TÉCNICO LISBOA

Projeto de Sistemas Digitais

MEEC

Architectural Synthesis

Autores:

João Marques Vinagre (93095) marquesvinagre@tecnico.ulisboa.pt
André Alexandre Costa Santos (96152) andresantos1.alf@gmail.com
Diogo Batista Araújo (96180) diogo.batista.araujo@tecnico.ulisboa.pt

Grupo 2

2022/2023 – 1º Semestre, P1

Conteúdo

1 Introdução

Para o segundo projeto da unidade curricular de PSD foi proposto o desenvolvimento de um sistema com o objetivo de calcular o resultado de uma equação simples com multiplicações, somas e uma subtração. Foi necessário analisar e sintetizar todo o processo de como a operação seria realizada em hardware, otimizando os recursos disponíveis de modo a despender o mínimo possível destes e maximizando a eficiência temporal. A equação a resolver é a seguinte.

$$\overline{y} = \frac{\sum_{i=1}^{N} m \cdot x_i + b}{N} \qquad onde \qquad \overline{y}, m, x_i, b \in \mathbb{C}$$

2 Processo de design

2.1 Organização das operações

O pseudocódigo para solução desta equação é o seguinte:

```
YTr = YTi = 0;
for(i=1; i < N; i++){
    Yr = Mr*Xr[i] - Mi*Xi[i] + Br;
    Yi = Mr*Xi[i] + Mi*Xr[i] + Bi;
    YTr = YTr + Yr;
    YTi = YTi + Yi ;
}
YAr = YTr / N;
YAi = YTi / N;
```

Com o fim de melhorar o desempenho, pensou-se em alterar a parte interior do ciclo da seguinte forma com o objetivo de reaproveitar os resultados $Mr \cdot (Xr[i] + Xi[i])$:

```
YTr = YTi = 0;
for(i=1; i < N; i++){
    Yr = Mr*(Xr[i] + Xi[i]) - Xi[i](Mr + Mi) + Br;
    Yi = Mr*(Xr[i] + Xi[i]) + Xr[i](Mi - Mr) + Bi;
    YTr = YTr + Yr;
    YTi = YTi + Yi ;
}
YAr = YTr / N;
YAi = YTi / N;
```

No entanto verificou-se que esta alteração não traria qualquer beneficio, tendo em conta os recursos de hardware disponiveis. Seria necessário o mesmo número de ciclos de relógio para realizar todas as operações, mesmo que neste caso o número destas seja mais pequeno.

Ponderou-se também efetuar a soma do valor B, respetivamente ao total real e imaginário, fora do loop, de modo a diminuir as operações dentro deste. Para tal, multiplicar-se-iam

os valores de B por N, somando de seguida ao valor de YT obtido do ciclo. Este processo acrescentaria quatro operações após o loop. Chegou-se à conclusão de que a velocidade do loop não seria alterada, dado que seria necessário o mesmo número de ciclos de relógio para as restantes operações.

2.2 Data Flow Graph

O *Data Flow Graph* obtido contém apenas uma pequena alteração em relação ao pseudo-código apresentado anteriormente, sendo esta a soma de B ao acumulador em vez de ao total da equação de forma a acomodar as operações em menos ciclos de relógio e maximizar a utilização de recursos.

