

Final Report

Group 11

Members:

111061642 王煒翔

112061578 吳松煌

112061601 吳睿宸

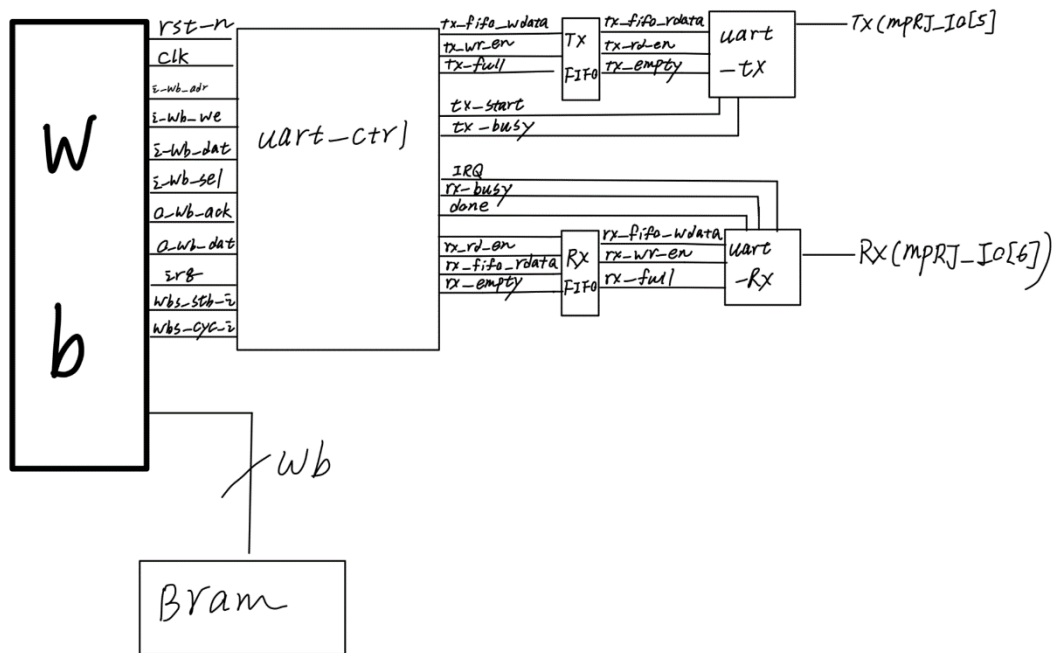
UART FIFO

1. Introduction & Block Diagram:

<Introduction>

原本 Lab 6 的 UART 是沒有加入 FIFO 的，因此每收到一筆資料就要進行一次中斷，如此頻繁性的中斷，不僅會造成 CPU 資源的消耗，也可能造成前一筆的數據還沒有讀完，下一筆數據就已經寫入，覆蓋之前的數據。因此本次的期末 project 是要在 TX 和 RX 兩端皆加入一個 FIFO，使之每讀到一定的數據量才會進行一次中斷，以減少資源的消耗。

<Block Diagram>



當 tx_start 升起到 1 的時候，TX 藉由 RX_mprjIO[6] 開始傳送資料到 UART_RX，接著經過 1 個 RX FIFO 和 UART_ctrl，透過 WB 把資料傳輸給 CPU 端，而在傳輸的部分，藉由 WB、TX、FIFO、UART_TX 和 TX_mprjIO[5] 傳回。

2. Code

Module synchronous FIFO

```
module synfifo#(
    parameter depth = 4
)(
    input clk,
    input rst,
    input wr_en,
    input rd_en,
    input [31:0] wr_data,
    output reg [31:0] rd_data,
    output wire fifo_full,
    output wire fifo_empty
);
```

Data write in FIFO

```
// data write in fifo
always@(posedge clk or negedge rst)begin
    if(!rst) begin
        for(i=0;i<depth;i=i+1) mem[i] <= 32'b0;
    end
    else if(wr_en) mem[wr_ptr] <= wr_data;
    else mem[wr_ptr] <= mem[wr_ptr];
end
```

Data read from FIFO

```
// data read from fifo
always@(posedge clk or negedge rst)begin
    if(!rst) rd_data <= 32'b0;
    else if(rd_en) rd_data <= mem[rd_ptr];
    else rd_data <= rd_data;
end
```

