

SoC Lab: Final Project

Workload Optimized SoC

Team 1

邱政岡 311511022

蔡宇冠 311514037

高振翔 412010020

Outline

- Improvement
- SDRAM with prefetch controller
- UART with FIFO
- Hardware software co-design // DMA
- DMA using exmem_pipeline
- Hardware Accelerator (FIR 、 MM 、 Qsort)

Improvement

- Workload latency

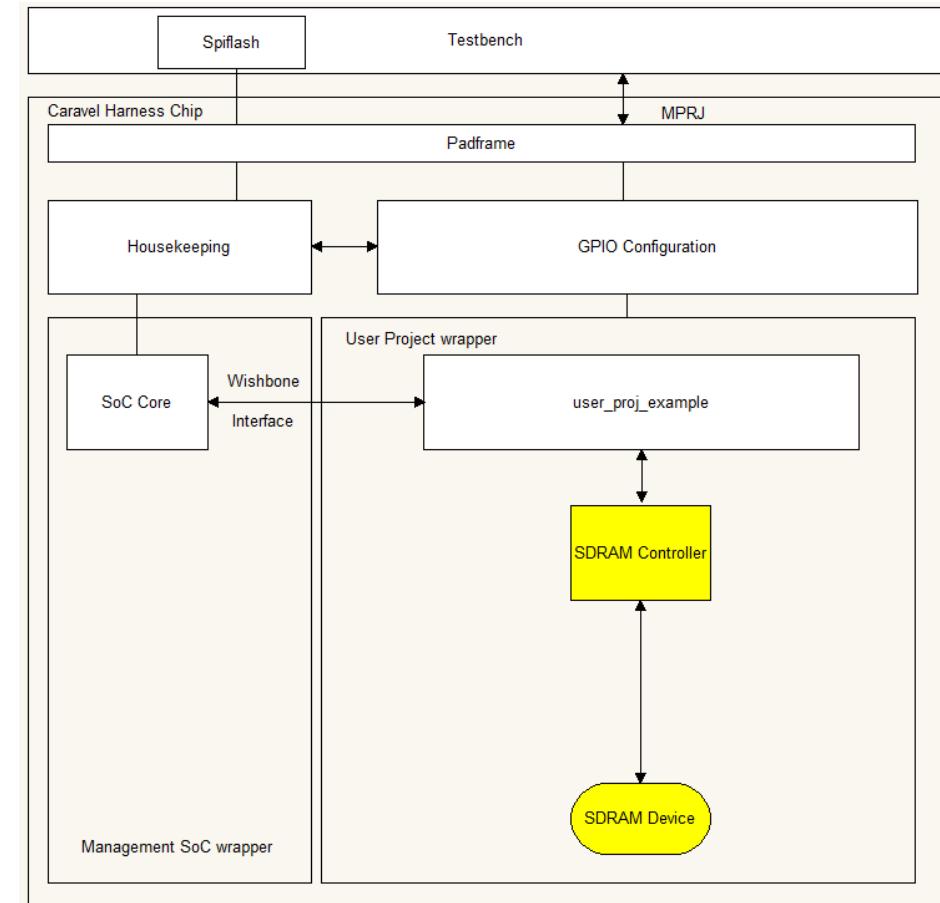
#clk	Baseline (lab6)	Compiler Opt.	SDRAM+prefetch	Hardware acc.
FIR	86583	84522 (x1.02)	38076 (x2.27)	123 (x704)
MM	39604	38080 (x1.04)	46166 (x0.86)	142 (x279)
Qsort	17553	14396 (x1.2)	12402 (x1.42)	139 (x126)
Total	143740	136998 (x1.05)	96644 (x1.49)	199 (x722)

- UART latency

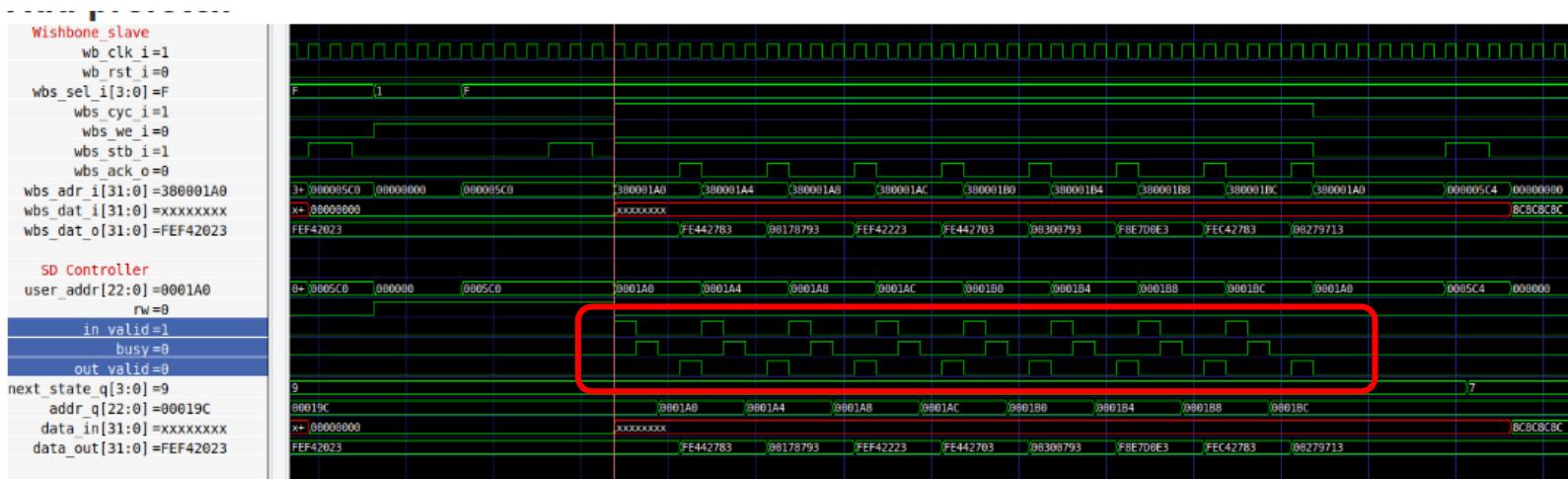
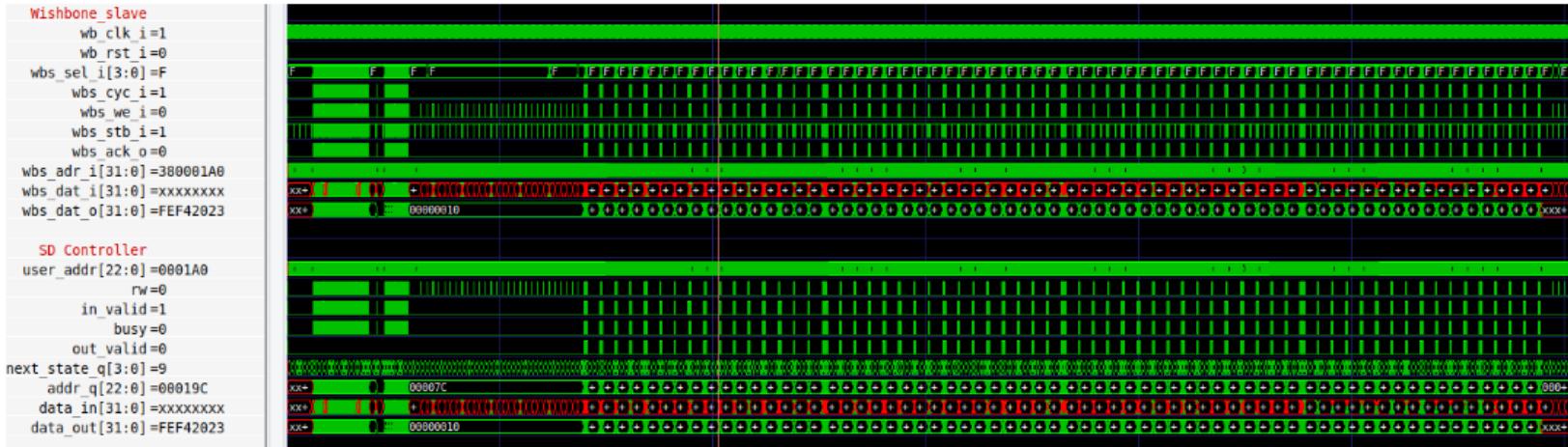
	Latency
UART	1.493 s
UART w/ FIFO	0.645 s (x2.31)

