Universidade Federal de Minas Gerais

Escola de Engenharia

Laboratório de Arquitetura e Organização de Computadores

Turma PA1

Discentes:
Danilo Garcia Mariano
Raynner Schnneider Carvalho
Stephanie Costa de Avelar

Matrículas:
2019027466
2019111530
2019101070

22 de julho de 2022

1 Introdução

O presente trabalho tem como objetivo a implementação de uma microarquitetura ciclo único utilizando a linguagem de descrição de *hardware* VHDL. Optou-se por criar uma arquitetura baseada em RISC (*Reduced Instruction Set Computer*), implementando um subconjunto de instruções do MIPS.

2 Definição da Arquitetura de Ciclo Único

A arquitetura implementada tem as seguintes características:

- Tamanho do dado: 32 bits
- Tamanho das instruções: 32 bits
- Capacidade de endereçamento de memória: 8kB
- Formas de endereçamento: Register-Only, Imediato, com Base, Relativo ao PC, Pseudo-direto
- Formas de IO: interrupções, polling e IOs mapeados em memória.
- Tipo do conjunto de instruções: RISC
- Modelo de arquitetura: Harvard
- Modelo de CPU: Ciclo Único
- Endianess: Little Endian

2.1 *Instruction Set*

O primeiro passo para o desenvolvimento da microarquitetura de um processador de ciclo único é o *Instruction Set*, que nada mais é do que o conjunto de instruções básicas que o processador tem. Será implementado o seguinte subconjunto de instruções MIPS:

- Lógicas/aritméticas: ADD, SUB, OR, XOR, NOR, SLT, DIV, MULT
- Com imediato: ADDI, ORI, LUI
- Deslocamentos: SLL, SRL
- Resultados de divisões e multiplicações: MFHI, MFLO
- Saltos condicionais: BEQ, BGTZ
- Saltos incondicionais: J, JAL, JR

- Interação com a controladora de interrupções: MFC0, MTC0, ACK (Essa instrução não existe no MIPS)
- Interação com memória e periféricos: LW, SW

O documento "Discriminação de Instruções"detalha o uso de cada instrução. Pelo fato desse processador ser baseado em MIPS32, pode-se usar o cartão mips como referência também, exceto para a instrução "ACK".

2.2 Mapeamento de Memória

O mapa de memória é mostrado na figura 1.

Faixa de endereços (hex)	Tamanho da mem. (bytes)	Utilidade da memória	Nome do componente vhd de memória
0x1FFF – 0x1800	2 KB (2048 B)	Reservado	memd
0x17FF – 0x1400	1 KB (1024 B)	Memória de dados dinâmica (*stack e heap*)	memd
0x13FF – 0x1000	1 KB (1024 B)	Memória de dados estática	memd
0x0FFF – 0x0040	~4 KB (4032 B)	Memória de programa	memi
0x003F – 0x0000	64	Periféricos	Periféricos

Figura 1: Mapa de memória

Para traduzir os endereços virtuais do mapa de memória em endereços reais das memórias, foi implementado um decodificador de endereços, que ativa o dispositivo em que se deseja escrever/ler quando o endereço virtual mapeia para tal dispositivo. A figura 2 mostra esse bloco. A saída SEL_READ_DEVICE serve para selecionar de qual dispositivo a leitura de dados será realizada. O endereço do PC também precisa ser decodificado para ser enviado para a memória de instruções.

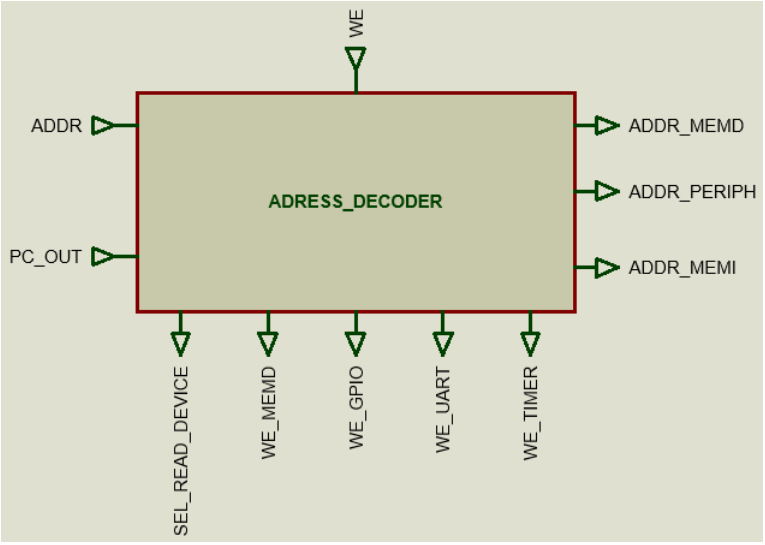

Figura 2: Decodificador de endereços

Os endereços reservados para os periféricos implementados são mostrados na figura 3.

