Projeto de Sistemas Digitais

MEEC

Best Approximation

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 terceiro projeto, foi proposto o desenvolvimento de uma máquina que tem como objetivo encontrar, entre várias retas fornecidas, a melhor aproximação de um dado conjunto de pontos. Foi desenvolvida uma solução de modo a realizar esta operação do modo mais rápido possível.

2 Processo de design

O pseudocódigo dado como base para obter a a melhor reta para o conjunto de pontos é o seginte:

```
minError = maxInt;
bestLine = null;
for(k = 0; k < Nlines; k++){
    fitError = 0;
    for(i = 0; i < Npoints; i++){
        yki = m[k] * x[i] + b[k];
        error = |yki - y[i]|;
        fitError = fitError + error;
    }
    if(fitError < minError){
        bestLine = k;
        minError = fitError;
    }
}
```

Este consiste em verificar reta a reta qual a melhor, ou seja, qual a que tem a menor erro, o qual é definido como a soma da diferença entre os pontos dados ($y[i]$) e os pontos resultantes das equações das retas (yki) correspondentes.

Inicialmente, é calculado dentro dos dois ciclos for o ponto yki e a sua diferença com o $y[i]$ correspondente. Ainda dentro do ciclo for interior, a referida diferença é somada com as calculadas anteriormente num acumulador ($fitError$) e já fora do ciclo for interior, ou seja, no ciclo que muda a reta, o valor calculado ($fitError$) é comparado com o menor obtido anteriormente ($minError$), guardando o melhor, ou guardando diretamente caso seja a primeira reta calculada.

2.1 Pipelining / Datapath

Numa primeira fase, este algoritmo foi implementado em *pipeline* com 4 estágios, no qual os primeiros dois ciclos correspondem à primeira linha de código no loop interior e os restantes correspondem às restantes linhas, também no loop interior, do pseudocódigo apresentado. Desta maneira é possível obter um *throughput* de um ponto por ciclo de *clock*.

De forma a aproveitar os recursos da placa, é usada uma DSP48E1 para a operação $yki = mx + b$ assinalada em baixo.

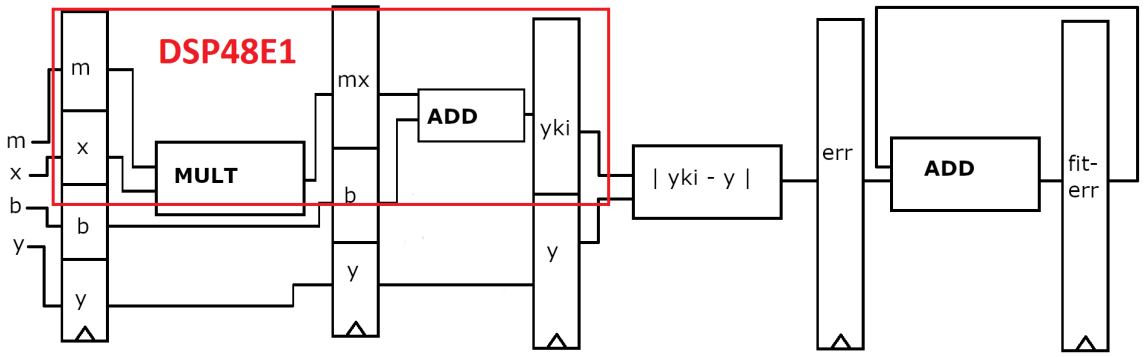

Figura 1: Datapath

2.2 FSM

Para controlar o pipeline, foi criada uma máquina de estados finita de Mealy com 5 estados: *Init*, *readLine*, *readPoints*, *writeBest* e *done*. Para auxiliar a organização dos dados, são também usados dois contadores, um referente aos pontos e outro para as retas.

O estado *Init* é o estado inicial que espera o sinar de *start* para seguir então para o estado *readLine*. Este, por sua vez, aciona o contador de retas, cujo valor é também utilizado numa das entradas da memória para selecionar o endereço dos valores m e b da reta. De seguida o estado muda para o *readPoints*.

No estado *readPoints* é acionado o contador de pontos, que está ligado à outra entrada da memória, tendo como objetivo selecionar o endereço correspondente aos valores x e y. É também acionado o pipeline para o célculo do *fiterror*. Ao chegar ao último ponto o estado muda para *writeBest*.