RX FIFO

```
// RX write in rx fifo
always@(posedge clk or negedge rst_n)begin
    if(!rst_n)begin
        rx_fifo_wdata <= 0;
        rx_wr_en <= 0;
    end else begin
        if(done && !stat_reg[1] && !i_frame_err && !rx_full)begin //rx can write in fifo rx finish
            rx_fifo_wdata <= {24'b0,i_rx};
            rx_wr_en <= 1;
            //$display("rx_buffer: %d", i_rx);
        end
        else
            rx_wr_en <= 0;
    end
end
```

TX FIFO read

```
//tx fifo read
always@(posedge clk or negedge rst_n)begin
    if(!rst_n || i_tx_start_clear)begin
        tx_rd_en<=0;
    end else begin
        if (tx_rd_en) tx_rd_en <= 0;
        else if(/*i_wb_valid && i_wb_we && i_wb_adr==TX_DATA &&*/ !i_tx_busy && !tx_empty && stat_reg[3:2]==2'b01 && !tx_start_local )begin
            tx_rd_en<=1;
        end
        else tx_rd_en <=0;
    end
end
```

TX FIFO write

```
//tx fifo write
always@(posedge clk or negedge rst_n)begin
    if(!rst_n)begin
        rd_buffer<= 0;
    end else begin
        if(i_wb_valid && rx_rd_en)
            rd_buffer <= 1;
        else
            rd_buffer <= 0;
    end
end
always@(posedge clk or negedge rst_n)begin
    if(!rst_n)begin
        tx_fifo_wdata <=32'b0;
        tx_wr_en <= 1'b0;
    end else begin
        if(rd_buffer && !tx_full)begin
            tx_fifo_wdata <= rx_fifo_rdata;
            tx_wr_en <= 1'b1;
        end
        else begin
            tx_fifo_wdata <= tx_fifo_wdata;
            tx_wr_en<=1'b0;
        end
    end
end
end
```

3.Simulation Result& Waveform

(1)run_sim result:

5 LA Test 1 started		
6 tx data bit index 0: 1	42 tx data bit index 0: 1	78 rx data bit index 0: 1
7 tx data bit index 1: 0	43 tx data bit index 1: 0	79 rx data bit index 1: 1
8 tx data bit index 2: 0	44 tx data bit index 2: 1	80 tx data bit index 0: 1
9 tx data bit index 3: 0	45 tx data bit index 3: 0	81 tx data bit index 1: 1
10 tx data bit index 4: 0	46 rx data bit index 0: 1	82 rx data bit index 2: 0
11 tx data bit index 5: 0	47 rx data bit index 1: 0	83 rx data bit index 3: 0
12 tx data bit index 6: 0	48 tx data bit index 4: 0	84 tx data bit index 2: 1
13 tx data bit index 7: 0	49 tx data bit index 5: 0	85 tx data bit index 3: 0
14 tx data1 complete	50 rx data bit index 2: 0	86 rx data bit index 4: 0
15 tx data bit index 0: 0	51 rx data bit index 3: 0	87 rx data bit index 5: 0
16 tx data bit index 1: 1	52 tx data bit index 6: 0	88 tx data bit index 4: 0
17 tx data bit index 2: 0	53 tx data bit index 7: 0	89 tx data bit index 5: 0
18 tx data bit index 3: 0	54 rx data bit index 4: 0	90 rx data bit index 6: 0
19 tx data bit index 4: 0	55 rx data bit index 5: 0	91 rx data bit index 7: 0
20 tx data bit index 5: 0	56 tx data5 complete	92 tx data bit index 6: 0
21 tx data bit index 6: 0	57 rx data bit index 6: 0	93 tx data bit index 7: 0
22 tx data bit index 7: 0	58 rx data bit index 7: 0	94 recevied word 3
23 tx data2 complete	59 recevied word 1	95 tx data7 complete
24 tx data bit index 0: 1	60 tx data bit index 0: 0	96 rx data bit index 0: 0
25 tx data bit index 1: 1	61 tx data bit index 1: 1	97 rx data bit index 1: 0
26 tx data bit index 2: 0	62 rx data bit index 0: 0	98 rx data bit index 2: 1
27 tx data bit index 3: 0	63 rx data bit index 1: 1	99 rx data bit index 3: 0
28 tx data bit index 4: 0	64 tx data bit index 2: 1	100 tx data bit index 0: 0
29 tx data bit index 5: 0	65 tx data bit index 3: 0	101 tx data bit index 1: 0
30 tx data bit index 6: 0	66 rx data bit index 2: 0	102 rx data bit index 4: 0
31 tx data bit index 7: 0	67 rx data bit index 3: 0	103 rx data bit index 5: 0
32 tx data3 complete	68 tx data bit index 4: 0	104 tx data bit index 2: 0
33 tx data bit index 0: 0	69 tx data bit index 5: 0	105 tx data bit index 3: 1
34 tx data bit index 1: 0	70 rx data bit index 4: 0	106 rx data bit index 6: 0
35 tx data bit index 2: 1	71 rx data bit index 5: 0	107 rx data bit index 7: 0
36 tx data bit index 3: 0	72 tx data bit index 6: 0	108 tx data bit index 4: 0
37 tx data bit index 4: 0	73 tx data bit index 7: 0	109 tx data bit index 5: 0
38 tx data bit index 5: 0	74 rx data bit index 6: 0	110 recevied word 4
39 tx data bit index 6: 0	75 rx data bit index 7: 0	111 tx data bit index 6: 0
40 tx data bit index 7: 0	76 tx data6 complete	112 tx data bit index 7: 0
41 tx data4 complete	77 recevied word 2	113 tx data8 complete

