Bridge of Life Education

SOC Design

Caravel Testbench & Lab4-2 Explained

Jiin Lai

Objectives

Understand Caravel Testbench Structure, particularly, the relationship among

1. Firmware – the c code runs on RISC-V
2. Testbench – the verilog top module which composes
    1. Instance of Caravel design which includes user project
    2. Spiflash which is loaded with firmware .hex file
    3. Verilog code to interact with firmware/user-project through mprj pins
3. User Projects

Caravel Background Knowledge

Caravel SOC Introduction - Video

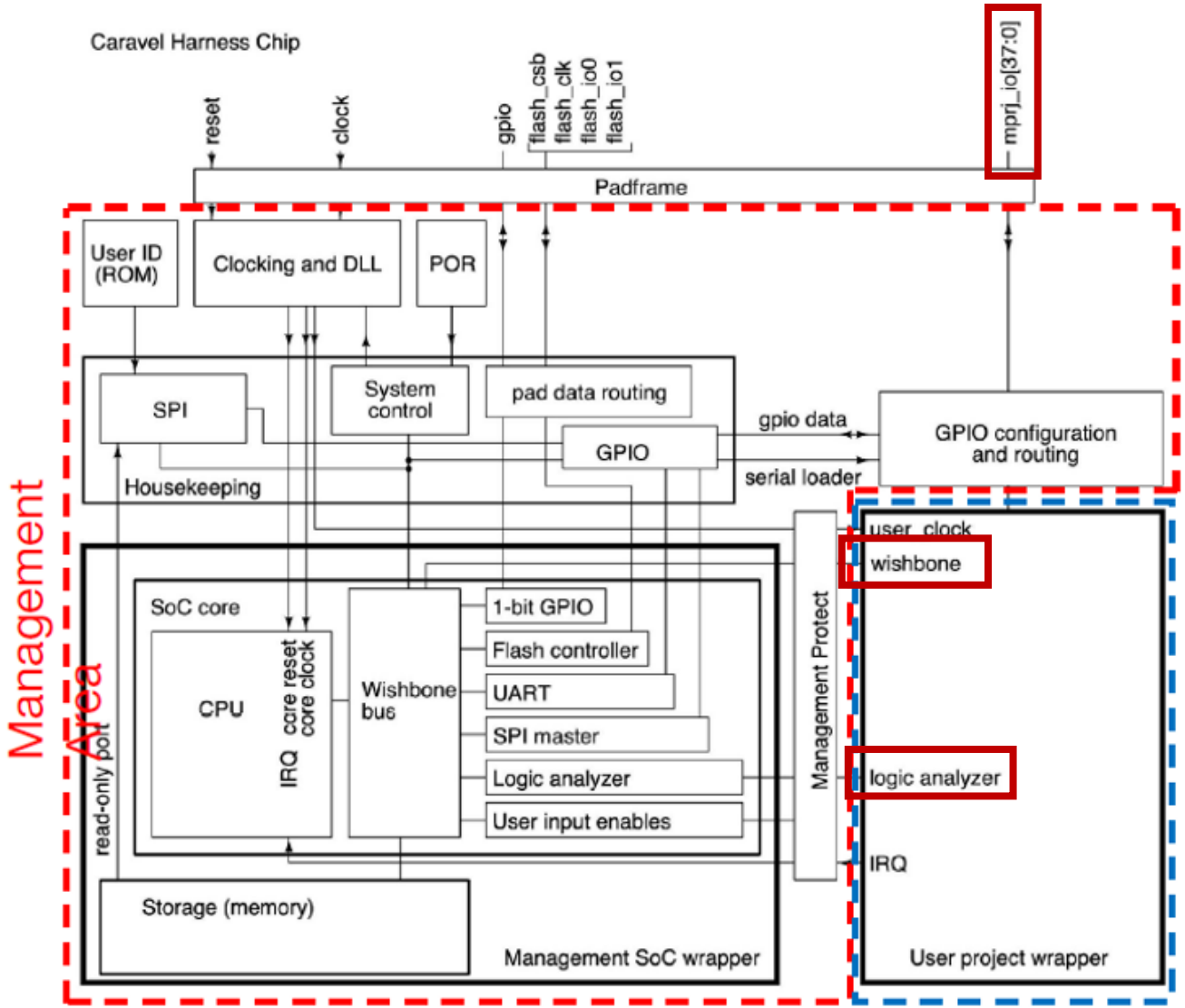
Topics	Author	duration	Video
System Block Diagram			
Reset POR	Tony		
Management Project Area	Tony		https://youtu.be/hbISphnvVYg
DLL, Configuration SPI	Tony	23:04	
Housekeep SPI	Willy		
GPIO	Willy	32:55	https://youtu.be/Vw3TGc-YV8E
SPI	Willy		
Memory-mapped IO	Willy		
Counter/Timer/UART	Hurry	31:36	https://youtu.be/-o87eNkqmPo
Wishbone	Josh	11:37	https://youtu.be/Xvk4jCB9l7U
IRQ	Josh	10:26	https://youtu.be/G3oT0DJfZMk
SRM	Josh	4:56	https://youtu.be/X8sMMfrXKac
User Project Interface	Josh		
Testbench	Josh	12:42	https://youtu.be/0neIx5DOK1g
Firmware	Josh		

