2º Projeto:
Scheduling and Resource Sharing

Projeto de Sistemas Digitais
Docente: Paulo Flores

Afonso Rodrigues Nº 67528
Afonso Pereira Nº 78949
Miguel Rodrigues Nº 76176

08/11/2017

1 Introdução

Neste segundo projeto de laboratório são explorados os conceitos de gestão de recursos e calendarização, bem como o uso de memórias para receber dados e onde escrever dados.

Com o conhecimento adquirido no laboratório anterior em relação ao funcionamento do software Vivado e à linguagem de descrição de hardware VHDL, é criado um programa que resolva uma iteração de um algoritmo (Método do Gradiente) para encontrar um mínimo local de uma equação polinomial de segundo grau. Os valores de cada variável (A, B, C, Xi e Step) estão presentes em memórias em posições consecutivas de memória, com 10 bits, as quais são acedidas sequencialmente de modo a realizar a primeira iteração de quatro conjuntos diferentes de dados. Cada iteração do algoritmo dá origem a três resultados a guardar (dy, Xn e y), originando um total de 12 valores de 40 bits a guardar. Estes são armazenados em posições sequenciais numa única memória de saída.

Considerando restrições de hardware, este projeto leva a que seja feita uma gestão e otimização dos recursos a utilizar que envolve calendarizar as operações a serem feitas, dando uso a diferentes recursos, como o *ASAP Scheduling* ou a lista de prioridades.

2 Scheduling

Considerando apenas uma iteração do algoritmo completo, e admitindo que cada operação tem a duração de um ciclo de relógio, são obtidos os seguintes *data flows* e lista de prioridades:

2.1 ASAP Scheduling

As Soon As Possible Scheduling: Assim que seja possível realizar uma operação, esta é realizada, mesmo que o seu resultado só seja necessário muito tempo depois.

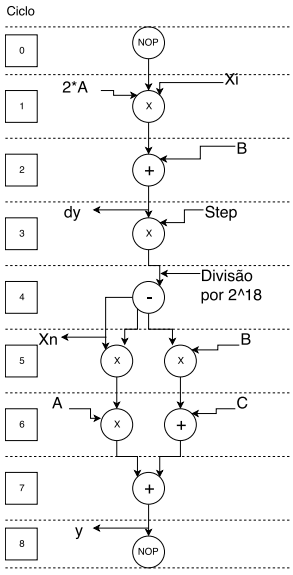


Figura 1: ASAP Scheduling

Nota: Tanto a multiplicação por 2 como a divisão por 2^{18} não requerem o uso de recursos adicionais, visto que são ambos múltiplos de 2, constituem apenas *shifts* à esquerda (multiplicação) ou à direita (divisão).

2.2 ALAP Scheduling

As Late As Possible Scheduling: As operações são realizadas o mais tarde possível sem afetar o número total de ciclos de relógio (em comparação com o caso anterior), isto é, cada operação só se realiza no ciclo anterior a ser necessário o seu resultado.

Neste projeto, este diagrama não vai ser diferente do presente na secção anterior, pois nenhuma operação pode ser realizada mais tarde sem atrasar toda a iteração (e nenhuma poderia ter sido feita mais cedo na secção anterior). Assim, o *scheduling* escolhido é o único possível nesta situação.

2.3 List Scheduling

A lista de prioridades usando o caminho crítico como métrica é a apresentada na Tabela 1. Esta lista tem em conta que os recursos utilizados são os especificados: uma *Arithmetic and Logic Unit* (com a sigla ALU na lista) e dois multiplicadores (com o símbolo X na lista).

Tabela 1: List Scheduling

Nr	Operação	Prioridade
1	X	7
2	ALU	6
3	X	5
4	ALU	4
5	X	3
6	X	3
7	X	2
8	ALU	2
9	ALU	1

3 Circuito VHDL

3.1 Unidade de Controlo

Neste circuito a máquina de estados finita usada é bastante simples, sendo apenas um conjunto de estados sequencial que nunca muda, visto que se quer realizar sempre o mesmo conjunto de operações (em cada iteração do algoritmo), alterando apenas as posições de memória onde são lidos os valores iniciais e a posição de memória onde são escritas as respostas. Assim, a máquina de estados, que é uma máquina de Moore, visto que os *outputs* dependem apenas do estado em que a máquina se encontra é a representada na Figura 2. Não se econtram representadas as transições por *reset*, o sinal que leva sempre ao estado inicial e retorna todos os valores guardados a zero.