114 tx data bit index 0: 1	149 received word 6	186 tx data bit index 0: 1
115 tx data bit index 1: 0	150 rx data bit index 0: 1	187 tx data bit index 1: 0
116 tx data bit index 2: 0	151 rx data bit index 1: 1	188 tx data bit index 2: 1
117 tx data bit index 3: 1	152 tx data bit index 0: 1	189 tx data bit index 3: 1
118 rx data bit index 0: 1	153 tx data bit index 1: 1	190 rx data bit index 0: 1
119 rx data bit index 1: 0	154 rx data bit index 2: 1	191 rx data bit index 1: 0
120 tx data bit index 4: 0	155 rx data bit index 3: 0	192 tx data bit index 4: 0
121 tx data bit index 5: 0	156 tx data bit index 2: 0	193 tx data bit index 5: 0
122 rx data bit index 2: 1	157 tx data bit index 3: 1	194 rx data bit index 2: 0
123 rx data bit index 3: 0	158 rx data bit index 4: 0	195 rx data bit index 3: 1
124 tx data bit index 6: 0	159 rx data bit index 5: 0	196 tx data bit index 6: 0
125 tx data bit index 7: 0	160 tx data bit index 4: 0	197 tx data bit index 7: 0
126 rx data bit index 4: 0	161 tx data bit index 5: 0	198 rx data bit index 4: 0
127 rx data bit index 5: 0	162 rx data bit index 6: 0	199 rx data bit index 5: 0
128 tx data9 complete	163 rx data bit index 7: 0	200 tx data13 complete
129 rx data bit index 6: 0	164 tx data bit index 6: 0	201 rx data bit index 6: 0
130 rx data bit index 7: 0	165 tx data bit index 7: 0	202 rx data bit index 7: 0
131 received word 5	166 received word 7	203 received word 9
132 tx data bit index 0: 0	167 tx data11 complete	204 tx data bit index 0: 0
133 tx data bit index 1: 1	168 rx data bit index 0: 0	205 tx data bit index 1: 1
134 rx data bit index 0: 0	169 rx data bit index 1: 0	206 rx data bit index 0: 0
135 rx data bit index 1: 1	170 rx data bit index 2: 0	207 rx data bit index 1: 1
136 tx data bit index 2: 0	171 rx data bit index 3: 1	208 tx data bit index 2: 1
137 tx data bit index 3: 1	172 tx data bit index 0: 0	209 tx data bit index 3: 1
138 rx data bit index 2: 1	173 tx data bit index 1: 0	210 rx data bit index 2: 0
139 rx data bit index 3: 0	174 rx data bit index 4: 0	211 rx data bit index 3: 1
140 tx data bit index 4: 0	175 rx data bit index 5: 0	212 tx data bit index 4: 0
141 tx data bit index 5: 0	176 tx data bit index 2: 1	213 tx data bit index 5: 0
142 rx data bit index 4: 0	177 tx data bit index 3: 1	214 rx data bit index 4: 0
143 rx data bit index 5: 0	178 rx data bit index 6: 0	215 rx data bit index 5: 0
144 tx data bit index 6: 0	179 rx data bit index 7: 0	216 tx data bit index 6: 0
145 tx data bit index 7: 0	180 tx data bit index 4: 0	217 tx data bit index 7: 0
146 rx data bit index 6: 0	181 tx data bit index 5: 0	218 rx data bit index 6: 0
147 rx data bit index 7: 0	182 received word 8	219 rx data bit index 7: 0
148 tx data10 complete	183 tx data bit index 6: 0	220 tx data14 complete
	184 tx data bit index 7: 0	221 received word 10
	185 tx data12 complete	

