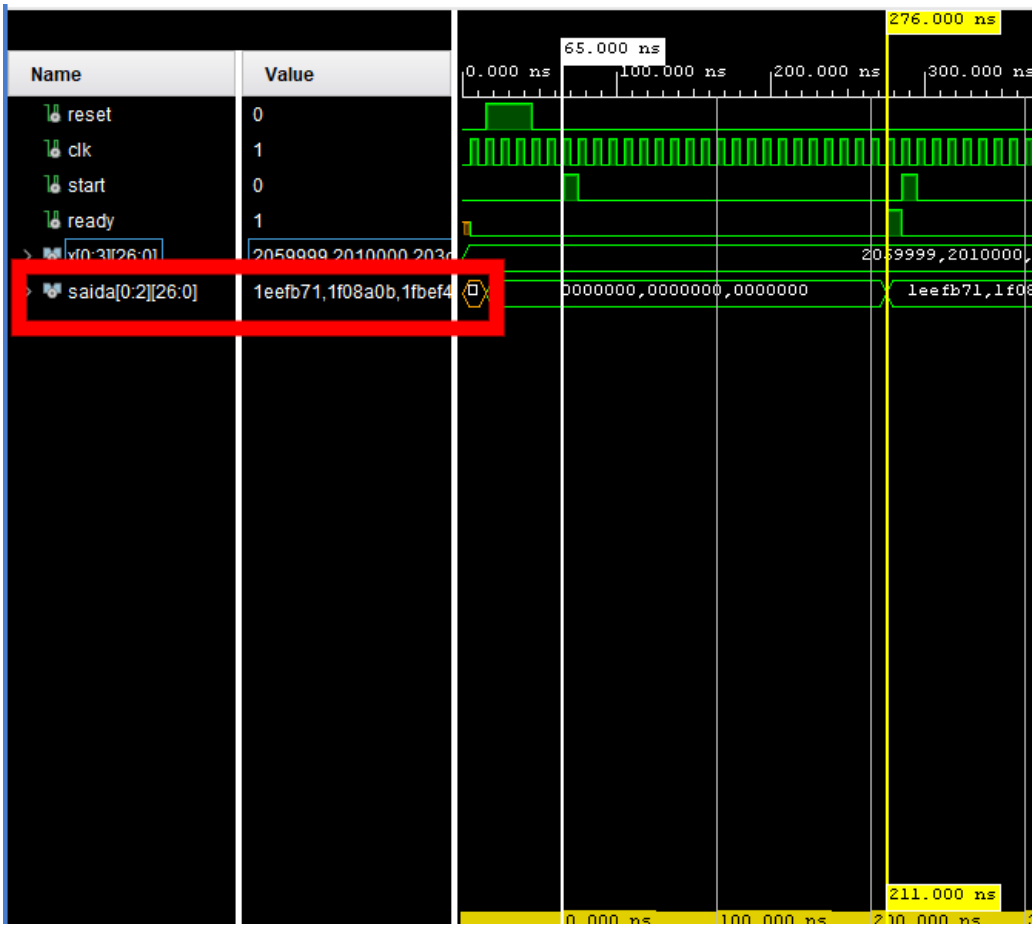

Figura 6.4 – Simulação funcional do MLP443

6.1.3 Simulação do *firmware*

O arquivo *.hex* gerado pelo *firmware* é utilizado na simulação. Esta simulação pode ser comparada com a simulação funcional do módulo MLP443 observada na figura 6.4 de acordo com os valores gerados e o tempo de execução do neurônio, observados na figura 6.5. É importante destacar que o tempo de execução do neurônio implementado no *RISC-V* foi de 7.24 [us], porém esse tempo de execução depende diretamente do *firmware* que utiliza outras instruções entre as leituras, logo esse tempo pode ser otimizado.

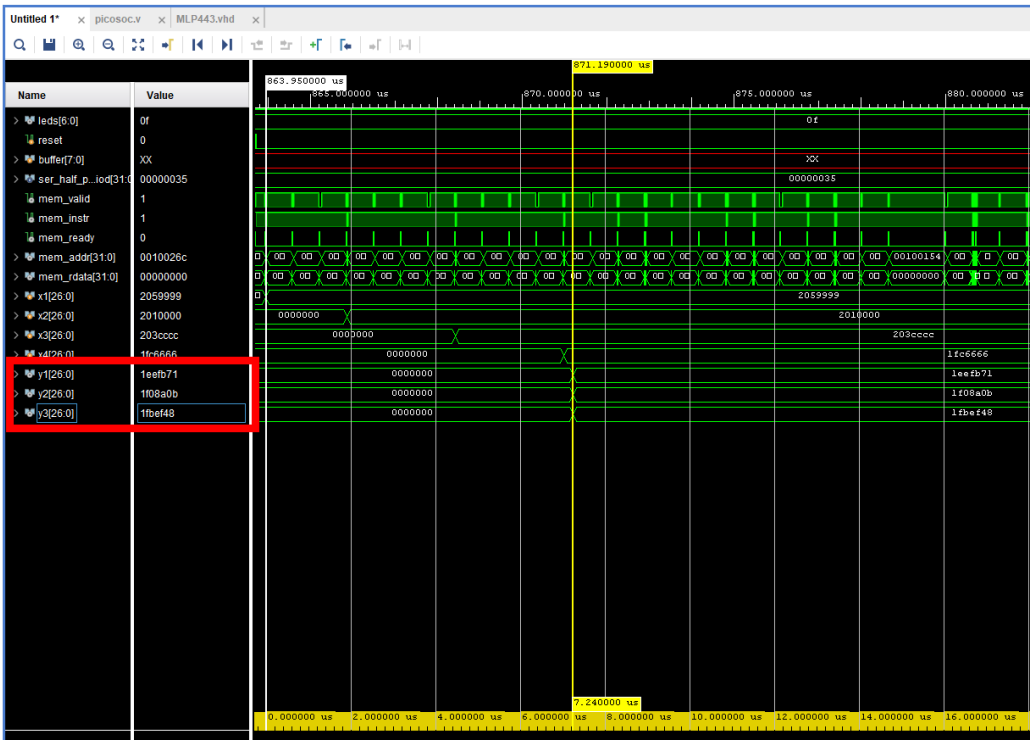

Figura 6.5 – Simulação do *firmware*

6.1.4 Estimativa de recursos

Foi realizada a implementação do circuito e podemos observar na tabela 6.1 as estimativas de recursos.

Tabela 6.1 – Estimativa de recursos

Recursos	Estimativa	Disponível	Utilização
LUT	12554	63400	21,38
LUTRAM	48	19000	0,25
FFs	2703	126800	2,13
BRAM	32	135	23,70
DSP	18	240	7,50
IO	16	210	7,62

A figura 6.6 apresenta a estimativa de potência consumida pela placa.

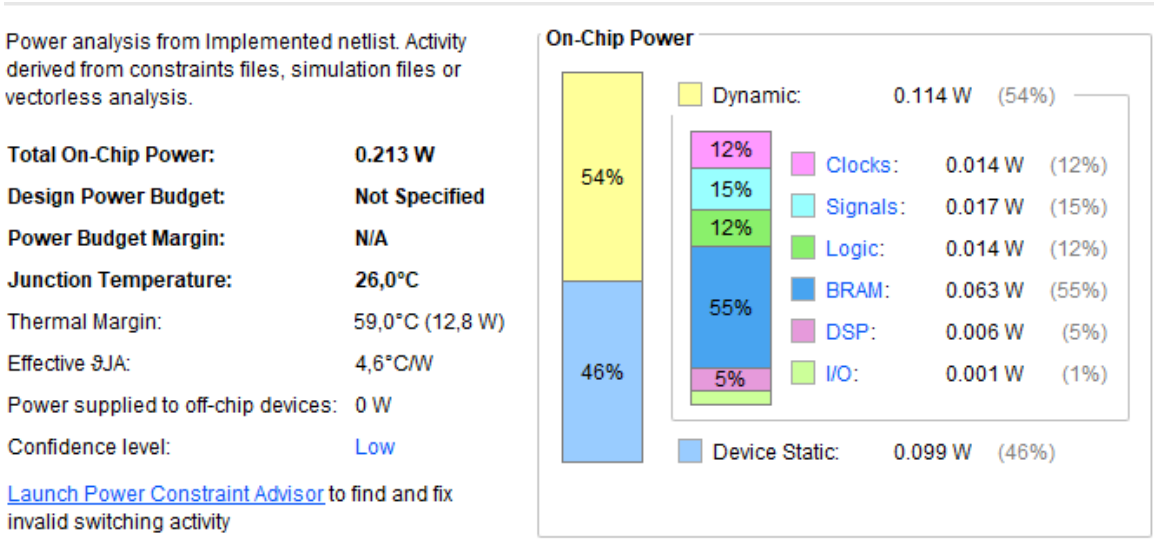

Figura 6.6 – Estimativa de consumo de potência

Por fim, a figura 6.7 apresenta o *report* de *timing*, que obteve uma folga de temporização negativa. Esta folga, apesar do circuito funcionar, deve ser retirada.

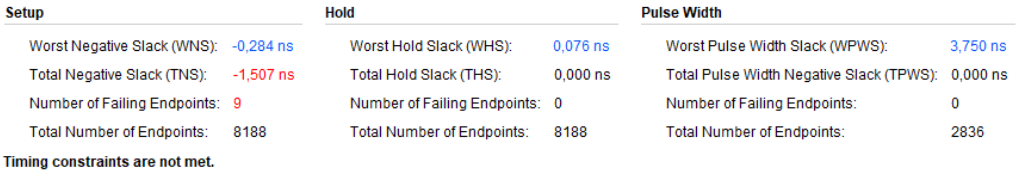

Figura 6.7 – *Report* de *timing*

6.1.5 Transmissão Serial

O arquivo *.bin* gerado pelo *firmware* é utilizado para configurar a memória *flash* da placa *Nexys 4*, após a configuração, é realizada a programação da placa, assim o programa Hercules pode realizar os *prints* exigidos no *firmware*, como pode ser observado na figura 6.8

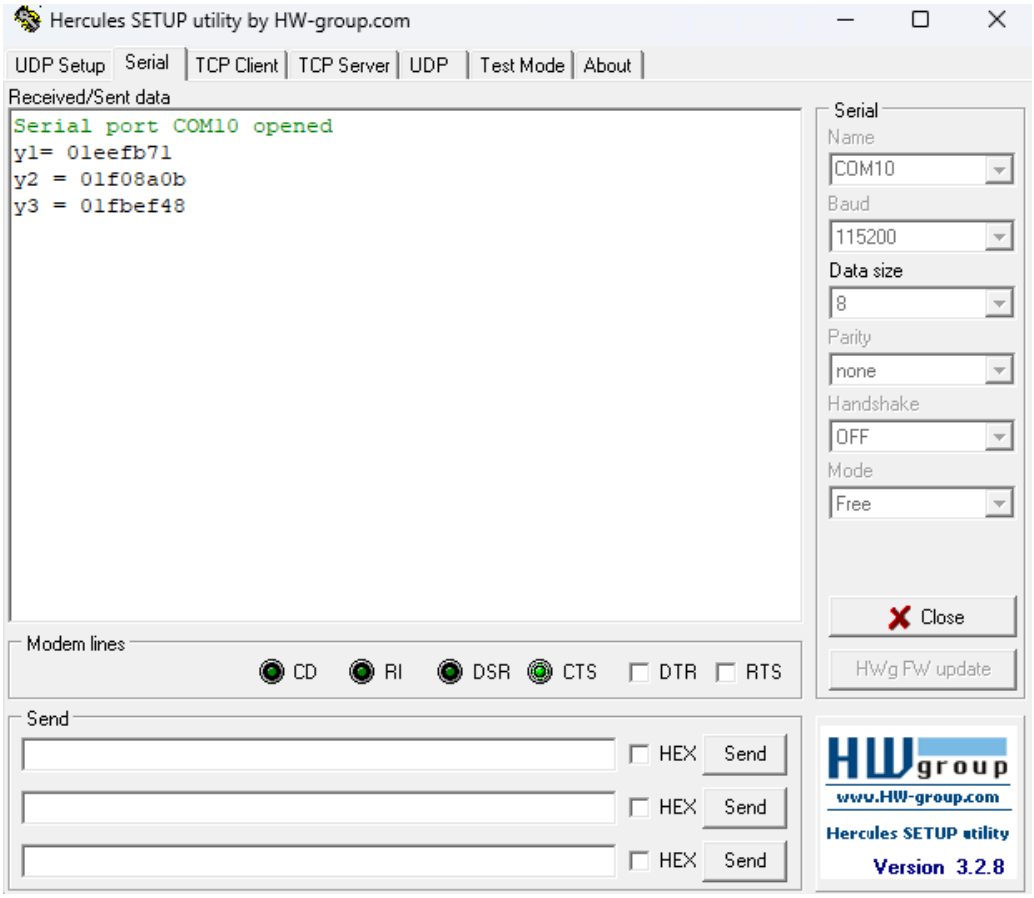

Figura 6.8 – Visualizador de porta serial Hercules

6.2 Implementação nas ferramentas *Cadence*

Para otimizar o processo, os comandos foram separados por *scripts*, pastas e todos são acionados por um arquivo *MAKEFILE*, que pode ser observado no apêndice G.

6.2.1 Simulações

Para realizar as simulações funcionais e pós-síntese, foi utilizado o arquivo *testbench* disponibilizado pelos desenvolvedores do *PicoRV32* com algumas modificações, disponível no apêndice H. Este *testbench* busca o arquivo *firmware.hex*, que é gerado pelo arquivo *firmware.h*, disponível no anexo H, já o arquivo *.hex* busca outras funções que testam as instruções do *RISC-V*, descritas na tabela 2.6 e entrega a mensagem "*TEST PASSED!*", em seguida ele testa os barramentos *axi* e, como não é utilizado, aparece a mensagem "ERROR!". Os comandos para realizar as simulações são descritas nas seções "sim", "sim_ps" e "sim_ps_dft" do apêndice G.

Para a simulação de vetores de teste, foi utilizado um *script* gerado pela ferramenta ***Modus***, porém os comandos foram modificados para adaptar à ferramenta ***Xcelium***, e são descritos na seção "*sim_atpg*" do apêndice G.

6.2.1.1 Simulação funcional

A simulação funcional é acionada pelo comando "*make sim*". A simulação pós-síntese sem *DFT* é acionada pelo comando "*make sim_ps*". Já com *DFT* é "*make sim_ps_dft*". A figura 6.9 apresenta a mensagem após realizada a simulação, em todas as três foi obtido sucesso com o mesmo número de ciclos.

```
xcelium> run

Hello World! If you can read this message then
the PicoRV32 CPU seems to be working just fine.

                TEST PASSED!

TRAP after 12158 clock cycles
ERROR!
Simulation stopped via $stop(1) at time 122680 NS + 1
xcelium>
```

Figura 6.9 – Simulação Funcional

6.2.1.2 Simulação de vetores de teste

A simulação de vetores é acionado pelo comando "*make sim_atpg*". É possível observar na figura 6.10 que foram realizados 213 testes e todos passaram e foram realizadas 530862 comparações e todas foram aprovadas pela ferramenta.

```
INFO (TVE-210): Results for vector file: ./et_atpg_output/testresults/verilog/VER.FULLSCAN.picorv32_wrapper_atpg.dat
                        Number of Cycles:           640
                        Number of Tests:            212
                          - Passed Tests:           212
                          - Failed Tests:           0
                        Number of Compares:         528516
                          - Good Compares:          528516
                          - Miscompares:            0
                        Time:                       51680000.00 ps


INFO (TVE-209): Cumulative Results:
                        Number of Files Simulated:  2
                        Total Number of Cycles:     646
                        Total Number of Tests:      213
                          - Total Passed Tests:     213
                          - Total Failed Tests:     0
                        Total Number of Compares:   530862
                          - Total Good Compares:    530862
                          - Total Miscompares:      0

Simulation complete via $finish(1) at time 51680 NS + 0
./et_atpg_output/testresults/verilog/VER.FULLSCAN.picorv32_wrapper_atpg.mainsim.v:5247       $finish;
xcelium>
```

Figura 6.10 – Simulação dos vetores de teste

6.2.2 Síntese Lógica

O comando para realizar a síntese lógica sem *DFT* pode ser observado na seção "*make synthesis*" do apêndice G e o comando aciona o *script* descrito no apêndice A. Já com *DFT* o comando está na seção "*make synthesis_dft*" descrita no apêndice G, comando esse que aciona

o *script* descrito no apêndice B. Os esquemáticos sem e com *DFT* podem ser observados nas figuras 6.11 e 6.12, respectivamente

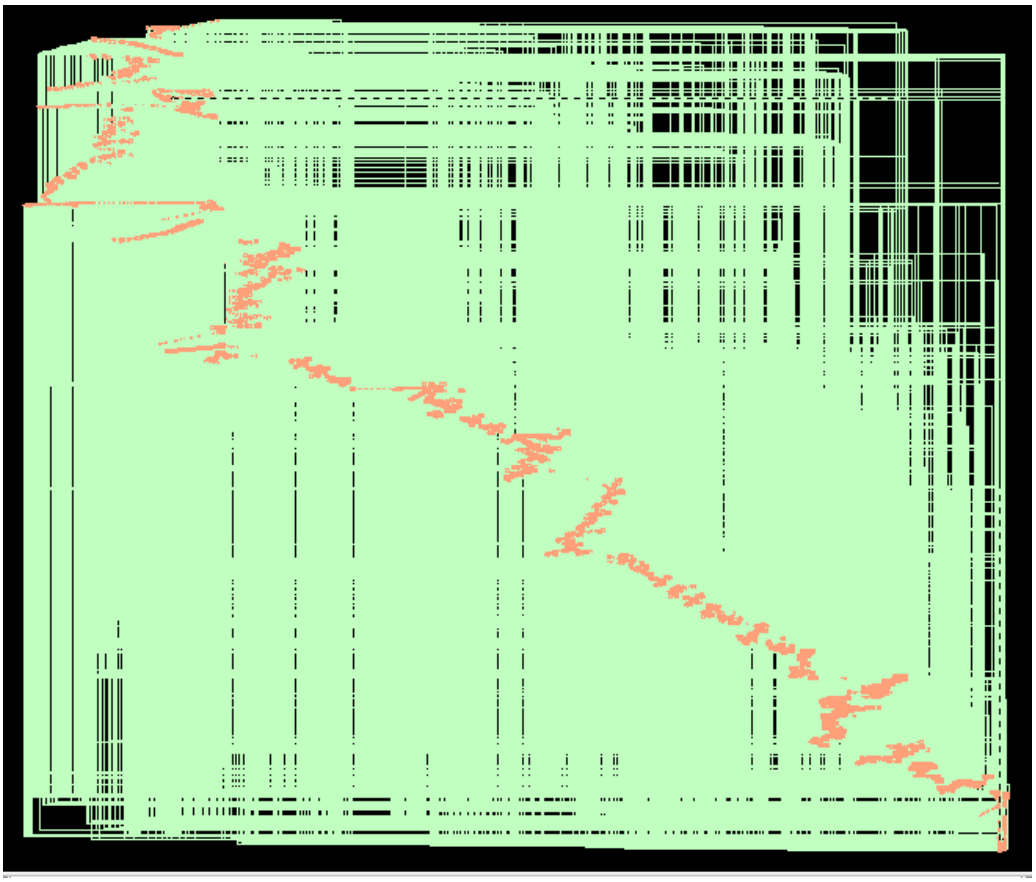

Figura 6.11 – Esquemático Pós Síntese sem *DFT*

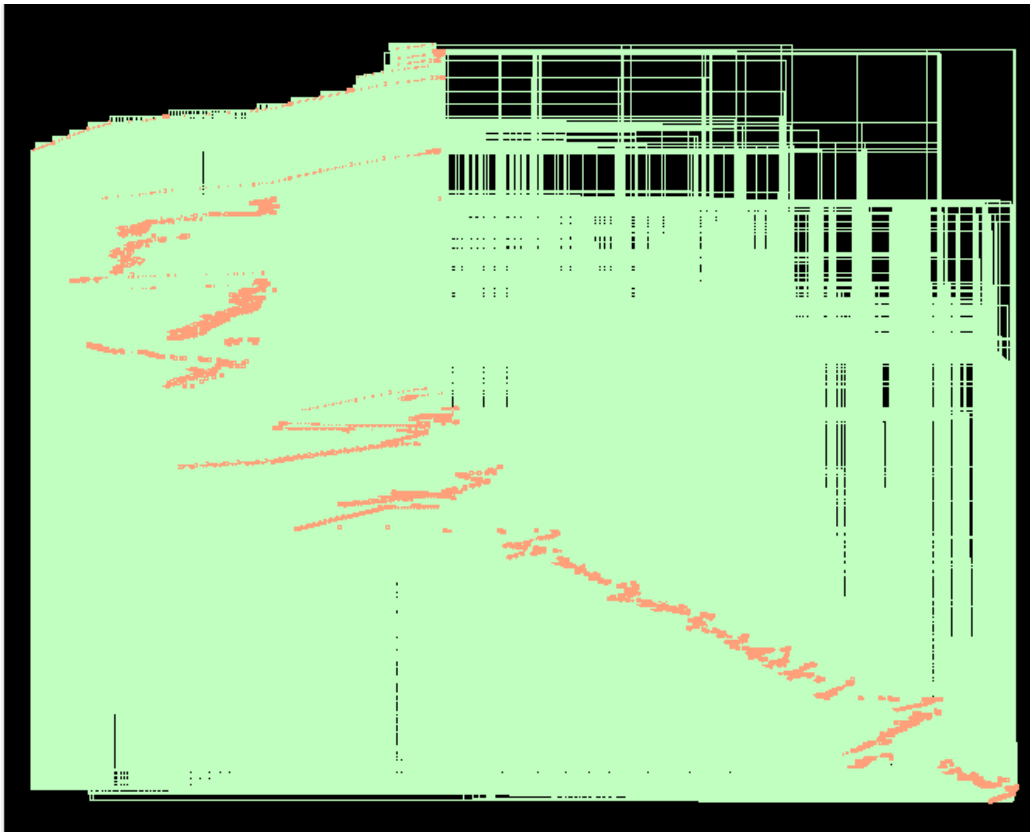

Figura 6.12 – Esquemático Pós Síntese com *DFT*

6.2.2.1 Comparações

A tabela 6.2 realiza uma comparação entre as estimativas de recursos, que a ferramenta ***Genus*** entrega após a síntese lógica, com e sem *DFT*. É possível observar que a área com *DFT* representa aproximadamente 116% da área sem *DTF*, o que está entre o esperado, de acordo com a seção 2.4. Já na parte do consumo de potência, houve um aumento de praticamente 18%, o que também está de acordo com a seção 2.4 e com um resultado bastante satisfatório se comparado ao consumo em *FPGA* indicado na figura 6.6. Sobre a temporização, a folga está positiva, o que é um bom sinal, pois não há atrasos, porém é uma folga muito curta, qualquer instabilidade pode causar um atraso, seja com ou sem *DFT*.

Tabela 6.2 – Estimativa de recursos pela ferramenta ***Cadence***

Recursos	**Sem *DFT***	**Com *DFT***
Área (μm^2)	297467	345000
Potência (nW)	67476850,765	80140023,369
Folga de temporização (ps)	10	3

6.2.3 *ATPG*

O comando para criar os vetores de teste está descrito na seção "*make modus*" do apêndice G, o comando aciona o *script* descrito no anexo A. A mensagem após a criação dos vetores de teste é descrita na figura 6.13, na imagem é possível observar que os vetores de teste cobrem 99,99% do circuito. Esta cobertura está relacionada à quantidade de testes descritos na seção 6.2.1.2.

```
INFO (TDA-220):   --- Tests ---   Faults     ---- ATCov ----   -- Faults --   - Elapsed Time -  [end TDA_220]
INFO (TDA-220):    Sim.    Eff.  Detected    Tmode    Global     Untested                        [end TDA_220]
INFO (TDA-220):       1       1     16429   21.73%   21.73%        59193          00:00.17    [end TDA_220]
INFO (TDA-220):   --- Tests ---   Faults     ---- ATCov ----   -- Faults --   - Elapsed Time -  [end TDA_220]
INFO (TDA-220):    Sim.    Eff.  Detected    Tmode    Global     Untested                        [end TDA_220]
INFO (TDA-220):      16      16     41283   76.32%   76.32%        17909          00:02.05    [end TDA_220]
INFO (TDA-220):      32      32      4859   82.75%   82.75%        13042          00:02.41    [end TDA_220]
INFO (TDA-220):      48      48      2795   86.45%   86.45%        10247          00:02.65    [end TDA_220]
INFO (TDA-220):      64      64      2839   90.21%   90.21%         7405          00:02.89    [end TDA_220]
INFO (TDA-220):      80      80      1477   92.16%   92.16%         5924          00:03.07    [end TDA_220]
INFO (TDA-220):      96      96      1416   94.04%   94.04%         4504          00:03.22    [end TDA_220]
INFO (TDA-220):     112     112      1155   95.58%   95.58%         3340          00:03.40    [end TDA_220]
INFO (TDA-220):     128     128      1105   97.05%   97.05%         2228          00:03.54    [end TDA_220]
INFO (TDA-220):     144     144       667   97.94%   97.94%         1560          00:03.64    [end TDA_220]
INFO (TDA-220):     160     160       683   98.84%   98.84%          877          00:03.73    [end TDA_220]
INFO (TDA-220):     176     176       568   99.59%   99.59%          309          00:03.80    [end TDA_220]
INFO (TDA-220):     192     192       267   99.94%   99.94%           42          00:03.85    [end TDA_220]
INFO (TDA-220):     208     208        36   99.99%   99.99%            5          00:03.88    [end TDA_220]
INFO (TDA-220):     212     212         4   99.99%   99.99%            0          00:03.90    [end TDA_220]
```

Figura 6.13 – Cobertura dos vetores de teste

7 Conclusões

Nos capítulos anteriores foram explorados conceitos sobre técnicas de implementação de *design* para testabilidade, arquitetura de processador *RISC-V* e um aprofundamento sobre algumas ferramentas ***Cadence***. Esta implementação permite que testes *ATPG* verifiquem falhas de produção no circuito. Para a validação em *hardware*, o objetivo era a implementação de uma rede neural *MLP* com quatro neurônios de entrada, quatro na camada oculta e três na saída, utilizando o processador *PICORV32*.

7.1 Objetivos Alcançados

O objetivo principal deste trabalho, de aplicar metodologias de *DFT* e um processador de arquitetura *RISC-V*, foi concluído com sucesso.

Na validação em *hardware*, de acordo com o resultado das simulações foi possível constatar o correto funcionamento quando comparada com a simulação do neurônio por si só, que já havia sido validado. O consumo de potência torna promissor a utilização em aplicações de baixo consumo, em comparação com outros processadores. Apesar da remoção do ILA, a análise de *timing* permaneceu apresentando folga negativa logo, uma análise futura no código deve ser realizada para mitigar o problema.

Ao iniciar o processo de síntese, foi determinado que 100% dos registradores seriam substituídos por *scan-flops*. O *design* de *DFT* e as características propostas foram sintetizados com sucesso. Os relatórios gerados pela síntese detalharam uma aumento de aproximadamente 16% da área original, o que era esperado, um aumento de aproximadamente de 18% no consumo de potência, um resultado satisfatório, porém, uma implementação de baixo consumo pode melhorar ainda mais o consumo de potência, por fim, os relatórios de temporização entregaram uma folga bem ajustada, logo, um estudo sobre flexibilização do *clock* deve ser feito. Já os relatórios da geração de padrões de teste detalharam que, com o design implementado, é possível alcançar valores extremamente altos de coberturas para falhas estáticas, obtendo uma cobertura de 99,99% do circuito. Os vetores de testes criados foram gerados corretamente e podem ser aplicados no chip uma vez fabricado.

7.2 Perspectivas futuras

Embora o trabalho tenha avançado positivamente, ainda há várias oportunidades para aprimoramentos, tais como:

- Análise na validação em *hardware* para retirar a folga de temporização negativa na implementação;
- Estudar sobre a implementação de *DFT* de baixo consumo;

- Realizar a etapa de síntese física;
- Realizar o processo em outra tecnologia.

Referências

AERTS, J.; MARINISSEN, E. Scan chain design for test time reduction in core-based ics. *In*: **Proceedings International Test Conference 1998 (IEEE Cat. No.98CH36270)**. [*S.l.: s.n.*], 1998. p. 448–457. Citado na p. 19.

ARTHUR, D. M. B. **Projeto e Desenvolvimento de Processadores RISC-V com a ISA RV32IMF Usando as Microarquiteturas Uniciclo, Multiciclo e Pipeline em FPGA**. Tese (Doutorado) — Universidade de Brasília, Brasil, 2021. Citado nas pp. 27 e 28.

BREUER, M.; GUPTA, S.; MAK, T. Defect and error tolerance in the presence of massive numbers of defects. **IEEE Design Test of Computers**, v. 21, n. 3, p. 216–227, 2004. Citado na p. 19.

CADENCE. **Cadence Modus DFT Software Solution**. 2023. Disponível em: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/test/modus-test.html. Citado na p. 20.

CADENCE. **Cadence RTL-to-GDSII Flow Training**. 2023. Disponível em: https://www.cadence.com/en_US/home/training/all-courses/86136.html. Citado nas pp. 7, 16 e 46.

CADENCE. **Design for Test Fundamentals Training**. 2023. Disponível em: https://www.cadence.com/en_US/home/training/all-courses/82125.html. Citado nas pp. 7 e 19.

CALçADA, R. d. O. Design of Steel : a RISC-V Core. 2020. Accepted: 2021-03-19T04:18:56Z. Disponível em: https://lume.ufrgs.br/handle/10183/219134. Citado na p. 29.

COLLEGE A.; DOYLE, C. R. A. **FPGA Flexible Architecture - Olin College of Engineering.** 2023. Disponível em: http://ca.olin.edu/2005/fpga_dsp/images/fpga001.jpg. Citado nas pp. 7 e 27.

MANNUCCI, A. **Design for Testability Implementation using Cadence DFT Compiler**. 2018. Publisher: California State University, Northridge. Citado nas pp. 7, 20, 25, 26, 47 e 53.

MUSTé, P. F. **Design for Testability methodologies applied to a RISC-V processor**. Dissertação (Mestrado) — Faculty of the Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona Universitat Politècnica de Catalunya, Barcelona, 01 2021. An optional note. Citado nas pp. 9, 19, 26 e 33.

PANDEY, R.; PANDEY, S.; HAMMED, C. M. S. Security in design for testability (dft). *In*: **2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)**. [*S.l.: s.n.*], 2017. p. 1–4. Citado nas pp. 20 e 22.

PICORV32 - A Size-Optimized RISC-V CPU. 2023. Disponível em: https://github.com/YosysHQ/picorv32. Citado nas pp. 7, 17, 31 e 32.

ROSTAMI, F. K. M.; KARRI, R. A primer on hardware security: Models, methods, and metrics. *In*: **Proceedings of the IEEE (Volume: 102, Issue: 8, August 2014)**. [*S.l.: s.n.*], 2014. p. 1283 – 1295. Citado na p. 22.

VETHAMUTHU, E. R. S.; SIVANANTHAM, S.; SAKTHIVEL, R. Implementation of hierarchical dft approach for better testability. *In*: **2018 International Conference on Emerging Trends and Innovations In Engineering And Technological Research (ICETIETR)**. [*S.l.: s.n.*], 2018. p. 1–4. Citado na p. 19.

WANG, L.-T.; WU, C.-W.; WEN, X. **VLSI test principles and Architectures Design for Testability**. [*S.l.*]: Elsevier Morgan Kaufmann Publishers, 2006. Citado nas pp. 7, 21, 22, 23, 24, 25 e 26.

WATERMAN, A.; ASANOVIć, K. **The RISC-V Instruction Set Manual, Volume I: User-Level ISA**. Berkeley, CA: RISC-V Foundation, 2019. Citado nas pp. 28 e 29.

