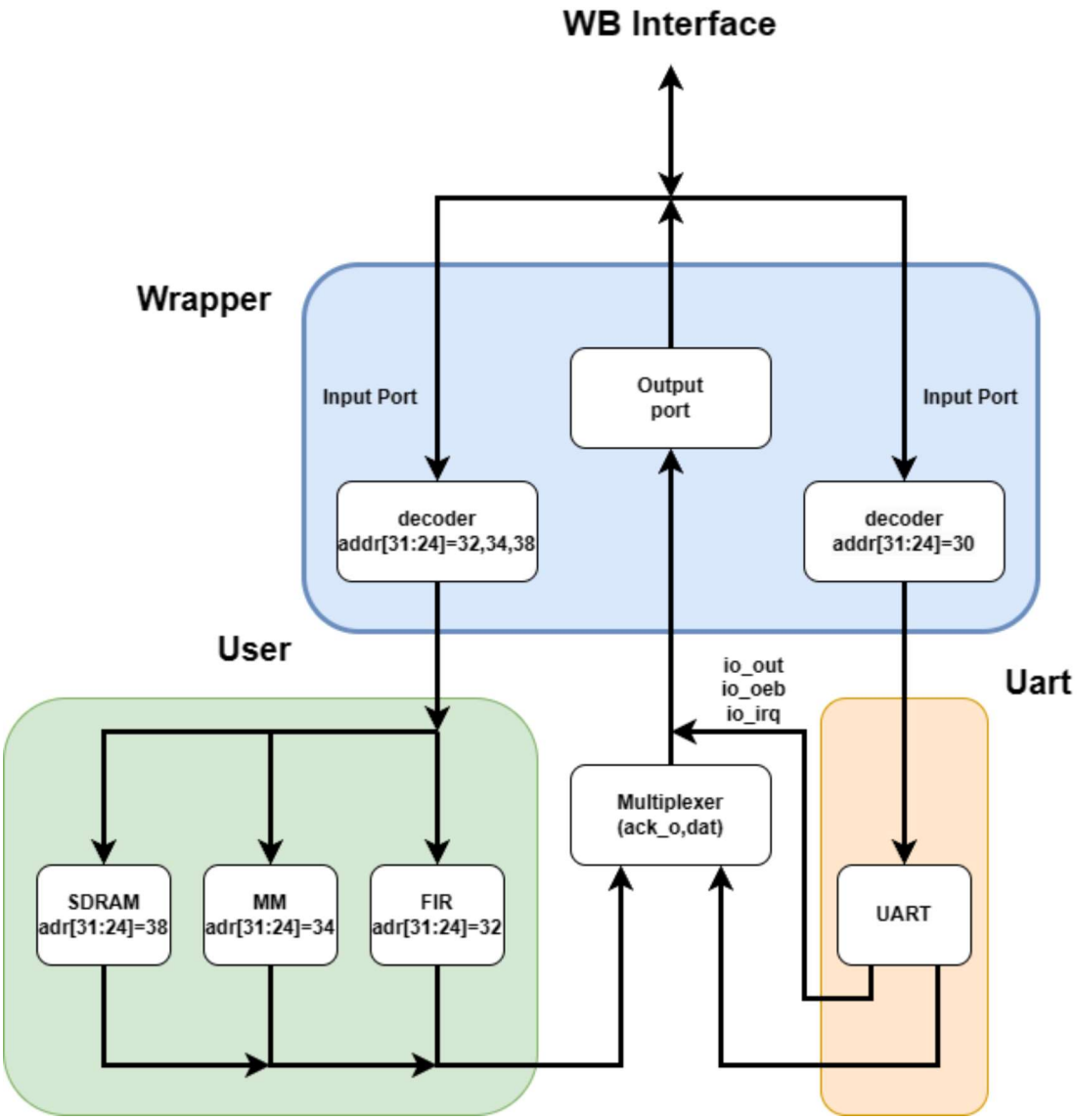
Final Project

Outline

本次期末專題在於優化Lab6的四種功能，首先因為有完成Lab-SDRAM的Prefetch功能，故直接將SDRAM應用在本次專題，由原先bram所需要的10個cycle讀寫，縮減至SDRAM寫入三個cycle以及讀取平均約5~7個cycle，大幅縮小讀寫時間。且另外在compiler加入-Os指令使其更符合嵌入式系統以加速運算。最後硬體因為FIR和矩陣乘法為同步系統中第一和第二耗費時間的運算，故將FIR和矩陣乘法的計算用硬體做加速，並利用Wishbone傳輸結果。