Cada estado transmite ao datapath um diferente conjunto de operandos, *oper*, que vai decidir que operações são feitas e que valores são guardados. Com dois contadores, a unidade de controlo transmite, também, os valores do endereços de memória onde ler e onde escrever. O contador para a posição de memória de leitura é incrementado sempre que se está no estado *s_eight*, para que seja lido um novo conjunto de valores e é transmitido ao datapath, para que esta unidade leia os valores certos na realização das operações. Quanto à posição de memória de escrita, o contador é incrementado três vezes por iteração, nos estados *s_three*, *s_five* e *s_eight*, sendo que o sinal que permite a escrita na memória de saída passa a '1' nos estados imediatamente anteriores a estes (*s_two*, *s_four* e *s_seven*) e o sinal de *done*, que indica o final da iteração, passa a '1' no último estado da iteração, *s_eight*.

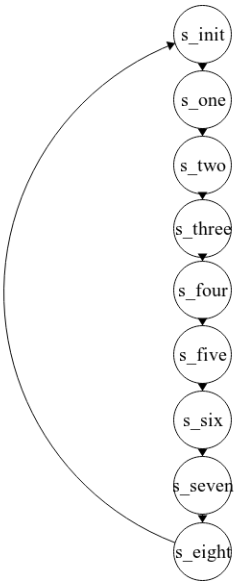


Figura 2: Máquina de Estados

3.2 Datapath

Nesta unidade são feitas as operações necessárias para realizar o algoritmo. Os valores obtidos em cada operação são guardados em respetivos registos intermédios (R1 para o multiplicador 1, R2 para o multiplicador 2 e R3 para a ALU), de forma a serem usados na operação seguinte. Como existem limitações no que toca a hardware de operações, são usados multiplexers para escolher as entradas do multiplicador 1 e da ALU, sendo que no caso do multiplicador 2 não é necessário, é feita apenas uma operação neste. Os valores iniciais de A, B, C, Xi e Step são lidos diretamente da respetiva memória.

Com recurso aos sinais recebidos da unidade de controlo (*oper* e *cnt_big*), são decididas que operações a realizar, quais as entradas a usar nos multiplicadores e ALU e quais valores a guardar nos registos, bem como a posição de memória a ler. Os valores a serem guardados na memória de resposta originam nesta unidade, mas só serão escritos em função do sinal de enable da unidade de controlo e da posição de memória dada pelo respetivo contador também na unidade de controlo.

3.3 Diagrama de Blocos, Recursos e Especificações

O diagrama de blocos presente na Figura 3 descreve visualmente como funciona o datapath, recebendo os dados das memórias, o clk e o rst do circuito de topo e o oper do circuito de controlo. No entanto, recebe também outros sinais, como o valor do contador que vai decidir o endereço de memória onde ler.

A área total deste circuito é constituída por 280 LUTs e 109 FFs. Quanto ao número de multiplexers, existem oito multiplexers a considerar, sendo que os restantes não são relevantes e são usados três registos no datapath, mas o número poderia ser reduzido para dois, pois a cada ciclo são gravados nos registos, no máximo, dois valores. Assim, olhando para a Figura 1, as operações iniciais e as restantes do lado esquerdo poderiam estar guardadas em R1 e as duas do lado direito no registo R2, reduzindo o número de registos. No entanto, ter-se-ia de recorrer a diferentes multiplexers dos implementados.

De acordo com os relatórios disponíveis após a síntese e implementação, com um período de clock de 11ns obtém-se um *Total Negative Slack* de 0ns e um *Worst Negative Slack* de 0.428ns, indicando um período adequado. No entanto, para que a *Post-Implementation Timing Simulation* seja feita de forma correta, é necessário que o período a usar no *testbench* seja de 15.0ns, originando uma frequência máxima de 66,(6) MHz.

