Lab4-1 Report

Explanation of your firmware code:

1. 對 inputbuffer 與 outputsignal 進行 initialize。
2. 完成 fir 的運算，inputbuffer 進行類似 shift register 的行為來進行 input data 對 tap coefficients 的 alignment。Tap coefficients 會與對應 inputbuffer 位置的 data 相乘得出 partial sum，而 outputsignal 則作為儲存 FIR output result 的變數累加 fir 運算的 partial sum 取得單點 result。重複 N 次計算得到所有 outputsignal 並 return。

```c
#include "fir.h"

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

Comment Code
int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir(){
    initfir();
    //write down your fir
    for (int i = 0; i < N; i++) {
        for (int j = N-1; j > 0; j--) {
            inputbuffer[j] = inputbuffer[j - 1];
            outputsignal[i] += inputbuffer[j] * taps[j];
        }
        inputbuffer[0] = inputsignal[i];
        outputsignal[i] += inputbuffer[0] * taps[0];
    }

    return outputsignal;
}
```

Explanation of your assembly code:

在 counter_la_fir.c 中的 main 裡面，首先讓 mprj_io pin 在對應的 GPIO address 存取值，可以看到圖中標記的 0x2600003c 為在 caravel.h 中定義好的位置，從 assembly code 中可以看出，先計算 offset 將 address 給入 a5 register，再計算 GPIO 位置給如 a4，最後將 a4 存回 a5 所代表的位置。

```c
// User Project Control (0x2300_0000)
#define reg_mprj_xfer (*(volatile uint32_t*)0x26000000)
#define reg_mprj_pwr  (*(volatile uint32_t*)0x26000004)
#define reg_mprj_irq  (*(volatile uint32_t*)0x26100014)
#define reg_mprj_datal (*(volatile uint32_t*)0x2600000c)
#define reg_mprj_datah (*(volatile uint32_t*)0x26000010)

#define reg_mprj_io_0 (*(volatile uint32_t*)0x26000024)
#define reg_mprj_io_1 (*(volatile uint32_t*)0x26000028)
#define reg_mprj_io_2 (*(volatile uint32_t*)0x2600002c)
#define reg_mprj_io_3 (*(volatile uint32_t*)0x26000030)
#define reg_mprj_io_4 (*(volatile uint32_t*)0x26000034)
#define reg_mprj_io_5 (*(volatile uint32_t*)0x26000038)
#define reg_mprj_io_6 (*(volatile uint32_t*)0x2600003c)
```

```c
#define GPIO_MODE_MGMT_STD_OUTPUT      0x1809
#define GPIO_MODE_USER_STD_OUTPUT      0x1808
```

```
1000050c: 00002737           lui a4,0x2
10000510: 80870713           addi  a4,a4,-2040 # 1808 <_fstack+0x1208>
10000514: 00e7a023           sw  a4,0(a5)
10000518: 260007b7           lui a5,0x26000
1000051c: 03078793           addi  a5,a5,48 # 26000030 <_esram_rom+0x15fff828>
10000520: 00002737           lui a4,0x2
10000524: 80870713           addi  a4,a4,-2040 # 1808 <_fstack+0x1208>
10000528: 00e7a023           sw  a4,0(a5)
1000052c: 260007b7           lui a5,0x26000
10000530: 02c78793           addi  a5,a5,44 # 2600002c <_esram_rom+0x15fff824>
10000534: 00002737           lui a4,0x2
10000538: 80870713           addi  a4,a4,-2040 # 1808 <_fstack+0x1208>
1000053c: 00e7a023           sw  a4,0(a5)
10000540: 260007b7           lui a5,0x26000
10000544: 02878793           addi  a5,a5,40 # 26000028 <_esram_rom+0x15fff820>
10000548: 00002737           lui a4,0x2
1000054c: 80870713           addi  a4,a4,-2040 # 1808 <_fstack+0x1208>
10000550: 00e7a023           sw  a4,0(a5)
10000554: 260007b7           lui a5,0x26000
10000558: 02478793           addi  a5,a5,36 # 26000024 <_esram_rom+0x15fff81c>
1000055c: 00002737           lui a4,0x2
10000560: 80870713           addi  a4,a4,-2040 # 1808 <_fstack+0x1208>
10000564: 00e7a023           sw  a4,0(a5)
10000568: 260007b7           lui a5,0x26000
1000056c: 03c78793           addi  a5,a5,60 # 2600003c <_esram_rom+0x15fff834>
10000570: 00002737           lui a4,0x2
10000574: 80970713           addi  a4,a4,-2039 # 1809 <_fstack+0x1209>
10000578: 00e7a023           sw  a4,0(a5)
1000057c: f00067b7           lui a5,0xf0006
```

```
reg_mprj_io_31 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_30 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_29 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_28 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_27 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_26 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_25 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_24 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_23 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_22 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_21 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_20 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_19 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_18 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_17 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_16 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_15 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_14 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_13 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_12 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_11 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_10 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_9  = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_8  = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_7  = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_5  = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_4  = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_3  = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_2  = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_1  = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_0  = GPIO_MODE_USER_STD_OUTPUT;

reg_mprj_io_6  = GPIO_MODE_MGMT_STD_OUTPUT;
```