Faixa de endereços (hex)	Utilidade da memória	Nome do componente vhd
0x17 – 0x10	TIMER	TMER_CTL
0x0F – 0x0A	UART	UART_CTL
0x09 – 0x00	GPIO	gpio_address_decoder

Figura 3: Mapa de endereços dos periféricos

2.3 Registradores

Foram definidos no datapath 32 registradores para realizar operações rápidas evitando demorados acessos à memória. Os registradores são definidos da mesma forma que no MIPS32 e são mostrados na figura 4.

Name	Number	Use
$0	0	the constant value 0
$at	1	assembler temporary
$v0–$v1	2–3	function return value
$a0–$a3	4–7	function arguments
$t0–$t7	8–15	temporary variables
$s0–$s7	16–23	saved variables
$t8–$t9	24–25	temporary variables
$k0–$k1	26–27	operating system (OS) temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	function return address

Figura 4: Registradores

3 Microarquitetura

A microarquitetura foi implementada de acordo com o fluxo de projeto sugerido pelo professor, no qual implementa-se primeiramente o datapath de acordo com as operações necessárias para realizar as instruções do set e após isso a controladora é construída para gerar os sinais de controle de acordo com os campos opcode e funct das instruções.

3.1 Caminho de Dados Ciclo Único

O caminho de dados projetado é mostrado na figura 6. Ele foi implementado em VHDL, o módulo Top-Level é o via_de_dados_ciclo_unico e os seguintes módulos são instanciados nele: ula, somador, registrador, pc, mux81, mux41, mux21, multiplicador_divisor, extensor, deslocador, banco_registradores.

