Efficient interrupt-driven UART FIFO Workload and Workload Optimized SoC

2023 Fall SOC Design – Final Project Presentation

311551095 資工科碩二 林聖博

312551174 資工科碩一 張祐誠

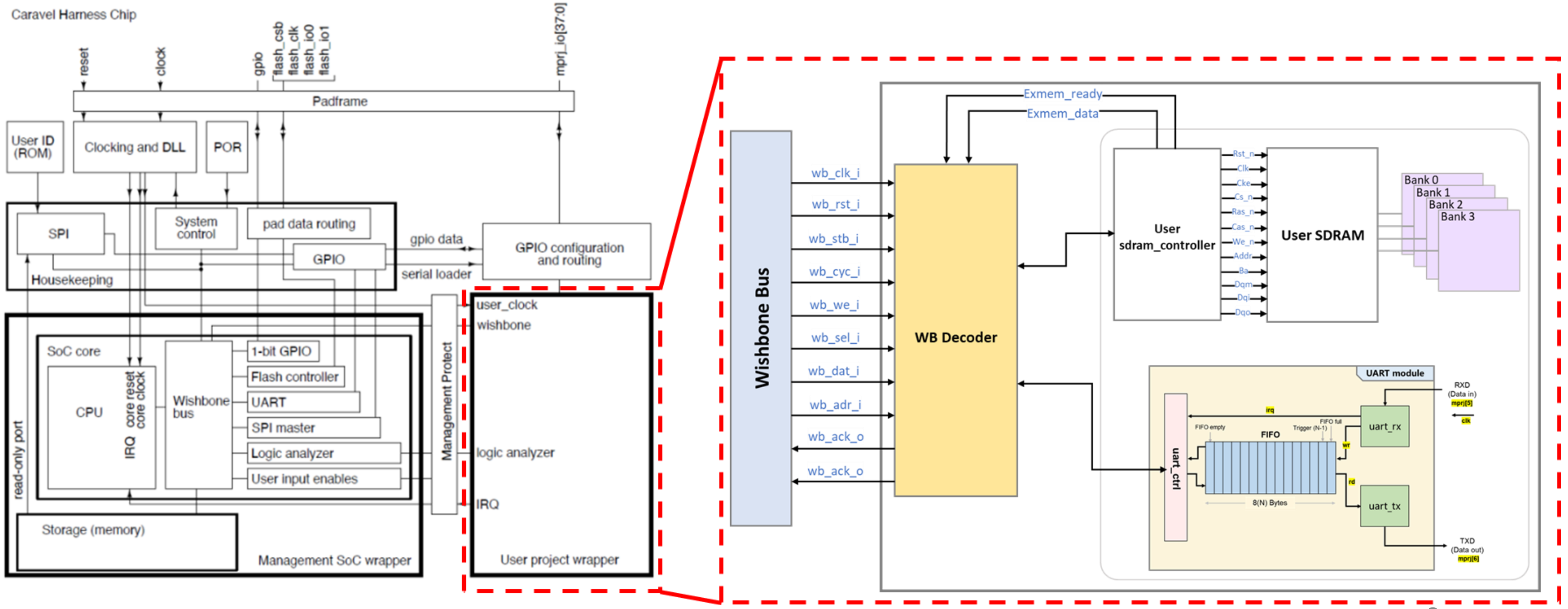
Group 11 (Team 13)

2024/01/16

Outline

Project Overview

1. SDRAM Controller with Prefetch
   - ➢ Matmul, Fir, Qsort use Firmware code
2. UART with I/O FIFO



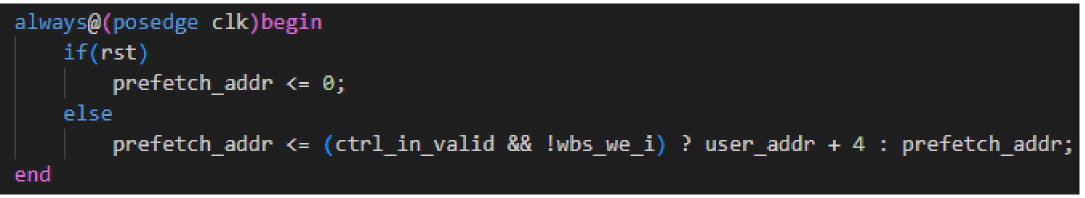
Configuration Register Address Map

Module	Base Address	Offset	Function
UART	0x3000_0000	0x00	UART **RX port**
		0x04	UART **TX port**
		0x08	**Status** of UART IP
		0x0C	UART **clkdiv**
		0x10	UART **RX FIFO Cnt**
SDRAM	0x3800_0000	0x000 (length: 200)	Code (sdrcode)
		0x200 (length: 200)	Data (sdrdata)

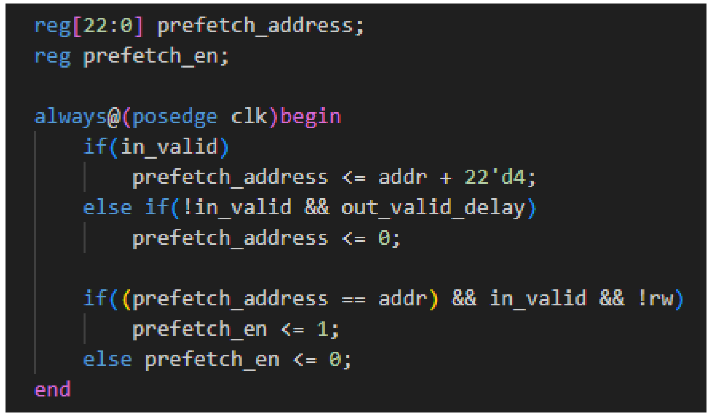
SDRAM Controller with Prefetch

SDRAM Controller with Prefetch

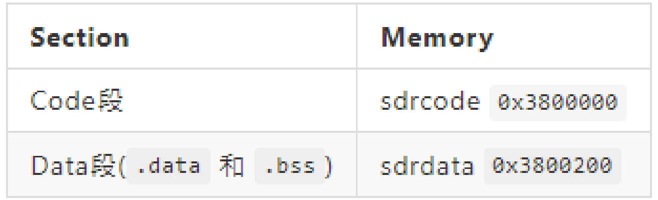
- Each clock cycle triggers an update to the **prefetch_address** based on the current state and address
- The decision to prefetch is made by comparing the **prefetch_address** to the current address alongside the read/write status
- It is noted that if the address 'x' is accessed in the current cycle, the CPU is expected to access address 'x+4' in the subsequent cycle



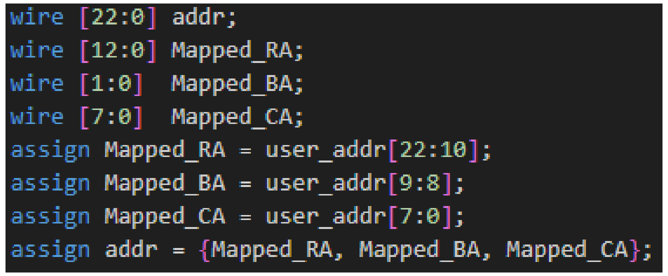
```
always@(posedge clk)begin
    if(rst)
        prefetch_addr <= 0;
    else
        prefetch_addr <= (ctrl_in_valid && !wbs_we_i) ? user_addr + 4 : prefetch_addr;
end
```



```
reg[22:0] prefetch_address;
reg prefetch_en;

always@(posedge clk)begin
    if(in_valid)
        prefetch_address <= addr + 22'd4;
    else if(!in_valid && out_valid_delay)
        prefetch_address <= 0;

    if((prefetch_address == addr) && in_valid && !rw)
        prefetch_en <= 1;
    else prefetch_en <= 0;
end
```



Section	Memory
Code段	sdrcode `0x3800000`
Data段(`.data` 和 `.bss`)	sdrdata `0x3800200`