以下這段則是執行右邊中所描述的 firmware code

```
// Now, apply the configuration
reg_mprj_xfer = 1;
while (reg_mprj_xfer == 1);
```

```
1000057c: f00067b7            lui a5,0xf0006
10000580: 00100713            li  a4,1
10000584: 00e7a023            sw  a4,0(a5) # f0006000 <_esram+0xb8005e08>
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>
```

接下來依照 main function 中描述，對不同 address 進行寫入:

```
    // Configure LA probes [31:0], [127:64] as inputs to the cpu
// Configure LA probes [63:32] as outputs from the cpu
reg_la0_oenb = reg_la0_iena = 0x00000000;    // [31:0]
reg_la1_oenb = reg_la1_iena = 0xFFFFFFFF;    // [63:32]
reg_la2_oenb = reg_la2_iena = 0x00000000;    // [95:64]
reg_la3_oenb = reg_la3_iena = 0x00000000;    // [127:96]

// Flag start of the test
reg_mprj_datal = 0xAB400000;

// Set Counter value to zero through LA probes [63:32]
reg_la1_data = 0x00000000;

// Configure LA probes from [63:32] as inputs to disable counter write
reg_la1_oenb = reg_la1_iena = 0x00000000;
```

```
100005a8: f00037b7            lui a5,0xf0003
100005ac: 00c78713            addi  a4,a5,12 # f000300c <_esram+0xb8002e14>
100005b0: 00000793            li  a5,0
100005b4: 00f72023            sw  a5,0(a4)
100005b8: f0003737            lui a4,0xf0003
100005bc: 01c70713            addi  a4,a4,28 # f000301c <_esram+0xb8002e24>
100005c0: 00f72023            sw  a5,0(a4)
100005c4: f00037b7            lui a5,0xf0003
100005c8: 00878713            addi  a4,a5,8 # f0003008 <_esram+0xb8002e10>
100005cc: fff00793            li  a5,-1
100005d0: 00f72023            sw  a5,0(a4)
100005d4: f0003737            lui a4,0xf0003
100005d8: 01870713            addi  a4,a4,24 # f0003018 <_esram+0xb8002e20>
100005dc: 00f72023            sw  a5,0(a4)
100005e0: f00037b7            lui a5,0xf0003
100005e4: 00478713            addi  a4,a5,4 # f0003004 <_esram+0xb8002e0c>
100005e8: 00000793            li  a5,0
100005ec: 00f72023            sw  a5,0(a4)
100005f0: f0003737            lui a4,0xf0003
100005f4: 01470713            addi  a4,a4,20 # f0003014 <_esram+0xb8002e1c>
100005f8: 00f72023            sw  a5,0(a4)
100005fc: f0003737            lui a4,0xf0003
10000600: 00000793            li  a5,0
10000604: 00f72023            sw  a5,0(a4) # f0003000 <_esram+0xb8002e08>
10000608: f0003737            lui a4,0xf0003
1000060c: 01070713            addi  a4,a4,16 # f0003010 <_esram+0xb8002e18>
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+0xb8002e40>
1000062c: 0007a023            sw  zero,0(a5)
10000630: f00037b7            lui a5,0xf0003
10000634: 00878713            addi  a4,a5,8 # f0003008 <_esram+0xb8002e10>
10000638: 00000793            li  a5,0
1000063c: 00f72023            sw  a5,0(a4) # ab400000 <_esram+0x733ffe08>
10000640: f0003737            lui a4,0xf0003
10000644: 01870713            addi  a4,a4,24 # f0003018 <_esram+0xb8002e20>
10000648: 00f72023            sw  a5,0(a4)
```

