Relatório do Trabalho em Grupo - PCID

UNIVERSIDADE FEDERAL DA BAHIA
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Ana Clara Batista
Salvador, BA
clara.malheiro@ufba.br

André Paiva C. Rodrigues
Salvador, BA
andre.paiva@ufba.br

Júlia Carvalho de Souza
Salvador, BA
julia.souza@ufba.br

Resumo

Este é um relatório sobre o trabalho em grupo do semestre de 2021.2 da matéria ENGG56 – Projeto de Circuitos Integrados Digitais, lecionada pelo Professor Wagner L. A. de Oliveira. O projeto foi realizado em Verilog, utilizando o programa Quartus, visando síntese e execução na DE2-115, placa de desenvolvimento baseada em FPGA da linha Cyclone IV.

Palavras-Chave: Circuitos, Digital, FPGA, Verilog, Circuit Design, Quartus, ModelSim

Introdução

Neste trabalho, procuramos aplicar os conhecimentos da matéria Projeto de Circuitos Integrados Digitais aprendidos ao longo do semestre. Foram implementados 4 projetos em Verilog no programa Quartus versão Lite 20.1 da Intel Corporation, bem como testbenches para avaliar o correto funcionamento de todos os módulos desenvolvidos. Todos os projetos visaram a implementação na placa DE2-115 e contaram com simulações e validação no ModelSim.

Os 4 projetos desenvolvidos em Verilog se referem a um leitor e reconhecedor de informações provenientes de um controle remoto infra-vermelho, um acumulador conectado à uma memória RAM que lê seus conteúdos para somá-los e gravar novamente na memória seguindo um padrão, uma máquina de estados finitos visando geração de sinais de controle de outros módulos responsáveis por executar compressão MPEG de um bloco de pixels, e um circuito combinacional simples para controle de semáforo.

Todos os arquivos do trabalho podem ser encontrados através deste repositório no GitHub e no arquivo enviado através do e-mail para o Professor. Todas as imagens utilizadas neste relatório podem ser encontradas no repositório para visualização mais clara, no diretório Doc/assets.

1 Controle Remoto

O problema do Controle Remoto consta em realizar um módulo em Verilog capaz de identificar qual tecla está sendo pressionada por um controle remoto infravermelho a partir de um frame de um circuito receptor. O módulo apresenta três entradas: Clock, Reset e Serial, que é as informações recebidas pelo infra vermelho, e duas saídas: Tecla, o código da tecla pressionada, e Ready, que é ativada por 3 pulsos de clock caso o dado seja válido para ser decodificado.

1.1 Implementação

De acordo com a diagramação da entrada Serial apresentada no roteiro, implementamos uma série de condicionais (if e elses) para testar os casos apresentados e um contador para manter o monitoramento do estado atual.

Primeiramente, caso nenhuma entrada esteja presente na portal Serial e o contador esteja zerado, que representa o Lead Code, colocamos o valor 1 no contador. Com o começo das entradas do Custom Code, nos próximos 16 bits, ou seja, 16 pulsos de clock, os bits que chegam são salvos através de um shift na variável custom. Após o passo anterior, por mais 8 pulsos de clock, as informações contidas no Key Code são salvas na variável data. Ainda com as entradas seriais, por mais 8 pulsos de clock as informações contidas no Inv Key Code são salvas na variável data_inv.

Finalmente, durante o recebimento do bit final, concebido como End Code, é feito o teste de validação, onde compara se o data e o data_inv são frames complementares e se a tecla está dentro das especificações do projeto dadas através de uma tabela no relatório. Caso as informações estejam corretas, a saída Ready é ativada e a informação contida em data é passada para a saída Tecla. Mantemos então por 3 pulsos de clock a tecla Ready ativada com os dados na saída e, por fim, zeramos todas as variáveis e as saídas para o possível recebimento da próxima entrada.

1.2 Pin Assignments

Após a implementação da lógica, foi iniciado o planejamento dos pinos. Para isso, foi utilizado um módulo PLL para transformar o clock de entrada básico da placa de 50MHz para o usado usualmente num controle remoto de 38kHz, no qual alimentou o sinal de Clock no nosso módulo. As saídas da Tecla foram implementadas em 8 LED‘s vermelhos e o sinal Ready foi implementado em um LED verde da placa. Por fim, o Reset foi colocado em uma chave KEY[0] e a entrada Serial foi ligado diretamente no leitor infravermelho da placa IRDA_RXD.

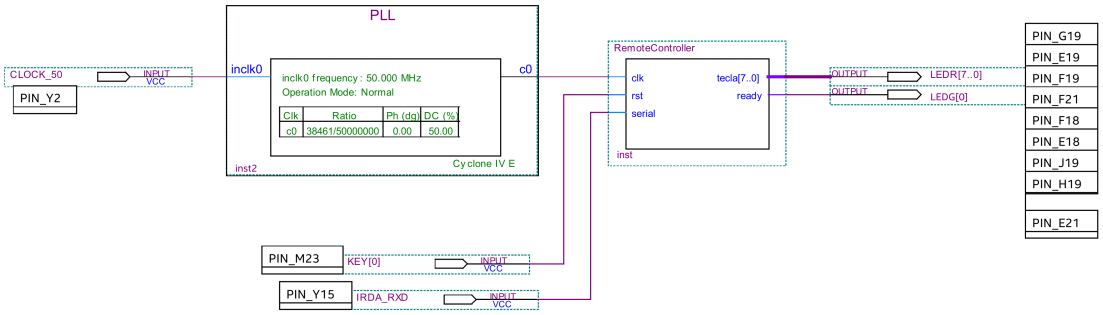

Figura 1 – Diagrama de conexões do módulo RemoteController

1.3 Time Constraints

Após fazer a análise e síntese e executar o Timing Analyzer, vimos que, inicialmente, as configurações de projeto não atendiam às especificações de limite de clock e previsão de delay de input/output. O clock configurado por padrão nos projetos iniciados no Quartus é de 1 GHz, o FPGA com o qual trabalhamos funciona com uma frequência máxima de 250 MHz e a análise resultou numa frequência máxima de operação de uma frequência na casa das centenas de MHz.

Geralmente, o sinal de controles remotos opera numa frequência na região dos 38 kHz, que é uma frequência demasiadamente inferior em relação à frequência máxima de operação do FPGA. Nesse sentido, não tivemos problemas na aplicação de Time Constraints; aplicamos o período de clock de 26 μs (aproximadamente 38461 Hz) e definimos o delay de input/output para 2,6 μs (10% do período do clock). Após as aplicações de constraints, rodamos o Timing Analyzer novamente e desta vez, como definimos um delay de input/output demasiadamente alto, o Timing Analyzer restringiu a frequência máxima de operação do circuito para 0.38 MHz (380 kHz). De qualquer forma, é uma frequência máxima suficiente, já que configuramos o circuito para operar a um clock de 38 kHz.

Para gerar a frequência de clock de 38 kHz, foi utilizado um PLL cuja entrada foi conectada ao pino de clock fixo de 50 MHz da placa, de modo que o PLL foi configurado para converter a frequência de 50 MHz em 38461 Hz.

1.4 Testbenches e resultados

Para finalizar, construímos um testbench onde realizamos quatro testes de acordo com as figuras 2, 3, 4 e 5. No primeiro caso, cujo waveform de simulação pode ser visualizado na Figura 2, temos um teste onde a Tecla 0x09, correspondente ao número 9 do controle, foi selecionada, detectada e validada corretamente pelo módulo RemoteController, sendo então o código redirecionado para a saída e o indicador de Ready permanecendo ativado por 3 pulsos de Clock. Além disso, foi exibida pelo testbench a primeira mensagem em azul no console como visto na Figura 6.