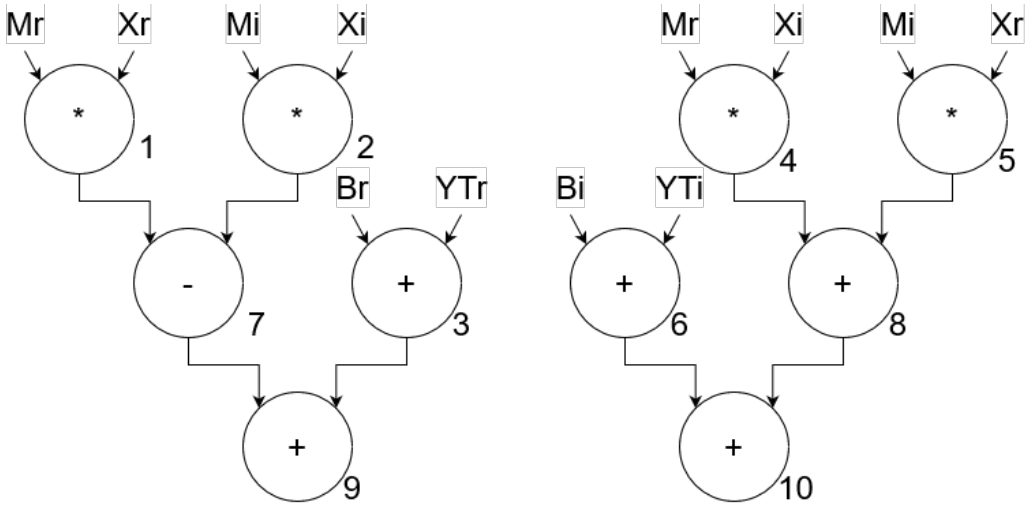

Figura 1: Data Flow Graph

2.3 List scheduling

Para definir a lista de prioridades das operações foi usado o método de caminho crítico. Com recurso ao *Data Flow Graph* é fácil perceber as dependências das operações e rapidamente se chegou à seguinte lista:

Operação	Dependências	Prioridade
1 (x)		3
2 (x)		3
3 (+)		2
4 (x)		3
5 (x)		3
6 (+)		2
7 (-)	(1,2)	2
8 (+)	(4,5)	2
9 (+)	(7,3)	1
10 (+)	(8,6)	1

Com a lista de prioridades obtida, e tendo 2 multiplicadores e 2 ALUs, obtém-se o seguinde escalonamento temporal:

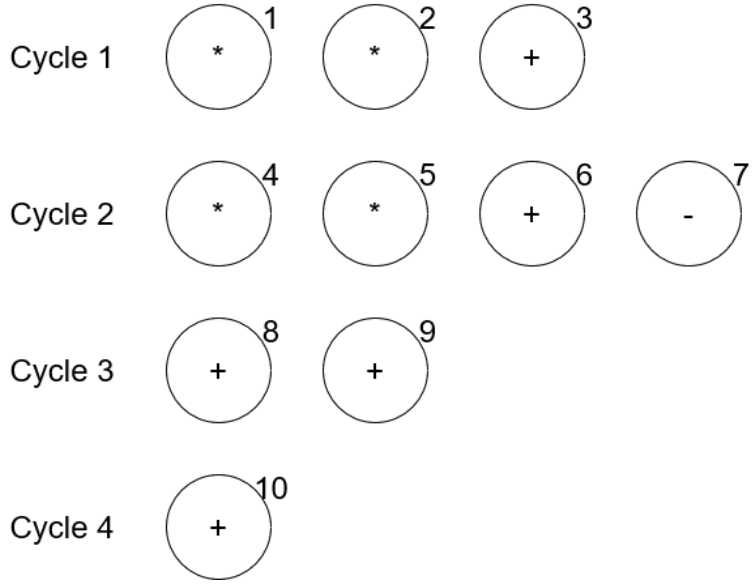

Figura 2: Escalonamento temporal

2.4 Datapath

Depois de ter o escalonamento temporal definido, foi possível desenhar o datapath. Este foi desenhado com o objetivo de minimizar o número de sinais, registos e multiplexers necessários, desta forma facilitando o design da máquina de estados.

De modo a minimizar o hardware necessário, pensou-se em ligar diretamente cada unidade de cálculo a um registo. Deste modo é possível evitar o uso de *multiplexers* nas saídas quer dos multiplicadores, quer das ALUs, apesar de estes serem necessários nas entradas. Tendo esta escolha em conta, foi necessário analisar quais as operações que seriam realizadas em cada unidade de cálculo, de modo a determinar a melhor maneira de ligar as entradas a estas. Foi determinado que, dado que as operações de multiplicação são feitas a pares, estas seriam guardadas cada uma no seu registo. Utilizam-se assim dois multiplicadores e dois registos. Por sua vez, estes resultados vão ser somados ou subtraidos, logo é utilizada uma ALU e por sua vez um registo para guardar o total desta operação. A ultima ALU tem como objetivo realizar todas as operações referentes aos valores totais (YTr *e* YTi), então à entrada apresenta os acumuladores onde estes valores são guardados, e todos os valores necessários para as restantes operações. Foram também necessários dois multiplexers, cada um à entrada do respetivo multiplicador, de modo a selecionar qual dos valores de X (Xr *ou* Xi) seria multiplicado.

