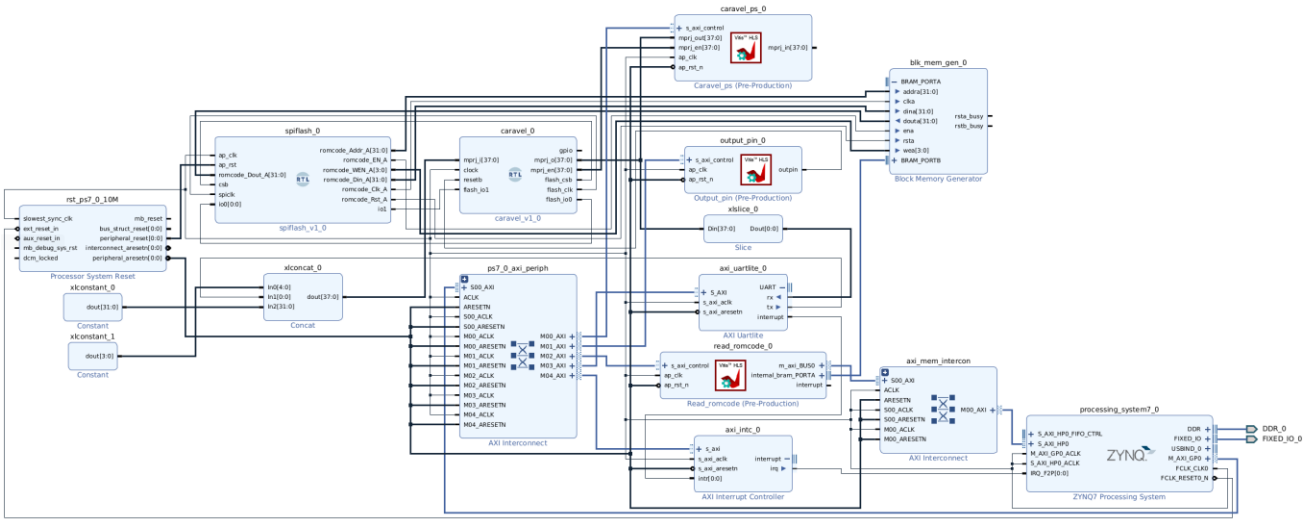
SOC DESIGN LAB 6

TEAM 8

Block Design



Utilization

Slice logic

```
1. Slice Logic
--------------

+----------------------------+------+-------+------------+-----------+-------+
|          Site Type         | Used | Fixed | Prohibited | Available | Util% |
+----------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*                | 6118 |     0 |          0 |     53200 | 11.50 |
|   LUT as Logic             | 5877 |     0 |          0 |     53200 | 11.05 |
|   LUT as Memory            |  241 |     0 |          0 |     17400 |  1.39 |
|     LUT as Distributed RAM |   18 |     0 |            |           |       |
|     LUT as Shift Register  |  223 |     0 |            |           |       |
| Slice Registers            | 6901 |     0 |          0 |    106400 |  6.49 |
|   Register as Flip Flop    | 6826 |     0 |          0 |    106400 |  6.42 |
|   Register as Latch        |   75 |     0 |          0 |    106400 |  0.07 |
| F7 Muxes                   |  171 |     0 |          0 |     26600 |  0.64 |
| F8 Muxes                   |   47 |     0 |          0 |     13300 |  0.35 |
+----------------------------+------+-------+------------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and full implementation, is typically
lower. Run opt_design after synthesis, if not already completed, for a more realistic count.
```

Memory

```
2. Memory
---------

+-------------------+------+-------+------------+-----------+-------+
|     Site Type     | Used | Fixed | Prohibited | Available | Util% |
+-------------------+------+-------+------------+-----------+-------+
| Block RAM Tile    |    8 |     0 |          0 |       140 |  5.71 |
|   RAMB36/FIFO*    |    5 |     0 |          0 |       140 |  3.57 |
|     RAMB36E1 only |    5 |       |            |           |       |
|   RAMB18          |    6 |     0 |          0 |       280 |  2.14 |
|     RAMB18E1 only |    6 |       |            |           |       |
+-------------------+------+-------+------------+-----------+-------+
```

Approximate utilize resource

Ref Name	Used	Functional Category
FDRE	5332	Flop & Latch
LUT6	2521	LUT
LUT4	1256	LUT
LUT5	1153	LUT
LUT3	1085	LUT
FDCE	1077	Flop & Latch
LUT2	588	LUT
FDPE	285	Flop & Latch
CARRY4	270	CarryLogic
LUT1	269	LUT
MUXF7	171	MuxFx
SRL16E	152	Distributed Memory
FDSE	132	Flop & Latch
BIBUF	130	IO
LDCE	75	Flop & Latch
SRLC32E	71	Distributed Memory
MUXF8	47	MuxFx
RAMD32	26	Distributed Memory
RAMS32	8	Distributed Memory
RAMB18E1	6	Block Memory
RAMB36E1	5	Block Memory
PS7	1	Specialized Resource
BUFG	1	Clock

