SOC Lab 4-1

Group 10

R11943022 電子所碩二 范詠為
R11943025 電子所碩二 謝郁楷
R11943124 電子所碩二 曾郁瑄

Overview

在 lab 4-1 中，我們設計 controller 去控制 BRAM 與 wishbone bus，並設計 FIR filter 的 firmware code，再運用建立好的 Caravel SoC 環境去模擬電路行為。

1. Explanation of your firmware code

如下圖，fir.c 運作步驟如下:

I. 初始化 outputsignal

II. 運用 2 for loop 去進行 FIR filter，並用 shift register 去將對應的 inputsignal 與 tap 做運算。outputsignal 在每次 iteration 得到該點的 partial sum，並累加起來儲存，完整運算完後會 reutrn 回 counter_la_fir.c 中的 fir function。

```c
#include "fir.h"

void __attribute__ ( ( section ( ".mprjram" ) ) ) initfir() {
    //initial your fir
    //int i = 0;
    for (int i = 0; i < N; i++)
    {
        outputsignal[i] = 0;
    }
}

int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir(){
    initfir();
    //write down your fir
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if ((i - j) >= 0)
            {
                outputsignal[i] = outputsignal[i] + inputsignal[i - j] * taps[j];
            }
        }
    }
    return outputsignal;
}
```

如下圖，fir.h 中設 tap 以及 inputsignal data

```c
#ifndef __FIR_H__
#define __FIR_H__

#define N 11

int taps[N] = {0,-10,-9,23,56,63,56,23,-9,-10,0};
int inputbuffer[N];
int inputsignal[N] = {1,2,3,4,5,6,7,8,9,10,11};
int outputsignal[N];
#endif
```

2. Explanation of your assembly code

- Apply the configuration
  - Assembly code

```
1000057c: f00067b7          lui a5,0xf0006
10000580: 00100713          li  a4,1
10000584: 00e7a023          sw  a4,0(a5) # f0006000 <_esram+0xb8005e98>
10000588: 260007b7          lui a5,0x26000
1000058c: 00100713          li  a4,1
10000590: 00e7a023          sw  a4,0(a5) # 26000000 <_esram_rom+0x15fff7f8>
10000594: 00000013          nop
10000598: 260007b7          lui a5,0x26000
1000059c: 0007a703          lw  a4,0(a5) # 26000000 <_esram_rom+0x15fff7f8>
100005a0: 00100793          li  a5,1
100005a4: fef70ae3          beq a4,a5,10000598 <main+0x2ac>
```

  - counter_fir_la.c

```
103     reg_mprj_xfer = 1;
104     while (reg_mprj_xfer == 1);
```

- 與對應到的 address 進行寫入
  - Assembly code

```
100005a4: fef70ae3          beq a4,a5,10000598 <main+0x2ac>
100005a8: f00037b7          lui a5,0xf0003
100005ac: 00c78713          addi  a4,a5,12 # f000300c <_esram+0xb8002ea4>
100005b0: 00000793          li  a5,0
100005b4: 00f72023          sw  a5,0(a4)
100005b8: f0003737          lui a4,0xf0003
100005bc: 01c70713          addi  a4,a4,28 # f000301c <_esram+0xb8002eb4>
100005c0: 00f72023          sw  a5,0(a4)
100005c4: f00037b7          lui a5,0xf0003
100005c8: 00878713          addi  a4,a5,8 # f0003008 <_esram+0xb8002ea0>
100005cc: fff00793          li  a5,-1
100005d0: 00f72023          sw  a5,0(a4)
100005d4: f0003737          lui a4,0xf0003
100005d8: 01870713          addi  a4,a4,24 # f0003018 <_esram+0xb8002eb0>
100005dc: 00f72023          sw  a5,0(a4)
100005e0: f00037b7          lui a5,0xf0003
100005e4: 00478713          addi  a4,a5,4 # f0003004 <_esram+0xb8002e9c>
100005e8: 00000793          li  a5,0
100005ec: 00f72023          sw  a5,0(a4)
100005f0: f0003737          lui a4,0xf0003
100005f4: 01470713          addi  a4,a4,20 # f0003014 <_esram+0xb8002eac>
100005f8: 00f72023          sw  a5,0(a4)
100005fc: f0003737          lui a4,0xf0003
10000600: 00000793          li  a5,0
10000604: 00f72023          sw  a5,0(a4) # f0003000 <_esram+0xb8002e98>
10000608: f0003737          lui a4,0xf0003
1000060c: 01070713          addi  a4,a4,16 # f0003010 <_esram+0xb8002ea8>
10000610: 00f72023          sw  a5,0(a4)
10000614: 260007b7          lui a5,0x26000
10000618: 00c78793          addi  a5,a5,12 # 2600000c <_esram_rom+0x15fff804
1000061c: ab400737          lui a4,0xab400
10000620: 00e7a023          sw  a4,0(a5)
10000624: f00037b7          lui a5,0xf0003
10000628: 03878793          addi  a5,a5,56 # f0003038 <_esram+0xb8002ed0>
1000062c: 0007a023          sw  zero,0(a5)
10000630: f00037b7          lui a5,0xf0003
10000634: 00878713          addi  a4,a5,8 # f0003008 <_esram+0xb8002ea0>
10000638: 00000793          li  a5,0
1000063c: 00f72023          sw  a5,0(a4) # ab400000 <_esram+0x733ffe98>
10000640: f0003737          lui a4,0xf0003
10000644: 01870713          addi  a4,a4,24 # f0003018 <_esram+0xb8002eb0>
10000648: 00f72023          sw  a5,0(a4)
```