Caravel SOC ppt: https://github.com/bol-edu/caravel-soc_fpga-lab/tree/main/caravel-soc-ppt

Reference - Must Study

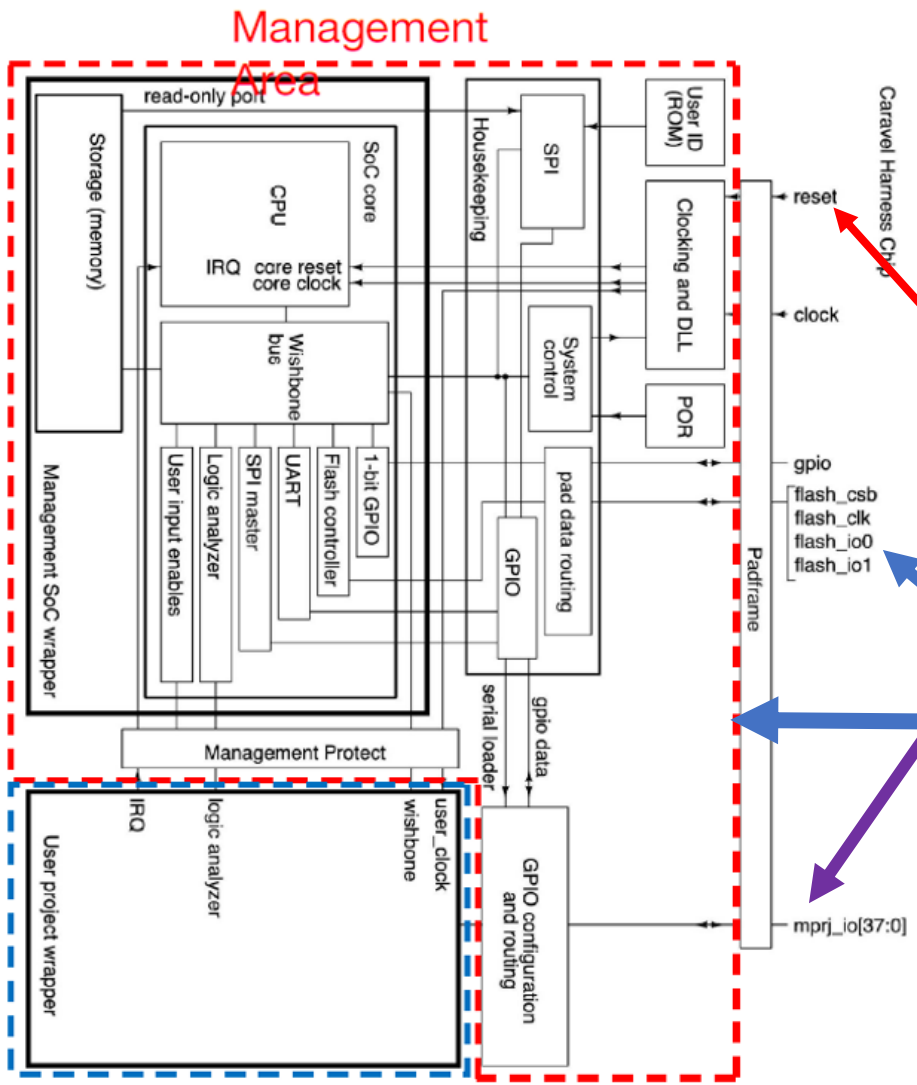
- Firmware
  - ppt: https://github.com/bol-edu/caravel-soc_fpga-lab/blob/main/caravel-soc-ppt/caravel-intr-sram-wb-usrprj-firmware_josh.pdf
  - Video: https://www.youtube.com/watch?v=0neIx5DOK1g&list=PL5CoDA0gtOHVeF4FkotwDY-buLVIoeGzf&index=1&t=5s
- GPIO / MMIO
  - ppt: https://github.com/bol-edu/caravel-soc_fpga-lab/blob/main/caravel-soc-ppt/caravel-hk-gpio-spi-mmio_willy.pdf
  - Video: https://youtu.be/Vw3TGc-YV8E