No segundo caso, cujo waveform de simulação pode ser visualizado na Figura 3, foi encontrada uma inconsistência em relação a informação do data e do data_inv, sendo elas não complementares. Assim, o teste teve um comportamento correto e finalizou a sua execução sem redirecionar informações para a saída nem ativar o sinal de Ready, e o testbench exibiu a segunda mensagem no console.

No terceiro caso, cujo waveform de simulação pode ser visualizado na Figura 4, foi encontrada uma inconsistência em relação a tecla recebida e a tabela de teclas existentes. Apesar de as informações de data e data_inv serem complementares, a informação lida não foi encontrada dentre as teclas reconhecíveis. Por tanto, a execução foi finalizada com uma mensagem de Valor de tecla fora da tabela, como mostra a imagem do console.

Por fim, o quarto caso, cujo waveform de simulação pode ser visualizado na Figura 5, assim como o primeiro caso, também demonstra um caso de sucesso, com consistência em relação aos dados e a tecla 0x1F pressionada, que corresponde a uma das setas do controle, sendo então enviada a informação de Tecla pressionada para o console, bem como sendo redirecionada a informação de tecla e o pulso de Ready para suas respectivas saídas por uma duração de 3 pulsos de clock.

Importante frisar que é possível perceber através do código presente no repositório no caminho ENGG56/RemoteController/Remote_TB.v que colocamos dois casos de comportamento incorreto do teste e suas respectivas mensagens de erro. No primeiro caso, checamos caso houve alguma inconsistência entre o comportamento do módulo e o testbench: Caso o sinal de Ready, que deveria estar ativo, não esteja ativo por um erro da saída deste pino ou por um erro de validação dos dados, a mensagem de Erro é sinalizada através do console juntamente com a indicação de que a simulação foi finalizada com erros, e a simulação para. No segundo caso, caso o sinal de Ready esteja ativado e as informações de tecla não forem válidas, também finalizamos a simulação com uma mensagem de Erro.

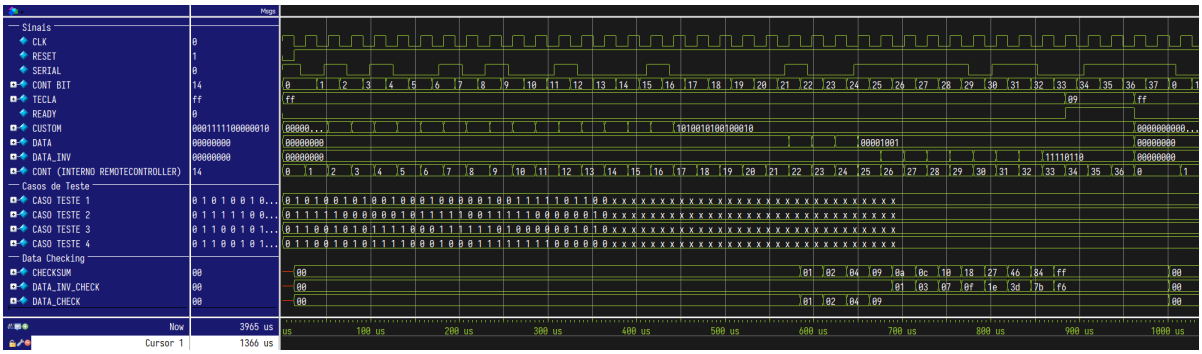

Figura 2 – Waveform da simulação do caso 1 do testbench

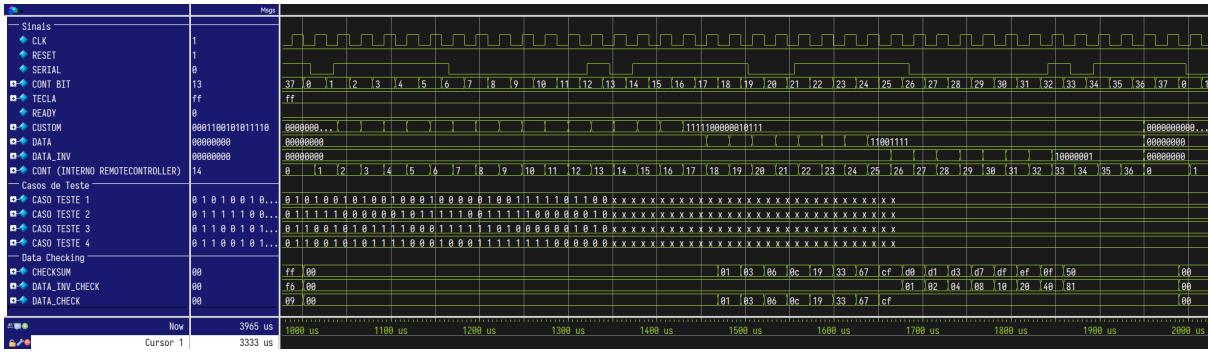

Figura 3 – Waveform da simulação do caso 2 do testbench

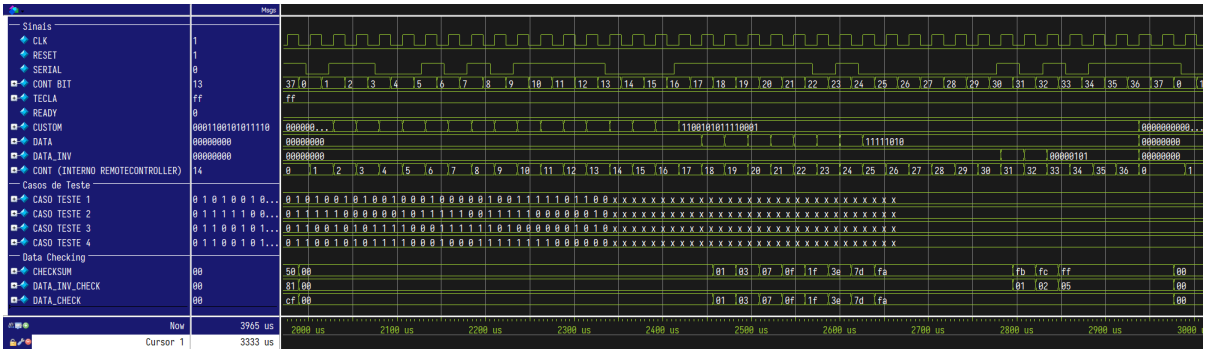

Figura 4 – Waveform da simulação do caso 3 do testbench

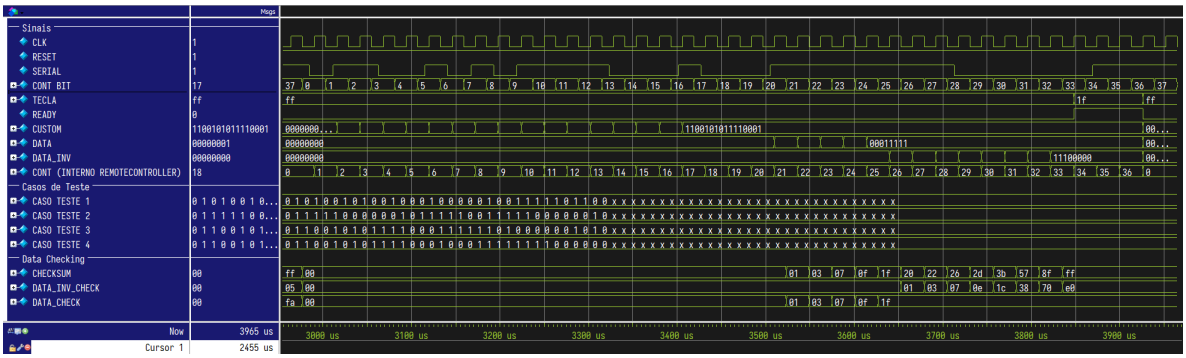

Figura 5 – Waveform da simulação do caso 4 do testbench

```
VSIM 24> run -all
# Tecla 09
# Data + Data_Inv == 50 != 0xff, comportamento correto
# Valor de tecla fora da tabela
# Tecla 1f
# Simulacao finalizada com sucesso.
# ** Note: $stop    : E:/ENGG56/RemoteController/Remote_TB.v(66)
#    Time: 3965 us  Iteration: 1  Instance: /Remote_TB
# Break in Module Remote_TB at E:/ENGG56/RemoteController/Remote_TB.v line 66
```

Figura 6 – Mensagens de console resultantes da simulação do testbench

2 Acumulador de dados de Memória Externa

2.1 Implementação

Para a implementação do acumulador de dados lidos de memória externa, construímos três módulos em Verilog, sendo um Acumulador, uma máquina de estados para gerar os sinais de controle (FSM) e um módulo TOP, que engloba e interconecta os dois módulos anteriormente citados. Além dos módulos construídos em Verilog, criamos também uma instância de memória RAM com 32 endereços de 16 bits cada a partir de ferramentas já disponibilizadas no Quartus.

