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

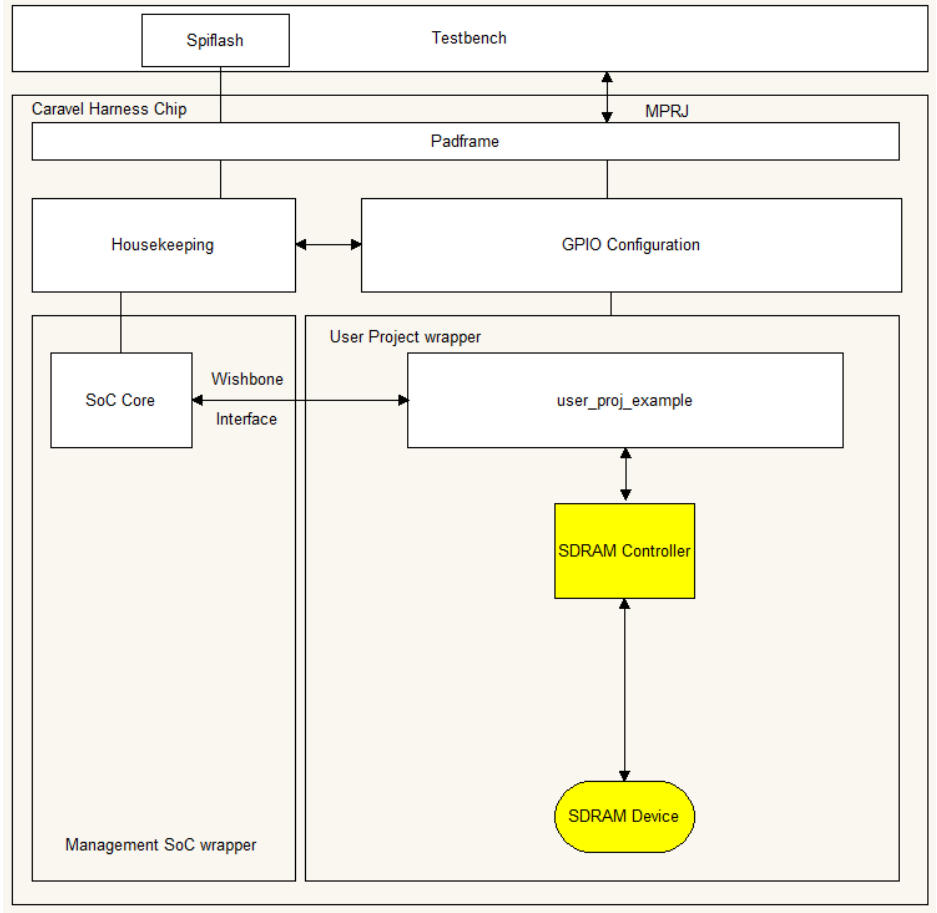
#clk	Baseline (lab6)	Compiler Opt.	SDRAM+prefetch	Hardware acc.
FIR	86583	84522 (x1.02)	38076 (x2.27)	123 (x704)
MM	39604	38080 (x1.04)	46166 (x0.86)	142 (x279)
Qsort	17553	14396 (x1.2)	12402 (x1.42)	139 (x126)
Total	143740	136998 (x1.05)	96644 (x1.49)	199 (x722)

- UART latency

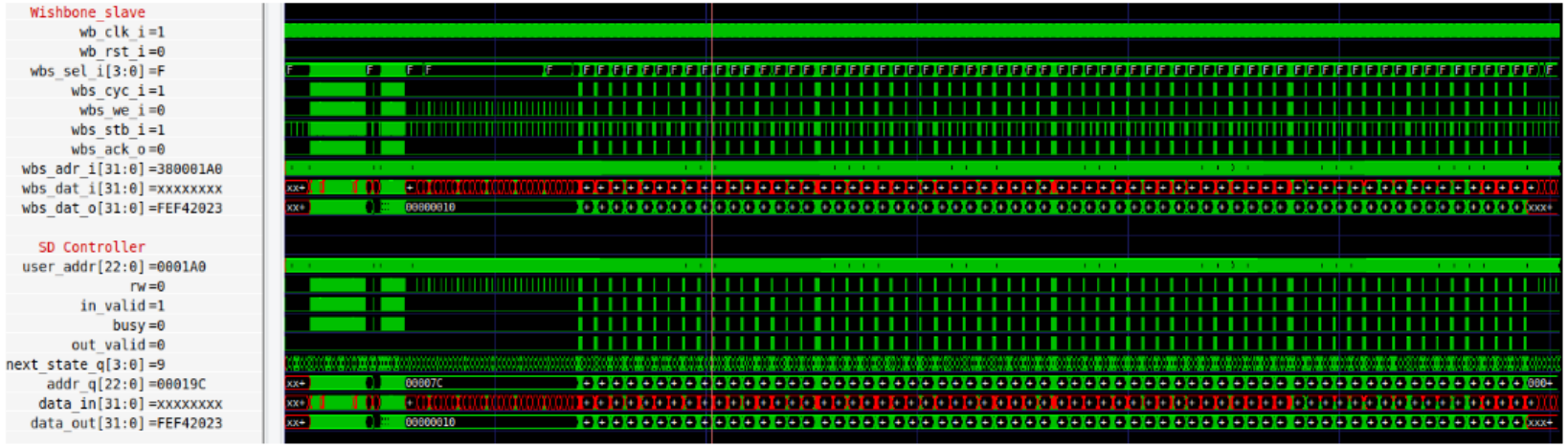
	Latency
UART	1.493 s
UART w/ FIFO	0.645 s (x2.31)