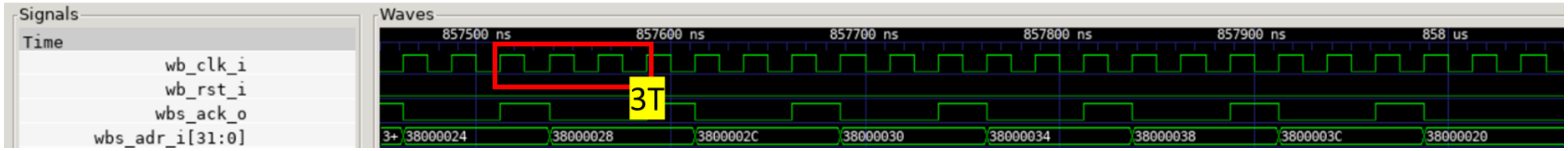


```
wire [22:0] addr;
wire [12:0] Mapped_RA;
wire [1:0]  Mapped_BA;
wire [7:0]  Mapped_CA;
assign Mapped_RA = user_addr[22:10];
assign Mapped_BA = user_addr[9:8];
assign Mapped_CA = user_addr[7:0];
assign addr = {Mapped_RA, Mapped_BA, Mapped_CA};
```

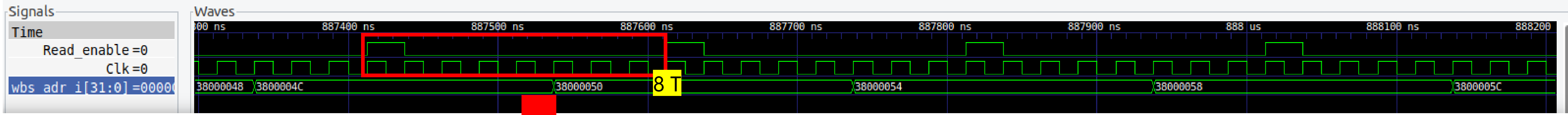
SDRAM Controller with Prefetch

- In Qsort
  - Discovered that Qsort cannot perform Prefetch in 3T
    - ✓ Our solution: Add a Delay in the READ status to avoid immediate Prefetching

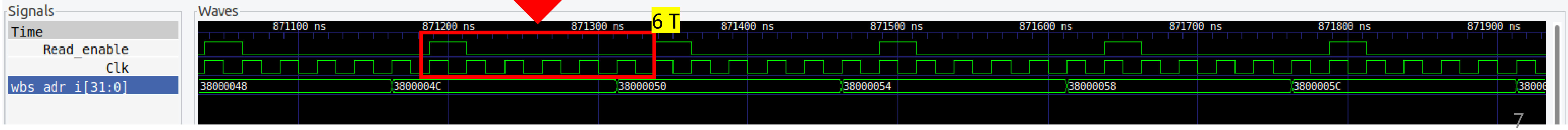
➢ **Original** SDRAM with prefetch (LabD)



➢ SDRAM without prefetch

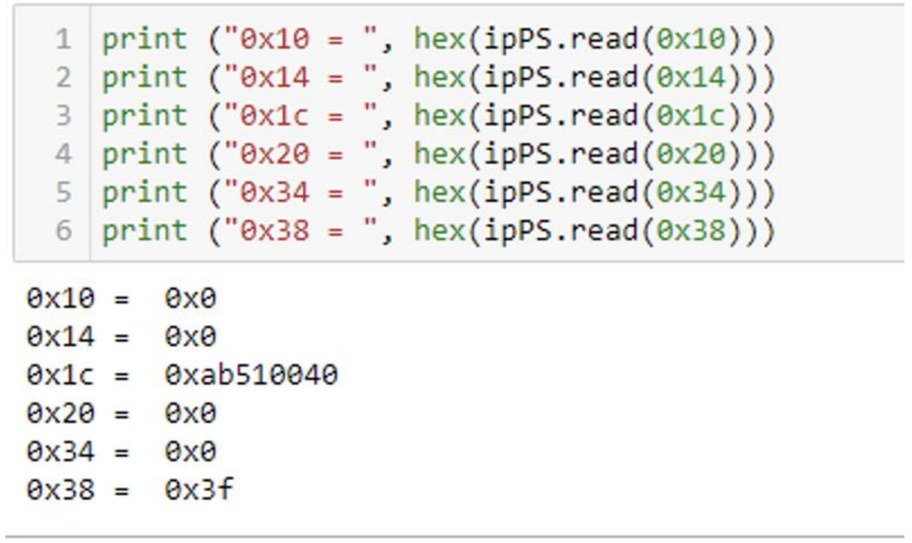


➢ SDRAM with prefetch



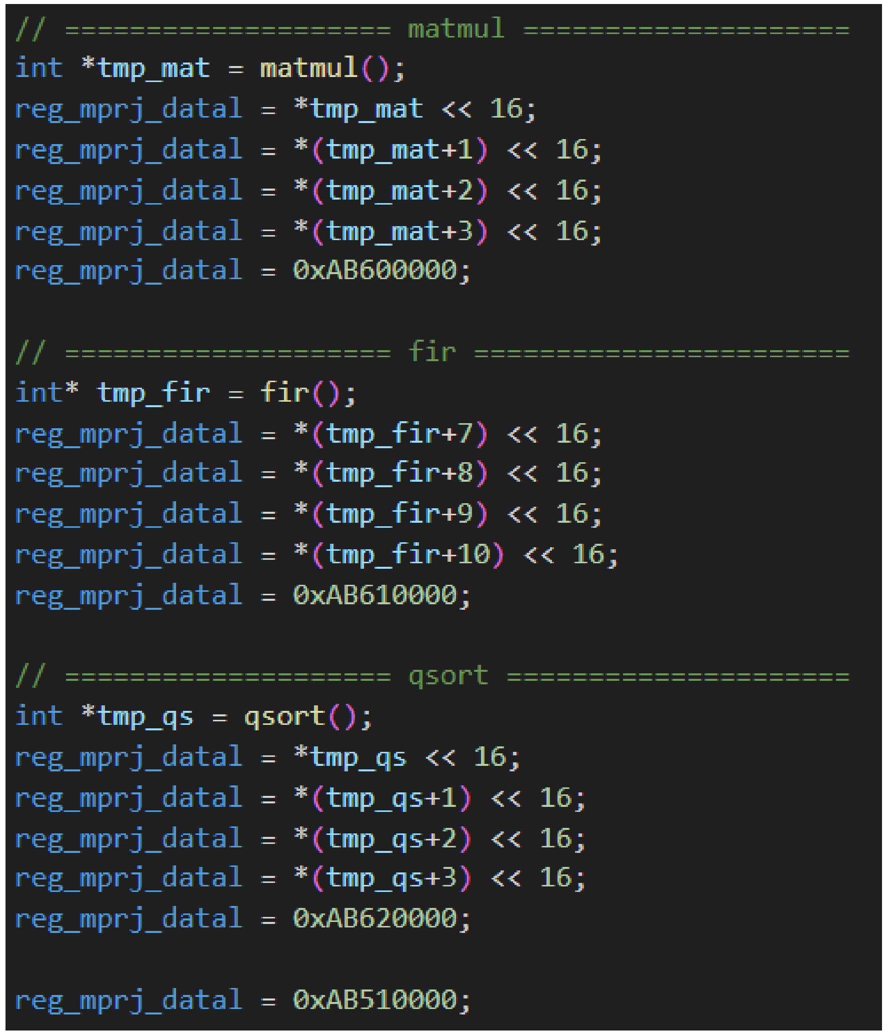
SDRAM Controller with Prefetch

- Based on lab6, matmul, fir, qsor, uart and SDRAM have been **successfully synthesized and verified**
- Use the last stroke of mprj to check the correctness of the operation



```
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 =  0xab510040
0x20 =  0x0
0x34 =  0x0
0x38 =  0x3f
```