Block design



Improvements with the Integration of SDRAM

SDRAM主要使用的是Prefetch的功能，用以縮短連續讀寫時Wishbone需使用ack來確認輸出資料的問題，使controller在IDLE state可以只停留一個cycle而非3個cycle(紅圈處)。

詳細內容可參考：https://hackmd.io/CLFwIGIURNGsBOIRYBocYA
(https://hackmd.io/CLFwIGIURNGsBOIRYBocYA)

compiler -Os optimize

-O1, -O2, -O3 以及 -Os 參數是 gcc 中的最佳化選項，可以最佳化程式。其中，-Os 參數適合用於嵌入式系統或者記憶體有限的環境，可以節省空間和提高執行速度。

Matrix Multiplication Hardware Design

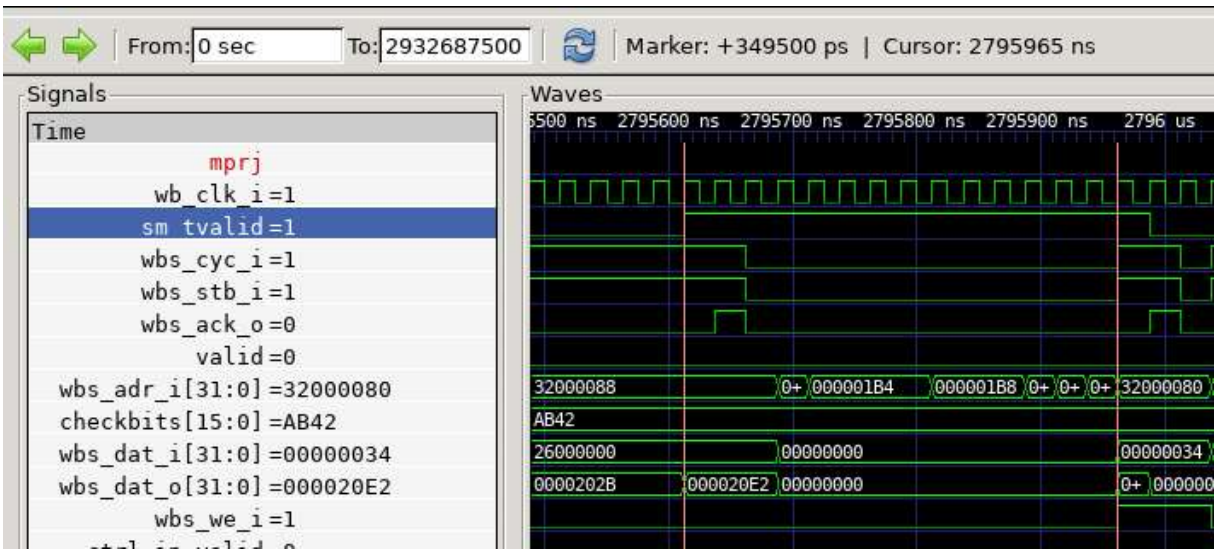
矩陣乘法主要透過Wishbone的介面做傳輸，其運作原理如下圖所示，在reset時將各矩陣內部的數值設定好，並在Wishbone等待時開始運算(由valid控制)再利用ack將計算好的值送出，其計算並不需要透過cpu，故可以減少data的搬運的時間。

```
always @(posedge wb_clk_i) begin
  if (wb_rst_i) begin
    // Initial A
    // Initial B
  end
end

always @(posedge wb_clk_i) begin
  if(wb_rst_i) begin
    // Initial sum, i, j, k, output, counter
  end
  else begin
    if (valid & !wbs_we_i) begin
      if (i == 4'd4) begin
          // finish matmul, set i, j, k = 15
          sum <= 8'd0;
          i <= 4'd15;
          j <= 4'd15;
          k <= 4'd15;
          if(wbs_ack_o)
            check_out <= 0;
      end
      else if (j == 4'd4) begin
          // i+1 for next A's row
          j <= 4'd0;
          k <= 4'd0;
          i <= i + 1;
          if(wbs_ack_o)
            check_out <= 0;
      end
      else if (k == 4'd4) begin
          // output sum and counter++
          result <= sum;
          j <= j + 1;
          k <= 4'd0;
          sum <= 8'd0;
          check_out <= 1;
          output_counter <= output_counter + 1;
       end
      else if (k < 4'd4) begin
          // sum += A[i][k] * B[k][j]
          sum <= sum + A[(i << 2) + k] * B[(k << 2) + j];
          k <= k + 1;
          if(wbs_ack_o)
            check_out <= 0;
      end
    end
  end
end
```