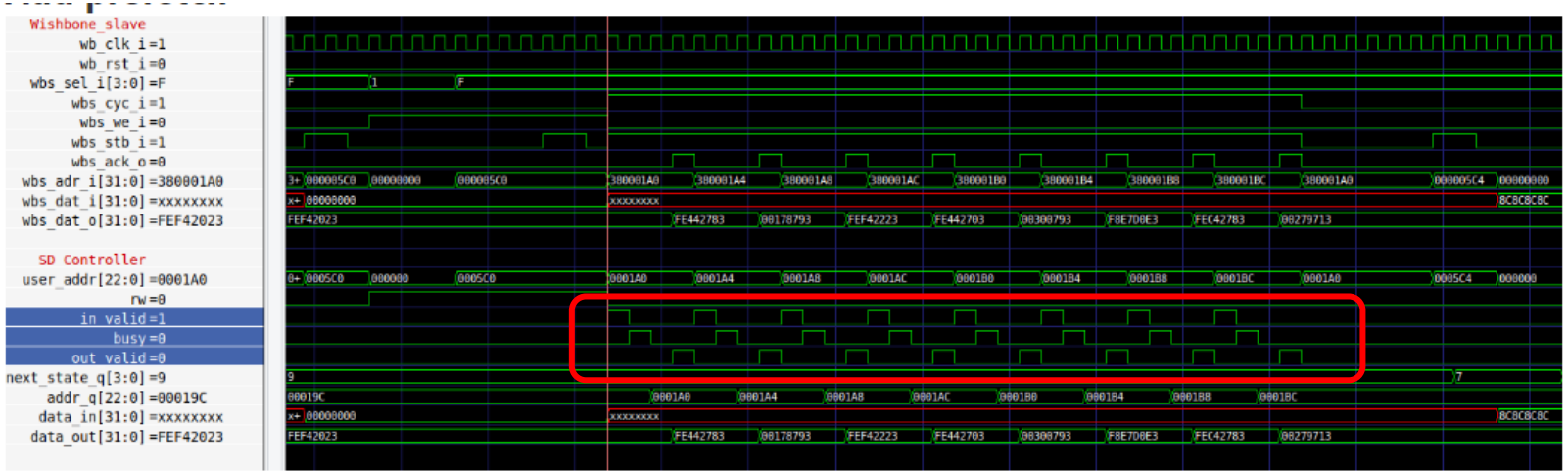
System Overview – SDRAM

- 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

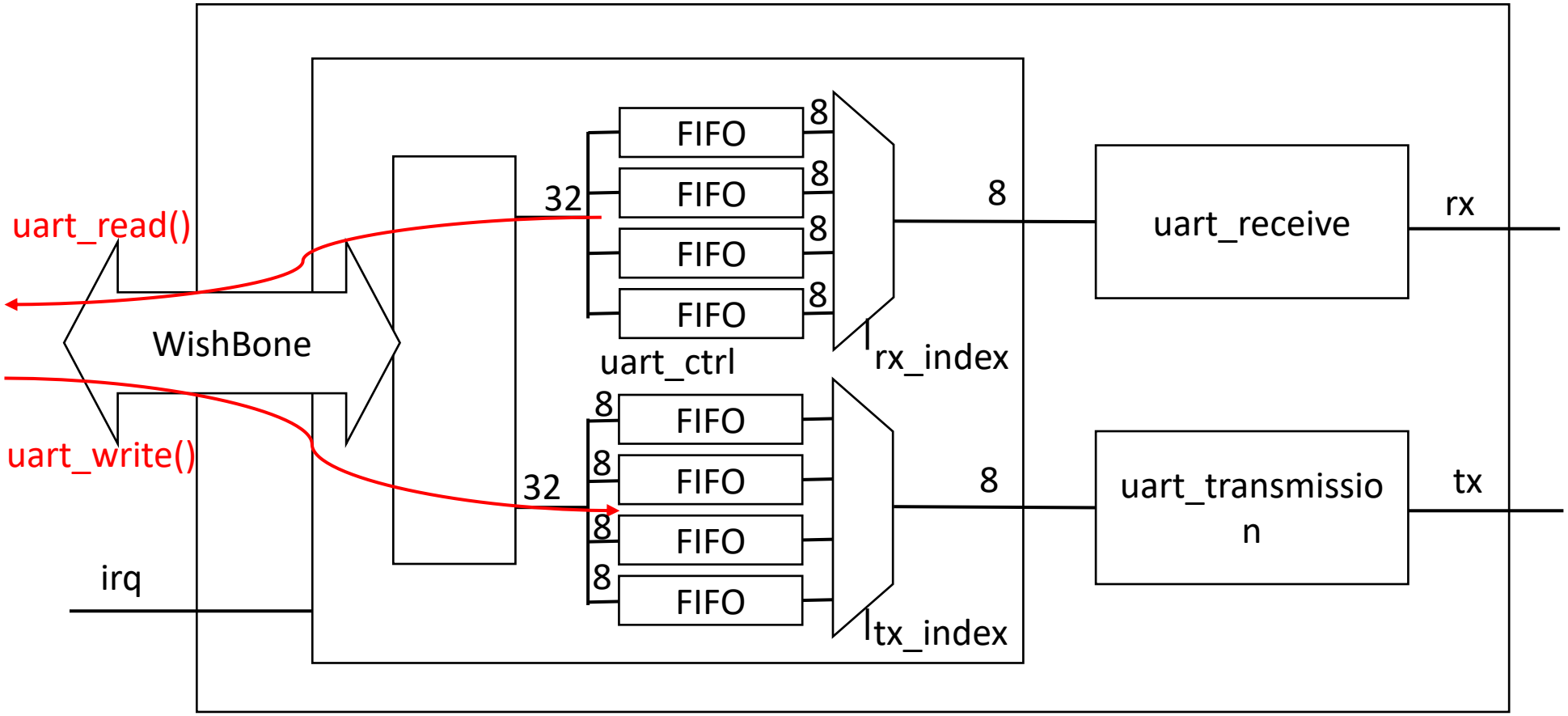
- FIFO between Wishbone and UART
- Concatenate 4 parallel 8bit-FIFO to match Wishbone width
- ISR flow pipeline
  - Every 4 UART rx transition (if (rx_fifo[n] not full) write rx_fifo[n]) -> IRQ
  - ISR -> if (!frame_err) read rx_fifo /
    wait (all tx_fifo not full) write tx_fifo
  - Any tx_fifo is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

UART – Configure register

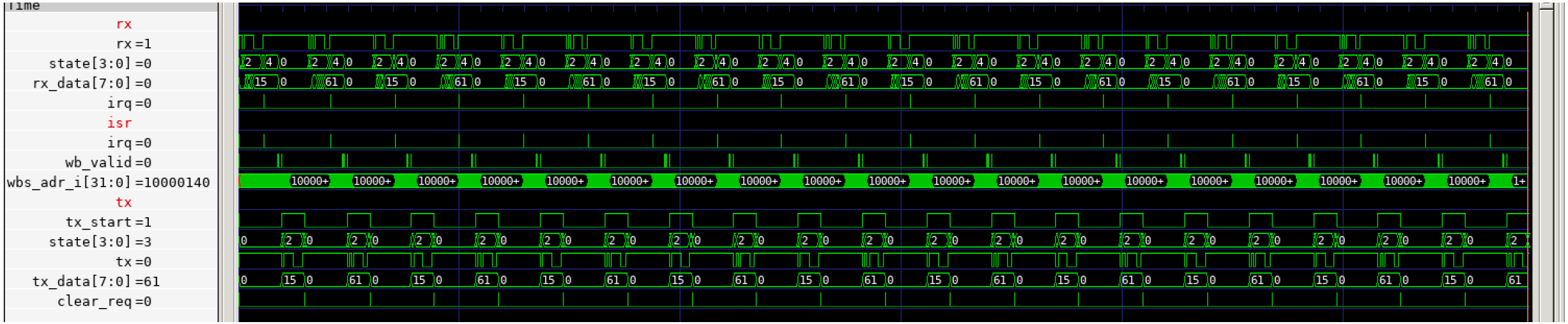
- RX_DATA: 0x3000_0000
- TX_DATA: 0x3000_0004
- STAT_REG: 0x3000_0008

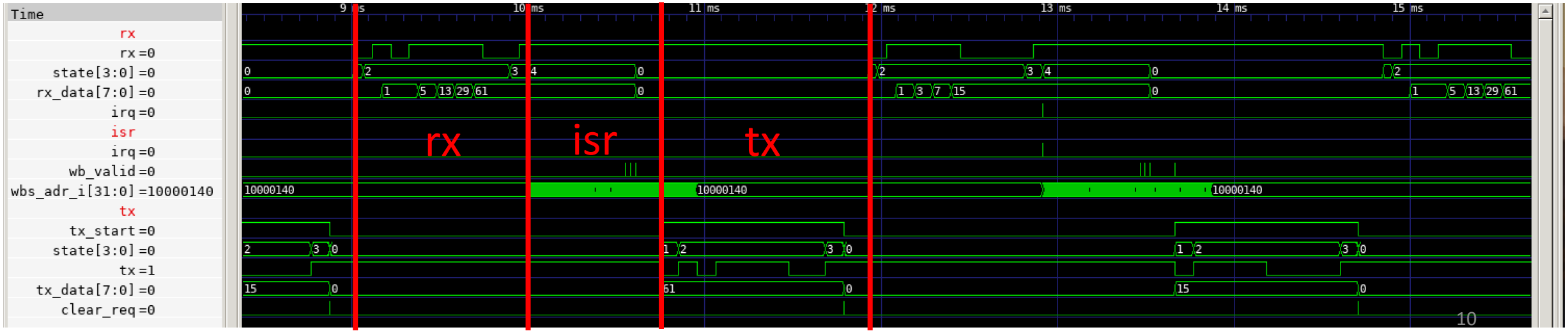
RX_DATA	RX_FIFO_3 31-24		RX_FIFO_2 23-16		RX_FIFO_1 15-8		RX_FIFO_0 7-0	
TX_DATA	TX_FIFO_3 31-24		TX_FIFO_2 23-16		TX_FIFO_1 15-8		TX_FIFO_0 7-0	
STAT_REG	RESERVERD 31-7	Tx_fifo_full 6	Frame Err 5	Overrun Err 4	Tx_full 3	Tx_empty 2	Rx_full 1	Rx_empty 0

