Lab6

Workload optimized SOC – baseline

Content

- Specification
  - Firmware
    - Matrix Multiplication
    - Quick Sort
    - FIR
    - ISR – UART/rx & tx
  - Hardware
    - Lab4-1 – exmem
    - UART
- Verification
  - Simulation
  - FPGA Implementation
    - Block Design
    - Notebook

Firmware – Matrix Multiplication

- 4*4 matrix matrix multiplication

```
cd ~/caravel-soc_fpga-lab/lab-wlos_baseline/testbench/counter_la_mm
source run_clean
source run_sim
```

Firmware – Quick Sort

- 10-elements quick sort

```
cd ~/caravel-soc_fpga-lab/lab-wlos_baseline/testbench/counter_la_qs
source run_clean
source run_sim
```

Firmware – FIR

- Same as Lab4-1

```
cd ~/caravel-soc_fpga-lab/lab-wlos_baseline/testbench/counter_la_fir
source run_clean
source run_sim
```

Firmware – UART

Under Firmware

isr.c

Define Interrupt service routine for UART Receiver

user_uart.h

Define Configuration Register for UART

Under Testbench

uart.c

UART R/W Function

counter_la_uart.c

Main Firmware

Firmware – UART

Configuration Register for UART

```
1   #ifndef _USER_UART_H_
2   #define _USER_UART_H_
3
4   #include <stdint.h>
5   #include <stdbool.h>
6
7   #define reg_rx_data          (*(volatile uint32_t*)0x30000000)
8   #define reg_tx_data          (*(volatile uint32_t*)0x30000004)
9   #define reg_uart_stat        (*(volatile uint32_t*)0x30000008)
10  #define reg_baud_rate        (*(volatile uint32_t*)0x3000000c)
```

USER project UART base address

- reg_rx_data: 32'h3000_0000
- reg_tx_data: 32'h3000_0004
- reg_uart_stat: 32'h3000_0008
- reg_baud_rate: 32'h3000_000c

Note that we don't implement reg_baud_rate design in hardware design, you can modify the design if you want to use higher baud rate for better performance.

Firmware – UART

Configuration Register for UART

```
//+------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
//|RX_DATA |  RESERVERD  |                       DATA BITS                       |
//|        |    31-8     | 7     | 6     | 5     | 4     | 3     | 2     | 1     | 0     |
//+------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
//|TX_DATA |  RESERVERD  |                       DATA BITS                       |
//|        |    31-8     | 7     | 6     | 5     | 4     | 3     | 2     | 1     | 0     |
//+------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
//|STAT_REG|  RESERVERD  |  Frame Err  |  Overrun Err  |  Tx_full  |  Tx_empty  |  Rx_full  |  Rx_empty |
//|        |    31-6     |  5          |  4            |  3        |  2         |  1        |  0        |
//+------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
```

Register bits explanation in our design

Table 2-8: **Status Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31 - 8	Reserved	N/A	0h	Reserved
7	Parity Error	Read	0h	Indicates that a parity error has occurred after the last time the status register was read. If the UART is configured without any parity handling, this bit is always 0. The received character is written into the receive FIFO. This bit is cleared when the status register is read. 0 = No parity error has occurred 1 = Parity error has occurred
6	Frame Error	Read	0h	Indicates that a frame error has occurred after the last time the status register was read. Frame error is defined as detection of a stop bit with the value 0. The receive character is ignored and not written to the receive FIFO. This bit is cleared when the status register is read. 0 = No frame error has occurred 1 = Frame error has occurred
5	Overrun Error	Read	0h	Indicates that a overrun error has occurred after the last time the status register was read. Overrun is when a new character has been received but the receive FIFO is full. The received character is ignored and not written into the receive FIFO. This bit is cleared when the status register is read. 0 = No overrun error has occurred 1 = Overrun error has occurred
4	Intr Enabled	Read	0h	Indicates that interrupts is enabled. 0 = Interrupt is disabled 1 = Interrupt is enabled
3	Tx FIFO Full	Read	0h	Indicates if the transmit FIFO is full. 0 = Transmit FIFO is not full 1 = Transmit FIFO is full
2	Tx FIFO Empty	Read	01h	Indicates if the transmit FIFO is empty. 0 = Transmit FIFO is not empty 1 = Transmit FIFO is empty
1	Rx FIFO Full	Read	0h	Indicates if the receive FIFO is full. 0 = Receive FIFO is not full 1 = Receive FIFO is full
0	Rx FIFO Valid Data	Read	0h	Indicates if the receive FIFO has data. 0 = Receive FIFO is empty 1 = Receive FIFO has data