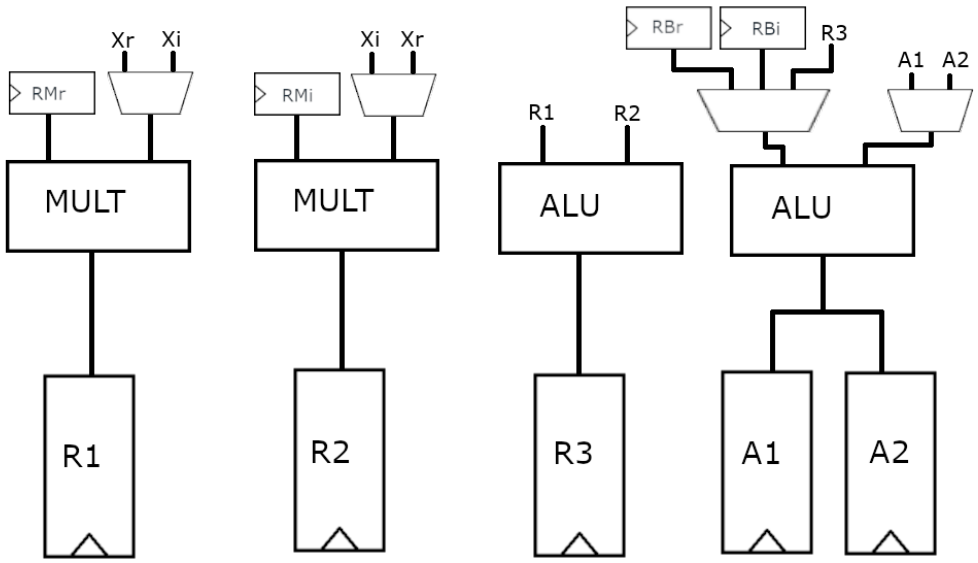

Figura 3: Datapath

2.5 Finite State Machine (FSM)

Com o conhecimento de que sinais são usados para controlar o datapath, foi possível desenhar a FSM. O primeiro estado, *Init*, serve para inicializar todos os registos a 0 com o sinal de *reset*. De seguida, o estado *Read* seleciona o set de valores usados como entrada na equação. Os valores são lidos da memória no estado de *Load* com o auxílio do contador acionado no estado *count1*. Como o escalonamento temporal mostra que são necessários 4 ciclos de relógio para completar uma iteração do algoritmo, foram adicionados 4 estados correspondentes. Para controlar o número de iterações verificando quando está disponível o resultado, é usado um estado, *count1*, em que é acionado um contador e o valor do mesmo comparado com N, permitindo passar para o estado *write* caso todos os cálculos estejas concluidos. No estado *write* é feita a escrita do resultado na memória de saida em dois endereços correspondentes à parte real e imaginária do resultado. Para ajudar a organização da escrita é usado um contador que define quantas escritas foram feitas e a parte menos significativa do endereço de memoria a usar. Finalmente a máquina para no estade *done* que liga o sinal que indica o fim da tarefa.

Na imagem seguinte são indicados os valores dos sinais gerados em cada estado bem como as condições de trasição entre os mesmos.