Apêndices

Apêndice A – *genus.tcl*

```
set DESIGN "picorv32_wrapper"
set GEN_EFF medium
set MAP_OPT_EFF high
set _OUTPUTS_PATH outputs
set _REPORTS_PATH reports
set _LOG_PATH logs

############### Library setup - (Read Target Libraries) ###############
## Set library search path
## Indica os arquivos da biblioteca alvo
set_db init_lib_search_path
   "/opt/cadence/tsmc_018_6M20K/tcb018gbwp7t_270a_FE/
tcb018gbwp7t_270a_nldm/TSMCHOME/digital/Front_End/
timing_power_noise/NLDM/tcb018gbwp7t_270a"
## Read Target Libraries
read_libs tcb018gbwp7ttc.lib
## Read LEF Files
set_db lef_library "/opt/cadence/tsmc_018_6M20K/tcb018gbwp7t_270a_FE/
tcb018gbwp7t_270a_sef/TSMCHOME/digital/Back_End/
lef/tcb018gbwp7t_270a/lef/tcb018gbwp7t_6lm.lef"

############### Load Design - (Read HDL Files)###############
## Set RTL search path
set_db init_hdl_search_path ../src/
set rtl_files [glob ../src/*.v]
## Read the mapped to scan netlist
read_hdl -v2001 ${rtl_files}

############### Elaborate Design ##############
## Elaborate overall design
elaborate ${DESIGN}

############### Constraints Setup - (Set Timing & Design
   Constraints)###############
read_sdc ../sdc/picorv32.sdc

############### Synthesizing to generic ##############
set_db syn_generic_effort $GEN_EFF
syn_generic
write_snapshot -outdir $_REPORTS_PATH -tag generic
report datapath > $_REPORTS_PATH/generic/${DESIGN}_datapath.rpt
report_summary -directory $_REPORTS_PATH/generic/${DESIGN}_gen.rpt

############### Synthesizing to gates ##############
set_db syn_map_effort $MAP_OPT_EFF
syn_map
write_snapshot -outdir $_REPORTS_PATH -tag map
report_summary -directory $_REPORTS_PATH/map/${DESIGN}_map.rpt
```

```
report datapath > $_REPORTS_PATH/map/${DESIGN}_datapath.rpt

############### Optimize Netlist ###############
set_db syn_opt_effort $MAP_OPT_EFF
syn_opt
write_snapshot -outdir $_REPORTS_PATH -tag opt
report_summary -directory $_REPORTS_PATH/map/${DESIGN}_opt.rpt

############### Reports ###############
report_timing > ${_REPORTS_PATH}/${DESIGN}_report_timing.rpt
report_power > ${_REPORTS_PATH}/${DESIGN}_report_power.rpt
report_area > ${_REPORTS_PATH}/${DESIGN}_report_area.rpt
report_qor > ${_REPORTS_PATH}/${DESIGN}_report_qor.rpt

############### Outputs ###############
write_hdl -mapped > ${_OUTPUTS_PATH}/${DESIGN}_netlist.v
write_sdc > ${_OUTPUTS_PATH}/${DESIGN}_sdc.sdc
write_sdf -timescale ns -nonegchecks -recrem split -edges check_edge
   -setuphold split > ${_OUTPUTS_PATH}/${DESIGN}_delays.sdf

puts "================================="
puts "Synthesis DFT Finished ........."
puts "================================="
```

Apêndice B – *genus_dft.tcl*

```
set DESIGN "picorv32_wrapper"
set GEN_EFF medium
set MAP_OPT_EFF high
set _OUTPUTS_PATH outputs_dft
set _REPORTS_PATH reports_dft
set _LOG_PATH logs_dft
set ET_WORKDIR modus

############## Library setup - (Read Target Libraries) ##############
## Set library search path
## Indica os arquivos da biblioteca alvo
set_db init_lib_search_path
   "/opt/cadence/tsmc_018_6M20K/tcb018gbwp7t_270a_FE/
tcb018gbwp7t_270a_nldm/TSMCHOME/digital/Front_End/
timing_power_noise/NLDM/tcb018gbwp7t_270a"
## Read Target Libraries
read_libs tcb018gbwp7ttc.lib
## Read LEF Files
set_db lef_library "/opt/cadence/tsmc_018_6M20K/tcb018gbwp7t_270a_FE/
tcb018gbwp7t_270a_sef/TSMCHOME/digital/Back_End/
lef/tcb018gbwp7t_270a/lef/tcb018gbwp7t_6lm.lef"

############## Load Design - (Read HDL Files)##############
## Set RTL search path
set_db init_hdl_search_path ../src/
set rtl_files [glob ../src/*.v]
## Read the mapped to scan netlist
read_hdl -v2001 ${rtl_files}

############## Elaborate Design ##############
## Elaborate overall design
elaborate ${DESIGN}

############## Constraints Setup - (Set Timing & Design
   Constraints)##############
read_sdc ../sdc/picorv32.sdc

############## DFT Setup - (Setup for DFT Rule Checker)##############
set_db dft_scan_style muxed_scan
set_db dft_prefix dft_
define_shift_enable -name se -active high -create_port se

############## DFT Rule Checker - (Run DFT Rule Checker and Report
   Registers)##############
check_dft_rules > $_REPORTS_PATH/${DESIGN}-tdrcs
report dft_registers > $_REPORTS_PATH/${DESIGN}-DFTregs
report dft_setup > $_REPORTS_PATH/${DESIGN}-DFTsetup_tdrc
report dft_registers
```

```
check_design -all
report dft_violations > $_REPORTS_PATH/${DESIGN}-AdvancedDFTViols
#fix_dft_violations -design $DESIGN -test_control
#check_atpg_rules
#analyze_testability

############### Synthesizing to generic ###############
set_db syn_generic_effort $GEN_EFF
syn_generic
write_snapshot -outdir $_REPORTS_PATH -tag generic
report datapath > $_REPORTS_PATH/generic/${DESIGN}_datapath.rpt
report_summary -directory $_REPORTS_PATH/generic/${DESIGN}_gen.rpt

############### Synthesizing to gates ###############
set_db syn_map_effort $MAP_OPT_EFF
syn_map
write_snapshot -outdir $_REPORTS_PATH -tag map
report_summary -directory $_REPORTS_PATH/map/${DESIGN}_map.rpt
report datapath > $_REPORTS_PATH/map/${DESIGN}_datapath.rpt

################### Scan Chains ###############
set_db design:${DESIGN} .dft_min_number_of_scan_chains 1
define_scan_chain -name top_chain -sdi scan_in -sdo scan_out
   -create_ports
connect_scan_chains -auto_create_chains
check_dft_rules
report dft_registers
report dft_chains

############### Optimize Netlist ###############
set_db syn_opt_effort $MAP_OPT_EFF
syn_opt
write_snapshot -outdir $_REPORTS_PATH -tag opt
report_summary -directory $_REPORTS_PATH/map/${DESIGN}_opt.rpt

############### Reports ###############
report dft_setup > ${_REPORTS_PATH}/${DESIGN}-DFTsetup_final
report_timing > ${_REPORTS_PATH}/${DESIGN}_report_timing.rpt
report_timing -lint > ${_REPORTS_PATH}/${DESIGN}_report_timing_lint.rpt
report_power > ${_REPORTS_PATH}/${DESIGN}_report_power.rpt
report_area > ${_REPORTS_PATH}/${DESIGN}_report_area.rpt
report_gates > ${_REPORTS_PATH}/${DESIGN}_report_cell.rpt
report_qor > ${_REPORTS_PATH}/${DESIGN}_report_qor.rpt
report_scan_chains > ${_REPORTS_PATH}/${DESIGN}_report_scan_chains.rpt

############### Outputs ###############
write_dft_atpg -library
"/opt/cadence/tsmc_018_6M20K/tcb018gbwp7t_270a_FE/tcb018gbwp7t_270a_vlg/
TSMCHOME/digital/Front_End/verilog/tcb018gbwp7t_270a/tcb018gbwp7t.v" -
directory ./et_atpg_output $DESIGN
write_hdl -mapped > ${_OUTPUTS_PATH}/${DESIGN}_dft_netlist.v
write_do_lec -revised_design ${DESIGN}_dft_netlist.v >
   ${_OUTPUTS_PATH}/${DESIGN}_dft.lec.do
```

```
97  write_sdc > ${_OUTPUTS_PATH}/${DESIGN}_dft_sdc.sdc
98  write_sdf -timescale ns -nonegchecks -recrem split -edges check_edge
        -setuphold split > ${_OUTPUTS_PATH}/${DESIGN}_dft_delays.sdf
99  write_scandef > ${_OUTPUTS_PATH}/${DESIGN}_dft_scanDEF.scandef
100
101 puts "================================="
102 puts "Synthesis DFT Finished ........."
103 puts "================================="
```

Apêndice C – *picorv32.sdc*

```
#####################################################################
## Logical / Physical synthesis constraints ##
## Ref: GAPH/FACIN/PUCRS ##
#####################################################################

## DEFINE VARS
set_load_unit -picofarads 1

create_clock -name {clk} -period 10.0 [get_ports {clk}]
set_false_path -from [get_ports {resetn}]

## INPUTS
# Maior delay de um INV12
set_input_delay -clock clk -max 0.8 [all_inputs]
# Valor de um INV12, no caso tt1p8v25c, valores tplh e tphl
set_input_transition -min -rise 0.041 [all_inputs]
set_input_transition -max -rise 1.034 [all_inputs]
set_input_transition -min -fall 0.016 [all_inputs]
set_input_transition -max -fall 0.564 [all_inputs]

## OUTPUTS
# Valor de capacitancia de entrada de um INV1S, e de um INV12
set_load -min 0.0036 [all_outputs]
set_load -max 0.0632 [all_outputs]
# Maior delay de um INV12
set_output_delay -clock clk -max 0.8 [all_outputs]
```

Apêndice D – *picosoc.v*

```
\begin{figure}
    \centering
    \includegraphics[width=1\linewidth]{figuras/rtl_1.png}
    \caption{Enter Caption}
    \label{fig:enter-label}
\end{figure}
/*
 *  PicoSoC - A simple example SoC using PicoRV32
 *
 *  Copyright (C) 2017  Claire Xenia Wolf <claire@yosyshq.com>
 *
 *  Permission to use, copy, modify, and/or distribute this software for any
 *  purpose with or without fee is hereby granted, provided that the above
 *  copyright notice and this permission notice appear in all copies.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 *  WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 *  MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 *  ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 *  WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 *  ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 *  OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */

`ifndef PICORV32_REGS
//`ifdef PICORV32_V
//`error "picosoc.v must be read before picorv32.v!"
//`endif

`define PICORV32_REGS picosoc_regs
`endif

`ifndef PICOSOC_MEM
`define PICOSOC_MEM picosoc_mem
`endif

// this macro can be used to check if the verilog files in your
// design are read in the correct order.
`define PICOSOC_V

module picosoc (
```

```
43    input clk,
44    input resetn,
45
46    output        iomem_valid,
47    input         iomem_ready,
48    output [ 3:0] iomem_wstrb,
49    output [31:0] iomem_addr,
50    output [31:0] iomem_wdata,
51    input  [31:0] iomem_rdata,
52
53    input  irq_5,
54    input  irq_6,
55    input  irq_7,
56
57    output ser_tx,
58    input  ser_rx,
59
60    output flash_csb,
61    output flash_clk,
62
63    output flash_io0_oe,
64    output flash_io1_oe,
65    output flash_io2_oe,
66    output flash_io3_oe,
67
68    output flash_io0_do,
69    output flash_io1_do,
70    output flash_io2_do,
71    output flash_io3_do,
72
73    input  flash_io0_di,
74    input  flash_io1_di,
75    input  flash_io2_di,
76    input  flash_io3_di
77  );
78    parameter [0:0] BARREL_SHIFTER = 1;
79    parameter [0:0] ENABLE_MUL = 1;
80    parameter [0:0] ENABLE_DIV = 1;
81    parameter [0:0] ENABLE_FAST_MUL = 0;
82    parameter [0:0] ENABLE_COMPRESSED = 1;
83    parameter [0:0] ENABLE_COUNTERS = 1;
84    parameter [0:0] ENABLE_IRQ_QREGS = 0;
85
86    parameter integer MEM_WORDS = 256;
87    parameter [31:0] STACKADDR = (4*MEM_WORDS);       // end of memory
88    parameter [31:0] PROGADDR_RESET = 32'h 0010_0000; // 1 MB into flash
89    parameter [31:0] PROGADDR_IRQ = 32'h 0000_0000;
90
91    reg [31:0] irq;
92    wire irq_stall = 0;
93    wire irq_uart = 0;
94
95    always @* begin
```

```
96      irq = 0;
97      irq[3] = irq_stall;
98      irq[4] = irq_uart;
99      irq[5] = irq_5;
100     irq[6] = irq_6;
101     irq[7] = irq_7;
102   end
103
104   wire mem_valid;
105   wire mem_instr;
106   wire mem_ready;
107   wire [31:0] mem_addr;
108   wire [31:0] mem_wdata;
109   wire [3:0] mem_wstrb;
110   wire [31:0] mem_rdata;
111
112   wire spimem_ready;
113   wire [31:0] spimem_rdata;
114
115   reg ram_ready;
116   wire [31:0] ram_rdata;
117
118   assign iomem_valid = mem_valid && (mem_addr[31:24] > 8'h 01);
119   assign iomem_wstrb = mem_wstrb;
120   assign iomem_addr = mem_addr;
121   assign iomem_wdata = mem_wdata;
122
123   wire spimemio_cfgreg_sel = mem_valid && (mem_addr == 32'h 0200_0000);
124   wire [31:0] spimemio_cfgreg_do;
125
126   wire        simpleuart_reg_div_sel = mem_valid && (mem_addr == 32'h
        0200_0004);
127   wire [31:0] simpleuart_reg_div_do;
128
129   wire        simpleuart_reg_dat_sel = mem_valid && (mem_addr == 32'h
        0200_0008);
130   wire        saida_y1 = mem_valid && (mem_addr == 32'h 0200_001C);
131   wire        saida_y2 = mem_valid && (mem_addr == 32'h 0200_0020);
132   wire        saida_y3 = mem_valid && (mem_addr == 32'h 0200_0024);
133
134     wire        x1_valid = mem_valid && (mem_addr == 32'h 0200_000C);
135     wire        x2_valid = mem_valid && (mem_addr == 32'h 0200_0010);
136     wire        x3_valid = mem_valid && (mem_addr == 32'h 0200_0014);
137     wire        x4_valid = mem_valid && (mem_addr == 32'h 0200_0018);
138
139   wire [31:0] simpleuart_reg_dat_do;
140   wire        simpleuart_reg_dat_wait;
141
142     reg [26:0] x1; // mem_valid && mem_addr == 32'h 0200_000C;
143
144     reg [26:0] x2; // mem_valid && mem_addr == 32'h 0200_000C;+4
145     reg [26:0] x3; // mem_valid && mem_addr == 32'h 0200_000C;+4
146     reg [26:0] x4; // mem_valid && mem_addr == 32'h 0200_000C;+4
```

```
147     wire [26:0] y1;
148     wire [26:0] y2;
149     wire [26:0] y3;
150     wire [26:0] x_in [0:3];
151     reg start_neuronio;
152     wire ready_neuronio;
153 
154 
155   assign mem_ready = (iomem_valid && iomem_ready) || spimem_ready ||
        ram_ready || spimemio_cfgreg_sel ||
156       simpleuart_reg_div_sel || (simpleuart_reg_dat_sel &&
          !simpleuart_reg_dat_wait) || x1_valid ||
157       x2_valid || x3_valid || x4_valid || saida_y1 || saida_y2 ||
          saida_y3;
158 
159 
160   assign mem_rdata = (iomem_valid && iomem_ready) ? iomem_rdata :
        spimem_ready ? spimem_rdata : ram_ready ? ram_rdata :
161       spimemio_cfgreg_sel ? spimemio_cfgreg_do : simpleuart_reg_div_sel
          ? simpleuart_reg_div_do :
162       simpleuart_reg_dat_sel ? simpleuart_reg_dat_do : saida_y1 ?
          {5'h0,y1}: saida_y2 ? {5'h0,y2}:
163       saida_y3 ? {5'h0,y3}: 32'h 0000_0000;
164 
165   picorv32 #(
166     .STACKADDR(STACKADDR),
167     .PROGADDR_RESET(PROGADDR_RESET),
168     .PROGADDR_IRQ(PROGADDR_IRQ),
169     .BARREL_SHIFTER(BARREL_SHIFTER),
170     .COMPRESSED_ISA(ENABLE_COMPRESSED),
171     .ENABLE_COUNTERS(ENABLE_COUNTERS),
172     .ENABLE_MUL(ENABLE_MUL),
173     .ENABLE_DIV(ENABLE_DIV),
174     .ENABLE_FAST_MUL(ENABLE_FAST_MUL),
175     .ENABLE_IRQ(1),
176     .ENABLE_IRQ_QREGS(ENABLE_IRQ_QREGS)
177   ) cpu (
178     .clk         (clk        ),
179     .resetn      (resetn     ),
180     .mem_valid   (mem_valid  ),
181     .mem_instr   (mem_instr  ),
182     .mem_ready   (mem_ready  ),
183     .mem_addr    (mem_addr   ),
184     .mem_wdata   (mem_wdata  ),
185     .mem_wstrb   (mem_wstrb  ),
186     .mem_rdata   (mem_rdata  ),
187     .irq         (irq        )
188   );
189 
190 //    ila_0 ila (
191 //        .clk(clk),
192 //        .probe0({flash_io3_di, flash_io2_di, flash_io1_di,
      flash_io0_di, flash_io3_do, flash_io2_do, flash_io1_do, flash_io0_do,
```

```
    flash_io3_oe, flash_io2_oe, flash_io1_oe, flash_io0_oe}),
//        .probe1(mem_addr),
//        .probe2(spimem_rdata),
//        .probe3({clk, resetn, mem_valid && mem_addr >= 4*MEM_WORDS &&
    mem_addr < 32'h 0200_0000, flash_csb, flash_clk, spimemio.state,
    spimemio.xfer.next_ibuffer, spimemio.xfer.ibuffer})
//    );

  spimemio spimemio (
    .clk    (clk),
    .resetn (resetn),
    .valid  (mem_valid && mem_addr >= 4*MEM_WORDS && mem_addr < 32'h
       0200_0000),
    .ready  (spimem_ready),
    .addr   (mem_addr[23:0]),
    .rdata  (spimem_rdata),

    .flash_csb    (flash_csb   ),
    .flash_clk    (flash_clk   ),

    .flash_io0_oe (flash_io0_oe),
    .flash_io1_oe (flash_io1_oe),
    .flash_io2_oe (flash_io2_oe),
    .flash_io3_oe (flash_io3_oe),

    .flash_io0_do (flash_io0_do),
    .flash_io1_do (flash_io1_do),
    .flash_io2_do (flash_io2_do),
    .flash_io3_do (flash_io3_do),

    .flash_io0_di (flash_io0_di),
    .flash_io1_di (flash_io1_di),
    .flash_io2_di (flash_io2_di),
    .flash_io3_di (flash_io3_di),

    .cfgreg_we(spimemio_cfgreg_sel ? mem_wstrb : 4'b 0000),
    .cfgreg_di(mem_wdata),
    .cfgreg_do(spimemio_cfgreg_do)
  );

    always @(posedge clk) begin

        if (~resetn) begin
            x1<=0;
            x2<=0;
            x3<=0;
            x4<=0;
        end else begin
            start_neuronio = 0;
            if (mem_valid && mem_addr == 32'h 0200_000C) begin
              if (mem_wstrb[0]) x1[ 7: 0] <= mem_wdata[ 7: 0];
              if (mem_wstrb[1]) x1[15: 8] <= mem_wdata[15: 8];
              if (mem_wstrb[2]) x1[23:16] <= mem_wdata[23:16];
```

```
242               if (mem_wstrb[3]) x1[26:24] <= mem_wdata[26:24];
243             end
244
245             if (mem_valid && mem_addr == 32'h 0200_0010) begin
246               if (mem_wstrb[0]) x2[ 7: 0] <= mem_wdata[ 7: 0];
247               if (mem_wstrb[1]) x2[15: 8] <= mem_wdata[15: 8];
248               if (mem_wstrb[2]) x2[23:16] <= mem_wdata[23:16];
249               if (mem_wstrb[3]) x2[26:24] <= mem_wdata[26:24];
250             end
251
252             if (mem_valid && mem_addr == 32'h 0200_0014) begin
253               if (mem_wstrb[0]) x3[ 7: 0] <= mem_wdata[ 7: 0];
254               if (mem_wstrb[1]) x3[15: 8] <= mem_wdata[15: 8];
255               if (mem_wstrb[2]) x3[23:16] <= mem_wdata[23:16];
256               if (mem_wstrb[3]) x3[26:24] <= mem_wdata[26:24];
257             end
258
259             if (mem_valid && mem_addr == 32'h 0200_0018) begin
260               if (mem_wstrb[0]) x4[ 7: 0] <= mem_wdata[ 7: 0];
261               if (mem_wstrb[1]) x4[15: 8] <= mem_wdata[15: 8];
262               if (mem_wstrb[2]) x4[23:16] <= mem_wdata[23:16];
263               if (mem_wstrb[3]) x4[26:24] <= mem_wdata[26:24];
264               start_neuronio = 1;
265             end
266         end
267       end
268
269
270         // mem_valid && !mem_ready && mem_addr < 4*MEM_WORDS;
271
272         MLP443 neuronio(
273                .reset   (~resetn),
274                .clk     (clk),
275                .start   (start_neuronio),
276                .x1      (x1),
277                .x2      (x2),
278                .x3      (x3),
279                .x4      (x4),
280                .y1      (y1),
281                .y2      (y2),
282                .y3      (y3),
283                .ready   (ready_neuronio)
284         );
285
286       simpleuart simpleuart (
287         .clk         (clk         ),
288         .resetn      (resetn     ),
289
290         .ser_tx      (ser_tx      ),
291         .ser_rx      (ser_rx      ),
292
293         .reg_div_we  (simpleuart_reg_div_sel ? mem_wstrb : 4'b 0000),
294         .reg_div_di  (mem_wdata),
```

```
295     .reg_div_do  (simpleuart_reg_div_do),
296
297     .reg_dat_we  (simpleuart_reg_dat_sel ? mem_wstrb[0] : 1'b 0),
298     .reg_dat_re  (simpleuart_reg_dat_sel && !mem_wstrb),
299     .reg_dat_di  (mem_wdata),
300     .reg_dat_do  (simpleuart_reg_dat_do),
301     .reg_dat_wait(simpleuart_reg_dat_wait)
302   );
303
304   always @(posedge clk)
305     ram_ready <= mem_valid && !mem_ready && mem_addr < 4*MEM_WORDS;
306
307   `PICOSOC_MEM #(
308     .WORDS(MEM_WORDS)
309   ) memory (
310     .clk(clk),
311     .wen((mem_valid && !mem_ready && mem_addr < 4*MEM_WORDS) ? mem_wstrb
          : 4'b0),
312     .addr(mem_addr[23:2]),
313     .wdata(mem_wdata),
314     .rdata(ram_rdata)
315   );
316 endmodule
317
318 // Implementation note:
319 // Replace the following two modules with wrappers for your SRAM cells.
320
321 module picosoc_regs (
322   input clk, wen,
323   input [5:0] waddr,
324   input [5:0] raddr1,
325   input [5:0] raddr2,
326   input [31:0] wdata,
327   output [31:0] rdata1,
328   output [31:0] rdata2
329 );
330   reg [31:0] regs [0:31];
331
332   always @(posedge clk)
333     if (wen) regs[waddr[4:0]] <= wdata;
334
335   assign rdata1 = regs[raddr1[4:0]];
336   assign rdata2 = regs[raddr2[4:0]];
337 endmodule
338
339 module picosoc_mem #(
340   parameter integer WORDS = 256
341 ) (
342   input clk,
343   input [3:0] wen,
344   input [21:0] addr,
345   input [31:0] wdata,
346   output reg [31:0] rdata
```

```
);
  reg [31:0] mem [0:WORDS-1];

  always @(posedge clk) begin
    rdata <= mem[addr];
    if (wen[0]) mem[addr][ 7: 0] <= wdata[ 7: 0];
    if (wen[1]) mem[addr][15: 8] <= wdata[15: 8];
    if (wen[2]) mem[addr][23:16] <= wdata[23:16];
    if (wen[3]) mem[addr][31:24] <= wdata[31:24];
  end
endmodule
```

Apêndice E – *firmware.c*

```
/*
 *  PicoSoC - A simple example SoC using PicoRV32
 *
 *  Copyright (C) 2017  Claire Xenia Wolf <claire@yosyshq.com>
 *
 *  Permission to use, copy, modify, and/or distribute this software for any
 *  purpose with or without fee is hereby granted, provided that the above
 *  copyright notice and this permission notice appear in all copies.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 *  WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 *  MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 *  ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 *  WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 *  ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 *  OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */

#include <stdint.h>
#include <stdbool.h>

#ifdef ICEBREAKER
#  define MEM_TOTAL 0x20000 /* 128 KB */
#elif HX8KDEMO
#  define MEM_TOTAL 0x200 /* 2 KB */
#else
#  error "Set -DICEBREAKER or -DHX8KDEMO when compiling firmware.c"
#endif

// make icebreaker_fw.bin
// a pointer to this is a null pointer, but the compiler does not
// know that because "sram" is a linker symbol from sections.lds.
extern uint32_t sram;

#define reg_spictrl (*(volatile uint32_t*)0x02000000)
#define reg_uart_clkdiv (*(volatile uint32_t*)0x02000004)
#define reg_uart_data (*(volatile uint32_t*)0x02000008)
#define reg_leds (*(volatile uint32_t*)0x03000000)
#define x1 (*(volatile uint32_t*)0x0200000C)
#define x2 (*(volatile uint32_t*)0x02000010)
#define x3 (*(volatile uint32_t*)0x02000014)
```

```
43 #define x4 (*(volatile uint32_t*)0x02000018)
44 #define y1 (*(volatile uint32_t*)0x0200001C)
45 #define y2 (*(volatile uint32_t*)0x02000020)
46 #define y3 (*(volatile uint32_t*)0x02000024)
47
48 void putchar(char c)
49 {
50   if (c == '\n')
51     putchar('\r');
52   reg_uart_data = c;
53 }
54
55 void print(const char *p)
56 {
57   while (*p)
58     putchar(*(p++));
59 }
60
61 void print_hex(uint32_t v, int digits)
62 {
63   for (int i = 7; i >= 0; i--) {
64     char c = "0123456789abcdef"[(v >> (4*i)) & 15];
65     if (c == '0' && i >= digits) continue;
66     putchar(c);
67     digits = i;
68   }
69 }
70
71 void main()
72 {
73   reg_leds = 31;
74   reg_uart_clkdiv = 868;
75   x1 = 0b01000000101100110011001001;
76   x2 = 0b01000000001000000000000000;
77   x3 = 0b01000000011110011001001100;
78   x4 = 0b00111111100011001100110110;
79
80   int y1_aux;
81   int y2_aux;
82   int y3_aux;
83
84   y1_aux = y1;
85   y2_aux = y2;
86   y3_aux = y3;
87
88   print ("y1= ");
89   print_hex (y1_aux,8);
90   print("\n");
91
92   print ("y2 = ");
93   print_hex (y2_aux,8);
94   print("\n");
95
```

```
96    print ("y3 = ");
97    print_hex (y3_aux,8);
98    print("\n");
99
100 }
```

Apêndice F – *MLP443.vhd*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.fpupack.all;
use work.rnapack.all;

entity MLP443 is
    Port ( reset : in STD_LOGIC;
           clk : in STD_LOGIC;
           start : in STD_LOGIC;
           x1 : in std_logic_vector(26 downto 0);
           x2 : in std_logic_vector(26 downto 0);
           x3 : in std_logic_vector(26 downto 0);
           x4 : in std_logic_vector(26 downto 0);
           y1 : out std_logic_vector(26 downto 0);
           y2 : out std_logic_vector(26 downto 0);
           y3 : out std_logic_vector(26 downto 0);
           ready : out STD_LOGIC);
end MLP443;

architecture Behavioral of MLP443 is

component neuronio_o is
    Port ( reset : in STD_LOGIC;
           clk : in STD_LOGIC;
           x : in array1D_h;
           w : in array1D_h;
           bias : in STD_LOGIC_VECTOR (FP_WIDTH-1 downto 0);
           start : in STD_LOGIC;
           saida : out STD_LOGIC_VECTOR (FP_WIDTH-1 downto 0);
           ready_pl : out STD_LOGIC;
           ready : out STD_LOGIC);
end component;

component neuronio_h is
    Port ( reset : in STD_LOGIC;
           clk : in STD_LOGIC;
           x : in array1D_in;
           w : in array1D_in;
           bias : in STD_LOGIC_VECTOR (FP_WIDTH-1 downto 0);
           start : in STD_LOGIC;
           saida : out STD_LOGIC_VECTOR (FP_WIDTH-1 downto 0);
           ready_pl : out STD_LOGIC;
           ready : out STD_LOGIC);
end component;

signal entradas : array1D_in;
signal y_out : array1D_h;
```

```
signal readyh : std_logic_vector(0 to num_h-1) := (others=>'0');
signal readyo : std_logic_vector(0 to num_out-1) := (others=>'0');
signal readyplh : std_logic_vector(0 to num_h-1) := (others=>'0');
signal readyplo : std_logic_vector(0 to num_out-1) := (others=>'0');

signal out_nh : array1D_h := (others=>(others=>'0'));
begin

entradas <= (x1,x2,x3,x4);
y1 <= y_out(0);
y2 <= y_out(1);
y3 <= y_out(2);

    par_nh: FOR i IN 1 to num_h GENERATE
        nh: neuronio_h port map(
            reset       => reset,
            clk         => clk,
            x           => entradas,
            w           => w_inh(i-1),
            bias        => biash(i-1),
            start       => start,
            saida       => out_nh(i-1),
            ready_pl    => readyplh(i-1),
            ready       => readyh(i-1));
    END GENERATE;

    par_no: FOR i IN 1 to num_out GENERATE
        no: neuronio_o port map(
            reset       => reset,
            clk         => clk,
            x           => out_nh,
            w           => w_hout(i-1),
            bias        => biaso(i-1),
            start       => readyh(0),
            saida       => y_out(i-1),
            ready_pl    => readyplo(i-1),
            ready       => readyo(i-1));
    END GENERATE;
    ready <= readyo(1);

end Behavioral;
```

Apêndice G – *MAKEFILE*

```
BASENAME:=picorv32_wrapper
PDK_PATH:=/opt/cadence/tsmc_018_6M20K/tcb018gbwp7t_270a_FE

sim:
  cd xcelium; \
  rm -rf work*/* PS_Demo*.sdf.X *.log .simvision *.dsn *.trn *.vcd wave* *.mem; \
  xmvlog -work work_sub ../src/*.v; \
  xmvlog -work work ../tb/*.v; \
  xmelab -work work -timescale '1ns/1ps' -status work.${BASENAME}_tb -access wrc; \
  xmsim -gui -status work.${BASENAME}_tb:module

sim_ps:
  cd xcelium; \
  rm -rf work*/* PS_Demo*.sdf.X *.log .simvision *.dsn *.trn *.vcd wave* *.mem; \
  \
  xmvlog -work work_lib -cdslib ./cds.lib -logfile log/xmvlog_postsyn.log \
    -errormax 15 -update -linedebug -status -define DISPLAY_PD_PU_EN \
    ${PDK_PATH}/tcb018gbwp7t_270a_vlg/TSMCHOME/digital/Front_End/verilog/tcb018gbwp7t_270a/tcb018gbwp7t.v; \
  \
  xmvlog -work work_sub ../genus/outputs/${BASENAME}_netlist.v; \
  xmvlog -work work ../tb/*.v; \
  xmelab -work work -timescale '1ns/1ps' \
       -sdf_file ../genus/outputs/${DESIGN}_delays.sdf \
       -status work.${BASENAME}_tb  -access wrc; \
  xmsim  -gui -status work.${BASENAME}_tb:module

sim_ps_dft:
  cd xcelium; \
  rm -rf work*/* PS_Demo*.sdf.X *.log .simvision *.dsn *.trn *.vcd wave* *.mem; \
  \
  xmvlog -work work_lib -cdslib ./cds.lib -logfile log/xmvlog_postsyn.log \
    -errormax 15 -update -linedebug -status -define DISPLAY_PD_PU_EN \
    ${PDK_PATH}/tcb018gbwp7t_270a_vlg/TSMCHOME/digital/Front_End/verilog/tcb018gbwp7t_270a/tcb018gbwp7t.v; \
  \
  xmvlog -work work_sub ../genus/outputs_dft/${BASENAME}_dft_netlist.v; \
  xmvlog -define DFT_netlist -work work ../tb/*.v; \
  xmelab -work work -timescale '1ns/1ps' \
       -sdf_file ../genus/outputs_dft/${BASENAME}_dft_delays.sdf \
       -status work.${BASENAME}_tb  -access wrc; \
  xmsim  -gui -status work.${BASENAME}_tb:module
```

```
43
44 sim_atpg:
45   cd genus; \
46     xrun -gui -access rwc +xmstatus +xm64bit
47     +TESTFILE1=./et_atpg_output/testresults/verilog \
48     /VER.FULLSCAN.picorv32_wrapper_atpg.data.scan.ex1.ts1.verilog
49     +TESTFILE2=./et_atpg_output/testresults/verilog \
50     /VER.FULLSCAN.picorv32_wrapper_atpg.data.logic.ex1.ts2.verilog
51     +HEARTBEAT +FAILSET +xmtimescale+1ns/1ps +xmoverride_timescale
52     +xmseq_udp_delay+2ps +libext+.v+.V+.z+.Z+.gz
53     +xmlibdirname+./et_atpg_output/Inca_libs_13_49_43
54     -l ./et_atpg_output/ncverilog_FULLSCAN.log
55     -v /opt/cadence/tsmc_018_6M20K/tcb018gbwp7t_270a_FE \
56     /tcb018gbwp7t_270a_vlg/TSMCHOME/digital/Front_End \
57     /verilog/tcb018gbwp7t_270a/tcb018gbwp7t.v
58     ./et_atpg_output/picorv32_wrapper.test_netlist.v
59     ./et_atpg_output/testresults/verilog \
60     /VER.FULLSCAN.picorv32_wrapper_atpg.mainsim.v
61
62 synthesis_dft:
63   cd genus; \
64   genus -gui -files scripts/genus_dft.tcl -log log/synthesis_dft; \
65
66 synthesis:
67   cd genus; \
68   genus -gui -files scripts/genus.tcl -log log/synthesis; \
69
70 modus:
71   cd genus; \
72   modus -gui -files et_atpg_output/runmodus.atpg.tcl
73
74 .PHONY: help
75 help:
76   @echo
77   @echo "-------------- Cadence Makefile ---------------"
78   @echo "-----------------------------------------------"
79   @echo "-- For simulation, run:                      -"
80   @echo "--       make sim                           -"
81   @echo "--                                          -"
82   @echo "-- For simulation post synthesis, run:       -"
83   @echo "--        make sim_ps                       -"
84   @echo "--                                          -"
85   @echo "-- For simulation post synthesis dft, run:   -"
86   @echo "--        make sim_ps_dft                   -"
87   @echo "--                                          -"
88   @echo "-- For atpg simulation, run:                -"
89   @echo "--       make sim_atpg                      -"
90   @echo "--                                          -"
91   @echo "-- For synthesis, run:                      -"
92   @echo "--        make synthesis                    -"
93   @echo "--                                          -"
94   @echo "-- For synthesis DFT, run:                  -"
95   @echo "--        make synthesis_dft                -"
```

```
96    @echo "--                                                -"
97    @echo "-- For reseting all synthesis files, run:     -"
98    @echo "--         make clean_synthesis                 -"
99    @echo "-------------------------------------------------"
100   @echo
101
102 .PHONY: clean_synthesis
103 clean_synthesis:
104   cd genus; \
105   rm -rf reports/* reports_dft/* outputs/* outputs_dft/*
          et_atpg_output/* fv/* log/*
106
107 .PHONY: clean_simulation
108 clean_simulation:
109   cd xcelium; \
110   rm -rf work*/* PS_Demo*.sdf.X *.log .simvision *.dsn *.trn *.vcd wave*
          *.mem; \
111
112 .PHONY: clean_all
113 clean_all: clean_simulation clean_synthesis
```

Apêndice H – *picorv32_wrapper_tb.v*

```
1  // This is free and unencumbered software released into the public
       domain.
2  //
3  // Anyone is free to copy, modify, publish, use, compile, sell, or
4  // distribute this software, either in source code form or as a compiled
5  // binary, for any purpose, commercial or non-commercial, and by any
6  // means.
7
8  `timescale 1 ns / 1 ps
9  `define COMPRESSED_ISA
10 `ifndef VERILATOR
11 module picorv32_wrapper_tb #(
12   parameter AXI_TEST = 0,
13   parameter VERBOSE = 0
14 );
15   reg clk = 1;
16   reg resetn = 0;
17   wire trap;
18
19   always #5 clk = ~clk;
20
21   initial begin
22     repeat (100) @(posedge clk);
23     resetn <= 1;
24   end
25
26   initial begin
27     if ($test$plusargs("vcd")) begin
28       $dumpfile("testbench.vcd");
29       $dumpvars(0, picorv32_wrapper_tb);
30     end
31     repeat (1000000) @(posedge clk);
32     $display("TIMEOUT");
33     $finish;
34   end
35
36   wire trace_valid;
37   wire [35:0] trace_data;
38   integer trace_file;
39
40   initial begin
41     if ($test$plusargs("trace")) begin
42       trace_file = $fopen("testbench.trace", "w");
43       repeat (10) @(posedge clk);
44       while (!trap) begin
45         @(posedge clk);
46         if (trace_valid)
47           $fwrite(trace_file, "%x\n", trace_data);
```