之後呼叫 fir function，並將結果寫入 reg_mprj_datal 所對應的 address:

```
int* tmp = fir();

reg_mprj_datal = *tmp << 16;
reg_mprj_datal = *(tmp+1) << 16;
reg_mprj_datal = *(tmp+2) << 16;
reg_mprj_datal = *(tmp+3) << 16;
reg_mprj_datal = *(tmp+4) << 16;
reg_mprj_datal = *(tmp+5) << 16;
reg_mprj_datal = *(tmp+6) << 16;
reg_mprj_datal = *(tmp+7) << 16;
reg_mprj_datal = *(tmp+8) << 16;
reg_mprj_datal = *(tmp+9) << 16;
reg_mprj_datal = *(tmp+10) << 16;
```

initfir function’s assembly code:

```
38000024 <initfir>:
38000024: fe010113          addi  sp,sp,-32
38000028: 00812e23          sw  s0,28(sp)
3800002c: 02010413          addi  s0,sp,32
38000030: fe042623          sw  zero,-20(s0)
38000034: 0380006f          j 3800006c <initfir+0x48>
38000038: 05c00713          li  a4,92
3800003c: fec42783          lw  a5,-20(s0)
38000040: 00279793          slli  a5,a5,0x2
38000044: 00f707b3          add a5,a4,a5
38000048: 0007a023          sw  zero,0(a5)
3800004c: 08800713          li  a4,136
38000050: fec42783          lw  a5,-20(s0)
38000054: 00279793          slli  a5,a5,0x2
38000058: 00f707b3          add a5,a4,a5
3800005c: 0007a023          sw  zero,0(a5)
38000060: fec42783          lw  a5,-20(s0)
38000064: 00178793          addi  a5,a5,1
38000068: fef42623          sw  a5,-20(s0)
3800006c: fec42703          lw  a4,-20(s0)
38000070: 00a00793          li  a5,10
38000074: fce7d2e3          bge a5,a4,38000038 <initfir+0x14>
38000078: 00000013          nop
3800007c: 00000013          nop
38000080: 01c12403          lw  s0,28(sp)
38000084: 02010113          addi  sp,sp,32
38000088: 00008067          ret
```