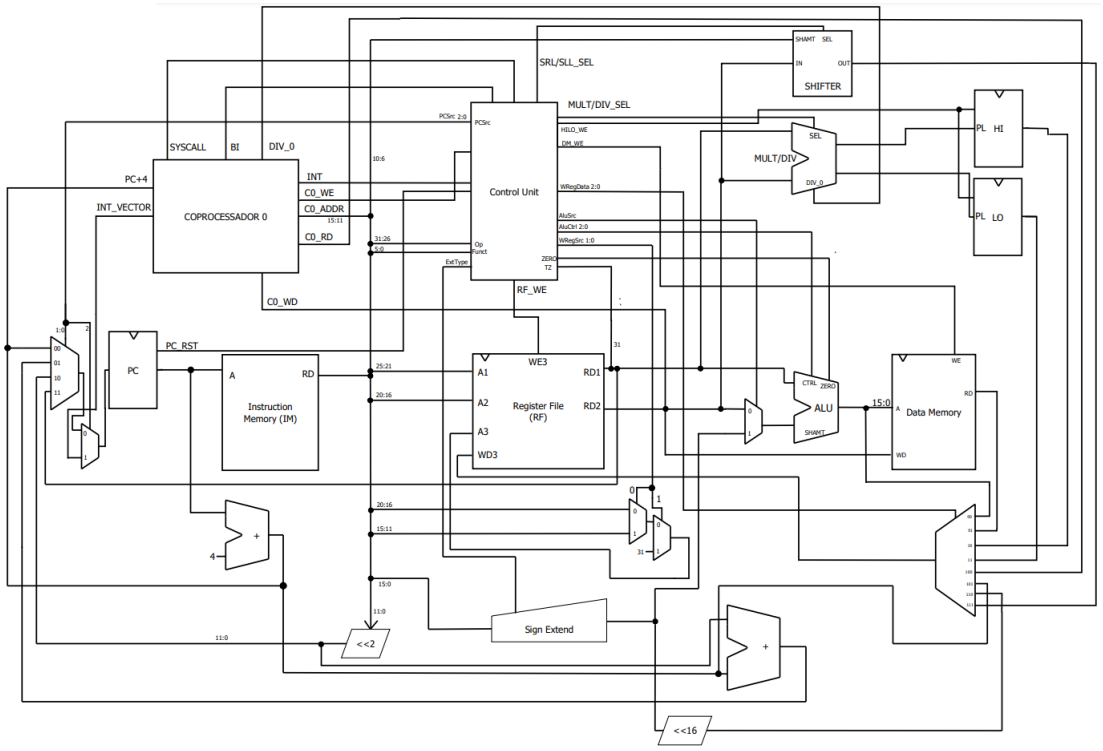

Figura 5: Datapath

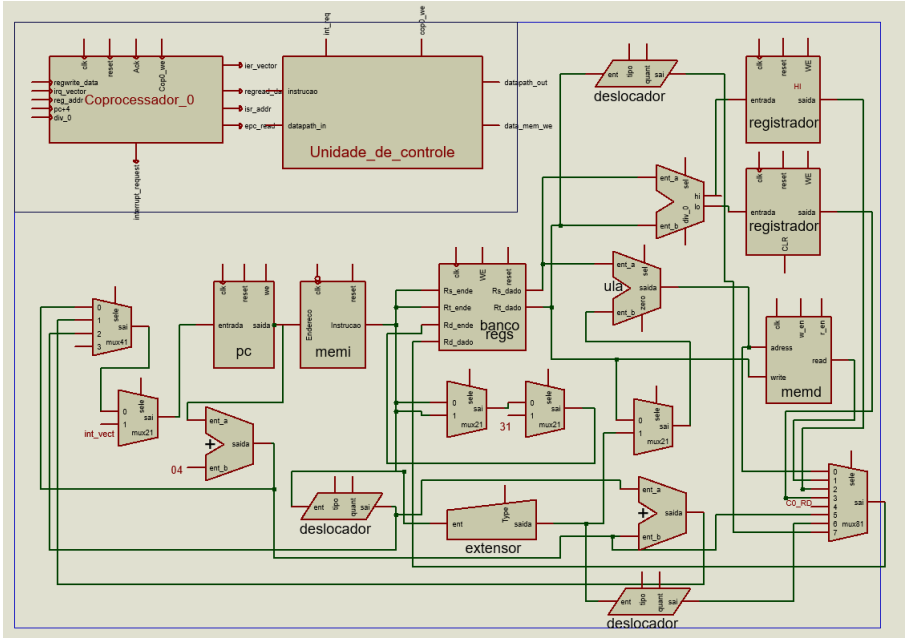

Figura 6: Datapath

3.2 Controladora Ciclo Único

A controladora (unidade_de_controle_ciclo_unico) foi construída em dois blocos: o decodificador principal (main_decoder) e o decodificador da ALU (ALU_Decoder), que geram os sinais para o datapath, memórias e coprocessador 0, a partir dos campos opcode e funct da instrução e também entradas provenientes de outros blocos. Foi criada uma tabela verdade para cada decodificador para facilitar a codificação em VHDL, que foi feita utilizando estruturas "case"com opcode e funct.

3.3 Memórias

Tanto a memória de dados (memd) quanto a de instruções (memi) foram implementadas utilizando as Megafunctions do Quartus (altsyncram), para mapear para memórias do FPGA, sem consumir blocos lógicos. Na memória de instruções foi utilizado arquivo MIF para inicializar a memória com o programa mais facilmente.

4 Implementação dos Periféricos

Baseando-se em códigos disponibilizados pelo professor, foi feita a implementação de alguns periféricos, como apresentado a seguir. A validação dos mesmos foi feita via simulação no ModelSim e em seguida via gravação em dispositivo FPGA.

4.1 Controladora de Interrupções (Coprocessador 0)

A controladora de interrupções (coprocessador_0) é responsável por receber as requisições de interrupção provenientes dos periféricos e desviar o processador para a rotina de interrupção de cada periférico, de acordo com a prioridade de cada um. A figura 7 mostra as entradas e saídas desse bloco.