```
48        end
49        $fclose(trace_file);
50        $display("Finished writing testbench.trace.");
51      end
52    end
53
54    picorv32_wrap #(
55      .AXI_TEST (AXI_TEST),
56      .VERBOSE  (VERBOSE)
57    ) top (
58      .clk(clk),
59      .resetn(resetn),
60      .trap(trap),
61      .trace_valid(trace_valid),
62      .trace_data(trace_data)
63    );
64  endmodule
65  `endif
66
67  module picorv32_wrap #(
68    parameter AXI_TEST = 0,
69    parameter VERBOSE = 0
70  ) (
71    input clk,
72    input resetn,
73    output trap,
74    output trace_valid,
75    output [35:0] trace_data
76  );
77    wire tests_passed;
78    reg [31:0] irq = 0;
79
80    reg [15:0] count_cycle = 0;
81    always @(posedge clk) count_cycle <= resetn ? count_cycle + 1 : 0;
82
83    always @* begin
84      irq = 0;
85      irq[4] = &count_cycle[12:0];
86      irq[5] = &count_cycle[15:0];
87    end
88
89    wire        mem_axi_awvalid;
90    wire        mem_axi_awready;
91    wire [31:0] mem_axi_awaddr;
92    wire [ 2:0] mem_axi_awprot;
93
94    wire        mem_axi_wvalid;
95    wire        mem_axi_wready;
96    wire [31:0] mem_axi_wdata;
97    wire [ 3:0] mem_axi_wstrb;
98
99    wire        mem_axi_bvalid;
100   wire        mem_axi_bready;
```

```
101
102   wire        mem_axi_arvalid;
103   wire        mem_axi_arready;
104   wire [31:0] mem_axi_araddr;
105   wire [ 2:0] mem_axi_arprot;
106
107   wire        mem_axi_rvalid;
108   wire        mem_axi_rready;
109   wire [31:0] mem_axi_rdata;
110
111   axi4_memory #(
112     .AXI_TEST (AXI_TEST),
113     .VERBOSE  (VERBOSE)
114   ) mem (
115     .clk             (clk             ),
116     .mem_axi_awvalid (mem_axi_awvalid ),
117     .mem_axi_awready (mem_axi_awready ),
118     .mem_axi_awaddr  (mem_axi_awaddr  ),
119     .mem_axi_awprot  (mem_axi_awprot  ),
120
121     .mem_axi_wvalid  (mem_axi_wvalid  ),
122     .mem_axi_wready  (mem_axi_wready  ),
123     .mem_axi_wdata   (mem_axi_wdata   ),
124     .mem_axi_wstrb   (mem_axi_wstrb   ),
125
126     .mem_axi_bvalid  (mem_axi_bvalid  ),
127     .mem_axi_bready  (mem_axi_bready  ),
128
129     .mem_axi_arvalid (mem_axi_arvalid ),
130     .mem_axi_arready (mem_axi_arready ),
131     .mem_axi_araddr  (mem_axi_araddr  ),
132     .mem_axi_arprot  (mem_axi_arprot  ),
133
134     .mem_axi_rvalid  (mem_axi_rvalid  ),
135     .mem_axi_rready  (mem_axi_rready  ),
136     .mem_axi_rdata   (mem_axi_rdata   ),
137
138     .tests_passed    (tests_passed    )
139   );
140
141
142   picorv32_wrapper uut (
143     .clk            (clk            ),
144     .resetn         (resetn         ),
145     .trap           (trap           ),
146     .mem_axi_awvalid(mem_axi_awvalid),
147     .mem_axi_awready(mem_axi_awready),
148     .mem_axi_awaddr (mem_axi_awaddr ),
149     .mem_axi_awprot (mem_axi_awprot ),
150     .mem_axi_wvalid (mem_axi_wvalid ),
151     .mem_axi_wready (mem_axi_wready ),
152     .mem_axi_wdata  (mem_axi_wdata  ),
153     .mem_axi_wstrb  (mem_axi_wstrb  ),
```

```
154     .mem_axi_bvalid (mem_axi_bvalid ),
155     .mem_axi_bready (mem_axi_bready ),
156     .mem_axi_arvalid(mem_axi_arvalid),
157     .mem_axi_arready(mem_axi_arready),
158     .mem_axi_araddr (mem_axi_araddr ),
159     .mem_axi_arprot (mem_axi_arprot ),
160     .mem_axi_rvalid (mem_axi_rvalid ),
161     .mem_axi_rready (mem_axi_rready ),
162     .mem_axi_rdata  (mem_axi_rdata  ),
163     .irq            (irq            ),
164     .trace_valid    (trace_valid    ),
165     .trace_data     (trace_data     )
166   `ifdef DFT_netlist
167     ,
168     .se     (0     ),
169     .scan_in     (DISCONNECTED     )
170   `endif
171   );
172
173   reg [1023:0] firmware_file;
174   initial begin
175     if (!$value$plusargs("firmware=%s", firmware_file))
176       firmware_file = "../firmware/firmware.hex";
177     $readmemh(firmware_file, mem.memory);
178   end
179
180   integer cycle_counter;
181   always @(posedge clk) begin
182     cycle_counter <= resetn ? cycle_counter + 1 : 0;
183     if (resetn && trap) begin
184 `ifndef VERILATOR
185       repeat (10) @(posedge clk);
186 `endif
187       $display("TRAP after %1d clock cycles", cycle_counter);
188       if (tests_passed) begin
189         $display("ALL TESTS PASSED.");
190         $finish;
191       end else begin
192         $display("ERROR!");
193         if ($test$plusargs("noerror"))
194           $finish;
195         $stop;
196       end
197     end
198   end
199
200 endmodule
201
202 module axi4_memory #(
203   parameter AXI_TEST = 0,
204   parameter VERBOSE = 0
205 ) (
206   /* verilator lint_off MULTIDRIVEN */
```

```
207
208   input             clk,
209   input             mem_axi_awvalid,
210   output reg        mem_axi_awready,
211   input      [31:0] mem_axi_awaddr,
212   input      [ 2:0] mem_axi_awprot,
213
214   input             mem_axi_wvalid,
215   output reg        mem_axi_wready,
216   input      [31:0] mem_axi_wdata,
217   input      [ 3:0] mem_axi_wstrb,
218
219   output reg        mem_axi_bvalid,
220   input             mem_axi_bready,
221
222   input             mem_axi_arvalid,
223   output reg        mem_axi_arready,
224   input      [31:0] mem_axi_araddr,
225   input      [ 2:0] mem_axi_arprot,
226
227   output reg        mem_axi_rvalid,
228   input             mem_axi_rready,
229   output reg [31:0] mem_axi_rdata,
230
231   output reg        tests_passed
232 );
233   reg [31:0]   memory [0:4096-1] /* verilator public */;
234   reg verbose;
235   initial verbose = $test$plusargs("verbose") || VERBOSE;
236
237   reg axi_test;
238   initial axi_test = $test$plusargs("axi_test") || AXI_TEST;
239
240   initial begin
241     mem_axi_awready = 0;
242     mem_axi_wready = 0;
243     mem_axi_bvalid = 0;
244     mem_axi_arready = 0;
245     mem_axi_rvalid = 0;
246     tests_passed = 0;
247   end
248
249   reg [63:0] xorshift64_state = 64'd88172645463325252;
250
251   task xorshift64_next;
252     begin
253       // see page 4 of Marsaglia, George (July 2003). "Xorshift RNGs".
          Journal of Statistical Software 8 (14).
254       xorshift64_state = xorshift64_state ^ (xorshift64_state << 13);
255       xorshift64_state = xorshift64_state ^ (xorshift64_state >>  7);
256       xorshift64_state = xorshift64_state ^ (xorshift64_state << 17);
257     end
258   endtask
```

```
259
260   reg [2:0] fast_axi_transaction = ~0;
261   reg [4:0] async_axi_transaction = ~0;
262   reg [4:0] delay_axi_transaction = 0;
263
264   always @(posedge clk) begin
265     if (axi_test) begin
266         xorshift64_next;
267         {fast_axi_transaction, async_axi_transaction,
              delay_axi_transaction} <= xorshift64_state;
268     end
269   end
270
271   reg latched_raddr_en = 0;
272   reg latched_waddr_en = 0;
273   reg latched_wdata_en = 0;
274
275   reg fast_raddr = 0;
276   reg fast_waddr = 0;
277   reg fast_wdata = 0;
278
279   reg [31:0] latched_raddr;
280   reg [31:0] latched_waddr;
281   reg [31:0] latched_wdata;
282   reg [ 3:0] latched_wstrb;
283   reg        latched_rinsn;
284
285   task handle_axi_arvalid; begin
286     mem_axi_arready <= 1;
287     latched_raddr = mem_axi_araddr;
288     latched_rinsn = mem_axi_arprot[2];
289     latched_raddr_en = 1;
290     fast_raddr <= 1;
291   end endtask
292
293   task handle_axi_awvalid; begin
294     mem_axi_awready <= 1;
295     latched_waddr = mem_axi_awaddr;
296     latched_waddr_en = 1;
297     fast_waddr <= 1;
298   end endtask
299
300   task handle_axi_wvalid; begin
301     mem_axi_wready <= 1;
302     latched_wdata = mem_axi_wdata;
303     latched_wstrb = mem_axi_wstrb;
304     latched_wdata_en = 1;
305     fast_wdata <= 1;
306   end endtask
307
308   task handle_axi_rvalid; begin
309     if (verbose)
```

```
310       $display("RD: ADDR=%08x DATA=%08x%s", latched_raddr,
              memory[latched_raddr >> 2], latched_rinsn ? " INSN" : "");
311     if (latched_raddr < 128*1024) begin
312       mem_axi_rdata <= memory[latched_raddr >> 2];
313       mem_axi_rvalid <= 1;
314       latched_raddr_en = 0;
315     end else begin
316       $display("OUT-OF-BOUNDS MEMORY READ FROM %08x", latched_raddr);
317       $finish;
318     end
319   end endtask
320
321   task handle_axi_bvalid; begin
322     if (verbose)
323       $display("WR: ADDR=%08x DATA=%08x STRB=%04b", latched_waddr,
              latched_wdata, latched_wstrb);
324     if (latched_waddr < 128*1024) begin
325       if (latched_wstrb[0]) memory[latched_waddr >> 2][ 7: 0] <=
              latched_wdata[ 7: 0];
326       if (latched_wstrb[1]) memory[latched_waddr >> 2][15: 8] <=
              latched_wdata[15: 8];
327       if (latched_wstrb[2]) memory[latched_waddr >> 2][23:16] <=
              latched_wdata[23:16];
328       if (latched_wstrb[3]) memory[latched_waddr >> 2][31:24] <=
              latched_wdata[31:24];
329     end else
330     if (latched_waddr == 32'h1000_0000) begin
331       if (verbose) begin
332         if (32 <= latched_wdata && latched_wdata < 128)
333           $display("OUT: '%c'", latched_wdata[7:0]);
334         else
335           $display("OUT: %3d", latched_wdata);
336       end else begin
337         $write("%c", latched_wdata[7:0]);
338 `ifndef VERILATOR
339         $fflush();
340 `endif
341       end
342     end else
343     if (latched_waddr == 32'h2000_0000) begin
344       if (latched_wdata == 123456789)
345         tests_passed = 1;
346     end else begin
347       $display("OUT-OF-BOUNDS MEMORY WRITE TO %08x", latched_waddr);
348       $finish;
349     end
350     mem_axi_bvalid <= 1;
351     latched_waddr_en = 0;
352     latched_wdata_en = 0;
353   end endtask
354
355   always @(negedge clk) begin
```

```
356     if (mem_axi_arvalid && !(latched_raddr_en || fast_raddr) &&
            async_axi_transaction[0]) handle_axi_arvalid;
357     if (mem_axi_awvalid && !(latched_waddr_en || fast_waddr) &&
            async_axi_transaction[1]) handle_axi_awvalid;
358     if (mem_axi_wvalid  && !(latched_wdata_en || fast_wdata) &&
            async_axi_transaction[2]) handle_axi_wvalid;
359     if (!mem_axi_rvalid && latched_raddr_en && async_axi_transaction[3])
            handle_axi_rvalid;
360     if (!mem_axi_bvalid && latched_waddr_en && latched_wdata_en &&
            async_axi_transaction[4]) handle_axi_bvalid;
361   end
362
363   always @(posedge clk) begin
364     mem_axi_arready <= 0;
365     mem_axi_awready <= 0;
366     mem_axi_wready <= 0;
367
368     fast_raddr <= 0;
369     fast_waddr <= 0;
370     fast_wdata <= 0;
371
372     if (mem_axi_rvalid && mem_axi_rready) begin
373       mem_axi_rvalid <= 0;
374     end
375
376     if (mem_axi_bvalid && mem_axi_bready) begin
377       mem_axi_bvalid <= 0;
378     end
379
380     if (mem_axi_arvalid && mem_axi_arready && !fast_raddr) begin
381       latched_raddr = mem_axi_araddr;
382       latched_rinsn = mem_axi_arprot[2];
383       latched_raddr_en = 1;
384     end
385
386     if (mem_axi_awvalid && mem_axi_awready && !fast_waddr) begin
387       latched_waddr = mem_axi_awaddr;
388       latched_waddr_en = 1;
389     end
390
391     if (mem_axi_wvalid && mem_axi_wready && !fast_wdata) begin
392       latched_wdata = mem_axi_wdata;
393       latched_wstrb = mem_axi_wstrb;
394       latched_wdata_en = 1;
395     end
396
397     if (mem_axi_arvalid && !(latched_raddr_en || fast_raddr) &&
            !delay_axi_transaction[0]) handle_axi_arvalid;
398     if (mem_axi_awvalid && !(latched_waddr_en || fast_waddr) &&
            !delay_axi_transaction[1]) handle_axi_awvalid;
399     if (mem_axi_wvalid  && !(latched_wdata_en || fast_wdata) &&
            !delay_axi_transaction[2]) handle_axi_wvalid;
400
```

```
401     if (!mem_axi_rvalid && latched_raddr_en &&
            !delay_axi_transaction[3]) handle_axi_rvalid;
402     if (!mem_axi_bvalid && latched_waddr_en && latched_wdata_en &&
            !delay_axi_transaction[4]) handle_axi_bvalid;
403   end
404 endmodule
```

Anexos

Anexo A – *runmodus.atpg.tcl*

```
1
2
3  # Template script intended to jumpstart the user to run Modus Automatic
       Test Pattern Generation (ATPG) software.
4
5  #  Generated by Genus(TM) Synthesis Solution 17.10-p007_1
6
7  # Modus ATPG Script
8
9  #-------------------------------------------------------------
10 # INITIALIZE VARIABLES
11 #-------------------------------------------------------------
12 # "set_db workdir" is recommended to be set first to initialize where
       the Database is to be located
13 # WORKDIR Tcl variable can be used in the script to reference file
       locations
14 set    WORKDIR ./et_atpg_output;              # Define Tcl variable
15 set_db workdir $WORKDIR;                      # Specify the directory
       for the ATPG database
16
17 # Configure the database
18 set_option stdout     summary;                                  # Only
       print a summary for each command to the terminal
19                                                          # Logs contain
                                                               the
                                                               complete
                                                               command
                                                               output
20 set ::env(CDS_LIC_REPORT) yes
21 # test_checks.tcl provides check_log proc to stop script on message
       severity
22 source  $::env(Install_Dir)/bin/64bit/test_checks.tcl;    # Load
       test_checks.tcl from Modus installation - provides check_log below
23 set STOP_ON_MSG_SEV ERROR;                               # Messagee
       severity for check_log to stop at (ERROR, WARNING_SEVERE, WARNING)
24 set LOGDIR          $WORKDIR/testresults/logs;          # Location of
       logs for check_log
25
26 file delete -force $WORKDIR/tbdata;                      # Delete Test
       Database
27 file delete -force $WORKDIR/testresults;                 # Delete Test
       Output Files/Logs
28
29
30
31 #-------------------------------------------------------------
32 # BUILD THE LOGIC MODEL
33 #-------------------------------------------------------------
```

```
build_model \
    -cell  picorv32_wrapper \
    -techlib   /opt/cadence/tsmc_018_6M20K/tcb018gbwp7t_270a_FE/
    tcb018gbwp7t_270a_vlg/TSMCHOME/digital/Front_End/verilog/
    tcb018gbwp7t_270a/tcb018gbwp7t.v \
    -designsource  $WORKDIR/picorv32_wrapper.test_netlist.v \
    -blackboxoutputs   z \
    -allowmissingmodules   no \

check_log log_build_model

#-------------------------------------------------------------
# BUILD THE TEST MODEL
#-------------------------------------------------------------
build_testmode \
    -testmode  FULLSCAN \
    -assignfile  $WORKDIR/picorv32_wrapper.FULLSCAN.pinassign \
    -modedef   FULLSCAN \

check_log log_build_testmode_FULLSCAN

#-------------------------------------------------------------
# REPORT THE TEST MODEL
#-------------------------------------------------------------
report_test_structures \
    -testmode  FULLSCAN \

check_log log_report_test_structures_FULLSCAN

#-------------------------------------------------------------
# VERIFY THE TEST MODEL
#-------------------------------------------------------------
verify_test_structures \
    -messagecount  TSV-016=10,TSV-024=10,TSV-315=10,TSV-027=10 \
    -testmode  FULLSCAN \

check_log log_verify_test_structures_FULLSCAN

#-------------------------------------------------------------
# BUILD THE FAULT MODEL
#-------------------------------------------------------------
build_faultmodel \
    -includedynamic  no \

check_log log_build_faultmodel

#-------------------------------------------------------------
# ATPG - TEST GENERATION
#-------------------------------------------------------------
create_scanchain_tests \
    -experiment  picorv32_wrapper_atpg \
    -testmode  FULLSCAN \

```

```
create_logic_tests \
    -experiment  picorv32_wrapper_atpg \
    -testmode  FULLSCAN

check_log log_create_logic_tests_FULLSCAN_picorv32_wrapper_atpg

#-------------------------------------------------------------
# ATPG - Report the Scan and Capture Switching
#-------------------------------------------------------------
write_toggle_gram \
    -experiment  picorv32_wrapper_atpg \
    -testmode  FULLSCAN \

#-------------------------------------------------------------
# VERILOG VECTORS - For PARALLEL Simulation
#-------------------------------------------------------------
write_vectors \
    -inexperiment  picorv32_wrapper_atpg \
    -testmode  FULLSCAN \
    -language  verilog \
    -scanformat  parallel \

check_log log_write_vectors_FULLSCAN_picorv32_wrapper_atpg

#-------------------------------------------------------------
# ATPG - Save Experiment to the Master Database for the Testmode
#-------------------------------------------------------------
commit_tests \
    -inexperiment  picorv32_wrapper_atpg \
    -testmode  FULLSCAN

check_log log_commit_tests_FULLSCAN_picorv32_wrapper_atpg

exit
```

Anexo B – *icebreaker.v*

```
/*
 *  PicoSoC - A simple example SoC using PicoRV32
 *
 *  Copyright (C) 2017  Claire Xenia Wolf <claire@yosyshq.com>
 *
 *  Permission to use, copy, modify, and/or distribute this software for any
 *  purpose with or without fee is hereby granted, provided that the above
 *  copyright notice and this permission notice appear in all copies.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 *  WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 *  MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 *  ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 *  WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 *  ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 *  OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */


module nexys4 (
  input clk,

  output ser_tx,
  input ser_rx,

  output led1,
  output led2,
  output led3,
  output led4,
  output led5,

  output ledr_n,
  output ledg_n,

  output flash_csb,
//  output flash_clk,
  inout  flash_io0,
  inout  flash_io1,
  inout  flash_io2,
  inout  flash_io3,
  input reset
```

```
43
44 );
45   parameter integer MEM_WORDS = 32768;
46
47   reg [5:0] reset_cnt = 0;
48   wire resetn = &reset_cnt & (!reset);
49
50   always @(posedge clk) begin
51     reset_cnt <= reset_cnt + !resetn;
52   end
53
54   wire [7:0] leds;
55
56   assign led1 = leds[1];
57   assign led2 = leds[2];
58   assign led3 = leds[3];
59   assign led4 = leds[4];
60   assign led5 = leds[5];
61
62   assign ledr_n = !leds[6];
63   assign ledg_n = !leds[7];
64
65   wire flash_io0_oe, flash_io0_do, flash_io0_di;
66   wire flash_io1_oe, flash_io1_do, flash_io1_di;
67   wire flash_io2_oe, flash_io2_do, flash_io2_di;
68   wire flash_io3_oe, flash_io3_do, flash_io3_di;
69
70 //  SB_IO #(
71 //    .PIN_TYPE(6'b 1010_01),
72 //    .PULLUP(1'b 0)
73 //  ) flash_io_buf [3:0] (
74 //    .PACKAGE_PIN({flash_io3, flash_io2, flash_io1, flash_io0}),
75 //    .OUTPUT_ENABLE({flash_io3_oe, flash_io2_oe, flash_io1_oe,
       flash_io0_oe}),
76 //    .D_OUT_0({flash_io3_do, flash_io2_do, flash_io1_do, flash_io0_do}),
77 //    .D_IN_0({flash_io3_di, flash_io2_di, flash_io1_di, flash_io0_di})
78 //  );
79
80   IOBUF #(
81        .DRIVE(12), // Specify the output drive strength
82        .IBUF_LOW_PWR("TRUE"),  // Low Power - "TRUE", High Performance =
           "FALSE"
83        .IOSTANDARD("DEFAULT"), // Specify the I/O standard
84        .SLEW("FAST") // Specify the output slew rate
85    ) flash_io_buf [3:0] (
86        .O({flash_io3_di, flash_io2_di, flash_io1_di, flash_io0_di}),
           // Buffer output
87        .IO({flash_io3, flash_io2, flash_io1, flash_io0}),   // Buffer
           inout port (connect directly to top-level port)
88        .I({flash_io3_do, flash_io2_do, flash_io1_do, flash_io0_do}),
           // Buffer input
89        .T({!flash_io3_oe, !flash_io2_oe, !flash_io1_oe, !flash_io0_oe})
                // 3-state enable input, high=input, low=output
```

```
90      );
91
92
93      STARTUPE2 #(
94         .PROG_USR("FALSE"),  // Activate program event security feature.
              Requires encrypted bitstreams.
95         .SIM_CCLK_FREQ(0.0)  // Set the Configuration Clock Frequency(ns)
              for simulation.
96      )
97      STARTUPE2_inst (
98         .CFGCLK(/* NC */),       // 1-bit output: Configuration main
              clock output
99         .CFGMCLK(/* NC */),     // 1-bit output: Configuration internal
              oscillator clock output
100        .EOS(/* NC */),             // 1-bit output: Active high output
              signal indicating the End Of Startup.
101        .PREQ(/* NC */),           // 1-bit output: PROGRAM request to
              fabric output
102        .CLK(0),              // 1-bit input: User start-up clock input
103        .GSR(0),              // 1-bit input: Global Set/Reset input (GSR
              cannot be used for the port name)
104        .GTS(0),              // 1-bit input: Global 3-state input (GTS
              cannot be used for the port name)
105        .KEYCLEARB(0), // 1-bit input: Clear AES Decrypter Key input from
              Battery-Backed RAM (BBRAM)
106        .PACK(0),           // 1-bit input: PROGRAM acknowledge input
107        .USRCCLKO(flash_clk),   // 1-bit input: User CCLK input
108                              // For Zynq-7000 devices, this input must
                                    be tied to GND
109        .USRCCLKTS(0), // 1-bit input: User CCLK 3-state enable input
110                              // For Zynq-7000 devices, this input must
                                    be tied to VCC
111        .USRDONEO(0),   // 1-bit input: User DONE pin output control
112        .USRDONETS(1)  // 1-bit input: User DONE 3-state enable output
113     );
114
115
116  wire        iomem_valid;
117  reg         iomem_ready;
118  wire [3:0]  iomem_wstrb;
119  wire [31:0] iomem_addr;
120  wire [31:0] iomem_wdata;
121  reg  [31:0] iomem_rdata;
122
123  reg [31:0] gpio;
124  assign leds = gpio;
125
126  always @(posedge clk) begin
127    if (!resetn) begin
128      gpio <= 0;
129    end else begin
130      iomem_ready <= 0;
```

```
131       if (iomem_valid && !iomem_ready && iomem_addr[31:24] == 8'h 03)
              begin
132         iomem_ready <= 1;
133         iomem_rdata <= gpio;
134         if (iomem_wstrb[0]) gpio[ 7: 0] <= iomem_wdata[ 7: 0];
135         if (iomem_wstrb[1]) gpio[15: 8] <= iomem_wdata[15: 8];
136         if (iomem_wstrb[2]) gpio[23:16] <= iomem_wdata[23:16];
137         if (iomem_wstrb[3]) gpio[31:24] <= iomem_wdata[31:24];
138       end
139     end
140   end
141
142   picosoc #(
143     .BARREL_SHIFTER(0),
144     .ENABLE_MUL(0),
145     .ENABLE_DIV(0),
146     .ENABLE_FAST_MUL(1),
147     .MEM_WORDS(MEM_WORDS)
148   ) soc (
149     .clk          (clk         ),
150     .resetn       (resetn      ),
151
152     .ser_tx       (ser_tx      ),
153     .ser_rx       (ser_rx      ),
154
155     .flash_csb    (flash_csb   ),
156     .flash_clk    (flash_clk   ),
157
158     .flash_io0_oe (flash_io0_oe),
159     .flash_io1_oe (flash_io1_oe),
160     .flash_io2_oe (flash_io2_oe),
161     .flash_io3_oe (flash_io3_oe),
162
163     .flash_io0_do (flash_io0_do),
164     .flash_io1_do (flash_io1_do),
165     .flash_io2_do (flash_io2_do),
166     .flash_io3_do (flash_io3_do),
167
168     .flash_io0_di (flash_io0_di),
169     .flash_io1_di (flash_io1_di),
170     .flash_io2_di (flash_io2_di),
171     .flash_io3_di (flash_io3_di),
172
173     .irq_5        (1'b0        ),
174     .irq_6        (1'b0        ),
175     .irq_7        (1'b0        ),
176
177     .iomem_valid  (iomem_valid ),
178     .iomem_ready  (iomem_ready ),
179     .iomem_wstrb  (iomem_wstrb ),
180     .iomem_addr   (iomem_addr  ),
181     .iomem_wdata  (iomem_wdata ),
182     .iomem_rdata  (iomem_rdata )
```

```
183   );
184 endmodule
```

Anexo C – *picorv32.v*

```
/*
 *  PicoRV32 -- A Small RISC-V (RV32I) Processor Core
 *
 *  Copyright (C) 2015  Claire Xenia Wolf <claire@yosyshq.com>
 *
 *  Permission to use, copy, modify, and/or distribute this software for any
 *  purpose with or without fee is hereby granted, provided that the above
 *  copyright notice and this permission notice appear in all copies.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 *  WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 *  MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 *  ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 *  WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 *  ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 *  OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */

/* verilator lint_off WIDTH */
/* verilator lint_off PINMISSING */
/* verilator lint_off CASEOVERLAP */
/* verilator lint_off CASEINCOMPLETE */

`timescale 1 ns / 1 ps
// `default_nettype none
// `define DEBUGNETS
// `define DEBUGREGS
// `define DEBUGASM
// `define DEBUG

`ifdef DEBUG
  `define debug(debug_command) debug_command
`else
  `define debug(debug_command)
`endif

`ifdef FORMAL
  `define FORMAL_KEEP (* keep *)
  `define assert(assert_expr) assert(assert_expr)
`else
  `ifdef DEBUGNETS
```

```
43      `define FORMAL_KEEP (* keep *)
44    `else
45      `define FORMAL_KEEP
46    `endif
47    `define assert(assert_expr) empty_statement
48  `endif
49
50  // uncomment this for register file in extra module
51  // `define PICORV32_REGS picorv32_regs
52
53  // this macro can be used to check if the verilog files in your
54  // design are read in the correct order.
55  `define PICORV32_V
56
57
58  /***************************************************************
59   * picorv32
60   ***************************************************************/
61
62  module picorv32 #(
63    parameter [ 0:0] ENABLE_COUNTERS = 1,
64    parameter [ 0:0] ENABLE_COUNTERS64 = 1,
65    parameter [ 0:0] ENABLE_REGS_16_31 = 1,
66    parameter [ 0:0] ENABLE_REGS_DUALPORT = 1,
67    parameter [ 0:0] LATCHED_MEM_RDATA = 0,
68    parameter [ 0:0] TWO_STAGE_SHIFT = 1,
69    parameter [ 0:0] BARREL_SHIFTER = 0,
70    parameter [ 0:0] TWO_CYCLE_COMPARE = 0,
71    parameter [ 0:0] TWO_CYCLE_ALU = 0,
72    parameter [ 0:0] COMPRESSED_ISA = 0,
73    parameter [ 0:0] CATCH_MISALIGN = 1,
74    parameter [ 0:0] CATCH_ILLINSN = 1,
75    parameter [ 0:0] ENABLE_PCPI = 0,
76    parameter [ 0:0] ENABLE_MUL = 0,
77    parameter [ 0:0] ENABLE_FAST_MUL = 0,
78    parameter [ 0:0] ENABLE_DIV = 0,
79    parameter [ 0:0] ENABLE_IRQ = 0,
80    parameter [ 0:0] ENABLE_IRQ_QREGS = 1,
81    parameter [ 0:0] ENABLE_IRQ_TIMER = 1,
82    parameter [ 0:0] ENABLE_TRACE = 0,
83    parameter [ 0:0] REGS_INIT_ZERO = 0,
84    parameter [31:0] MASKED_IRQ = 32'h 0000_0000,
85    parameter [31:0] LATCHED_IRQ = 32'h ffff_ffff,
86    parameter [31:0] PROGADDR_RESET = 32'h 0000_0000,
87    parameter [31:0] PROGADDR_IRQ = 32'h 0000_0010,
88    parameter [31:0] STACKADDR = 32'h ffff_ffff
89  ) (
90    input clk, resetn,
91    output reg trap,
92
93    output reg        mem_valid,
94    output reg        mem_instr,
95    input             mem_ready,
```

```
96
97    output reg [31:0] mem_addr,
98    output reg [31:0] mem_wdata,
99    output reg [ 3:0] mem_wstrb,
100   input      [31:0] mem_rdata,
101
102   // Look-Ahead Interface
103   output            mem_la_read,
104   output            mem_la_write,
105   output     [31:0] mem_la_addr,
106   output reg [31:0] mem_la_wdata,
107   output reg [ 3:0] mem_la_wstrb,
108
109   // Pico Co-Processor Interface (PCPI)
110   output reg        pcpi_valid,
111   output reg [31:0] pcpi_insn,
112   output     [31:0] pcpi_rs1,
113   output     [31:0] pcpi_rs2,
114   input             pcpi_wr,
115   input      [31:0] pcpi_rd,
116   input             pcpi_wait,
117   input             pcpi_ready,
118
119   // IRQ Interface
120   input      [31:0] irq,
121   output reg [31:0] eoi,
122
123 `ifdef RISCV_FORMAL
124   output reg        rvfi_valid,
125   output reg [63:0] rvfi_order,
126   output reg [31:0] rvfi_insn,
127   output reg        rvfi_trap,
128   output reg        rvfi_halt,
129   output reg        rvfi_intr,
130   output reg [ 1:0] rvfi_mode,
131   output reg [ 1:0] rvfi_ixl,
132   output reg [ 4:0] rvfi_rs1_addr,
133   output reg [ 4:0] rvfi_rs2_addr,
134   output reg [31:0] rvfi_rs1_rdata,
135   output reg [31:0] rvfi_rs2_rdata,
136   output reg [ 4:0] rvfi_rd_addr,
137   output reg [31:0] rvfi_rd_wdata,
138   output reg [31:0] rvfi_pc_rdata,
139   output reg [31:0] rvfi_pc_wdata,
140   output reg [31:0] rvfi_mem_addr,
141   output reg [ 3:0] rvfi_mem_rmask,
142   output reg [ 3:0] rvfi_mem_wmask,
143   output reg [31:0] rvfi_mem_rdata,
144   output reg [31:0] rvfi_mem_wdata,
145
146   output reg [63:0] rvfi_csr_mcycle_rmask,
147   output reg [63:0] rvfi_csr_mcycle_wmask,
148   output reg [63:0] rvfi_csr_mcycle_rdata,
```

```
149   output reg [63:0] rvfi_csr_mcycle_wdata,
150
151   output reg [63:0] rvfi_csr_minstret_rmask,
152   output reg [63:0] rvfi_csr_minstret_wmask,
153   output reg [63:0] rvfi_csr_minstret_rdata,
154   output reg [63:0] rvfi_csr_minstret_wdata,
155 `endif
156
157   // Trace Interface
158   output reg         trace_valid,
159   output reg [35:0] trace_data
160 );
161   localparam integer irq_timer = 0;
162   localparam integer irq_ebreak = 1;
163   localparam integer irq_buserror = 2;
164
165   localparam integer irqregs_offset = ENABLE_REGS_16_31 ? 32 : 16;
166   localparam integer regfile_size = (ENABLE_REGS_16_31 ? 32 : 16) +
         4*ENABLE_IRQ*ENABLE_IRQ_QREGS;
167   localparam integer regindex_bits = (ENABLE_REGS_16_31 ? 5 : 4) +
         ENABLE_IRQ*ENABLE_IRQ_QREGS;
168
169   localparam WITH_PCPI = ENABLE_PCPI || ENABLE_MUL || ENABLE_FAST_MUL ||
         ENABLE_DIV;
170
171   localparam [35:0] TRACE_BRANCH = {4'b 0001, 32'b 0};
172   localparam [35:0] TRACE_ADDR   = {4'b 0010, 32'b 0};
173   localparam [35:0] TRACE_IRQ    = {4'b 1000, 32'b 0};
174
175   reg [63:0] count_cycle, count_instr;
176   reg [31:0] reg_pc, reg_next_pc, reg_op1, reg_op2, reg_out;
177   reg [4:0] reg_sh;
178
179   reg [31:0] next_insn_opcode;
180   reg [31:0] dbg_insn_opcode;
181   reg [31:0] dbg_insn_addr;
182
183   wire dbg_mem_valid = mem_valid;
184   wire dbg_mem_instr = mem_instr;
185   wire dbg_mem_ready = mem_ready;
186   wire [31:0] dbg_mem_addr  = mem_addr;
187   wire [31:0] dbg_mem_wdata = mem_wdata;
188   wire [ 3:0] dbg_mem_wstrb = mem_wstrb;
189   wire [31:0] dbg_mem_rdata = mem_rdata;
190
191   assign pcpi_rs1 = reg_op1;
192   assign pcpi_rs2 = reg_op2;
193
194   wire [31:0] next_pc;
195
196   reg irq_delay;
197   reg irq_active;
198   reg [31:0] irq_mask;
```

