SOC Design

Lab4-1 Execute Code in User Memory

Content

- Prepare firmware code & RTL
  - FIR in C code
  - Firmware management in main()
  - Linker for address arrangement
  - Design your BRAM in user_project
  - Compilation
- Synthesis & Verification
- Reference
  - https://github.com/bol-edu/caravel-soc_fpga-lab/tree/main/lab-exmem_fir

FIR in C code

- Generate data in header file
- In lab-exmem_fir/testbench/fir.h

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

Defined by yourself

FIR in C code

- In lab-exmem_fir/testbench/fir.c

```
1   #include "fir.h"
2
3   void __attribute__ ( ( section ( ".mprjram" ) ) ) initfir() {
4           //initial your fir
5   }
6
7   int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir(){
8           initfir();
9           //write your fir
10          return outputsignal;
11  }
12
```

The function is located at section “mprjram”. Don’t modify it.

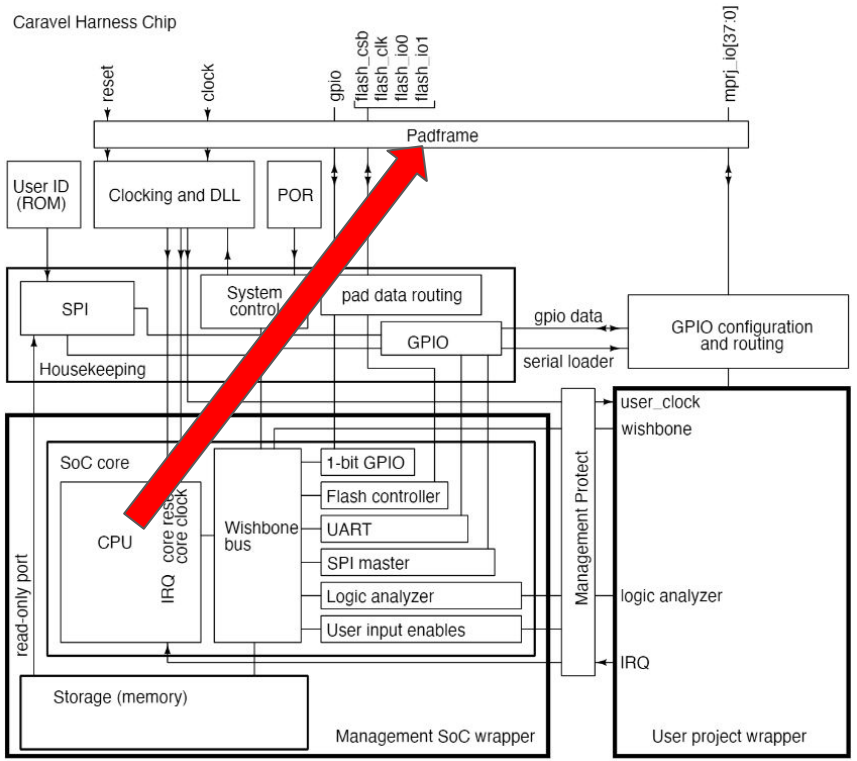
Firmware management in main()

- In lab-exmem_fir/testbench/counter_la_fir.c

```
63          reg_mprj_io_31 = GPIO_MODE_MGMT_STD_OUTPUT;
64          reg_mprj_io_30 = GPIO_MODE_MGMT_STD_OUTPUT;
65          reg_mprj_io_29 = GPIO_MODE_MGMT_STD_OUTPUT;
66          reg_mprj_io_28 = GPIO_MODE_MGMT_STD_OUTPUT;
67          reg_mprj_io_27 = GPIO_MODE_MGMT_STD_OUTPUT;
68          reg_mprj_io_26 = GPIO_MODE_MGMT_STD_OUTPUT;
69          reg_mprj_io_25 = GPIO_MODE_MGMT_STD_OUTPUT;
70          reg_mprj_io_24 = GPIO_MODE_MGMT_STD_OUTPUT;
71          reg_mprj_io_23 = GPIO_MODE_MGMT_STD_OUTPUT;
72          reg_mprj_io_22 = GPIO_MODE_MGMT_STD_OUTPUT;
73          reg_mprj_io_21 = GPIO_MODE_MGMT_STD_OUTPUT;
74          reg_mprj_io_20 = GPIO_MODE_MGMT_STD_OUTPUT;
75          reg_mprj_io_19 = GPIO_MODE_MGMT_STD_OUTPUT;
76          reg_mprj_io_18 = GPIO_MODE_MGMT_STD_OUTPUT;
77          reg_mprj_io_17 = GPIO_MODE_MGMT_STD_OUTPUT;
78          reg_mprj_io_16 = GPIO_MODE_MGMT_STD_OUTPUT;
```