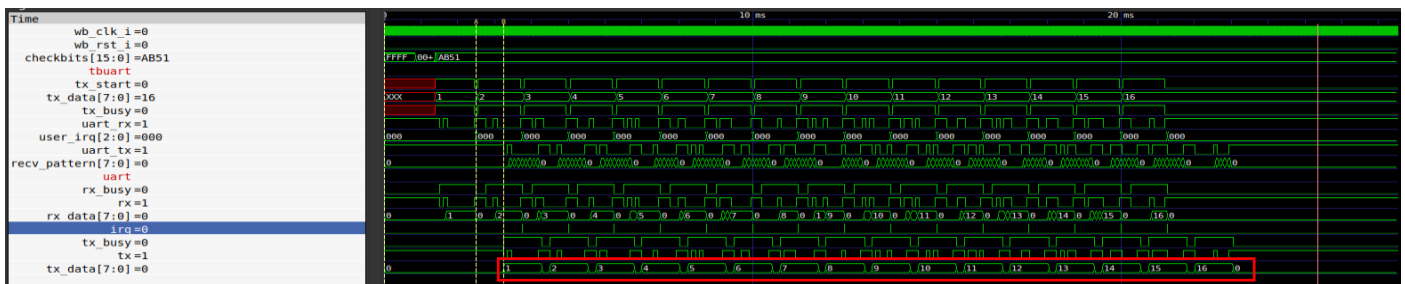
```

222 rx data bit index 0: 1 257 tx data16 complete
223 rx data bit index 1: 1 258 tx complete
224 tx data bit index 0: 1 259 rx data bit index 0: 1
225 tx data bit index 1: 1 260 rx data bit index 1: 0
226 rx data bit index 2: 0 261 rx data bit index 2: 1
227 rx data bit index 3: 1 262 rx data bit index 3: 1
228 tx data bit index 2: 1 263 rx data bit index 4: 0
229 tx data bit index 3: 1 264 rx data bit index 5: 0
230 rx data bit index 4: 0 265 rx data bit index 6: 0
231 rx data bit index 5: 0 266 rx data bit index 7: 0
232 tx data bit index 4: 0 267 received word 13
233 tx data bit index 5: 0 268 rx data bit index 0: 0
234 rx data bit index 6: 0 269 rx data bit index 1: 1
235 rx data bit index 7: 0 270 rx data bit index 2: 1
236 tx data bit index 6: 0 271 rx data bit index 3: 1
237 tx data bit index 7: 0 272 rx data bit index 4: 0
238 received word 11 273 rx data bit index 5: 0
239 tx data15 complete 274 rx data bit index 6: 0
240 rx data bit index 0: 0 275 rx data bit index 7: 0
241 rx data bit index 1: 0 276 received word 14
242 rx data bit index 2: 1 277 rx data bit index 0: 1
243 rx data bit index 3: 1 278 rx data bit index 1: 1
244 tx data bit index 0: 0 279 rx data bit index 2: 1
245 tx data bit index 1: 0 280 rx data bit index 3: 1
246 rx data bit index 4: 0 281 rx data bit index 4: 0
247 rx data bit index 5: 0 282 rx data bit index 5: 0
248 tx data bit index 2: 0 283 rx data bit index 6: 0
249 tx data bit index 3: 0 284 rx data bit index 7: 0
250 rx data bit index 6: 0 285 received word 15
251 rx data bit index 7: 0 286 rx data bit index 0: 0
252 tx data bit index 4: 1 287 rx data bit index 1: 0
253 tx data bit index 5: 0 288 rx data bit index 2: 0
254 received word 12 289 rx data bit index 3: 0
255 tx data bit index 6: 0 290 rx data bit index 4: 1
256 tx data bit index 7: 0 291 rx data bit index 5: 0
257 tx data16 complete 292 rx data bit index 6: 0
293 rx data bit index 7: 0
294 received word 16