Basicamente a função principal do circuito montado é agrupar os endereços da memória RAM em grupos de 8 bits, e a cada grupo são somados os conteúdos dos 7 primeiros endereços de modo que o resultado é gravado no oitavo endereço.

A interligação de todos os módulos instanciados pode ser visualizado no diagrama de blocos presente na Figura 7.

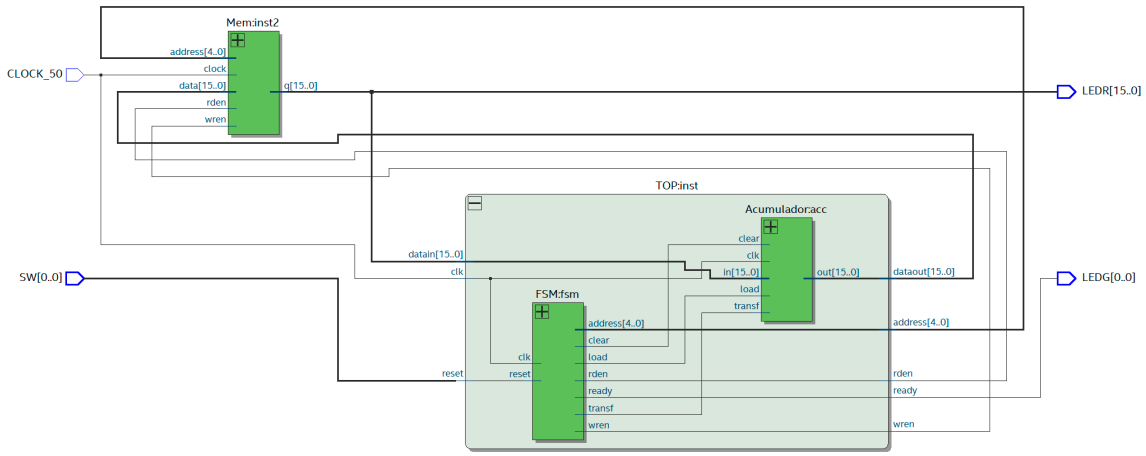

Figura 7 – Diagrama de blocos do circuito completo de interface entre Acumulador e RAM

2.1.1 Módulo *Acumulador*

A partir dos requisitos pedidos na questão, começamos criando um acumulador, que basicamente consiste em um somador de 16 bits com registradores de entrada e registradores de saída. Este acumulador possui uma entrada e uma saída de dados de 16 bits cada para interfaceamento com os dados da RAM. Possui também uma entrada de Clock, para sincronizar a base de tempo da transferência de dados, tanto de entrada quanto de saída. Além disso ele também dispõe uma entrada de Load, para armazenar os dados lidos na entrada no banco de registradores da entrada do somador; de uma entrada de Clear de valor lógico invertido, onde caso seja 0 o conteúdo acumulado nos registradores de saída é zerado; e por fim, uma entrada Transfer, onde permite que seja armazenado nos registradores de saída a soma do conteúdo anterior com o conteúdo dos registradores de entrada.. Com essa lógica, implementamos o Acumulador, cujo código pode ser acessado no repositório do GitHub no caminho ENGG56/Somador/Acumulador.v.

2.1.2 Módulo *FSM*

Após a construção do Acumulador, partimos para a montagem da FSM. Começamos criando um diagrama de estados para nos auxiliar nesta montagem. É importante frisar que o Clock de borda de descida é usado pra a troca de estados na FSM, bem como também para as ações no Acumulador, enquanto o Clock de subida é usado pra as operações de leitura e escrita da Memória RAM. O código da da FSM pode ser encontrada no repositório no caminho ENGG56/Somador/FSM.v. O diagrama da máquina de estados pode ser visualizado na Figura 8.

Basicamente o ciclo da FSM se inicia com a configuração de todos os parâmetros para 0 e a aplicação de um CLEAR no Acumulador. Logo depois, como o endereço inicial está setado por 0, o sinal RDEN é ativado e é esperado um ciclo de CLOCK para que haja a propagação dos dados lidos da memória RAM para a saída da mesma. Após o ciclo de espera, é ativado e desativado o sinal de LOAD para a captura do dado proveniente da memória RAM pelo acumulador, e um ciclo depois, RDEN é desativado. Com a captura efetuada, o sinal de TRANSFER é habilitado e desabilitado para efetuar a soma da informação capturada com a informação retida anteriormente. Após isso, a saída ADDRESS é incrementada, é esperado um ciclo de CLOCK para a propagação da informação e então ADDRESS é conferida. Caso ADDRESS seja igual a 7, 15, 23 ou 31, a informação do acumulador é gravada no endereço de Memória com a ativação e desativação do sinal WREN. Após a gravação, caso ADDRESS seja igual a 31, ADDRESS é zerado e o sinal de READY é ativado por um pulso de clock; caso contrário, ADDRESS é incrementado e volta ao ciclo de leitura e soma.

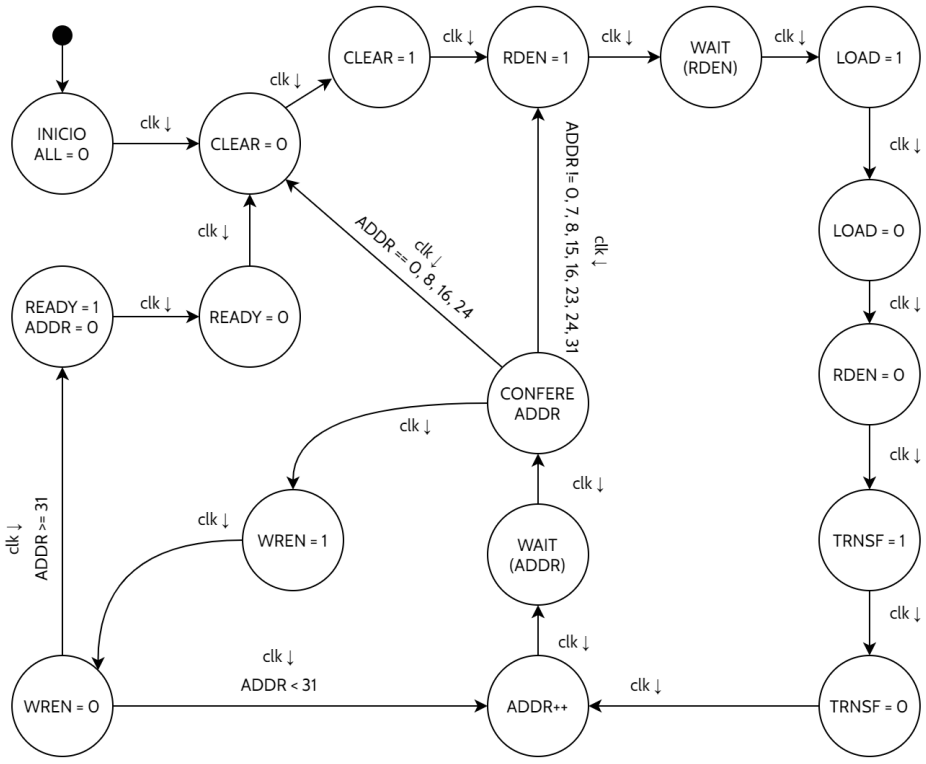

Figura 8 – Máquina de estados de controle de interfaceamento entre Acumulador e RAM

O módulo possui uma saída de 5 bits para endereço da memória RAM (Address), uma entrada de Reset e uma entrada de Clock, duas saídas Rden e o Wren que são os sinais de ReadEnable e de WriteEnable respectivamente enviados para a memória externa, as saídas Load, Clear e Transfer de sinais de controle que serão enviados para o acumulador, e a saída Ready que sinaliza a completude da operação sobre todos os endereços de Memória.