fir function’s assembly code:

```
3800008c <fir>:
3800008c: fe010113          addi  sp,sp,-32
38000090: 00112e23          sw  ra,28(sp)
38000094: 00812c23          sw  s0,24(sp)
38000098: 00912a23          sw  s1,20(sp)
3800009c: 02010413          addi  s0,sp,32
380000a0: f85ff0ef          jal ra,38000024 <initfir>
380000a4: fe042623          sw  zero,-20(s0)
380000a8: 1280006f          j 380001d0 <fir+0x144>
380000ac: 00a00793          li  a5,10
380000b0: fef42423          sw  a5,-24(s0)
380000b4: 0a00006f          j 38000154 <fir+0xc8>
380000b8: fe842783          lw  a5,-24(s0)
380000bc: fff78793          addi  a5,a5,-1
380000c0: 05c00713          li  a4,92
380000c4: 00279793          slli  a5,a5,0x2
380000c8: 00f707b3          add a5,a4,a5
380000cc: 0007a703          lw  a4,0(a5)
380000d0: 05c00693          li  a3,92
380000d4: fe842783          lw  a5,-24(s0)
380000d8: 00279793          slli  a5,a5,0x2
380000dc: 00f687b3          add a5,a3,a5
380000e0: 00e7a023          sw  a4,0(a5)
380000e4: 08800713          li  a4,136
380000e8: fec42783          lw  a5,-20(s0)
380000ec: 00279793          slli  a5,a5,0x2
380000f0: 00f707b3          add a5,a4,a5
380000f4: 0007a483          lw  s1,0(a5)
380000f8: 05c00713          li  a4,92
380000fc: fe842783          lw  a5,-24(s0)
38000100: 00279793          slli  a5,a5,0x2
38000104: 00f707b3          add a5,a4,a5
38000108: 0007a683          lw  a3,0(a5)
3800010c: 00000713          li  a4,0
38000110: fe842783          lw  a5,-24(s0)
38000114: 00279793          slli  a5,a5,0x2
38000118: 00f707b3          add a5,a4,a5
3800011c: 0007a783          lw  a5,0(a5)
38000120: 00078593          mv  a1,a5
38000124: 00068513          mv  a0,a3
38000128: ed9ff0ef          jal ra,38000000 <__mulsi3>
3800012c: 00050793          mv  a5,a0
38000130: 00f48733          add a4,s1,a5
38000134: 08800693          li  a3,136
38000138: fec42783          lw  a5,-20(s0)
3800013c: 00279793          slli  a5,a5,0x2
38000140: 00f687b3          add a5,a3,a5
38000144: 00e7a023          sw  a4,0(a5)
38000148: fe842783          lw  a5,-24(s0)
3800014c: fff78793          addi  a5,a5,-1
38000150: fef42423          sw  a5,-24(s0)
38000154: fe842783          lw  a5,-24(s0)
38000158: f6f040e3          bgtz  a5,380000b8 <fir+0x2c>
3800015c: 02c00713          li  a4,44
38000160: fec42783          lw  a5,-20(s0)
38000164: 00279793          slli  a5,a5,0x2
38000168: 00f707b3          add a5,a4,a5
3800016c: 0007a703          lw  a4,0(a5)
38000170: 05c00793          li  a5,92
38000174: 00e7a023          sw  a4,0(a5)
38000178: 08800713          li  a4,136
3800017c: fec42783          lw  a5,-20(s0)
38000180: 00279793          slli  a5,a5,0x2
38000184: 00f707b3          add a5,a4,a5
38000188: 0007a483          lw  s1,0(a5)
3800018c: 05c00793          li  a5,92
38000190: 0007a703          lw  a4,0(a5)
38000194: 00000793          li  a5,0
38000198: 0007a783          lw  a5,0(a5)
3800019c: 00078593          mv  a1,a5
380001a0: 00070513          mv  a0,a4
380001a4: e5dff0ef          jal ra,38000000 <__mulsi3>
380001a8: 00050793          mv  a5,a0
380001ac: 00f48733          add a4,s1,a5
380001b0: 08800693          li  a3,136
380001b4: fec42783          lw  a5,-20(s0)
380001b8: 00279793          slli  a5,a5,0x2
380001bc: 00f687b3          add a5,a3,a5
380001c0: 00e7a023          sw  a4,0(a5)
380001c4: fec42783          lw  a5,-20(s0)
380001c8: 00178793          addi  a5,a5,1
380001cc: fef42623          sw  a5,-20(s0)
380001d0: fec42703          lw  a4,-20(s0)
380001d4: 00a00793          li  a5,10
380001d8: ece7dae3          bge a5,a4,380000ac <fir+0x20>
380001dc: 08800793          li  a5,136
380001e0: 00078513          mv  a0,a5
380001e4: 01c12083          lw  ra,28(sp)
380001e8: 01812403          lw  s0,24(sp)
380001ec: 01412483          lw  s1,20(sp)
380001f0: 02010113          addi  sp,sp,32
380001f4: 00008067          ret
```

Where 38000000 is the starting address of multiplication function.

How does it execute a multiplication in assembly code?