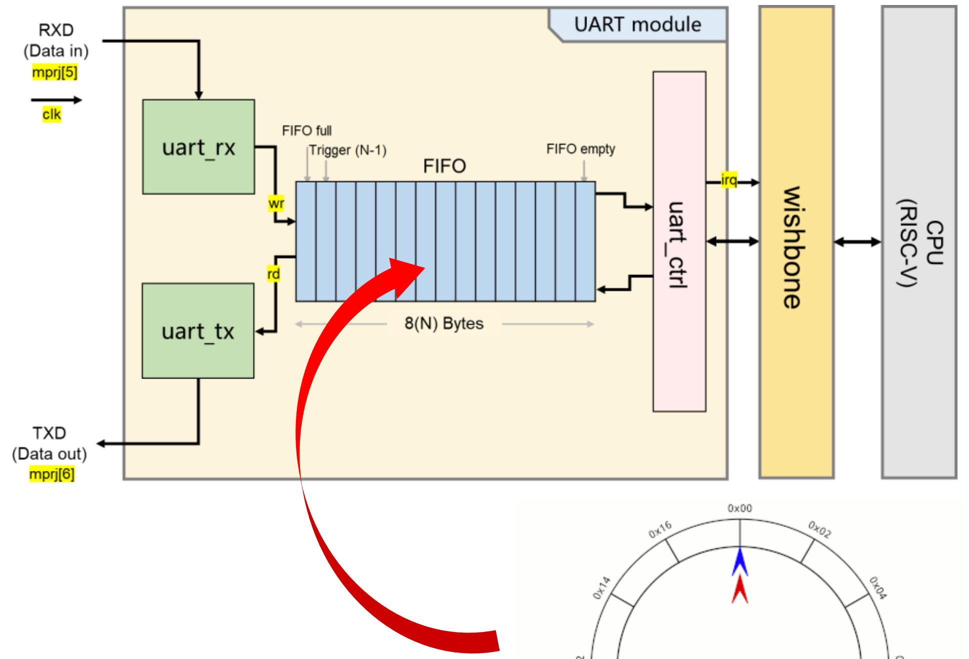


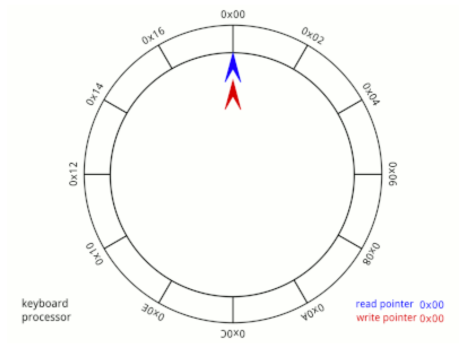
```
// ==================== matmul ====================
int *tmp_mat = matmul();
reg_mprj_datal = *tmp_mat << 16;
reg_mprj_datal = *(tmp_mat+1) << 16;
reg_mprj_datal = *(tmp_mat+2) << 16;
reg_mprj_datal = *(tmp_mat+3) << 16;
reg_mprj_datal = 0xAB600000;

// ==================== fir ======================
int* tmp_fir = fir();
reg_mprj_datal = *(tmp_fir+7) << 16;
reg_mprj_datal = *(tmp_fir+8) << 16;
reg_mprj_datal = *(tmp_fir+9) << 16;
reg_mprj_datal = *(tmp_fir+10) << 16;
reg_mprj_datal = 0xAB610000;

// ==================== qsort ====================
int *tmp_qs = qsort();
reg_mprj_datal = *tmp_qs << 16;
reg_mprj_datal = *(tmp_qs+1) << 16;
reg_mprj_datal = *(tmp_qs+2) << 16;
reg_mprj_datal = *(tmp_qs+3) << 16;
reg_mprj_datal = 0xAB620000;

reg_mprj_datal = 0xAB510000;
```

UART with I/O FIFO

FIFO on Rx/Tx

- **Feature**
  - ➢ FIFO depth (bytes)： 4/8/16/32/64
  - ➢ Baud rate： 9600
  - ➢ Use Circular Buffer as FIFO and one FIFO
    - ➢ Hardware FIFOs have a fixed size, and if data exceeds their capacity, a circular buffer is used to prevent busy waiting
    - ➢ Circular Buffer length： DEPTH - 1





Circular Buffer

```
assign full = ((w_ptr + 1) % DEPTH) == r_ptr;
```

```
assign full = ((w_ptr+1'b1) == r_ptr);
```

- **Goal**
  - ➢ Reduce CPU load and the number of interrupts
    - ✓ Based on our UART interrupt mechanism (Handshake)
  - ➢ Write TX data when ISR finished

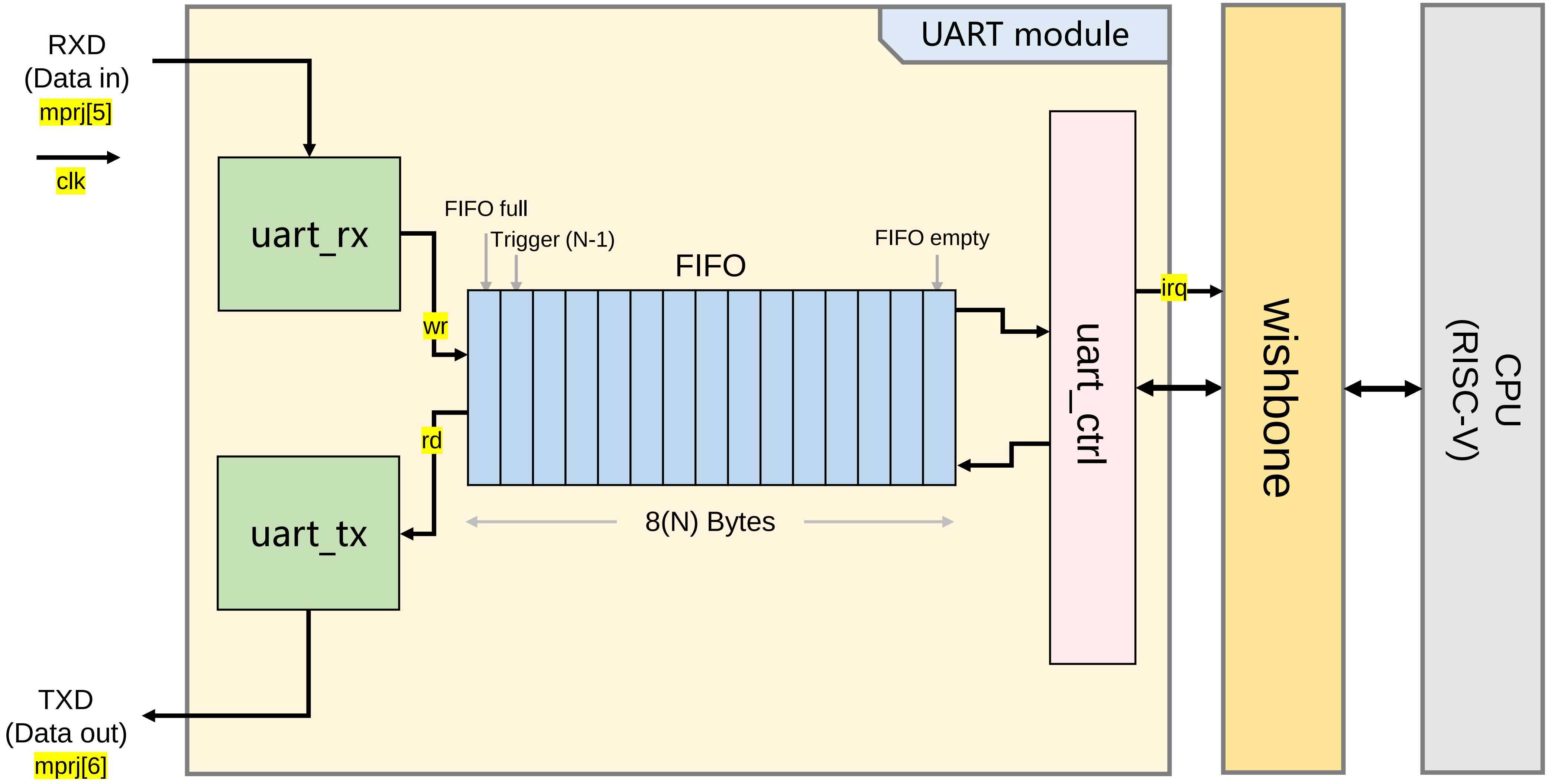
```
#define reg_rx_data          (*(volatile uint32_t*)0x30000000)
#define reg_tx_data          (*(volatile uint32_t*)0x30000004)
#define reg_uart_stat        (*(volatile uint32_t*)0x30000008)
#define reg_uart_clkdiv      (*(volatile uint32_t*)0x3000000c)
#define reg_uart_rx_fifo_cnt (*(volatile uint32_t*)0x30000010)
```

user_uart.h

```
uint32_t irqs = irq_pending() & irq_getmask();
int buf[32] = {0};

if ( irqs & (1 << USER_IRQ_0_INTERRUPT)) {
    user_irq_0_ev_pending_write(1); //Clear Interrupt Pending Event
    int cnt = reg_uart_rx_fifo_cnt;
    for (int i = 0; i < cnt; i++) {
        buf[i] = uart_read();
        uart_write(buf[i]);
    }
}
```

isr.c

UART - Block Diagram



UART FIFO Advantages

- Interrupt management
  - **Reduce the number of interrupts**：**Without using FIFO, an interrupt may be generated every time a character (Byte) is received.** Using the FIFO mechanism, interrupt trigger conditions can be set
    - For example: an interrupt is generated when the FIFO reaches a certain fill level (eg: DEPTH - 1)
  - **Estimated interrupt frequency (Interrupt Frequency)**：Assuming that 1000 Bytes are received per second, the interrupt frequency using FIFO is reduced to approximately 31 times per second, which is a reduction of approximately 97% compared to the original 1000 interrupts
    - If an interrupt is triggered every 32 bytes (Half full), the number of interrupts per second is:

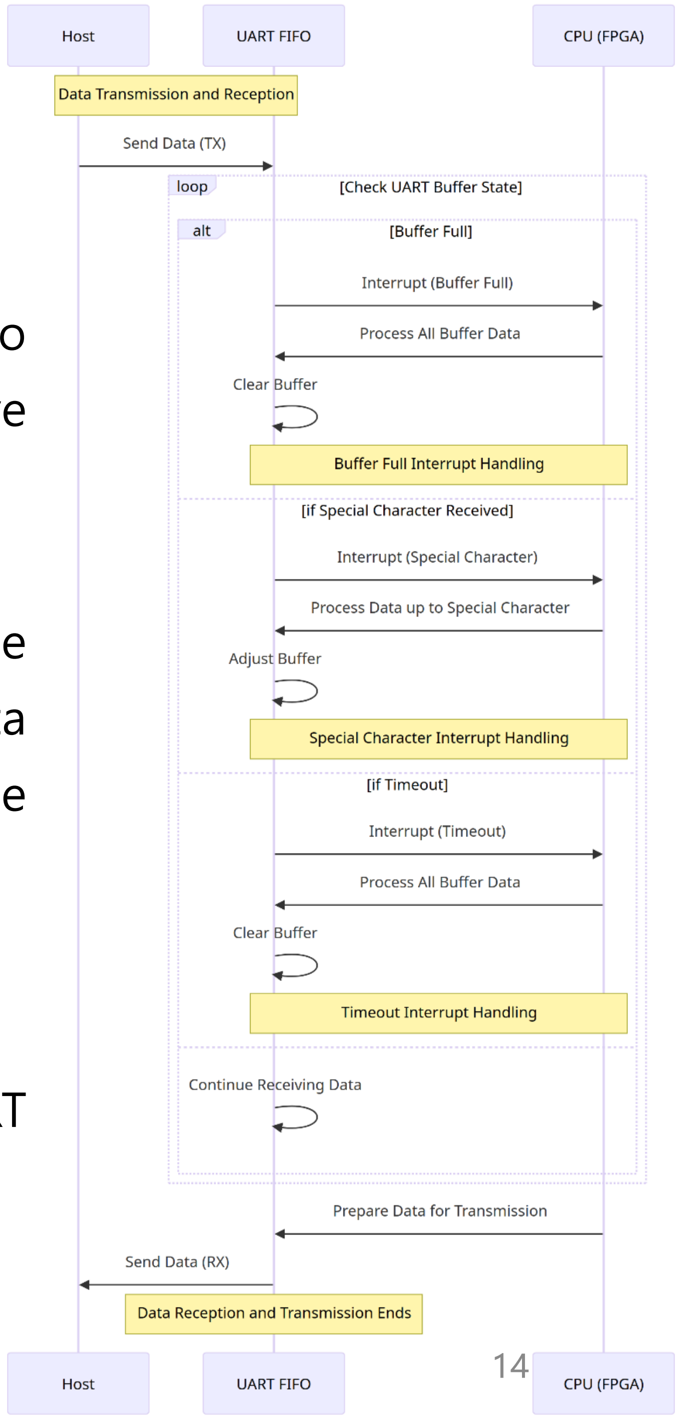
$$Interrupts\ per\ second = \frac{Number\ of\ bytes\ per\ interrupt}{bytes\ per\ second} = \frac{32_{bytes/interrupt}}{1000_{bytes/sec}} \approx 31.25\ times/sec$$

Handshake

1. **Buffer Full：** When the Buffer is full, the UART FIFO will notify the CPU to process all the data in the Buffer, and then clear the Buffer to receive subsequent data

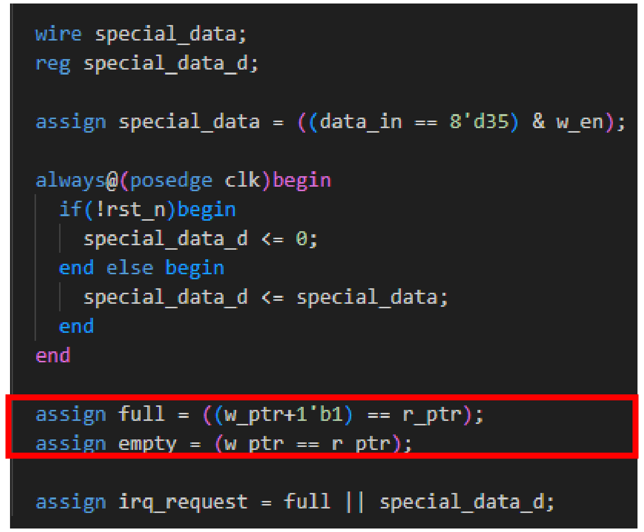
2. **Special Character Received：** If a specific end character is received in the data, such as: \n, the UART FIFO will notify the CPU to process all the data up to the special character, and then adjust the remaining part of the Buffer
   - **Our project defines Special Character as "#", which is 0x35**

3. **Timeout：** If no new data is received for more than the set time, the UART FIFO will notify the CPU to process all Buffer data and clear the Buffer