Caravel Harness



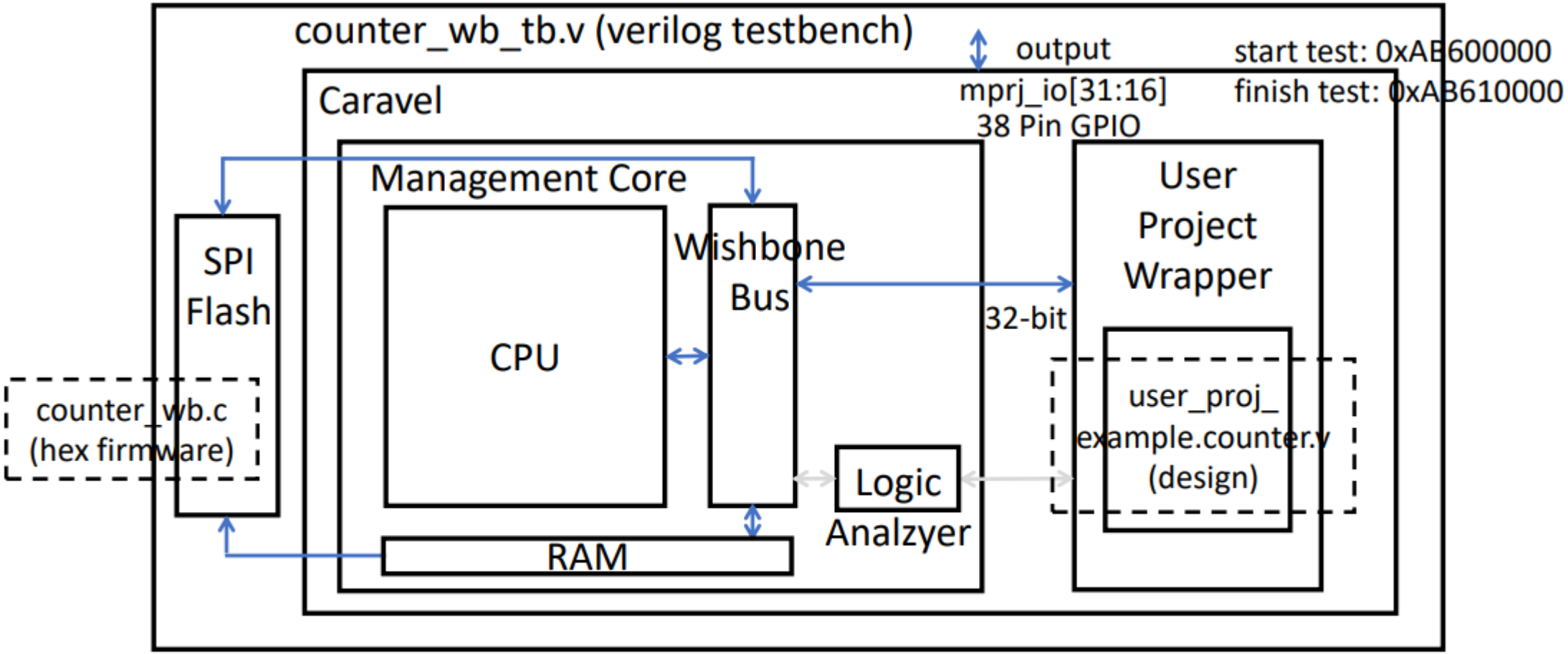
Caravel Simulation Verification System

Verilog Testbench




```
module counter_tb;
Initial beigin
 wait(checkbits == 16'hAB40);
 wait(checkbits = ..
end

Initial begin
  RSB <= 1'b0;
  RSB <= 1'b1;
end

caravel uut(    );

spiflash #( .FILENAME("counter_la.hex"} spiflash ( );

tbuart tburat (     );

endmodule
```

Counter - WB



counter_wb_tb.v: https://github.com/bol-edu/caravel-soc/blob/main/testbench/counter_wb/counter_wb_tb.v
counter_wb.c: https://github.com/bol-edu/caravel-soc/blob/main/testbench/counter_wb/counter_wb.c
user_proj_example.counter.v: https://github.com/bol-edu/caravel-soc/blob/main/rtl/user/user_proj_example.counter.v

https://github.com/bol-edu/caravel-soc/files/11052078/caravel-soc_testbench.firmwave.pdf