- counter_fir_la.c

```
106         // Configure LA probes [31:0], [127:64] as inputs to the cpu
107     // Configure LA probes [63:32] as outputs from the cpu
108     reg_la0_oenb = reg_la0_iena = 0x00000000;    // [31:0]
109     reg_la1_oenb = reg_la1_iena = 0xFFFFFFFF;    // [63:32]
110     reg_la2_oenb = reg_la2_iena = 0x00000000;    // [95:64]
111     reg_la3_oenb = reg_la3_iena = 0x00000000;    // [127:96]
112 
113     // Flag start of the test
114     reg_mprj_datal = 0xAB400000;
115 
116     // Set Counter value to zero through LA probes [63:32]
117     reg_la1_data = 0x00000000;
118 
119     // Configure LA probes from [63:32] as inputs to disable counter write
120     reg_la1_oenb = reg_la1_iena = 0x00000000;
```

- 呼叫 fir function 並將結果寫入 reg_mprj_datal 指定的 address
  - Assembly code

```
609  38000024 <initfir>:
610  38000024: fe010113           addi  sp,sp,-32
611  38000028: 00812e23           sw  s0,28(sp)
612  3800002c: 02010413           addi  s0,sp,32
613  38000030: fe042623           sw  zero,-20(s0)
614  38000034: 0240006f           j 38000058 <initfir+0x34>
615  38000038: 08800713           li  a4,136
616  3800003c: fec42783           lw  a5,-20(s0)
617  38000040: 00279793           slli  a5,a5,0x2
618  38000044: 00f707b3           add a5,a4,a5
619  38000048: 0007a023           sw  zero,0(a5)
620  3800004c: fec42783           lw  a5,-20(s0)
621  38000050: 00178793           addi  a5,a5,1
622  38000054: fef42623           sw  a5,-20(s0)
623  38000058: fec42703           lw  a4,-20(s0)
624  3800005c: 00a00793           li  a5,10
625  38000060: fce7dce3           bge a5,a4,38000038 <initfir+0x14>
626  38000064: 00000013           nop
627  38000068: 00000013           nop
628  3800006c: 01c12403           lw  s0,28(sp)
629  38000070: 02010113           addi  sp,sp,32
630  38000074: 00008067           ret
```

  - counter_fir_la.c

```
130     int* tmp = fir();
131     reg_mprj_datal = *tmp << 16;
132     reg_mprj_datal = *(tmp+1) << 16;
133     reg_mprj_datal = *(tmp+2) << 16;
134     reg_mprj_datal = *(tmp+3) << 16;
135     reg_mprj_datal = *(tmp+4) << 16;
136     reg_mprj_datal = *(tmp+5) << 16;
137     reg_mprj_datal = *(tmp+6) << 16;
138     reg_mprj_datal = *(tmp+7) << 16;
139     reg_mprj_datal = *(tmp+8) << 16;
140     reg_mprj_datal = *(tmp+9) << 16;
141     reg_mprj_datal = *(tmp+10) << 16;
```