以下這段則為 fir.c 中使用到的 multiplication 的 assembly code block，這個 function 用 a0 與 a1 作為 function 的 argument 將參數給進來，其中 a0 作為被乘數先給入 a2，再將 a0 歸零用來累加，而 a1 會透過不斷掃描 LSB，若為 1 則進行累加 (add a2 to a0)，若為 0 則不累加，並在掃描後進行 shift (a1 & a2)，直到 a1 值為零後即完成累加。這裡利用 shift right 及 and operation with 1 來確定乘數中那些位置為 1，即為會產生乘積的位置。利用 shift left 及 add 來進行乘積的進位及累加。以拆解的方式完成乘法運算。

```
Disassembly of section .mprjram:

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
```

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

```
MEMORY {
        vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
        dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
        dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
        flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
        mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000
        mprjram : ORIGIN = 0x38000000, LENGTH = 0x00400000
        hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
        csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
}
```

Allocate 4MB memory for mapping but notice that the size of BRAM in user_project is limited

The space allocated for the user project is based from 0x38000000 to 0x383FFFFF.
The space required to allocate for our fir firmware code is from 0x38000000 to 0x380001F7, total 504 bytes.

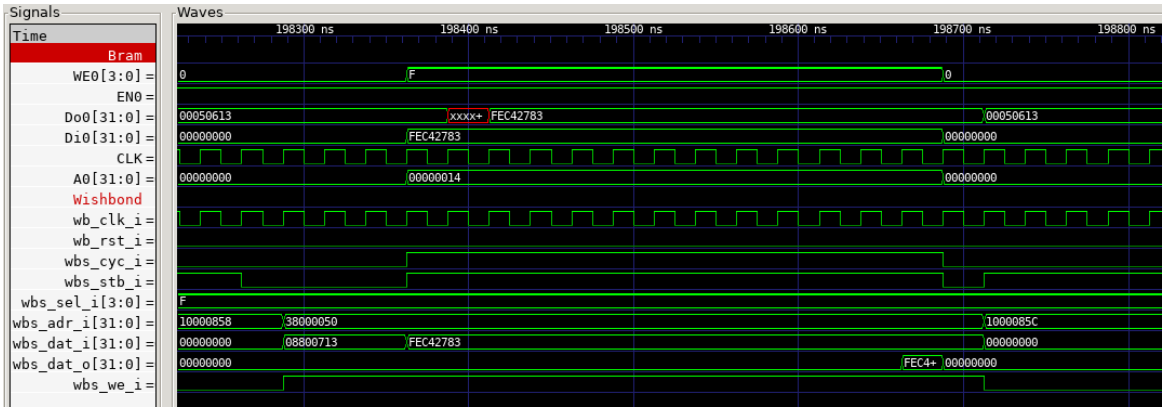
```
38000150: fef42423            sw  a5,-24(s0)
38000154: fe842783            lw  a5,-24(s0)
38000158: f6f040e3            bgtz  a5,380000b8 <fir+0x2c>
3800015c: 02c00713            li  a4,44
38000160: fec42783            lw  a5,-20(s0)
38000164: 00279793            slli  a5,a5,0x2
38000168: 00f707b3            add a5,a4,a5
3800016c: 0007a703            lw  a4,0(a5)
38000170: 05c00793            li  a5,92
38000174: 00e7a023            sw  a4,0(a5)
38000178: 08800713            li  a4,136
3800017c: fec42783            lw  a5,-20(s0)
38000180: 00279793            slli  a5,a5,0x2
38000184: 00f707b3            add a5,a4,a5
38000188: 0007a483            lw  s1,0(a5)
3800018c: 05c00793            li  a5,92
38000190: 0007a703            lw  a4,0(a5)
38000194: 00000793            li  a5,0
38000198: 0007a783            lw  a5,0(a5)
3800019c: 00078593            mv  a1,a5
380001a0: 00070513            mv  a0,a4
380001a4: e5dff0ef            jal ra,38000000 <__mulsi3>
380001a8: 00050793            mv  a5,a0
380001ac: 00f48733            add a4,s1,a5
380001b0: 08800693            li  a3,136
380001b4: fec42783            lw  a5,-20(s0)
380001b8: 00279793            slli  a5,a5,0x2
380001bc: 00f687b3            add a5,a3,a5
380001c0: 00e7a023            sw  a4,0(a5)
380001c4: fec42783            lw  a5,-20(s0)
380001c8: 00178793            addi  a5,a5,1
380001cc: fef42623            sw  a5,-20(s0)
380001d0: fec42703            lw  a4,-20(s0)
380001d4: 00a00793            li  a5,10
380001d8: ece7dae3            bge a5,a4,380000ac <fir+0x20>
380001dc: 08800793            li  a5,136
380001e0: 00078513            mv  a0,a5
380001e4: 01c12083            lw  ra,28(sp)
380001e8: 01812403            lw  s0,24(sp)
380001ec: 01412483            lw  s1,20(sp)
380001f0: 02010113            addi  sp,sp,32
380001f4: 00008067            ret

Disassembly of section .riscv.attributes:
```