System Overview – SDRAM

- Propose adding prefetch
 - Cache size 64.
 - implemented by register and can store all data in one bank.
 - `switch_bank_q`
 - decide when should I switch bank



SDRAM with prefetch controller



UART - Optimize objective

- Continuous rx/tx
- Reduce CPU overhead (ISR execution time)
- Operation overlap
 - Uart_rx – receive data
 - Uart_tx – transmission data
 - CPU – ISR

UART - Optimize proposal

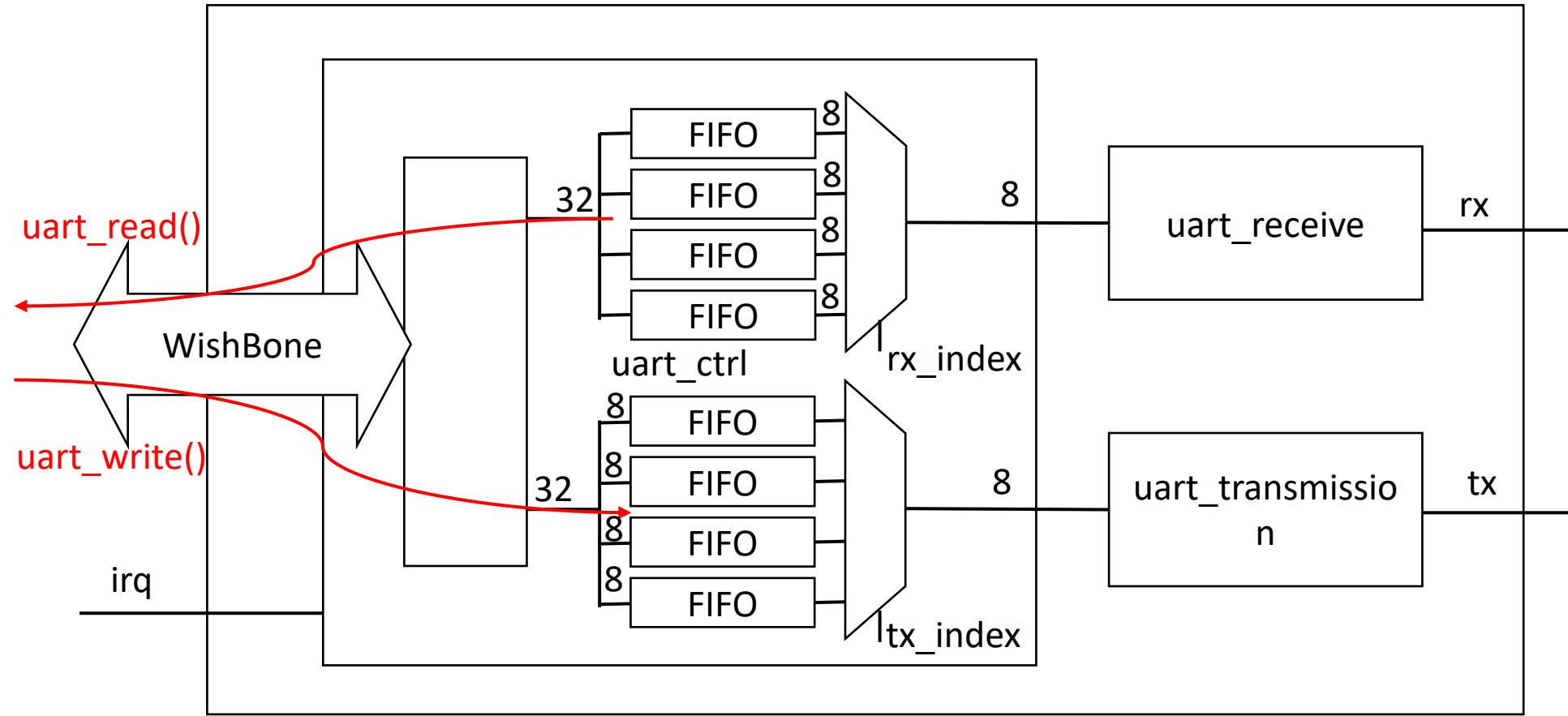
- FIFO between Wishbone and UART
- Concatenate 4 parallel 8bit-FIFO to match Wishbone width
- ISR flow pipeline
 - Every 4 UART rx transition (if (rx_fifo[n] not full) write rx_fifo[n]) -> IRQ
 - ISR -> if (!frame_err) read rx_fifo /
 wait (all tx_fifo not full) write tx_fifo
 - Any tx_fifo is not empty && !tx_busy -> start tx transition

UART – Configure register

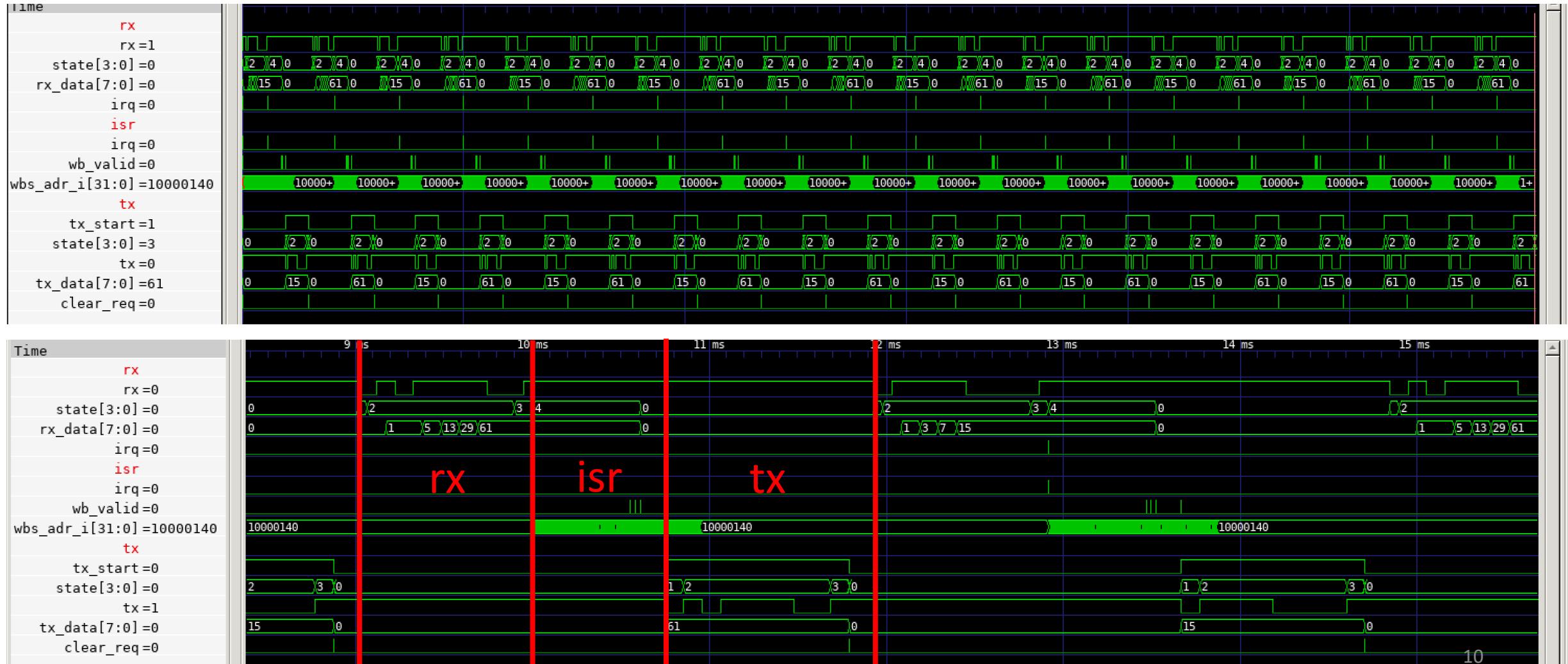
- RX_DATA: 0x3000_0000
- TX_DATA: 0x3000_0004
- STAT_REG: 0x3000_0008