- fir.c 中作 fir filter 的運算
  - Assembly code

```
38000078 <fir>:
38000078: fe010113          addi  sp,sp,-32
3800007c: 00112e23          sw  ra,28(sp)
38000080: 00812c23          sw  s0,24(sp)
38000084: 00912a23          sw  s1,20(sp)
38000088: 02010413          addi  s0,sp,32
3800008c: f99ff0ef          jal ra,38000024 <initfir>
38000090: fe042623          sw  zero,-20(s0)
38000094: 0ac0006f          j 38000140 <fir+0xc8>
38000098: fe042423          sw  zero,-24(s0)
3800009c: 08c0006f          j 38000128 <fir+0xb0>
380000a0: fec42703          lw  a4,-20(s0)
380000a4: fe842783          lw  a5,-24(s0)
380000a8: 40f707b3          sub a5,a4,a5
380000ac: 0607c863          bltz  a5,3800011c <fir+0xa4>
380000b0: 08800713          li  a4,136
380000b4: fec42783          lw  a5,-20(s0)
380000b8: 00279793          slli  a5,a5,0x2
380000bc: 00f707b3          add a5,a4,a5
380000c0: 0007a483          lw  s1,0(a5)
380000c4: fec42703          lw  a4,-20(s0)
380000c8: fe842783          lw  a5,-24(s0)
380000cc: 40f707b3          sub a5,a4,a5
380000d0: 02c00713          li  a4,44
380000d4: 00279793          slli  a5,a5,0x2
380000d8: 00f707b3          add a5,a4,a5
380000dc: 0007a683          lw  a3,0(a5)
380000e0: 00000713          li  a4,0
380000e4: fe842783          lw  a5,-24(s0)
380000e8: 00279793          slli  a5,a5,0x2
380000ec: 00f707b3          add a5,a4,a5
```

```
380000f0: 0007a783          lw  a5,0(a5)
380000f4: 00078593          mv  a1,a5
380000f8: 00068513          mv  a0,a3
380000fc: f05ff0ef          jal ra,38000000 <__mulsi3>
38000100: 00050793          mv  a5,a0
38000104: 00f48733          add a4,s1,a5
38000108: 08800693          li  a3,136
3800010c: fec42783          lw  a5,-20(s0)
38000110: 00279793          slli  a5,a5,0x2
38000114: 00f687b3          add a5,a3,a5
38000118: 00e7a023          sw  a4,0(a5)
3800011c: fe842783          lw  a5,-24(s0)
38000120: 00178793          addi  a5,a5,1
38000124: fef42423          sw  a5,-24(s0)
38000128: fe842703          lw  a4,-24(s0)
3800012c: 00a00793          li  a5,10
38000130: f6e7d8e3          bge a5,a4,380000a0 <fir+0x28>
38000134: fec42783          lw  a5,-20(s0)
38000138: 00178793          addi  a5,a5,1
3800013c: fef42623          sw  a5,-20(s0)
38000140: fec42703          lw  a4,-20(s0)
38000144: 00a00793          li  a5,10
38000148: f4e7d8e3          bge a5,a4,38000098 <fir+0x20>
3800014c: 08800793          li  a5,136
38000150: 00078513          mv  a0,a5
38000154: 01c12083          lw  ra,28(sp)
38000158: 01812403          lw  s0,24(sp)
3800015c: 01412483          lw  s1,20(sp)
38000160: 02010113          addi  sp,sp,32
38000164: 00008067          ret
```

  - fir.c

```c
#include "fir.h"

void __attribute__ ( ( section ( ".mprjram" ) ) ) initfir() {
    //initial your fir
    //int i = 0;
    for (int i = 0; i < N; i++)
    {
        outputsignal[i] = 0;
    }
}

int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir(){
    initfir();
    //write down your fir
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if ((i - j) >= 0)
            {
                outputsignal[i] = outputsignal[i] + inputsignal[i - j] * taps[j];
            }
        }
    }
    return outputsignal;
}
```

3. How does it execute a multiplication in assembly code?

將乘法拆成不斷累加做計算。counter_la_fir.out 中顯示了 fir.c 中做 multiplication 的 assembly code，如下圖。

```
38000000 <__mulsi3>:
38000000: 00050613          mv  a2,a0
38000004: 00000513          li  a0,0
38000008: 0015f693          andi  a3,a1,1
3800000c: 00068463          beqz  a3,38000014 <__mulsi3+0x14>
38000010: 00c50533          add a0,a0,a2
38000014: 0015d593          srli  a1,a1,0x1
38000018: 00161613          slli  a2,a2,0x1
3800001c: fe0596e3          bnez  a1,38000008 <__mulsi3+0x8>
38000020: 00008067          ret
```

a0 及 a1 將要做 multiplication 的 data 送進來，a0 assign 給 a2，再將 a0 設為 0 作為後續累加。當 a3 為不為 0 時，進行 a0+a2 的累加運算。當 a3 為 0 時，跳過累加部分，直接進到將 a1 右移一位的指令，後將 a2 左移一位作乘積的進位，再去判斷如果 a1 為 0 時為完成累加的運算，反之則繼續回到 38000008 的位子進行運算。