UART - Architecture



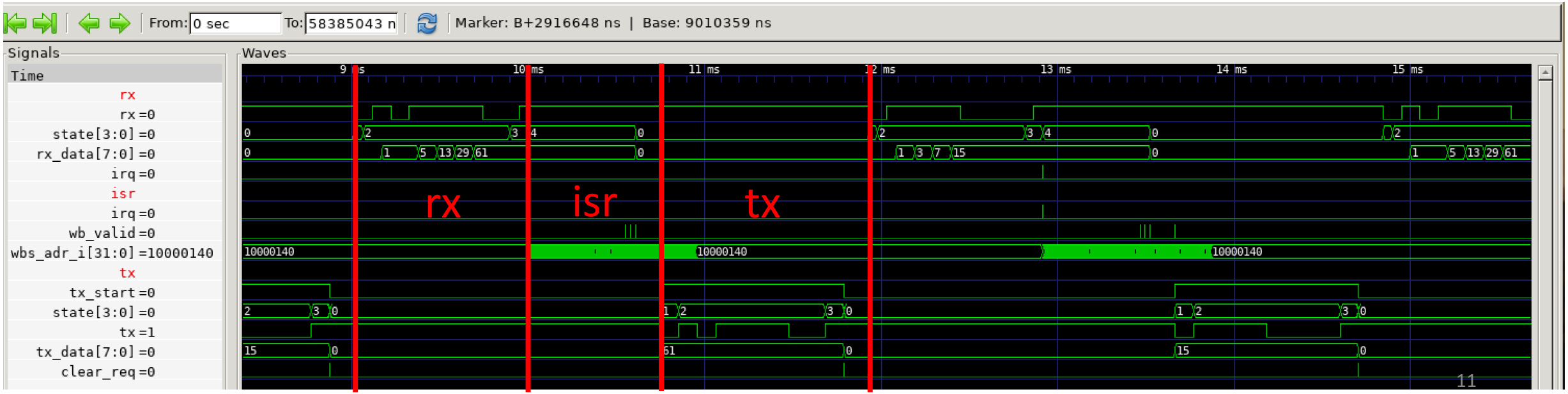
UART – Waveform (without FIFO)



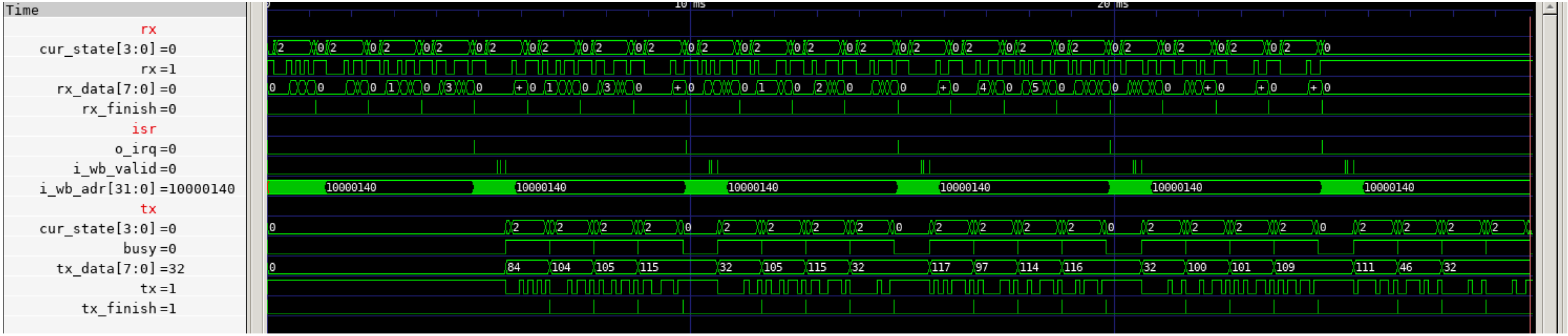


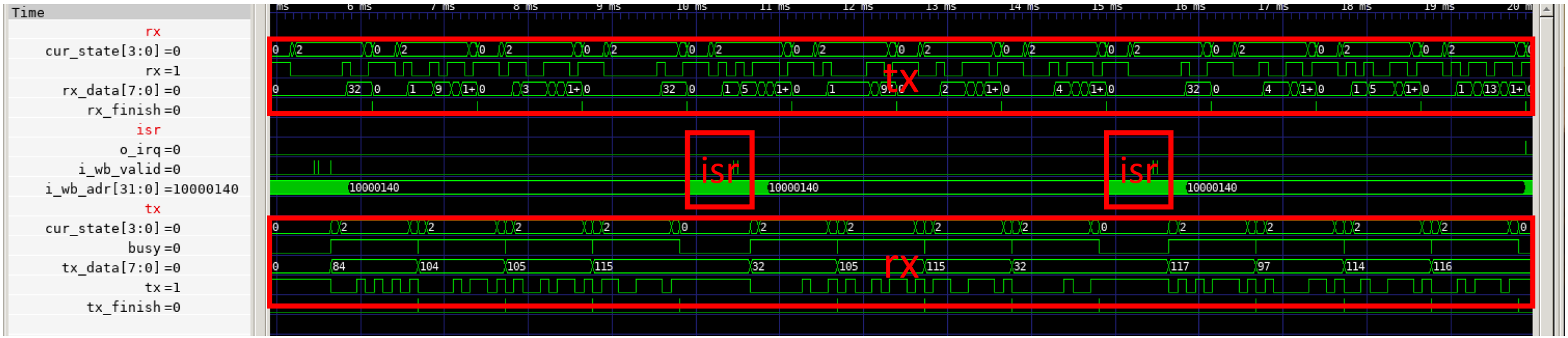
UART – Latency (without FIFO)

- (Rx + ISR + Tx) latency = 2.916648 ms
- 512 characters latency = 512*2.916648 ms = 1.49 s



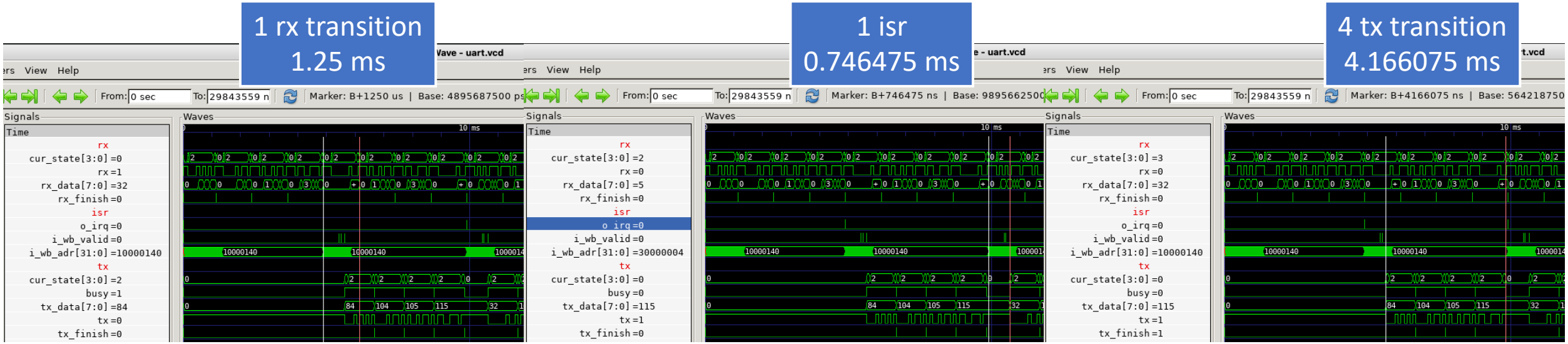
UART – Waveform (with FIFO)





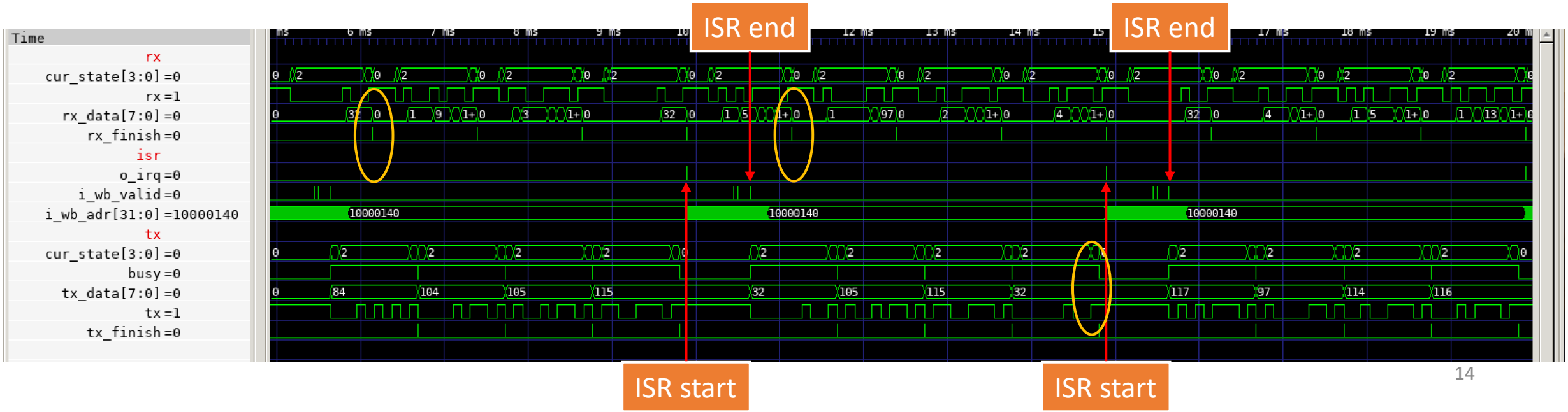
UART – Latency (with FIFO)