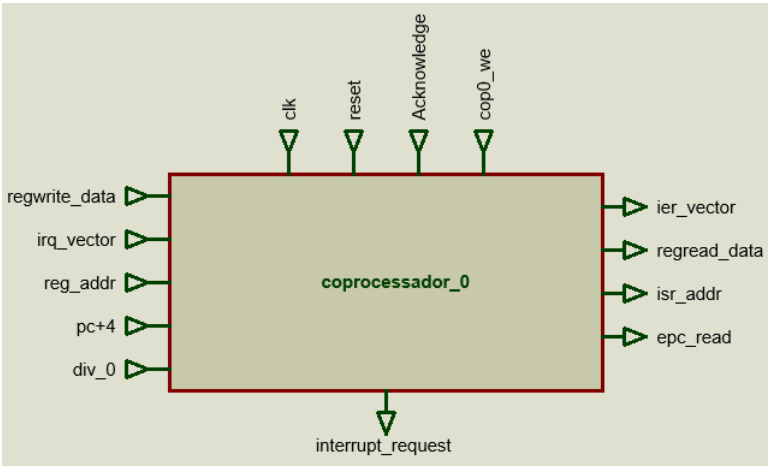

Figura 7: Coprocessador 0

A entrada Acknowledge vem do processador (gerada pela instrução ACK) e indica para a controladora que a interrupção em questão foi tratada, de forma que ela possa limpar essa interrupção (apenas na controladora, pois a interrupção de cada periférico deve ser limpada via software dentro da rotina de tratamento através de instruções de salvamento na memória (sw). Na seção de cada periférico isso será explicado em mais detalhes) e ir para a próxima, caso haja. O bloco interrupt_ctl, interno ao coprocessador é o que de fato controla as interrupções e decide qual deve ser atendida. Dentro da controladora também há um banco de registradores para armazenar as rotinas de interrupção de cada periférico. Além disso, existem três registradores avulsos: EPC, que armazena o endereço para qual o processador deve retornar após o tratamento da interrupção (vem da entrada pc+4); IER, no qual é configurado quais interrupções estão habilitadas e o IFR, que armazena as instruções pendentes. Existe uma saída ier_vector que tem o valor do registrador IER, para que certos periféricos saibam se a interrupção está habilitada ou não para eles. A figura 8 mostra os endereços dos registradores.

Banco de regs do C0		
Numero	**Registrador**	**Descrição**
0	Scratchpad Memory	ISR UART_Received
1		ISR UART_Transmited
2		ISR TIMER_A
3		ISR TIMER_B
4		ISR GPIO_A
5		ISR GPIO_B
6		ISR DIV_0
7		ISR Syscall
8		ISR Bad Instruction
9	IER	Interrupt Enable Register
10	IFR	Interrupt Flag Register
11	EPC	Endereço de retorno do PC

Figura 8: Registradores coprocessador 0

A figura 9 mostra o que cada bit dos registradores IER e IFR representa. Essa é a ordem de prioridade do atendimento às interrupções, sendo o bit menos significativo o de maior prioridade. Nessa mesma sequência são organizados os bits do vetor de interrupção que vem dos periféricos (irq_vector): a interrupção do Rx da UART deve ser conectado no bit menos significativo do vetor, e assim por diante. O registrador EPC é de 12 bits e armazenará endereços apenas, então não é necessário detalhar o que cada bit significa.

IER							
BITS	6	5	4	3	2	1	0
Habilita qual exceção?	DIV_0	GPIO_B	GPIO_A	TIMER_B	TIMER_A	UART_Transmited	UART_Received
IFR							
BITS	6	5	4	3	2	1	0
Quem solicitou?	DIV_0	GPIO_B	GPIO_A	TIMER_B	TIMER_A	UART_Transmited	UART_Received

Figura 9: Detalhamento dos bits dos registradores

Para escrever nos registradores do coprocessador deve-se utilizar a instrução MTC0 e para ler, MFC0. Nessas instruções deve-se especificar o endereço do registrador desejado e o conteúdo a ser escrito nele (MTC0) ou então o registrador do processador que receberá o conteúdo (MFC0). As entradas regwrite_data, reg_addr e cop_we são utilizadas para escrever, e a saída de dados para leitura é em regread_data. Os endereços das rotinas de tratamento das interrupções devem ser previamente escritos no banco de registradores utilizando MTC0. O coprocessador foi validado inicialmente através de simulações no ModelSim e posteriormente no FPGA, integrado no microcontrolador completo. A figura 10 mostra a simulação que foi utilizada para validar o design, utilizando o testbench tb_coprocessador_0.