```
199   reg [31:0] irq_pending;
200   reg [31:0] timer;
201
202 `ifndef PICORV32_REGS
203   reg [31:0] cpuregs [0:regfile_size-1];
204
205   integer i;
206   initial begin
207     if (REGS_INIT_ZERO) begin
208       for (i = 0; i < regfile_size; i = i+1)
209         cpuregs[i] = 0;
210     end
211   end
212 `endif
213
214   task empty_statement;
215     // This task is used by the `assert directive in non-formal mode to
216     // avoid empty statement (which are unsupported by plain Verilog
          syntax).
217     begin end
218   endtask
219
220 `ifdef DEBUGREGS
221   wire [31:0] dbg_reg_x0  = 0;
222   wire [31:0] dbg_reg_x1  = cpuregs[1];
223   wire [31:0] dbg_reg_x2  = cpuregs[2];
224   wire [31:0] dbg_reg_x3  = cpuregs[3];
225   wire [31:0] dbg_reg_x4  = cpuregs[4];
226   wire [31:0] dbg_reg_x5  = cpuregs[5];
227   wire [31:0] dbg_reg_x6  = cpuregs[6];
228   wire [31:0] dbg_reg_x7  = cpuregs[7];
229   wire [31:0] dbg_reg_x8  = cpuregs[8];
230   wire [31:0] dbg_reg_x9  = cpuregs[9];
231   wire [31:0] dbg_reg_x10 = cpuregs[10];
232   wire [31:0] dbg_reg_x11 = cpuregs[11];
233   wire [31:0] dbg_reg_x12 = cpuregs[12];
234   wire [31:0] dbg_reg_x13 = cpuregs[13];
235   wire [31:0] dbg_reg_x14 = cpuregs[14];
236   wire [31:0] dbg_reg_x15 = cpuregs[15];
237   wire [31:0] dbg_reg_x16 = cpuregs[16];
238   wire [31:0] dbg_reg_x17 = cpuregs[17];
239   wire [31:0] dbg_reg_x18 = cpuregs[18];
240   wire [31:0] dbg_reg_x19 = cpuregs[19];
241   wire [31:0] dbg_reg_x20 = cpuregs[20];
242   wire [31:0] dbg_reg_x21 = cpuregs[21];
243   wire [31:0] dbg_reg_x22 = cpuregs[22];
244   wire [31:0] dbg_reg_x23 = cpuregs[23];
245   wire [31:0] dbg_reg_x24 = cpuregs[24];
246   wire [31:0] dbg_reg_x25 = cpuregs[25];
247   wire [31:0] dbg_reg_x26 = cpuregs[26];
248   wire [31:0] dbg_reg_x27 = cpuregs[27];
249   wire [31:0] dbg_reg_x28 = cpuregs[28];
250   wire [31:0] dbg_reg_x29 = cpuregs[29];
```

```
251   wire [31:0] dbg_reg_x30 = cpuregs[30];
252   wire [31:0] dbg_reg_x31 = cpuregs[31];
253 `endif
254
255   // Internal PCPI Cores
256
257   wire        pcpi_mul_wr;
258   wire [31:0] pcpi_mul_rd;
259   wire        pcpi_mul_wait;
260   wire        pcpi_mul_ready;
261
262   wire        pcpi_div_wr;
263   wire [31:0] pcpi_div_rd;
264   wire        pcpi_div_wait;
265   wire        pcpi_div_ready;
266
267   reg        pcpi_int_wr;
268   reg [31:0] pcpi_int_rd;
269   reg        pcpi_int_wait;
270   reg        pcpi_int_ready;
271
272   generate if (ENABLE_FAST_MUL) begin
273     picorv32_pcpi_fast_mul pcpi_mul (
274       .clk       (clk            ),
275       .resetn    (resetn         ),
276       .pcpi_valid(pcpi_valid     ),
277       .pcpi_insn (pcpi_insn      ),
278       .pcpi_rs1  (pcpi_rs1       ),
279       .pcpi_rs2  (pcpi_rs2       ),
280       .pcpi_wr   (pcpi_mul_wr    ),
281       .pcpi_rd   (pcpi_mul_rd    ),
282       .pcpi_wait (pcpi_mul_wait  ),
283       .pcpi_ready(pcpi_mul_ready )
284     );
285   end else if (ENABLE_MUL) begin
286     picorv32_pcpi_mul pcpi_mul (
287       .clk       (clk            ),
288       .resetn    (resetn         ),
289       .pcpi_valid(pcpi_valid     ),
290       .pcpi_insn (pcpi_insn      ),
291       .pcpi_rs1  (pcpi_rs1       ),
292       .pcpi_rs2  (pcpi_rs2       ),
293       .pcpi_wr   (pcpi_mul_wr    ),
294       .pcpi_rd   (pcpi_mul_rd    ),
295       .pcpi_wait (pcpi_mul_wait  ),
296       .pcpi_ready(pcpi_mul_ready )
297     );
298   end else begin
299     assign pcpi_mul_wr = 0;
300     assign pcpi_mul_rd = 32'bx;
301     assign pcpi_mul_wait = 0;
302     assign pcpi_mul_ready = 0;
303   end endgenerate
```

```
304
305   generate if (ENABLE_DIV) begin
306     picorv32_pcpi_div pcpi_div (
307       .clk       (clk            ),
308       .resetn    (resetn         ),
309       .pcpi_valid(pcpi_valid     ),
310       .pcpi_insn (pcpi_insn      ),
311       .pcpi_rs1  (pcpi_rs1       ),
312       .pcpi_rs2  (pcpi_rs2       ),
313       .pcpi_wr   (pcpi_div_wr    ),
314       .pcpi_rd   (pcpi_div_rd    ),
315       .pcpi_wait (pcpi_div_wait  ),
316       .pcpi_ready(pcpi_div_ready )
317     );
318   end else begin
319     assign pcpi_div_wr = 0;
320     assign pcpi_div_rd = 32'bx;
321     assign pcpi_div_wait = 0;
322     assign pcpi_div_ready = 0;
323   end endgenerate
324
325   always @* begin
326     pcpi_int_wr = 0;
327     pcpi_int_rd = 32'bx;
328     pcpi_int_wait  = |{ENABLE_PCPI && pcpi_wait,  (ENABLE_MUL ||
          ENABLE_FAST_MUL) && pcpi_mul_wait,  ENABLE_DIV && pcpi_div_wait};
329     pcpi_int_ready = |{ENABLE_PCPI && pcpi_ready, (ENABLE_MUL ||
          ENABLE_FAST_MUL) && pcpi_mul_ready, ENABLE_DIV && pcpi_div_ready};
330
331     (* parallel_case *)
332     case (1'b1)
333       ENABLE_PCPI && pcpi_ready: begin
334         pcpi_int_wr = ENABLE_PCPI ? pcpi_wr : 0;
335         pcpi_int_rd = ENABLE_PCPI ? pcpi_rd : 0;
336       end
337       (ENABLE_MUL || ENABLE_FAST_MUL) && pcpi_mul_ready: begin
338         pcpi_int_wr = pcpi_mul_wr;
339         pcpi_int_rd = pcpi_mul_rd;
340       end
341       ENABLE_DIV && pcpi_div_ready: begin
342         pcpi_int_wr = pcpi_div_wr;
343         pcpi_int_rd = pcpi_div_rd;
344       end
345     endcase
346   end
347
348
349   // Memory Interface
350
351   reg [1:0] mem_state;
352   reg [1:0] mem_wordsize;
353   reg [31:0] mem_rdata_word;
354   reg [31:0] mem_rdata_q;
```

```
355   reg mem_do_prefetch;
356   reg mem_do_rinst;
357   reg mem_do_rdata;
358   reg mem_do_wdata;
359
360   wire mem_xfer;
361   reg mem_la_secondword, mem_la_firstword_reg, last_mem_valid;
362   wire mem_la_firstword = COMPRESSED_ISA && (mem_do_prefetch ||
          mem_do_rinst) && next_pc[1] && !mem_la_secondword;
363   wire mem_la_firstword_xfer = COMPRESSED_ISA && mem_xfer &&
          (!last_mem_valid ? mem_la_firstword : mem_la_firstword_reg);
364
365   reg prefetched_high_word;
366   reg clear_prefetched_high_word;
367   reg [15:0] mem_16bit_buffer;
368
369   wire [31:0] mem_rdata_latched_noshuffle;
370   wire [31:0] mem_rdata_latched;
371
372   wire mem_la_use_prefetched_high_word = COMPRESSED_ISA &&
          mem_la_firstword && prefetched_high_word &&
          !clear_prefetched_high_word;
373   assign mem_xfer = (mem_valid && mem_ready) ||
          (mem_la_use_prefetched_high_word && mem_do_rinst);
374
375   wire mem_busy = |{mem_do_prefetch, mem_do_rinst, mem_do_rdata,
          mem_do_wdata};
376   wire mem_done = resetn && ((mem_xfer && |mem_state && (mem_do_rinst ||
          mem_do_rdata || mem_do_wdata)) || (&mem_state && mem_do_rinst)) &&
377        (!mem_la_firstword || (~&mem_rdata_latched[1:0] && mem_xfer));
378
379   assign mem_la_write = resetn && !mem_state && mem_do_wdata;
380   assign mem_la_read = resetn && ((!mem_la_use_prefetched_high_word &&
          !mem_state && (mem_do_rinst || mem_do_prefetch || mem_do_rdata)) ||
381        (COMPRESSED_ISA && mem_xfer && (!last_mem_valid ? mem_la_firstword
             : mem_la_firstword_reg) && !mem_la_secondword &&
             &mem_rdata_latched[1:0]));
382   assign mem_la_addr = (mem_do_prefetch || mem_do_rinst) ?
          {next_pc[31:2] + mem_la_firstword_xfer, 2'b00} : {reg_op1[31:2],
          2'b00};
383
384   assign mem_rdata_latched_noshuffle = (mem_xfer || LATCHED_MEM_RDATA) ?
          mem_rdata : mem_rdata_q;
385
386   assign mem_rdata_latched = COMPRESSED_ISA &&
          mem_la_use_prefetched_high_word ? {16'bx, mem_16bit_buffer} :
387        COMPRESSED_ISA && mem_la_secondword ?
             {mem_rdata_latched_noshuffle[15:0], mem_16bit_buffer} :
388        COMPRESSED_ISA && mem_la_firstword ? {16'bx,
             mem_rdata_latched_noshuffle[31:16]} :
             mem_rdata_latched_noshuffle;
389
390   always @(posedge clk) begin
```

```
391     if (!resetn) begin
392       mem_la_firstword_reg <= 0;
393       last_mem_valid <= 0;
394     end else begin
395       if (!last_mem_valid)
396         mem_la_firstword_reg <= mem_la_firstword;
397       last_mem_valid <= mem_valid && !mem_ready;
398     end
399   end
400
401   always @* begin
402     (* full_case *)
403     case (mem_wordsize)
404       0: begin
405         mem_la_wdata = reg_op2;
406         mem_la_wstrb = 4'b1111;
407         mem_rdata_word = mem_rdata;
408       end
409       1: begin
410         mem_la_wdata = {2{reg_op2[15:0]}};
411         mem_la_wstrb = reg_op1[1] ? 4'b1100 : 4'b0011;
412         case (reg_op1[1])
413           1'b0: mem_rdata_word = {16'b0, mem_rdata[15: 0]};
414           1'b1: mem_rdata_word = {16'b0, mem_rdata[31:16]};
415         endcase
416       end
417       2: begin
418         mem_la_wdata = {4{reg_op2[7:0]}};
419         mem_la_wstrb = 4'b0001 << reg_op1[1:0];
420         case (reg_op1[1:0])
421           2'b00: mem_rdata_word = {24'b0, mem_rdata[ 7: 0]};
422           2'b01: mem_rdata_word = {24'b0, mem_rdata[15: 8]};
423           2'b10: mem_rdata_word = {24'b0, mem_rdata[23:16]};
424           2'b11: mem_rdata_word = {24'b0, mem_rdata[31:24]};
425         endcase
426       end
427     endcase
428   end
429
430   always @(posedge clk) begin
431     if (mem_xfer) begin
432       mem_rdata_q <= COMPRESSED_ISA ? mem_rdata_latched : mem_rdata;
433       next_insn_opcode <= COMPRESSED_ISA ? mem_rdata_latched : mem_rdata;
434     end
435
436     if (COMPRESSED_ISA && mem_done && (mem_do_prefetch || mem_do_rinst))
        begin
437       case (mem_rdata_latched[1:0])
438         2'b00: begin // Quadrant 0
439           case (mem_rdata_latched[15:13])
440             3'b000: begin // C.ADDI4SPN
441               mem_rdata_q[14:12] <= 3'b000;
```

```
442                 mem_rdata_q[31:20] <= {2'b0, mem_rdata_latched[10:7],
                        mem_rdata_latched[12:11], mem_rdata_latched[5],
                        mem_rdata_latched[6], 2'b00};
443               end
444               3'b010: begin // C.LW
445                 mem_rdata_q[31:20] <= {5'b0, mem_rdata_latched[5],
                        mem_rdata_latched[12:10], mem_rdata_latched[6], 2'b00};
446                 mem_rdata_q[14:12] <= 3'b 010;
447               end
448               3'b 110: begin // C.SW
449                 {mem_rdata_q[31:25], mem_rdata_q[11:7]} <= {5'b0,
                        mem_rdata_latched[5], mem_rdata_latched[12:10],
                        mem_rdata_latched[6], 2'b00};
450                 mem_rdata_q[14:12] <= 3'b 010;
451               end
452             endcase
453           end
454           2'b01: begin // Quadrant 1
455             case (mem_rdata_latched[15:13])
456               3'b 000: begin // C.ADDI
457                 mem_rdata_q[14:12] <= 3'b000;
458                 mem_rdata_q[31:20] <= $signed({mem_rdata_latched[12],
                        mem_rdata_latched[6:2]});
459               end
460               3'b 010: begin // C.LI
461                 mem_rdata_q[14:12] <= 3'b000;
462                 mem_rdata_q[31:20] <= $signed({mem_rdata_latched[12],
                        mem_rdata_latched[6:2]});
463               end
464               3'b 011: begin
465                 if (mem_rdata_latched[11:7] == 2) begin // C.ADDI16SP
466                   mem_rdata_q[14:12] <= 3'b000;
467                   mem_rdata_q[31:20] <= $signed({mem_rdata_latched[12],
                          mem_rdata_latched[4:3],
468                           mem_rdata_latched[5], mem_rdata_latched[2],
                              mem_rdata_latched[6], 4'b 0000});
469                 end else begin // C.LUI
470                   mem_rdata_q[31:12] <= $signed({mem_rdata_latched[12],
                          mem_rdata_latched[6:2]});
471                 end
472               end
473               3'b100: begin
474                 if (mem_rdata_latched[11:10] == 2'b00) begin // C.SRLI
475                   mem_rdata_q[31:25] <= 7'b0000000;
476                   mem_rdata_q[14:12] <= 3'b 101;
477                 end
478                 if (mem_rdata_latched[11:10] == 2'b01) begin // C.SRAI
479                   mem_rdata_q[31:25] <= 7'b0100000;
480                   mem_rdata_q[14:12] <= 3'b 101;
481                 end
482                 if (mem_rdata_latched[11:10] == 2'b10) begin // C.ANDI
483                   mem_rdata_q[14:12] <= 3'b111;
```

```
484                 mem_rdata_q[31:20] <= $signed({mem_rdata_latched[12],
                        mem_rdata_latched[6:2]});
485               end
486               if (mem_rdata_latched[12:10] == 3'b011) begin // C.SUB,
                      C.XOR, C.OR, C.AND
487                 if (mem_rdata_latched[6:5] == 2'b00) mem_rdata_q[14:12]
                        <= 3'b000;
488                 if (mem_rdata_latched[6:5] == 2'b01) mem_rdata_q[14:12]
                        <= 3'b100;
489                 if (mem_rdata_latched[6:5] == 2'b10) mem_rdata_q[14:12]
                        <= 3'b110;
490                 if (mem_rdata_latched[6:5] == 2'b11) mem_rdata_q[14:12]
                        <= 3'b111;
491                 mem_rdata_q[31:25] <= mem_rdata_latched[6:5] == 2'b00 ?
                        7'b0100000 : 7'b0000000;
492               end
493             end
494             3'b 110: begin // C.BEQZ
495               mem_rdata_q[14:12] <= 3'b000;
496               { mem_rdata_q[31], mem_rdata_q[7], mem_rdata_q[30:25],
                      mem_rdata_q[11:8] } <=
497                   $signed({mem_rdata_latched[12],
                          mem_rdata_latched[6:5], mem_rdata_latched[2],
498                       mem_rdata_latched[11:10], mem_rdata_latched[4:3]});
499             end
500             3'b 111: begin // C.BNEZ
501               mem_rdata_q[14:12] <= 3'b001;
502               { mem_rdata_q[31], mem_rdata_q[7], mem_rdata_q[30:25],
                      mem_rdata_q[11:8] } <=
503                   $signed({mem_rdata_latched[12],
                          mem_rdata_latched[6:5], mem_rdata_latched[2],
504                       mem_rdata_latched[11:10], mem_rdata_latched[4:3]});
505             end
506           endcase
507         end
508         2'b10: begin // Quadrant 2
509           case (mem_rdata_latched[15:13])
510             3'b000: begin // C.SLLI
511               mem_rdata_q[31:25] <= 7'b0000000;
512               mem_rdata_q[14:12] <= 3'b 001;
513             end
514             3'b010: begin // C.LWSP
515               mem_rdata_q[31:20] <= {4'b0, mem_rdata_latched[3:2],
                      mem_rdata_latched[12], mem_rdata_latched[6:4], 2'b00};
516               mem_rdata_q[14:12] <= 3'b 010;
517             end
518             3'b100: begin
519               if (mem_rdata_latched[12] == 0 && mem_rdata_latched[6:2]
                      == 0) begin // C.JR
520                 mem_rdata_q[14:12] <= 3'b000;
521                 mem_rdata_q[31:20] <= 12'b0;
522               end
```

```
523                 if (mem_rdata_latched[12] == 0 && mem_rdata_latched[6:2]
                        != 0) begin // C.MV
524                   mem_rdata_q[14:12] <= 3'b000;
525                   mem_rdata_q[31:25] <= 7'b0000000;
526                 end
527                 if (mem_rdata_latched[12] != 0 && mem_rdata_latched[11:7]
                        != 0 && mem_rdata_latched[6:2] == 0) begin // C.JALR
528                   mem_rdata_q[14:12] <= 3'b000;
529                   mem_rdata_q[31:20] <= 12'b0;
530                 end
531                 if (mem_rdata_latched[12] != 0 && mem_rdata_latched[6:2]
                        != 0) begin // C.ADD
532                   mem_rdata_q[14:12] <= 3'b000;
533                   mem_rdata_q[31:25] <= 7'b0000000;
534                 end
535               end
536               3'b110: begin // C.SWSP
537                 {mem_rdata_q[31:25], mem_rdata_q[11:7]} <= {4'b0,
                        mem_rdata_latched[8:7], mem_rdata_latched[12:9], 2'b00};
538                 mem_rdata_q[14:12] <= 3'b 010;
539               end
540             endcase
541           end
542         endcase
543       end
544     end
545
546     always @(posedge clk) begin
547       if (resetn && !trap) begin
548         if (mem_do_prefetch || mem_do_rinst || mem_do_rdata)
549           `assert(!mem_do_wdata);
550
551         if (mem_do_prefetch || mem_do_rinst)
552           `assert(!mem_do_rdata);
553
554         if (mem_do_rdata)
555           `assert(!mem_do_prefetch && !mem_do_rinst);
556
557         if (mem_do_wdata)
558           `assert(!(mem_do_prefetch || mem_do_rinst || mem_do_rdata));
559
560         if (mem_state == 2 || mem_state == 3)
561           `assert(mem_valid || mem_do_prefetch);
562       end
563     end
564
565     always @(posedge clk) begin
566       if (!resetn || trap) begin
567         if (!resetn)
568           mem_state <= 0;
569         if (!resetn || mem_ready)
570           mem_valid <= 0;
571         mem_la_secondword <= 0;
```

```
572         prefetched_high_word <= 0;
573       end else begin
574         if (mem_la_read || mem_la_write) begin
575           mem_addr <= mem_la_addr;
576           mem_wstrb <= mem_la_wstrb & {4{mem_la_write}};
577         end
578         if (mem_la_write) begin
579           mem_wdata <= mem_la_wdata;
580         end
581         case (mem_state)
582           0: begin
583             if (mem_do_prefetch || mem_do_rinst || mem_do_rdata) begin
584               mem_valid <= !mem_la_use_prefetched_high_word;
585               mem_instr <= mem_do_prefetch || mem_do_rinst;
586               mem_wstrb <= 0;
587               mem_state <= 1;
588             end
589             if (mem_do_wdata) begin
590               mem_valid <= 1;
591               mem_instr <= 0;
592               mem_state <= 2;
593             end
594           end
595           1: begin
596             `assert(mem_wstrb == 0);
597             `assert(mem_do_prefetch || mem_do_rinst || mem_do_rdata);
598             `assert(mem_valid == !mem_la_use_prefetched_high_word);
599             `assert(mem_instr == (mem_do_prefetch || mem_do_rinst));
600             if (mem_xfer) begin
601               if (COMPRESSED_ISA && mem_la_read) begin
602                 mem_valid <= 1;
603                 mem_la_secondword <= 1;
604                 if (!mem_la_use_prefetched_high_word)
605                   mem_16bit_buffer <= mem_rdata[31:16];
606               end else begin
607                 mem_valid <= 0;
608                 mem_la_secondword <= 0;
609                 if (COMPRESSED_ISA && !mem_do_rdata) begin
610                   if (~&mem_rdata[1:0] || mem_la_secondword) begin
611                     mem_16bit_buffer <= mem_rdata[31:16];
612                     prefetched_high_word <= 1;
613                   end else begin
614                     prefetched_high_word <= 0;
615                   end
616                 end
617                 mem_state <= mem_do_rinst || mem_do_rdata ? 0 : 3;
618               end
619             end
620           end
621           2: begin
622             `assert(mem_wstrb != 0);
623             `assert(mem_do_wdata);
624             if (mem_xfer) begin
```

```
625             mem_valid <= 0;
626             mem_state <= 0;
627           end
628         end
629         3: begin
630           `assert(mem_wstrb == 0);
631           `assert(mem_do_prefetch);
632           if (mem_do_rinst) begin
633             mem_state <= 0;
634           end
635         end
636       endcase
637     end
638
639     if (clear_prefetched_high_word)
640       prefetched_high_word <= 0;
641   end
642
643
644   // Instruction Decoder
645
646   reg instr_lui, instr_auipc, instr_jal, instr_jalr;
647   reg instr_beq, instr_bne, instr_blt, instr_bge, instr_bltu, instr_bgeu;
648   reg instr_lb, instr_lh, instr_lw, instr_lbu, instr_lhu, instr_sb,
      instr_sh, instr_sw;
649   reg instr_addi, instr_slti, instr_sltiu, instr_xori, instr_ori,
      instr_andi, instr_slli, instr_srli, instr_srai;
650   reg instr_add, instr_sub, instr_sll, instr_slt, instr_sltu, instr_xor,
      instr_srl, instr_sra, instr_or, instr_and;
651   reg instr_rdcycle, instr_rdcycleh, instr_rdinstr, instr_rdinstrh,
      instr_ecall_ebreak;
652   reg instr_getq, instr_setq, instr_retirq, instr_maskirq,
      instr_waitirq, instr_timer;
653   wire instr_trap;
654
655   reg [regindex_bits-1:0] decoded_rd, decoded_rs1, decoded_rs2;
656   reg [31:0] decoded_imm, decoded_imm_j;
657   reg decoder_trigger;
658   reg decoder_trigger_q;
659   reg decoder_pseudo_trigger;
660   reg decoder_pseudo_trigger_q;
661   reg compressed_instr;
662
663   reg is_lui_auipc_jal;
664   reg is_lb_lh_lw_lbu_lhu;
665   reg is_slli_srli_srai;
666   reg is_jalr_addi_slti_sltiu_xori_ori_andi;
667   reg is_sb_sh_sw;
668   reg is_sll_srl_sra;
669   reg is_lui_auipc_jal_jalr_addi_add_sub;
670   reg is_slti_blt_slt;
671   reg is_sltiu_bltu_sltu;
672   reg is_beq_bne_blt_bge_bltu_bgeu;
```

```
673   reg is_lbu_lhu_lw;
674   reg is_alu_reg_imm;
675   reg is_alu_reg_reg;
676   reg is_compare;
677
678   assign instr_trap = (CATCH_ILLINSN || WITH_PCPI) && !{instr_lui,
          instr_auipc, instr_jal, instr_jalr,
679           instr_beq, instr_bne, instr_blt, instr_bge, instr_bltu, instr_bgeu,
680           instr_lb, instr_lh, instr_lw, instr_lbu, instr_lhu, instr_sb,
                instr_sh, instr_sw,
681           instr_addi, instr_slti, instr_sltiu, instr_xori, instr_ori,
                instr_andi, instr_slli, instr_srli, instr_srai,
682           instr_add, instr_sub, instr_sll, instr_slt, instr_sltu, instr_xor,
                instr_srl, instr_sra, instr_or, instr_and,
683           instr_rdcycle, instr_rdcycleh, instr_rdinstr, instr_rdinstrh,
684           instr_getq, instr_setq, instr_retirq, instr_maskirq,
                instr_waitirq, instr_timer};
685
686   wire is_rdcycle_rdcycleh_rdinstr_rdinstrh;
687   assign is_rdcycle_rdcycleh_rdinstr_rdinstrh = |{instr_rdcycle,
          instr_rdcycleh, instr_rdinstr, instr_rdinstrh};
688
689   reg [63:0] new_ascii_instr;
690   `FORMAL_KEEP reg [63:0] dbg_ascii_instr;
691   `FORMAL_KEEP reg [31:0] dbg_insn_imm;
692   `FORMAL_KEEP reg [4:0] dbg_insn_rs1;
693   `FORMAL_KEEP reg [4:0] dbg_insn_rs2;
694   `FORMAL_KEEP reg [4:0] dbg_insn_rd;
695   `FORMAL_KEEP reg [31:0] dbg_rs1val;
696   `FORMAL_KEEP reg [31:0] dbg_rs2val;
697   `FORMAL_KEEP reg dbg_rs1val_valid;
698   `FORMAL_KEEP reg dbg_rs2val_valid;
699
700   always @* begin
701     new_ascii_instr = "";
702
703     if (instr_lui)      new_ascii_instr = "lui";
704     if (instr_auipc)    new_ascii_instr = "auipc";
705     if (instr_jal)      new_ascii_instr = "jal";
706     if (instr_jalr)     new_ascii_instr = "jalr";
707
708     if (instr_beq)      new_ascii_instr = "beq";
709     if (instr_bne)      new_ascii_instr = "bne";
710     if (instr_blt)      new_ascii_instr = "blt";
711     if (instr_bge)      new_ascii_instr = "bge";
712     if (instr_bltu)     new_ascii_instr = "bltu";
713     if (instr_bgeu)     new_ascii_instr = "bgeu";
714
715     if (instr_lb)       new_ascii_instr = "lb";
716     if (instr_lh)       new_ascii_instr = "lh";
717     if (instr_lw)       new_ascii_instr = "lw";
718     if (instr_lbu)      new_ascii_instr = "lbu";
719     if (instr_lhu)      new_ascii_instr = "lhu";
```

```
720     if (instr_sb)       new_ascii_instr = "sb";
721     if (instr_sh)       new_ascii_instr = "sh";
722     if (instr_sw)       new_ascii_instr = "sw";
723
724     if (instr_addi)     new_ascii_instr = "addi";
725     if (instr_slti)     new_ascii_instr = "slti";
726     if (instr_sltiu)    new_ascii_instr = "sltiu";
727     if (instr_xori)     new_ascii_instr = "xori";
728     if (instr_ori)      new_ascii_instr = "ori";
729     if (instr_andi)     new_ascii_instr = "andi";
730     if (instr_slli)     new_ascii_instr = "slli";
731     if (instr_srli)     new_ascii_instr = "srli";
732     if (instr_srai)     new_ascii_instr = "srai";
733
734     if (instr_add)      new_ascii_instr = "add";
735     if (instr_sub)      new_ascii_instr = "sub";
736     if (instr_sll)      new_ascii_instr = "sll";
737     if (instr_slt)      new_ascii_instr = "slt";
738     if (instr_sltu)     new_ascii_instr = "sltu";
739     if (instr_xor)      new_ascii_instr = "xor";
740     if (instr_srl)      new_ascii_instr = "srl";
741     if (instr_sra)      new_ascii_instr = "sra";
742     if (instr_or)       new_ascii_instr = "or";
743     if (instr_and)      new_ascii_instr = "and";
744
745     if (instr_rdcycle)  new_ascii_instr = "rdcycle";
746     if (instr_rdcycleh) new_ascii_instr = "rdcycleh";
747     if (instr_rdinstr)  new_ascii_instr = "rdinstr";
748     if (instr_rdinstrh) new_ascii_instr = "rdinstrh";
749
750     if (instr_getq)     new_ascii_instr = "getq";
751     if (instr_setq)     new_ascii_instr = "setq";
752     if (instr_retirq)   new_ascii_instr = "retirq";
753     if (instr_maskirq)  new_ascii_instr = "maskirq";
754     if (instr_waitirq)  new_ascii_instr = "waitirq";
755     if (instr_timer)    new_ascii_instr = "timer";
756   end
757
758   reg [63:0] q_ascii_instr;
759   reg [31:0] q_insn_imm;
760   reg [31:0] q_insn_opcode;
761   reg [4:0] q_insn_rs1;
762   reg [4:0] q_insn_rs2;
763   reg [4:0] q_insn_rd;
764   reg dbg_next;
765
766   wire launch_next_insn;
767   reg dbg_valid_insn;
768
769   reg [63:0] cached_ascii_instr;
770   reg [31:0] cached_insn_imm;
771   reg [31:0] cached_insn_opcode;
772   reg [4:0] cached_insn_rs1;
```

```
773   reg [4:0] cached_insn_rs2;
774   reg [4:0] cached_insn_rd;
775
776   always @(posedge clk) begin
777     q_ascii_instr <= dbg_ascii_instr;
778     q_insn_imm <= dbg_insn_imm;
779     q_insn_opcode <= dbg_insn_opcode;
780     q_insn_rs1 <= dbg_insn_rs1;
781     q_insn_rs2 <= dbg_insn_rs2;
782     q_insn_rd <= dbg_insn_rd;
783     dbg_next <= launch_next_insn;
784
785     if (!resetn || trap)
786       dbg_valid_insn <= 0;
787     else if (launch_next_insn)
788       dbg_valid_insn <= 1;
789
790     if (decoder_trigger_q) begin
791       cached_ascii_instr <= new_ascii_instr;
792       cached_insn_imm <= decoded_imm;
793       if (&next_insn_opcode[1:0])
794         cached_insn_opcode <= next_insn_opcode;
795       else
796         cached_insn_opcode <= {16'b0, next_insn_opcode[15:0]};
797       cached_insn_rs1 <= decoded_rs1;
798       cached_insn_rs2 <= decoded_rs2;
799       cached_insn_rd <= decoded_rd;
800     end
801
802     if (launch_next_insn) begin
803       dbg_insn_addr <= next_pc;
804     end
805   end
806
807   always @* begin
808     dbg_ascii_instr = q_ascii_instr;
809     dbg_insn_imm = q_insn_imm;
810     dbg_insn_opcode = q_insn_opcode;
811     dbg_insn_rs1 = q_insn_rs1;
812     dbg_insn_rs2 = q_insn_rs2;
813     dbg_insn_rd = q_insn_rd;
814
815     if (dbg_next) begin
816       if (decoder_pseudo_trigger_q) begin
817         dbg_ascii_instr = cached_ascii_instr;
818         dbg_insn_imm = cached_insn_imm;
819         dbg_insn_opcode = cached_insn_opcode;
820         dbg_insn_rs1 = cached_insn_rs1;
821         dbg_insn_rs2 = cached_insn_rs2;
822         dbg_insn_rd = cached_insn_rd;
823       end else begin
824         dbg_ascii_instr = new_ascii_instr;
825         if (&next_insn_opcode[1:0])
```

```
826             dbg_insn_opcode = next_insn_opcode;
827           else
828             dbg_insn_opcode = {16'b0, next_insn_opcode[15:0]};
829           dbg_insn_imm = decoded_imm;
830           dbg_insn_rs1 = decoded_rs1;
831           dbg_insn_rs2 = decoded_rs2;
832           dbg_insn_rd = decoded_rd;
833         end
834       end
835     end
836
837 `ifdef DEBUGASM
838     always @(posedge clk) begin
839       if (dbg_next) begin
840         $display("debugasm %x %x %s", dbg_insn_addr, dbg_insn_opcode,
              dbg_ascii_instr ? dbg_ascii_instr : "*");
841       end
842     end
843 `endif
844
845 `ifdef DEBUG
846     always @(posedge clk) begin
847       if (dbg_next) begin
848         if (&dbg_insn_opcode[1:0])
849           $display("DECODE: 0x%08x 0x%08x %-0s", dbg_insn_addr,
                dbg_insn_opcode, dbg_ascii_instr ? dbg_ascii_instr :
                "UNKNOWN");
850         else
851           $display("DECODE: 0x%08x     0x%04x %-0s", dbg_insn_addr,
                dbg_insn_opcode[15:0], dbg_ascii_instr ? dbg_ascii_instr :
                "UNKNOWN");
852       end
853     end
854 `endif
855
856     always @(posedge clk) begin
857       is_lui_auipc_jal <= |{instr_lui, instr_auipc, instr_jal};
858       is_lui_auipc_jal_jalr_addi_add_sub <= |{instr_lui, instr_auipc,
            instr_jal, instr_jalr, instr_addi, instr_add, instr_sub};
859       is_slti_blt_slt <= |{instr_slti, instr_blt, instr_slt};
860       is_sltiu_bltu_sltu <= |{instr_sltiu, instr_bltu, instr_sltu};
861       is_lbu_lhu_lw <= |{instr_lbu, instr_lhu, instr_lw};
862       is_compare <= |{is_beq_bne_blt_bge_bltu_bgeu, instr_slti, instr_slt,
            instr_sltiu, instr_sltu};
863
864       if (mem_do_rinst && mem_done) begin
865         instr_lui     <= mem_rdata_latched[6:0] == 7'b0110111;
866         instr_auipc   <= mem_rdata_latched[6:0] == 7'b0010111;
867         instr_jal     <= mem_rdata_latched[6:0] == 7'b1101111;
868         instr_jalr    <= mem_rdata_latched[6:0] == 7'b1100111 &&
              mem_rdata_latched[14:12] == 3'b000;
869         instr_retirq  <= mem_rdata_latched[6:0] == 7'b0001011 &&
              mem_rdata_latched[31:25] == 7'b0000010 && ENABLE_IRQ;
```