- 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

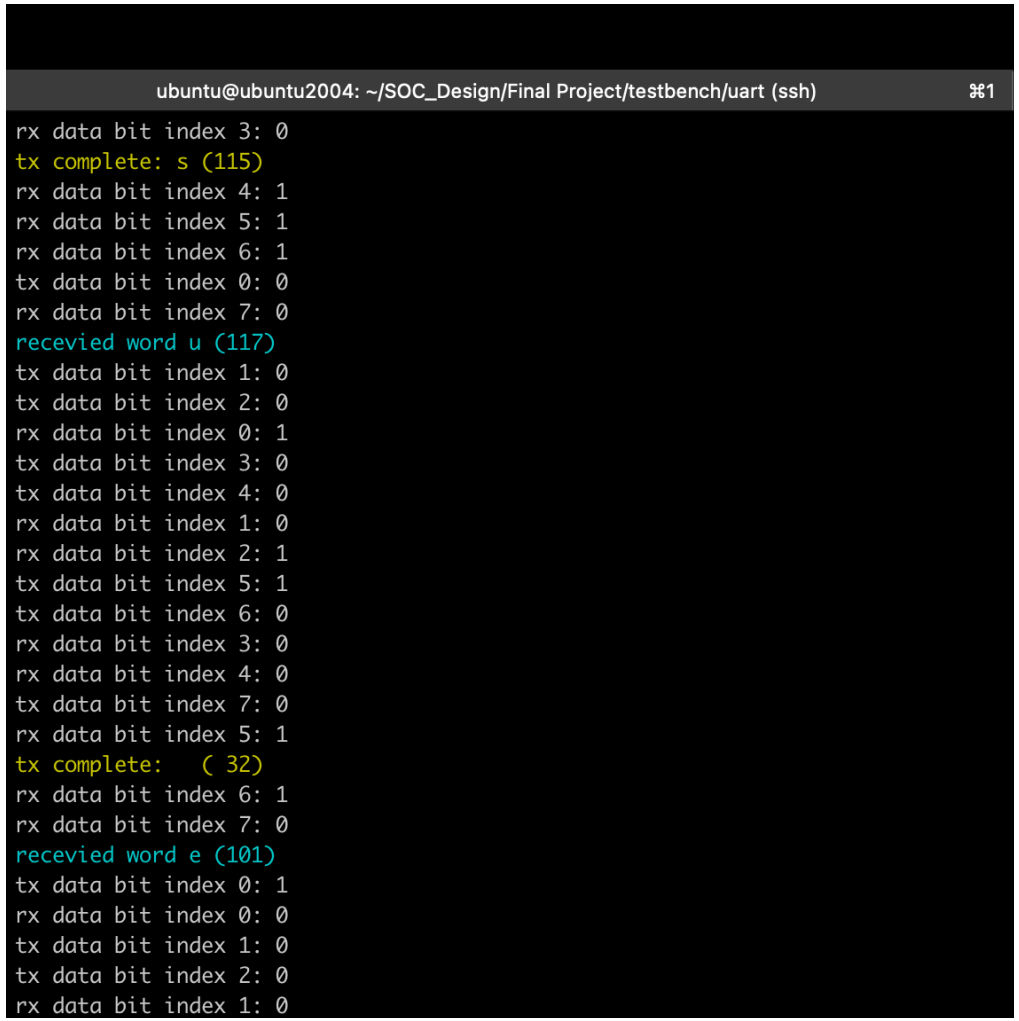


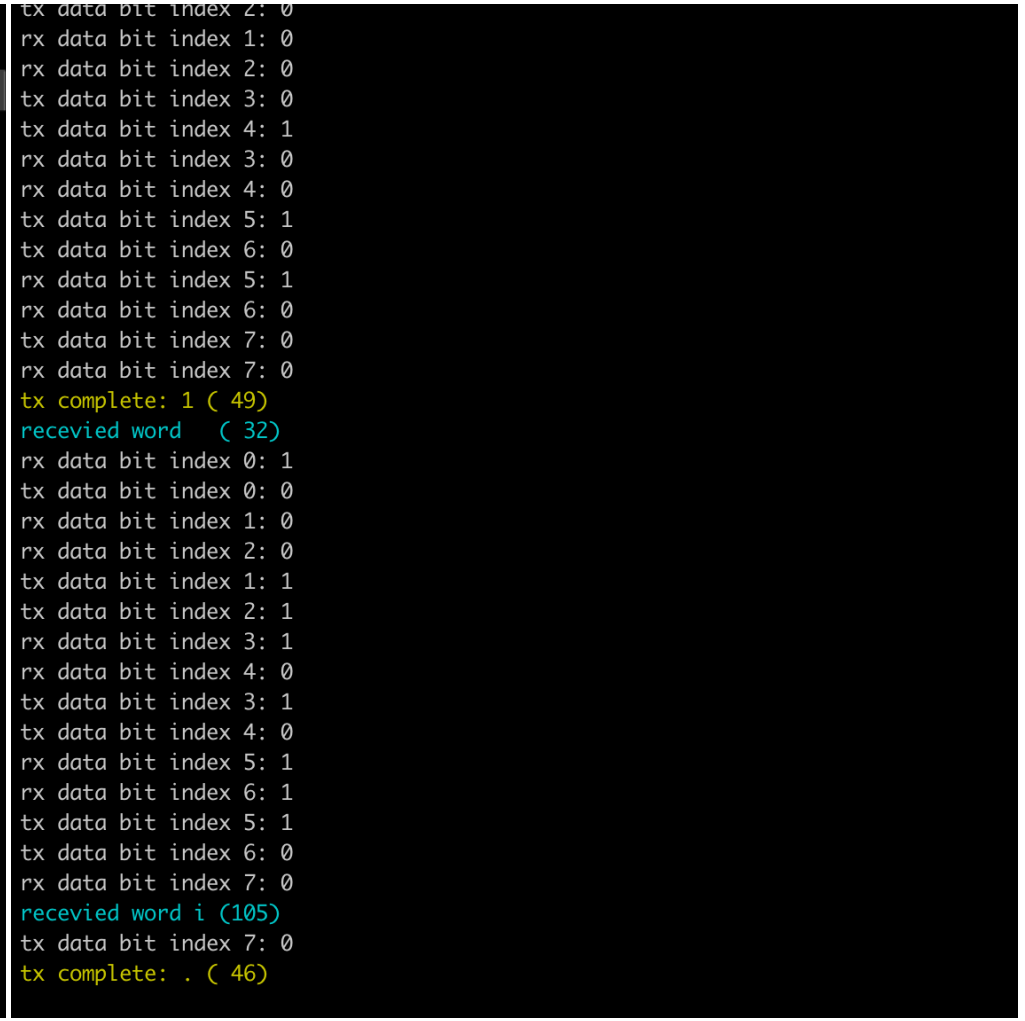
UART – FIFO depth

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