```

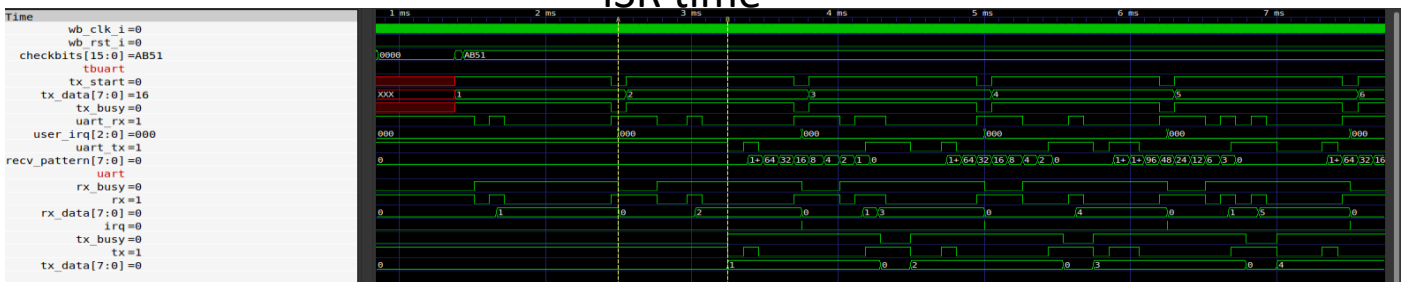
(2)Waveform:

Without FIFO:



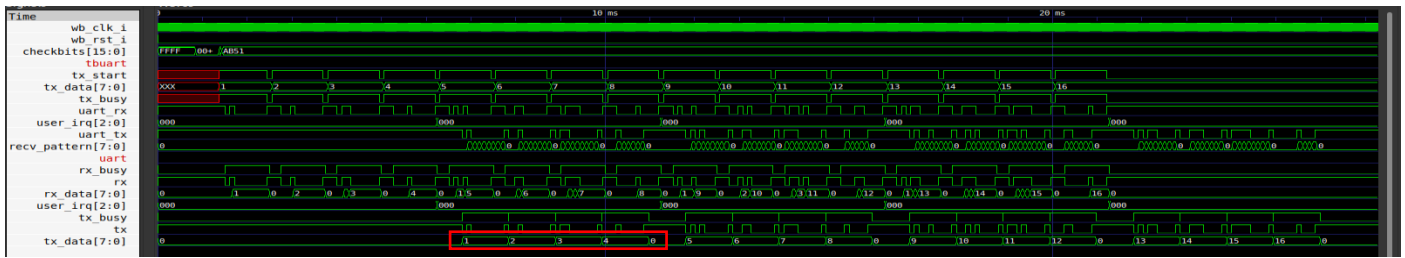
沒有加 FIFO 的時候，會一次性傳完所有的資料。

ISR time



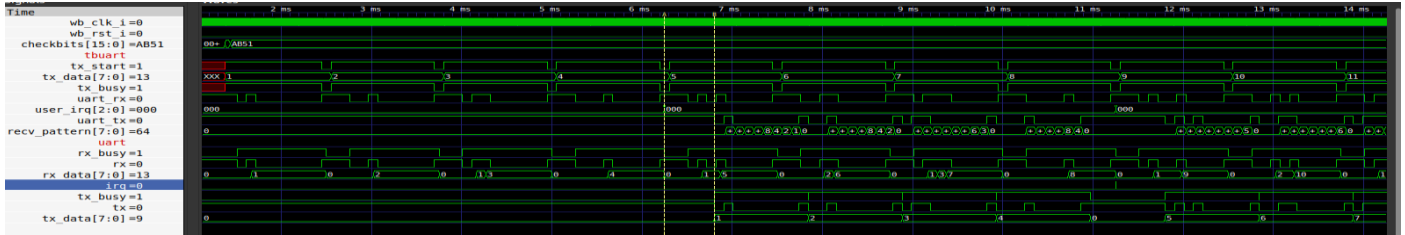
Interrupt one time per data:(data transmission time*2)+ISR time=(41664*2)+29892= 113220 cycle per data.

With FIFO:



從以上有加 FIFO 的圖可以知道，當 user_irq 發出 interrupt(000)訊號的時候，tx_busy 會等一段時間才會到 1，而此時 tx_data 開始傳送訊號，每傳送 4 筆資料即進行一次中斷。

ISR time



Interrupt one time per 4 data: $((\text{data transmission time}) * 4 + \text{ISR time}) / 4 = (41667 * 4 + 22024) / 4 = 47173$ cycle per data.