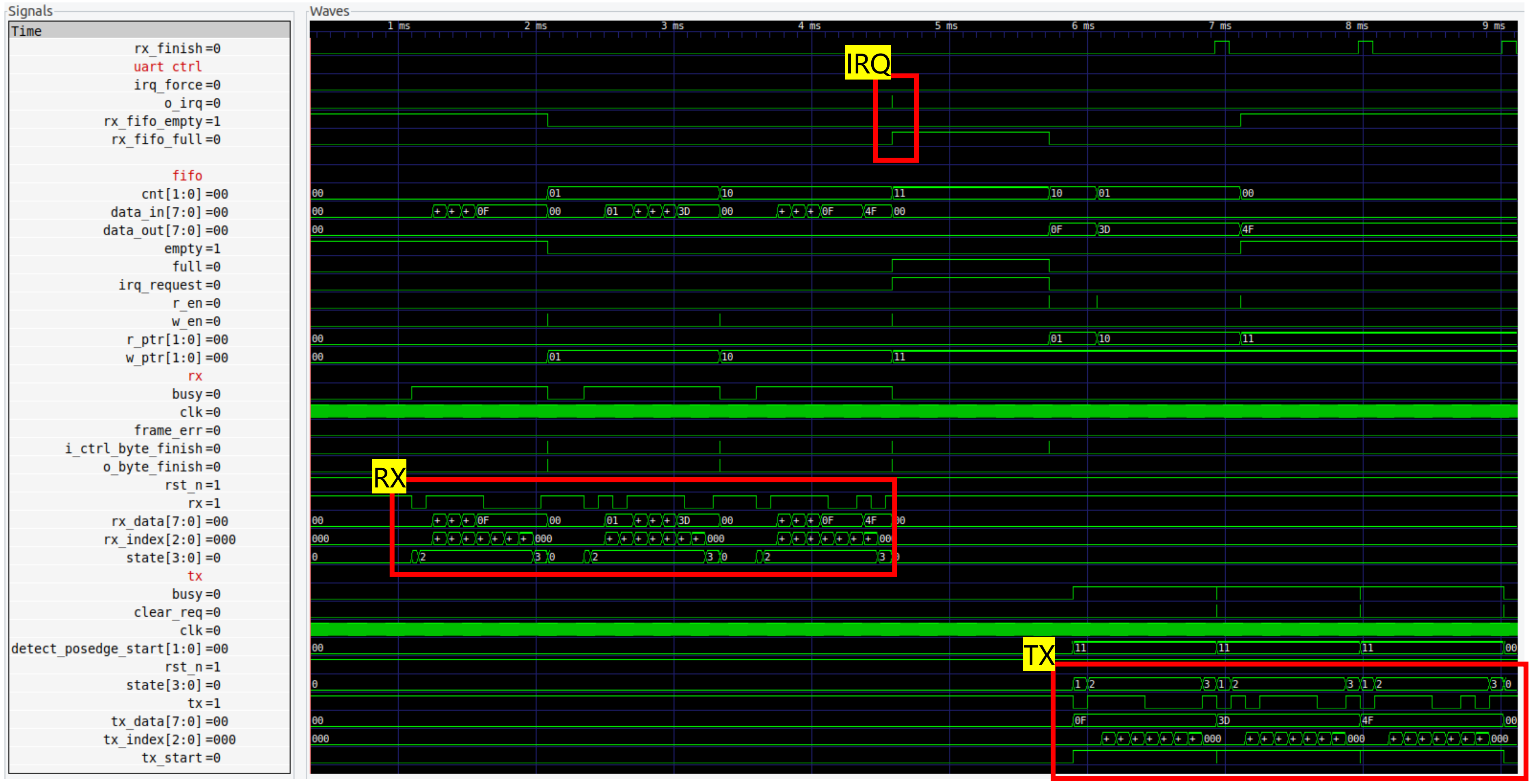

Host
UART FIFO
CPU (FPGA)
Data Transmission and Reception
Send Data (TX)
loop
[Check UART Buffer State]
alt
[Buffer Full]
Interrupt (Buffer Full)
Process All Buffer Data
Clear Buffer
Buffer Full Interrupt Handling
[if Special Character Received]
Interrupt (Special Character)
Process Data up to Special Character
Adjust Buffer
Special Character Interrupt Handling
[if Timeout]
Interrupt (Timeout)
Process All Buffer Data
Clear Buffer
Timeout Interrupt Handling
Continue Receiving Data
Prepare Data for Transmission
Send Data (RX)
Data Reception and Transmission Ends
Host
UART FIFO
CPU (FPGA)

UART - Buffer Full

- DEPTH = 4
- TB send data = 3



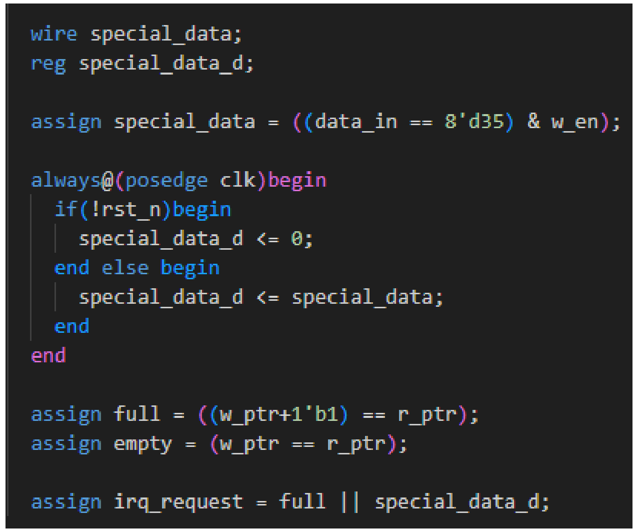
```
wire special_data;
reg special_data_d;

assign special_data = ((data_in == 8'd35) & w_en);

always@(posedge clk)begin
    if(!rst_n)begin
        special_data_d <= 0;
    end else begin
        special_data_d <= special_data;
    end
end

assign full = ((w_ptr+1'b1) == r_ptr);
assign empty = (w_ptr == r_ptr);

assign irq_request = full || special_data_d;
```



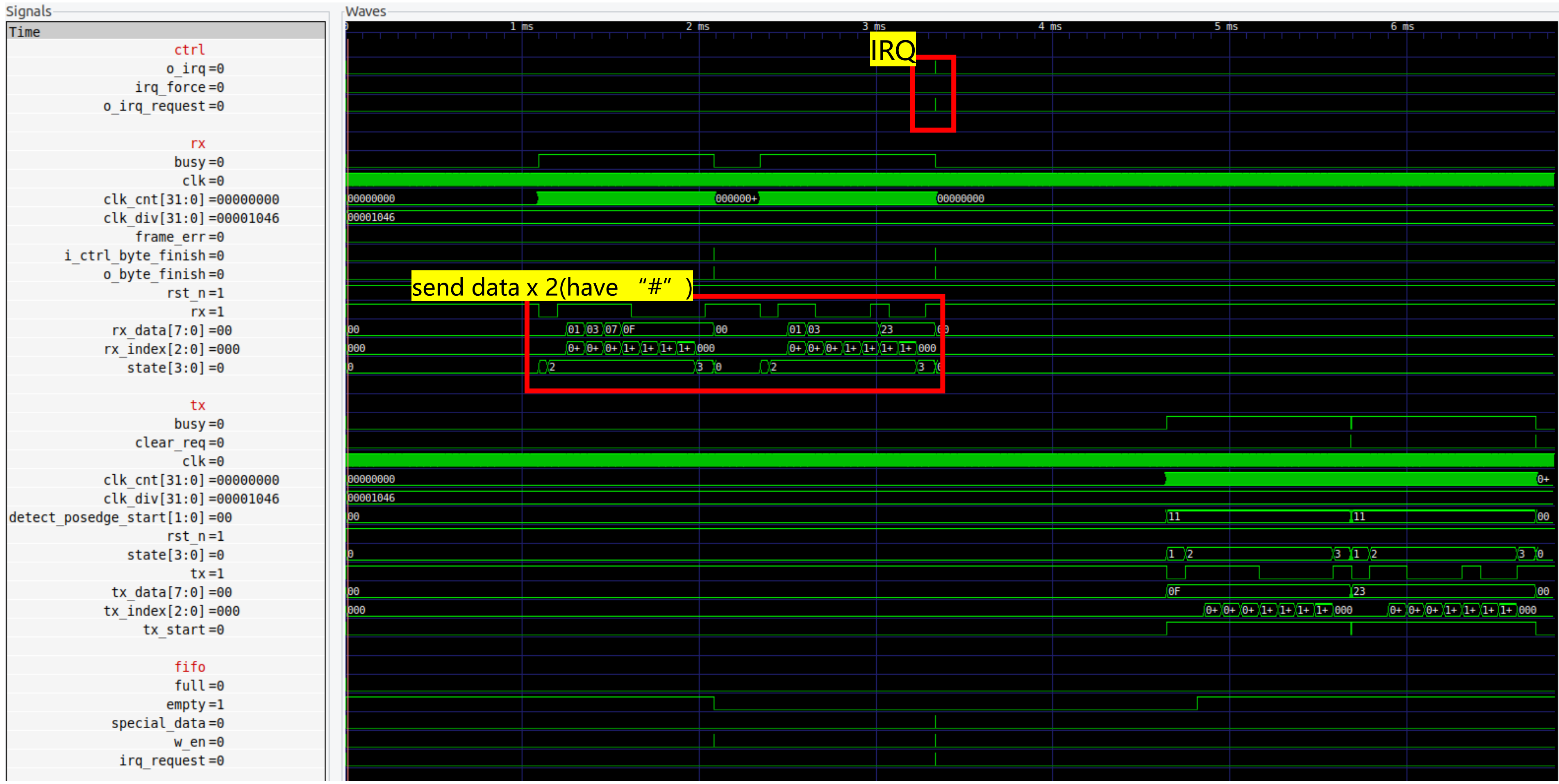
```
Time
rx_finish=0
uart ctrl
irq_force=0
o_irq=0
rx_fifo_empty=1
rx_fifo_full=0
fifo
cnt[1:0]=00
data_in[7:0]=00
data_out[7:0]=00
empty=1
full=0
irq_request=0
r_en=0
w_en=0
r_ptr[1:0]=00
w_ptr[1:0]=00
rx
busy=0
clk=0
frame_err=0
i_ctrl_byte_finish=0
o_byte_finish=0
rst_n=1
rx=1
rx_data[7:0]=00
rx_index[2:0]=000
state[3:0]=0
tx
busy=0
clear_req=0
clk=0
detect_posedge_start[1:0]=00
rst_n=1
state[3:0]=0
tx=1
tx_data[7:0]=00
tx_index[2:0]=000
tx_start=0
```

UART - Special Character Received

- DEPTH = 4
- TB send data = 2
  - Include "#"