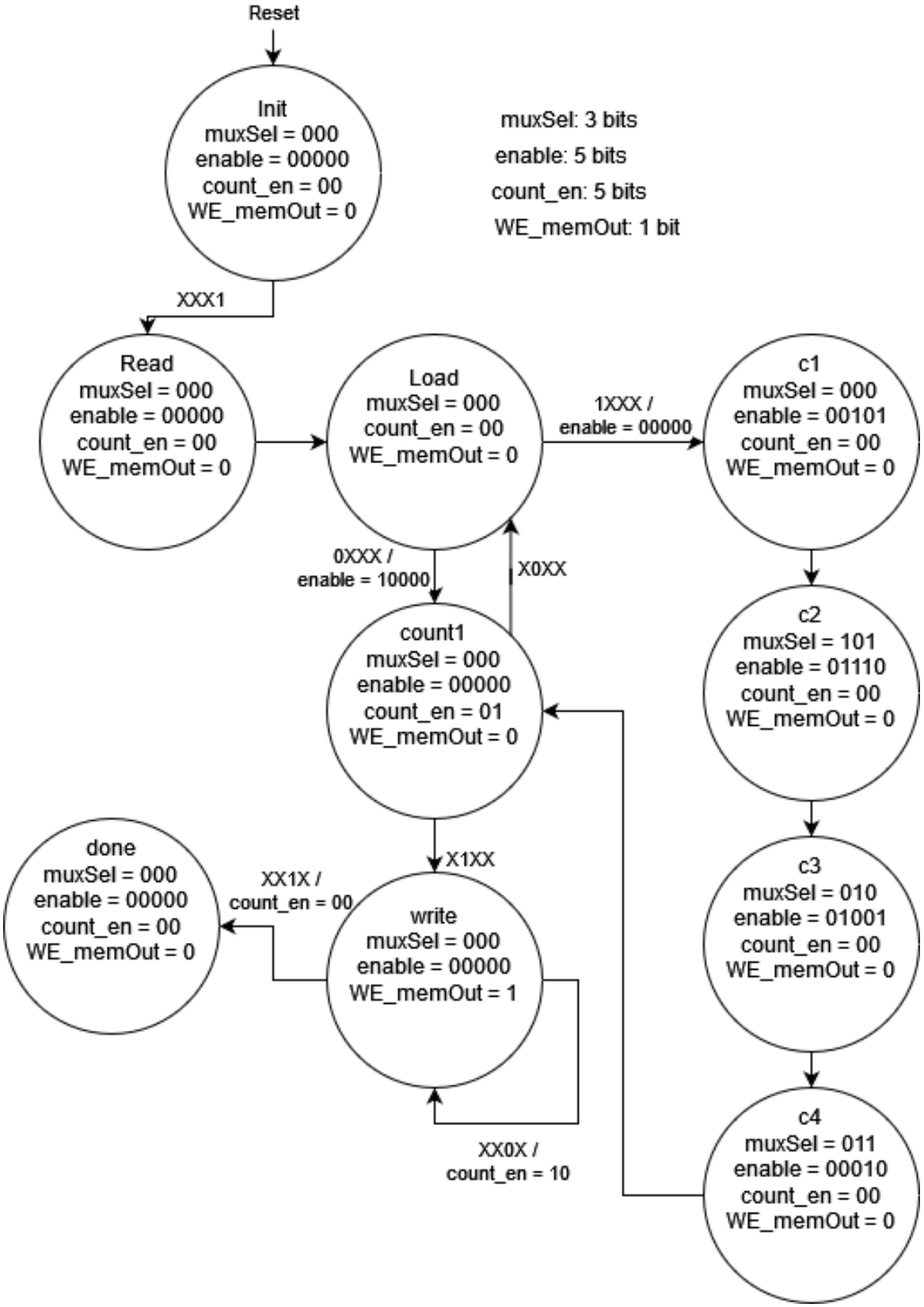

Figura 4: Finite State Machine

3 Testes realizados

3.1 Simulação

Para a verificação do funcionamento da lógica na máquina de estados foi criado o ficheiro *circuito_tb.vhd* que escolhia o endereço de memória a aceder e dava início à simulação. Utilizando as funcionalidades do Vivado, é possível observar os valores gerados pelo circuito ao longo do tempo com os estímulos definidos no *circuito_tb.vhd*. Com esta ferramenta é possivel visualizar os sinais em todo o circuito, sendo assim possível encontrar erros e desafios que foram posteriormente corrigidos.

Um dos desafios encontrados foi a existência de overflow na soma ao valor total. Inicialmente, foram calculados os bits necessários para todas as operações mas não foi tida em conta a possibilidade de, após várias somas, o valor superar o máximo estabelecido. Foi necessário então calcular de novo os bits necessários, desta vez tendo em conta a possível existência de overflow nas operações. Na entrada dos multiplicadores estão duas palavras: uma de 10 bits e outra de 8 bits. Para evitar a perda de informação do resultado desta operação, são necessários à saida 18 bits correspondendo à soma do número de bits de entrada. Estes resultados são utilizados posteriormente numa soma, neste caso, para evitar o *overflow* são necessários N+1 bits de saida em que N é o número de bits nas entradas da ALU. Como o caso anterior acontece duas vezes numa iteração o número de bits necessários é 20. Como o número de iterações é 8 e, sabendo que o número extra de bits que evitam o *overflow* é $\log_2(8) = 3$, o número final de bits é 23. Os restantes erros encontrados foram apenas referentes a pequenos erros no código, nomeadamente na utilização do muxSel, dado que este foi optimizado para utilizar o menos número de sinais possível.