2.1.3 Módulo *TOP*

O módulo TOP é um módulo global com o intuito de instanciar dentro de si um módulo Acumulador e um módulo FSM, interligá-los e disponibilizar nas entradas e saídas apenas o necessário para a comunicação com a memória RAM, de modo que o meio externo não interfira diretamente nos sinais de controle entre a FSM e o Acumulador, como os sinais de Load e Transfer, por exemplo.

As entradas do módulo TOP são somente entradas para dados da memória RAM, Clock e Reset. As saídas do módulo são saídas de dados para memória RAM, seletor de endereço, sinais de enable de leitura e escrita e também o sinal de Ready que indica completude da execução do ciclo sobre todos os endereços da Memória RAM. O TOP pode ser acessado no repositório do GitHub no caminho ENGG56/Somador/TOP.v.

2.2 Pin Assignments

Após a implementação dos módulos, foram feitas as devidas conexões com os pinos físicos do FPGA. A saída da memória RAM foi disponibilizada nos LED's vermelhos de 15 a 0 (da ordem do mais significativo para o menos significativo), e a saída de READY foi disponibilizada no LED verde 0. A entrada de RESET foi associada ao Switch 0 e a entrada de Clock foi conectada ao pino de clock de 50 MHz do FPGA. O diagrama de conexões pode ser visualizado na Figura 9.

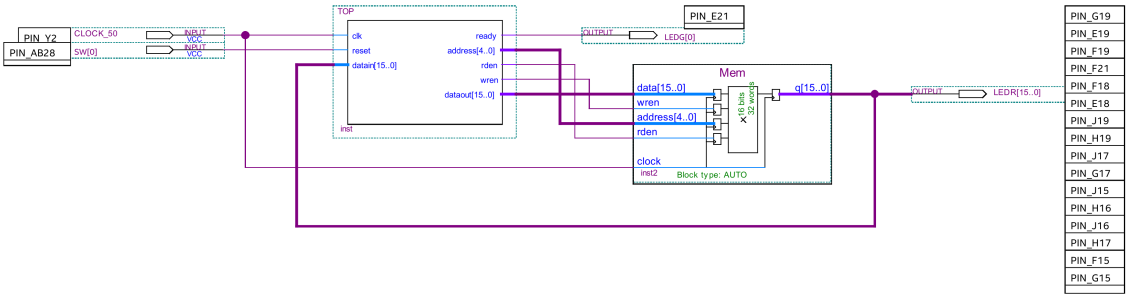

Figura 9 – Diagrama de pinos do acumulador

2.3 Time Constraints

Após fazer a análise e síntese e executar o Timing Analyzer, vimos que, inicialmente, as configurações de projeto não atendiam às especificações de limite de clock e previsão de delay de input/output. O clock configurado por padrão nos projetos iniciados no Quartus é de 1 GHz, o FPGA com o qual trabalhamos funciona com uma frequência máxima de 250 MHz e a análise resultou numa frequência máxima de operação de uma frequência na casa das centenas de MHz. Como a placa DE2-115 já conta com um pino de clock fixo de 50 MHz, configuramos o clock do projeto para 50 MHz (20 ns de período) e configuramos o delay de propagação de input/output para 2 ns (10% do período de clock). Após a aplicação dos time constraints, executamos o Timing Analyzer novamente e verificamos que todas as configurações de temporização atendiam ao necessário para garantir o correto funcionamento do circuito.

2.4 Testbenches e resultados

Foram implementados três testbenches, um para cada um dos módulos desenvolvidos. A implementação dos mesmos e as respectivas execuções serão explanadas a seguir.

2.4.1 Testbench do *Acumulador*

O testbench do acumulador foi construído baseado em uma contagem de 0 a 100. A cada ciclo de contagem, caso o valor do contador seja diferente de 20, 40, 60, 80 e 100, é gerado um novo número aleatório de 16 bits por meio da diretiva $random. Caso contrário, o sinal de clear é ativado para zerar o conteúdo do acumulador. O testbench contém um registrador interno de 17 bits que vai somando os valores a cada incremento de contagem que gera um novo valor aleatório, e os 16 bits menos significativos são comparados com o resultado da soma do acumulador. Caso a soma seja diferente dos 16 bits menos significativos, uma mensagem de erro é exibida no console e a simulação para. A cada soma correspondente ao resultado esperado, é exibida uma mensagem no console confirmando a coerência dos dados do acumulador, bem como um aviso em caso de estouro de carry (detectado pelo décimo sétimo bit do registrador interno do testbench). Caso a simulação corra bem em todas as etapas, uma mensagem indicando o êxito é exibida no console.

Na Figura 10 é possível visualizar a forma de onda resultante da execução dos estágios iniciais da simulação, e na Figura 11 é possível visualizar a forma de onda dos estágios finais. Na Figura 12 é possível visualizar as mensagens exibidas pelo testbench nos estágios finais da execução.

O código do testbench do Acumulador pode ser acessado no repositório do GitHub no caminho ENGG56/Somador/Acumulador_TB.v.