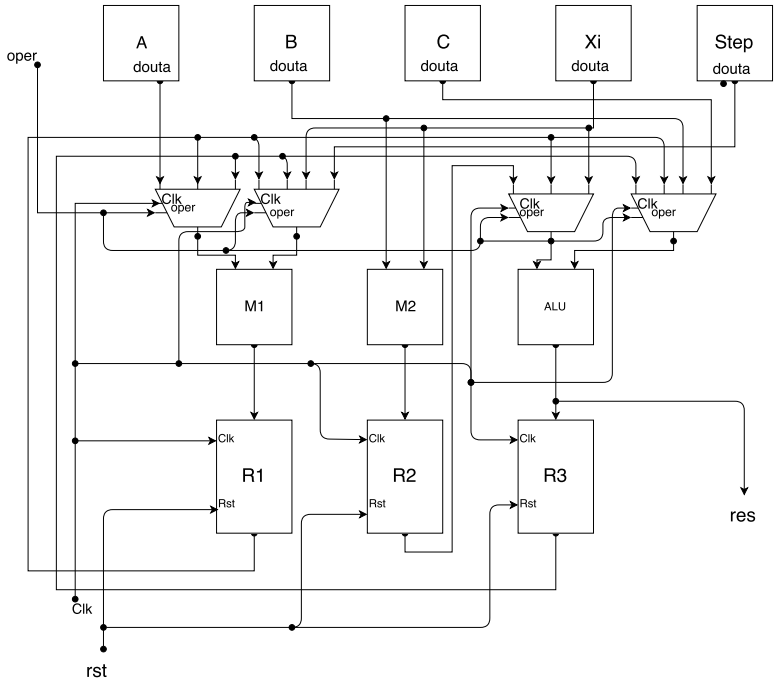


Figura 3: Diagrama de Blocos

Nesta simulação é possível obter a latência do circuito, isto é, quanto tempo demora a iteração toda a ocorrer. Olhando para a Figura 4, feita com os dois primeiros conjuntos de valores disponibilizados e com os dois últimos e cujos resultados se apresentaram corretos, com recurso a cursores no momento em que é feito o reset e no momento em que o valor de y é escrito, infere-se que a latência é dada por, aproximadamente, 262.046 - 158.763 = 102.283ns.

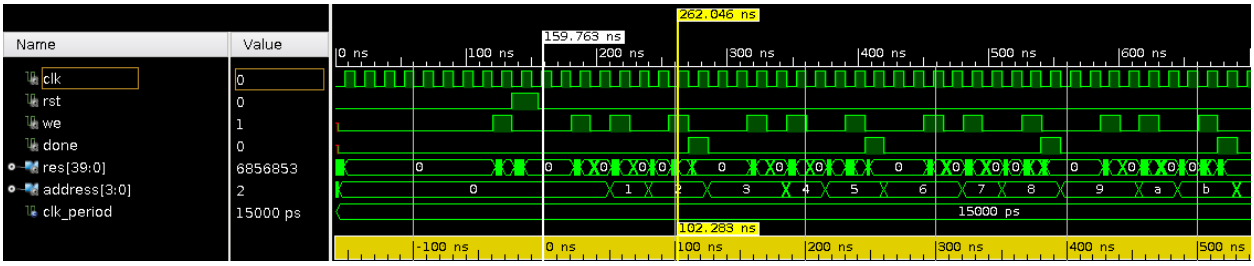


Figura 4: Simulação

4 Pipelining

Para melhorar a performance do circuito, é possível usar uma arquitetura de *pipeline*, começando operações da iteração seguinte sem que a iteração atual esteja concluída. Sem utilizar recursos adicionais e admitindo que são usados apenas dois registos como referido na secção anterior, seria possível, apenas, encurtar um ciclo às restantes iterações após a primeira, pois como visto na Tabela 1, é necessário o input de A até ao sexto ciclo, portanto a iteração seguinte só poderia começar no sétimo ciclo da iteração atual, onde se iria buscar o valor de A seguinte.

No entanto, usando recursos adicionais é possível melhorar a performance do circuito, sendo que no máximo estas quatro iterações (inicialmente feitas em 4 * 7 = 28 ciclos,segundo a Figura 1) seriam feitas em 10 ciclos. A partir do momento em que uma iteração passa para a segunda operação, a iteração seguinte começa, funcionando como uma linha de montagem. Para que tal ocorra, é necessário que se usem quatro multiplicadores (em vez de dois), pois simultaneamente podem estar em funcionamento, para diferentes iterações, os ciclos 3, 4, 5 e 6 e três ALUs, para os ciclos 4, 5, 6 e 7 ao mesmo tempo. É necessário, ainda, que existam em número superior registos intermédios para guardar resultados e valores lidos da memória (que são diferentes para cada iteração) e multiplexers para realizar as escolhas a fazer nas operações.

Um rascunho do projeto utilizando pipeline está representado no Anexo 1.