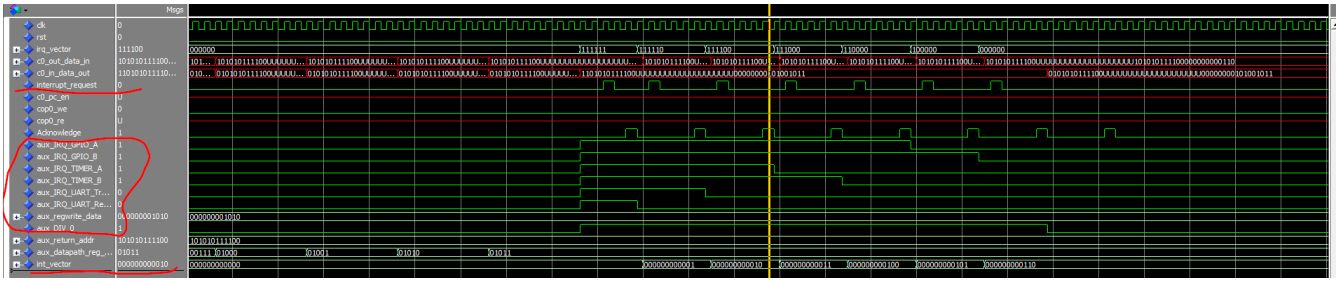

Figura 10: Simulação do C0

4.2 UART

A UART contém 3 blocos: UART_RX, UART_TX e UART_CTL, que é o principal, pois controla os outros dois e faz a interface com o processador, de forma que o periférico é enxergado como uma faixa de endereços de memória, podendo ser acessado via instruções sw e lw. OS blocos de Tx e Rx são quase identicos aos fornecidos pelo professor, mudando apenas que o parâmetro de clocks por bit (que confira a baud rate) é agora passado através de uma entrada binária e não mais pelo generic map. A UART possui 6 endereços de memória dedicados para interagir com o processador, e são mostrados na figura 11.

Endereço (hex)	W/R	Propósito
A	W/R	Configurar a quantidade de clocks por bit para definir a baud rate
B	R	Ler dado recebido no Rx
C	W/R	Configurar o Tx
D	R	Ler bit de status do Tx para saber se ele está ocupado
E	W	Limpar interrupção do Tx
F	W	Limpar interrupção do Rx

Figura 11: Registradores da UART

O primeiro endereço serve para configurar a baud rate tanto para Tx quanto para Rx. A configuração é indireta e se dá pelo parâmetro de clocks por bit. Para calcular esse parâmetro, basta utilizar a equação 1. O número máximo a ser representado por esse parâmetro é um inteiro não sinalizado de 19 bits (isso permite configurar a menor baut rate padrão utilizando o clock máximo de 50MHz do kit DE10).

$$CPB = \frac{Clock}{BaudRate} \tag{1}$$

Para ler o dado de 8 bits recém recebido no Rx, basta ler o endereço B. Os dados estarão na parte menos significativa da palavra de 32 bits. A configuração do Tx é feita pelo endereço xC. Ele deve ser configurado como mostra a figura 12.

Bits		
31:9	8	7:0
X	Ativação	Dado a ser enviado

Figura 12: Setup do Tx

Quando o bit de ativação é 1 o Tx envia o dado. É importante saber que quando o Tx termina de enviar o dado e ativa a interrupção, o valor armazenado nesse endereço é apagado, para que ele não fique enviando dados indefinidamente. Portanto, se deseja enviar mais algum dado é necessário reescrever o setup nesse endereço.

O endereço xD é apenas de leitura e serve para verificar se o Tx está ocupado. Caso o bit mais significativo for alto, ele está enviando um dado. Os dois últimos endereços servem para limpar as interrupções e reiniciar Tx e Rx para que eles possam enviar/receber mais dados. Para limpar é necessário que o dado a ser escrito nesses endereços seja 1.

A UART foi validada primeiramente no ModelSim (figura 13) com o testbench UART_TB e posteriormente no FPGA, com microcontrolador completo.

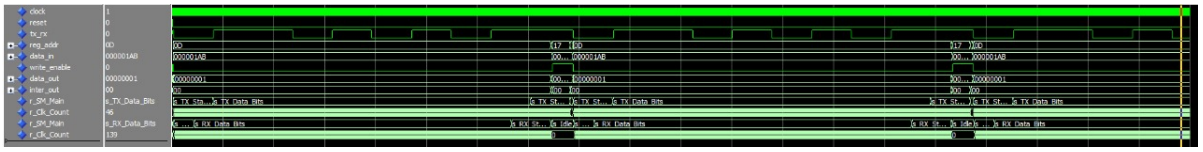

Figura 13: Simulação da UART

4.3 GPIO

O periférico GPIO é responsável por gerenciar os pinos de I/O's do microcontrolador, ele é capaz de definir quais pinos serão configurados como entrada e quais serão configurados como saída, ele quem determina os valores das saídas, faz a leitura dos pinos de entradas e habilita interrupção para uma borda de subida ou de descida para o pino em específico que foi configurado.