Timing report

```
| Design Timing Summary
| ---------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints     WPWS(ns)     TPWS(ns)  TPWS Failing Endpoints  TPWS Total Endpoints
    -------      -------  ---------------------  -------------------      -------      -------  ---------------------  -------------------     --------     --------  ----------------------  --------------------
     11.848        0.000                      0                14037       -0.762       -1.523                      2                14037       11.250        0.000                       0                  5808


Timing constraints are not met.


| Clock Summary
| -------------

Clock      Waveform(ns)       Period(ns)      Frequency(MHz)
-----      ------------       ----------      --------------
clk_fpga_0  {0.000 12.500}     25.000          40.000
```

Simulation

```
FIR Test started
tx data bit index 0: 1 at              3281229
tx data bit index 1: 1 at              3385395
tx data bit index 2: 1 at              3489561
tx data bit index 3: 1 at              3593727
tx data bit index 4: 0 at              3697893
tx data bit index 5: 0 at              3802059
tx data bit index 6: 0 at              3906225
tx data bit index 7: 0 at              4010391
tx complete 1
UART receive passed
rx data bit index 0: 1 at                    4843719
rx data bit index 1: 1 at                    4947885
rx data bit index 2: 1 at                    5052051
rx data bit index 3: 1 at                    5156217
rx data bit index 4: 0 at                    5260383
rx data bit index 5: 0 at                    5364549
rx data bit index 6: 0 at                    5468715
rx data bit index 7: 0 at                    5572881
transmission passed; recevied word  15 at              5677047
FIR Test passed

MM Test started
Call function matmul() in User Project BRAM, return value passed, 0x003e
Call function matmul() in User Project BRAM, return value passed, 0x0044
Call function matmul() in User Project BRAM, return value passed, 0x004a

UART Test 2 started
Call function matmul() in User Project BRAM, return value passed, 0x0050
MM Test passed

QS Test started
tx data bit index 0: 1 at              8281197
tx data bit index 1: 0 at              8385363
tx data bit index 2: 1 at              8489529
tx data bit index 3: 1 at              8593695
tx data bit index 4: 1 at              8697861
Call function qsort() in User Project BRAM, return value passed, 0x0028
tx data bit index 5: 1 at              8802027
Call function qsort() in User Project BRAM, return value passed, 0x037d
Call function qsort() in User Project BRAM, return value passed, 0x09ed
Call function qsort() in User Project BRAM, return value passed, 0x0a6d
tx data bit index 6: 0 at              8906193
tx data bit index 7: 0 at              9010359
QS Test passed
```

```
QS Test started
tx complete 2
UART receive passed
rx data bit index 0: 1 at                    9843687
rx data bit index 1: 0 at                    9947853
rx data bit index 2: 1 at                   10052019
rx data bit index 3: 1 at                   10156185
rx data bit index 4: 1 at                   10260351
rx data bit index 5: 1 at                   10364517
rx data bit index 6: 0 at                   10468683
rx data bit index 7: 0 at                   10572849
transmission passed; recevied word  61 at            10677015
Call function qsort() in User Project BRAM, return value passed, 0x0028
Call function qsort() in User Project BRAM, return value passed, 0x037d
Call function qsort() in User Project BRAM, return value passed, 0x09ed
Call function qsort() in User Project BRAM, return value passed, 0x0a6d
QS Test passed

Test Passed
```

Read_romcode

This block is to copy the data in PS side into BRAM or write the data from BRAM to PS side

In each transaction, we have different bus to send the different base address.

For example, PS side set the port _BUS_0 just to send data to PL side, another transaction is vice versa.

Spiflash

spi slave only support read command and its mmio address is 0x03 owing to return the data from BRAM to Caravel

SPI protocal consist of MOSI MISO SCK SS and the transaction is serial transmiting from lsb to msb and the slave side also.

UART

UART Control: This block consists of Rx Control - This block samples received data with respect to generated baud rate and writes it to Receive Data FIFO.

Tx Control: This block reads data from Transmit Data FIFO and sends it out on the UART Tx interface.