4. What address allocate for user project and how many space is required to allocate to firmware code

從 section.lds 可以看到 user project 的 original address 為 0x3800000。

```
11  MEMORY {
12      vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
13      dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
14      dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
15      flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
16      mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000
17      mprjram : ORIGIN = 0x38000000, LENGTH = 0x00400000
18      hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
19      csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
20  }
21
22  SECTIONS
```

從 counter_la_fir.out 可以看到 fir filter 計算的空間為 0x38000000~0x38000164，總計需要 356 bytes (Hex 164)。

```
598  38000000 <__mulsi3>:
599  38000000: 00050613          mv  a2,a0
600  38000004: 00000513          li  a0,0
601  38000008: 0015f693          andi  a3,a1,1
602  3800000c: 00068463          beqz  a3,38000014 <__mulsi3+0x14>
603  38000010: 00c50533          add a0,a0,a2
604  38000014: 0015d593          srli  a1,a1,0x1
605  38000018: 00161613          slli  a2,a2,0x1
606  3800001c: fe0596e3          bnez  a1,38000008 <__mulsi3+0x8>
607  38000020: 00008067          ret
608
609  38000024 <initfir>:
610  38000024: fe010113          addi  sp,sp,-32
611  38000028: 00812e23          sw  s0,28(sp)
612  3800002c: 02010413          addi  s0,sp,32
613  38000030: fe042623          sw  zero,-20(s0)
614  38000034: 0240006f          j 38000058 <initfir+0x34>
615  38000038: 08800713          li  a4,136
616  3800003c: fec42783          lw  a5,-20(s0)
617  38000040: 00279793          slli  a5,a5,0x2
618  38000044: 00f707b3          add a5,a4,a5
619  38000048: 0007a023          sw  zero,0(a5)
620  3800004c: fec42783          lw  a5,-20(s0)
621  38000050: 00178793          addi  a5,a5,1
622  38000054: fef42623          sw  a5,-20(s0)
623  38000058: fec42703          lw  a4,-20(s0)
624  3800005c: 00a00793          li  a5,10
625  38000060: fce7dce3          bge a5,a4,38000038 <initfir+0x14>
626  38000064: 00000013          nop
627  38000068: 00000013          nop
628  3800006c: 01c12403          lw  s0,28(sp)
629  38000070: 02010113          addi  sp,sp,32
630  38000074: 00008067          ret
```

```
632  38000078 <fir>:
633  38000078: fe010113          addi  sp,sp,-32
634  3800007c: 00112e23          sw  ra,28(sp)
635  38000080: 00812c23          sw  s0,24(sp)
636  38000084: 00912a23          sw  s1,20(sp)
637  38000088: 02010413          addi  s0,sp,32
638  3800008c: f99ff0ef          jal ra,38000024 <initfir>
639  38000090: fe042623          sw  zero,-20(s0)
640  38000094: 0ac0006f          j 38000140 <fir+0xc8>
641  38000098: fe042423          sw  zero,-24(s0)
642  3800009c: 08c0006f          j 38000128 <fir+0xb0>
643  380000a0: fec42703          lw  a4,-20(s0)
644  380000a4: fe842783          lw  a5,-24(s0)
645  380000a8: 40f707b3          sub a5,a4,a5
646  380000ac: 0607c863          bltz  a5,3800011c <fir+0xa4>
647  380000b0: 08800713          li  a4,136
648  380000b4: fec42783          lw  a5,-20(s0)
649  380000b8: 00279793          slli  a5,a5,0x2
650  380000bc: 00f707b3          add a5,a4,a5
651  380000c0: 0007a483          lw  s1,0(a5)
652  380000c4: fec42703          lw  a4,-20(s0)
653  380000c8: fe842783          lw  a5,-24(s0)
654  380000cc: 40f707b3          sub a5,a4,a5
655  380000d0: 02c00713          li  a4,44
656  380000d4: 00279793          slli  a5,a5,0x2
657  380000d8: 00f707b3          add a5,a4,a5
658  380000dc: 0007a683          lw  a3,0(a5)
659  380000e0: 00000713          li  a4,0
660  380000e4: fe842783          lw  a5,-24(s0)
661  380000e8: 00279793          slli  a5,a5,0x2
662  380000ec: 00f707b3          add a5,a4,a5
```