Register bits explanation in Xilinx UART IP

Firmware – UART

uart.c

```c
void __attribute__ ( ( section ( ".mprj" ) ) ) uart_write(int n)
{
    while(((reg_uart_stat>>3) & 1));
    reg_tx_data = n;
}

void __attribute__ ( ( section ( ".mprj" ) ) ) uart_write_char(char c)
{
    if (c == '\n')
        uart_write_char('\r');

    // wait until tx_full = 0
    while(((reg_uart_stat>>3) & 1));
    reg_tx_data = c;
}

void __attribute__ ( ( section ( ".mprj" ) ) ) uart_write_string(const char *s)
{
    while (*s)
        uart_write_char(*(s++));
}


char __attribute__ ( ( section ( ".mprj" ) ) ) uart_read_char()
{
    char word;
    if((((reg_uart_stat>>5) | 0) == 0) && (((reg_uart_stat>>4) | 0) == 0)){
        for(int i = 0; i < 1; i++)
            asm volatile ("nop");

        word = reg_rx_data;
    }

    return word;
}

int __attribute__ ( ( section ( ".mprj" ) ) ) uart_read()
{
    int num;
    if((((reg_uart_stat>>5) | 0) == 0) && (((reg_uart_stat>>4) | 0) == 0)){
        for(int i = 0; i < 1; i++)
            asm volatile ("nop");

        num = reg_rx_data;
    }

    return num;
}
```

In this firmware code, we define some basic function to realize the UART function. You can also modify it or add extra function depends on your need.

Firmware – UART

counter_la_uart.c

```
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_MGMT_STD_OUTPUT;
reg_mprj_io_14 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_13 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_12 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_11 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_10 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_9  = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_8  = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_7  = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_4  = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_3  = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_2  = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_1  = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_0  = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_6  = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_5  = GPIO_MODE_USER_STD_INPUT_NOPULL;
```

Note that the screenshot beside is the mprj_io configuration.
Do not modify it.
For mprj_io_6 & mprj_io_5, they are corresponding to the uart tx and rx pins.
If you are interested in the detail configuration about these io you can reference the attached datasheet file “gpio.rst”

```
``GPIO_MODE_USER_STD_OUTPUT``:
   The user project controls the GPIO pin.
   Pin is configured as a digital output (output only).
```