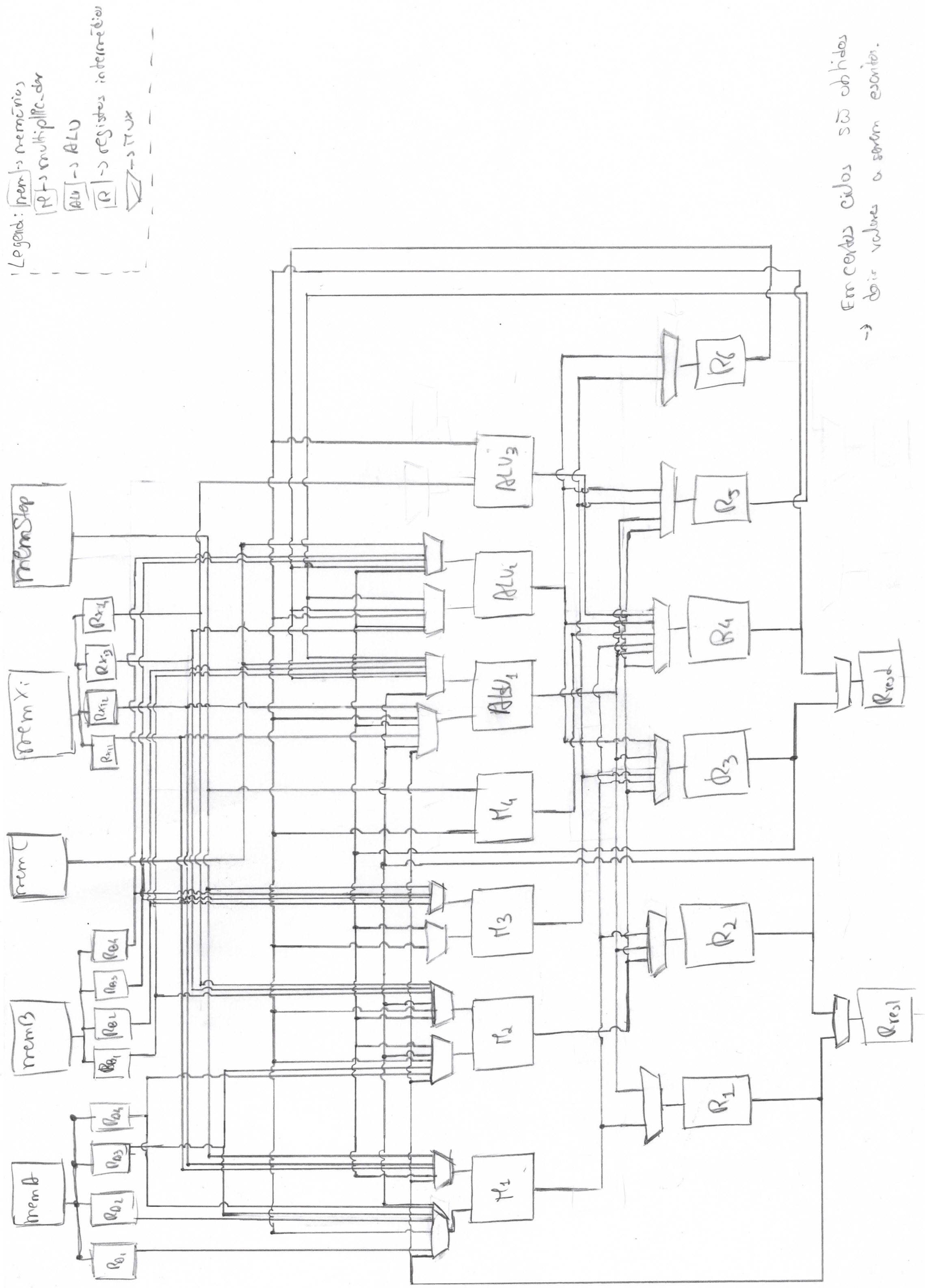
5 Conclusão

Com a realização deste trabalho laboratorial analisámos e aplicámos os conceitos de gestão de recursos, calendarização e listas de prioridades, e uso de memórias para ler e escrever dados. Deste modo, contribuiu para o aprofundamento da escrita, desenho e descrição de algoritmos em VHDL e da sua aplicação em sistemas mais complexos devido às restrições de hardware apresentadas.

Apresentando os resultados relativamente à calendarização, utilizámos os conceitos dos tipos ASAP e ALAP para desenhar os grafos correspondentes a um ciclo iteração do algoritmo, que verificámos serem iguais. No desenho e desenvolvimento do circuito em VHDL que implementa o algoritmo em estudo, cumprimos todos os requisitos e restrições de modo a obter um circuito com a melhor performance possível, como foi pedido, usando uma arquitetura FSMD. Demonstrando através de simulações, conseguimos uma frequência máxima de 66,(6) MHz e uma latência de 8,85 ns, utilizando 8 multiplexers e 3 registos, um clock de 11 ns na simulação de pré implementação e 15 ns na simulação de pós implementação. Na otimização do circuito usando pipelining, descrevemos que esta é possível com e sem recursos adicionais, sendo que sem recursos seria possível encurtar 1 ciclo por iteração e com recursos adicionais seria possível encurtar 18 ciclos no total.

Concluímos assim que os objetivos deste trabalho foram alcançados com sucesso, resolvendo todos os problemas propostos e abordando as temáticas e conceitos envolvidos.

Anexo 1


memA
memB
memC
memXi
memStep
M1
M2
M3
M4
ALU1
ALU2
ALU3
R1
R2
R3
R4
R5
R6
Rres1
Rres2