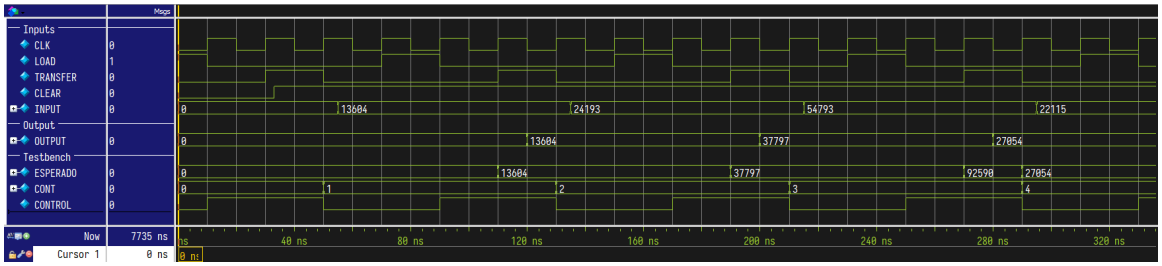

Figura 10 – Trecho da execução do testbench do acumulador

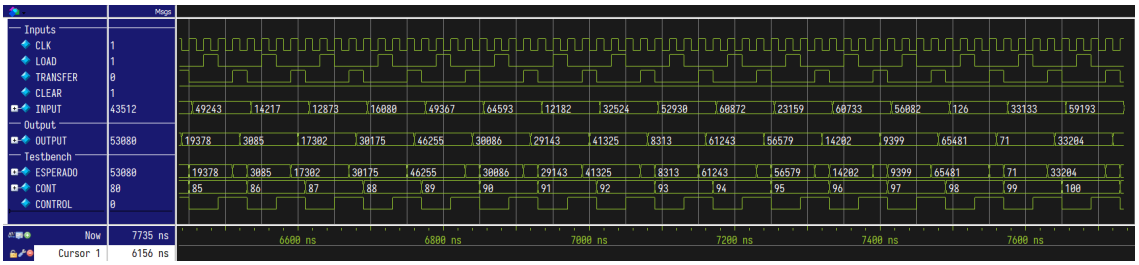

Figura 11 – Trecho da execução do testbench do acumulador

```
# Esperado =  68621, Resultado:  3085 (Carry ativado)
# Esperado =  17302, Resultado: 17302
# Esperado =  30175, Resultado: 30175
# Esperado =  46255, Resultado: 46255
# Esperado =  95622, Resultado: 30086 (Carry ativado)
# Esperado =  94679, Resultado: 29143 (Carry ativado)
# Esperado =  41325, Resultado: 41325
# Esperado =  73849, Resultado:  8313 (Carry ativado)
# Esperado =  61243, Resultado: 61243
# Esperado = 122115, Resultado: 56579 (Carry ativado)
# Esperado =  79738, Resultado: 14202 (Carry ativado)
# Esperado =  74935, Resultado:  9399 (Carry ativado)
# Esperado =  65481, Resultado: 65481
# Esperado =  65607, Resultado:    71 (Carry ativado)
# Esperado =  33204, Resultado: 33204
# Esperado =  92397, Resultado: 26861 (Carry ativado)
# Simulacao concluida com exito.
# ** Note: $stop    : E:/ENGG56/Somador/Acumulador_TB.v(83)
#    Time: 7735 ns  Iteration: 0  Instance: /Acumulador_TB
# Break in Module Acumulador_TB at E:/ENGG56/Somador/Acumulador_TB.v line 83
```

Figura 12 – Mensagens de console resultantes da execução do testbench do acumulador

2.4.2 Testbench da *FSM*

No testbench da FSM, verificamos o funcionamento durante um ciclo completo de execução para os 32 endereços, sempre capturando instantes de tempo de variação de estado dos sinais de controle para efetuar comparações e conferir se o intervalo de tempo entre variações de determinados sinais é suficiente para garantir o correto funcionamento do circuito e para garantir a não ocorrência de metaestabilidade e corrupção de dados da memória.

Caso haja variações entre determinados sinais num intervalo inferior a um ciclo de clock, uma mensagem de erro correspondente é exibida no console e a simulação é finalizada. Todas as condições possíveis foram tratadas e é possível visualizar nitidamente no código, que está disponível no caminho Caso a simulação corra bem por entre todos os ciclos, uma mensagem de sucesso é exibida no final da simulação.

Um trecho da waveform da execução e dos sinais da FSM pode ser visualizado na Figura 13. A waveform da execução completa pode ser visualizada na Figura 14. As mensagens de console resultantes da execução podem ser visualizadas na Figura 15. O código do testbench da FSM pode ser acessado no repositório do GitHub no caminho ENGG56/Somador/FSM_TB.v.

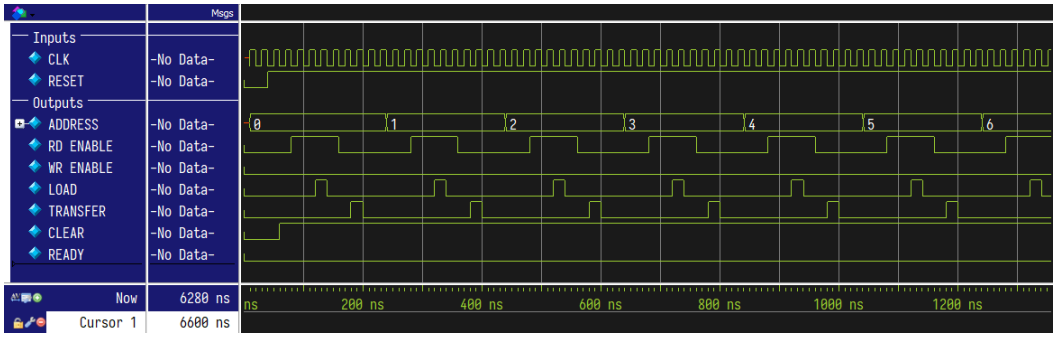

Figura 13 – Trecho da execução do testbench da FSM do Somador

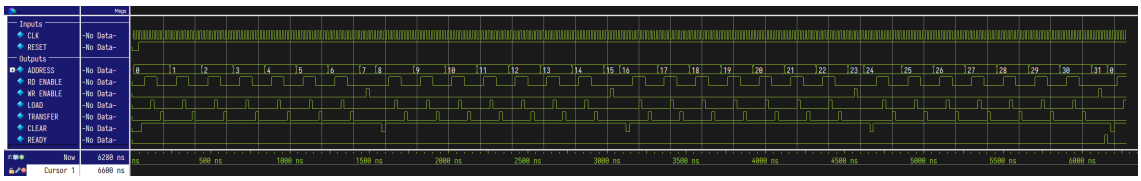

Figura 14 – Execução completa do testbench da FSM do Somador

```
VSIM 48> run -all
# Simulacao completada com sucesso.
# ** Note: $stop    : E:/ENGG56/Somador/FSM_TB.v(59)
#    Time: 6280 ns  Iteration: 0  Instance: /FSM_TB
# Break in Module FSM_TB at E:/ENGG56/Somador/FSM_TB.v line 59
```

Figura 15 – Mensagens de console resultantes da execução do testbench da FSM do acumulador

2.4.3 Testbench do *TOP*

No início do testbench do módulo TOP, todos os 32 endereços da memória RAM são clonados para um registrador interno e qualquer alteração nos dados é capturada posteriormente mediante ativação do sinal de WREN. Neste testbench, são avaliadas quatro condições de erro possíveis, e caso a simulação corra bem até o final da execução do ciclo de 32 endereços, uma mensagem de sucesso é exibida no console. Caso uma das condições de erro seja detectada, uma mensagem de erro é exibida no console e a simulação para. As condições de erro avaliadas são as seguintes:

- Alteração dos dados de entrada do acumulador muito próxima ao pulso de LOAD;
- Alteração de endereço de acesso da memória muito próxima à ativação de WREN;
- Alteração de endereço de acesso da memória durante WREN ativo;
- Pulso de RDEN não durar o suficiente para permitir a correta captura de dados pelo acumulador.

O código do testbench do módulo TOP pode ser acessado no repositório do GitHub no caminho ENGG56/Somador/TOP_TB.v.