RX_DATA	RX_FIFO_3 31-24		RX_FIFO_2 23-16		RX_FIFO_1 15-8		RX_FIFO_0 7-0	
TX_DATA	TX_FIFO_3 31-24		TX_FIFO_2 23-16		TX_FIFO_1 15-8		TX_FIFO_0 7-0	
STAT_REG	RESERVERD 31-7	Tx_fifo_full 6	Frame Err 5	Overrun Err 4	Tx_full 3	Tx_empty 2	Rx_full 1	Rx_empty 0

UART - Architecture

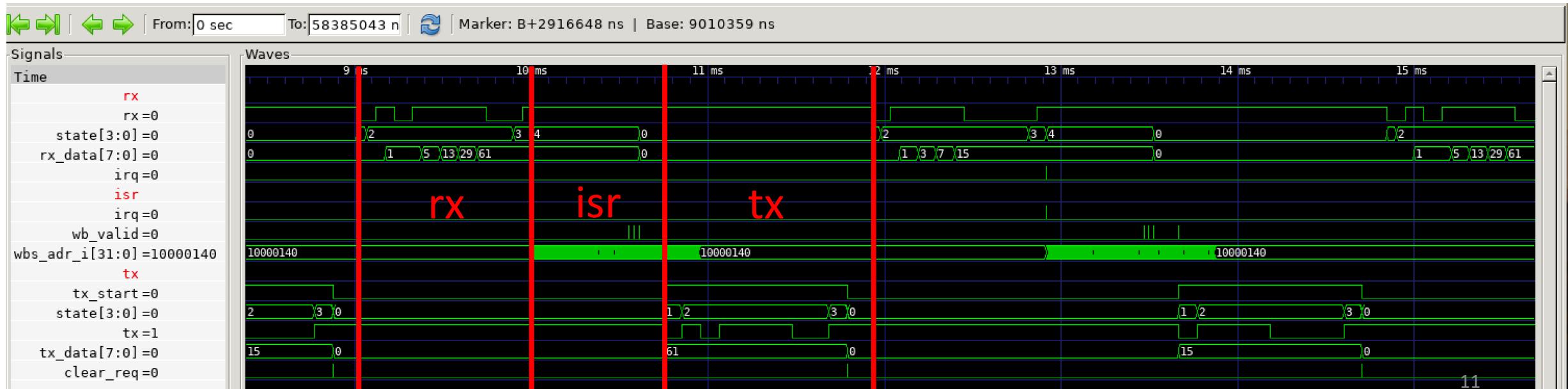


UART – Waveform (without FIFO)

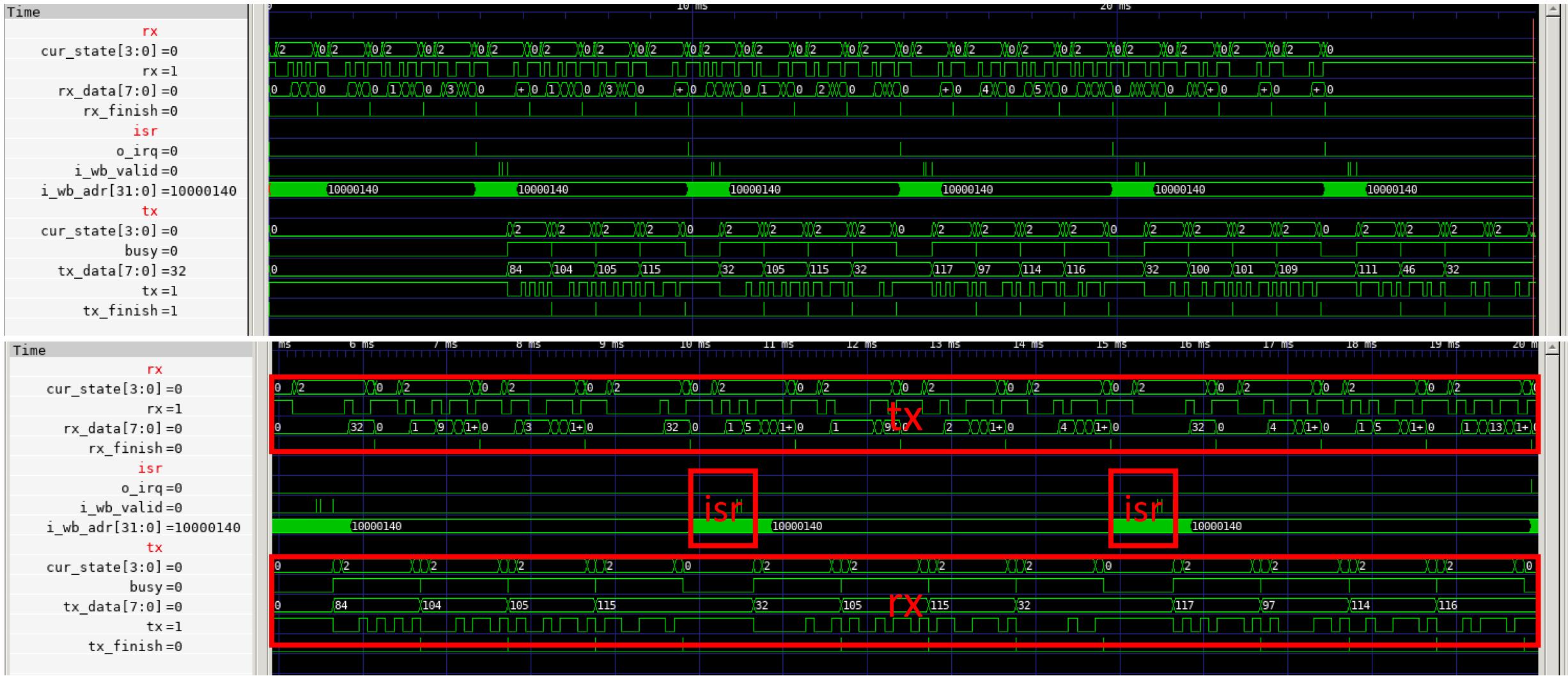


UART – Latency (without FIFO)

- (Rx + ISR + Tx) latency = 2.916648 ms
- 512 characters latency = $512 * 2.916648 \text{ ms} = 1.49 \text{ s}$

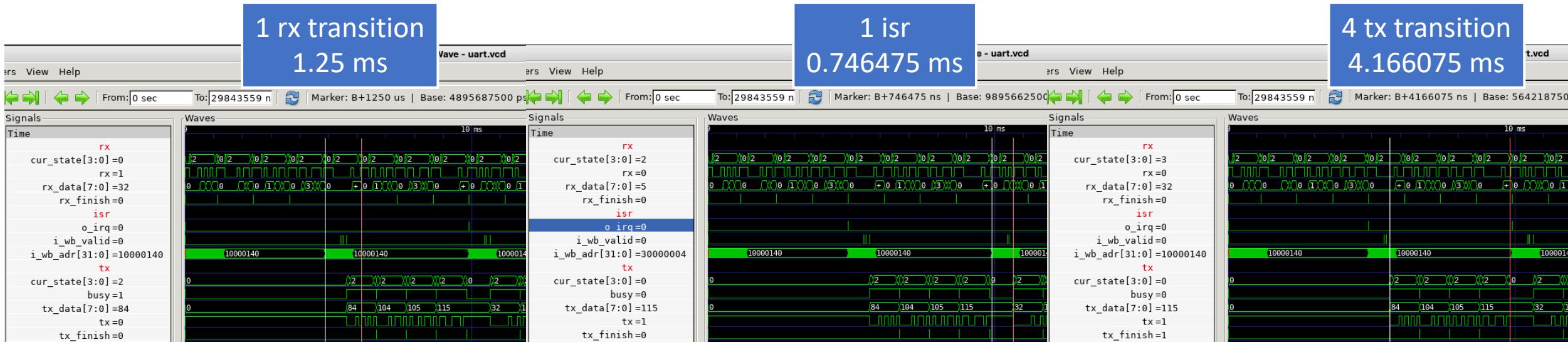


UART – Waveform (with FIFO)