```
870         instr_waitirq <= mem_rdata_latched[6:0] == 7'b0001011 &&
                mem_rdata_latched[31:25] == 7'b0000100 && ENABLE_IRQ;
871
872         is_beq_bne_blt_bge_bltu_bgeu <= mem_rdata_latched[6:0] ==
                7'b1100011;
873         is_lb_lh_lw_lbu_lhu          <= mem_rdata_latched[6:0] ==
                7'b0000011;
874         is_sb_sh_sw                  <= mem_rdata_latched[6:0] ==
                7'b0100011;
875         is_alu_reg_imm               <= mem_rdata_latched[6:0] ==
                7'b0010011;
876         is_alu_reg_reg               <= mem_rdata_latched[6:0] ==
                7'b0110011;
877
878         { decoded_imm_j[31:20], decoded_imm_j[10:1], decoded_imm_j[11],
                decoded_imm_j[19:12], decoded_imm_j[0] } <=
                $signed({mem_rdata_latched[31:12], 1'b0});
879
880         decoded_rd <= mem_rdata_latched[11:7];
881         decoded_rs1 <= mem_rdata_latched[19:15];
882         decoded_rs2 <= mem_rdata_latched[24:20];
883
884         if (mem_rdata_latched[6:0] == 7'b0001011 &&
                mem_rdata_latched[31:25] == 7'b0000000 && ENABLE_IRQ &&
                ENABLE_IRQ_QREGS)
885           decoded_rs1[regindex_bits-1] <= 1; // instr_getq
886
887         if (mem_rdata_latched[6:0] == 7'b0001011 &&
                mem_rdata_latched[31:25] == 7'b0000010 && ENABLE_IRQ)
888           decoded_rs1 <= ENABLE_IRQ_QREGS ? irqregs_offset : 3; //
                  instr_retirq
889
890         compressed_instr <= 0;
891         if (COMPRESSED_ISA && mem_rdata_latched[1:0] != 2'b11) begin
892           compressed_instr <= 1;
893           decoded_rd <= 0;
894           decoded_rs1 <= 0;
895           decoded_rs2 <= 0;
896
897           { decoded_imm_j[31:11], decoded_imm_j[4], decoded_imm_j[9:8],
                  decoded_imm_j[10], decoded_imm_j[6],
898             decoded_imm_j[7], decoded_imm_j[3:1], decoded_imm_j[5],
                    decoded_imm_j[0] } <= $signed({mem_rdata_latched[12:2],
                    1'b0});
899
900           case (mem_rdata_latched[1:0])
901             2'b00: begin // Quadrant 0
902               case (mem_rdata_latched[15:13])
903                 3'b000: begin // C.ADDI4SPN
904                   is_alu_reg_imm <= |mem_rdata_latched[12:5];
905                   decoded_rs1 <= 2;
906                   decoded_rd <= 8 + mem_rdata_latched[4:2];
907                 end
```

```
908                 3'b010: begin // C.LW
909                   is_lb_lh_lw_lbu_lhu <= 1;
910                   decoded_rs1 <= 8 + mem_rdata_latched[9:7];
911                   decoded_rd <= 8 + mem_rdata_latched[4:2];
912                 end
913                 3'b110: begin // C.SW
914                   is_sb_sh_sw <= 1;
915                   decoded_rs1 <= 8 + mem_rdata_latched[9:7];
916                   decoded_rs2 <= 8 + mem_rdata_latched[4:2];
917                 end
918               endcase
919             end
920             2'b01: begin // Quadrant 1
921               case (mem_rdata_latched[15:13])
922                 3'b000: begin // C.NOP / C.ADDI
923                   is_alu_reg_imm <= 1;
924                   decoded_rd <= mem_rdata_latched[11:7];
925                   decoded_rs1 <= mem_rdata_latched[11:7];
926                 end
927                 3'b001: begin // C.JAL
928                   instr_jal <= 1;
929                   decoded_rd <= 1;
930                 end
931                 3'b 010: begin // C.LI
932                   is_alu_reg_imm <= 1;
933                   decoded_rd <= mem_rdata_latched[11:7];
934                   decoded_rs1 <= 0;
935                 end
936                 3'b 011: begin
937                   if (mem_rdata_latched[12] || mem_rdata_latched[6:2])
                          begin
938                     if (mem_rdata_latched[11:7] == 2) begin // C.ADDI16SP
939                       is_alu_reg_imm <= 1;
940                       decoded_rd <= mem_rdata_latched[11:7];
941                       decoded_rs1 <= mem_rdata_latched[11:7];
942                     end else begin // C.LUI
943                       instr_lui <= 1;
944                       decoded_rd <= mem_rdata_latched[11:7];
945                       decoded_rs1 <= 0;
946                     end
947                   end
948                 end
949                 3'b100: begin
950                   if (!mem_rdata_latched[11] && !mem_rdata_latched[12])
                          begin // C.SRLI, C.SRAI
951                     is_alu_reg_imm <= 1;
952                     decoded_rd <= 8 + mem_rdata_latched[9:7];
953                     decoded_rs1 <= 8 + mem_rdata_latched[9:7];
954                     decoded_rs2 <= {mem_rdata_latched[12],
                            mem_rdata_latched[6:2]};
955                   end
956                   if (mem_rdata_latched[11:10] == 2'b10) begin // C.ANDI
957                     is_alu_reg_imm <= 1;
```

```
958                   decoded_rd <= 8 + mem_rdata_latched[9:7];
959                   decoded_rs1 <= 8 + mem_rdata_latched[9:7];
960                 end
961                 if (mem_rdata_latched[12:10] == 3'b011) begin // C.SUB,
                       C.XOR, C.OR, C.AND
962                   is_alu_reg_reg <= 1;
963                   decoded_rd <= 8 + mem_rdata_latched[9:7];
964                   decoded_rs1 <= 8 + mem_rdata_latched[9:7];
965                   decoded_rs2 <= 8 + mem_rdata_latched[4:2];
966                 end
967               end
968               3'b101: begin // C.J
969                 instr_jal <= 1;
970               end
971               3'b110: begin // C.BEQZ
972                 is_beq_bne_blt_bge_bltu_bgeu <= 1;
973                 decoded_rs1 <= 8 + mem_rdata_latched[9:7];
974                 decoded_rs2 <= 0;
975               end
976               3'b111: begin // C.BNEZ
977                 is_beq_bne_blt_bge_bltu_bgeu <= 1;
978                 decoded_rs1 <= 8 + mem_rdata_latched[9:7];
979                 decoded_rs2 <= 0;
980               end
981             endcase
982           end
983           2'b10: begin // Quadrant 2
984             case (mem_rdata_latched[15:13])
985               3'b000: begin // C.SLLI
986                 if (!mem_rdata_latched[12]) begin
987                   is_alu_reg_imm <= 1;
988                   decoded_rd <= mem_rdata_latched[11:7];
989                   decoded_rs1 <= mem_rdata_latched[11:7];
990                   decoded_rs2 <= {mem_rdata_latched[12],
                         mem_rdata_latched[6:2]};
991                 end
992               end
993               3'b010: begin // C.LWSP
994                 if (mem_rdata_latched[11:7]) begin
995                   is_lb_lh_lw_lbu_lhu <= 1;
996                   decoded_rd <= mem_rdata_latched[11:7];
997                   decoded_rs1 <= 2;
998                 end
999               end
1000              3'b100: begin
1001                if (mem_rdata_latched[12] == 0 &&
                       mem_rdata_latched[11:7] != 0 &&
                       mem_rdata_latched[6:2] == 0) begin // C.JR
1002                  instr_jalr <= 1;
1003                  decoded_rd <= 0;
1004                  decoded_rs1 <= mem_rdata_latched[11:7];
1005                end
```

```
1006                if (mem_rdata_latched[12] == 0 && mem_rdata_latched[6:2]
                        != 0) begin // C.MV
1007                  is_alu_reg_reg <= 1;
1008                  decoded_rd <= mem_rdata_latched[11:7];
1009                  decoded_rs1 <= 0;
1010                  decoded_rs2 <= mem_rdata_latched[6:2];
1011                end
1012                if (mem_rdata_latched[12] != 0 &&
                        mem_rdata_latched[11:7] != 0 &&
                        mem_rdata_latched[6:2] == 0) begin // C.JALR
1013                  instr_jalr <= 1;
1014                  decoded_rd <= 1;
1015                  decoded_rs1 <= mem_rdata_latched[11:7];
1016                end
1017                if (mem_rdata_latched[12] != 0 && mem_rdata_latched[6:2]
                        != 0) begin // C.ADD
1018                  is_alu_reg_reg <= 1;
1019                  decoded_rd <= mem_rdata_latched[11:7];
1020                  decoded_rs1 <= mem_rdata_latched[11:7];
1021                  decoded_rs2 <= mem_rdata_latched[6:2];
1022                end
1023              end
1024              3'b110: begin // C.SWSP
1025                is_sb_sh_sw <= 1;
1026                decoded_rs1 <= 2;
1027                decoded_rs2 <= mem_rdata_latched[6:2];
1028              end
1029            endcase
1030          end
1031        endcase
1032      end
1033    end
1034
1035    if (decoder_trigger && !decoder_pseudo_trigger) begin
1036      pcpi_insn <= WITH_PCPI ? mem_rdata_q : 'bx;
1037
1038      instr_beq   <= is_beq_bne_blt_bge_bltu_bgeu && mem_rdata_q[14:12]
                         == 3'b000;
1039      instr_bne   <= is_beq_bne_blt_bge_bltu_bgeu && mem_rdata_q[14:12]
                         == 3'b001;
1040      instr_blt   <= is_beq_bne_blt_bge_bltu_bgeu && mem_rdata_q[14:12]
                         == 3'b100;
1041      instr_bge   <= is_beq_bne_blt_bge_bltu_bgeu && mem_rdata_q[14:12]
                         == 3'b101;
1042      instr_bltu  <= is_beq_bne_blt_bge_bltu_bgeu && mem_rdata_q[14:12]
                         == 3'b110;
1043      instr_bgeu  <= is_beq_bne_blt_bge_bltu_bgeu && mem_rdata_q[14:12]
                         == 3'b111;
1044
1045      instr_lb    <= is_lb_lh_lw_lbu_lhu && mem_rdata_q[14:12] == 3'b000;
1046      instr_lh    <= is_lb_lh_lw_lbu_lhu && mem_rdata_q[14:12] == 3'b001;
1047      instr_lw    <= is_lb_lh_lw_lbu_lhu && mem_rdata_q[14:12] == 3'b010;
1048      instr_lbu   <= is_lb_lh_lw_lbu_lhu && mem_rdata_q[14:12] == 3'b100;
```

```
1049        instr_lhu   <= is_lb_lh_lw_lbu_lhu && mem_rdata_q[14:12] == 3'b101;
1050
1051        instr_sb    <= is_sb_sh_sw && mem_rdata_q[14:12] == 3'b000;
1052        instr_sh    <= is_sb_sh_sw && mem_rdata_q[14:12] == 3'b001;
1053        instr_sw    <= is_sb_sh_sw && mem_rdata_q[14:12] == 3'b010;
1054
1055        instr_addi  <= is_alu_reg_imm && mem_rdata_q[14:12] == 3'b000;
1056        instr_slti  <= is_alu_reg_imm && mem_rdata_q[14:12] == 3'b010;
1057        instr_sltiu <= is_alu_reg_imm && mem_rdata_q[14:12] == 3'b011;
1058        instr_xori  <= is_alu_reg_imm && mem_rdata_q[14:12] == 3'b100;
1059        instr_ori   <= is_alu_reg_imm && mem_rdata_q[14:12] == 3'b110;
1060        instr_andi  <= is_alu_reg_imm && mem_rdata_q[14:12] == 3'b111;
1061
1062        instr_slli  <= is_alu_reg_imm && mem_rdata_q[14:12] == 3'b001 &&
               mem_rdata_q[31:25] == 7'b0000000;
1063        instr_srli  <= is_alu_reg_imm && mem_rdata_q[14:12] == 3'b101 &&
               mem_rdata_q[31:25] == 7'b0000000;
1064        instr_srai  <= is_alu_reg_imm && mem_rdata_q[14:12] == 3'b101 &&
               mem_rdata_q[31:25] == 7'b0100000;
1065
1066        instr_add   <= is_alu_reg_reg && mem_rdata_q[14:12] == 3'b000 &&
               mem_rdata_q[31:25] == 7'b0000000;
1067        instr_sub   <= is_alu_reg_reg && mem_rdata_q[14:12] == 3'b000 &&
               mem_rdata_q[31:25] == 7'b0100000;
1068        instr_sll   <= is_alu_reg_reg && mem_rdata_q[14:12] == 3'b001 &&
               mem_rdata_q[31:25] == 7'b0000000;
1069        instr_slt   <= is_alu_reg_reg && mem_rdata_q[14:12] == 3'b010 &&
               mem_rdata_q[31:25] == 7'b0000000;
1070        instr_sltu  <= is_alu_reg_reg && mem_rdata_q[14:12] == 3'b011 &&
               mem_rdata_q[31:25] == 7'b0000000;
1071        instr_xor   <= is_alu_reg_reg && mem_rdata_q[14:12] == 3'b100 &&
               mem_rdata_q[31:25] == 7'b0000000;
1072        instr_srl   <= is_alu_reg_reg && mem_rdata_q[14:12] == 3'b101 &&
               mem_rdata_q[31:25] == 7'b0000000;
1073        instr_sra   <= is_alu_reg_reg && mem_rdata_q[14:12] == 3'b101 &&
               mem_rdata_q[31:25] == 7'b0100000;
1074        instr_or    <= is_alu_reg_reg && mem_rdata_q[14:12] == 3'b110 &&
               mem_rdata_q[31:25] == 7'b0000000;
1075        instr_and   <= is_alu_reg_reg && mem_rdata_q[14:12] == 3'b111 &&
               mem_rdata_q[31:25] == 7'b0000000;
1076
1077        instr_rdcycle  <= ((mem_rdata_q[6:0] == 7'b1110011 &&
               mem_rdata_q[31:12] == 'b11000000000000000010) ||
1078                           (mem_rdata_q[6:0] == 7'b1110011 &&
                                  mem_rdata_q[31:12] ==
                                  'b11000000000100000010)) && ENABLE_COUNTERS;
1079        instr_rdcycleh <= ((mem_rdata_q[6:0] == 7'b1110011 &&
               mem_rdata_q[31:12] == 'b11001000000000000010) ||
1080                           (mem_rdata_q[6:0] == 7'b1110011 &&
                                  mem_rdata_q[31:12] ==
                                  'b11001000000100000010)) && ENABLE_COUNTERS
                                  && ENABLE_COUNTERS64;
```

```
1081        instr_rdinstr  <=  (mem_rdata_q[6:0] == 7'b1110011 &&
               mem_rdata_q[31:12] == 'b11000000001000000010) &&
               ENABLE_COUNTERS;
1082        instr_rdinstrh <=  (mem_rdata_q[6:0] == 7'b1110011 &&
               mem_rdata_q[31:12] == 'b11001000001000000010) &&
               ENABLE_COUNTERS && ENABLE_COUNTERS64;
1083
1084        instr_ecall_ebreak <= ((mem_rdata_q[6:0] == 7'b1110011 &&
               !mem_rdata_q[31:21] && !mem_rdata_q[19:7]) ||
1085            (COMPRESSED_ISA && mem_rdata_q[15:0] == 16'h9002));
1086
1087        instr_getq    <= mem_rdata_q[6:0] == 7'b0001011 &&
               mem_rdata_q[31:25] == 7'b0000000 && ENABLE_IRQ &&
               ENABLE_IRQ_QREGS;
1088        instr_setq    <= mem_rdata_q[6:0] == 7'b0001011 &&
               mem_rdata_q[31:25] == 7'b0000001 && ENABLE_IRQ &&
               ENABLE_IRQ_QREGS;
1089        instr_maskirq <= mem_rdata_q[6:0] == 7'b0001011 &&
               mem_rdata_q[31:25] == 7'b0000011 && ENABLE_IRQ;
1090        instr_timer   <= mem_rdata_q[6:0] == 7'b0001011 &&
               mem_rdata_q[31:25] == 7'b0000101 && ENABLE_IRQ &&
               ENABLE_IRQ_TIMER;
1091
1092        is_slli_srli_srai <= is_alu_reg_imm && |{
1093          mem_rdata_q[14:12] == 3'b001 && mem_rdata_q[31:25] == 7'b0000000,
1094          mem_rdata_q[14:12] == 3'b101 && mem_rdata_q[31:25] == 7'b0000000,
1095          mem_rdata_q[14:12] == 3'b101 && mem_rdata_q[31:25] == 7'b0100000
1096        };
1097
1098        is_jalr_addi_slti_sltiu_xori_ori_andi <= instr_jalr ||
               is_alu_reg_imm && |{
1099          mem_rdata_q[14:12] == 3'b000,
1100          mem_rdata_q[14:12] == 3'b010,
1101          mem_rdata_q[14:12] == 3'b011,
1102          mem_rdata_q[14:12] == 3'b100,
1103          mem_rdata_q[14:12] == 3'b110,
1104          mem_rdata_q[14:12] == 3'b111
1105        };
1106
1107        is_sll_srl_sra <= is_alu_reg_reg && |{
1108          mem_rdata_q[14:12] == 3'b001 && mem_rdata_q[31:25] == 7'b0000000,
1109          mem_rdata_q[14:12] == 3'b101 && mem_rdata_q[31:25] == 7'b0000000,
1110          mem_rdata_q[14:12] == 3'b101 && mem_rdata_q[31:25] == 7'b0100000
1111        };
1112
1113        is_lui_auipc_jal_jalr_addi_add_sub <= 0;
1114        is_compare <= 0;
1115
1116        (* parallel_case *)
1117        case (1'b1)
1118          instr_jal:
1119            decoded_imm <= decoded_imm_j;
1120          |{instr_lui, instr_auipc}:
```

```
1121            decoded_imm <= mem_rdata_q[31:12] << 12;
1122          |{instr_jalr, is_lb_lh_lw_lbu_lhu, is_alu_reg_imm}:
1123            decoded_imm <= $signed(mem_rdata_q[31:20]);
1124          is_beq_bne_blt_bge_bltu_bgeu:
1125            decoded_imm <= $signed({mem_rdata_q[31], mem_rdata_q[7],
                   mem_rdata_q[30:25], mem_rdata_q[11:8], 1'b0});
1126          is_sb_sh_sw:
1127            decoded_imm <= $signed({mem_rdata_q[31:25],
                   mem_rdata_q[11:7]});
1128          default:
1129            decoded_imm <= 1'bx;
1130        endcase
1131      end
1132
1133      if (!resetn) begin
1134        is_beq_bne_blt_bge_bltu_bgeu <= 0;
1135        is_compare <= 0;
1136
1137        instr_beq   <= 0;
1138        instr_bne   <= 0;
1139        instr_blt   <= 0;
1140        instr_bge   <= 0;
1141        instr_bltu  <= 0;
1142        instr_bgeu  <= 0;
1143
1144        instr_addi  <= 0;
1145        instr_slti  <= 0;
1146        instr_sltiu <= 0;
1147        instr_xori  <= 0;
1148        instr_ori   <= 0;
1149        instr_andi  <= 0;
1150
1151        instr_add   <= 0;
1152        instr_sub   <= 0;
1153        instr_sll   <= 0;
1154        instr_slt   <= 0;
1155        instr_sltu  <= 0;
1156        instr_xor   <= 0;
1157        instr_srl   <= 0;
1158        instr_sra   <= 0;
1159        instr_or    <= 0;
1160        instr_and   <= 0;
1161      end
1162    end
1163
1164
1165    // Main State Machine
1166
1167    localparam cpu_state_trap   = 8'b10000000;
1168    localparam cpu_state_fetch  = 8'b01000000;
1169    localparam cpu_state_ld_rs1 = 8'b00100000;
1170    localparam cpu_state_ld_rs2 = 8'b00010000;
1171    localparam cpu_state_exec   = 8'b00001000;
```

```
1172   localparam cpu_state_shift  = 8'b00000100;
1173   localparam cpu_state_stmem  = 8'b00000010;
1174   localparam cpu_state_ldmem  = 8'b00000001;
1175
1176   reg [7:0] cpu_state;
1177   reg [1:0] irq_state;
1178
1179   `FORMAL_KEEP reg [127:0] dbg_ascii_state;
1180
1181   always @* begin
1182     dbg_ascii_state = "";
1183     if (cpu_state == cpu_state_trap)   dbg_ascii_state = "trap";
1184     if (cpu_state == cpu_state_fetch)  dbg_ascii_state = "fetch";
1185     if (cpu_state == cpu_state_ld_rs1) dbg_ascii_state = "ld_rs1";
1186     if (cpu_state == cpu_state_ld_rs2) dbg_ascii_state = "ld_rs2";
1187     if (cpu_state == cpu_state_exec)   dbg_ascii_state = "exec";
1188     if (cpu_state == cpu_state_shift)  dbg_ascii_state = "shift";
1189     if (cpu_state == cpu_state_stmem)  dbg_ascii_state = "stmem";
1190     if (cpu_state == cpu_state_ldmem)  dbg_ascii_state = "ldmem";
1191   end
1192
1193   reg set_mem_do_rinst;
1194   reg set_mem_do_rdata;
1195   reg set_mem_do_wdata;
1196
1197   reg latched_store;
1198   reg latched_stalu;
1199   reg latched_branch;
1200   reg latched_compr;
1201   reg latched_trace;
1202   reg latched_is_lu;
1203   reg latched_is_lh;
1204   reg latched_is_lb;
1205   reg [regindex_bits-1:0] latched_rd;
1206
1207   reg [31:0] current_pc;
1208   assign next_pc = latched_store && latched_branch ? reg_out & ~1 :
       reg_next_pc;
1209
1210   reg [3:0] pcpi_timeout_counter;
1211   reg pcpi_timeout;
1212
1213   reg [31:0] next_irq_pending;
1214   reg do_waitirq;
1215
1216   reg [31:0] alu_out, alu_out_q;
1217   reg alu_out_0, alu_out_0_q;
1218   reg alu_wait, alu_wait_2;
1219
1220   reg [31:0] alu_add_sub;
1221   reg [31:0] alu_shl, alu_shr;
1222   reg alu_eq, alu_ltu, alu_lts;
1223
```

```
1224    generate if (TWO_CYCLE_ALU) begin
1225      always @(posedge clk) begin
1226        alu_add_sub <= instr_sub ? reg_op1 - reg_op2 : reg_op1 + reg_op2;
1227        alu_eq <= reg_op1 == reg_op2;
1228        alu_lts <= $signed(reg_op1) < $signed(reg_op2);
1229        alu_ltu <= reg_op1 < reg_op2;
1230        alu_shl <= reg_op1 << reg_op2[4:0];
1231        alu_shr <= $signed({instr_sra || instr_srai ? reg_op1[31] : 1'b0,
              reg_op1}) >>> reg_op2[4:0];
1232      end
1233    end else begin
1234      always @* begin
1235        alu_add_sub = instr_sub ? reg_op1 - reg_op2 : reg_op1 + reg_op2;
1236        alu_eq = reg_op1 == reg_op2;
1237        alu_lts = $signed(reg_op1) < $signed(reg_op2);
1238        alu_ltu = reg_op1 < reg_op2;
1239        alu_shl = reg_op1 << reg_op2[4:0];
1240        alu_shr = $signed({instr_sra || instr_srai ? reg_op1[31] : 1'b0,
              reg_op1}) >>> reg_op2[4:0];
1241      end
1242    end endgenerate
1243
1244    always @* begin
1245      alu_out_0 = 'bx;
1246      (* parallel_case, full_case *)
1247      case (1'b1)
1248        instr_beq:
1249          alu_out_0 = alu_eq;
1250        instr_bne:
1251          alu_out_0 = !alu_eq;
1252        instr_bge:
1253          alu_out_0 = !alu_lts;
1254        instr_bgeu:
1255          alu_out_0 = !alu_ltu;
1256        is_slti_blt_slt && (!TWO_CYCLE_COMPARE ||
              !{instr_beq,instr_bne,instr_bge,instr_bgeu}):
1257          alu_out_0 = alu_lts;
1258        is_sltiu_bltu_sltu && (!TWO_CYCLE_COMPARE ||
              !{instr_beq,instr_bne,instr_bge,instr_bgeu}):
1259          alu_out_0 = alu_ltu;
1260      endcase
1261
1262      alu_out = 'bx;
1263      (* parallel_case, full_case *)
1264      case (1'b1)
1265        is_lui_auipc_jal_jalr_addi_add_sub:
1266          alu_out = alu_add_sub;
1267        is_compare:
1268          alu_out = alu_out_0;
1269        instr_xori || instr_xor:
1270          alu_out = reg_op1 ^ reg_op2;
1271        instr_ori || instr_or:
1272          alu_out = reg_op1 | reg_op2;
```

```
1273        instr_andi || instr_and:
1274          alu_out = reg_op1 & reg_op2;
1275        BARREL_SHIFTER && (instr_sll || instr_slli):
1276          alu_out = alu_shl;
1277        BARREL_SHIFTER && (instr_srl || instr_srli || instr_sra ||
                instr_srai):
1278          alu_out = alu_shr;
1279      endcase
1280
1281  `ifdef RISCV_FORMAL_BLACKBOX_ALU
1282      alu_out_0 = $anyseq;
1283      alu_out = $anyseq;
1284  `endif
1285    end
1286
1287    reg clear_prefetched_high_word_q;
1288    always @(posedge clk) clear_prefetched_high_word_q <=
           clear_prefetched_high_word;
1289
1290    always @* begin
1291      clear_prefetched_high_word = clear_prefetched_high_word_q;
1292      if (!prefetched_high_word)
1293        clear_prefetched_high_word = 0;
1294      if (latched_branch || irq_state || !resetn)
1295        clear_prefetched_high_word = COMPRESSED_ISA;
1296    end
1297
1298    reg cpuregs_write;
1299    reg [31:0] cpuregs_wrdata;
1300    reg [31:0] cpuregs_rs1;
1301    reg [31:0] cpuregs_rs2;
1302    reg [regindex_bits-1:0] decoded_rs;
1303
1304    always @* begin
1305      cpuregs_write = 0;
1306      cpuregs_wrdata = 'bx;
1307
1308      if (cpu_state == cpu_state_fetch) begin
1309        (* parallel_case *)
1310        case (1'b1)
1311          latched_branch: begin
1312            cpuregs_wrdata = reg_pc + (latched_compr ? 2 : 4);
1313            cpuregs_write = 1;
1314          end
1315          latched_store && !latched_branch: begin
1316            cpuregs_wrdata = latched_stalu ? alu_out_q : reg_out;
1317            cpuregs_write = 1;
1318          end
1319          ENABLE_IRQ && irq_state[0]: begin
1320            cpuregs_wrdata = reg_next_pc | latched_compr;
1321            cpuregs_write = 1;
1322          end
1323          ENABLE_IRQ && irq_state[1]: begin
```

```
1324            cpuregs_wrdata = irq_pending & ~irq_mask;
1325            cpuregs_write = 1;
1326          end
1327        endcase
1328      end
1329    end
1330
1331  `ifndef PICORV32_REGS
1332    always @(posedge clk) begin
1333      if (resetn && cpuregs_write && latched_rd)
1334  `ifdef PICORV32_TESTBUG_001
1335        cpuregs[latched_rd ^ 1] <= cpuregs_wrdata;
1336  `elsif PICORV32_TESTBUG_002
1337        cpuregs[latched_rd] <= cpuregs_wrdata ^ 1;
1338  `else
1339        cpuregs[latched_rd] <= cpuregs_wrdata;
1340  `endif
1341    end
1342
1343    always @* begin
1344      decoded_rs = 'bx;
1345      if (ENABLE_REGS_DUALPORT) begin
1346  `ifndef RISCV_FORMAL_BLACKBOX_REGS
1347        cpuregs_rs1 = decoded_rs1 ? cpuregs[decoded_rs1] : 0;
1348        cpuregs_rs2 = decoded_rs2 ? cpuregs[decoded_rs2] : 0;
1349  `else
1350        cpuregs_rs1 = decoded_rs1 ? $anyseq : 0;
1351        cpuregs_rs2 = decoded_rs2 ? $anyseq : 0;
1352  `endif
1353      end else begin
1354        decoded_rs = (cpu_state == cpu_state_ld_rs2) ? decoded_rs2 :
             decoded_rs1;
1355  `ifndef RISCV_FORMAL_BLACKBOX_REGS
1356        cpuregs_rs1 = decoded_rs ? cpuregs[decoded_rs] : 0;
1357  `else
1358        cpuregs_rs1 = decoded_rs ? $anyseq : 0;
1359  `endif
1360        cpuregs_rs2 = cpuregs_rs1;
1361      end
1362    end
1363  `else
1364    wire[31:0] cpuregs_rdata1;
1365    wire[31:0] cpuregs_rdata2;
1366
1367    wire [5:0] cpuregs_waddr = latched_rd;
1368    wire [5:0] cpuregs_raddr1 = ENABLE_REGS_DUALPORT ? decoded_rs1 :
       decoded_rs;
1369    wire [5:0] cpuregs_raddr2 = ENABLE_REGS_DUALPORT ? decoded_rs2 : 0;
1370
1371    `PICORV32_REGS cpuregs (
1372      .clk(clk),
1373      .wen(resetn && cpuregs_write && latched_rd),
1374      .waddr(cpuregs_waddr),
```

```
1375      .raddr1(cpuregs_raddr1),
1376      .raddr2(cpuregs_raddr2),
1377      .wdata(cpuregs_wrdata),
1378      .rdata1(cpuregs_rdata1),
1379      .rdata2(cpuregs_rdata2)
1380    );
1381
1382    always @* begin
1383      decoded_rs = 'bx;
1384      if (ENABLE_REGS_DUALPORT) begin
1385        cpuregs_rs1 = decoded_rs1 ? cpuregs_rdata1 : 0;
1386        cpuregs_rs2 = decoded_rs2 ? cpuregs_rdata2 : 0;
1387      end else begin
1388        decoded_rs = (cpu_state == cpu_state_ld_rs2) ? decoded_rs2 :
             decoded_rs1;
1389        cpuregs_rs1 = decoded_rs ? cpuregs_rdata1 : 0;
1390        cpuregs_rs2 = cpuregs_rs1;
1391      end
1392    end
1393  `endif
1394
1395    assign launch_next_insn = cpu_state == cpu_state_fetch &&
         decoder_trigger && (!ENABLE_IRQ || irq_delay || irq_active ||
         !(irq_pending & ~irq_mask));
1396
1397    always @(posedge clk) begin
1398      trap <= 0;
1399      reg_sh <= 'bx;
1400      reg_out <= 'bx;
1401      set_mem_do_rinst = 0;
1402      set_mem_do_rdata = 0;
1403      set_mem_do_wdata = 0;
1404
1405      alu_out_0_q <= alu_out_0;
1406      alu_out_q <= alu_out;
1407
1408      alu_wait <= 0;
1409      alu_wait_2 <= 0;
1410
1411      if (launch_next_insn) begin
1412        dbg_rs1val <= 'bx;
1413        dbg_rs2val <= 'bx;
1414        dbg_rs1val_valid <= 0;
1415        dbg_rs2val_valid <= 0;
1416      end
1417
1418      if (WITH_PCPI && CATCH_ILLINSN) begin
1419        if (resetn && pcpi_valid && !pcpi_int_wait) begin
1420          if (pcpi_timeout_counter)
1421            pcpi_timeout_counter <= pcpi_timeout_counter - 1;
1422        end else
1423          pcpi_timeout_counter <= ~0;
1424        pcpi_timeout <= !pcpi_timeout_counter;
```

```
1425      end
1426
1427      if (ENABLE_COUNTERS) begin
1428        count_cycle <= resetn ? count_cycle + 1 : 0;
1429        if (!ENABLE_COUNTERS64) count_cycle[63:32] <= 0;
1430      end else begin
1431        count_cycle <= 'bx;
1432        count_instr <= 'bx;
1433      end
1434
1435      next_irq_pending = ENABLE_IRQ ? irq_pending & LATCHED_IRQ : 'bx;
1436
1437      if (ENABLE_IRQ && ENABLE_IRQ_TIMER && timer) begin
1438        timer <= timer - 1;
1439      end
1440
1441      decoder_trigger <= mem_do_rinst && mem_done;
1442      decoder_trigger_q <= decoder_trigger;
1443      decoder_pseudo_trigger <= 0;
1444      decoder_pseudo_trigger_q <= decoder_pseudo_trigger;
1445      do_waitirq <= 0;
1446
1447      trace_valid <= 0;
1448
1449      if (!ENABLE_TRACE)
1450        trace_data <= 'bx;
1451
1452      if (!resetn) begin
1453        reg_pc <= PROGADDR_RESET;
1454        reg_next_pc <= PROGADDR_RESET;
1455        if (ENABLE_COUNTERS)
1456          count_instr <= 0;
1457        latched_store <= 0;
1458        latched_stalu <= 0;
1459        latched_branch <= 0;
1460        latched_trace <= 0;
1461        latched_is_lu <= 0;
1462        latched_is_lh <= 0;
1463        latched_is_lb <= 0;
1464        pcpi_valid <= 0;
1465        pcpi_timeout <= 0;
1466        irq_active <= 0;
1467        irq_delay <= 0;
1468        irq_mask <= ~0;
1469        next_irq_pending = 0;
1470        irq_state <= 0;
1471        eoi <= 0;
1472        timer <= 0;
1473        if (~STACKADDR) begin
1474          latched_store <= 1;
1475          latched_rd <= 2;
1476          reg_out <= STACKADDR;
1477        end
```