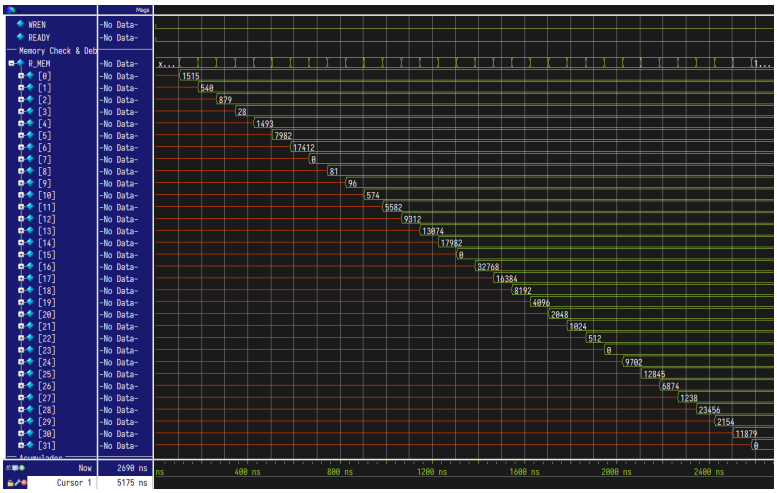

Figura 16 – Clonagem de memória RAM no Testbench do TOP

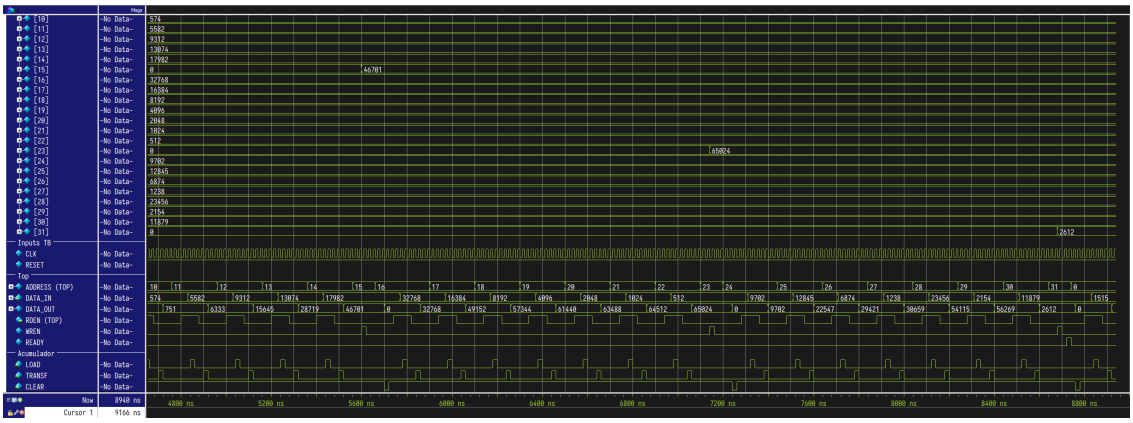

Figura 17 – Trecho final da execução do testbench do módulo TOP do Somador

```
VSIM 5> run -all
# Memoria RAM Clonada.
# Simulacao concluida com sucesso.
# ** Note: $stop    : E:/ENGG56/Somador/TOP_TB.v(140)
#    Time: 8940 ns  Iteration: 0  Instance: /TOP_TB
# Break in Module TOP_TB at E:/ENGG56/Somador/TOP_TB.v line 140
```

Figura 18 – Mensagens de console resultantes da execução do testbench do módulo TOP

3 Codificação de vídeo MPEG

Nesse terceiro projeto, foi apresentado o processo de codificação de vídeo MPEG, utilizado para compressão de vídeo, onde a transformada cosseno discreta é utilizada para realizar a quantização e, assim, a reorganização das informações de brilho e cor dos pixels. A fórmula matemática utiliza os valores x e y para as coordenadas dos pixels no domínio do espaço e u e v para as coordenadas no domínio da frequência. Também nos foi apresentado um diagrama de bloco que contém diversos módulos de funções, bem como um módulo MAC e um módulo de memória RAM, onde está gravada uma look-up table com os resultados de uma função dependente de u e v. A equação utilizada é a seguinte:

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} [C(u) \cdot C(v) \cdot F(u,v) \cdot cos\frac{(2x+1)u\pi}{2N} \cdot cos\frac{(2y+1)v\pi}{2N}]$$

Com estas informações, nos foi passado o desafio de projetar uma máquina de estados, em um módulo chamado FSM_Control, com o objetivo de arquitetar todo o funcionamento do circuito e gerenciar os sinais de controle e as informações requeridas por cada etapa do cálculo.

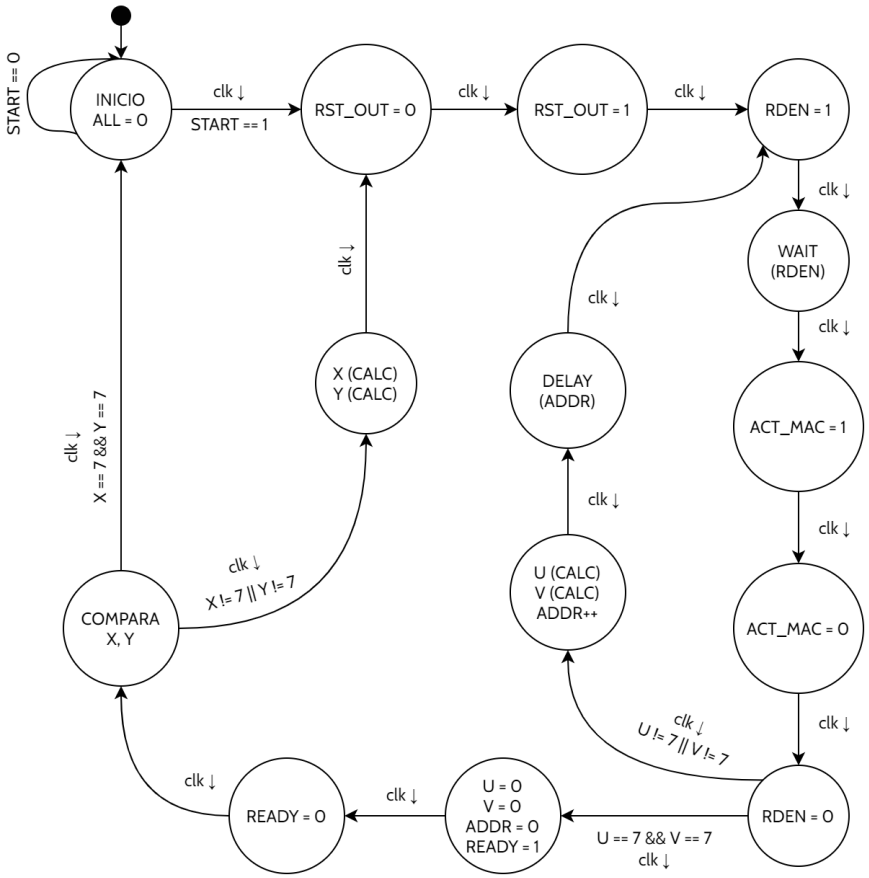

Figura 19 – Máquina de estados do módulo FSM_Control

3.1 Implementação

O módulo FSM_Control foi implementado como visto no diagrama da Figura 19. O ciclo se inicializa com um pulso de Start e termina após o cálculo ser encerrado, o que só acontece quando são efetuados todos os cálculos mediante excursionamento das variáveis v, u, x e y de 0 a 7.

O primeiro estado é um estado de inicialização onde as variáveis são setadas em 0. A máquina permanece nesse estado até que seja recebido um pulso de Start de nível lógico 1; no momento do recebimento desse pulso, o sinal de reset é ativado e desativado, e como o endereço inicial é 0, o pulso RDEN para leitura da memória é habilitado, é aguardado um pulso de clock e então o sinal de habilitação do acumulador MAC (ACT_MAC) é ativado e desativado, sendo o sinal RDEN desativado um pulso de clock após a desativação de ACT_MAC. Após isso, é verificado se u e v são iguais a 7. Caso não aconteça, u e v são excursionados e ADDRESS é incrementado para corresponder ao endereço de acesso da look-up table de $F(u, v)$. Caso aconteça de u e v serem iguais a 7, significa que foi terminado o cálculo de $f(x, y)$ para aqueles valores de x e y específicos, e então o pulso de READY é ativado por um ciclo, e u, v e ADDRESS são zerados. Após isso, é feita uma checagem dos valores de x e y. Caso x e y não sejam iguais à 7, os mesmos são excursionados, o sinal de reset é ativado e desativado e um novo ciclo de cálculo de $f(x, y)$ se inicia. Caso x e y sejam iguais à 7, todos os cálculos de $f(x, y)$ necessários já foram efetuados, então a FSM volta para o estado inicial, onde zera suas variáveis e entra em espera até um novo pulso de Start.

O código do módulo FSM_Control pode ser acessado no repositório do GitHub no caminho ENGG56/MPEG/FSM_Control.v.

3.2 Pin Assignments

Após a montagem e a implementação da FSM, foram feitas as conexões e o planejamento de pinos. Com relação as entradas, o clock foi ligado diretamente com o clock da placa de 50 MHz, a entrada de Reset foi atribuída a um botão KEY[0] e o sinal de Start foi atribuído ao primeiro switch da placa, o SW[0]. Já partindo para as saídas, os LED‘S verdes foram usados para os sinais de Ready, act_mac, Read_Enable e a saída de Reset, sendo utilizado o LEDG[0], LEDG[1], LEDG[2] e LEDG[3] respectivamente. As saída das variáveis x, y, u e v foram dividas em conjunto de 3 LED‘s vermelhos por serem saídas de 3 bits. Por fim, algo parecido foi feito com o endereço (a saída Address), onde foram reservados os LED‘S vermelhos de 12 a 17 para a saída que possui 6 bits.