O *writeBest* é respnsável por comparar o erro da reta atual com o melhor erro até ao momento. Desta forma, tanto o valor do erro, como o índice da melhor reta, estão sempre guardados. Caso o erro da última reta esteja calculado, o estado seguinte é o *done*. Caso contrário, a máquina de estados volta ao estado *readLine*.

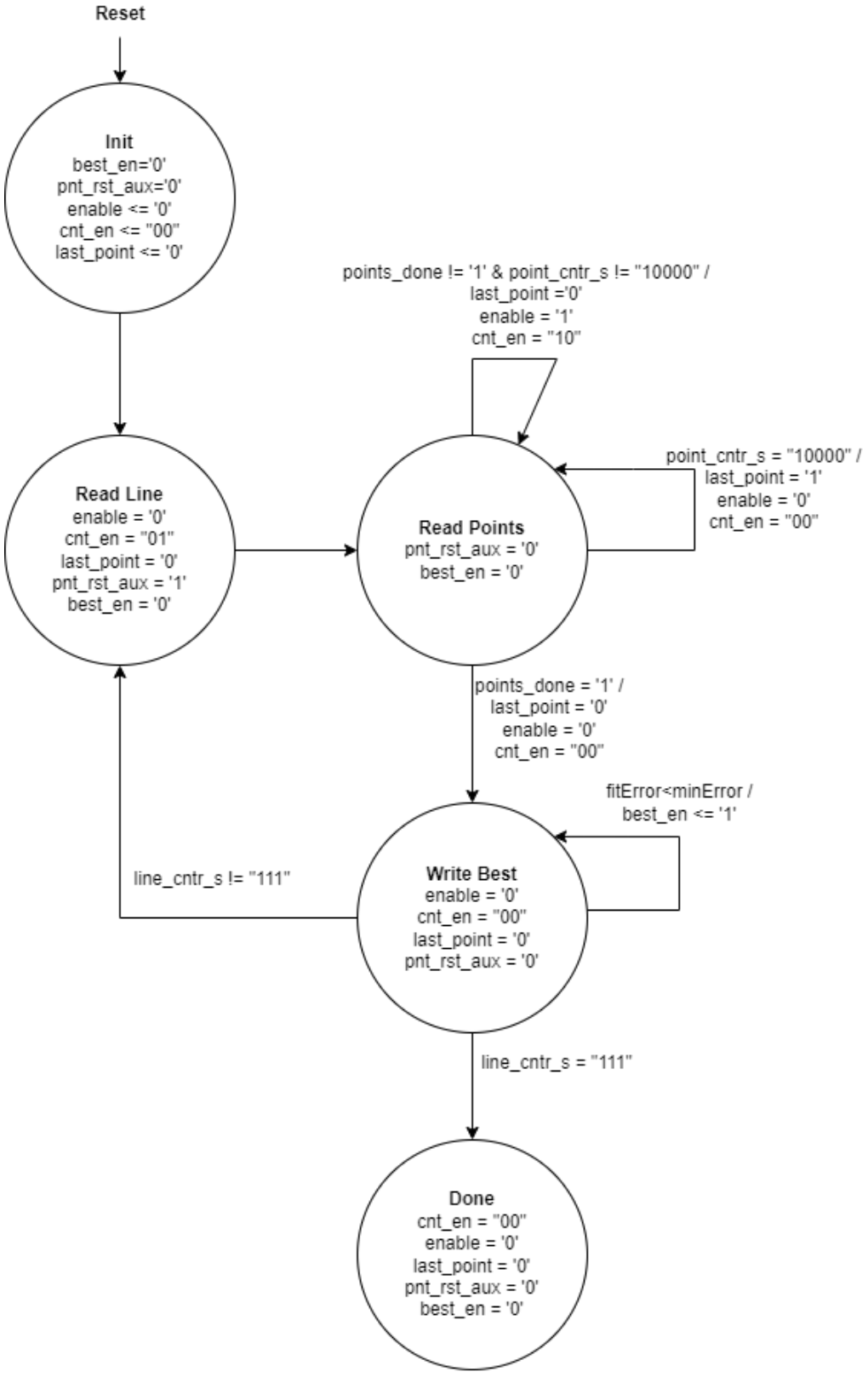
Reset
Init
best_en='0'
pnt_rst_aux='0'
enable <= '0'
cnt_en <= "00"
last_point <= '0'
points_done != '1' & point_cntr_s != "10000" /
last_point ='0'
enable = '1'
cnt_en = "10"
Read Line
enable = '0'
cnt_en = "01"
last_point = '0'
pnt_rst_aux = '1'
best_en = '0'
Read Points
pnt_rst_aux = '0'
best_en = '0'
point_cntr_s = "10000" /
last_point = '1'
enable = '0'
cnt_en = "00"
points_done = '1' /
last_point = '0'
enable = '0'
cnt_en = "00"
fitError<minError /
best_en <= '1'
line_cntr_s != "111"
Write Best
enable = '0'
cnt_en = "00"
last_point = '0'
pnt_rst_aux = '0'
line_cntr_s = "111"
Done
cnt_en = "00"
enable = '0'
last_point = '0'
pnt_rst_aux = '0'
best_en = '0'