```
1478        cpu_state <= cpu_state_fetch;
1479      end else
1480      (* parallel_case, full_case *)
1481      case (cpu_state)
1482        cpu_state_trap: begin
1483          trap <= 1;
1484        end
1485
1486        cpu_state_fetch: begin
1487          mem_do_rinst <= !decoder_trigger && !do_waitirq;
1488          mem_wordsize <= 0;
1489
1490          current_pc = reg_next_pc;
1491
1492          (* parallel_case *)
1493          case (1'b1)
1494            latched_branch: begin
1495              current_pc = latched_store ? (latched_stalu ? alu_out_q :
                     reg_out) & ~1 : reg_next_pc;
1496              `debug($display("ST_RD:  %2d 0x%08x, BRANCH 0x%08x",
                     latched_rd, reg_pc + (latched_compr ? 2 : 4),
                     current_pc);)
1497            end
1498            latched_store && !latched_branch: begin
1499              `debug($display("ST_RD:  %2d 0x%08x", latched_rd,
                     latched_stalu ? alu_out_q : reg_out);)
1500            end
1501            ENABLE_IRQ && irq_state[0]: begin
1502              current_pc = PROGADDR_IRQ;
1503              irq_active <= 1;
1504              mem_do_rinst <= 1;
1505            end
1506            ENABLE_IRQ && irq_state[1]: begin
1507              eoi <= irq_pending & ~irq_mask;
1508              next_irq_pending = next_irq_pending & irq_mask;
1509            end
1510          endcase
1511
1512          if (ENABLE_TRACE && latched_trace) begin
1513            latched_trace <= 0;
1514            trace_valid <= 1;
1515            if (latched_branch)
1516              trace_data <= (irq_active ? TRACE_IRQ : 0) | TRACE_BRANCH |
                     (current_pc & 32'hfffffffe);
1517            else
1518              trace_data <= (irq_active ? TRACE_IRQ : 0) | (latched_stalu
                     ? alu_out_q : reg_out);
1519          end
1520
1521          reg_pc <= current_pc;
1522          reg_next_pc <= current_pc;
1523
1524          latched_store <= 0;
```

```
1525          latched_stalu <= 0;
1526          latched_branch <= 0;
1527          latched_is_lu <= 0;
1528          latched_is_lh <= 0;
1529          latched_is_lb <= 0;
1530          latched_rd <= decoded_rd;
1531          latched_compr <= compressed_instr;
1532
1533          if (ENABLE_IRQ && ((decoder_trigger && !irq_active && !irq_delay
                 && |(irq_pending & ~irq_mask)) || irq_state)) begin
1534            irq_state <=
1535              irq_state == 2'b00 ? 2'b01 :
1536              irq_state == 2'b01 ? 2'b10 : 2'b00;
1537            latched_compr <= latched_compr;
1538            if (ENABLE_IRQ_QREGS)
1539              latched_rd <= irqregs_offset | irq_state[0];
1540            else
1541              latched_rd <= irq_state[0] ? 4 : 3;
1542          end else
1543          if (ENABLE_IRQ && (decoder_trigger || do_waitirq) &&
                 instr_waitirq) begin
1544            if (irq_pending) begin
1545              latched_store <= 1;
1546              reg_out <= irq_pending;
1547              reg_next_pc <= current_pc + (compressed_instr ? 2 : 4);
1548              mem_do_rinst <= 1;
1549            end else
1550              do_waitirq <= 1;
1551          end else
1552          if (decoder_trigger) begin
1553            `debug($display("-- %-0t", $time);)
1554            irq_delay <= irq_active;
1555            reg_next_pc <= current_pc + (compressed_instr ? 2 : 4);
1556            if (ENABLE_TRACE)
1557              latched_trace <= 1;
1558            if (ENABLE_COUNTERS) begin
1559              count_instr <= count_instr + 1;
1560              if (!ENABLE_COUNTERS64) count_instr[63:32] <= 0;
1561            end
1562            if (instr_jal) begin
1563              mem_do_rinst <= 1;
1564              reg_next_pc <= current_pc + decoded_imm_j;
1565              latched_branch <= 1;
1566            end else begin
1567              mem_do_rinst <= 0;
1568              mem_do_prefetch <= !instr_jalr && !instr_retirq;
1569              cpu_state <= cpu_state_ld_rs1;
1570            end
1571          end
1572        end
1573
1574        cpu_state_ld_rs1: begin
1575          reg_op1 <= 'bx;
```

```
1576          reg_op2 <= 'bx;
1577
1578          (* parallel_case *)
1579          case (1'b1)
1580            (CATCH_ILLINSN || WITH_PCPI) && instr_trap: begin
1581              if (WITH_PCPI) begin
1582                `debug($display("LD_RS1: %2d 0x%08x", decoded_rs1,
                      cpuregs_rs1);)
1583                reg_op1 <= cpuregs_rs1;
1584                dbg_rs1val <= cpuregs_rs1;
1585                dbg_rs1val_valid <= 1;
1586                if (ENABLE_REGS_DUALPORT) begin
1587                  pcpi_valid <= 1;
1588                  `debug($display("LD_RS2: %2d 0x%08x", decoded_rs2,
                        cpuregs_rs2);)
1589                  reg_sh <= cpuregs_rs2;
1590                  reg_op2 <= cpuregs_rs2;
1591                  dbg_rs2val <= cpuregs_rs2;
1592                  dbg_rs2val_valid <= 1;
1593                  if (pcpi_int_ready) begin
1594                    mem_do_rinst <= 1;
1595                    pcpi_valid <= 0;
1596                    reg_out <= pcpi_int_rd;
1597                    latched_store <= pcpi_int_wr;
1598                    cpu_state <= cpu_state_fetch;
1599                  end else
1600                  if (CATCH_ILLINSN && (pcpi_timeout ||
                        instr_ecall_ebreak)) begin
1601                    pcpi_valid <= 0;
1602                    `debug($display("EBREAK OR UNSUPPORTED INSN AT
                          0x%08x", reg_pc);)
1603                    if (ENABLE_IRQ && !irq_mask[irq_ebreak] &&
                          !irq_active) begin
1604                      next_irq_pending[irq_ebreak] = 1;
1605                      cpu_state <= cpu_state_fetch;
1606                    end else
1607                      cpu_state <= cpu_state_trap;
1608                  end
1609                end else begin
1610                  cpu_state <= cpu_state_ld_rs2;
1611                end
1612              end else begin
1613                `debug($display("EBREAK OR UNSUPPORTED INSN AT 0x%08x",
                      reg_pc);)
1614                if (ENABLE_IRQ && !irq_mask[irq_ebreak] && !irq_active)
                      begin
1615                  next_irq_pending[irq_ebreak] = 1;
1616                  cpu_state <= cpu_state_fetch;
1617                end else
1618                  cpu_state <= cpu_state_trap;
1619              end
1620            end
1621            ENABLE_COUNTERS && is_rdcycle_rdcycleh_rdinstr_rdinstrh: begin
```

```
1622              (* parallel_case, full_case *)
1623              case (1'b1)
1624                instr_rdcycle:
1625                  reg_out <= count_cycle[31:0];
1626                instr_rdcycleh && ENABLE_COUNTERS64:
1627                  reg_out <= count_cycle[63:32];
1628                instr_rdinstr:
1629                  reg_out <= count_instr[31:0];
1630                instr_rdinstrh && ENABLE_COUNTERS64:
1631                  reg_out <= count_instr[63:32];
1632              endcase
1633              latched_store <= 1;
1634              cpu_state <= cpu_state_fetch;
1635            end
1636            is_lui_auipc_jal: begin
1637              reg_op1 <= instr_lui ? 0 : reg_pc;
1638              reg_op2 <= decoded_imm;
1639              if (TWO_CYCLE_ALU)
1640                alu_wait <= 1;
1641              else
1642                mem_do_rinst <= mem_do_prefetch;
1643              cpu_state <= cpu_state_exec;
1644            end
1645            ENABLE_IRQ && ENABLE_IRQ_QREGS && instr_getq: begin
1646              `debug($display("LD_RS1: %2d 0x%08x", decoded_rs1,
                     cpuregs_rs1);)
1647              reg_out <= cpuregs_rs1;
1648              dbg_rs1val <= cpuregs_rs1;
1649              dbg_rs1val_valid <= 1;
1650              latched_store <= 1;
1651              cpu_state <= cpu_state_fetch;
1652            end
1653            ENABLE_IRQ && ENABLE_IRQ_QREGS && instr_setq: begin
1654              `debug($display("LD_RS1: %2d 0x%08x", decoded_rs1,
                     cpuregs_rs1);)
1655              reg_out <= cpuregs_rs1;
1656              dbg_rs1val <= cpuregs_rs1;
1657              dbg_rs1val_valid <= 1;
1658              latched_rd <= latched_rd | irqregs_offset;
1659              latched_store <= 1;
1660              cpu_state <= cpu_state_fetch;
1661            end
1662            ENABLE_IRQ && instr_retirq: begin
1663              eoi <= 0;
1664              irq_active <= 0;
1665              latched_branch <= 1;
1666              latched_store <= 1;
1667              `debug($display("LD_RS1: %2d 0x%08x", decoded_rs1,
                     cpuregs_rs1);)
1668              reg_out <= CATCH_MISALIGN ? (cpuregs_rs1 & 32'h fffffffe) :
                     cpuregs_rs1;
1669              dbg_rs1val <= cpuregs_rs1;
1670              dbg_rs1val_valid <= 1;
```

```
1671              cpu_state <= cpu_state_fetch;
1672            end
1673            ENABLE_IRQ && instr_maskirq: begin
1674              latched_store <= 1;
1675              reg_out <= irq_mask;
1676              `debug($display("LD_RS1: %2d 0x%08x", decoded_rs1,
                     cpuregs_rs1);)
1677              irq_mask <= cpuregs_rs1 | MASKED_IRQ;
1678              dbg_rs1val <= cpuregs_rs1;
1679              dbg_rs1val_valid <= 1;
1680              cpu_state <= cpu_state_fetch;
1681            end
1682            ENABLE_IRQ && ENABLE_IRQ_TIMER && instr_timer: begin
1683              latched_store <= 1;
1684              reg_out <= timer;
1685              `debug($display("LD_RS1: %2d 0x%08x", decoded_rs1,
                     cpuregs_rs1);)
1686              timer <= cpuregs_rs1;
1687              dbg_rs1val <= cpuregs_rs1;
1688              dbg_rs1val_valid <= 1;
1689              cpu_state <= cpu_state_fetch;
1690            end
1691            is_lb_lh_lw_lbu_lhu && !instr_trap: begin
1692              `debug($display("LD_RS1: %2d 0x%08x", decoded_rs1,
                     cpuregs_rs1);)
1693              reg_op1 <= cpuregs_rs1;
1694              dbg_rs1val <= cpuregs_rs1;
1695              dbg_rs1val_valid <= 1;
1696              cpu_state <= cpu_state_ldmem;
1697              mem_do_rinst <= 1;
1698            end
1699            is_slli_srli_srai && !BARREL_SHIFTER: begin
1700              `debug($display("LD_RS1: %2d 0x%08x", decoded_rs1,
                     cpuregs_rs1);)
1701              reg_op1 <= cpuregs_rs1;
1702              dbg_rs1val <= cpuregs_rs1;
1703              dbg_rs1val_valid <= 1;
1704              reg_sh <= decoded_rs2;
1705              cpu_state <= cpu_state_shift;
1706            end
1707            is_jalr_addi_slti_sltiu_xori_ori_andi, is_slli_srli_srai &&
                  BARREL_SHIFTER: begin
1708              `debug($display("LD_RS1: %2d 0x%08x", decoded_rs1,
                     cpuregs_rs1);)
1709              reg_op1 <= cpuregs_rs1;
1710              dbg_rs1val <= cpuregs_rs1;
1711              dbg_rs1val_valid <= 1;
1712              reg_op2 <= is_slli_srli_srai && BARREL_SHIFTER ? decoded_rs2
                     : decoded_imm;
1713              if (TWO_CYCLE_ALU)
1714                alu_wait <= 1;
1715              else
1716                mem_do_rinst <= mem_do_prefetch;
```

```
1717              cpu_state <= cpu_state_exec;
1718            end
1719            default: begin
1720              `debug($display("LD_RS1: %2d 0x%08x", decoded_rs1,
                     cpuregs_rs1);)
1721              reg_op1 <= cpuregs_rs1;
1722              dbg_rs1val <= cpuregs_rs1;
1723              dbg_rs1val_valid <= 1;
1724              if (ENABLE_REGS_DUALPORT) begin
1725                `debug($display("LD_RS2: %2d 0x%08x", decoded_rs2,
                       cpuregs_rs2);)
1726                reg_sh <= cpuregs_rs2;
1727                reg_op2 <= cpuregs_rs2;
1728                dbg_rs2val <= cpuregs_rs2;
1729                dbg_rs2val_valid <= 1;
1730                (* parallel_case *)
1731                case (1'b1)
1732                  is_sb_sh_sw: begin
1733                    cpu_state <= cpu_state_stmem;
1734                    mem_do_rinst <= 1;
1735                  end
1736                  is_sll_srl_sra && !BARREL_SHIFTER: begin
1737                    cpu_state <= cpu_state_shift;
1738                  end
1739                  default: begin
1740                    if (TWO_CYCLE_ALU || (TWO_CYCLE_COMPARE &&
                           is_beq_bne_blt_bge_bltu_bgeu)) begin
1741                      alu_wait_2 <= TWO_CYCLE_ALU && (TWO_CYCLE_COMPARE &&
                             is_beq_bne_blt_bge_bltu_bgeu);
1742                      alu_wait <= 1;
1743                    end else
1744                      mem_do_rinst <= mem_do_prefetch;
1745                    cpu_state <= cpu_state_exec;
1746                  end
1747                endcase
1748              end else
1749                cpu_state <= cpu_state_ld_rs2;
1750            end
1751          endcase
1752        end
1753
1754        cpu_state_ld_rs2: begin
1755          `debug($display("LD_RS2: %2d 0x%08x", decoded_rs2, cpuregs_rs2);)
1756          reg_sh <= cpuregs_rs2;
1757          reg_op2 <= cpuregs_rs2;
1758          dbg_rs2val <= cpuregs_rs2;
1759          dbg_rs2val_valid <= 1;
1760
1761          (* parallel_case *)
1762          case (1'b1)
1763            WITH_PCPI && instr_trap: begin
1764              pcpi_valid <= 1;
1765              if (pcpi_int_ready) begin
```

```
1766                mem_do_rinst <= 1;
1767                pcpi_valid <= 0;
1768                reg_out <= pcpi_int_rd;
1769                latched_store <= pcpi_int_wr;
1770                cpu_state <= cpu_state_fetch;
1771              end else
1772              if (CATCH_ILLINSN && (pcpi_timeout || instr_ecall_ebreak))
                     begin
1773                pcpi_valid <= 0;
1774                `debug($display("EBREAK OR UNSUPPORTED INSN AT 0x%08x",
                       reg_pc);)
1775                if (ENABLE_IRQ && !irq_mask[irq_ebreak] && !irq_active)
                       begin
1776                  next_irq_pending[irq_ebreak] = 1;
1777                  cpu_state <= cpu_state_fetch;
1778                end else
1779                  cpu_state <= cpu_state_trap;
1780              end
1781            end
1782            is_sb_sh_sw: begin
1783              cpu_state <= cpu_state_stmem;
1784              mem_do_rinst <= 1;
1785            end
1786            is_sll_srl_sra && !BARREL_SHIFTER: begin
1787              cpu_state <= cpu_state_shift;
1788            end
1789            default: begin
1790              if (TWO_CYCLE_ALU || (TWO_CYCLE_COMPARE &&
                       is_beq_bne_blt_bge_bltu_bgeu)) begin
1791                alu_wait_2 <= TWO_CYCLE_ALU && (TWO_CYCLE_COMPARE &&
                         is_beq_bne_blt_bge_bltu_bgeu);
1792                alu_wait <= 1;
1793              end else
1794                mem_do_rinst <= mem_do_prefetch;
1795              cpu_state <= cpu_state_exec;
1796            end
1797          endcase
1798        end
1799
1800        cpu_state_exec: begin
1801          reg_out <= reg_pc + decoded_imm;
1802          if ((TWO_CYCLE_ALU || TWO_CYCLE_COMPARE) && (alu_wait ||
                 alu_wait_2)) begin
1803            mem_do_rinst <= mem_do_prefetch && !alu_wait_2;
1804            alu_wait <= alu_wait_2;
1805          end else
1806          if (is_beq_bne_blt_bge_bltu_bgeu) begin
1807            latched_rd <= 0;
1808            latched_store <= TWO_CYCLE_COMPARE ? alu_out_0_q : alu_out_0;
1809            latched_branch <= TWO_CYCLE_COMPARE ? alu_out_0_q : alu_out_0;
1810            if (mem_done)
1811              cpu_state <= cpu_state_fetch;
1812            if (TWO_CYCLE_COMPARE ? alu_out_0_q : alu_out_0) begin
```

```
1813              decoder_trigger <= 0;
1814              set_mem_do_rinst = 1;
1815            end
1816          end else begin
1817            latched_branch <= instr_jalr;
1818            latched_store <= 1;
1819            latched_stalu <= 1;
1820            cpu_state <= cpu_state_fetch;
1821          end
1822        end
1823
1824        cpu_state_shift: begin
1825          latched_store <= 1;
1826          if (reg_sh == 0) begin
1827            reg_out <= reg_op1;
1828            mem_do_rinst <= mem_do_prefetch;
1829            cpu_state <= cpu_state_fetch;
1830          end else if (TWO_STAGE_SHIFT && reg_sh >= 4) begin
1831            (* parallel_case, full_case *)
1832            case (1'b1)
1833              instr_slli || instr_sll: reg_op1 <= reg_op1 << 4;
1834              instr_srli || instr_srl: reg_op1 <= reg_op1 >> 4;
1835              instr_srai || instr_sra: reg_op1 <= $signed(reg_op1) >>> 4;
1836            endcase
1837            reg_sh <= reg_sh - 4;
1838          end else begin
1839            (* parallel_case, full_case *)
1840            case (1'b1)
1841              instr_slli || instr_sll: reg_op1 <= reg_op1 << 1;
1842              instr_srli || instr_srl: reg_op1 <= reg_op1 >> 1;
1843              instr_srai || instr_sra: reg_op1 <= $signed(reg_op1) >>> 1;
1844            endcase
1845            reg_sh <= reg_sh - 1;
1846          end
1847        end
1848
1849        cpu_state_stmem: begin
1850          if (ENABLE_TRACE)
1851            reg_out <= reg_op2;
1852          if (!mem_do_prefetch || mem_done) begin
1853            if (!mem_do_wdata) begin
1854              (* parallel_case, full_case *)
1855              case (1'b1)
1856                instr_sb: mem_wordsize <= 2;
1857                instr_sh: mem_wordsize <= 1;
1858                instr_sw: mem_wordsize <= 0;
1859              endcase
1860              if (ENABLE_TRACE) begin
1861                trace_valid <= 1;
1862                trace_data <= (irq_active ? TRACE_IRQ : 0) | TRACE_ADDR |
                       ((reg_op1 + decoded_imm) & 32'hffffffff);
1863              end
1864              reg_op1 <= reg_op1 + decoded_imm;
```

```
1865            set_mem_do_wdata = 1;
1866          end
1867          if (!mem_do_prefetch && mem_done) begin
1868            cpu_state <= cpu_state_fetch;
1869            decoder_trigger <= 1;
1870            decoder_pseudo_trigger <= 1;
1871          end
1872        end
1873      end
1874
1875      cpu_state_ldmem: begin
1876        latched_store <= 1;
1877        if (!mem_do_prefetch || mem_done) begin
1878          if (!mem_do_rdata) begin
1879            (* parallel_case, full_case *)
1880            case (1'b1)
1881              instr_lb || instr_lbu: mem_wordsize <= 2;
1882              instr_lh || instr_lhu: mem_wordsize <= 1;
1883              instr_lw: mem_wordsize <= 0;
1884            endcase
1885            latched_is_lu <= is_lbu_lhu_lw;
1886            latched_is_lh <= instr_lh;
1887            latched_is_lb <= instr_lb;
1888            if (ENABLE_TRACE) begin
1889              trace_valid <= 1;
1890              trace_data <= (irq_active ? TRACE_IRQ : 0) | TRACE_ADDR |
                     ((reg_op1 + decoded_imm) & 32'hffffffff);
1891            end
1892            reg_op1 <= reg_op1 + decoded_imm;
1893            set_mem_do_rdata = 1;
1894          end
1895          if (!mem_do_prefetch && mem_done) begin
1896            (* parallel_case, full_case *)
1897            case (1'b1)
1898              latched_is_lu: reg_out <= mem_rdata_word;
1899              latched_is_lh: reg_out <= $signed(mem_rdata_word[15:0]);
1900              latched_is_lb: reg_out <= $signed(mem_rdata_word[7:0]);
1901            endcase
1902            decoder_trigger <= 1;
1903            decoder_pseudo_trigger <= 1;
1904            cpu_state <= cpu_state_fetch;
1905          end
1906        end
1907      end
1908    endcase
1909
1910    if (ENABLE_IRQ) begin
1911      next_irq_pending = next_irq_pending | irq;
1912      if(ENABLE_IRQ_TIMER && timer)
1913        if (timer - 1 == 0)
1914          next_irq_pending[irq_timer] = 1;
1915    end
1916
```

```
1917      if (CATCH_MISALIGN && resetn && (mem_do_rdata || mem_do_wdata)) begin
1918        if (mem_wordsize == 0 && reg_op1[1:0] != 0) begin
1919          `debug($display("MISALIGNED WORD: 0x%08x", reg_op1);)
1920          if (ENABLE_IRQ && !irq_mask[irq_buserror] && !irq_active) begin
1921            next_irq_pending[irq_buserror] = 1;
1922          end else
1923            cpu_state <= cpu_state_trap;
1924        end
1925        if (mem_wordsize == 1 && reg_op1[0] != 0) begin
1926          `debug($display("MISALIGNED HALFWORD: 0x%08x", reg_op1);)
1927          if (ENABLE_IRQ && !irq_mask[irq_buserror] && !irq_active) begin
1928            next_irq_pending[irq_buserror] = 1;
1929          end else
1930            cpu_state <= cpu_state_trap;
1931        end
1932      end
1933      if (CATCH_MISALIGN && resetn && mem_do_rinst && (COMPRESSED_ISA ?
           reg_pc[0] : |reg_pc[1:0])) begin
1934        `debug($display("MISALIGNED INSTRUCTION: 0x%08x", reg_pc);)
1935        if (ENABLE_IRQ && !irq_mask[irq_buserror] && !irq_active) begin
1936          next_irq_pending[irq_buserror] = 1;
1937        end else
1938          cpu_state <= cpu_state_trap;
1939      end
1940      if (!CATCH_ILLINSN && decoder_trigger_q && !decoder_pseudo_trigger_q
           && instr_ecall_ebreak) begin
1941        cpu_state <= cpu_state_trap;
1942      end
1943
1944      if (!resetn || mem_done) begin
1945        mem_do_prefetch <= 0;
1946        mem_do_rinst <= 0;
1947        mem_do_rdata <= 0;
1948        mem_do_wdata <= 0;
1949      end
1950
1951      if (set_mem_do_rinst)
1952        mem_do_rinst <= 1;
1953      if (set_mem_do_rdata)
1954        mem_do_rdata <= 1;
1955      if (set_mem_do_wdata)
1956        mem_do_wdata <= 1;
1957
1958      irq_pending <= next_irq_pending & ~MASKED_IRQ;
1959
1960      if (!CATCH_MISALIGN) begin
1961        if (COMPRESSED_ISA) begin
1962          reg_pc[0] <= 0;
1963          reg_next_pc[0] <= 0;
1964        end else begin
1965          reg_pc[1:0] <= 0;
1966          reg_next_pc[1:0] <= 0;
1967        end
```

```
1968      end
1969      current_pc = 'bx;
1970    end
1971
1972  `ifdef RISCV_FORMAL
1973    reg dbg_irq_call;
1974    reg dbg_irq_enter;
1975    reg [31:0] dbg_irq_ret;
1976    always @(posedge clk) begin
1977      rvfi_valid <= resetn && (launch_next_insn || trap) && dbg_valid_insn;
1978      rvfi_order <= resetn ? rvfi_order + rvfi_valid : 0;
1979
1980      rvfi_insn <= dbg_insn_opcode;
1981      rvfi_rs1_addr <= dbg_rs1val_valid ? dbg_insn_rs1 : 0;
1982      rvfi_rs2_addr <= dbg_rs2val_valid ? dbg_insn_rs2 : 0;
1983      rvfi_pc_rdata <= dbg_insn_addr;
1984      rvfi_rs1_rdata <= dbg_rs1val_valid ? dbg_rs1val : 0;
1985      rvfi_rs2_rdata <= dbg_rs2val_valid ? dbg_rs2val : 0;
1986      rvfi_trap <= trap;
1987      rvfi_halt <= trap;
1988      rvfi_intr <= dbg_irq_enter;
1989      rvfi_mode <= 3;
1990      rvfi_ixl <= 1;
1991
1992      if (!resetn) begin
1993        dbg_irq_call <= 0;
1994        dbg_irq_enter <= 0;
1995      end else
1996      if (rvfi_valid) begin
1997        dbg_irq_call <= 0;
1998        dbg_irq_enter <= dbg_irq_call;
1999      end else
2000      if (irq_state == 1) begin
2001        dbg_irq_call <= 1;
2002        dbg_irq_ret <= next_pc;
2003      end
2004
2005      if (!resetn) begin
2006        rvfi_rd_addr <= 0;
2007        rvfi_rd_wdata <= 0;
2008      end else
2009      if (cpuregs_write && !irq_state) begin
2010  `ifdef PICORV32_TESTBUG_003
2011        rvfi_rd_addr <= latched_rd ^ 1;
2012  `else
2013        rvfi_rd_addr <= latched_rd;
2014  `endif
2015  `ifdef PICORV32_TESTBUG_004
2016        rvfi_rd_wdata <= latched_rd ? cpuregs_wrdata ^ 1 : 0;
2017  `else
2018        rvfi_rd_wdata <= latched_rd ? cpuregs_wrdata : 0;
2019  `endif
2020      end else
```

```
2021      if (rvfi_valid) begin
2022        rvfi_rd_addr <= 0;
2023        rvfi_rd_wdata <= 0;
2024      end
2025
2026      casez (dbg_insn_opcode)
2027        32'b 0000000_?????_000??_???_?????_0001011: begin // getq
2028          rvfi_rs1_addr <= 0;
2029          rvfi_rs1_rdata <= 0;
2030        end
2031        32'b 0000001_?????_?????_???_000??_0001011: begin // setq
2032          rvfi_rd_addr <= 0;
2033          rvfi_rd_wdata <= 0;
2034        end
2035        32'b 0000010_?????_00000_???_00000_0001011: begin // retirq
2036          rvfi_rs1_addr <= 0;
2037          rvfi_rs1_rdata <= 0;
2038        end
2039      endcase
2040
2041      if (!dbg_irq_call) begin
2042        if (dbg_mem_instr) begin
2043          rvfi_mem_addr <= 0;
2044          rvfi_mem_rmask <= 0;
2045          rvfi_mem_wmask <= 0;
2046          rvfi_mem_rdata <= 0;
2047          rvfi_mem_wdata <= 0;
2048        end else
2049        if (dbg_mem_valid && dbg_mem_ready) begin
2050          rvfi_mem_addr <= dbg_mem_addr;
2051          rvfi_mem_rmask <= dbg_mem_wstrb ? 0 : ~0;
2052          rvfi_mem_wmask <= dbg_mem_wstrb;
2053          rvfi_mem_rdata <= dbg_mem_rdata;
2054          rvfi_mem_wdata <= dbg_mem_wdata;
2055        end
2056      end
2057    end
2058
2059    always @* begin
2060  `ifdef PICORV32_TESTBUG_005
2061      rvfi_pc_wdata = (dbg_irq_call ? dbg_irq_ret : dbg_insn_addr) ^ 4;
2062  `else
2063      rvfi_pc_wdata = dbg_irq_call ? dbg_irq_ret : dbg_insn_addr;
2064  `endif
2065
2066      rvfi_csr_mcycle_rmask = 0;
2067      rvfi_csr_mcycle_wmask = 0;
2068      rvfi_csr_mcycle_rdata = 0;
2069      rvfi_csr_mcycle_wdata = 0;
2070
2071      rvfi_csr_minstret_rmask = 0;
2072      rvfi_csr_minstret_wmask = 0;
2073      rvfi_csr_minstret_rdata = 0;
```

```
2074      rvfi_csr_minstret_wdata = 0;
2075
2076      if (rvfi_valid && rvfi_insn[6:0] == 7'b 1110011 && rvfi_insn[13:12]
              == 3'b010) begin
2077        if (rvfi_insn[31:20] == 12'h C00) begin
2078          rvfi_csr_mcycle_rmask = 64'h 0000_0000_FFFF_FFFF;
2079          rvfi_csr_mcycle_rdata = {32'h 0000_0000, rvfi_rd_wdata};
2080        end
2081        if (rvfi_insn[31:20] == 12'h C80) begin
2082          rvfi_csr_mcycle_rmask = 64'h FFFF_FFFF_0000_0000;
2083          rvfi_csr_mcycle_rdata = {rvfi_rd_wdata, 32'h 0000_0000};
2084        end
2085        if (rvfi_insn[31:20] == 12'h C02) begin
2086          rvfi_csr_minstret_rmask = 64'h 0000_0000_FFFF_FFFF;
2087          rvfi_csr_minstret_rdata = {32'h 0000_0000, rvfi_rd_wdata};
2088        end
2089        if (rvfi_insn[31:20] == 12'h C82) begin
2090          rvfi_csr_minstret_rmask = 64'h FFFF_FFFF_0000_0000;
2091          rvfi_csr_minstret_rdata = {rvfi_rd_wdata, 32'h 0000_0000};
2092        end
2093      end
2094    end
2095  `endif
2096
2097    // Formal Verification
2098  `ifdef FORMAL
2099    reg [3:0] last_mem_nowait;
2100    always @(posedge clk)
2101      last_mem_nowait <= {last_mem_nowait, mem_ready || !mem_valid};
2102
2103    // stall the memory interface for max 4 cycles
2104    restrict property (|last_mem_nowait || mem_ready || !mem_valid);
2105
2106    // resetn low in first cycle, after that resetn high
2107    restrict property (resetn != $initstate);
2108
2109    // this just makes it much easier to read traces. uncomment as needed.
2110    // assume property (mem_valid || !mem_ready);
2111
2112    reg ok;
2113    always @* begin
2114      if (resetn) begin
2115        // instruction fetches are read-only
2116        if (mem_valid && mem_instr)
2117          assert (mem_wstrb == 0);
2118
2119        // cpu_state must be valid
2120        ok = 0;
2121        if (cpu_state == cpu_state_trap)   ok = 1;
2122        if (cpu_state == cpu_state_fetch)  ok = 1;
2123        if (cpu_state == cpu_state_ld_rs1) ok = 1;
2124        if (cpu_state == cpu_state_ld_rs2) ok = !ENABLE_REGS_DUALPORT;
2125        if (cpu_state == cpu_state_exec)   ok = 1;
```

```
2126        if (cpu_state == cpu_state_shift)  ok = 1;
2127        if (cpu_state == cpu_state_stmem)  ok = 1;
2128        if (cpu_state == cpu_state_ldmem)  ok = 1;
2129        assert (ok);
2130      end
2131    end
2132
2133    reg last_mem_la_read = 0;
2134    reg last_mem_la_write = 0;
2135    reg [31:0] last_mem_la_addr;
2136    reg [31:0] last_mem_la_wdata;
2137    reg [3:0] last_mem_la_wstrb = 0;
2138
2139    always @(posedge clk) begin
2140      last_mem_la_read <= mem_la_read;
2141      last_mem_la_write <= mem_la_write;
2142      last_mem_la_addr <= mem_la_addr;
2143      last_mem_la_wdata <= mem_la_wdata;
2144      last_mem_la_wstrb <= mem_la_wstrb;
2145
2146      if (last_mem_la_read) begin
2147        assert(mem_valid);
2148        assert(mem_addr == last_mem_la_addr);
2149        assert(mem_wstrb == 0);
2150      end
2151      if (last_mem_la_write) begin
2152        assert(mem_valid);
2153        assert(mem_addr == last_mem_la_addr);
2154        assert(mem_wdata == last_mem_la_wdata);
2155        assert(mem_wstrb == last_mem_la_wstrb);
2156      end
2157      if (mem_la_read || mem_la_write) begin
2158        assert(!mem_valid || mem_ready);
2159      end
2160    end
2161  `endif
2162  endmodule
2163
2164  // This is a simple example implementation of PICORV32_REGS.
2165  // Use the PICORV32_REGS mechanism if you want to use custom
2166  // memory resources to implement the processor register file.
2167  // Note that your implementation must match the requirements of
2168  // the PicoRV32 configuration. (e.g. QREGS, etc)
2169  module picorv32_regs (
2170    input clk, wen,
2171    input [5:0] waddr,
2172    input [5:0] raddr1,
2173    input [5:0] raddr2,
2174    input [31:0] wdata,
2175    output [31:0] rdata1,
2176    output [31:0] rdata2
2177  );
2178    reg [31:0] regs [0:30];
```

```
2179
2180   always @(posedge clk)
2181     if (wen) regs[~waddr[4:0]] <= wdata;
2182
2183   assign rdata1 = regs[~raddr1[4:0]];
2184   assign rdata2 = regs[~raddr2[4:0]];
2185 endmodule
2186
2187
2188 /***************************************************************
2189  * picorv32_pcpi_mul
2190  ***************************************************************/
2191
2192 module picorv32_pcpi_mul #(
2193   parameter STEPS_AT_ONCE = 1,
2194   parameter CARRY_CHAIN = 4
2195 ) (
2196   input clk, resetn,
2197
2198   input             pcpi_valid,
2199   input      [31:0] pcpi_insn,
2200   input      [31:0] pcpi_rs1,
2201   input      [31:0] pcpi_rs2,
2202   output reg        pcpi_wr,
2203   output reg [31:0] pcpi_rd,
2204   output reg        pcpi_wait,
2205   output reg        pcpi_ready
2206 );
2207   reg instr_mul, instr_mulh, instr_mulhsu, instr_mulhu;
2208   wire instr_any_mul = |{instr_mul, instr_mulh, instr_mulhsu,
         instr_mulhu};
2209   wire instr_any_mulh = |{instr_mulh, instr_mulhsu, instr_mulhu};
2210   wire instr_rs1_signed = |{instr_mulh, instr_mulhsu};
2211   wire instr_rs2_signed = |{instr_mulh};
2212
2213   reg pcpi_wait_q;
2214   wire mul_start = pcpi_wait && !pcpi_wait_q;
2215
2216   always @(posedge clk) begin
2217     instr_mul <= 0;
2218     instr_mulh <= 0;
2219     instr_mulhsu <= 0;
2220     instr_mulhu <= 0;
2221
2222     if (resetn && pcpi_valid && pcpi_insn[6:0] == 7'b0110011 &&
          pcpi_insn[31:25] == 7'b0000001) begin
2223       case (pcpi_insn[14:12])
2224         3'b000: instr_mul <= 1;
2225         3'b001: instr_mulh <= 1;
2226         3'b010: instr_mulhsu <= 1;
2227         3'b011: instr_mulhu <= 1;
2228       endcase
2229     end
```