A GPIO do microcontrolador, que foi projetada do zero, pois os templates disponibilizados não atendiam aos requisitados desejados pelo grupo, possui 2 grupos de portas, sendo eles: PORT A e PORT B, cada um gerenciando 8 pinos. Para executar essas tarefas, o periférico foi subdivido em módulos: gpio, gpio_module, port_a e port_b, sendo os dois últimos um instanciamento do componente "cell_io", que representa um conjunto genérico de portas bidirecionais com buffers tristate para utilizar escrita e leitura conforme programação, enquanto que no "gpio_module"se encontram os registradores de configuração e o detector de interrupção de cada pino e no "gpio"estão essas três instâncias se interconectando. Dentro do VHDL "gpio_module"existem os 10 registradores utilizados no

periférico e a decodificadora de endereços "gpio_address_decoder", que através do endereço "addr"define qual registrador será escrito ou qual conteúdo do registrador será selecionado pelo componente "mux41"para ser enviado ao processador.

Os registradores referentes ao GPIO são mostrados na figura 14 abaixo, junto com seus respectivos endereços, descrição da função e como o dado é interpretado.

REGISTRADORES DO GPIO				
Endereço	**Index**	**Registradores**	**Função**	**Configuração**
0x0000	0	DIR_A	Define se o pino é entrada ou saída de cada pino do PORT_A	0 = Saída / 1 = Entrada
0x0001	1	DIR_B	Define se o pino é entrada ou saída de cada pino do PORT_B	0 = Saída / 1 = Entrada
0x0002	2	DATAIN_A	Armazena o estado lógico da entrada de cada pino do PORT_A	
0x0003	3	DATAIN_B	Armazena o estado lógico da entrada de cada pino do PORT_B	
0x0004	4	DATAOUT_A	Armazena o estado lógico da saída de cada pino do PORT_A	
0x0005	5	DATAOUT_B	Armazena o estado lógico da saída de cada pino do PORT_B	
0x0006	6	IE_A	Habilita a interrupção de cada pino do PORT_A	1 = Habilitado / 0 =Desabilitado
0x0007	7	IE_B	Habilita a interrupção de cada pino do PORT_B	1 = Habilitado / 0 =Desabilitado
0x0008	8	IF_A	Status das interrupções dos pinos	1 = Houve interrupção
0x0009	9	IF_B	Status das interrupções dos pinos	1 = Houve interrupção

Figura 14: Registradores do GPIO

Os registradores de escrita correspondem a: DIR_A, DIR_B, DATAOUT_A, DATAOUT_B, IE_A e IE_B, sendo esses impossibilitados de serem requisitados para leitura pela cpu. Já os demais registradores: DATAIN_A, DATAIN_B, IF_A, IF_B são registradores de leitura e não podem ser escritos, com exceção dos dois últimos, que por serem utilizados para indicar que houve uma interrupção, devem permitir que a cpu limpe os flags para indicar que a interrupção já foi atendida.

O módulo GPIO foi validado no ModelSim (figura 15), e posteriormente no FPGA. Vale ressaltar que os sinais destacados em azul é proposital, pois como são utilizados buffers tristates, alguns sinais devem permanecer em alta impedância para indicar que o pino está sendo utilizado para operação contrária, leitura ou escrita.

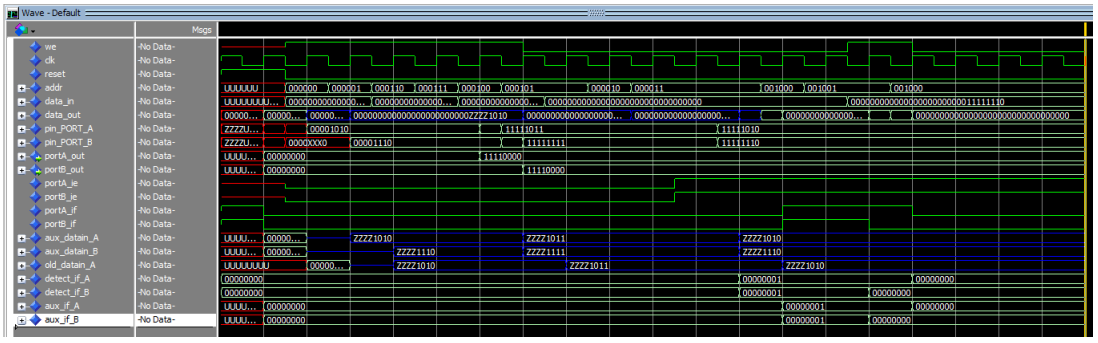

Figura 15: Simulação do GPIO

4.4 Timers

Foram implementados dois timers (TimerA e TimerB), que são instanciados e controlados pelo bloco timer_ctl. A ideia do timer_ctl é praticamente a mesma do UART_CTL, servindo apenas para integrar os sub blocos e realizar a interface com o processador. Os endereços do timer são mostrados na figura 16.