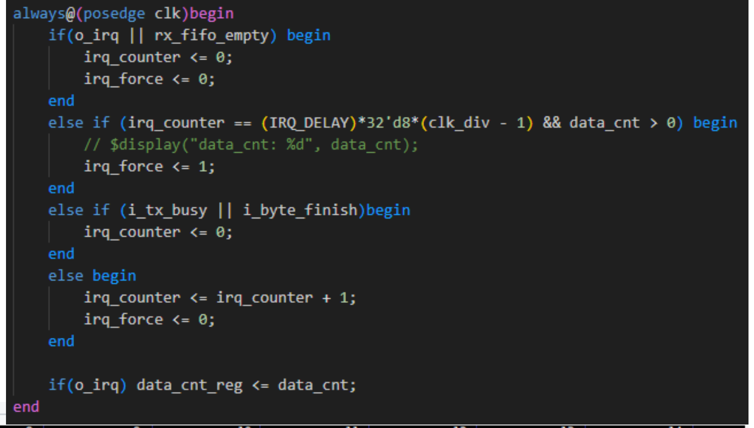
```
wire special_data;
reg special_data_d;

assign special_data = ((data_in == 8'd35) & w_en);

always@(posedge clk)begin
  if(!rst_n)begin
    special_data_d <= 0;
  end else begin
    special_data_d <= special_data;
  end
end

assign full = ((w_ptr+1'b1) == r_ptr);
assign empty = (w_ptr == r_ptr);

assign irq_request = full || special_data_d;
```



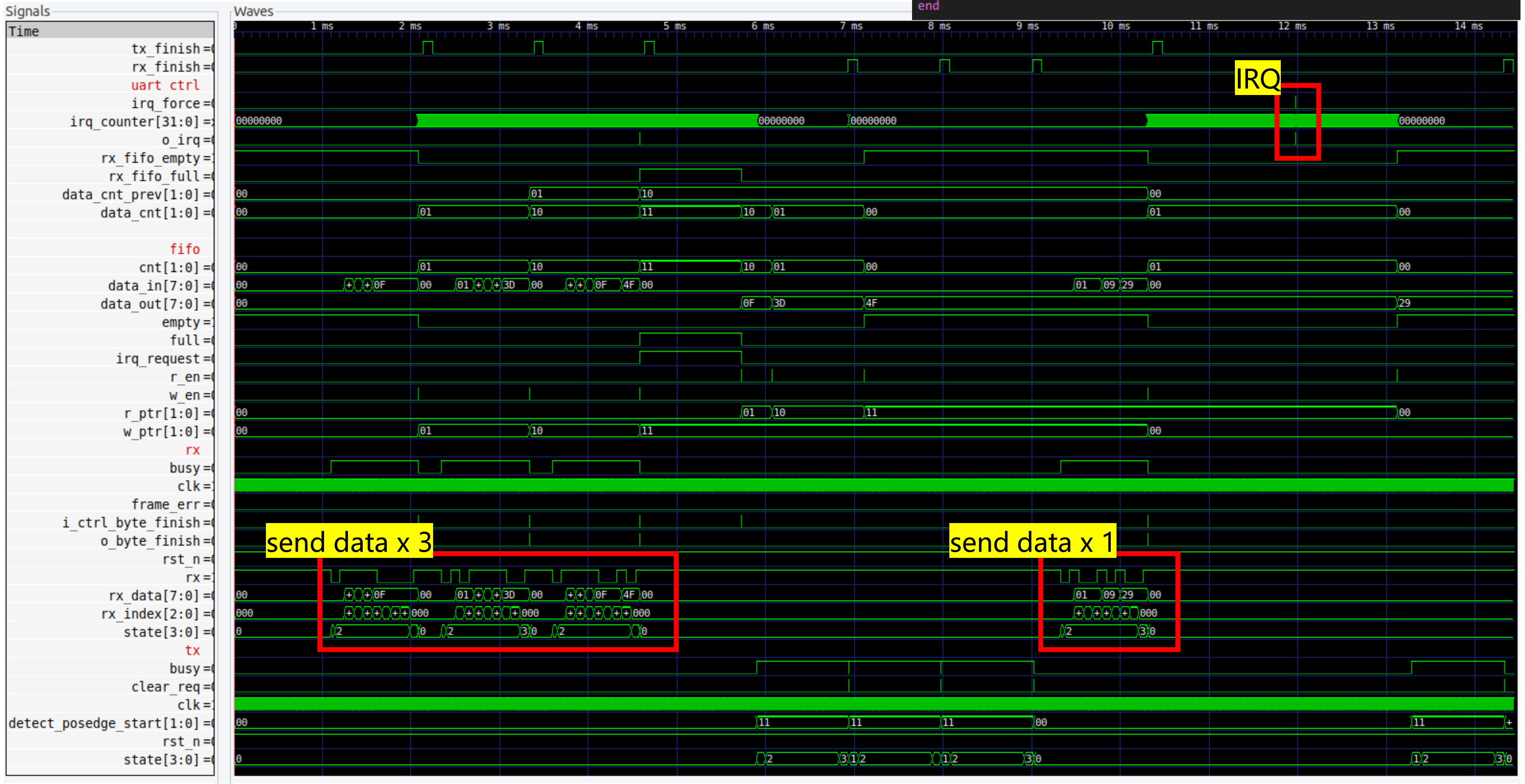
```
Signals
Time
ctrl
o_irq=0
irq_force=0
o_irq_request=0
rx
busy=0
clk=0
clk_cnt[31:0]=00000000
clk_div[31:0]=00001046
frame_err=0
i_ctrl_byte_finish=0
o_byte_finish=0
rst_n=1
rx=1
rx_data[7:0]=00
rx_index[2:0]=000
state[3:0]=0
tx
busy=0
clear_req=0
clk=0
clk_cnt[31:0]=00000000
clk_div[31:0]=00001046
detect_posedge_start[1:0]=00
rst_n=1
state[3:0]=0
tx=1
tx_data[7:0]=00
tx_index[2:0]=000
tx_start=0
fifo
full=0
empty=1
special_data=0
w_en=0
irq_request=0
```

UART - Timeout

- DEPTH = 4
- IRQ_delay = 2
- TB send data = 3
- Then send data = 1

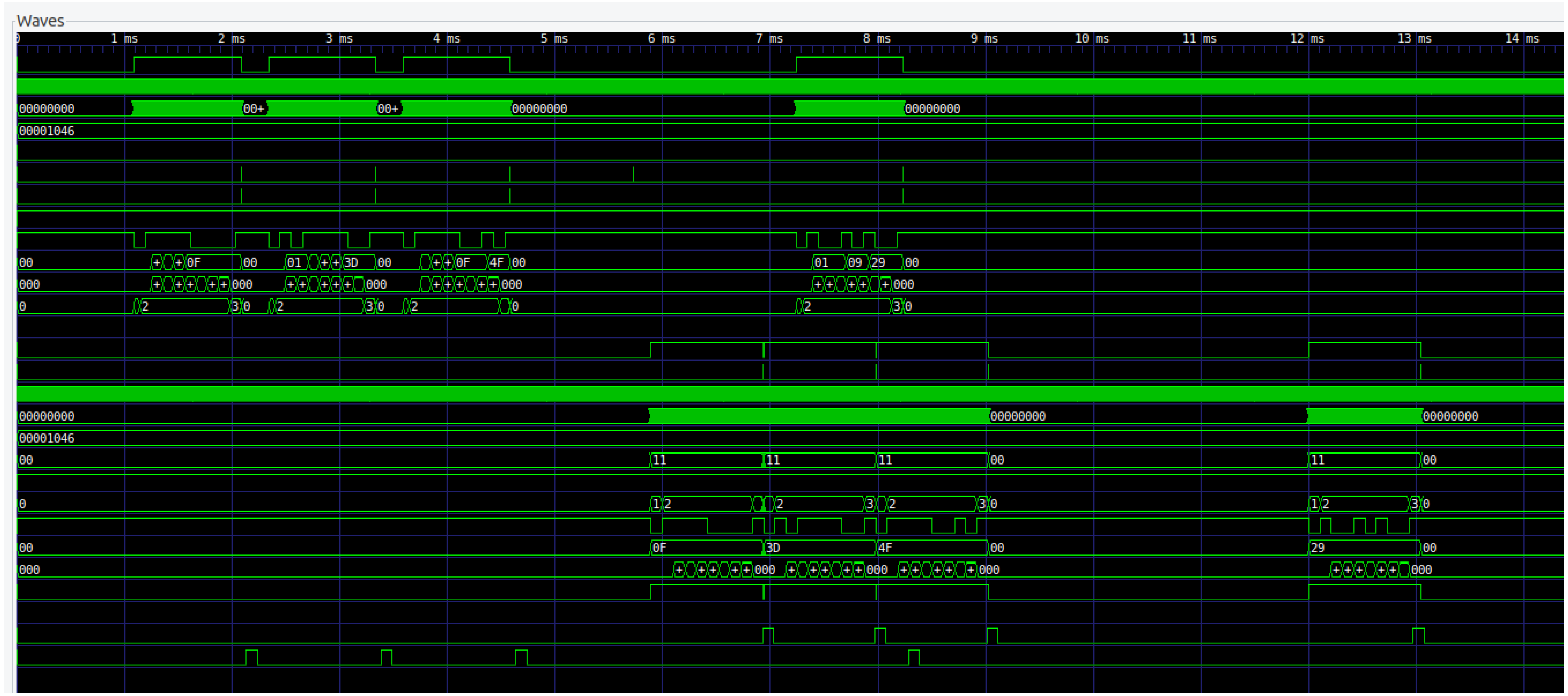


```
always@(posedge clk)begin
    if(o_irq || rx_fifo_empty) begin
        irq_counter <= 0;
        irq_force <= 0;
    end
    else if (irq_counter == (IRQ_DELAY)*32'd8*(clk_div - 1) && data_cnt > 0) begin
        // $display("data_cnt: %d", data_cnt);
        irq_force <= 1;
    end
    else if (i_tx_busy || i_byte_finish)begin
        irq_counter <= 0;
    end
    else begin
        irq_counter <= irq_counter + 1;
        irq_force <= 0;
    end

    if(o_irq) data_cnt_reg <= data_cnt;
end
```