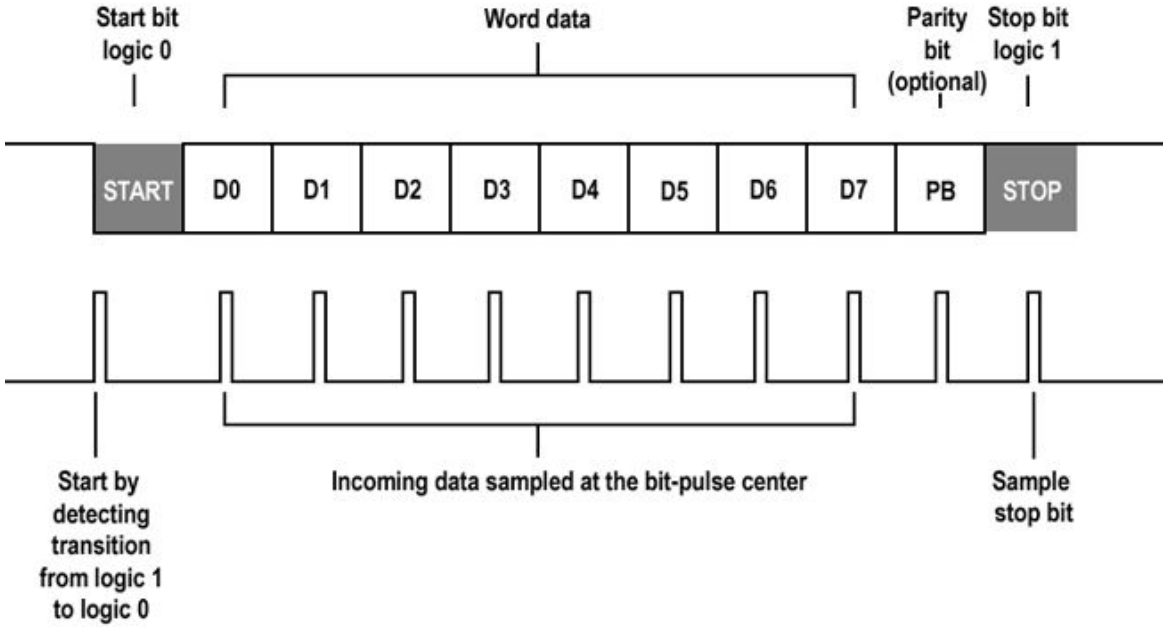
```
``GPIO_MODE_MGMT_STD_INPUT_NOPULL``:
   Management controls the GPIO pin.
   Pin is configured as a digital input, no pull-up or pull-down.
```

Firmware – ISR (UART/rx & tx)

- As receiving rx, user project needs generate interrupt to make CPU jump to Interrupt Service Routine (ISR)
  - Setting CSR
    - https://github.com/bol-edu/caravel-soc_fpga-lab/blob/main/lab-wlos_baseline/firmware/crt0_vex.S#L102-L104
  - After interrupt triggers, CPU jump to ***trap_entry***
    - https://github.com/bol-edu/caravel-soc_fpga-lab/blob/main/lab-wlos_baseline/firmware/crt0_vex.S#L15-L53
  - And then call function isr()
    - https://github.com/bol-edu/caravel-soc_fpga-lab/blob/main/lab-wlos_baseline/firmware/isr.c
- On ISR, it receives rx and send tx
  - https://github.com/bol-edu/caravel-soc_fpga-lab/blob/main/lab-wlos_baseline/firmware/isr.c#L37-L38