If we consider the main function defined in counter_la_fir.c, the space allocated for the main firmware code is from 0x100002ec to 0x100007ab, total 1216 bytes.
Hence, the total space for the firmware code is 1216 + 504 = 1720 bytes.

```
100002ec <main>:
100002ec: fe010113          addi  sp,sp,-32
100002f0: 00112e23          sw  ra,28(sp)
100002f4: 00812c23          sw  s0,24(sp)
100002f8: 02010413          addi  s0,sp,32
100002fc: 260007b7          lui a5,0x26000
10000300: 0a078793          addi  a5,a5,160 # 260000a0 <_esram_rom+0x15fff898>
10000304: 00002737          lui a4,0x2
10000308: 80970713          addi  a4,a4,-2039 # 1809 <_fstack+0x1209>
1000030c: 00e7a023          sw  a4,0(a5)
10000310: 260007b7          lui a5,0x26000
10000314: 09c78793          addi  a5,a5,156 # 2600009c <_esram_rom+0x15fff894>
```

```
10000778: 01079713          slli  a4,a5,0x10
1000077c: 260007b7          lui a5,0x26000
10000780: 00c78793          addi  a5,a5,12 # 2600000c <_esram_rom+0x15fff804>
10000784: 00e7a023          sw  a4,0(a5)
10000788: 260007b7          lui a5,0x26000
1000078c: 00c78793          addi  a5,a5,12 # 2600000c <_esram_rom+0x15fff804>
10000790: ab510737          lui a4,0xab510
10000794: 00e7a023          sw  a4,0(a5)
10000798: 00000013          nop
1000079c: 01c12083          lw  ra,28(sp)
100007a0: 01812403          lw  s0,24(sp)
100007a4: 02010113          addi  sp,sp,32
100007a8: 00008067          ret
```

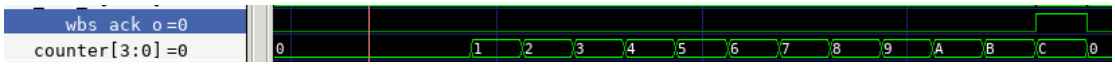
Waveform of user-bram and Wishbond interface:



When wbs_cyc_i and wbs_stb_i pulled high, the signals at the wishbond ports are valid. Wbs_we_i determines whether the transmission is read (wbs_we_i = 0) or write (wbs_we_i = 1). Wbs_dat_i is the data being written in and the wbs_dat_o is the data

being read out. Wbs_sel_i indicates where valid data is placed on the [wbs_dat_i] signal array during WRITE cycles, and where it should be present on the [wbs_dat_o] signal array during READ cycles. Wbs_adr_i indicates where the read or write transmission take place. Wbs_ack_o is the finish response which stands for the end of the transaction sent back from the wishbond slave. Notice that the address presented at wbs_adr_i is not equal to the address to the bram. This is because that (1) the address sent in by wishbond bus including both device address (more significant bits) and internal register address (less significant bits) while the address given to the bram required internal register address only. (2) wishbond address is byte address while bram address is word address. Also due to (2), wbs_we_i will extended to 4 bits write_enable signal for the bram with the information in wbs_sel_i.

FSM:



When the counter stays at 0, the state is idle, meaning that no transferring occurs. When counter start counting, the state is processing. When the counter is at DELAY+2, the state is response, and the wishbond slave will return ack and data to the master. The state will go back to idle after response.