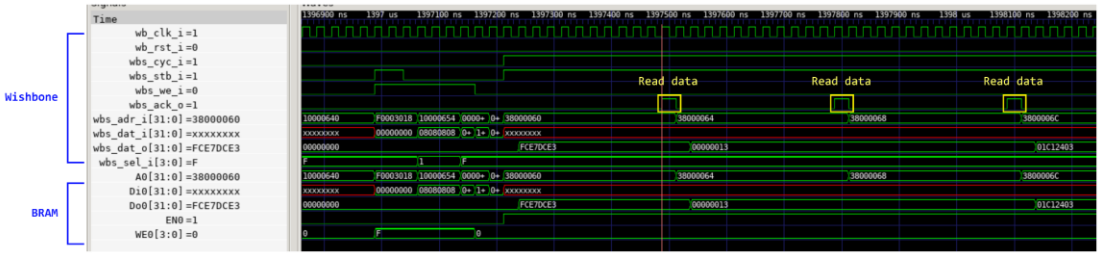
```
663  380000f0: 0007a783          lw  a5,0(a5)
664  380000f4: 00078593          mv  a1,a5
665  380000f8: 00068513          mv  a0,a3
666  380000fc: f05ff0ef          jal ra,38000000 <__mulsi3>
667  38000100: 00050793          mv  a5,a0
668  38000104: 00f48733          add a4,s1,a5
669  38000108: 08800693          li  a3,136
670  3800010c: fec42783          lw  a5,-20(s0)
671  38000110: 00279793          slli  a5,a5,0x2
672  38000114: 00f687b3          add a5,a3,a5
673  38000118: 00e7a023          sw  a4,0(a5)
674  3800011c: fe842783          lw  a5,-24(s0)
675  38000120: 00178793          addi  a5,a5,1
676  38000124: fef42423          sw  a5,-24(s0)
677  38000128: fe842703          lw  a4,-24(s0)
678  3800012c: 00a00793          li  a5,10
679  38000130: f6e7d8e3          bge a5,a4,380000a0 <fir+0x28>
680  38000134: fec42783          lw  a5,-20(s0)
681  38000138: 00178793          addi  a5,a5,1
682  3800013c: fef42623          sw  a5,-20(s0)
683  38000140: fec42703          lw  a4,-20(s0)
684  38000144: 00a00793          li  a5,10
685  38000148: f4e7d8e3          bge a5,a4,38000098 <fir+0x20>
686  3800014c: 08800793          li  a5,136
687  38000150: 00078513          mv  a0,a5
688  38000154: 01c12083          lw  ra,28(sp)
689  38000158: 01812403          lw  s0,24(sp)
690  3800015c: 01412483          lw  s1,20(sp)
691  38000160: 02010113          addi  sp,sp,32
692  38000164: 00008067          ret
```

5. Interface between BRAM and wishbone interface

A. Waveform from xsim

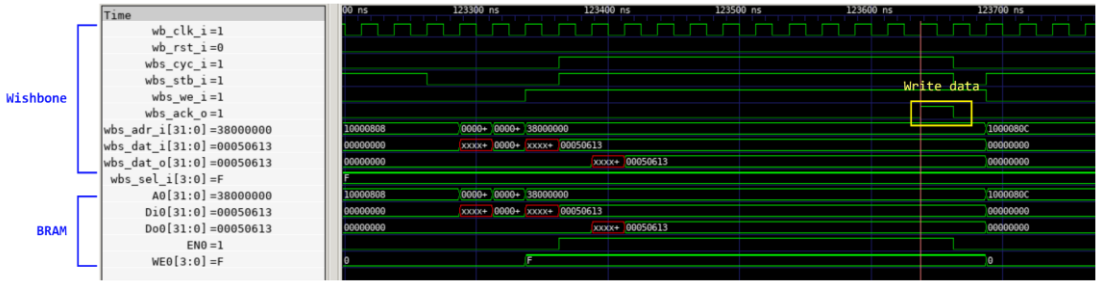
- Read data from BRAM(wbs_we_i = 1)

下圖為從 BRAM read data 的 waveform，可以看到 wbs_cyc_i 與 wbs_stb_i 都為 1 時，表示 BRAM 的資料可以被讀出(valid)，當 wbs_ack_0 為 1 時，Do0 上的 data 被讀出。