Caravel Memory Map IO (defs.h)

Address (bytes)	Function
0x 00 00 00 00	Flash SPI / overlaid SRAM (4k words) start of memory block
0x 00 00 3f ff	End of SRAM
0x 10 00 00 00	Flash SPI start of program block. Program to run starts here on reset (see :ref:`SPI Flash initialization <initial_spi_instruction_sequence>`).
0x 10 ff ff ff	Maximum SPI flash addressable space (16MB) with QSPI 3-byte addressing
0x 1f ff ff ff	Maximum SPI flash addressable space (32MB)
0x 20 00 00 00	UART clock divider select (:ref:`reg_uart_clkdiv`)
0x 20 00 00 04	UART data (:ref:`reg_uart_data`)
0x 20 00 00 08	UART enable (:ref:`reg_uart_enable`)
0x 21 00 00 00	GPIO input/output (bit 16/bit 0) (:ref:`reg_gpio_data`). 1 general-purpose digital, management area only.
0x 21 00 00 04	GPIO output enable (:ref:`reg_gpio_ena`)
0x 21 00 00 08	GPIO pullup enable (:ref:`reg_gpio_pu`)
0x 21 00 00 0c	GPIO pulldown enable (:ref:`reg_gpio_pd`)
0x 22 00 00 00	Counter/Timer 0 configuration register (:ref:`reg_timer0_config`)
0x 22 00 00 04	Counter/Timer 0 current value (:ref:`reg_timer0_value`)
0x 22 00 00 08	Counter/Timer 0 reset value (:ref:`reg_timer0_data`)
0x 23 00 00 00	Counter/Timer 1 configuration register (:ref:`reg_timer1_config`)
0x 23 00 00 04	Counter/Timer 1 current value (:ref:`reg_timer1_value`)
0x 23 00 00 08	Counter/Timer 1 reset value (:ref:`reg_timer1_data`)
0x 24 00 00 00	SPI controller configuration register (:ref:`reg_spi_config`)
0x 24 00 00 08	SPI controller data register (:ref:`reg_spi_data`)
0x 25 00 00 00	Logic Analyzer Data 0
0x 25 00 00 04	Logic Analyzer Data 1
0x 25 00 00 08	Logic Analyzer Data 2
0x 25 00 00 0c	Logic Analyzer Data 3
0x 25 00 00 10	Logic Analyzer Enable 0
0x 25 00 00 14	Logic Analyzer Enable 1
0x 25 00 00 18	Logic Analyzer Enable 2
0x 25 00 00 1c	Logic Analyzer Enable 3
0x 26 00 00 00	User project area GPIO data (L)
0x 26 00 00 04	User project area GPIO data (H)
0x 26 00 00 08	User project area GPIO data transfer (bit 0, auto-zeroing)
0x 26 00 00 0c	User project area GPIO `mprj_io[0]` configure
...	...
0x 26 00 00 a0	User project area GPIO `mprj_io[37]` configure
0x 26 00 00 a4	User project area GPIO power[0] configure (currently undefined/unused)
...	...
0x 26 00 00 b4	User project area GPIO power[3] configure (currently undefined/unused)
0x 2d 00 00 00	QSPI controller config (:ref:`reg_spictrl`)
0x 2f 00 00 00	PLL clock output destination (:ref:`reg_pll_out_dest`)
0x 2f 00 00 04	Trap output destination (:ref:`reg_trap_out_dest`)
0x 2f 00 00 08	IRQ 7 input source (:ref:`reg_irq7_source`)
0x 30 00 00 00	User area base. A user project may define additional Wishbone responder modules starting at this address.
0x 80 00 00 00	QSPI controller
0x 90 00 00 00	:ref:`storage-area-sram`
0x a0 00 00 00	Any responder 1
0x b0 00 00 00	Any responder 2