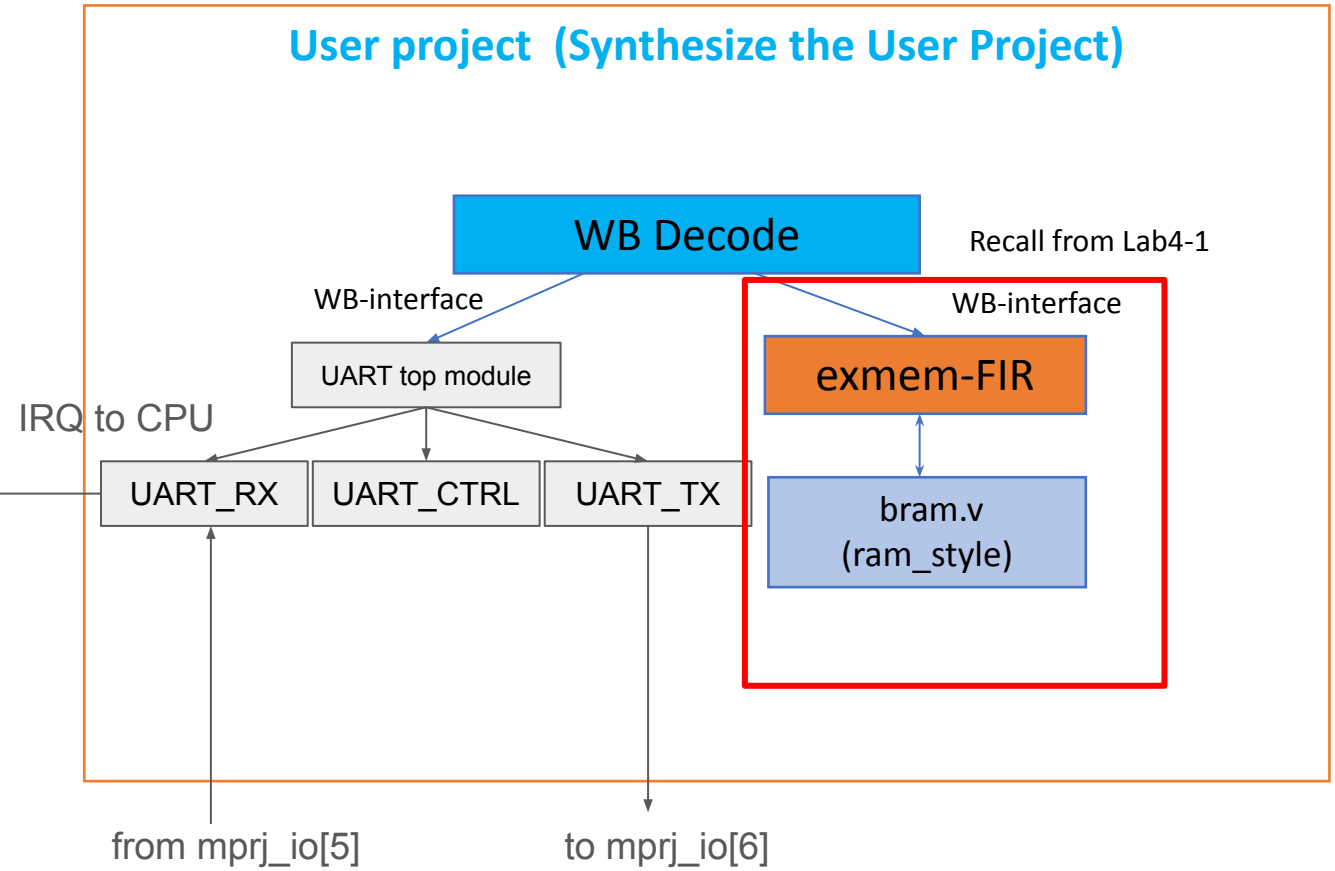
Firmware – ISR (UART/rx & tx)

- No Parity bit in our design
- Baud rate = 9600



Hardware Scope & Hierarchy

User Project Memory Starting: 3800_0000
User Project UART Base Address : 3000_0000