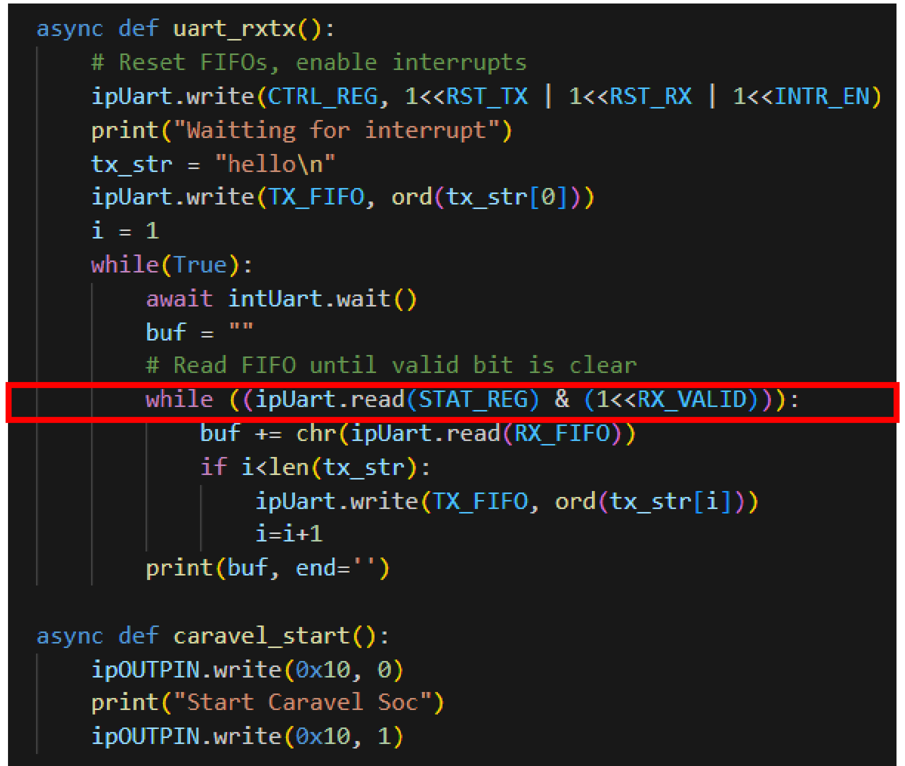
UART FIFO Overlap

- **Shallow FIFO depth:** Overlap has minimal effect on UART speed enhancement
- **Increased FIFO depth:** Overlap significantly boosts UART transmission efficiency