Figura 2: FSM

2.3 Paralelismo

Após confirmar o funcionamento da lógica do primeiro design, e com o objetivo de melhorar a performance do circuito, aplicou-se o conceito de paralelismo. Para isso, as operações foram divididas por quatro *"datapaths"* iguais permitindo reduzir o tempo de processamento para aproximadamente *1/4*. Os *"datapaths"* foram então divididos da seguinte maneira:

1. Cálculo de retas pares, pontos pares
2. Cálculo de retas pares, pontos ímpares
3. Cálculo de retas ímpares, pontos pares
4. Cálculo de retas ímpares, pontos ímpares

Com a divisão das operações foi necessário alterar a máquina de estados adicionando-lhe dois estados: *s_sum*, que soma os erros calculados para os pontos pares e ímpares da mesma reta, e *s_comp_best*, que compara os erros das duas retas, par e ímpar, para posteriormente comparar a melhor das duas com a melhor até ao momento. É apresentada de seguida a *FSM* obtida:

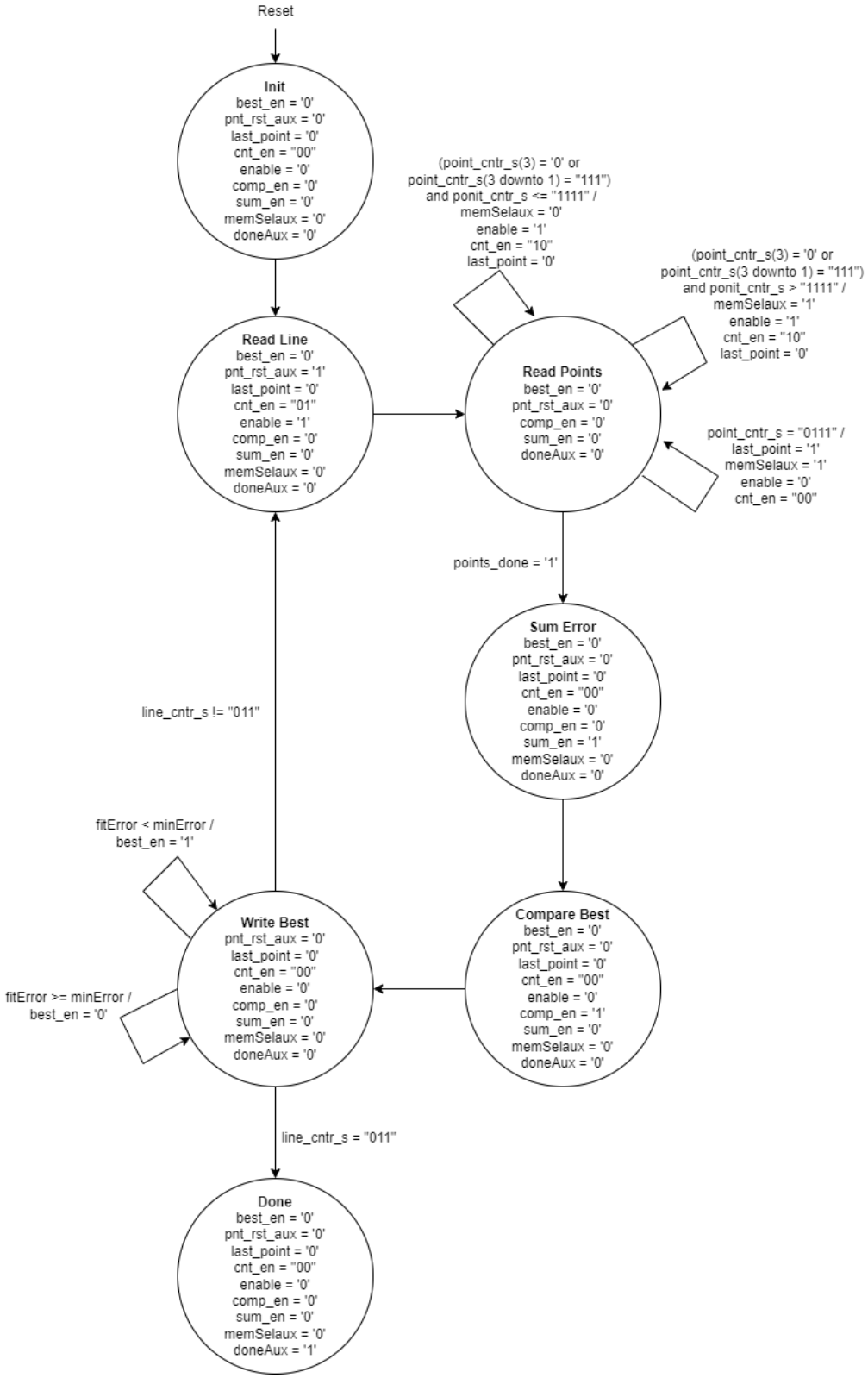

Figura 3: FSM com Paralelismo

3 Testes realizados

3.1 Simulação

Antes de realizar testes em hardware, foi criado um ficheiro de testes (*circuito_tb.vhd*) de modo a verificar o correto funcionamento do sistema desenvolvido através de uma simulação. Utilizando este método, foi possível obter e calcular o beneficio temporal da utilização de paralelismo, bem como verificar e resolver quaisqueres erros de cálculo. Utilizando a simulação foi possível fazer o *debug* de vários aspetos do sistema, como o alinhamento de *fixed point* e a correta utilização da *DSP*.

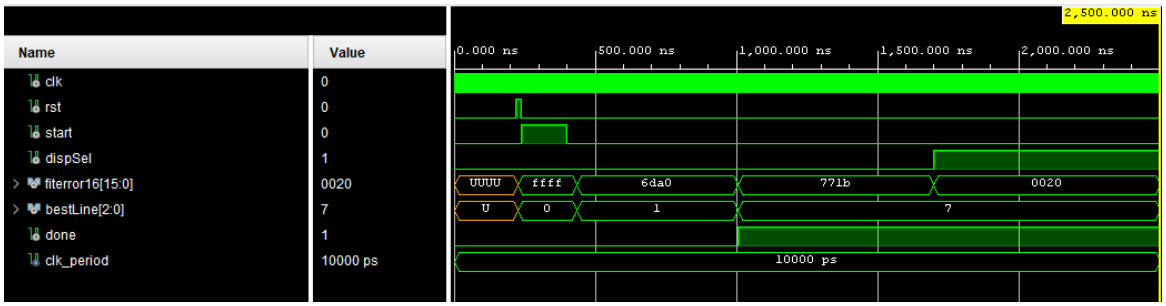

Figura 4: Simulação para a memória 3

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:

	s/ Paralelismo		c/ Paralelismo	
Recursos	Utilização	Utilização(%)	Utilização	Utilização(%)
LUT	197	0,95%	777	3,74%
FF	159	0,38%	644	1,55%
DSP	1	1,11%	4	4,44%
BUFG	3	9,38%	3	9,38%

A utilização de recursos foi a esperada. Foi possível obter uma utilização de I/O mais baixa através de optimizações no ficheiro de constraints, mas por outro lado, a utilização dos outros recursos foi um pouco mais alta. Esta alteração era expectável, dado à utilização de paralelismo, mas é possível verificar que a utilização de recursos não quadriplica, sendo assim também fácil concluir que esta alteração compensa.

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.

3.3 Testes na Placa

Os resultados obtidos durante os testes na placa foram os esperados, ou seja, correspondentes aos obtidos previamente em simulação. Foi testada uma diminuição do período de relógio, dado que existia um *WNS* positivo, mas rapidamente foi possível verificar que este alterava o resultado da simulação e não da placa, ou seja, o período de relogio não podia ser encurtado. Esta conclusão é possível de inferir dado que o período de relógio na placa é sempre de 10ns.

4 Conclusões

Concluindo, ao longo do desenvolvimento deste trabalho, foi possível aprofundar competências de optimização desenvolvidas durante os dois primeiros laboratórios, quer do *datapath* ou da máquina de estados, bem como na utilização da placa. Foi também possível utilizar o conceito de paralelismo, algo que ainda não tinha sido implementado nos laboratórios. Surgiram alguns impasses durante o desenvolvimento, mas, dado que estes foram ultrapassados, é possível concluir que o trabalho foi desenvolvido com sucesso.