O diagrama de conexões do módulo FSM_Control com os pinos físicos do FPGA pode ser visto na Figura 20.

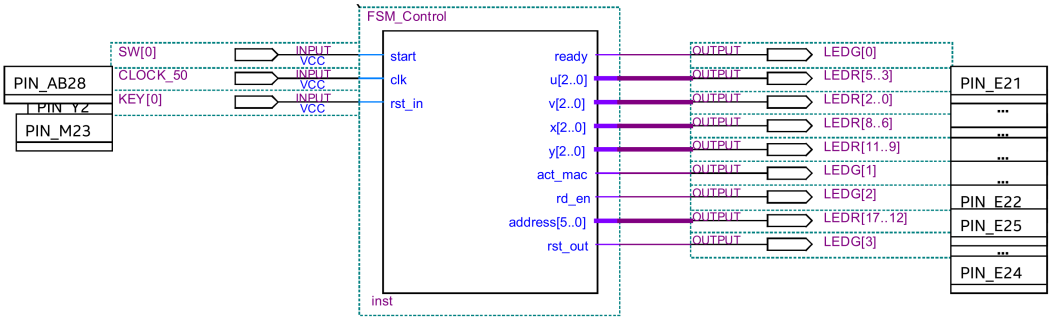

Figura 20 – Diagrama de conexões do módulo FSM_Control

3.3 Time Constraints

Novamente ocorreu de executar o Timing Analyzer e as configurações de projeto não atenderem às especificações de limite de clock e previsão de delay de input/output. O clock estava configurado com 1 GHz por padrão, e a análise resultou numa frequência máxima de operação de uma frequência na casa das centenas de MHz, novamente. Utilizamos a mesma estratégia da definição de Time Constraints do circuito Somador, configurando o clock do projeto para 50 MHz e o delay de propagação de input/output para 2 ns. Após a aplicação dos time constraints, re-executamos o Timing Analyzer e verificamos que todas as configurações de temporização atendiam ao necessário para garantir o correto funcionamento do circuito.

3.4 Testbench e resultados

O testbench do módulo FSM_Control foi desenvolvido de modo a gerar o pulso de Start, de Reset (Input) e também o sinal de clock. Durante a execução autônoma do FSM_Control após a aplicação do pulso de Start, são capturados os instantes de tempo das alterações dos sinais de controle advindos do módulo de modo a possibilitar a avaliação do intervalo de tempo entre alterações de estado de sinais criticamente interligados. Tais avaliações facilitam o diagnóstico de possíveis ocorrências de metaestabilidade e corrupção de informações. Caso seja detectada alguma condição de erro, uma mensagem correspondente é exibida no console e a simulação para. Caso a simulação corra sem erros durante toda sua execução, uma mensagem de êxito é exibida no console. As condições de erro avaliadas são as seguintes:

- Intervalo de tempo entre alterações de RDEN e ACT_MAC menor que um ciclo de clock;
- Excursionamento incorreto das variáveis u, v, x, y e ADDRESS;
- Pulsos de Ready/Reset Out disparados incorretamente.

Nas Figuras de 21 a 24 é possível visualizar a execução da FSM e o excursionamento das variáveis. Na Figura 25 é visualizada a saída de console decorrente da execução do testbench. O código do testbench do módulo FSM_Control pode ser acessado no repositório do GitHub no caminho ENGG56/MPEG/FSM_Control_TB.v.

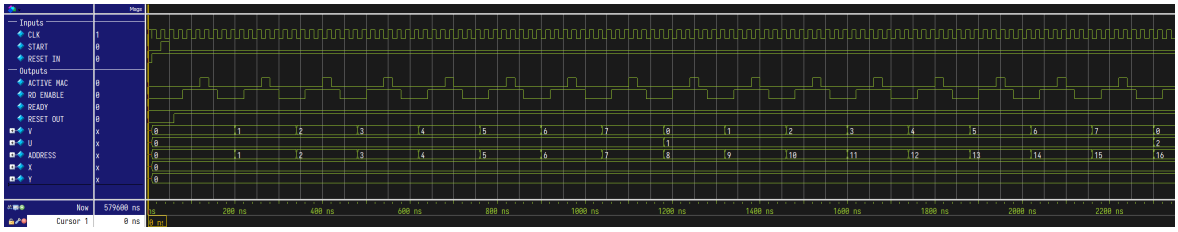

Figura 21 – Waveform com zoom aplicado (variação de u, v e *address*)

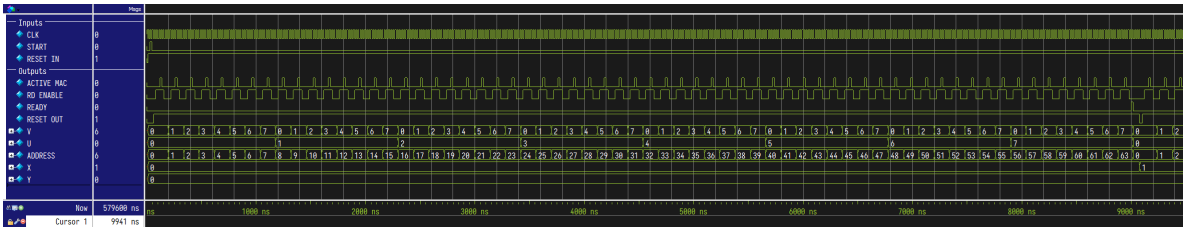

Figura 22 – Waveform - um ciclo de cálculo de $f(x, y)$ completo

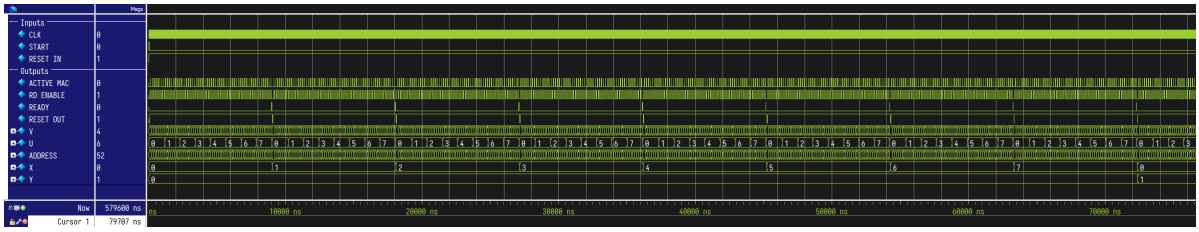

Figura 23 – Waveform - Excursionamento de x de 0 a 7

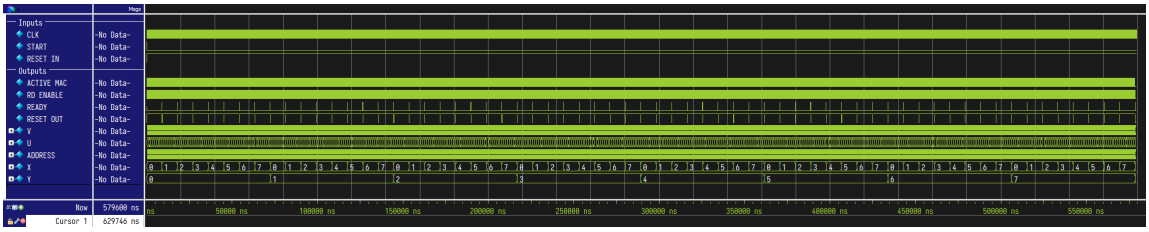

Figura 24 – Waveform - Excursionamento de y de 0 a 7

```
VSIM 32> run -all
# Pulso de Start                  30
# Simulacao terminada com sucesso no tempo              579600
# ** Note: $stop    : E:/ENGG56/MPEG/FSM_Control_TB.v(170)
#    Time: 579600 ns  Iteration: 0  Instance: /FSM_Control_TB
# Break in Module FSM_Control_TB at E:/ENGG56/MPEG/FSM_Control_TB.v line 170
```

Figura 25 – Mensagens no console decorrentes da execução do testbench de FSM_Control

4 Semáforo

O problema do semáforo apresenta um desafio de controle de tráfego através de sinaleiras em um cruzamento. Nesse cruzamento consta uma via principal, no qual chamamos de L_O, e uma via secundária, na qual chamamos de N_S. A questão apresenta diversas proposições com as lógicas necessárias para a implementação do projeto.