https://github.com/bol-edu/caravel-soc/blob/main/firmware/defs.h

https://github.com/efabless/caravel/blob/main/docs/pdf/memory_map.pdf

firmware/caravel.h - MMIO

```
#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)

#define reg_mprj_io_7 (*(volatile uint32_t*)0x26000040)
#define reg_mprj_io_8 (*(volatile uint32_t*)0x26000044)
#define reg_mprj_io_9 (*(volatile uint32_t*)0x26000048)
#define reg_mprj_io_10 (*(volatile uint32_t*)0x2600004c)

#define reg_mprj_io_11 (*(volatile uint32_t*)0x26000050)
#define reg_mprj_io_12 (*(volatile uint32_t*)0x26000054)
#define reg_mprj_io_13 (*(volatile uint32_t*)0x26000058)
#define reg_mprj_io_14 (*(volatile uint32_t*)0x2600005c)

#define reg_mprj_io_15 (*(volatile uint32_t*)0x26000060)
#define reg_mprj_io_16 (*(volatile uint32_t*)0x26000064)
#define reg_mprj_io_17 (*(volatile uint32_t*)0x26000068)
#define reg_mprj_io_18 (*(volatile uint32_t*)0x2600006c)

#define reg_mprj_io_19 (*(volatile uint32_t*)0x26000070)
#define reg_mprj_io_20 (*(volatile uint32_t*)0x26000074)
#define reg_mprj_io_21 (*(volatile uint32_t*)0x26000078)
#define reg_mprj_io_22 (*(volatile uint32_t*)0x2600007c)
```

```
#define reg_mprj_io_23 (*(volatile uint32_t*)0x26000080)
#define reg_mprj_io_24 (*(volatile uint32_t*)0x26000084)
#define reg_mprj_io_25 (*(volatile uint32_t*)0x26000088)
#define reg_mprj_io_26 (*(volatile uint32_t*)0x2600008c)

#define reg_mprj_io_27 (*(volatile uint32_t*)0x26000090)
#define reg_mprj_io_28 (*(volatile uint32_t*)0x26000094)
#define reg_mprj_io_29 (*(volatile uint32_t*)0x26000098)
#define reg_mprj_io_30 (*(volatile uint32_t*)0x2600009c)
#define reg_mprj_io_31 (*(volatile uint32_t*)0x260000a0)

#define reg_mprj_io_32 (*(volatile uint32_t*)0x260000a4)
#define reg_mprj_io_33 (*(volatile uint32_t*)0x260000a8)
#define reg_mprj_io_34 (*(volatile uint32_t*)0x260000ac)
#define reg_mprj_io_35 (*(volatile uint32_t*)0x260000b0)
#define reg_mprj_io_36 (*(volatile uint32_t*)0x260000b4)
#define reg_mprj_io_37 (*(volatile uint32_t*)0x260000b8)
```

rtl/header/user_defines.v

The power-on configuration for GPIO 0 to 4 is fixed and cannot be modified
(allowing the SPI and debug to always be accessible unless overridden by a flash program)

```
`define USER_CONFIG_GPIO_5_INIT  `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_6_INIT  `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_7_INIT  `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_8_INIT  `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_9_INIT  `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_10_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_11_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_12_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_13_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
```

Mask bit	Default	Description
10-12	001	Digital mode
9	TODO	input voltage trip point select
8	0	slow slew (0 - fast slew, 1 - slow slew)
7	TODO	analog bus polarity
6	TODO	analog bus select
5	TODO	analog bus enable (0 - disabled, 1 - enabled)
4	TODO	IB mode select
3	0	input disable (0 - input enabled, 1 - input disabled)
2	0	hold override value (value is the value during hold mode)
1	1	output disable (0 - output enabled, 1 - output disabled)
0	1	management control enable (0 - user control, 1 - management control)

```
// Configurations of GPIO 14 to 24 are used on caravel but not caravan.
`define USER_CONFIG_GPIO_14_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_15_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_16_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_17_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_18_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_19_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_20_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_21_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_22_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_23_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_24_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL

`define USER_CONFIG_GPIO_25_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_26_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_27_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_28_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_29_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_30_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_31_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_32_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_33_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_34_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_35_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_36_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
`define USER_CONFIG_GPIO_37_INIT `GPIO_MODE_MGMT_STD_INPUT_NOPULL
```

testbench/counter_la/counter_la.c

```
void main()
{
...
        reg_mprj_io_31 = 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_0  = GPIO_MODE_USER_STD_OUTPUT;

        reg_mprj_io_6  = GPIO_MODE_MGMT_STD_OUTPUT;
...
    // Now, apply the configuration
    reg_mprj_xfer = 1;
    while (reg_mprj_xfer == 1);
...

    // Flag start of the test
    reg_mprj_datal = 0xAB400000;
...
    reg_mprj_datal = 0xAB510000;
}
```