- Write data into BRAM (wbs_we_i = 1)

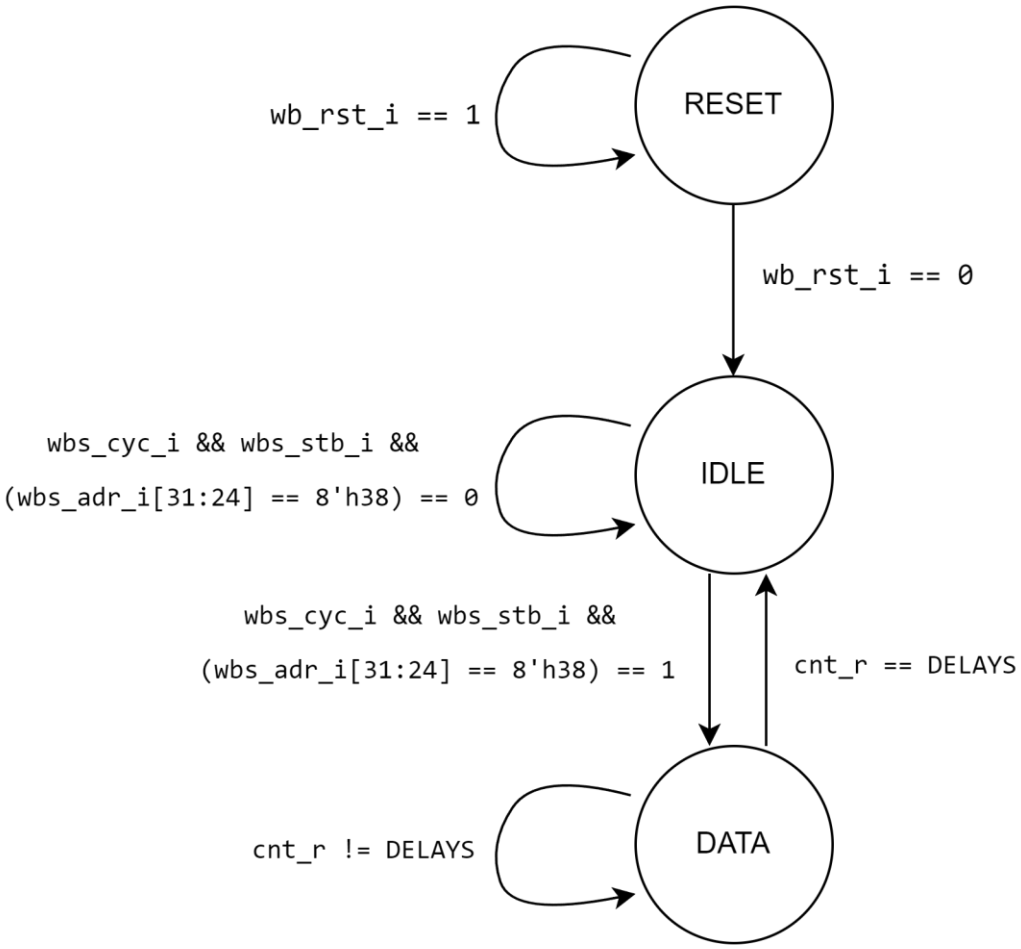
下圖為 write data 進 BRAM 的 waveform，可以看到 wbs_cyc_i 與 wbs_stb_i 都為 1 時，表示 wishbone 的 data 可以被 BRAM(valid)，當 wbs_ack_0 為 1 時，data 寫入 BRAM。



B. Design

FSM

- RESET: 初始狀態
- IDLE: 閒置狀態
- DATA: Delay 10 cycle 後準備 output 結果



6. Synthesis report

clock period = 10.547 (ns)

input delay = 1.299 (ns)

output delay = 2.167 (ns)

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.000 ns	Worst Hold Slack (WHS):	0.000 ns	Worst Pulse Width Slack (WPWS):	4.773 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	571	Total Number of Endpoints:	571	Total Number of Endpoints:	39

All user specified timing constraints are met.

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	10	0	0	53200	0.02
LUT as Logic	10	0	0	53200	0.02
LUT as Memory	0	0	0	17400	0.00
Slice Registers	6	0	0	106400	<0.01
Register as Flip Flop	6	0	0	106400	<0.01
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

2. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	16	0	0	140	11.43
RAMB36/FIFO*	16	0	0	140	11.43
RAMB36E1 only	16				
RAMB18	0	0	0	280	0.00