SOC Design Lab4-1 Report

National Yang Ming Chiao Tung University
311511022 邱政岡
412010020 高振翔
311514037 蔡宇冠

1. Introduction

1.1 Objective

- ➢ Understand the relation between the risc-v of management core、user project and testbench
- ➢ Use the wishbone to communicate with user project
- ➢ Use MPRJ to set checkbits and transfer data to testbench
- ➢ Learn how to write firmware code and compile by gcc toolchain
- ➢ Learn how to push compiled firmware code to the system
- ➢ Ust gtkwave to check waveform

1.2 Overview

In this lab, we need to program fir firmware which running on the risc-v of management core. It will access the SRAM which is implement by BRAM. The system will copy the firmware code from spiflash to SRAM. We set the BRAM size to 4KB (reg[31:0] RAM[0:2**N-1, N=10) and place the BRAM in the user project named exmem. Finally, we use testbench to wait checkbit flag from mprj and print out the result from firmware code.

2. Firmware

In the firmware code, we set the code placement in the fir.c file by "`section(".mprjram")`" and describe the fir behavior by two for loop. The counter_la_fir.c set the mprj config and push fir status(0xAB40、0XAB51) and result by mprj_datal to the testbench.

2.1 FIR function

2.1.1 Address allocate

Define the attribute of the function. The compiler will allocate this code to the defined section.

```
int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir() {
    ...
}
```

The address is defined in sections.lds file.

```
MEMORY {
    ...
    mprjram : ORIGIN = 0x38000000, LENGTH = 0x00400000
    ...
}
```

2.1.2 Assembly code

After programming the firmware code, we use the "`riscv32-unknown-elf-gcc`" command to compile it and use "`risc32-unknown-elf-objcopy`" to transfer elf to a hex file.
To see the assembly code, use the "`riscv32-unknown-elf-objdump`" command.
The address of the last instruction in BRAM is 0x38000160. The usage of BRAM is 352 bytes.

```
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
...
38000074 <fir>:
38000074:   fe010113                addi    sp,sp,-32
...
38000160:   00008067                ret
```

A function named "__mulsi3" is generated by compiler to do the multiplication. The corresponding c code is like the following.

```
unsigned int __mulsi3 (unsigned int a, unsigned int b) {
    unsigned int r = 0;
    while (a){
        if (a & 1) r += b;
```

```
            a >>= 1;
            b <<= 1;
        }
        return r;
    }
```
The function checks the value of a bit by bit from LSB to MSB. If the value is one, accumulate b to r which is initialized to zero. Then shift the b value right after checking a bit of a.

3. Interface Between BRAM and Wishbone

The communication between risc-v and user project is by wishbone. We need to follow the protocol to respond to the ack signal after 10 cycles when the *wb_cyc_i* signal rises and interconnects to the BRAM interface.

3.1 Design

3.1.1 Defined delay

```
    always @(posedge clk) begin
        if (rst) cnt <= 'd0;
        else begin
            if (wbs_ack_o) cnt <= 'd0;
            else if (wbs_stb_i & wbs_cyc_i) cnt <= cnt + 1'b1;
        end
    end
    assign wbs_ack_o = (cnt == DELAYS+1);
```
When both *wbs_stb_i* and *wbs_cyc_i* are asserted, start the counter. If the counter value equals the delay time, assert the EN0 signal in BRAM. The data will be read at the next clock posedge, so the *wb_ack_o* is asserted when the counter value equals delay+1.

3.1.2 BRAM port connection

```
    bram user_bram (
        .CLK(clk),
        .WE0(wbs_sel_i & {4{wbs_we_i}}),
        .EN0((cnt==DELAYS) & wbs_stb_i & wbs_cyc_i),
        .Di0(wbs_dat_i),
        .Do0(wbs_dat_o),
        .A0 (wbs_adr_i >> 2)
    );
```
EN0 is asserted only when the *wbs_stb_i* and *wbs_cyc_i* are holden some given

cycles.

A0 is shifted by two bits because BRAM uses the word address that differs from wishbone.

WE0 is determined by *wb_sel_i* and *wbs_we_i*. *wbs_we_i* represents the write operation, and *wbs_sel_i* represents which byte should be written.

Di0 and Do0 directly connect the wishbone input/output data.

3.2 Simulation

3.2.1 Testbench

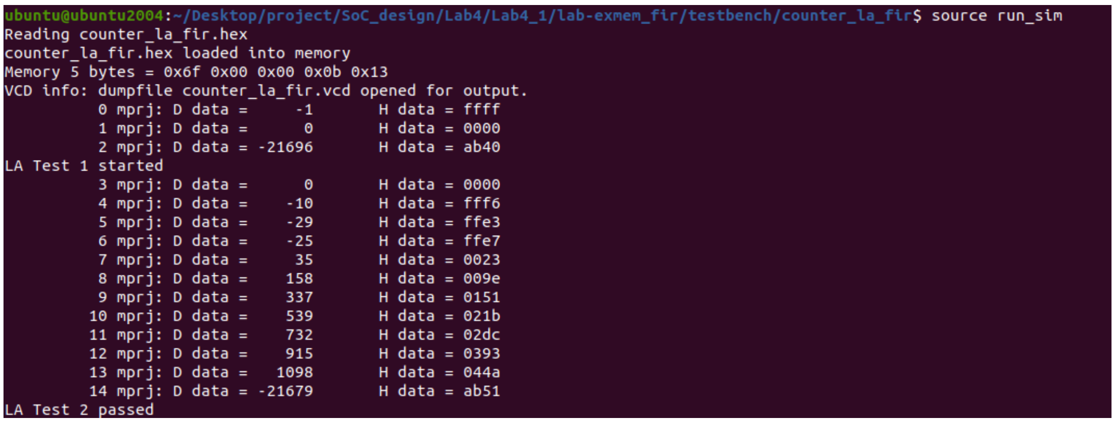
The testbench will put the compiled hex file in spiflash in order to be executed by the RISC-V CPU in Caravel SOC.

```
        spiflash #(
                .FILENAME("counter_la_fir.hex")
        ) spiflash (
                .csb(flash_csb),
                .clk(flash_clk),
                .io0(flash_io0),
                .io1(flash_io1),
                .io2(),                 // not used
                .io3()                  // not used
        );
```

