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 optimização 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.

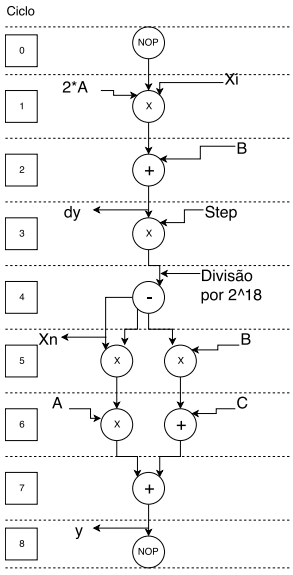

Figura 1: ASAP Scheduling

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 totald e 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).

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

FSM, recursos, diagrama de estados talvez

3.2 Datapath

descrição datapath, o que faz, como faz, que recursos usa etc

3.3 Diagrama de Blocos, Recursos e Especificações

O diagrama de blocos presente na Figura 2 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 dos contadores que vão decidir o endereço de memória onde escrever e o endereço 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 gardadas 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.

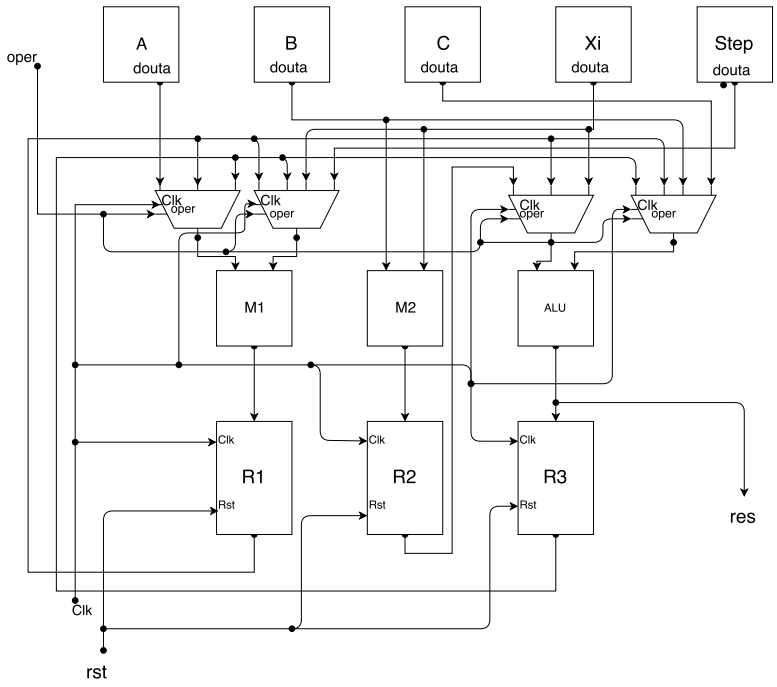

Figura 2: Diagrama de Blocos

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. Nesta simulação é possível obter a latência do circuito, isto é, quanto tempo demora a iteração toda a ocorrer. Olhando para a Figura 3, 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 se torna realmente zero, infere-se que a latência é dada por 144.040 - 135.190 = 8.85ns.

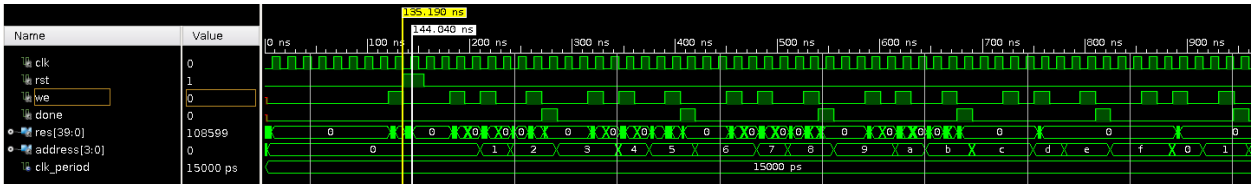

Figura 3: Simulação

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.

5 Conclusão