Firmware : counter_wb (defs.h, caravel.h)

```
#include <defs.h>
#include <stub.c>
#define reg_mprj_slave (*(volatile uint32_t*) 0x30000000)
```

```
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;

 /* Apply configuration */
reg_mprj_xfer = 1;
while (reg_mprj_xfer == 1);
```

```
// Flag start of the test
  reg_mprj_datal = 0xAB600000;

  reg_mprj_slave = 0x00002710;
  reg_mprj_datal = 0xAB610000;
  if (reg_mprj_slave == 0x2B3D) {
     reg_mprj_datal = 0xAB610000;
  }
```

caravel.h

```
#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)
```

firmware: https://github.com/bol-edu/caravel-soc/blob/main/testbench/counter_wb/counter_wb.c
https://github.com/bol-edu/caravel-soc/blob/main/firmware/defs.h
https://github.com/bol-edu/caravel-soc/blob/main/firmware/caravel.h

Testbench

```
initial begin
        wait(checkbits == 16'hAB60);
        $display("Monitor: MPRJ-Logic WB Started");
        wait(checkbits == 16'hAB61);
        $display("Monitor: Mega-Project WB (RTL) Passed");
        $finish;
end
```

https://github.com/bol-edu/caravel-soc/blob/main/testbench/counter_wb/counter_wb_tb.v

User Project – user_proj_example.counter.v

```
module user_proj_example #(
    parameter BITS = 32
)(

    // Wishbone Slave ports (WB MI A)
    input wb_clk_i,
    input wb_rst_i,
    input wbs_stb_i,
    input wbs_cyc_i,
    input wbs_we_i,
    input [3:0] wbs_sel_i,
    input [31:0] wbs_dat_i,
    input [31:0] wbs_adr_i,
    output wbs_ack_o,
    output [31:0] wbs_dat_o,

    // Logic Analyzer Signals
    input  [127:0] la_data_in,
    output [127:0] la_data_out,
    input  [127:0] la_oenb,

    // IOs
    input  [`MPRJ_IO_PADS-1:0] io_in,
    output [`MPRJ_IO_PADS-1:0] io_out,
    output [`MPRJ_IO_PADS-1:0] io_oeb,

    // IRQ
    output [2:0] irq
);
```

```
// WB MI A
  assign valid = wbs_cyc_i && wbs_stb_i;
  assign wstrb = wbs_sel_i & {4{wbs_we_i}};
  assign wbs_dat_o = rdata;
  assign wdata = wbs_dat_i;

  // IO
   assign io_out = count;
  assign io_oeb = {(`MPRJ_IO_PADS-1){rst}};

  // IRQ
  assign irq = 3'b000;          // Unused

  // LA
  assign la_data_out = {{(127-BITS){1'b0}}, count};
  // Assuming LA probes [63:32] are for controlling the count register
  assign la_write = ~la_oenb[63:32] & ~{BITS{valid}};
  // Assuming LA probes [65:64] are for controlling the count clk & reset
  assign clk = (~la_oenb[64]) ? la_data_in[64]: wb_clk_i;
  assign rst = (~la_oenb[65]) ? la_data_in[65]: wb_rst_i;
```

module: counter

```
always @(posedge clk) begin
      if (reset) begin
         count <= 0;
         ready <= 0;
      end else begin
         ready <= 1'b0;
         if (~|la_write) begin
            count <= count + 1;
         end
         if (valid && !ready) begin
            ready <= 1'b1;
            rdata <= count;
            if (wstrb[0]) count[7:0]   <= wdata[7:0];
            if (wstrb[1]) count[15:8]  <= wdata[15:8];
            if (wstrb[2]) count[23:16] <= wdata[23:16];
            if (wstrb[3]) count[31:24] <= wdata[31:24];
         end else if (|la_write) begin
            count <= la_write & la_input;
         end
      end
   end
```

https://github.com/bol-edu/caravel-soc/blob/main/rtl/user/user_proj_example.counter.v

firmware

```
// Flag start of the test
   reg_mprj_datal = 0xAB600000;

   reg_mprj_slave = 0x00002710;
   reg_mprj_datal = 0xAB610000;
   if (reg_mprj_slave == 0x2B3D){
      reg_mprj_datal = 0xAB610000;
   }
```

user_project

```
// WB MI A
   assign valid = wbs_cyc_i && wbs_stb_i;
   assign wstrb = wbs_sel_i & {4{wbs_we_i}};
   assign wbs_dat_o = rdata;
   assign wdata = wbs_dat_i;

   // IO
    assign io_out = count;
   assign io_oeb = {(`MPRJ_IO_PADS-1){rst}};

   // IRQ
   assign irq = 3'b000;          // Unused

   // LA
   assign la_data_out = {{(127-BITS){1'b0}}, count};
   // Assuming LA probes [63:32] are for controlling the count register
   assign la_write = ~la_oenb[63:32] & ~{BITS{valid}};
   // Assuming LA probes [65:64] are for controlling the count clk & reset
   assign clk = (~la_oenb[64]) ? la_data_in[64]: wb_clk_i;
   assign rst = (~la_oenb[65]) ? la_data_in[65]: wb_rst_i;
```