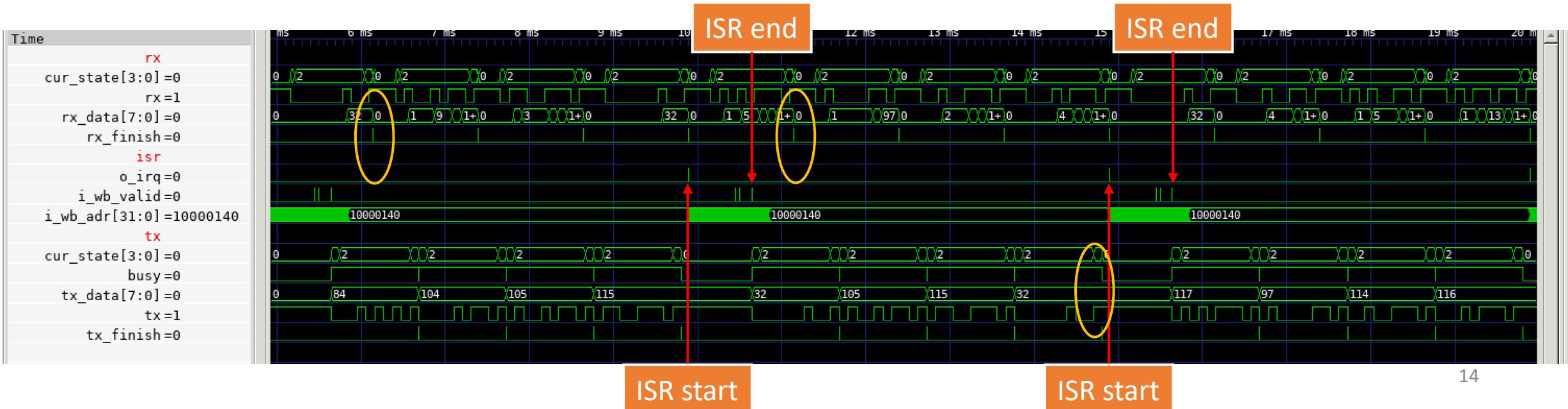
UART – Latency (with FIFO)

- Latency = N*(rx) + 1*(isr) + 4*(tx) = N*1.25 + 0.746475 + 4.166075
- 512 characters latency = 0.645 s
- Speed up = 1.49/0.645 = 2.31



UART – FIFO depth

- RX_FIFO: 1
- TX_FIFO: 1
- Depth=1 is enough in this case (no other IRQ, BAUD_RATE=9600)



UART – Simulation (1/2)

```
ubuntu@ubuntu2004: ~/SOC_Design/Final Project/testbench/uart (ssh) 361
rx data bit index 3: 0
tx complete: s (115)
rx data bit index 4: 1
rx data bit index 5: 1
rx data bit index 6: 1
tx data bit index 0: 0
rx data bit index 7: 0
received word u (117)
tx data bit index 1: 0
tx data bit index 2: 0
rx data bit index 0: 1
tx data bit index 3: 0
tx data bit index 4: 0
rx data bit index 1: 0
rx data bit index 2: 1
tx data bit index 5: 1
tx data bit index 6: 0
rx data bit index 3: 0
rx data bit index 4: 0
tx data bit index 7: 0
rx data bit index 5: 1
tx complete: ( 32)
rx data bit index 6: 1
rx data bit index 7: 0
received word e (101)
tx data bit index 0: 1
rx data bit index 0: 0
tx data bit index 1: 0
tx data bit index 2: 0
rx data bit index 1: 0
tx data bit index 3: 0
rx data bit index 2: 1
tx data bit index 4: 0
rx data bit index 3: 1
tx data bit index 5: 1
tx data bit index 6: 0
rx data bit index 7: 0
received word i (105)
tx data bit index 7: 0
tx complete: . ( 46)

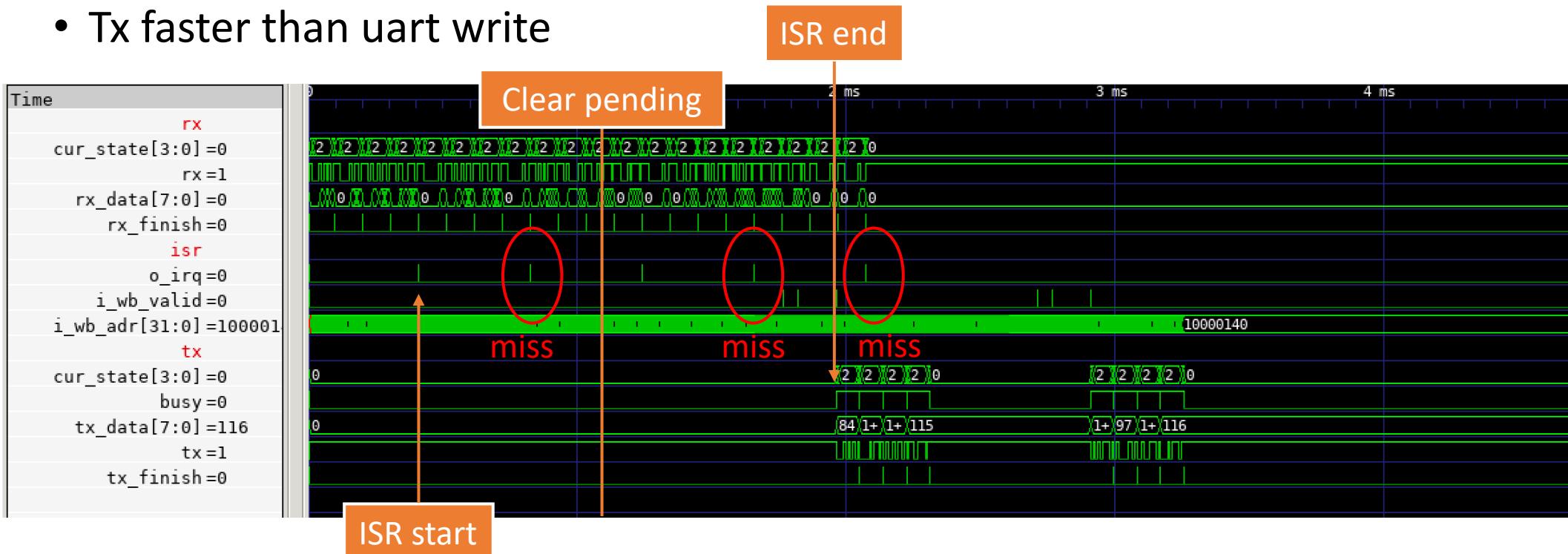
tx data bit index 2: 0
rx data bit index 1: 0
rx data bit index 2: 0
tx data bit index 3: 0
tx data bit index 4: 1
rx data bit index 3: 0
rx data bit index 4: 0
tx data bit index 5: 1
tx data bit index 6: 0
rx data bit index 5: 1
rx data bit index 6: 0
tx data bit index 7: 0
rx data bit index 7: 0
tx complete: 1 ( 49)
received word ( 32)
rx data bit index 0: 1
tx data bit index 0: 0
rx data bit index 1: 0
rx data bit index 2: 0
tx data bit index 1: 1
tx data bit index 2: 1
rx data bit index 3: 1
rx data bit index 4: 0
tx data bit index 5: 1
rx data bit index 6: 1
tx data bit index 5: 1
tx data bit index 6: 0
rx data bit index 7: 0
received word i (105)
tx data bit index 7: 0
tx complete: . ( 46)
```