Endereço (hex)	W/R	Propósito
10	W/R	ConfigA
11	W/R	TargetA
12	W	ClearA
13	R	CountA
14	W/R	ConfigB
15	W/R	TargetB
16	W	ClearB
17	R	CountB

Figura 16: Registradores do Timer

Para cada timer existem 4 endereços. O de configuração serve para iniciar/parar a contagem e para definir a divisão de clock desejada. O bit menos significativo quando em alto habilita a contagem e para quando estiver em 0. Os bits 3 a 1 definem a divisão de clock, de acordo com a figura 17.

bits 3:1	Divisão
b000	1
b001	2
b010	4
b011	8
b100	16
b101	32
b110	64
b111	128

Figura 17: Divisões de clock do Timer

O Target é o valor até o qual o timer deve contar. Pode ser configurado um valor de até 32 bits. Quando o timer conta até o target, ele gera uma interrupção. Os endereços de clear são destinados a limpar a interrupção e reiniciar a contagem do zero. Para limpar é necessário que o dado a ser escrito nesses endereços seja 1. Por fim, os endereços de Count contém o valor atual da contagem de cada um dos timers.

A figura 18 mostra a simulação no ModelSim realizada para validar esse periférico. Foi utilizado o testbench tb_timer. Ele foi validado posteriormente no FPGA junto com o microcontrolador completo.

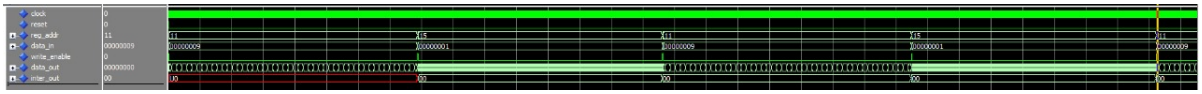

Figura 18: Simulação do Timer

5 Programação do microcontrolador

A programação do microcontrolador pode ser feita em assembly utilizando o MARS. Após a compilação é possível exportar um arquivo de texto que contém uma lista sequencial dos códigos de máquina das instruções em hexadecimal. A partir desse arquivo e do script "hex2mif.py"presente no repositório é possível gerar um arquivo de inicialização de memória (.MIF). Esse arquivo deve ser colocado na pasta do projeto do Quartus e seu nome deve ser colocado no generic map da megafunction da memi, como mostra a figura 19.

```
altsyncram_component : altsyncram
GENERIC MAP (
   address_aclr_a => "NONE",
   clock_enable_input_a => "BYPASS",
   clock_enable_output_a => "BYPASS",
   init_file => "rom_init.mif",
   intended_device_family => "MAX 10",
   lpm_hint => "ENABLE_RUNTIME_MOD=NO",
   lpm_type => "altsyncram",
   numwords_a => (2**(MI_ADDR_WIDTH-2)),
   operation_mode => "ROM",
   outdata_aclr_a => "NONE",
   --outdata_reg_a => "CLOCK0",
   widthad_a => (MI_ADDR_WIDTH-2),
   width_a => INSTR_WIDTH,
   width_byteena_a => INSTR_WIDTH
)
```

Figura 19: Arquivo MIF

O script de python deve ser chamado no terminal da seguinte forma:

-python hex2mif.py arquivo_do_mars.txt arquivo_mif.mif

Substituindo os nomes pelos nomes dos seus arquivos. Caso a máquina não tenha python instalado é possível executar em algum interpretador online como o replit. O script já corrige os endereços gerados pelo MARS, pois o MARS é feito para o MIPS32 original, e o mapa de endereços desse processador é diferente. O detalhe é que não será possível executar um programa que o MARS gere que tenha instruções fora do set desse processador, e isso deve ser corrigido manualmente. A instrução ACK não é nativa e também deve ser inserida normalmente. Quando for carregar um endereço utilizando a pseudoinstrução "la"do MARS, ela é traduzida para instruções LUI e ORI. Nesse caso também é necessário corrigir manualmente o endereço a ser carregado.