Module to design

RAM module provided

From previous project

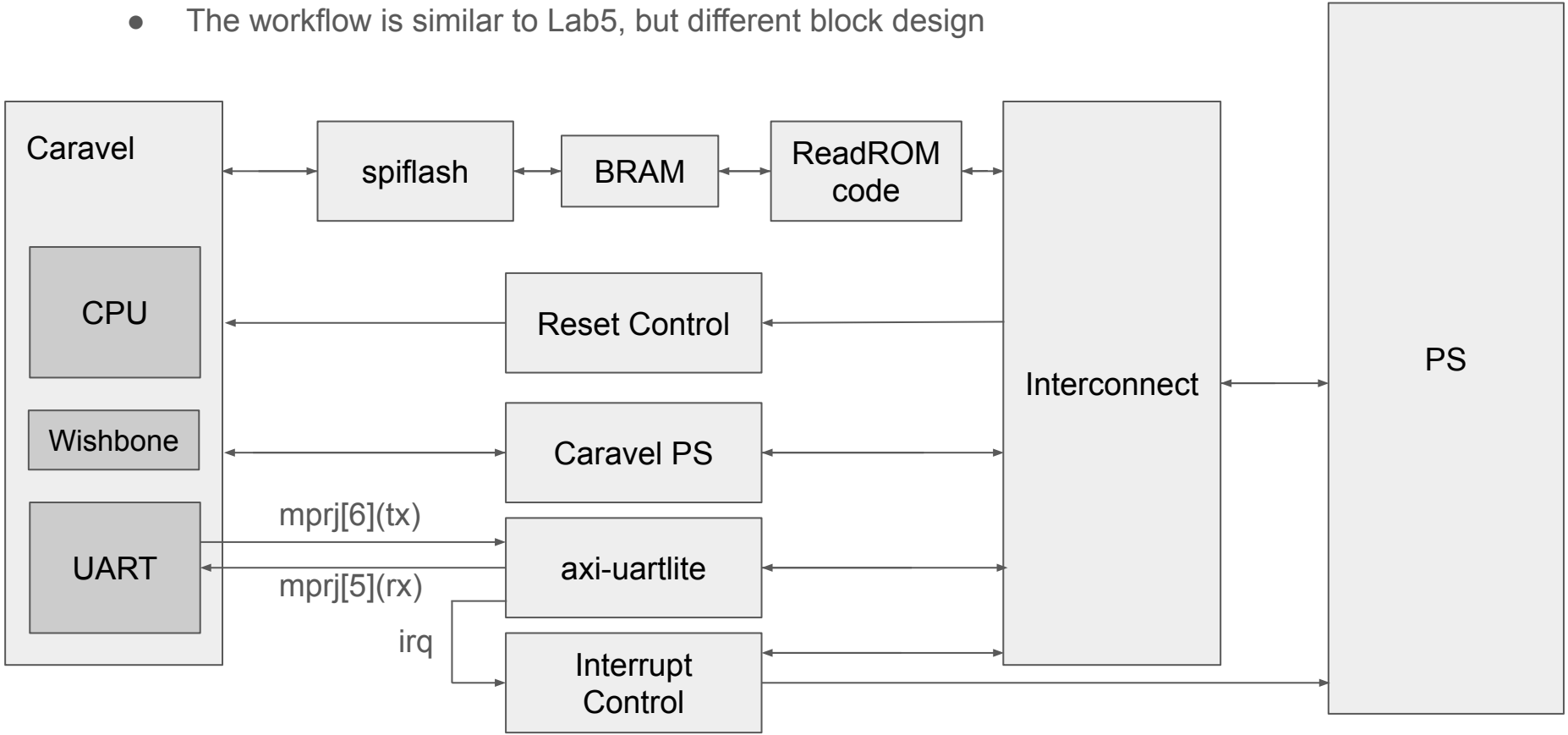
- Refer to https://github.com/bol-edu/caravel-soc_fpga-lab/tree/main/lab-wlos_baseline/rtl/user

Simulation

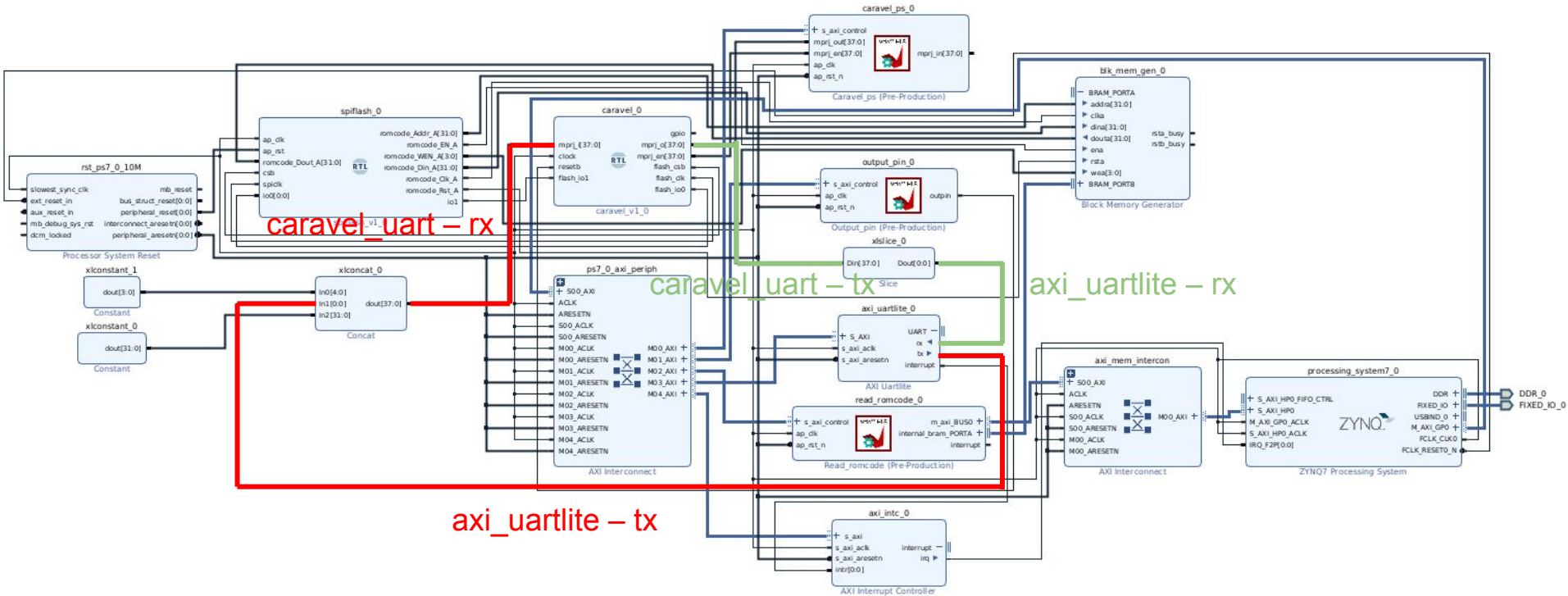
```
cd ~/caravel-soc_fpga-lab/lab-wlos_baseline/testbench/uart
source run_clean
source run_sim
```