4.1 Implementação

A partir das condições estabelecidas no enunciado, montamos a tabela-verdade (Tabela 1).

Na tabela chamamos de A, B, C e D as vias correspondentes, de N_S o semáforo das vias secundárias e de L_O o semáforo da via principal. O valor 1 nas vias corresponde a presença de carro na mesmas, enquanto o valor 1 presente nos semáforos corresponde ao seu estado em luz verde.

Com a construção da tabela, foi possível chegar na figura 26, onde montamos uma tabela de Karnaugh para a construção das equações booleanas das saídas.

A implementação em Verilog foi bem simples; bastou apenas declarar o módulo e fazer dois assigns, um para a saída L_O e outro para a saída N_S.

A	B	C	D	N_S	L_O
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	0	1

Tabela 1 – Tabela-verdade do semáforo

$$N\_S = (A + B).\overline{C}.\overline{D}$$

$$L\_O = \overline{N\_S}$$

L_O				
CD\AB	00	01	11	10
00	1	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

N_S				
CD\AB	00	01	11	10
00	0	1	1	1
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

Figura 26 – Mapas de Karnaugh de L_S e N_O

4.2 Pin Assignments

Após formada a lógica com as tabelas e equações acima, começamos a implementação do módulo no Quartus, onde não foi necessário executar *Time Analysis* nem definir *time constraints*, uma vez que toda a mudança de estados é feita de forma assíncrona, não necessitando de um sinal

de clock de entrada. É possível encontrar o projeto e o respectivo código no repositório através do caminho ENGG56/Semaforo/Semaforo.v.

O diagrama de blocos final da questão pode ser conferida na figura 27. Nela é possível perceber as entradas A, B, C e D e as saídas N_S e L_O, cada uma com um bit, e seus respectivos pinos com 4 switches para as entradas e 2 LED's para as saídas.

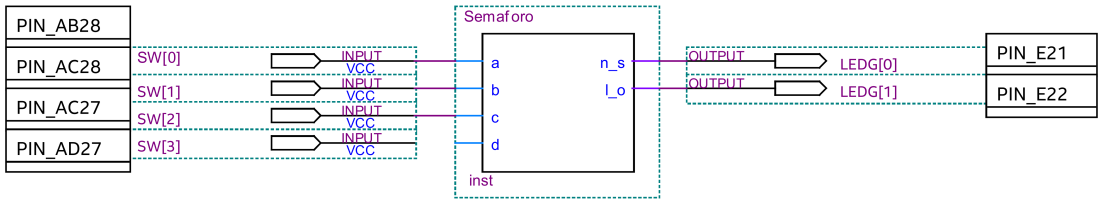

Figura 27 – Diagrama de conexões do módulo do semáforo

4.3 Testbenches e resultados

Por fim, foi realizado um testbench, onde foram testadas todas as 16 possibilidades de entradas, de modo a capturar as saídas do circuito. Ao comparar com as saídas esperadas, de acordo com a tabela-verdade definida na tabela 1, é possível perceber que o circuito responde como o esperado. Caso o circuito demonstrasse alguma resposta incorreta em alguma das suas saídas, o testbench exibiria uma mensagem de erro no console e pararia a execução.

A forma de onda decorrente da execução do testbench pode ser visualizada na figura 28, onde a borda baixa significa o valor lógico 0 e a borda alta significa o valor lógico 1 das entradas e saídas. Além disso, é possível ver a saída de console decorrente da execução na figura 29.

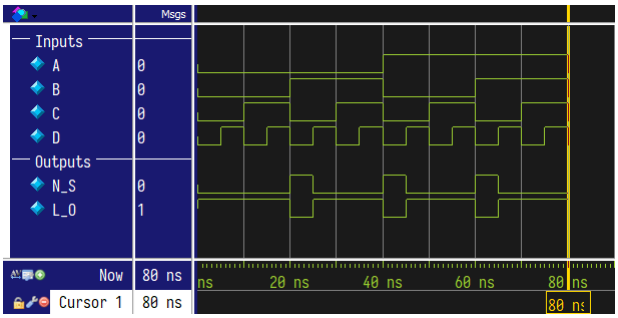

Figura 28 – Waveform da testbench do semáforo

```
VSIM 14> run -all
# Simulacao terminada com exito.
# ** Note: $stop    : E:/ENGG56/Semaforo/Semaforo_TB.v(28)
#    Time: 80 ns  Iteration: 1  Instance: /Semaforo_TB
# Break in Module Semaforo_TB at E:/ENGG56/Semaforo/Semaforo_TB.v line 28
```

Figura 29 – Saída de console do testbench do semáforo

Conclusão e considerações finais

Com o esforço dos integrantes do grupo e ajuda do Professor com as dúvidas que surgiram, conseguimos implementar de forma satisfatória os quatro projetos requisitados. Durante o caminho, as maiores dúvidas ficaram por conta da implementação do Time Analyzer em alguns casos, mas que foi facilmente sanada através do estudo e construção dos problemas.

Por meio das simulações e projetos fica mais evidente a importância da análise e validação de projetos em Verilog de modo a assegurar o correto funcionamento dos projetos e a prevenção de ocorrências de metaestabilidade ou corrupção de dados.

No decorrer do desenvolvimento enfrentamos alguns desafios nas implementações, como por exemplo a geração indesejada de latches em decorrência do modo de escrita do código, e também desafios no estabelecimento de uma consistência razoável e autonomia dos testbenches. Utilizamos dos recursos de captura de tempo de simulação e display em console para verificação de intervalo de tempo entre variação dos sinais de controle e para a criação de logs que apontariam no console os possíveis erros encontrados no DUV durante a simulação. Esses recursos nos possibilitaram notar inconsistências nos módulos, de modo a facilitar a identificação de pontos de falha na implementação e facilitar o processo de diagnóstico para correções.

Referências

[1] Intel® Quartus® Prime Download - Intel® Quartus® Prime Software", Intel®, 2021. [Online]. Available: https://www.intel.com.br/. [Accessed: 08-Nov-2021].

[2] DE2-115 User Manual, Altera. Available: https://www.terasic.com.tw/. [Accessed: 08-Nov-2021].

[3] ModelSim* - Intel® FPGA Edition Simulation Quick-Start. Intel® Quartus® Prime Standard Edition. Intel®, 2019.

[4] Intel® Quartus® Prime Standard Edition User Guide: Getting Started. Intel®, 2019. Available: https://www.intel.com/content/www/us/en/programmable/documentation/. [Accessed: 08-Nov-2021].

[5] Intel® Quartus® Prime Standard Edition User Guide: Timing Analyzer. Intel®, 2019. Available: https://www.intel.com/content/www/us/en/programmable/documentation/ [Accessed: 08-Nov-2021].

[6] Intel® Quartus® Prime Standard Edition User Guide: Design Constraints. Intel®, 2019. Available: https://www.intel.com/content/www/us/en/programmable/documentation/ [Accessed: 08-Nov-2021].