Interrupt Control: The AXI UART Lite core provides interrupt enable/disable control. If interrupts are enabled, a rising-edge sensitive interrupt is generated when the receive FIFO becomes non-empty or when the transmit FIFO becomes empty

How uart in this project work

This time we use isr routine inside the firmware code to generate interrupt signal to
CPU just for receive data for uart.
we use uart.h to define the register to configure uart reg

caravel_uart_tx is connected to axi_lite_uart_rx and caravel_uart_rx is coneected to axi_lire_uart_tx

PS site write data to uart_lite and caravel will use the data by using uart port
if caravel_soc wanna send data to PS side it will trigger a interupt to axi interrupt controller and controller will tell the irq inside PS side that there is a data inside axi_lite and then PS side read the data putting it into buffer

Screenshot of notebook result

Note that we add some for loops at firmware code of “main_for_jupy.c” (use run_sim_j to generate the hex file) to make sure that each checkbits can be read at the notebook demo.

The task to verify the result :

```
async def check():
    while((ipPS.read(0x1c) & 0xffff0000) != 0xab100000):
        await asyncio.sleep(0.01)
        continue
    print ("checkbits = ab10, FIR started")

    while((ipPS.read(0x1c) & 0xffff0000) != 0xab110000):
        await asyncio.sleep(0.01)
        continue
    print ("checkbits = ab11, FIR is done")

    while((ipPS.read(0x1c) & 0xffff0000) != 0xab200000):
        await asyncio.sleep(0.01)
        continue
    print ("checkbits = ab20, matmul is done")

    while((ipPS.read(0x1c) & 0xffff0000) != 0xab210000):
        await asyncio.sleep(0.01)
        continue
    print ("checkbits = ab21, matmul is done")

    while((ipPS.read(0x1c) & 0xffff0000) != 0xab300000):
        await asyncio.sleep(0.01)
        continue
    print ("checkbits = ab30, qsort1 is done")

    while((ipPS.read(0x1c) & 0xffff0000) != 0xab310000):
        await asyncio.sleep(0.01)
        continue
    print ("checkbits = ab31, qsort1 is done")

    while((ipPS.read(0x1c) & 0xffff0000) != 0xab300000):
        await asyncio.sleep(0.01)
        continue
    print ("checkbits = ab30, qsort2 started")

    while((ipPS.read(0x1c) & 0xffff0000) != 0xab310000):
        await asyncio.sleep(0.01)
        continue
    print ("checkbits = ab31, qsort2 is done")
```

入力 [*]:
```
asyncio.run(async_main())
```

```
Start Caravel Soc
checkbits = ab10, FIR started
checkbits = ab11, FIR is done
checkbits = ab20, matmul is done
checkbits = ab21, matmul is done
checkbits = ab30, qsort1 is done
checkbits = ab31, qsort1 is done
Waitting for interrupt
hello
checkbits = ab30, qsort2 started
checkbits = ab31, qsort2 is done
```

入力 [19]:
```
print ("0x10 = ", hex(ipPS.read(0x10)))
print ("0x14 = ", hex(ipPS.read(0x14)))
print ("0x1c = ", hex(ipPS.read(0x1c)))
print ("0x20 = ", hex(ipPS.read(0x20)))
print ("0x34 = ", hex(ipPS.read(0x34)))
print ("0x38 = ", hex(ipPS.read(0x38)))
```

```
0x10 =  0x0
0x14 =  0x0
0x1c =  0xab310040
0x20 =  0x0
0x34 =  0x20
0x38 =  0x3f
```

To evaluate a character loop back time, we then adjust the uart_rxtx function to report the timing

```
import time
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()
        timer = time.time()
        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='')
        print("\nloop back time = ", time.time() - timer)
```

```
Start Caravel Soc
checkbits = ab10, FIR started
checkbits = ab11, FIR is done
checkbits = ab20, matmul is done
checkbits = ab21, matmul is done
checkbits = ab30, qsort1 is done
checkbits = ab31, qsort1 is done
Waitting for interrupt

loop back time =  0.00033354759216308594
h
loop back time =  0.0004532337188720703

loop back time =  0.00028395652770996094
e
loop back time =  0.00043773651123046875

loop back time =  0.0002803802490234375
l
loop back time =  0.0004837512969970703

loop back time =  0.0002849102020263672
l
loop back time =  0.0004439353942871094
o
loop back time =  0.00042700767517089844

loop back time =  0.000278472900390625


loop back time =  0.00039076805114746094
checkbits = ab30, qsort2 started
checkbits = ab31, qsort2 is done
```

Average loop back time for a character is around 439us

Review

In this lab, we have learned how to use caravel soc in real FPGA to deploy a platform and its detail. In lab, writing a robust firmware code is essential because the firmware code can drastically affect the performance. It is also a good practice to have experience like how to use wishbone but in IP mode as protocol to transfer data to user project and use mprj pin to deliver result which is from fir back to RISC CPU. UART is a big challenge in this lab because we have to make sure we put the correct code into right place in the waveform we can observe that uart is accept data at first and send it to Tx port later on. In order to decrease the latency for uart. We may try to add an additional buffer to it and then use interrupt as a way to not to disturb CPU.