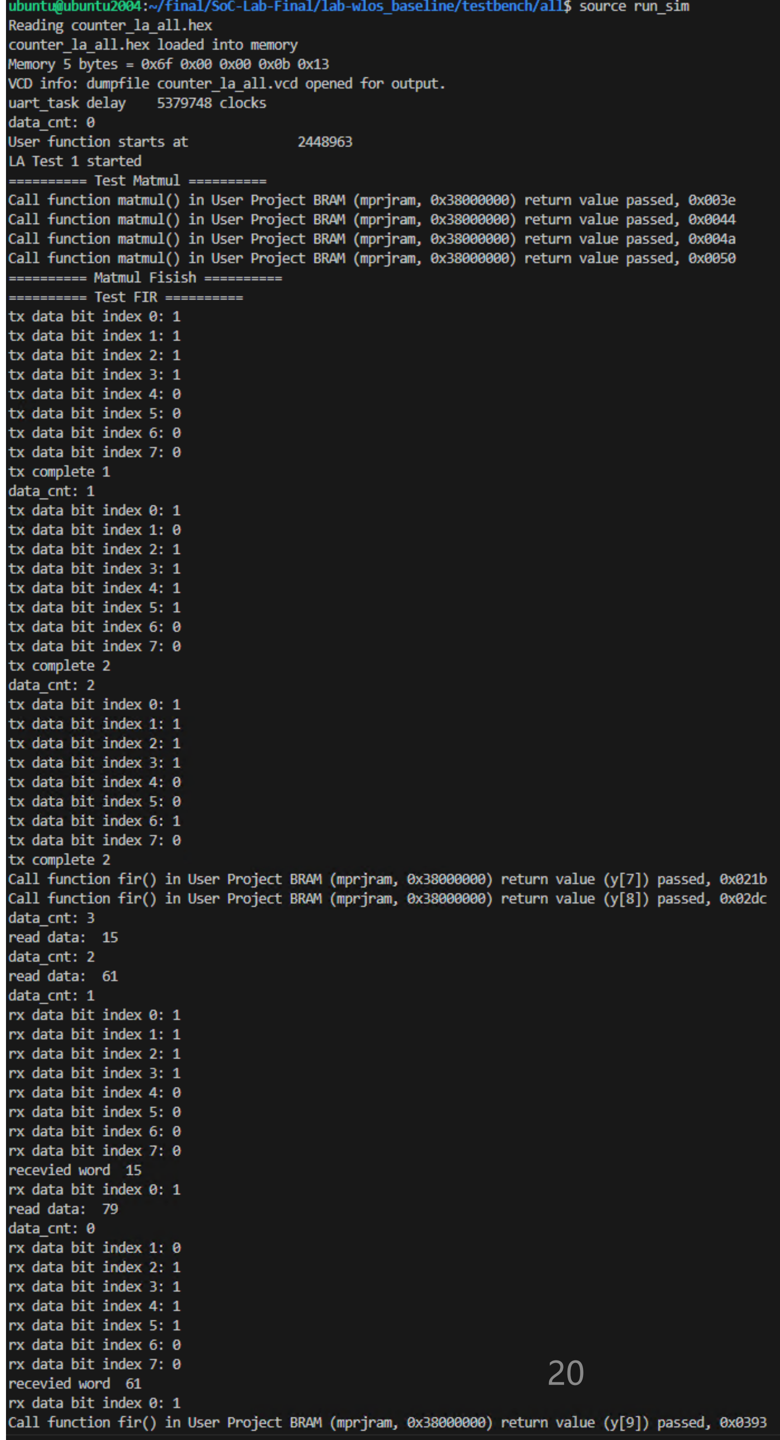


UART FIFO Verification

1. Ability to verify UART FIFO functionality during simulation phase
2. We have tried to synthesize the UART FIFO IP but there is currently an error that prevents us from successfully obtaining RX_VALID



```python
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='')

async def caravel_start():
    ipOUTPIN.write(0x10, 0)
    print("Start Caravel Soc")
    ipOUTPIN.write(0x10, 1)
```



```
ubuntu@ubuntu2004:~/final/SoC-Lab-Final/lab-wlos_baseline/testbench/all$ source run_sim
Reading counter_la_all.hex
counter_la_all.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_all.vcd opened for output.
uart_task delay    5379748 clocks
data_cnt: 0
User function starts at              2448963
LA Test 1 started
========== Test Matmul ==========
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x003e
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0044
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x004a
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0050
========== Matmul Fisish ==========
========== Test FIR ==========
tx data bit index 0: 1
tx data bit index 1: 1
tx data bit index 2: 1
tx data bit index 3: 1
tx data bit index 4: 0
tx data bit index 5: 0
tx data bit index 6: 0
tx data bit index 7: 0
tx complete 1
data_cnt: 1
tx data bit index 0: 1
tx data bit index 1: 0
tx data bit index 2: 1
tx data bit index 3: 1
tx data bit index 4: 1
tx data bit index 5: 1
tx data bit index 6: 0
tx data bit index 7: 0
tx complete 2
data_cnt: 2
tx data bit index 0: 1
tx data bit index 1: 1
tx data bit index 2: 1
tx data bit index 3: 1
tx data bit index 4: 0
tx data bit index 5: 0
tx data bit index 6: 1
tx data bit index 7: 0
tx complete 2
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[7]) passed, 0x021b
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[8]) passed, 0x02dc
data_cnt: 3
read data:  15
data_cnt: 2
read data:  61
data_cnt: 1
rx data bit index 0: 1
rx data bit index 1: 1
rx data bit index 2: 1
rx data bit index 3: 1
rx data bit index 4: 0
rx data bit index 5: 0
rx data bit index 6: 0
rx data bit index 7: 0
recevied word  15
rx data bit index 0: 1
read data:  79
data_cnt: 0
rx data bit index 1: 0
rx data bit index 2: 1
rx data bit index 3: 1
rx data bit index 4: 1
rx data bit index 5: 1
rx data bit index 6: 0
rx data bit index 7: 0
recevied word  61
rx data bit index 0: 1
Call function fir() in User Project BRAM (mprjram, 0x38000000) return value (y[9]) passed, 0x0393
```

Quality of Result (QoS)

- Computation (SDRAM)
  - Optimizer: -O1, -Os

Function	(1) Baseline	(2) Baseline with O1	(3) Ours with O1	(4) Ours with Os
qsort	2045	1061 (Os)	601	X
matmul	4946	2330	1549	1626
fir	8466	6234	4125	X

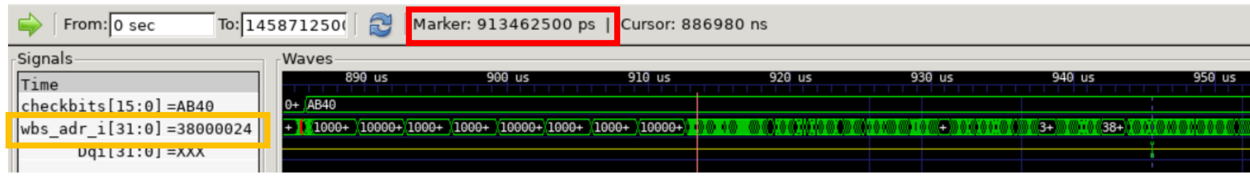
3.4x

3.19x

2.05x

- Communication (UART)
  - Rx transfer time per (8-bit) data
    - 2135412500 - 952312500 = 1183100 ns
  - Tx transfer time pre data
    - 6937312500 - 5894812500 = 1042500 ns
  - irq to isr time
    - 5894812500 - 4583237500 = 1311575 ns
  - **Latency = (rx * #data) + (isr time) + (tx * fifo_len)**
  - **Metrics = 0.6 s - 512 * (1/9600) = 0.54 s (In data=512, fifo_len = 8)**
  - **Speedup = 3.2x**

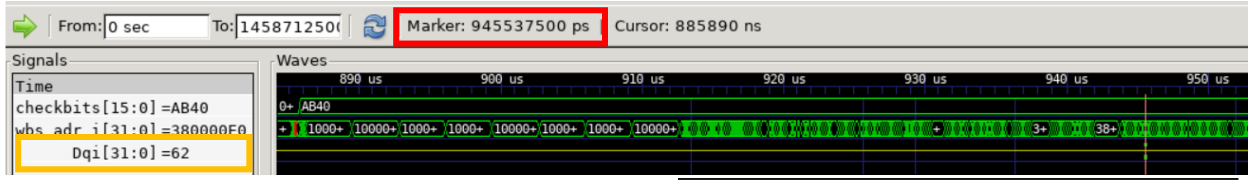
- First function addr



From: 0 sec To: 1458712500 Marker: 913462500 ps | Cursor: 886980 ns

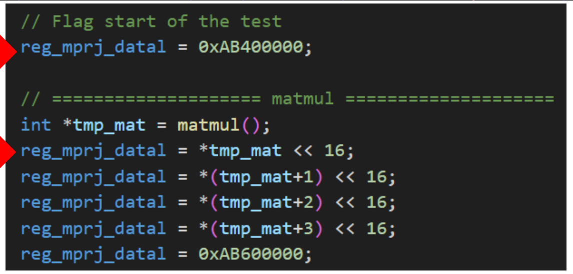
Signals
Time
checkbits[15:0] =AB40
wbs_adr_i[31:0] =38000024
Dqi[31:0] =XXX

- First mprj computed result out



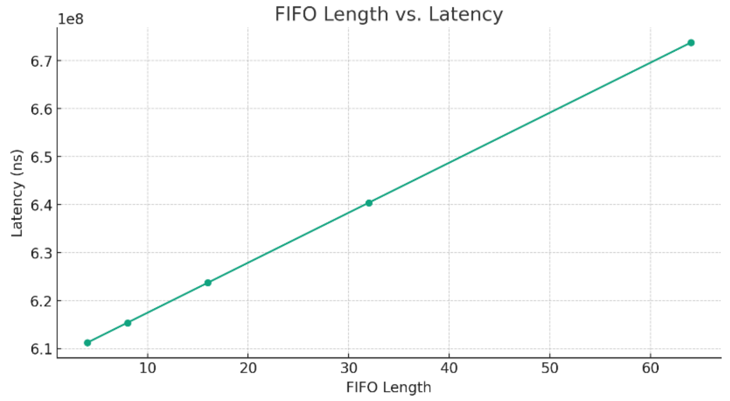
From: 0 sec To: 1458712500 Marker: 945537500 ps Cursor: 885890 ns

Signals
Time
checkbits[15:0] =AB40
wbs_adr_i[31:0] =380000F0
Dqi[31:0] =62



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

// ===================== matmul =====================
int *tmp_mat = matmul();
reg_mprj_datal = *tmp_mat << 16;
reg_mprj_datal = *(tmp_mat+1) << 16;
reg_mprj_datal = *(tmp_mat+2) << 16;
reg_mprj_datal = *(tmp_mat+3) << 16;
reg_mprj_datal = 0xAB600000;
```

# data	fifo_len	latency
512	4	611228775ns
512	8	615398775 ns
512	16	623738775 ns
512	32	640418775 ns
512	64	673778775 ns



Future work

1. Find the problem of UART verification on online FPGA Board
2. Solve the error of compiling with opt parameters with SDRAM
3. Add the TX FIFO in UART and compare with the current uart version

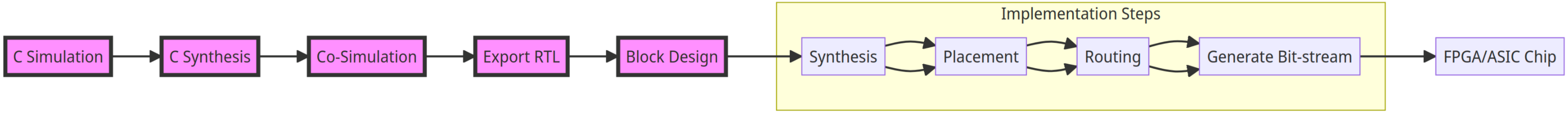
Work Partition

組員一： 林聖博

- 設計 Testbench
- Fireware Code 撰寫
- 整合 User Project Wrapper
- 優化與調整 UART FIFO
- FPGA 測試
- 撰寫報告與統整實驗流程與結果評估

組員二： 張祐誠

- 設計 UART FIFO module
- 設計 FIFO FSM
- 驗證 UART FIFO 邏輯與行為
- 優化與調整 UART FIFO
- FPGA 測試
- 撰寫報告與統整實驗流程與結果評估



Thank you for your attention ~

Q & A

1. Explain the problem with SDRAM prefetch
2. Project status, what not complete?
3. Project content (run in FPGA ?)