6 Pipeline

A fim de se reduzir a complexidade do *pipeline*, optou-se por utilizar instruções *nop* para evitar a ocorrência de *hazards*. Na figura a seguir pode-se ver a implementação do *pipeline* de 3 estágios.

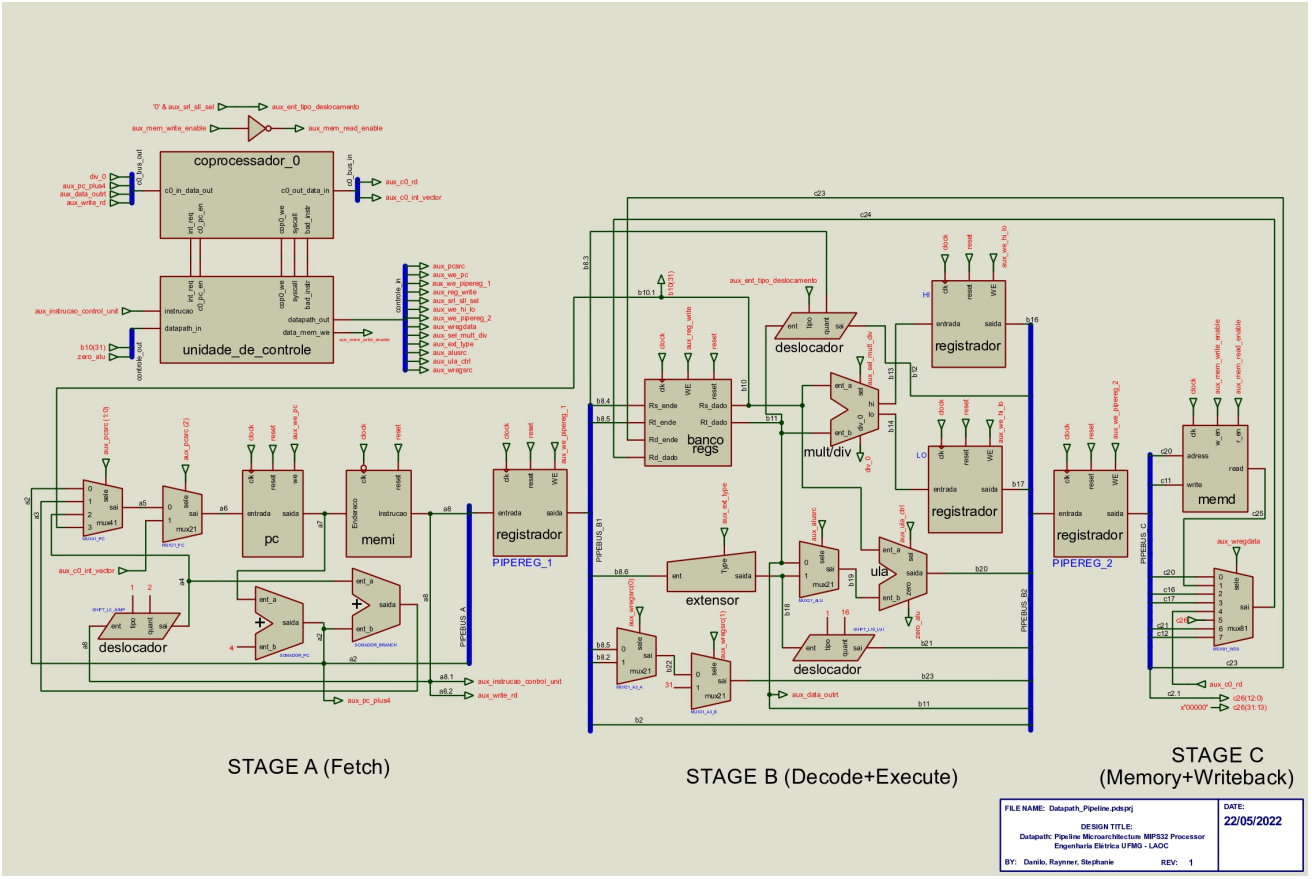

Figura 20: Caminho de Dados Pipeline

Pretendía-se, a priori, utilizar uma *Hazard Unit* para lidar com as ocorrências dos *hazards*. Todavia, diante do insucesso na validação da mesma, decidiu-se utilizar instruções nop para evitar tais ocorrências.

7 Conclusão

Através da realização do presente trabalho, foi possível projetar com sucesso um processador e seus periféricos,bem como validar o seu funcionamento através de um dispositivo FPGA, como demonstrado em sala em de aula.