```
2230
2231      pcpi_wait <= instr_any_mul;
2232      pcpi_wait_q <= pcpi_wait;
2233    end
2234
2235    reg [63:0] rs1, rs2, rd, rdx;
2236    reg [63:0] next_rs1, next_rs2, this_rs2;
2237    reg [63:0] next_rd, next_rdx, next_rdt;
2238    reg [6:0] mul_counter;
2239    reg mul_waiting;
2240    reg mul_finish;
2241    integer i, j;
2242
2243    // carry save accumulator
2244    always @* begin
2245      next_rd = rd;
2246      next_rdx = rdx;
2247      next_rs1 = rs1;
2248      next_rs2 = rs2;
2249
2250      for (i = 0; i < STEPS_AT_ONCE; i=i+1) begin
2251        this_rs2 = next_rs1[0] ? next_rs2 : 0;
2252        if (CARRY_CHAIN == 0) begin
2253          next_rdt = next_rd ^ next_rdx ^ this_rs2;
2254          next_rdx = ((next_rd & next_rdx) | (next_rd & this_rs2) |
                (next_rdx & this_rs2)) << 1;
2255          next_rd = next_rdt;
2256        end else begin
2257          next_rdt = 0;
2258          for (j = 0; j < 64; j = j + CARRY_CHAIN)
2259            {next_rdt[j+CARRY_CHAIN-1], next_rd[j +: CARRY_CHAIN]} =
2260                next_rd[j +: CARRY_CHAIN] + next_rdx[j +: CARRY_CHAIN] +
                    this_rs2[j +: CARRY_CHAIN];
2261          next_rdx = next_rdt << 1;
2262        end
2263        next_rs1 = next_rs1 >> 1;
2264        next_rs2 = next_rs2 << 1;
2265      end
2266    end
2267
2268    always @(posedge clk) begin
2269      mul_finish <= 0;
2270      if (!resetn) begin
2271        mul_waiting <= 1;
2272      end else
2273      if (mul_waiting) begin
2274        if (instr_rs1_signed)
2275          rs1 <= $signed(pcpi_rs1);
2276        else
2277          rs1 <= $unsigned(pcpi_rs1);
2278
2279        if (instr_rs2_signed)
2280          rs2 <= $signed(pcpi_rs2);
```

```
2281        else
2282          rs2 <= $unsigned(pcpi_rs2);
2283
2284        rd <= 0;
2285        rdx <= 0;
2286        mul_counter <= (instr_any_mulh ? 63 - STEPS_AT_ONCE : 31 -
               STEPS_AT_ONCE);
2287        mul_waiting <= !mul_start;
2288      end else begin
2289        rd <= next_rd;
2290        rdx <= next_rdx;
2291        rs1 <= next_rs1;
2292        rs2 <= next_rs2;
2293
2294        mul_counter <= mul_counter - STEPS_AT_ONCE;
2295        if (mul_counter[6]) begin
2296          mul_finish <= 1;
2297          mul_waiting <= 1;
2298        end
2299      end
2300    end
2301
2302    always @(posedge clk) begin
2303      pcpi_wr <= 0;
2304      pcpi_ready <= 0;
2305      if (mul_finish && resetn) begin
2306        pcpi_wr <= 1;
2307        pcpi_ready <= 1;
2308        pcpi_rd <= instr_any_mulh ? rd >> 32 : rd;
2309      end
2310    end
2311  endmodule
2312
2313  module picorv32_pcpi_fast_mul #(
2314    parameter EXTRA_MUL_FFS = 0,
2315    parameter EXTRA_INSN_FFS = 0,
2316    parameter MUL_CLKGATE = 0
2317  ) (
2318    input clk, resetn,
2319
2320    input             pcpi_valid,
2321    input      [31:0] pcpi_insn,
2322    input      [31:0] pcpi_rs1,
2323    input      [31:0] pcpi_rs2,
2324    output            pcpi_wr,
2325    output     [31:0] pcpi_rd,
2326    output            pcpi_wait,
2327    output            pcpi_ready
2328  );
2329    reg instr_mul, instr_mulh, instr_mulhsu, instr_mulhu;
2330    wire instr_any_mul = |{instr_mul, instr_mulh, instr_mulhsu,
          instr_mulhu};
2331    wire instr_any_mulh = |{instr_mulh, instr_mulhsu, instr_mulhu};
```

```
2332  wire instr_rs1_signed = |{instr_mulh, instr_mulhsu};
2333  wire instr_rs2_signed = |{instr_mulh};
2334
2335  reg shift_out;
2336  reg [3:0] active;
2337  reg [32:0] rs1, rs2, rs1_q, rs2_q;
2338  reg [63:0] rd, rd_q;
2339
2340  wire pcpi_insn_valid = pcpi_valid && pcpi_insn[6:0] == 7'b0110011 &&
          pcpi_insn[31:25] == 7'b0000001;
2341  reg pcpi_insn_valid_q;
2342
2343  always @* begin
2344    instr_mul = 0;
2345    instr_mulh = 0;
2346    instr_mulhsu = 0;
2347    instr_mulhu = 0;
2348
2349    if (resetn && (EXTRA_INSN_FFS ? pcpi_insn_valid_q :
          pcpi_insn_valid)) begin
2350      case (pcpi_insn[14:12])
2351        3'b000: instr_mul = 1;
2352        3'b001: instr_mulh = 1;
2353        3'b010: instr_mulhsu = 1;
2354        3'b011: instr_mulhu = 1;
2355      endcase
2356    end
2357  end
2358
2359  always @(posedge clk) begin
2360    pcpi_insn_valid_q <= pcpi_insn_valid;
2361    if (!MUL_CLKGATE || active[0]) begin
2362      rs1_q <= rs1;
2363      rs2_q <= rs2;
2364    end
2365    if (!MUL_CLKGATE || active[1]) begin
2366      rd <= $signed(EXTRA_MUL_FFS ? rs1_q : rs1) * $signed(EXTRA_MUL_FFS
          ? rs2_q : rs2);
2367    end
2368    if (!MUL_CLKGATE || active[2]) begin
2369      rd_q <= rd;
2370    end
2371  end
2372
2373  always @(posedge clk) begin
2374    if (instr_any_mul && !(EXTRA_MUL_FFS ? active[3:0] : active[1:0]))
          begin
2375      if (instr_rs1_signed)
2376        rs1 <= $signed(pcpi_rs1);
2377      else
2378        rs1 <= $unsigned(pcpi_rs1);
2379
2380      if (instr_rs2_signed)
```

```
2381          rs2 <= $signed(pcpi_rs2);
2382        else
2383          rs2 <= $unsigned(pcpi_rs2);
2384        active[0] <= 1;
2385      end else begin
2386        active[0] <= 0;
2387      end
2388
2389      active[3:1] <= active;
2390      shift_out <= instr_any_mulh;
2391
2392      if (!resetn)
2393        active <= 0;
2394    end
2395
2396    assign pcpi_wr = active[EXTRA_MUL_FFS ? 3 : 1];
2397    assign pcpi_wait = 0;
2398    assign pcpi_ready = active[EXTRA_MUL_FFS ? 3 : 1];
2399  `ifdef RISCV_FORMAL_ALTOPS
2400    assign pcpi_rd =
2401        instr_mul    ? (pcpi_rs1 + pcpi_rs2) ^ 32'h5876063e :
2402        instr_mulh   ? (pcpi_rs1 + pcpi_rs2) ^ 32'hf6583fb7 :
2403        instr_mulhsu ? (pcpi_rs1 - pcpi_rs2) ^ 32'hecfbe137 :
2404        instr_mulhu  ? (pcpi_rs1 + pcpi_rs2) ^ 32'h949ce5e8 : 1'bx;
2405  `else
2406    assign pcpi_rd = shift_out ? (EXTRA_MUL_FFS ? rd_q : rd) >> 32 :
         (EXTRA_MUL_FFS ? rd_q : rd);
2407  `endif
2408  endmodule
2409
2410
2411  /***************************************************************
2412   * picorv32_pcpi_div
2413   ***************************************************************/
2414
2415  module picorv32_pcpi_div (
2416    input clk, resetn,
2417
2418    input             pcpi_valid,
2419    input      [31:0] pcpi_insn,
2420    input      [31:0] pcpi_rs1,
2421    input      [31:0] pcpi_rs2,
2422    output reg        pcpi_wr,
2423    output reg [31:0] pcpi_rd,
2424    output reg        pcpi_wait,
2425    output reg        pcpi_ready
2426  );
2427    reg instr_div, instr_divu, instr_rem, instr_remu;
2428    wire instr_any_div_rem = |{instr_div, instr_divu, instr_rem,
         instr_remu};
2429
2430    reg pcpi_wait_q;
2431    wire start = pcpi_wait && !pcpi_wait_q;
```

```
2432
2433    always @(posedge clk) begin
2434      instr_div <= 0;
2435      instr_divu <= 0;
2436      instr_rem <= 0;
2437      instr_remu <= 0;
2438
2439      if (resetn && pcpi_valid && !pcpi_ready && pcpi_insn[6:0] ==
             7'b0110011 && pcpi_insn[31:25] == 7'b0000001) begin
2440        case (pcpi_insn[14:12])
2441          3'b100: instr_div <= 1;
2442          3'b101: instr_divu <= 1;
2443          3'b110: instr_rem <= 1;
2444          3'b111: instr_remu <= 1;
2445        endcase
2446      end
2447
2448      pcpi_wait <= instr_any_div_rem && resetn;
2449      pcpi_wait_q <= pcpi_wait && resetn;
2450    end
2451
2452    reg [31:0] dividend;
2453    reg [62:0] divisor;
2454    reg [31:0] quotient;
2455    reg [31:0] quotient_msk;
2456    reg running;
2457    reg outsign;
2458
2459    always @(posedge clk) begin
2460      pcpi_ready <= 0;
2461      pcpi_wr <= 0;
2462      pcpi_rd <= 'bx;
2463
2464      if (!resetn) begin
2465        running <= 0;
2466      end else
2467      if (start) begin
2468        running <= 1;
2469        dividend <= (instr_div || instr_rem) && pcpi_rs1[31] ? -pcpi_rs1 :
               pcpi_rs1;
2470        divisor <= ((instr_div || instr_rem) && pcpi_rs2[31] ? -pcpi_rs2 :
               pcpi_rs2) << 31;
2471        outsign <= (instr_div && (pcpi_rs1[31] != pcpi_rs2[31]) &&
               |pcpi_rs2) || (instr_rem && pcpi_rs1[31]);
2472        quotient <= 0;
2473        quotient_msk <= 1 << 31;
2474      end else
2475      if (!quotient_msk && running) begin
2476        running <= 0;
2477        pcpi_ready <= 1;
2478        pcpi_wr <= 1;
2479  `ifdef RISCV_FORMAL_ALTOPS
2480        case (1)
```

```
2481          instr_div:  pcpi_rd <= (pcpi_rs1 - pcpi_rs2) ^ 32'h7f8529ec;
2482          instr_divu: pcpi_rd <= (pcpi_rs1 - pcpi_rs2) ^ 32'h10e8fd70;
2483          instr_rem:  pcpi_rd <= (pcpi_rs1 - pcpi_rs2) ^ 32'h8da68fa5;
2484          instr_remu: pcpi_rd <= (pcpi_rs1 - pcpi_rs2) ^ 32'h3138d0e1;
2485        endcase
2486 `else
2487        if (instr_div || instr_divu)
2488          pcpi_rd <= outsign ? -quotient : quotient;
2489        else
2490          pcpi_rd <= outsign ? -dividend : dividend;
2491 `endif
2492      end else begin
2493        if (divisor <= dividend) begin
2494          dividend <= dividend - divisor;
2495          quotient <= quotient | quotient_msk;
2496        end
2497        divisor <= divisor >> 1;
2498 `ifdef RISCV_FORMAL_ALTOPS
2499        quotient_msk <= quotient_msk >> 5;
2500 `else
2501        quotient_msk <= quotient_msk >> 1;
2502 `endif
2503      end
2504    end
2505 endmodule
2506
2507
2508 /***************************************************************
2509  * picorv32_axi
2510  ***************************************************************/
2511
2512 module picorv32_axi #(
2513    parameter [ 0:0] ENABLE_COUNTERS = 1,
2514    parameter [ 0:0] ENABLE_COUNTERS64 = 1,
2515    parameter [ 0:0] ENABLE_REGS_16_31 = 1,
2516    parameter [ 0:0] ENABLE_REGS_DUALPORT = 1,
2517    parameter [ 0:0] TWO_STAGE_SHIFT = 1,
2518    parameter [ 0:0] BARREL_SHIFTER = 0,
2519    parameter [ 0:0] TWO_CYCLE_COMPARE = 0,
2520    parameter [ 0:0] TWO_CYCLE_ALU = 0,
2521    parameter [ 0:0] COMPRESSED_ISA = 0,
2522    parameter [ 0:0] CATCH_MISALIGN = 1,
2523    parameter [ 0:0] CATCH_ILLINSN = 1,
2524    parameter [ 0:0] ENABLE_PCPI = 0,
2525    parameter [ 0:0] ENABLE_MUL = 0,
2526    parameter [ 0:0] ENABLE_FAST_MUL = 0,
2527    parameter [ 0:0] ENABLE_DIV = 0,
2528    parameter [ 0:0] ENABLE_IRQ = 0,
2529    parameter [ 0:0] ENABLE_IRQ_QREGS = 1,
2530    parameter [ 0:0] ENABLE_IRQ_TIMER = 1,
2531    parameter [ 0:0] ENABLE_TRACE = 0,
2532    parameter [ 0:0] REGS_INIT_ZERO = 0,
2533    parameter [31:0] MASKED_IRQ = 32'h 0000_0000,
```

```
2534    parameter [31:0] LATCHED_IRQ = 32'h ffff_ffff,
2535    parameter [31:0] PROGADDR_RESET = 32'h 0000_0000,
2536    parameter [31:0] PROGADDR_IRQ = 32'h 0000_0010,
2537    parameter [31:0] STACKADDR = 32'h ffff_ffff
2538  ) (
2539    input clk, resetn,
2540    output trap,
2541
2542    // AXI4-lite master memory interface
2543
2544    output        mem_axi_awvalid,
2545    input         mem_axi_awready,
2546    output [31:0] mem_axi_awaddr,
2547    output [ 2:0] mem_axi_awprot,
2548
2549    output        mem_axi_wvalid,
2550    input         mem_axi_wready,
2551    output [31:0] mem_axi_wdata,
2552    output [ 3:0] mem_axi_wstrb,
2553
2554    input         mem_axi_bvalid,
2555    output        mem_axi_bready,
2556
2557    output        mem_axi_arvalid,
2558    input         mem_axi_arready,
2559    output [31:0] mem_axi_araddr,
2560    output [ 2:0] mem_axi_arprot,
2561
2562    input         mem_axi_rvalid,
2563    output        mem_axi_rready,
2564    input  [31:0] mem_axi_rdata,
2565
2566    // Pico Co-Processor Interface (PCPI)
2567    output        pcpi_valid,
2568    output [31:0] pcpi_insn,
2569    output [31:0] pcpi_rs1,
2570    output [31:0] pcpi_rs2,
2571    input         pcpi_wr,
2572    input  [31:0] pcpi_rd,
2573    input         pcpi_wait,
2574    input         pcpi_ready,
2575
2576    // IRQ interface
2577    input  [31:0] irq,
2578    output [31:0] eoi,
2579
2580  `ifdef RISCV_FORMAL
2581    output        rvfi_valid,
2582    output [63:0] rvfi_order,
2583    output [31:0] rvfi_insn,
2584    output        rvfi_trap,
2585    output        rvfi_halt,
2586    output        rvfi_intr,
```

```
2587   output [ 4:0] rvfi_rs1_addr,
2588   output [ 4:0] rvfi_rs2_addr,
2589   output [31:0] rvfi_rs1_rdata,
2590   output [31:0] rvfi_rs2_rdata,
2591   output [ 4:0] rvfi_rd_addr,
2592   output [31:0] rvfi_rd_wdata,
2593   output [31:0] rvfi_pc_rdata,
2594   output [31:0] rvfi_pc_wdata,
2595   output [31:0] rvfi_mem_addr,
2596   output [ 3:0] rvfi_mem_rmask,
2597   output [ 3:0] rvfi_mem_wmask,
2598   output [31:0] rvfi_mem_rdata,
2599   output [31:0] rvfi_mem_wdata,
2600 `endif
2601
2602   // Trace Interface
2603   output        trace_valid,
2604   output [35:0] trace_data
2605 );
2606   wire        mem_valid;
2607   wire [31:0] mem_addr;
2608   wire [31:0] mem_wdata;
2609   wire [ 3:0] mem_wstrb;
2610   wire        mem_instr;
2611   wire        mem_ready;
2612   wire [31:0] mem_rdata;
2613
2614   picorv32_axi_adapter axi_adapter (
2615     .clk            (clk            ),
2616     .resetn         (resetn         ),
2617     .mem_axi_awvalid(mem_axi_awvalid),
2618     .mem_axi_awready(mem_axi_awready),
2619     .mem_axi_awaddr (mem_axi_awaddr ),
2620     .mem_axi_awprot (mem_axi_awprot ),
2621     .mem_axi_wvalid (mem_axi_wvalid ),
2622     .mem_axi_wready (mem_axi_wready ),
2623     .mem_axi_wdata  (mem_axi_wdata  ),
2624     .mem_axi_wstrb  (mem_axi_wstrb  ),
2625     .mem_axi_bvalid (mem_axi_bvalid ),
2626     .mem_axi_bready (mem_axi_bready ),
2627     .mem_axi_arvalid(mem_axi_arvalid),
2628     .mem_axi_arready(mem_axi_arready),
2629     .mem_axi_araddr (mem_axi_araddr ),
2630     .mem_axi_arprot (mem_axi_arprot ),
2631     .mem_axi_rvalid (mem_axi_rvalid ),
2632     .mem_axi_rready (mem_axi_rready ),
2633     .mem_axi_rdata  (mem_axi_rdata  ),
2634     .mem_valid      (mem_valid      ),
2635     .mem_instr      (mem_instr      ),
2636     .mem_ready      (mem_ready      ),
2637     .mem_addr       (mem_addr       ),
2638     .mem_wdata      (mem_wdata      ),
2639     .mem_wstrb      (mem_wstrb      ),
```

```
2640      .mem_rdata      (mem_rdata      )
2641    );
2642
2643    picorv32 #(
2644      .ENABLE_COUNTERS     (ENABLE_COUNTERS     ),
2645      .ENABLE_COUNTERS64   (ENABLE_COUNTERS64   ),
2646      .ENABLE_REGS_16_31   (ENABLE_REGS_16_31   ),
2647      .ENABLE_REGS_DUALPORT(ENABLE_REGS_DUALPORT),
2648      .TWO_STAGE_SHIFT     (TWO_STAGE_SHIFT     ),
2649      .BARREL_SHIFTER      (BARREL_SHIFTER      ),
2650      .TWO_CYCLE_COMPARE   (TWO_CYCLE_COMPARE   ),
2651      .TWO_CYCLE_ALU       (TWO_CYCLE_ALU       ),
2652      .COMPRESSED_ISA      (COMPRESSED_ISA      ),
2653      .CATCH_MISALIGN      (CATCH_MISALIGN      ),
2654      .CATCH_ILLINSN       (CATCH_ILLINSN       ),
2655      .ENABLE_PCPI         (ENABLE_PCPI         ),
2656      .ENABLE_MUL          (ENABLE_MUL          ),
2657      .ENABLE_FAST_MUL     (ENABLE_FAST_MUL     ),
2658      .ENABLE_DIV          (ENABLE_DIV          ),
2659      .ENABLE_IRQ          (ENABLE_IRQ          ),
2660      .ENABLE_IRQ_QREGS    (ENABLE_IRQ_QREGS    ),
2661      .ENABLE_IRQ_TIMER    (ENABLE_IRQ_TIMER    ),
2662      .ENABLE_TRACE        (ENABLE_TRACE        ),
2663      .REGS_INIT_ZERO      (REGS_INIT_ZERO      ),
2664      .MASKED_IRQ          (MASKED_IRQ          ),
2665      .LATCHED_IRQ         (LATCHED_IRQ         ),
2666      .PROGADDR_RESET      (PROGADDR_RESET      ),
2667      .PROGADDR_IRQ        (PROGADDR_IRQ        ),
2668      .STACKADDR           (STACKADDR           )
2669    ) picorv32_core (
2670      .clk      (clk   ),
2671      .resetn   (resetn),
2672      .trap     (trap  ),
2673
2674      .mem_valid(mem_valid),
2675      .mem_addr (mem_addr ),
2676      .mem_wdata(mem_wdata),
2677      .mem_wstrb(mem_wstrb),
2678      .mem_instr(mem_instr),
2679      .mem_ready(mem_ready),
2680      .mem_rdata(mem_rdata),
2681
2682      .pcpi_valid(pcpi_valid),
2683      .pcpi_insn (pcpi_insn ),
2684      .pcpi_rs1  (pcpi_rs1  ),
2685      .pcpi_rs2  (pcpi_rs2  ),
2686      .pcpi_wr   (pcpi_wr   ),
2687      .pcpi_rd   (pcpi_rd   ),
2688      .pcpi_wait (pcpi_wait ),
2689      .pcpi_ready(pcpi_ready),
2690
2691      .irq(irq),
2692      .eoi(eoi),
```

```
2693
2694 `ifdef RISCV_FORMAL
2695     .rvfi_valid    (rvfi_valid    ),
2696     .rvfi_order    (rvfi_order    ),
2697     .rvfi_insn     (rvfi_insn     ),
2698     .rvfi_trap     (rvfi_trap     ),
2699     .rvfi_halt     (rvfi_halt     ),
2700     .rvfi_intr     (rvfi_intr     ),
2701     .rvfi_rs1_addr (rvfi_rs1_addr ),
2702     .rvfi_rs2_addr (rvfi_rs2_addr ),
2703     .rvfi_rs1_rdata(rvfi_rs1_rdata),
2704     .rvfi_rs2_rdata(rvfi_rs2_rdata),
2705     .rvfi_rd_addr  (rvfi_rd_addr  ),
2706     .rvfi_rd_wdata (rvfi_rd_wdata ),
2707     .rvfi_pc_rdata (rvfi_pc_rdata ),
2708     .rvfi_pc_wdata (rvfi_pc_wdata ),
2709     .rvfi_mem_addr (rvfi_mem_addr ),
2710     .rvfi_mem_rmask(rvfi_mem_rmask),
2711     .rvfi_mem_wmask(rvfi_mem_wmask),
2712     .rvfi_mem_rdata(rvfi_mem_rdata),
2713     .rvfi_mem_wdata(rvfi_mem_wdata),
2714 `endif
2715
2716     .trace_valid(trace_valid),
2717     .trace_data (trace_data)
2718   );
2719 endmodule
2720
2721
2722 /***************************************************************
2723  * picorv32_axi_adapter
2724  ***************************************************************/
2725
2726 module picorv32_axi_adapter (
2727   input clk, resetn,
2728
2729   // AXI4-lite master memory interface
2730
2731   output        mem_axi_awvalid,
2732   input         mem_axi_awready,
2733   output [31:0] mem_axi_awaddr,
2734   output [ 2:0] mem_axi_awprot,
2735
2736   output        mem_axi_wvalid,
2737   input         mem_axi_wready,
2738   output [31:0] mem_axi_wdata,
2739   output [ 3:0] mem_axi_wstrb,
2740
2741   input         mem_axi_bvalid,
2742   output        mem_axi_bready,
2743
2744   output        mem_axi_arvalid,
2745   input         mem_axi_arready,
```

```
2746   output [31:0] mem_axi_araddr,
2747   output [ 2:0] mem_axi_arprot,
2748
2749   input         mem_axi_rvalid,
2750   output        mem_axi_rready,
2751   input  [31:0] mem_axi_rdata,
2752
2753   // Native PicoRV32 memory interface
2754
2755   input         mem_valid,
2756   input         mem_instr,
2757   output        mem_ready,
2758   input  [31:0] mem_addr,
2759   input  [31:0] mem_wdata,
2760   input  [ 3:0] mem_wstrb,
2761   output [31:0] mem_rdata
2762 );
2763   reg ack_awvalid;
2764   reg ack_arvalid;
2765   reg ack_wvalid;
2766   reg xfer_done;
2767
2768   assign mem_axi_awvalid = mem_valid && |mem_wstrb && !ack_awvalid;
2769   assign mem_axi_awaddr = mem_addr;
2770   assign mem_axi_awprot = 0;
2771
2772   assign mem_axi_arvalid = mem_valid && !mem_wstrb && !ack_arvalid;
2773   assign mem_axi_araddr = mem_addr;
2774   assign mem_axi_arprot = mem_instr ? 3'b100 : 3'b000;
2775
2776   assign mem_axi_wvalid = mem_valid && |mem_wstrb && !ack_wvalid;
2777   assign mem_axi_wdata = mem_wdata;
2778   assign mem_axi_wstrb = mem_wstrb;
2779
2780   assign mem_ready = mem_axi_bvalid || mem_axi_rvalid;
2781   assign mem_axi_bready = mem_valid && |mem_wstrb;
2782   assign mem_axi_rready = mem_valid && !mem_wstrb;
2783   assign mem_rdata = mem_axi_rdata;
2784
2785   always @(posedge clk) begin
2786     if (!resetn) begin
2787       ack_awvalid <= 0;
2788     end else begin
2789       xfer_done <= mem_valid && mem_ready;
2790       if (mem_axi_awready && mem_axi_awvalid)
2791         ack_awvalid <= 1;
2792       if (mem_axi_arready && mem_axi_arvalid)
2793         ack_arvalid <= 1;
2794       if (mem_axi_wready && mem_axi_wvalid)
2795         ack_wvalid <= 1;
2796       if (xfer_done || !mem_valid) begin
2797         ack_awvalid <= 0;
2798         ack_arvalid <= 0;
```

```
2799          ack_wvalid <= 0;
2800        end
2801      end
2802    end
2803  endmodule
2804
2805
2806  /***************************************************************
2807   * picorv32_wb
2808   ***************************************************************/
2809
2810  module picorv32_wb #(
2811    parameter [ 0:0] ENABLE_COUNTERS = 1,
2812    parameter [ 0:0] ENABLE_COUNTERS64 = 1,
2813    parameter [ 0:0] ENABLE_REGS_16_31 = 1,
2814    parameter [ 0:0] ENABLE_REGS_DUALPORT = 1,
2815    parameter [ 0:0] TWO_STAGE_SHIFT = 1,
2816    parameter [ 0:0] BARREL_SHIFTER = 0,
2817    parameter [ 0:0] TWO_CYCLE_COMPARE = 0,
2818    parameter [ 0:0] TWO_CYCLE_ALU = 0,
2819    parameter [ 0:0] COMPRESSED_ISA = 0,
2820    parameter [ 0:0] CATCH_MISALIGN = 1,
2821    parameter [ 0:0] CATCH_ILLINSN = 1,
2822    parameter [ 0:0] ENABLE_PCPI = 0,
2823    parameter [ 0:0] ENABLE_MUL = 0,
2824    parameter [ 0:0] ENABLE_FAST_MUL = 0,
2825    parameter [ 0:0] ENABLE_DIV = 0,
2826    parameter [ 0:0] ENABLE_IRQ = 0,
2827    parameter [ 0:0] ENABLE_IRQ_QREGS = 1,
2828    parameter [ 0:0] ENABLE_IRQ_TIMER = 1,
2829    parameter [ 0:0] ENABLE_TRACE = 0,
2830    parameter [ 0:0] REGS_INIT_ZERO = 0,
2831    parameter [31:0] MASKED_IRQ = 32'h 0000_0000,
2832    parameter [31:0] LATCHED_IRQ = 32'h ffff_ffff,
2833    parameter [31:0] PROGADDR_RESET = 32'h 0000_0000,
2834    parameter [31:0] PROGADDR_IRQ = 32'h 0000_0010,
2835    parameter [31:0] STACKADDR = 32'h ffff_ffff
2836  ) (
2837    output trap,
2838
2839    // Wishbone interfaces
2840    input wb_rst_i,
2841    input wb_clk_i,
2842
2843    output reg [31:0] wbm_adr_o,
2844    output reg [31:0] wbm_dat_o,
2845    input [31:0] wbm_dat_i,
2846    output reg wbm_we_o,
2847    output reg [3:0] wbm_sel_o,
2848    output reg wbm_stb_o,
2849    input wbm_ack_i,
2850    output reg wbm_cyc_o,
2851
```

```
2852   // Pico Co-Processor Interface (PCPI)
2853   output        pcpi_valid,
2854   output [31:0] pcpi_insn,
2855   output [31:0] pcpi_rs1,
2856   output [31:0] pcpi_rs2,
2857   input         pcpi_wr,
2858   input  [31:0] pcpi_rd,
2859   input         pcpi_wait,
2860   input         pcpi_ready,
2861
2862   // IRQ interface
2863   input  [31:0] irq,
2864   output [31:0] eoi,
2865
2866 `ifdef RISCV_FORMAL
2867   output        rvfi_valid,
2868   output [63:0] rvfi_order,
2869   output [31:0] rvfi_insn,
2870   output        rvfi_trap,
2871   output        rvfi_halt,
2872   output        rvfi_intr,
2873   output [ 4:0] rvfi_rs1_addr,
2874   output [ 4:0] rvfi_rs2_addr,
2875   output [31:0] rvfi_rs1_rdata,
2876   output [31:0] rvfi_rs2_rdata,
2877   output [ 4:0] rvfi_rd_addr,
2878   output [31:0] rvfi_rd_wdata,
2879   output [31:0] rvfi_pc_rdata,
2880   output [31:0] rvfi_pc_wdata,
2881   output [31:0] rvfi_mem_addr,
2882   output [ 3:0] rvfi_mem_rmask,
2883   output [ 3:0] rvfi_mem_wmask,
2884   output [31:0] rvfi_mem_rdata,
2885   output [31:0] rvfi_mem_wdata,
2886 `endif
2887
2888   // Trace Interface
2889   output        trace_valid,
2890   output [35:0] trace_data,
2891
2892   output mem_instr
2893 );
2894   wire        mem_valid;
2895   wire [31:0] mem_addr;
2896   wire [31:0] mem_wdata;
2897   wire [ 3:0] mem_wstrb;
2898   reg         mem_ready;
2899   reg [31:0] mem_rdata;
2900
2901   wire clk;
2902   wire resetn;
2903
2904   assign clk = wb_clk_i;
```

```
2905   assign resetn = ~wb_rst_i;
2906
2907   picorv32 #(
2908     .ENABLE_COUNTERS     (ENABLE_COUNTERS     ),
2909     .ENABLE_COUNTERS64   (ENABLE_COUNTERS64   ),
2910     .ENABLE_REGS_16_31   (ENABLE_REGS_16_31   ),
2911     .ENABLE_REGS_DUALPORT(ENABLE_REGS_DUALPORT),
2912     .TWO_STAGE_SHIFT     (TWO_STAGE_SHIFT     ),
2913     .BARREL_SHIFTER      (BARREL_SHIFTER      ),
2914     .TWO_CYCLE_COMPARE   (TWO_CYCLE_COMPARE   ),
2915     .TWO_CYCLE_ALU       (TWO_CYCLE_ALU       ),
2916     .COMPRESSED_ISA      (COMPRESSED_ISA      ),
2917     .CATCH_MISALIGN      (CATCH_MISALIGN      ),
2918     .CATCH_ILLINSN       (CATCH_ILLINSN       ),
2919     .ENABLE_PCPI         (ENABLE_PCPI         ),
2920     .ENABLE_MUL          (ENABLE_MUL          ),
2921     .ENABLE_FAST_MUL     (ENABLE_FAST_MUL     ),
2922     .ENABLE_DIV          (ENABLE_DIV          ),
2923     .ENABLE_IRQ          (ENABLE_IRQ          ),
2924     .ENABLE_IRQ_QREGS    (ENABLE_IRQ_QREGS    ),
2925     .ENABLE_IRQ_TIMER    (ENABLE_IRQ_TIMER    ),
2926     .ENABLE_TRACE        (ENABLE_TRACE        ),
2927     .REGS_INIT_ZERO      (REGS_INIT_ZERO      ),
2928     .MASKED_IRQ          (MASKED_IRQ          ),
2929     .LATCHED_IRQ         (LATCHED_IRQ         ),
2930     .PROGADDR_RESET      (PROGADDR_RESET      ),
2931     .PROGADDR_IRQ        (PROGADDR_IRQ        ),
2932     .STACKADDR           (STACKADDR           )
2933   ) picorv32_core (
2934     .clk      (clk   ),
2935     .resetn   (resetn),
2936     .trap     (trap  ),
2937
2938     .mem_valid(mem_valid),
2939     .mem_addr (mem_addr ),
2940     .mem_wdata(mem_wdata),
2941     .mem_wstrb(mem_wstrb),
2942     .mem_instr(mem_instr),
2943     .mem_ready(mem_ready),
2944     .mem_rdata(mem_rdata),
2945
2946     .pcpi_valid(pcpi_valid),
2947     .pcpi_insn (pcpi_insn ),
2948     .pcpi_rs1  (pcpi_rs1  ),
2949     .pcpi_rs2  (pcpi_rs2  ),
2950     .pcpi_wr   (pcpi_wr   ),
2951     .pcpi_rd   (pcpi_rd   ),
2952     .pcpi_wait (pcpi_wait ),
2953     .pcpi_ready(pcpi_ready),
2954
2955     .irq(irq),
2956     .eoi(eoi),
2957
```