FIR Optimization

實作主要透過lab_fir中的硬體結合lab4-2的韌體，避免透過cpu去計算fir的結果，另外在韌體上將loop從i=0 ~ 63改成i=1 ~ 64，以及在收發XY時不在CPU端做data的檢查都可以有效減少其運行的cycle數，最後透過波型可以發現Y發起ready後下一個X進來需要14個cycle。

```
//i=1~64，在274即可傳送X
38000258 <fir>:
38000268:       00100793                li      a5,1
3800026c:       320006b7                lui     a3,0x32000
38000270:       04100613                li      a2,65
38000274:       08f6a023                sw      a5,128(a3) # 32000080 <_esram_rom+0x21fffb68>
38000278:       0886a583                lw      a1,136(a3)
3800027c:       00178793                addi    a5,a5,1
38000298:       00008067                ret

//i=0~63，在278才傳送X
38000258 <fir>:
38000268:       00000793                li      a5,0
3800026c:       320006b7                lui     a3,0x32000
38000270:       04000613                li      a2,64
38000274:       00178793                addi    a5,a5,1
38000278:       08f6a023                sw      a5,128(a3) # 32000080 <_esram_rom+0x21fffb68>
3800027c:       0886a583                lw      a1,136(a3)
38000298:       00008067                ret
```



Evaluation of Each Task Compared to Baseline

1. Each task execution time (with argument -Os)

Task	Lab6	Lab6 + sdram	Lab6 + sdram + fir	Lab6 + sdram + fir + mm
Fir	418357	273181	7429	7929
Qsort	17835	13254	14097	13048
Matmul	46472	29653	28078	2241
Overall	564654	396884	116907	93749

2. -Os

With Os	Without Os
93749	358109

FPGA

```
asyncio.run(async_main())
```

```
Start Caravel Soc
Waitting for interrupt
hello
```

```
print ("0x10 = ", hex(ipPS.read(0x10)))
print ("0x14 = ", hex(ipPS.read(0x14)))
print ("0x1c = ", hex(ipPS.read(0x1c)))
print ("0x20 = ", hex(ipPS.read(0x20)))
print ("0x34 = ", hex(ipPS.read(0x34)))
print ("0x38 = ", hex(ipPS.read(0x38)))
```

```
0x10 =  0x0
0x14 =  0x0
0x1c =  0xab520040
0x20 =  0x0
0x34 =  0x20
0x38 =  0x3f
```

Synthesis Report

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	6071	0	0	53200	11.41
LUT as Logic	5883	0	0	53200	11.06
LUT as Memory	188	0	0	17400	1.08
LUT as Distributed RAM	18	0			
LUT as Shift Register	170	0			
Slice Registers	7056	0	0	106400	6.63
Register as Flip Flop	6919	0	0	106400	6.50
Register as Latch	137	0	0	106400	0.13
F7 Muxes	169	0	0	26600	0.64
F8 Muxes	47	0	0	13300	0.35

Conclusion

本次實作以硬體加速為主，韌體與軟體為輔，主要以同步系統的硬體加速的原因是濾波器與矩陣乘法皆需要大量搬運資料，倘若使用cpu去計算則會浪費許多時間故以此點下手，到最後結果可以發現testbench已經在空等uart訊號，故再加速同步系統的其他硬體已無效用，若是要再繼續加速則必須要去加速uart的收發才可以更進一步壓低運算時間。