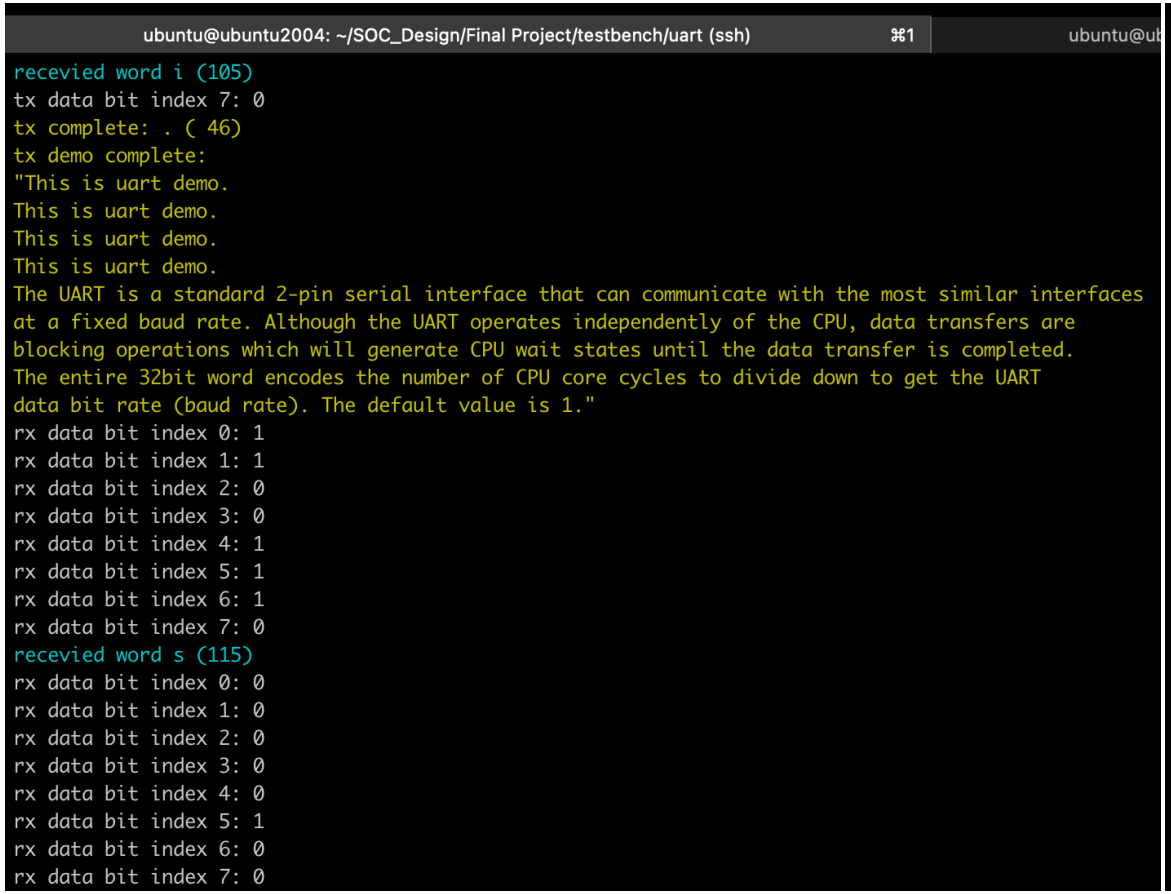


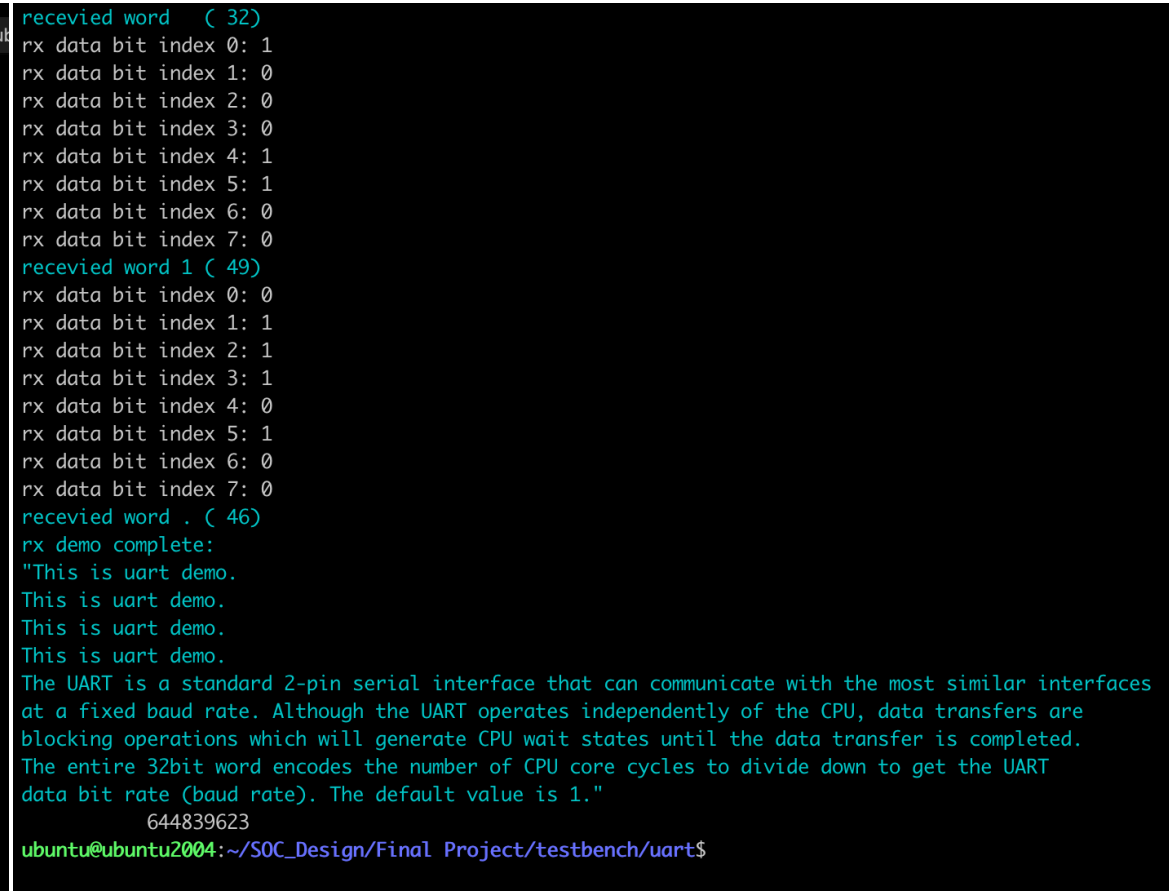
UART – Simulation (1/2)





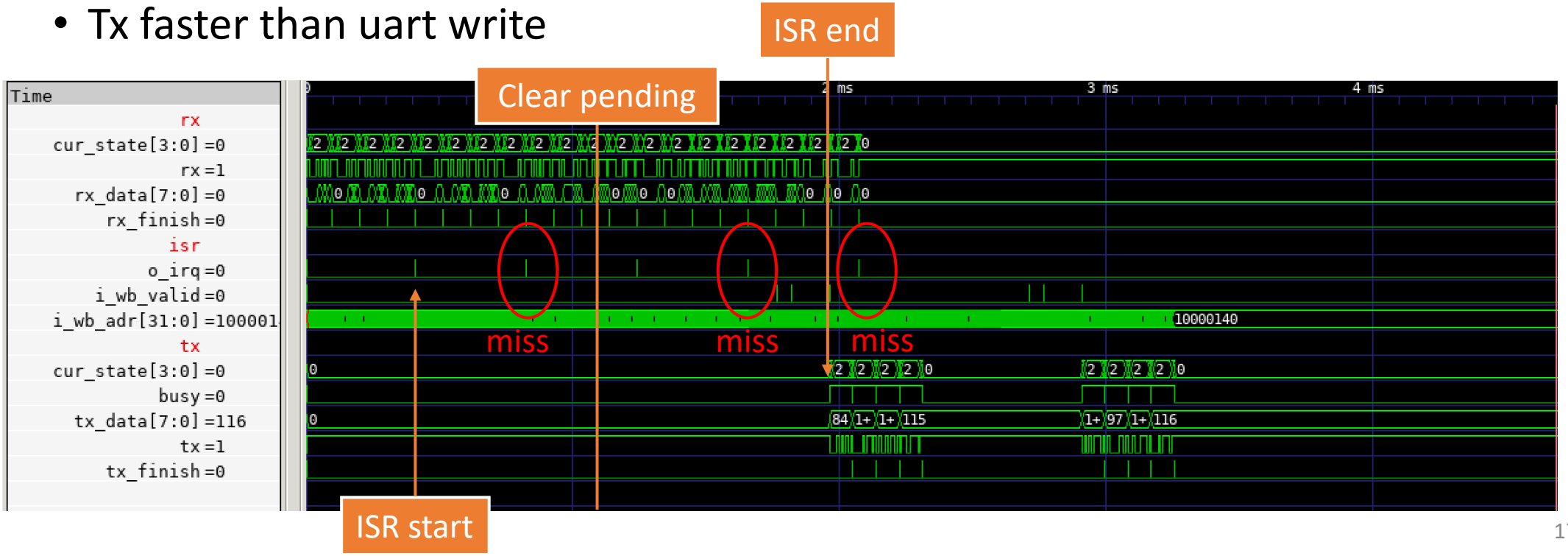
UART – Simulation (2/2)





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)