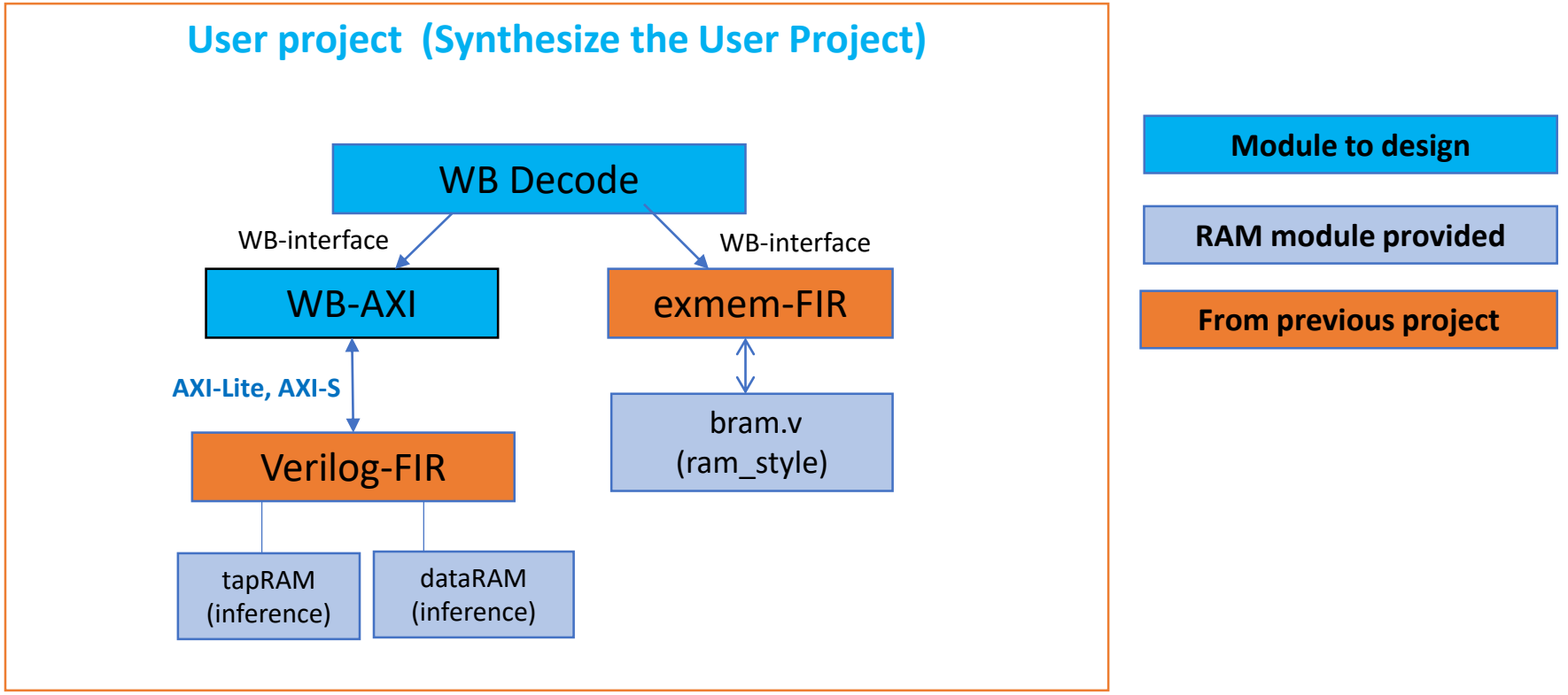
```
always @(posedge clk) begin
      if (reset) begin
         count <= 0;
         ready <= 0;
      end else begin
         ready <= 1'b0;
         if (~|la_write) begin
            count <= count + 1;
         end
         if (valid && !ready) begin
            ready <= 1'b1;
            rdata <= count;
            if (wstrb[0]) count[7:0]   <= wdata[7:0];
            if (wstrb[1]) count[15:8]  <= wdata[15:8];
            if (wstrb[2]) count[23:16] <= wdata[23:16];
            if (wstrb[3]) count[31:24] <= wdata[31:24];
         end else if (|la_write) begin
            count <= la_write & la_input;
         end
      end
   end
```