3.2 Síntese

Para preparar os testes na placa foi feita a síntese do círcuito e a implementação da arquitetura. Os recursos da placa utilizados, segundo a ferramenta Vivado, estão dispostos na seguinte tabela:

Recurso	Utilização	Utilização(%)
LUT	331	1.59%
FF	139	0,33%
DPS	0	0%
BUFG	1	3,3%

A utilização de recursos descrita pela ferramenta não corresponde ao esperado, pois, apesar da utilização de mutliplicadores e somadores, esta não aparece discriminada pelo Vivado. Por outro lado, ao abrir o esquema gerado, estes recursos aparecem como utilizados no datapath projetado. Relativamente a esta incongruência foi feita uma investigação do que podesse ser a causa, porém esta não obteve sucesso. Suspeita-se que os recursos apresentados correspondam apenas aos utilizados na FSM, dado que o número de multiplexers que a implementação em Vivado discrimina corresponde ao utilizado nesta.

Com um clock de período 10ns, obtivémos um *"Worst Negative Slack"* de 0,63ns. Este não permitiria encurtar o período com o objetivo de tornar a máquina mais responsiva.

4 Pipelining

É possível melhorar a *performance* do circuito utilizando uma arquitetura em pipeline, apesar desta ser consideràvelmente mais dispendiosa em termos de recursos de hardware. Para o caso desenvolvido, com um valor de *N=8*, este compromisso não é justificável, mas para uma situação em que *N=1024*, é algo que deve ser considerado. De modo a obter um *throughput* de um resultado por ciclo, desenvolveu-se a seguinte arquitetura de datapath:

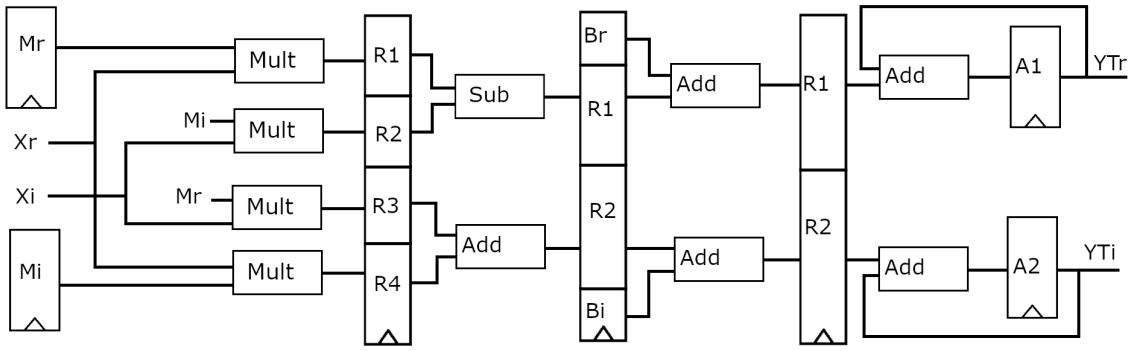

Figura 5: Datapaht de pipeline de 4 estágios

Como é evidente, o custo de hardware aumenta bastante: são necessários agora quatro multiplicadores, para além de 6 ALUs. O número de registos aumenta também. Dado que o período de relógio deve ser o mesmo do projeto desenvolvido, não é possível alterar o número de operações entre registos.

5 Conclusões

Concluindo, ao longo do desenvolvimento deste trabalho, foi possível desenvolver competências de optimização quer do datapath quer da *state machine*, e aprofundar conhecimentos e conceitos apresentados nas aulas teóricas relativos a escalonamento temporal. Surgiram pequenos impasses durante o desenvolvimento, referidos acima, mas foram facilmente ultrapassados. Como tal, é possível concluir que o trabalho foi desenvolvido com sucesso.