UART – Simulation (2/2)

```
ubuntu@ubuntu2004: ~/SOC_Design/Final Project/testbench/uart (ssh)  *1      ubuntu@ub
recevied word i (105)
tx data bit index 7: 0
tx complete: . ( 46)
tx demo complete:
"This is uart demo.
This is uart demo.
This is uart demo.
This is uart demo.
The UART is a standard 2-pin serial interface that can communicate with the most similar interfaces
at a fixed baud rate. Although the UART operates independently of the CPU, data transfers are
blocking operations which will generate CPU wait states until the data transfer is completed.
The entire 32bit word encodes the number of CPU core cycles to divide down to get the UART
data bit rate (baud rate). The default value is 1."
rx data bit index 0: 1
rx data bit index 1: 1
rx data bit index 2: 0
rx data bit index 3: 0
rx data bit index 4: 1
rx data bit index 5: 1
rx data bit index 6: 1
rx data bit index 7: 0
recevied word s (115)
rx data bit index 0: 0
rx data bit index 1: 0
rx data bit index 2: 0
rx data bit index 3: 0
rx data bit index 4: 0
rx data bit index 5: 1
rx data bit index 6: 0
rx data bit index 7: 0
recevied word   ( 32)
rx data bit index 0: 1
rx data bit index 1: 0
rx data bit index 2: 0
rx data bit index 3: 0
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 1 ( 49)
rx data bit index 0: 0
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: 1
rx data bit index 6: 0
rx data bit index 7: 0
recevied word . ( 46)
rx demo complete:
"This is uart demo.
This is uart demo.
This is uart demo.
This is uart demo.
The UART is a standard 2-pin serial interface that can communicate with the most similar interfaces
at a fixed baud rate. Although the UART operates independently of the CPU, data transfers are
blocking operations which will generate CPU wait states until the data transfer is completed.
The entire 32bit word encodes the number of CPU core cycles to divide down to get the UART
data bit rate (baud rate). The default value is 1."
644839623
ubuntu@ubuntu2004:~/SOC_Design/Final Project/testbench/uart$
```

UART – Discussion (1/2)

- Increase BAUD_RATE to 115200
 - Must increase RX_FIFO depth
 - IRQ overlap problem
 - Tx faster than uart write



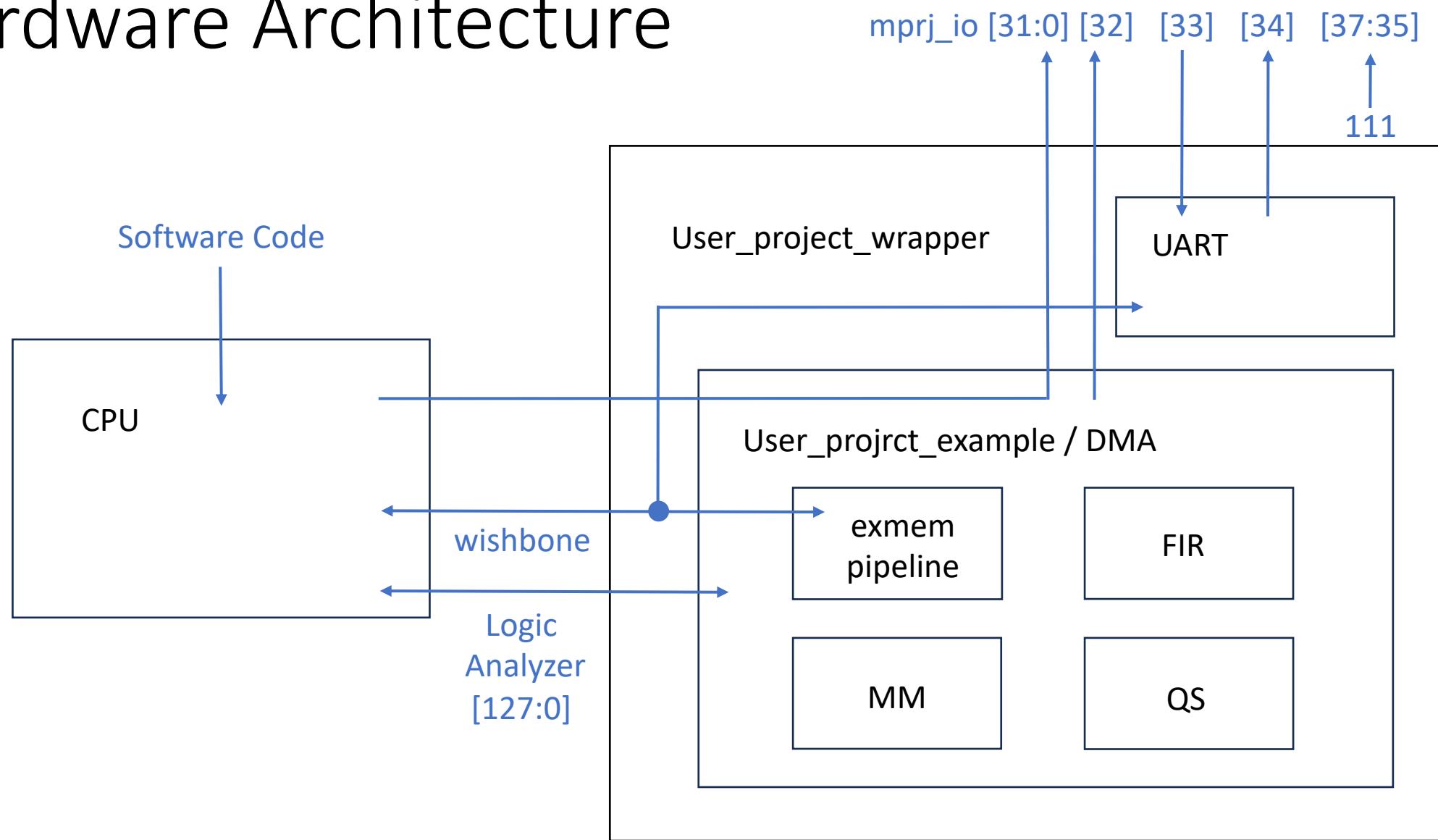
UART – Discussion (2/2)

- Solution
 - New IRQ mechanism: no IRQ if the previous IRQ is pending
 - New ISR procedure: transfer data until RX_FIFO is empty
- Future work
 - Fix the previous problem
 - FPGA verification