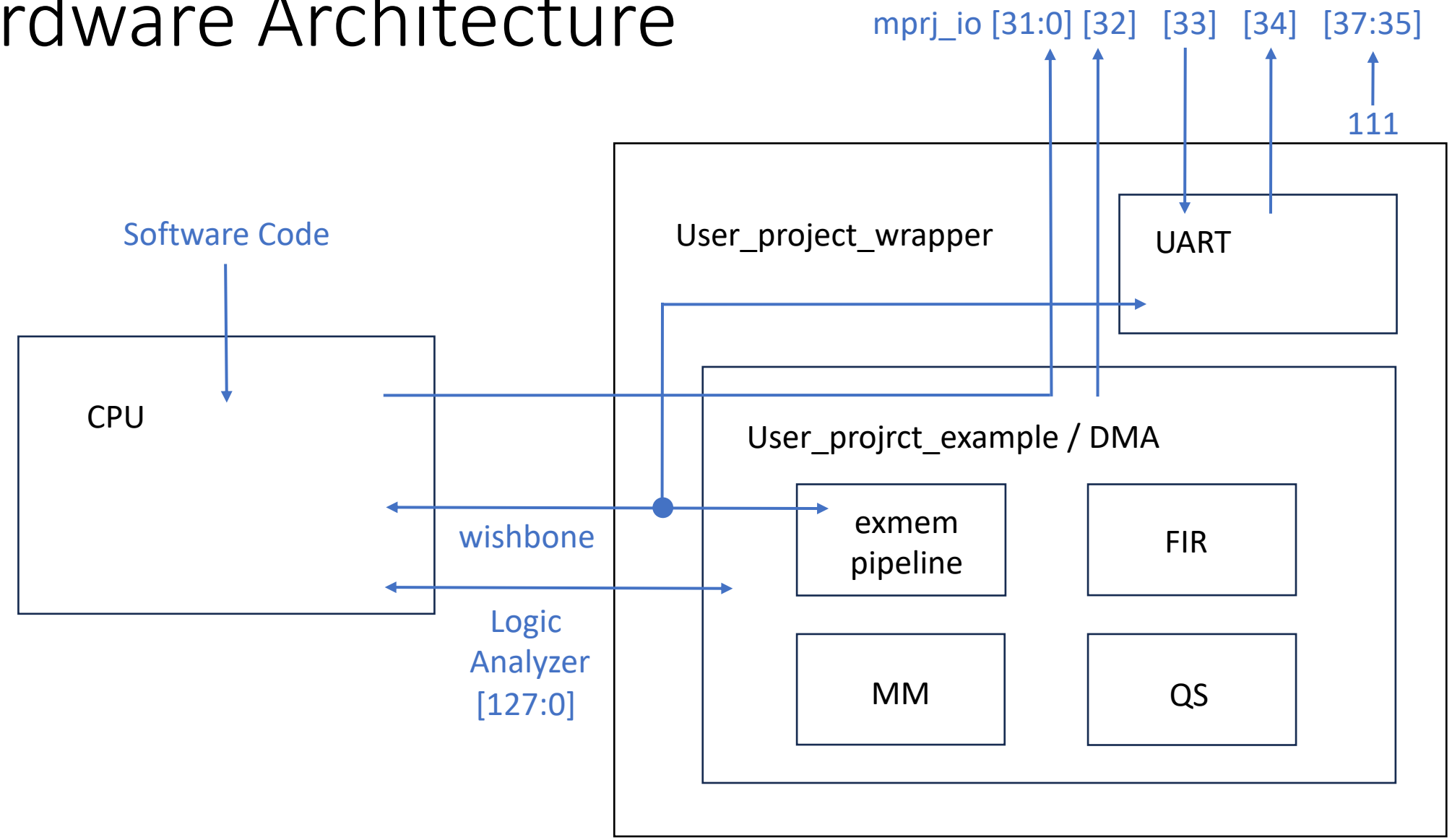
- Solution
  - New IRQ mechanism: no IRQ if the previous IRQ is pending
  - New ISR procedure: transfer data until RX_FIFO is empty

- 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

final.h 9+

testbench > final > 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

sections.lds 9+

firmware > 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 = 0x380000c0, 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

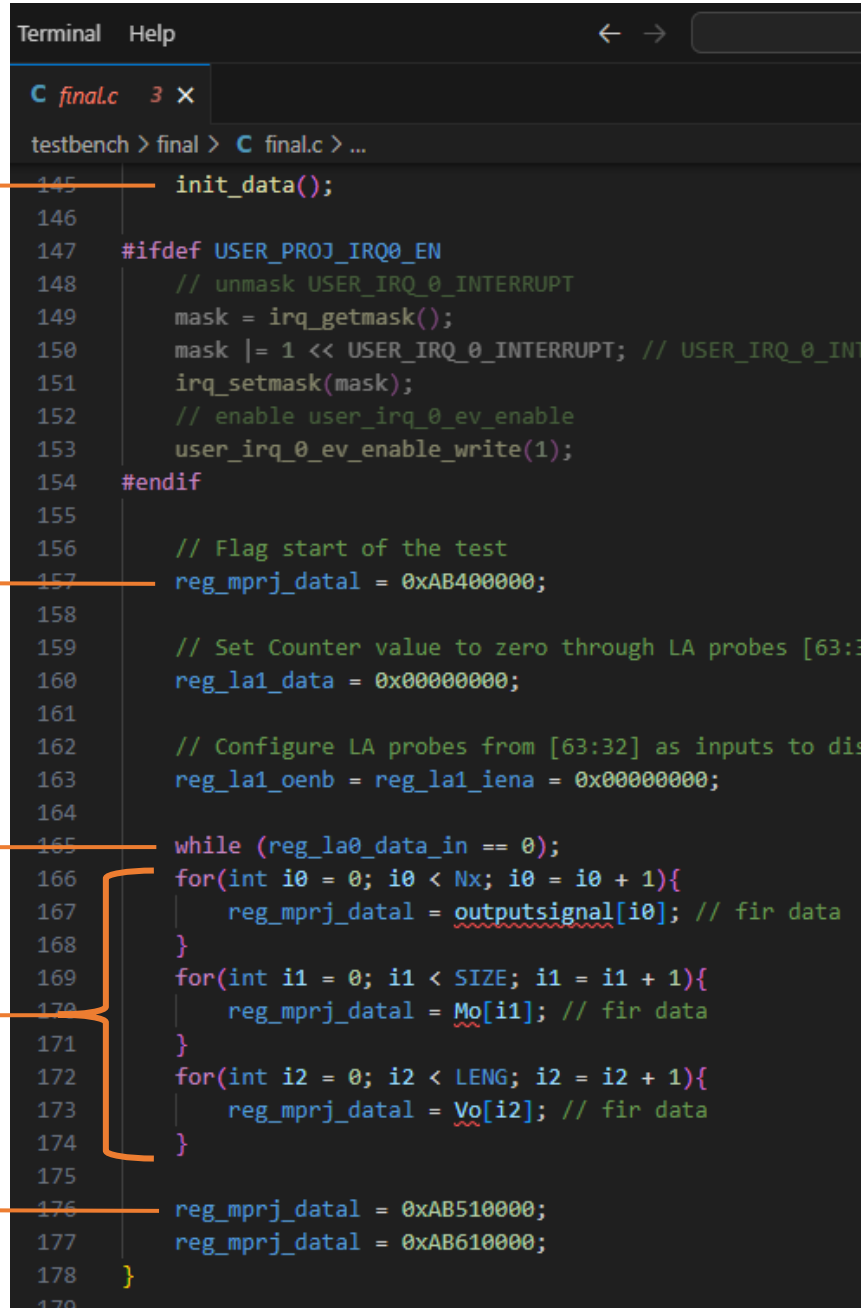
Initialize &
Pre-load data

Workload start

Wait workload to
be finished

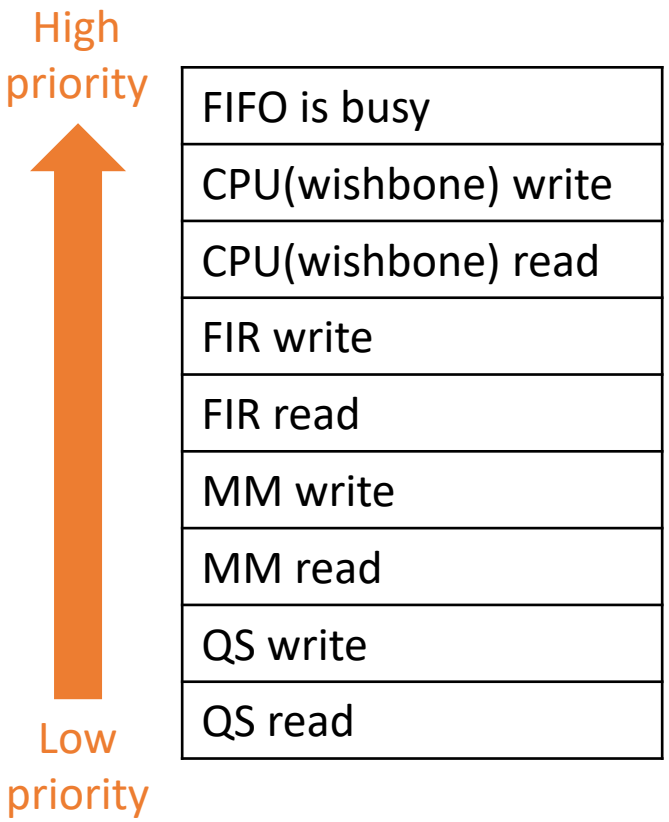
Read results from
exmem_pipeline to verify