```
2958 `ifdef RISCV_FORMAL
2959     .rvfi_valid    (rvfi_valid    ),
2960     .rvfi_order    (rvfi_order    ),
2961     .rvfi_insn     (rvfi_insn     ),
2962     .rvfi_trap     (rvfi_trap     ),
2963     .rvfi_halt     (rvfi_halt     ),
2964     .rvfi_intr     (rvfi_intr     ),
2965     .rvfi_rs1_addr (rvfi_rs1_addr ),
2966     .rvfi_rs2_addr (rvfi_rs2_addr ),
2967     .rvfi_rs1_rdata(rvfi_rs1_rdata),
2968     .rvfi_rs2_rdata(rvfi_rs2_rdata),
2969     .rvfi_rd_addr  (rvfi_rd_addr  ),
2970     .rvfi_rd_wdata (rvfi_rd_wdata ),
2971     .rvfi_pc_rdata (rvfi_pc_rdata ),
2972     .rvfi_pc_wdata (rvfi_pc_wdata ),
2973     .rvfi_mem_addr (rvfi_mem_addr ),
2974     .rvfi_mem_rmask(rvfi_mem_rmask),
2975     .rvfi_mem_wmask(rvfi_mem_wmask),
2976     .rvfi_mem_rdata(rvfi_mem_rdata),
2977     .rvfi_mem_wdata(rvfi_mem_wdata),
2978 `endif
2979 
2980     .trace_valid(trace_valid),
2981     .trace_data (trace_data)
2982   );
2983 
2984   localparam IDLE = 2'b00;
2985   localparam WBSTART = 2'b01;
2986   localparam WBEND = 2'b10;
2987 
2988   reg [1:0] state;
2989 
2990   wire we;
2991   assign we = (mem_wstrb[0] | mem_wstrb[1] | mem_wstrb[2] |
       mem_wstrb[3]);
2992 
2993   always @(posedge wb_clk_i) begin
2994     if (wb_rst_i) begin
2995       wbm_adr_o <= 0;
2996       wbm_dat_o <= 0;
2997       wbm_we_o <= 0;
2998       wbm_sel_o <= 0;
2999       wbm_stb_o <= 0;
3000       wbm_cyc_o <= 0;
3001       state <= IDLE;
3002     end else begin
3003       case (state)
3004         IDLE: begin
3005           if (mem_valid) begin
3006             wbm_adr_o <= mem_addr;
3007             wbm_dat_o <= mem_wdata;
3008             wbm_we_o <= we;
3009             wbm_sel_o <= mem_wstrb;
```

```
3010
3011              wbm_stb_o <= 1'b1;
3012              wbm_cyc_o <= 1'b1;
3013              state <= WBSTART;
3014            end else begin
3015              mem_ready <= 1'b0;
3016
3017              wbm_stb_o <= 1'b0;
3018              wbm_cyc_o <= 1'b0;
3019              wbm_we_o <= 1'b0;
3020            end
3021          end
3022          WBSTART:begin
3023            if (wbm_ack_i) begin
3024              mem_rdata <= wbm_dat_i;
3025              mem_ready <= 1'b1;
3026
3027              state <= WBEND;
3028
3029              wbm_stb_o <= 1'b0;
3030              wbm_cyc_o <= 1'b0;
3031              wbm_we_o <= 1'b0;
3032            end
3033          end
3034          WBEND: begin
3035            mem_ready <= 1'b0;
3036
3037            state <= IDLE;
3038          end
3039          default:
3040            state <= IDLE;
3041        endcase
3042      end
3043    end
3044  endmodule
```

Anexo D – *spimemio.v*

```
/*
 *  PicoSoC - A simple example SoC using PicoRV32
 *
 *  Copyright (C) 2017  Claire Xenia Wolf <claire@yosyshq.com>
 *
 *  Permission to use, copy, modify, and/or distribute this software for any
 *  purpose with or without fee is hereby granted, provided that the above
 *  copyright notice and this permission notice appear in all copies.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 *  WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 *  MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 *  ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 *  WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 *  ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 *  OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */

module spimemio (
  input clk, resetn,

  input valid,
  output ready,
  input [23:0] addr,
  output reg [31:0] rdata,

  output flash_csb,
  output flash_clk,

  output flash_io0_oe,
  output flash_io1_oe,
  output flash_io2_oe,
  output flash_io3_oe,

  output flash_io0_do,
  output flash_io1_do,
  output flash_io2_do,
  output flash_io3_do,

  input  flash_io0_di,
  input  flash_io1_di,
```

```
43    input  flash_io2_di,
44    input  flash_io3_di,
45
46    input   [3:0] cfgreg_we,
47    input  [31:0] cfgreg_di,
48    output [31:0] cfgreg_do
49  );
50    reg        xfer_resetn;
51    reg        din_valid;
52    wire       din_ready;
53    reg  [7:0] din_data;
54    reg  [3:0] din_tag;
55    reg        din_cont;
56    reg        din_qspi;
57    reg        din_ddr;
58    reg        din_rd;
59
60    wire       dout_valid;
61    wire [7:0] dout_data;
62    wire [3:0] dout_tag;
63
64    reg [23:0] buffer;
65
66    reg [23:0] rd_addr;
67    reg rd_valid;
68    reg rd_wait;
69    reg rd_inc;
70
71    assign ready = valid && (addr == rd_addr) && rd_valid;
72    wire jump = valid && !ready && (addr != rd_addr+4) && rd_valid;
73
74    reg softreset;
75
76    reg       config_en;      // cfgreg[31]
77    reg       config_ddr;     // cfgreg[22]
78    reg       config_qspi;    // cfgreg[21]
79    reg       config_cont;    // cfgreg[20]
80    reg [3:0] config_dummy;   // cfgreg[19:16]
81    reg [3:0] config_oe;      // cfgreg[11:8]
82    reg       config_csb;     // cfgreg[5]
83    reg       config_clk;     // cfgref[4]
84    reg [3:0] config_do;      // cfgreg[3:0]
85
86    assign cfgreg_do[31] = config_en;
87    assign cfgreg_do[30:23] = 0;
88    assign cfgreg_do[22] = config_ddr;
89    assign cfgreg_do[21] = config_qspi;
90    assign cfgreg_do[20] = config_cont;
91    assign cfgreg_do[19:16] = config_dummy;
92    assign cfgreg_do[15:12] = 0;
93    assign cfgreg_do[11:8] = {flash_io3_oe, flash_io2_oe, flash_io1_oe,
        flash_io0_oe};
94    assign cfgreg_do[7:6] = 0;
```

```
95    assign cfgreg_do[5] = flash_csb;
96    assign cfgreg_do[4] = flash_clk;
97    assign cfgreg_do[3:0] = {flash_io3_di, flash_io2_di, flash_io1_di,
          flash_io0_di};
98
99    always @(posedge clk) begin
100     softreset <= !config_en || cfgreg_we;
101     if (!resetn) begin
102       softreset <= 1;
103       config_en <= 1;
104       config_csb <= 0;
105       config_clk <= 0;
106       config_oe <= 0;
107       config_do <= 0;
108       config_ddr <= 0;
109       config_qspi <= 0;
110       config_cont <= 0;
111       config_dummy <= 8;
112     end else begin
113       if (cfgreg_we[0]) begin
114         config_csb <= cfgreg_di[5];
115         config_clk <= cfgreg_di[4];
116         config_do <= cfgreg_di[3:0];
117       end
118       if (cfgreg_we[1]) begin
119         config_oe <= cfgreg_di[11:8];
120       end
121       if (cfgreg_we[2]) begin
122         config_ddr <= cfgreg_di[22];
123         config_qspi <= cfgreg_di[21];
124         config_cont <= cfgreg_di[20];
125         config_dummy <= cfgreg_di[19:16];
126       end
127       if (cfgreg_we[3]) begin
128         config_en <= cfgreg_di[31];
129       end
130     end
131   end
132
133   wire xfer_csb;
134   wire xfer_clk;
135
136   wire xfer_io0_oe;
137   wire xfer_io1_oe;
138   wire xfer_io2_oe;
139   wire xfer_io3_oe;
140
141   wire xfer_io0_do;
142   wire xfer_io1_do;
143   wire xfer_io2_do;
144   wire xfer_io3_do;
145
146   reg xfer_io0_90;
```

```
147   reg xfer_io1_90;
148   reg xfer_io2_90;
149   reg xfer_io3_90;
150
151   always @(negedge clk) begin
152     xfer_io0_90 <= xfer_io0_do;
153     xfer_io1_90 <= xfer_io1_do;
154     xfer_io2_90 <= xfer_io2_do;
155     xfer_io3_90 <= xfer_io3_do;
156   end
157
158   assign flash_csb = config_en ? xfer_csb : config_csb;
159   assign flash_clk = config_en ? xfer_clk : config_clk;
160
161   assign flash_io0_oe = config_en ? xfer_io0_oe : config_oe[0];
162   assign flash_io1_oe = config_en ? xfer_io1_oe : config_oe[1];
163   assign flash_io2_oe = config_en ? xfer_io2_oe : config_oe[2];
164   assign flash_io3_oe = config_en ? xfer_io3_oe : config_oe[3];
165
166   assign flash_io0_do = config_en ? (config_ddr ? xfer_io0_90 :
        xfer_io0_do) : config_do[0];
167   assign flash_io1_do = config_en ? (config_ddr ? xfer_io1_90 :
        xfer_io1_do) : config_do[1];
168   assign flash_io2_do = config_en ? (config_ddr ? xfer_io2_90 :
        xfer_io2_do) : config_do[2];
169   assign flash_io3_do = config_en ? (config_ddr ? xfer_io3_90 :
        xfer_io3_do) : config_do[3];
170
171   wire xfer_dspi = din_ddr && !din_qspi;
172   wire xfer_ddr = din_ddr && din_qspi;
173
174   spimemio_xfer xfer (
175     .clk          (clk         ),
176     .resetn       (xfer_resetn ),
177     .din_valid    (din_valid   ),
178     .din_ready    (din_ready   ),
179     .din_data     (din_data    ),
180     .din_tag      (din_tag     ),
181     .din_cont     (din_cont    ),
182     .din_dspi     (xfer_dspi   ),
183     .din_qspi     (din_qspi    ),
184     .din_ddr      (xfer_ddr    ),
185     .din_rd       (din_rd      ),
186     .dout_valid   (dout_valid  ),
187     .dout_data    (dout_data   ),
188     .dout_tag     (dout_tag    ),
189     .flash_csb    (xfer_csb    ),
190     .flash_clk    (xfer_clk    ),
191     .flash_io0_oe (xfer_io0_oe ),
192     .flash_io1_oe (xfer_io1_oe ),
193     .flash_io2_oe (xfer_io2_oe ),
194     .flash_io3_oe (xfer_io3_oe ),
195     .flash_io0_do (xfer_io0_do ),
```

```
196     .flash_io1_do (xfer_io1_do ),
197     .flash_io2_do (xfer_io2_do ),
198     .flash_io3_do (xfer_io3_do ),
199     .flash_io0_di (flash_io0_di),
200     .flash_io1_di (flash_io1_di),
201     .flash_io2_di (flash_io2_di),
202     .flash_io3_di (flash_io3_di)
203   );
204
205   reg [3:0] state;
206
207   always @(posedge clk) begin
208     xfer_resetn <= 1;
209     din_valid <= 0;
210
211     if (!resetn || softreset) begin
212       state <= 0;
213       xfer_resetn <= 0;
214       rd_valid <= 0;
215       din_tag <= 0;
216       din_cont <= 0;
217       din_qspi <= 0;
218       din_ddr <= 0;
219       din_rd <= 0;
220     end else begin
221       if (dout_valid && dout_tag == 1) buffer[ 7: 0] <= dout_data;
222       if (dout_valid && dout_tag == 2) buffer[15: 8] <= dout_data;
223       if (dout_valid && dout_tag == 3) buffer[23:16] <= dout_data;
224       if (dout_valid && dout_tag == 4) begin
225         rdata <= {dout_data, buffer};
226         rd_addr <= rd_inc ? rd_addr + 4 : addr;
227         rd_valid <= 1;
228         rd_wait <= rd_inc;
229         rd_inc <= 1;
230       end
231
232       if (valid)
233         rd_wait <= 0;
234
235       case (state)
236         0: begin
237           din_valid <= 1;
238           din_data <= 8'h ff;
239           din_tag <= 0;
240           if (din_ready) begin
241             din_valid <= 0;
242             state <= 1;
243           end
244         end
245         1: begin
246           if (dout_valid) begin
247             xfer_resetn <= 0;
248             state <= 2;
```

```
          end
        end
        2: begin
          din_valid <= 1;
          din_data <= 8'h ab;
          din_tag <= 0;
          if (din_ready) begin
            din_valid <= 0;
            state <= 3;
          end
        end
        3: begin
          if (dout_valid) begin
            xfer_resetn <= 0;
            state <= 4;
          end
        end
        4: begin
          rd_inc <= 0;
          din_valid <= 1;
          din_tag <= 0;
          case ({config_ddr, config_qspi})
            2'b11: din_data <= 8'h ED;
            2'b01: din_data <= 8'h EB;
            2'b10: din_data <= 8'h BB;
            2'b00: din_data <= 8'h 03;
          endcase
          if (din_ready) begin
            din_valid <= 0;
            state <= 5;
          end
        end
        5: begin
          if (valid && !ready) begin
            din_valid <= 1;
            din_tag <= 0;
            din_data <= addr[23:16];
            din_qspi <= config_qspi;
            din_ddr <= config_ddr;
            if (din_ready) begin
              din_valid <= 0;
              state <= 6;
            end
          end
        end
        6: begin
          din_valid <= 1;
          din_tag <= 0;
          din_data <= addr[15:8];
          if (din_ready) begin
            din_valid <= 0;
            state <= 7;
          end
```

```
302           end
303           7: begin
304             din_valid <= 1;
305             din_tag <= 0;
306             din_data <= addr[7:0];
307             if (din_ready) begin
308               din_valid <= 0;
309               din_data <= 0;
310               state <= config_qspi || config_ddr ? 8 : 9;
311             end
312           end
313           8: begin
314             din_valid <= 1;
315             din_tag <= 0;
316             din_data <= config_cont ? 8'h A5 : 8'h FF;
317             if (din_ready) begin
318               din_rd <= 1;
319               din_data <= config_dummy;
320               din_valid <= 0;
321               state <= 9;
322             end
323           end
324           9: begin
325             din_valid <= 1;
326             din_tag <= 1;
327             if (din_ready) begin
328               din_valid <= 0;
329               state <= 10;
330             end
331           end
332           10: begin
333             din_valid <= 1;
334             din_data <= 8'h 00;
335             din_tag <= 2;
336             if (din_ready) begin
337               din_valid <= 0;
338               state <= 11;
339             end
340           end
341           11: begin
342             din_valid <= 1;
343             din_tag <= 3;
344             if (din_ready) begin
345               din_valid <= 0;
346               state <= 12;
347             end
348           end
349           12: begin
350             if (!rd_wait || valid) begin
351               din_valid <= 1;
352               din_tag <= 4;
353               if (din_ready) begin
354                 din_valid <= 0;
```

```
355                 state <= 9;
356               end
357             end
358           end
359         endcase
360
361         if (jump) begin
362           rd_inc <= 0;
363           rd_valid <= 0;
364           xfer_resetn <= 0;
365           if (config_cont) begin
366             state <= 5;
367           end else begin
368             state <= 4;
369             din_qspi <= 0;
370             din_ddr <= 0;
371           end
372           din_rd <= 0;
373         end
374       end
375     end
376   endmodule
377
378   module spimemio_xfer (
379     input clk, resetn,
380
381     input             din_valid,
382     output            din_ready,
383     input       [7:0] din_data,
384     input       [3:0] din_tag,
385     input             din_cont,
386     input             din_dspi,
387     input             din_qspi,
388     input             din_ddr,
389     input             din_rd,
390
391     output            dout_valid,
392     output      [7:0] dout_data,
393     output      [3:0] dout_tag,
394
395     output reg flash_csb,
396     output reg flash_clk,
397
398     output reg flash_io0_oe,
399     output reg flash_io1_oe,
400     output reg flash_io2_oe,
401     output reg flash_io3_oe,
402
403     output reg flash_io0_do,
404     output reg flash_io1_do,
405     output reg flash_io2_do,
406     output reg flash_io3_do,
407
```

```
408   input       flash_io0_di,
409   input       flash_io1_di,
410   input       flash_io2_di,
411   input       flash_io3_di
412 );
413   reg [7:0] obuffer;
414   reg [7:0] ibuffer;
415
416   reg [3:0] count;
417   reg [3:0] dummy_count;
418
419   reg xfer_cont;
420   reg xfer_dspi;
421   reg xfer_qspi;
422   reg xfer_ddr;
423   reg xfer_ddr_q;
424   reg xfer_rd;
425   reg [3:0] xfer_tag;
426   reg [3:0] xfer_tag_q;
427
428   reg [7:0] next_obuffer;
429   reg [7:0] next_ibuffer;
430   reg [3:0] next_count;
431
432   reg fetch;
433   reg next_fetch;
434   reg last_fetch;
435
436   always @(posedge clk) begin
437     xfer_ddr_q <= xfer_ddr;
438     xfer_tag_q <= xfer_tag;
439   end
440
441   assign din_ready = din_valid && resetn && next_fetch;
442
443   assign dout_valid = (xfer_ddr_q ? fetch && !last_fetch : next_fetch &&
        !fetch) && resetn;
444   assign dout_data = next_ibuffer;
445   assign dout_tag = xfer_tag_q;
446
447   always @* begin
448     flash_io0_oe = 0;
449     flash_io1_oe = 0;
450     flash_io2_oe = 0;
451     flash_io3_oe = 0;
452
453     flash_io0_do = 0;
454     flash_io1_do = 0;
455     flash_io2_do = 0;
456     flash_io3_do = 0;
457
458     next_obuffer = obuffer;
459     next_ibuffer = ibuffer;
```

```
460     next_count = count;
461     next_fetch = 0;
462
463     if (dummy_count == 0) begin
464       casez ({xfer_ddr, xfer_qspi, xfer_dspi})
465         3'b 000: begin
466           flash_io0_oe = 1;
467           flash_io0_do = obuffer[7];
468
469           if (flash_clk) begin
470             next_obuffer = {obuffer[6:0], 1'b 0};
471             next_count = count - |count;
472             next_ibuffer = {ibuffer[6:0], flash_io1_di};
473           end
474
475           next_fetch = (next_count == 0);
476         end
477         3'b 01?: begin
478           flash_io0_oe = !xfer_rd;
479           flash_io1_oe = !xfer_rd;
480           flash_io2_oe = !xfer_rd;
481           flash_io3_oe = !xfer_rd;
482
483           flash_io0_do = obuffer[4];
484           flash_io1_do = obuffer[5];
485           flash_io2_do = obuffer[6];
486           flash_io3_do = obuffer[7];
487
488           if (flash_clk) begin
489             next_obuffer = {obuffer[3:0], 4'b 0000};
490             next_count = count - {|count, 2'b00};
491           end else begin
492             next_ibuffer = {ibuffer[3:0], flash_io3_di, flash_io2_di,
                  flash_io1_di, flash_io0_di};
493           end
494
495           next_fetch = (next_count == 0);
496         end
497         3'b 11?: begin
498           flash_io0_oe = !xfer_rd;
499           flash_io1_oe = !xfer_rd;
500           flash_io2_oe = !xfer_rd;
501           flash_io3_oe = !xfer_rd;
502
503           flash_io0_do = obuffer[4];
504           flash_io1_do = obuffer[5];
505           flash_io2_do = obuffer[6];
506           flash_io3_do = obuffer[7];
507
508           next_obuffer = {obuffer[3:0], 4'b 0000};
509           next_ibuffer = {ibuffer[3:0], flash_io3_di, flash_io2_di,
              flash_io1_di, flash_io0_di};
510           next_count = count - {|count, 2'b00};
```

```
511
512           next_fetch = (next_count == 0);
513         end
514         3'b ??1: begin
515           flash_io0_oe = !xfer_rd;
516           flash_io1_oe = !xfer_rd;
517
518           flash_io0_do = obuffer[6];
519           flash_io1_do = obuffer[7];
520
521           if (flash_clk) begin
522             next_obuffer = {obuffer[5:0], 2'b 00};
523             next_count = count - {|count, 1'b0};
524           end else begin
525             next_ibuffer = {ibuffer[5:0], flash_io1_di, flash_io0_di};
526           end
527
528           next_fetch = (next_count == 0);
529         end
530       endcase
531     end
532   end
533
534   always @(posedge clk) begin
535     if (!resetn) begin
536       fetch <= 1;
537       last_fetch <= 1;
538       flash_csb <= 1;
539       flash_clk <= 0;
540       count <= 0;
541       dummy_count <= 0;
542       xfer_tag <= 0;
543       xfer_cont <= 0;
544       xfer_dspi <= 0;
545       xfer_qspi <= 0;
546       xfer_ddr <= 0;
547       xfer_rd <= 0;
548     end else begin
549       fetch <= next_fetch;
550       last_fetch <= xfer_ddr ? fetch : 1;
551       if (dummy_count) begin
552         flash_clk <= !flash_clk && !flash_csb;
553         dummy_count <= dummy_count - flash_clk;
554       end else
555       if (count) begin
556         flash_clk <= !flash_clk && !flash_csb;
557         obuffer <= next_obuffer;
558         ibuffer <= next_ibuffer;
559         count <= next_count;
560       end
561       if (din_valid && din_ready) begin
562         flash_csb <= 0;
563         flash_clk <= 0;
```

```
564
565         count <= 8;
566         dummy_count <= din_rd ? din_data : 0;
567         obuffer <= din_data;
568
569         xfer_tag <= din_tag;
570         xfer_cont <= din_cont;
571         xfer_dspi <= din_dspi;
572         xfer_qspi <= din_qspi;
573         xfer_ddr <= din_ddr;
574         xfer_rd <= din_rd;
575       end
576     end
577   end
578 endmodule
```

Anexo E – *simpleuart.v*

```
/*
 *  PicoSoC - A simple example SoC using PicoRV32
 *
 *  Copyright (C) 2017  Claire Xenia Wolf <claire@yosyshq.com>
 *
 *  Permission to use, copy, modify, and/or distribute this software for any
 *  purpose with or without fee is hereby granted, provided that the above
 *  copyright notice and this permission notice appear in all copies.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 *  WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 *  MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 *  ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 *  WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 *  ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 *  OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */

module simpleuart #(parameter integer DEFAULT_DIV = 1) (
  input clk,
  input resetn,

  output ser_tx,
  input  ser_rx,

  input   [3:0] reg_div_we,
  input  [31:0] reg_div_di,
  output [31:0] reg_div_do,

  input         reg_dat_we,
  input         reg_dat_re,
  input  [31:0] reg_dat_di,
  output [31:0] reg_dat_do,
  output        reg_dat_wait
);
  reg [31:0] cfg_divider;

  reg [3:0] recv_state;
  reg [31:0] recv_divcnt;
  reg [7:0] recv_pattern;
  reg [7:0] recv_buf_data;
```

```
43    reg recv_buf_valid;
44
45    reg [9:0] send_pattern;
46    reg [3:0] send_bitcnt;
47    reg [31:0] send_divcnt;
48    reg send_dummy;
49
50    assign reg_div_do = cfg_divider;
51
52    assign reg_dat_wait = reg_dat_we && (send_bitcnt || send_dummy);
53    assign reg_dat_do = recv_buf_valid ? recv_buf_data : ~0;
54
55    always @(posedge clk) begin
56      if (!resetn) begin
57        cfg_divider <= DEFAULT_DIV;
58      end else begin
59        if (reg_div_we[0]) cfg_divider[ 7: 0] <= reg_div_di[ 7: 0];
60        if (reg_div_we[1]) cfg_divider[15: 8] <= reg_div_di[15: 8];
61        if (reg_div_we[2]) cfg_divider[23:16] <= reg_div_di[23:16];
62        if (reg_div_we[3]) cfg_divider[31:24] <= reg_div_di[31:24];
63      end
64    end
65
66    always @(posedge clk) begin
67      if (!resetn) begin
68        recv_state <= 0;
69        recv_divcnt <= 0;
70        recv_pattern <= 0;
71        recv_buf_data <= 0;
72        recv_buf_valid <= 0;
73      end else begin
74        recv_divcnt <= recv_divcnt + 1;
75        if (reg_dat_re)
76          recv_buf_valid <= 0;
77        case (recv_state)
78          0: begin
79            if (!ser_rx)
80              recv_state <= 1;
81            recv_divcnt <= 0;
82          end
83          1: begin
84            if (2*recv_divcnt > cfg_divider) begin
85              recv_state <= 2;
86              recv_divcnt <= 0;
87            end
88          end
89          10: begin
90            if (recv_divcnt > cfg_divider) begin
91              recv_buf_data <= recv_pattern;
92              recv_buf_valid <= 1;
93              recv_state <= 0;
94            end
95          end
```

```
96          default: begin
97            if (recv_divcnt > cfg_divider) begin
98              recv_pattern <= {ser_rx, recv_pattern[7:1]};
99              recv_state <= recv_state + 1;
100             recv_divcnt <= 0;
101           end
102         end
103       endcase
104     end
105   end
106
107   assign ser_tx = send_pattern[0];
108
109   always @(posedge clk) begin
110     if (reg_div_we)
111       send_dummy <= 1;
112     send_divcnt <= send_divcnt + 1;
113     if (!resetn) begin
114       send_pattern <= ~0;
115       send_bitcnt <= 0;
116       send_divcnt <= 0;
117       send_dummy <= 1;
118     end else begin
119       if (send_dummy && !send_bitcnt) begin
120         send_pattern <= ~0;
121         send_bitcnt <= 15;
122         send_divcnt <= 0;
123         send_dummy <= 0;
124       end else
125       if (reg_dat_we && !send_bitcnt) begin
126         send_pattern <= {1'b1, reg_dat_di[7:0], 1'b0};
127         send_bitcnt <= 10;
128         send_divcnt <= 0;
129       end else
130       if (send_divcnt > cfg_divider && send_bitcnt) begin
131         send_pattern <= {1'b1, send_pattern[9:1]};
132         send_bitcnt <= send_bitcnt - 1;
133         send_divcnt <= 0;
134       end
135     end
136   end
137 endmodule
```

Anexo F – *icebreaker_tb.v*

```
/*
 *  PicoSoC - A simple example SoC using PicoRV32
 *
 *  Copyright (C) 2017  Claire Xenia Wolf <claire@yosyshq.com>
 *
 *  Permission to use, copy, modify, and/or distribute this software for any
 *  purpose with or without fee is hereby granted, provided that the above
 *  copyright notice and this permission notice appear in all copies.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 *  WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 *  MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 *  ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 *  WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 *  ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 *  OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */

`timescale 1 ns / 1 ps

module testbench;
  reg clk;
  always #5 clk = (clk === 1'b0);

  localparam ser_half_period = 53;
  event ser_sample;

  initial begin
    $dumpfile("testbench.vcd");
    $dumpvars(0, testbench);

    repeat (6) begin
      repeat (50000) @(posedge clk);
      $display("+50000 cycles");
    end
    $finish;
  end

  integer cycle_cnt = 0;

  always @(posedge clk) begin
```

```
43      cycle_cnt <= cycle_cnt + 1;
44    end
45
46    wire led1, led2, led3, led4, led5;
47    wire ledr_n, ledg_n;
48
49    wire [6:0] leds = {!ledg_n, !ledr_n, led5, led4, led3, led2, led1};
50
51    wire ser_rx;
52    wire ser_tx;
53
54    wire flash_csb;
55    wire flash_clk;
56    wire flash_io0;
57    wire flash_io1;
58    wire flash_io2;
59    wire flash_io3;
60    reg reset;
61
62    always @(leds) begin
63      #1 $display("%b", leds);
64    end
65
66    nexys4_tb #(
67      // We limit the amount of memory in simulation
68      // in order to avoid reduce simulation time
69      // required for intialization of RAM
70      .MEM_WORDS(256)
71    ) uut (
72      .clk      (clk      ),
73      .led1     (led1     ),
74      .led2     (led2     ),
75      .led3     (led3     ),
76      .led4     (led4     ),
77      .led5     (led5     ),
78      .ledr_n   (ledr_n   ),
79      .ledg_n   (ledg_n   ),
80      .ser_rx   (ser_rx   ),
81      .ser_tx   (ser_tx   ),
82      .flash_csb(flash_csb),
83      .flash_clk(flash_clk),
84      .flash_io0(flash_io0),
85      .flash_io1(flash_io1),
86      .flash_io2(flash_io2),
87      .flash_io3(flash_io3),
88      .reset    (reset)
89    );
90
91    spiflash spiflash (
92      .csb(flash_csb),
93      .clk(flash_clk),
94      .io0(flash_io0),
95      .io1(flash_io1),
```

```
96      .io2(flash_io2),
97      .io3(flash_io3)
98    );
99
100   reg [7:0] buffer;
101
102     initial begin
103         reset <= 1;
104         #10;
105         reset <= 0;
106     end
107
108   always begin
109     @(negedge ser_tx);
110
111     repeat (ser_half_period) @(posedge clk);
112     -> ser_sample; // start bit
113
114     repeat (8) begin
115       repeat (ser_half_period) @(posedge clk);
116       repeat (ser_half_period) @(posedge clk);
117       buffer = {ser_tx, buffer[7:1]};
118       -> ser_sample; // data bit
119     end
120
121     repeat (ser_half_period) @(posedge clk);
122     repeat (ser_half_period) @(posedge clk);
123     -> ser_sample; // stop bit
124
125     if (buffer < 32 || buffer >= 127)
126       $display("Serial data: %d", buffer);
127     else
128       $display("Serial data: '%c'", buffer);
129   end
130 endmodule
```

Anexo G – *testbench_mlp.v*

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use work.fpupack.all;
4  use work.rnapack.all;
5
6  entity tb_MLP443 is
7  --  Port ( );
8  end tb_MLP443;
9
10 architecture Behavioral of tb_MLP443 is
11 component MLP443 is
12 Port ( clk : in STD_LOGIC;
13 reset : in STD_LOGIC;
14 start : in STD_LOGIC;
15 x_in : in array1D_in;
16 y_out : out array1D_out;
17 ready : out STD_LOGIC);
18 end component;
19
20 signal reset,clk,start,ready : std_logic := '0';
21 signal x : array1D_in := ("01000000101000000000000000",
      "01000000011001100110011001", "00111111101100110011001",
      "00111110010011001100110");
22 signal saida: array1D_out := (others=>(others=>'0'));
23
24 begin
25
26 uut: MLP443 port map(
27 reset   => reset,
28 clk     => clk,
29 x_in    => x,
30 start   => start,
31 y_out   => saida,
32 ready   => ready);
33
34 reset <= '0', '1' after 15 ns, '0' after 45 ns;
35 clk <= not clk after 5 ns;
36 start <= '0', '1' after 65 ns, '0' after 75 ns, '1' after 285 ns, '0'
      after 295 ns, '1' after 515 ns, '0' after 525 ns;
37 x <= ("01000000101100110011001", "01000000001000000000000000",
      "01000000111100110011001100", "00111111100011001100110");
38
39
40 end Behavioral;
```

Anexo H – *firmware.h*

```c
// This is free and unencumbered software released into the public
   domain.
//
// Anyone is free to copy, modify, publish, use, compile, sell, or
// distribute this software, either in source code form or as a compiled
// binary, for any purpose, commercial or non-commercial, and by any
// means.

#ifndef FIRMWARE_H
#define FIRMWARE_H

#include <stdint.h>
#include <stdbool.h>

// irq.c
uint32_t *irq(uint32_t *regs, uint32_t irqs);

// print.c
void print_chr(char ch);
void print_str(const char *p);
void print_dec(unsigned int val);
void print_hex(unsigned int val, int digits);

// hello.c
void hello(void);

// sieve.c
void sieve(void);

// multest.c
uint32_t hard_mul(uint32_t a, uint32_t b);
uint32_t hard_mulh(uint32_t a, uint32_t b);
uint32_t hard_mulhsu(uint32_t a, uint32_t b);
uint32_t hard_mulhu(uint32_t a, uint32_t b);
uint32_t hard_div(uint32_t a, uint32_t b);
uint32_t hard_divu(uint32_t a, uint32_t b);
uint32_t hard_rem(uint32_t a, uint32_t b);
uint32_t hard_remu(uint32_t a, uint32_t b);
void multest(void);

// stats.c
void stats(void);

#endif
```

Anexo I – *picoRV32_wrapper*

```
module picorv32_wrapper (
    input clk            ,
    input resetn         ,
    output trap           ,

    output mem_axi_awvalid,
    input mem_axi_awready,
    output [31:0] mem_axi_awaddr ,
    output [2:0] mem_axi_awprot ,

    output mem_axi_wvalid ,
    input mem_axi_wready ,
    output [31:0] mem_axi_wdata  ,
    output [3:0] mem_axi_wstrb  ,

    input mem_axi_bvalid ,
    output mem_axi_bready ,

    output mem_axi_arvalid,
    input mem_axi_arready,
    output [31:0]  mem_axi_araddr ,
    input [2:0] mem_axi_arprot ,

    input mem_axi_rvalid ,
    output mem_axi_rready ,
    input [31:0] mem_axi_rdata  ,
    input [31:0] irq            ,
    output trace_valid    ,
    output [35:0] trace_data
);

  picorv32_axi #(
    .COMPRESSED_ISA(1),
    .ENABLE_MUL(1),
    .ENABLE_DIV(1),
    .ENABLE_IRQ(1),
    .ENABLE_TRACE(1)
  ) uut (
    .clk            (clk            ),
    .resetn         (resetn         ),
    .trap           (trap           ),
    .mem_axi_awvalid(mem_axi_awvalid),
    .mem_axi_awready(mem_axi_awready),
    .mem_axi_awaddr (mem_axi_awaddr ),
    .mem_axi_awprot (mem_axi_awprot ),
    .mem_axi_wvalid (mem_axi_wvalid ),
    .mem_axi_wready (mem_axi_wready ),
    .mem_axi_wdata  (mem_axi_wdata  ),
```

```
49      .mem_axi_wstrb  (mem_axi_wstrb  ),
50      .mem_axi_bvalid (mem_axi_bvalid ),
51      .mem_axi_bready (mem_axi_bready ),
52      .mem_axi_arvalid(mem_axi_arvalid),
53      .mem_axi_arready(mem_axi_arready),
54      .mem_axi_araddr (mem_axi_araddr ),
55      .mem_axi_arprot (mem_axi_arprot ),
56      .mem_axi_rvalid (mem_axi_rvalid ),
57      .mem_axi_rready (mem_axi_rready ),
58      .mem_axi_rdata  (mem_axi_rdata  ),
59      .irq            (irq            ),
60      .trace_valid    (trace_valid    ),
61      .trace_data     (trace_data     )
62    );
63
64  endmodule
```