The following command in testbench can display the output of FIR.

```
    wait(checkbits != pre_checkbits);
    $display("%d mprj: D data = %d      H data = %h", i, checkbits,
checkbits);
    i = i + 1 ;
    pre_checkbits = checkbits;
```

3.2.2 Simulation result



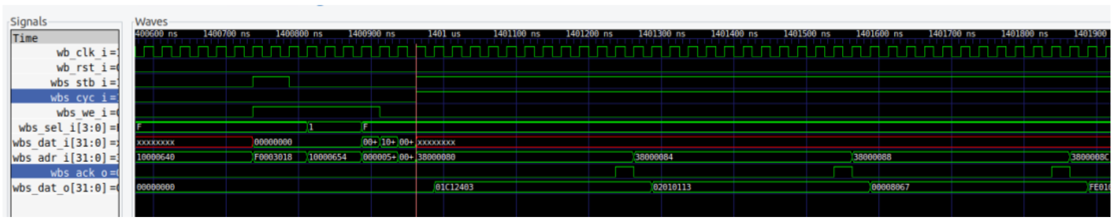
```
ubuntu@ubuntu2004:~/Desktop/project/SoC_design/Lab4/Lab4_1/lab-exmem_fir/testbench/counter_la_fir$ source run_sim
Reading counter_la_fir.hex
counter_la_fir.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_fir.vcd opened for output.
          0 mprj: D data =     -1       H data = ffff
          1 mprj: D data =      0       H data = 0000
          2 mprj: D data = -21696       H data = ab40
LA Test 1 started
          3 mprj: D data =      0       H data = 0000
          4 mprj: D data =    -10       H data = fff6
          5 mprj: D data =    -29       H data = ffe3
          6 mprj: D data =    -25       H data = ffe7
          7 mprj: D data =     35       H data = 0023
          8 mprj: D data =    158       H data = 009e
          9 mprj: D data =    337       H data = 0151
         10 mprj: D data =    539       H data = 021b
         11 mprj: D data =    732       H data = 02dc
         12 mprj: D data =    915       H data = 0393
         13 mprj: D data =   1098       H data = 044a
         14 mprj: D data = -21679       H data = ab51
LA Test 2 passed
```

3.3 Waveform

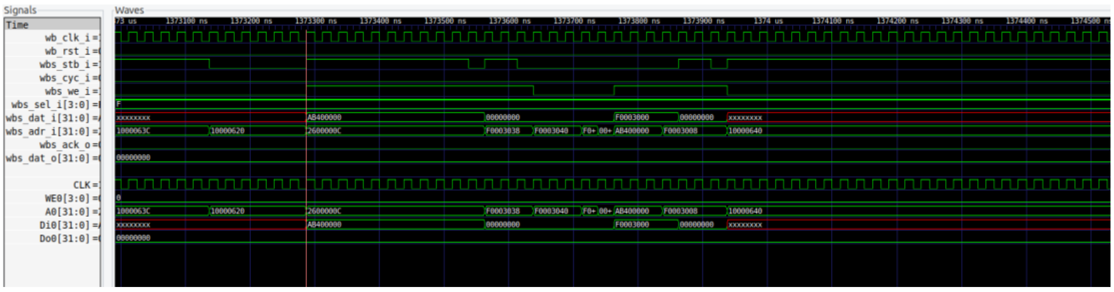
After the simulation, we check the status of fir and fir result by log from testbench. The testbench will dump waveform of the caraval soc. We can use gtkwave to observe the signal in the system and we can save the signal profile to waveform.gtk

3.3.1 Read operation waveform

The wishbone waveform of ack delay 10 cycles when *wb_cyc_*i rise.

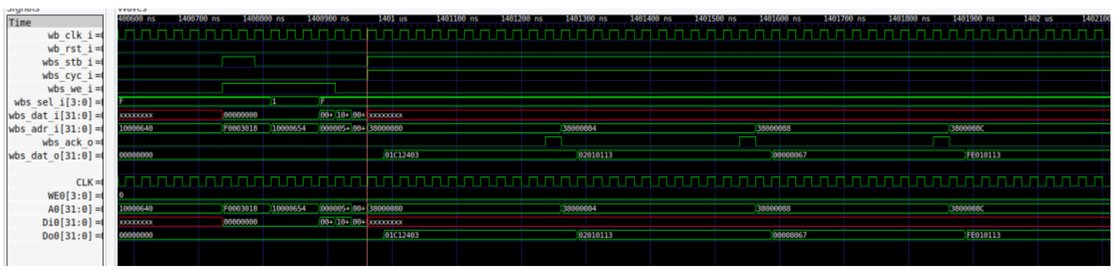


Wishbone write BRAM data.



3.3.2 Write operation waveform

Wishbone read BRAM data.


Time
wb_clk_i
wb_rst_i
wbs_stb_i
wbs_cyc_i
wbs_we_i
wbs_sel_i[3:0]
wbs_dat_i[31:0]
wbs_adr_i[31:0]
wbs_ack_o
wbs_dat_o[31:0]
CLK
WE0[3:0]
A0[31:0]
Di0[31:0]
Do0[31:0]