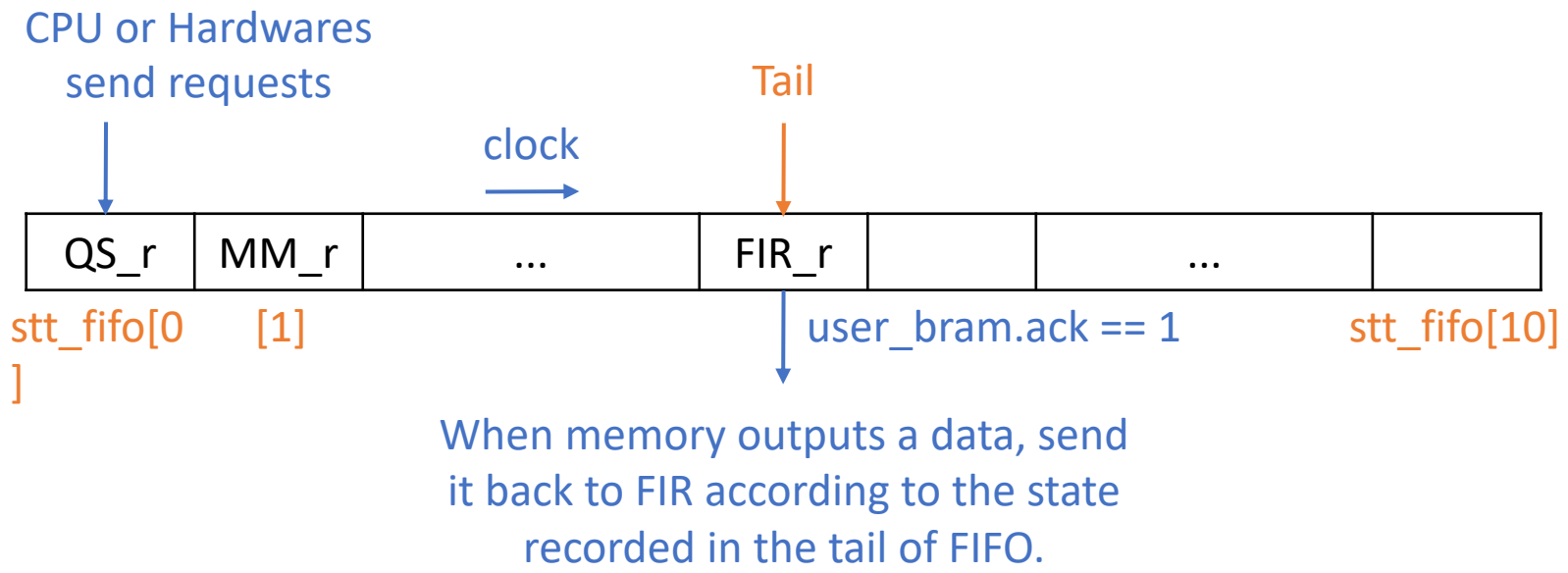
Software finished



DMA priority & state FIFO

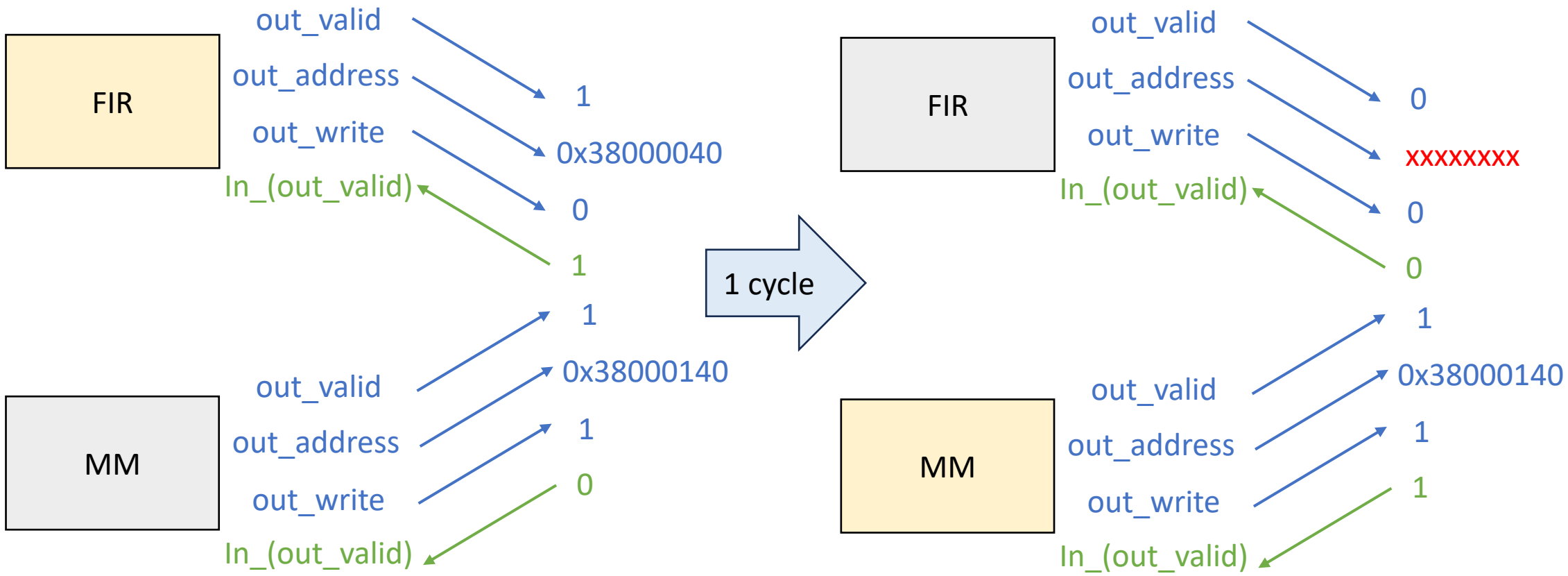
- Use state FIFO to be compatible with other memory (needn't to be 10T)





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 = 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 QUICKSORT$(A, 1, n)$.