<Compare>從以上的波形圖可以得知，沒加 FIFO 的 per cycle 數為 113220，有加 FIFO 的 per cycle 數為 47173，兩者的 cycle 數相差約略 3 倍。

<UART latency>

Message length: 512 characters, Baud rate: 9600

Without FIFO:

Cycles = 1227075 \rightarrow QoR (Quality Of Resolve) = $1227075 * 25\text{ns} - 512 * (1/9600) = 30.676\text{ms}$

With FIFO:

Cycles = 847911 \rightarrow QoR = $847911 * 25\text{ns} - 512 * (1/9600) = 21.197\text{ms}$

\rightarrow 可以看到 cycle 減少，QoR 少了約略 10ms

4.Synthesis report

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	5367	0	0	53200	10.09
LUT as Logic	5179	0	0	53200	9.73
LUT as Memory	188	0	0	17400	1.08
LUT as Distributed RAM	18	0			
LUT as Shift Register	170	0			
Slice Registers	6195	0	0	106400	5.82
Register as Flip Flop	6195	0	0	106400	5.82
Register as Latch	0	0	0	106400	0.00
F7 Muxes	169	0	0	26600	0.64
F8 Muxes	47	0	0	13300	0.35

Without FIFO

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	5494	0	0	53200	10.33
LUT as Logic	5306	0	0	53200	9.97
LUT as Memory	188	0	0	17400	1.08
LUT as Distributed RAM	18	0			
LUT as Shift Register	170	0			
Slice Registers	6417	0	0	106400	6.03
Register as Flip Flop	6417	0	0	106400	6.03
Register as Latch	0	0	0	106400	0.00
F7 Muxes	170	0	0	26600	0.64
F8 Muxes	47	0	0	13300	0.35

With FIFO

	Without FIFO	With FIFO	Compare
LUTs	5367	5494	2.4%more LUTs
Slice Registers	6195	6417	3.5%more Slice Registers

5.FPGA Verification:

We use pynq board to verify.

Without FIFO

```
Start Caravel Soc
Waiting for interrupt
hello
main(): uart_rx is cancelled now
UART latency: [0.004399538040161133, 0.009931325912475586, 0.015253305435180664, 0.020830154418945312, 0.026195287704467773, 0.03179597854614258]
Total UART latency: 0.10840559005737305
Average UART latency: 0.018067598342895508
```

沒加 FIFO 的平均 latency 是 0.018s 。

With FIFO

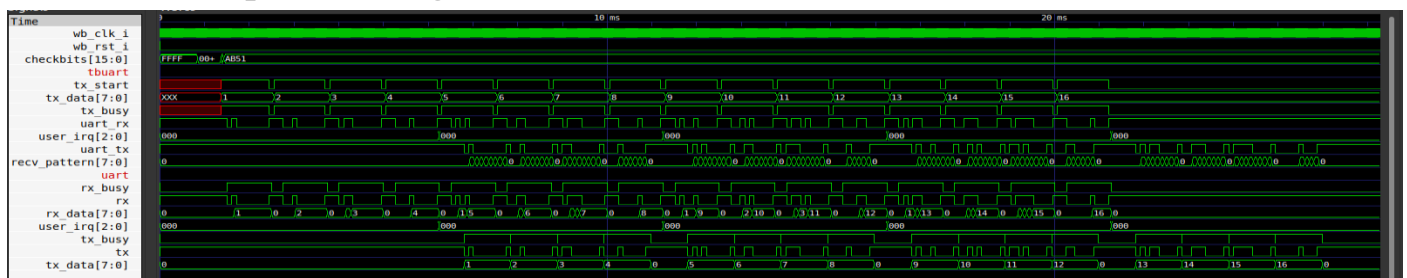
```
Start Caravel Soc
Waiting for interrupt
hello
main(): uart_rx is cancelled now
UART latency: [0.0070950984954833984, 0.011205911636352539, 0.016241788864135742, 0.019147872924804688, 0.022225141525268555, 0.025310754776000977]
Total UART latency: 0.1012265682220459
Average UART latency: 0.016871094703674316
```

有加 FIFO 的平均 latency 是 0.0168s 。

→在 FPGA 板上驗證的時候，可以看到有加 FIFO 的平均 latency 是 0.0168s 明顯比沒加 FIFO 的平均 latency 還要少。

6.Analysis & Insights

- FIFO Depth choosing



Our FIFO Depth are using 4,But we also try other FIFO depth:

For depth = 2 → 52769 cycles per data → higher than depth = 4

For depth = 8 → 44420 cycles per data → slightly lower than depth = 4, but larger power consumption and area!