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

Setting IO pad. CPU will keep idle until reg_mprj_xfer = 0



Linker for address arrangement

- In lab-exmem_fir/firmware/section.lds

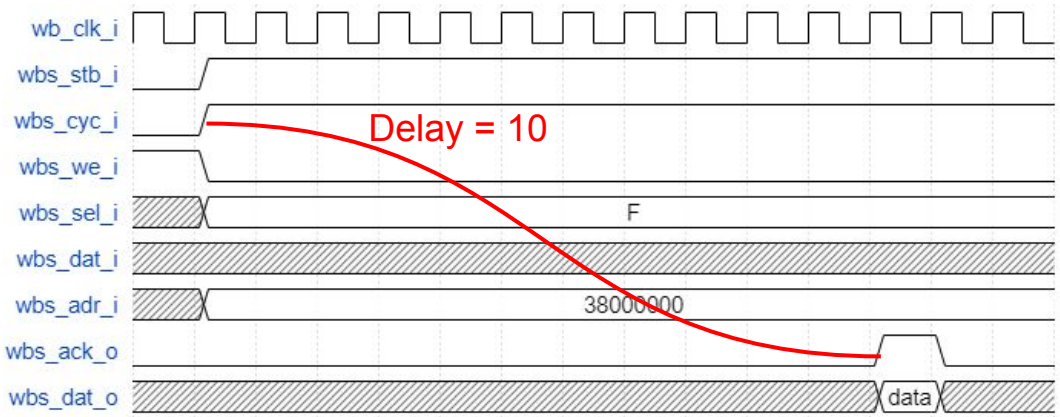
```
11    MEMORY {
12            vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
13            dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
14            dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
15            flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
16            mprj : ORIGIN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

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

Design your BRAM in user_project

- In lab-exmem_fir/rtl/user_proj_example.counter.v
  - Design the controller connected with wishbone bus
  - Response after the parameter Delay

```
48      input wb_clk_i,
49      input wb_rst_i,
50      input wbs_stb_i,
51      input wbs_cyc_i,
52      input wbs_we_i,
53      input [3:0] wbs_sel_i,
54      input [31:0] wbs_dat_i,
55      input [31:0] wbs_adr_i,
56      output wbs_ack_o,
57      output [31:0] wbs_dat_o,
```



Read BRAM

Design your BRAM in user_project

- In lab-exmem_fir/rtl/bram.v
  - Estimate the required size of BRAM

```
17      // Define the size of BRAM
18      parameter N = ;
19      (* ram_style = "block" *) reg [31:0] RAM[0:2**N-1];
20
21
22      always @(posedge CLK)
23          if(EN0) begin
24              Do0 <= RAM[A0[N-1:0]];
25              if(WE0[0]) RAM[A0[N-1:0]][7:0] <= Di0[7:0];
26              if(WE0[1]) RAM[A0[N-1:0]][15:8] <= Di0[15:8];
27              if(WE0[2]) RAM[A0[N-1:0]][23:16] <= Di0[23:16];
28              if(WE0[3]) RAM[A0[N-1:0]][31:24] <= Di0[31:24];
29          end
30          else
31              Do0 <= 32'b0;
```

Compilation

- Given script to compile
  - ```
    riscv32-unknown-elf-gcc -I../../firmware -o counter_la_fir.elf ..
    ```
- Transform .elf to .hex
  - ```
    riscv32-unknown-elf-objcopy -O verilog counter_la_fir.elf counter_la_fir.hex
    ```
- Export assembly code for debugging
  - ```
    riscv32-unknown-elf-objdump -D counter_la_fir.elf > counter_la_fir.out
    ```
    ```

Compilation

- Simulate by xsim after compiling C code
- Note that .hex is directly written on testbench
  - ```
    242             spiflash #(
    243                     .FILENAME("counter_la_fir.hex")
    244             ) spiflash (
    245                     .csb(flash_csb),
    246                     .clk(flash_clk),
    247                     .io0(flash_io0),
    248                     .io1(flash_io1),
    249                     .io2(),                 // not used
    250                     .io3()                  // not used
    251             );
    ```
    ```

In this Lab you need to …

- Write FIR C code
  - fir.c
  - fir.h
- Write RTL in user_project
  - Controller for delayed response and communication between BRAM and wishbone bus