DMA and Hardware Accelerators

Y.K. Tsai, IEO, NYCU

Hardware Architecture



Memory Mapping

Define sections for variables

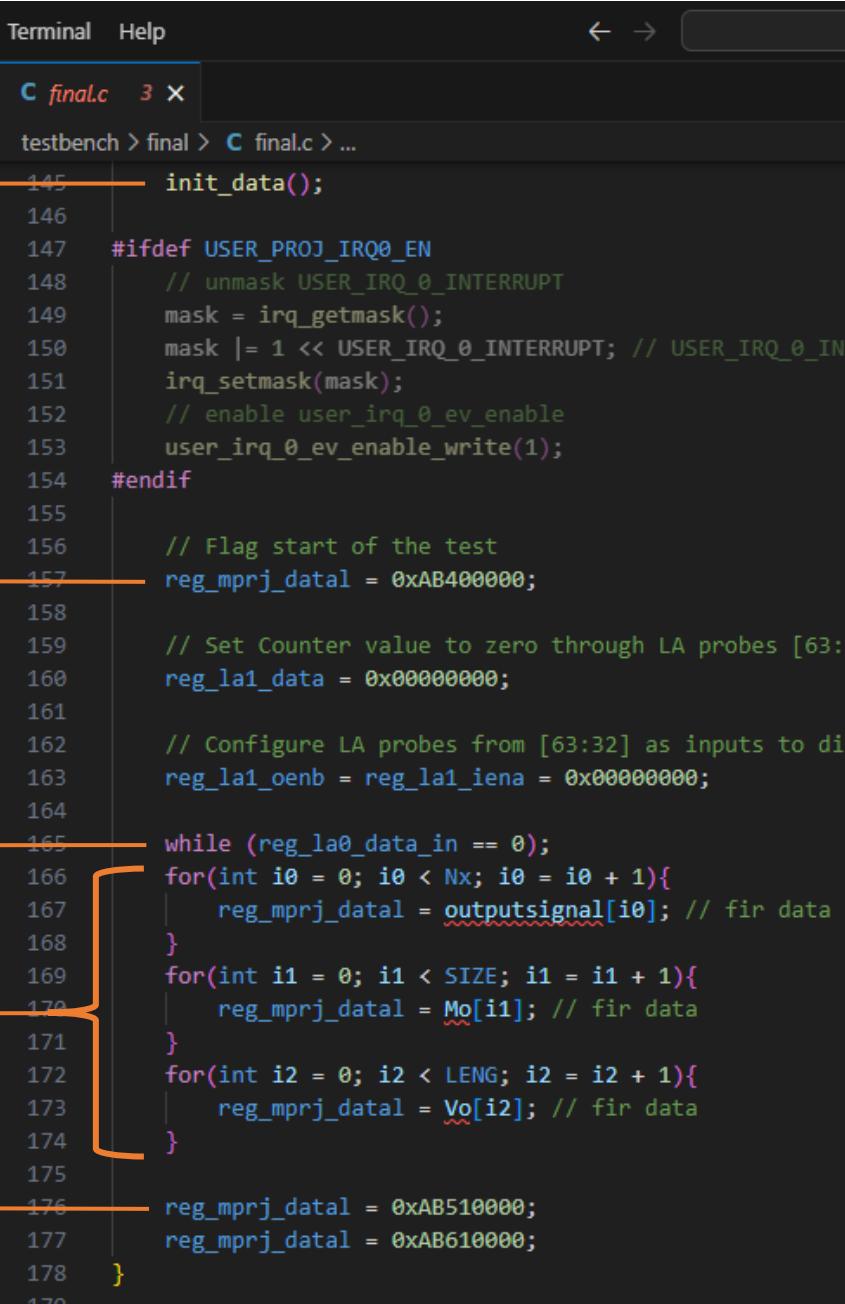
```
Terminal Help ← →
C final.h 9+ X
testbench > final > C final.h > ...
27 #define Nt 11
28 #define Nx 16
29 int __attribute__ (( section ( ".fir_tap" ) )) taps[Nt];
30 int __attribute__ (( section ( ".fir_x" ) )) inputsignal[Nx];
31 int __attribute__ (( section ( ".fir_y" ) )) outputsignal[Nx];
32
33 #define SIZE 16 // should be square number
34 int __attribute__ (( section ( ".mm_a" ) )) Am[SIZE];
35 int __attribute__ (( section ( ".mm_b" ) )) Bm[SIZE];
36 int __attribute__ (( section ( ".mm_o" ) )) Mo[SIZE];
37
38 #define LENG 16
39 int __attribute__ (( section ( ".qs_a" ) )) Aq[LENG];
40 int __attribute__ (( section ( ".qs_o" ) )) Vo[LENG];
41
42 void init_data(){
43     taps[0] = 0;
44     taps[1] = -10;
45     taps[2] = -9;
46     taps[3] = 23;
```

Define addresses for sections

```
Terminal Help ← →
C sections.lds 9+ X
firmware > C sections.lds
11 MEMORY {
12     vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
13     dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
14     dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
15     flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
16     mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000
17     /* mprjram : ORIGIN = 0x38000000, LENGTH = 0x00400000 */
18     fir_tap : ORIGIN = 0x38000000, LENGTH = 0x00000040
19     fir_x : ORIGIN = 0x38000040, LENGTH = 0x00000040
20     fir_y : ORIGIN = 0x38000080, LENGTH = 0x00000040
21     mm_a : ORIGIN = 0x38000c0, LENGTH = 0x00000040
22     mm_b : ORIGIN = 0x38000100, LENGTH = 0x00000040
23     mm_o : ORIGIN = 0x38000140, LENGTH = 0x00000040
24     qs_a : ORIGIN = 0x38000180, LENGTH = 0x00000040
25     qs_o : ORIGIN = 0x380001c0, LENGTH = 0x00000040
26     mprjram : ORIGIN = 0x38000200, LENGTH = 0x00000200
27     hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
28     csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
29 }
```

Software usage

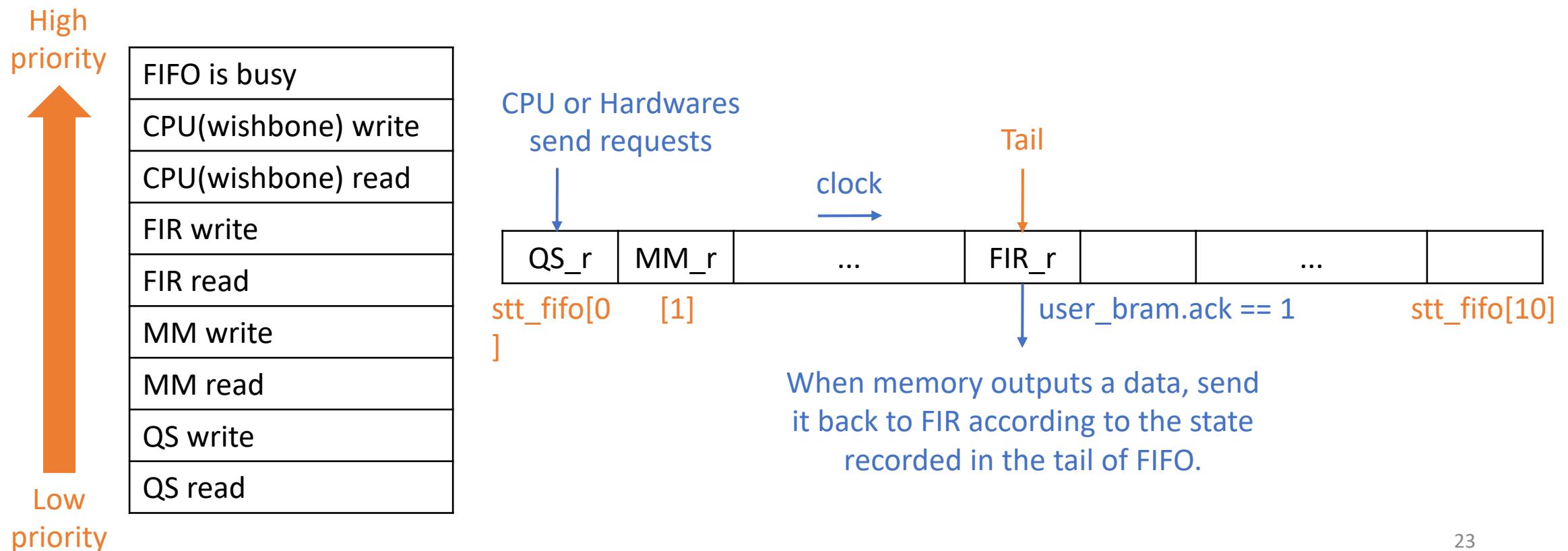
- Initialize & Pre-load data
- Workload start
- Wait workload to be finished
- Read results from exmem_pipeline to verify
- Software finished



```
Terminal Help
C final.c 3 ×
testbench > final > C final.c > ...
145     init_data();
146
147 #ifdef USER_PROJ_IRQ0_EN
148     // unmask USER_IRQ_0_INTERRUPT
149     mask = irq_getmask();
150     mask |= 1 << USER_IRQ_0_INTERRUPT; // USER_IRQ_0_INTERRUPT
151     irq_setmask(mask);
152     // enable user_irq_0_ev_enable
153     user_irq_0_ev_enable_write(1);
154 #endif
155
156     // Flag start of the test
157     reg_mprj_data1 = 0xAB400000;
158
159     // Set Counter value to zero through LA probes [63:32]
160     reg_la1_data = 0x00000000;
161
162     // Configure LA probes from [63:32] as inputs to display
163     reg_la1_oenb = reg_la1_iena = 0x00000000;
164
165     while (reg_la0_data_in == 0){
166         for(int i0 = 0; i0 < Nx; i0 = i0 + 1){
167             reg_mprj_data1 = outputsignal[i0]; // fir data
168         }
169         for(int i1 = 0; i1 < SIZE; i1 = i1 + 1){
170             reg_mprj_data1 = Mq[i1]; // fir data
171         }
172         for(int i2 = 0; i2 < LENG; i2 = i2 + 1){
173             reg_mprj_data1 = Vq[i2]; // fir data
174         }
175     }
176     reg_mprj_data1 = 0xAB510000;
177     reg_mprj_data1 = 0xAB610000;
178 }
```