FPGA Implementation – Block Design

- The workflow is similar to Lab5, but different block design



FPGA Implementation – Block Design



FPGA Implementation – Notebook

- Refer to https://github.com/bol-edu/caravel-soc_fpga-lab/blob/main/lab-wlos_baseline/vivado/jupyter_notebook/caravel_fpga_uart.ipynb

```
async def uart_rxtx():
    # Reset FIFOs, enable interrupts
    ipUart.write(CTRL_REG, 1<<RST_TX | 1<<RST_RX | 1<<INTR_EN)
    print("Waitting for interrupt")
    tx_str = "hello\n"
    ipUart.write(TX_FIFO, ord(tx_str[0]))
    i = 1
    while(True):
        await intUart.wait()
        buf = ""
        # Read FIFO until valid bit is clear
        while ((ipUart.read(STAT_REG) & (1<<RX_VALID))):
            buf += chr(ipUart.read(RX_FIFO))
            if i<len(tx_str):
                ipUart.write(TX_FIFO, ord(tx_str[i]))
                i=i+1
        print(buf, end='')
```

```
Start Caravel Soc
Waitting for interrupt
hello
main(): uart_rx is cancelled now
```

What you need to do

- Replicate baseline experiment
  - Simulation on Matrix Multiplication, Quick Sort, FIR and UART separately
  - UART FPGA
- Firmware code integrates Matrix Multiplication, Quick Sort, FIR and UART
- Hardware integrates exmem-fir with UART design in user project area, like Lab4-2.
- Simulation
  - Modify testbench to include the test for Matrix Multiplication, Quick Sort, FIR and UART
- Run on FPGA
  - Verify if the firmware code can execute on FPGA

Notice

- 40Mhz on axi-uartlite and caravel
- It will take 15 minutes on synthesis and implementation
- 5 slave channel on axi-interconnect
- Only support for non-continuous tx/rx on our UART design.
- If you finished hardware, copy the files under ***rtl/user*** folder to ***vivado/vvd_src/caravel_soc/rtl/user***
- For verification on FPGA, you need the following files uploading to PYNQ
    - caravel_fpga.bit
    - caravel_fpga.hwh
    - caravel_fpga_uart.ipynb
    - uartlite.py
    - uart.hex

Submission Guideline

- Report
  - How do you verify your answer from notebook
  - Block design
  - Timing report/ resource report after synthesis
  - Latency for a character loop back using UART
  - Suggestion for improving latency for UART loop back
  - What else do you observe
- Required files for FPGA
- Firmware code that you integrate all task
- Github link