Testbench

```
initial begin
      wait(checkbits == 16'hAB60);
      $display("Monitor: MPRJ-Logic WB Started");
      wait(checkbits == 16'hAB61);
      $display("Monitor: Mega-Project WB (RTL) Passed");
      $finish;
end
```

Lab 4-2 - Caravel FIR

Design Scope & Hierarchy



fir.c in Lab4-1, Lab4-2

- Lab4-1
  - fir.c calucates y[t] = Σ (h[i] * x[t - i])
  - The compiled fir.hex is loaded into bram in user-project area

- Lab4-2
  - y[t] = Σ (h[i] * x[t - i] is calucated in hardware accelerator (fir.v)
  - fir_control.c (let's rename it to avoid confuse) is the firmware
    - Control the hardware accelerator (fir.v)
    - Communicate with testbench

RTL Design

- exmem-FIR : Use design from Lab4-1
- verilog-FIR : Use design from Lab3
- WB-AXI: Wishbone to AXI-lite, AXI-S interface conversion
  - Refer to configuration address map
  - If wb address is in the range 3000_0000 – 3000_007F, read/write convert to Axilite
  - If wb address is 3000_0080 (send X[n])
    - write transaction converts to axi stream master to send data to verilog-FIR
    - If read, return previous written X. Note every transaction has to have completion, otherwise, system hang
  - If wb address is 3000_0084 (read Y[n])
    - Read transaction, converts to axi-stream slave to read Y[n] from verilog-FIR
    - Write transaction, ignore, and return ack
- WB-Decode: decode wishbone transaction address and dispatch to exmem-FIR, and WB-AXI
  - Interface with user_project_example
  - Address 3000_xxxx transaction sent to WB_AXI
  - Address 3800_xxxx, transaction sent to exmem-FIR

Firmware: fir_control.c

0. Firmware code loaded into exmem and execute from it (refer Lab4-1)
1. Initialization code:
    1. Configure mprj pin, e.g. direction (in/out), select output from risc-v or user-project …
    2. Define mmio registers , e.g.
        1. **#define reg_fir_control (*(volatile uint32_t*)0x38000000)**
        2. **#define reg_fir_coeff (*(volatile uint32_t*)0x38000040)**
        3. **#define reg_fir_x(*(volatile uint32_t*)0x38000080)**
        4. #define reg_fir_y(*(volatile uint32_t*)0x38000084)
2. Program coeff, len by simple, e.g. reg_fir_coeff_0 = xx ….
3. RISC-V outputs a StartMark ‘hA5 on mprj[23:16] to notify Testbench to start latency-timer (in testbench)
    1. reg_mprj_datal = 0x00A50000;
4. RISC-V sends X[n] to FIR (note: make sure FIR is readily to accept X[n)) : **reg_fir_x** = value
5. RISC-V receives Y[n] from FIR (note: make sure Y[n] is ready y_value = reg_fir_y;
6. Repeat 3, 4, until len of Y[n] is received
7. When finish, write final Y (Y[7:0] output to mprj[31:24]), EndMark (‘h5A – mprj[23:16]), record the latency-timer
8. Testbench check correctness by checking mprj[31:24], and print out the latency-timer.
9. Repeat 2 – 7 for three times, and record and add up the latency timer

Test Data

- Due to Caravel SOC has only limited data memory, and there is no file system for the data file.
- The tap parameters is defined in the program code (Global data)

  int[10:0] tap = {tap0, tap1, tap2, … };
- The data set is generated by RISC-V program. Using the following loop to generate X[n] - data_length = 64

```
// Design your own sequence – area for optimization
for(n = 0; n < data_length; n++)  {
    x[n] = n;
    // send x[n] to FIR
    // receive y[n] from FIR
}
```

Configuration Register Address map (Suggested)

User Project Memory Starting: 3800_0000

User Project FIR Base Address : 3000_0000

0x00 – [0] - ap_start (r/w)

set, when ap_start signal assert

reset, when start data transfer, i.e. 1st axi-stream data come in

[1] – ap_done (ro) -> when FIR process all the dataset, i.e. receive tlast and last Y generated/transferred

[2] – ap_idle (ro) -> indicate FIR is actively processing data

[3] – Reserved (ro) -> read zero

[4] – X[n]_ready to accept input (ro) -> X[n] is ready to accept input.

[5] - Y[n] is ready to read -> set when Y[n] is ready, reset when 0x00 is read

0x10-14 - data-length

0x40-7F – Tap parameters, (e.g., 0x20-24 Tap0, in sequence …

0x80-83 – X[n] input (r/w)

0x84-87 – Y[n] output (ro)