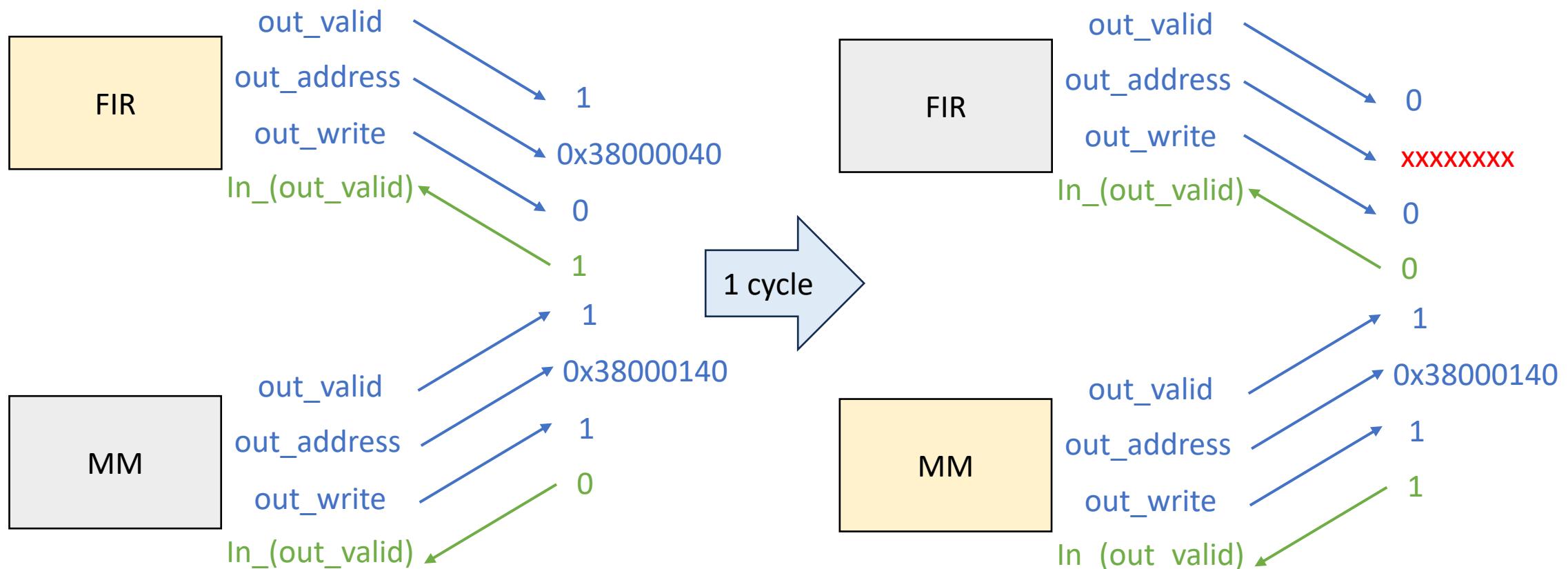
DMA priority & state FIFO

- Use state FIFO to be compatible with other memory (needn't to be 10T)



Simultaneous R/W Requests

- FIR_read + MM_write



Quick Sort

PSEUDOCODE

```
QUICKSORT( $A, p, r$ )
1  if  $p < r$ 
2      // Partition the subarray around the pivot, which ends up in  $A[q]$ .
3       $q = \text{PARTITION}(A, p, r)$ 
4      QUICKSORT( $A, p, q - 1$ ) // recursively sort the low side
5      QUICKSORT( $A, q + 1, r$ ) // recursively sort the high side
```

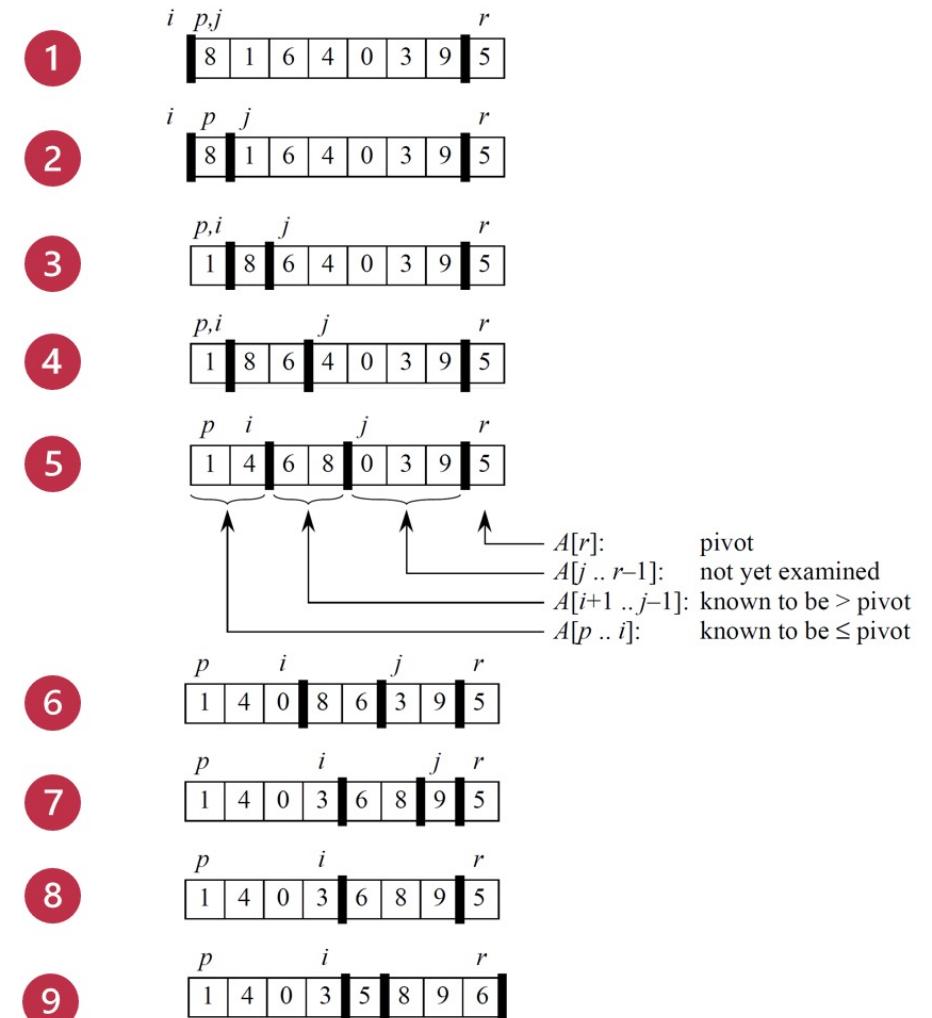
Initial call is $\text{QUICKSORT}(A, 1, n)$.

PARTITIONING

Partition subarray $A[p:r]$ by the following procedure:

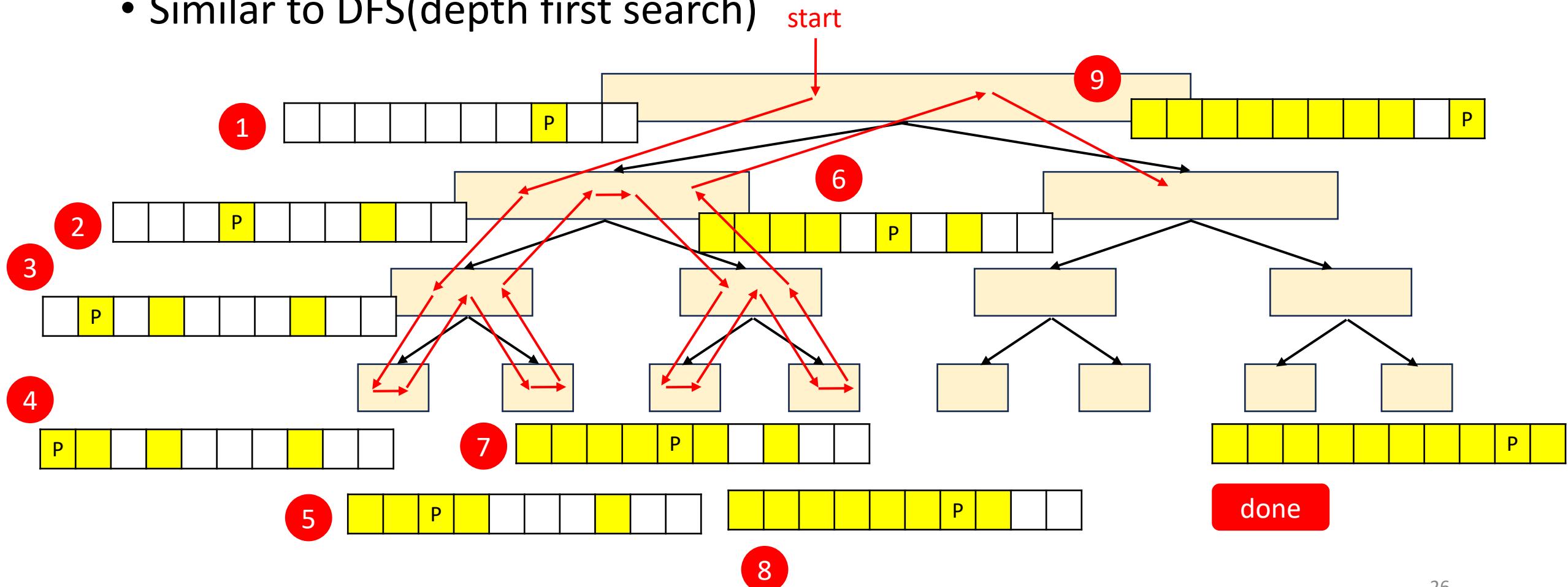
```
PARTITION( $A, p, r$ )
1   $x = A[r]$                                 // the pivot
2   $i = p - 1$                                // highest index into the low side
3  for  $j = p$  to  $r - 1$                 // process each element other than the pivot
4      if  $A[j] \leq x$                       // does this element belong on the low side?
5           $i = i + 1$                          // index of a new slot in the low side
6          exchange  $A[i]$  with  $A[j]$  // put this element there
7  exchange  $A[i + 1]$  with  $A[r]$  // pivot goes just to the right of the low side
8  return  $i + 1$                           // new index of the pivot
```

PARTITION always selects the last element $A[r]$ in the subarray $A[p:r]$ as the **pivot**—the element around which to partition.



Quick Sort with Verilog

- Similar to DFS(depth first search)



RTL simulation

- Tap[11]:

0	-10	-9	23	...	0
---	-----	----	----	-----	---
- X[16]:

1	2	3	4	...	16
---	---	---	---	-----	----
- A[16]:

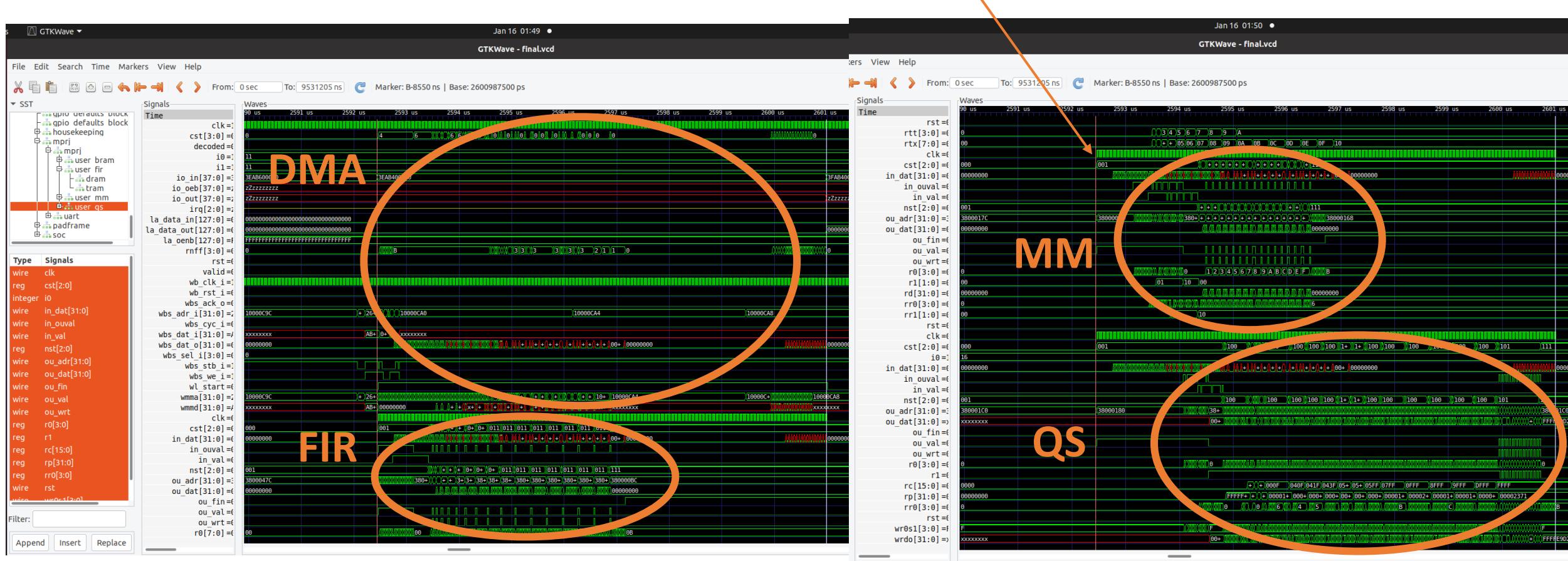
0	1	2	3	0	...
---	---	---	---	---	-----
- B[16]:

1	2	3	4	...	16
---	---	---	---	-----	----
- V[16]:

893	40	...	0	-1
-----	----	-----	---	----
- Latency: 342 cycles; 8550 ns

```
ubuntu@ubuntu2004:~/soc/lab_final/testbench/final$ s
ubuntu@ubuntu2004:~/soc/lab_final/testbench/final$ s
.../rtl/user/qsv:90: warning: extra digits given
.../rtl/user/qsv:90: warning: Numeric constant tr
Reading final.hex
final.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile final.vcd opened for output.
UART Test 1 started
LA Test 1 started
FIR      0 pass:      0
FIR      1 pass:    -10
FIR      2 pass:    -29
FIR      3 pass:    -25
FIR      4 pass:     35
FIR      5 pass:   158
FIR      6 pass:   337
FIR      7 pass:   539
FIR      8 pass:   732
FIR      9 pass:  915
FIR     10 pass: 1098
FIR     11 pass: 1281
FIR     12 pass: 1464
FIR     13 pass: 1647
FIR     14 pass: 1830
FIR     15 pass: 2013
MM      0 pass:    62
MM      1 pass:    68
MM      2 pass:    74
MM      3 pass:    80
MM      4 pass:    62
MM      5 pass:    68
MM      6 pass:    74
MM      7 pass:    80
MM      8 pass:    62
MM      9 pass:    68
MM     10 pass:    74
MM     11 pass:    80
MM     12 pass:    62
MM     13 pass:    68
MM     14 pass:    74
MM     15 pass:    80
QS      0 pass: -5678
QS      1 pass: -1234
QS      2 pass: -1234
QS      3 pass:    -1
QS      4 pass:     0
QS      5 pass:     1
QS      6 pass:    40
LA Test 2 passed
work load spent cycles:          342, time:    8550 (ns)
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
UART Test 2 passed
rx data bit index 0: 1
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
received word 61
ubuntu@ubuntu2004:~/soc/lab_final/testbench/final$ 
ubuntu@ubuntu2004:~/soc/lab_final/testbench/final$
```

Waveform



Future Work

- Replace exmem_pipeline with other memories (i.e. 1T, 8T pipeline, 12T pipeline, (1 or 3)T) to verify the compatibility of DMA.
- Synthesis to check the LUT/FF usage

Reference

- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2022) [1990]. Introduction to Algorithms (4th ed.). MIT Press and McGraw-Hill. ISBN 0-262-04630-X.