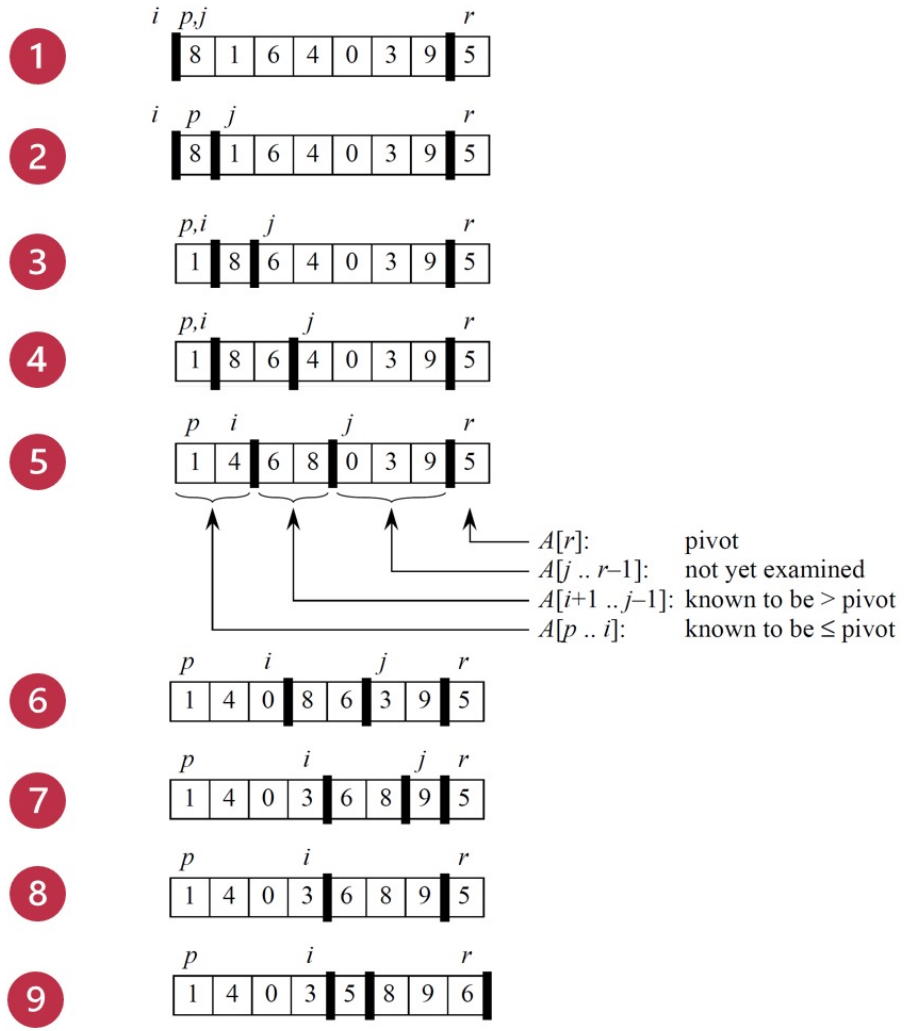
PARTITIONING

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

PARTITION(A, p, r)

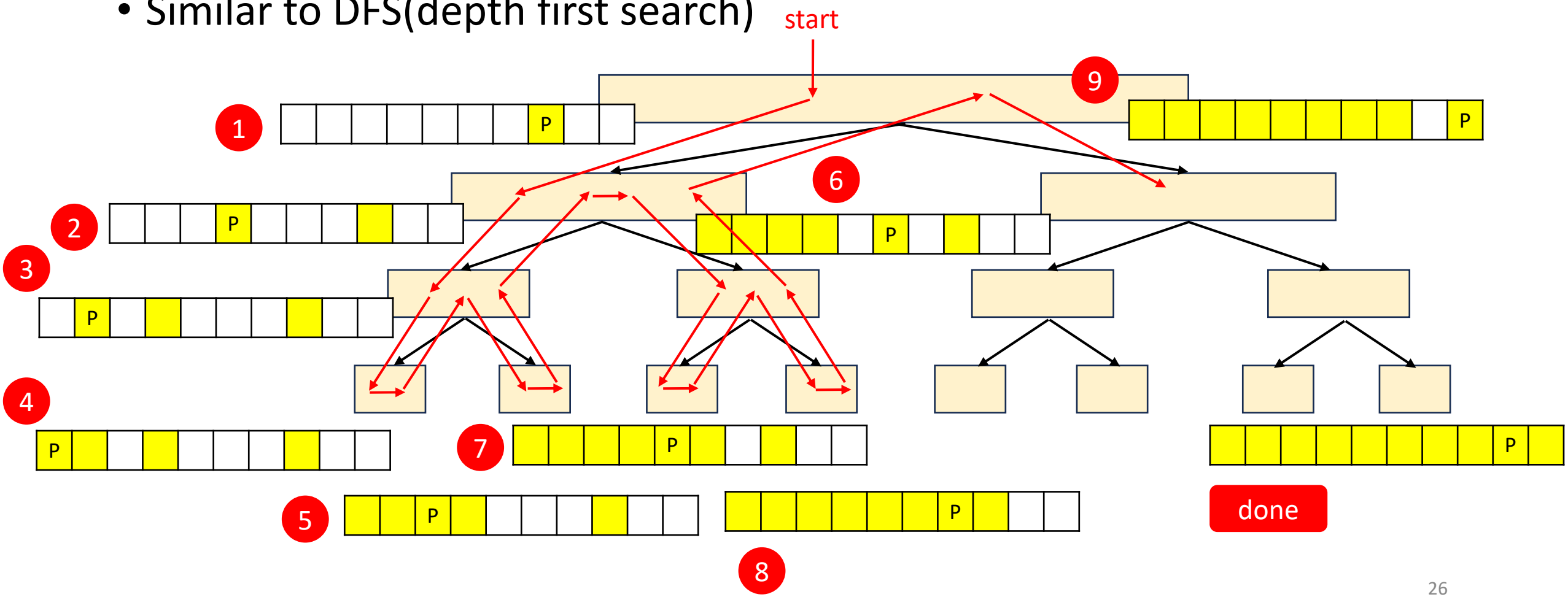
```
1  x = A[r]                          // the pivot
2  i = p − 1                         // highest index into the low side
3  for j = p to r − 1                // process each element other than the pivot
4      if A[j] ≤ x                   // does this element belong on the low side?
5          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	…

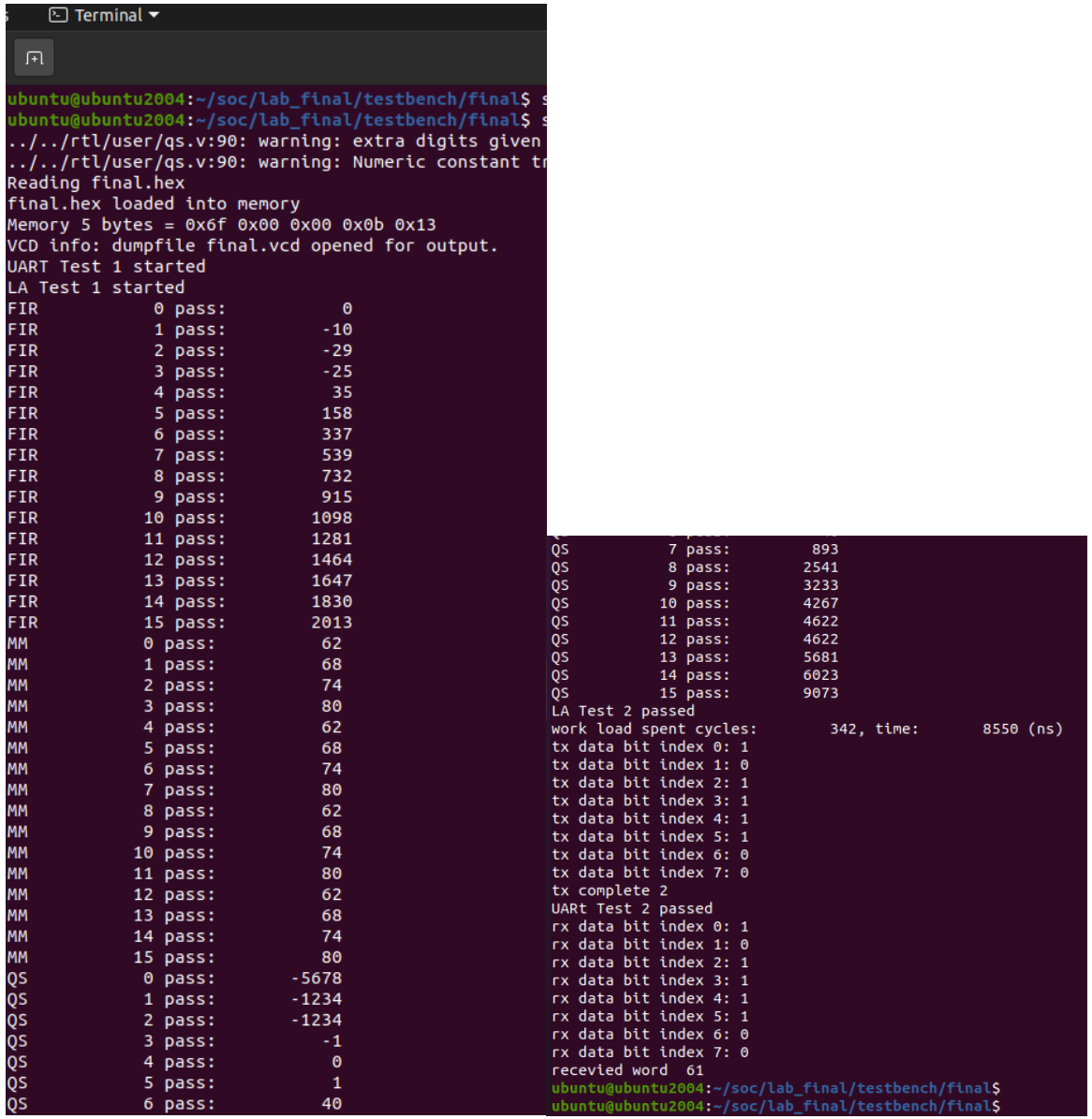
- B[16]:

1	2	3	4	…	16

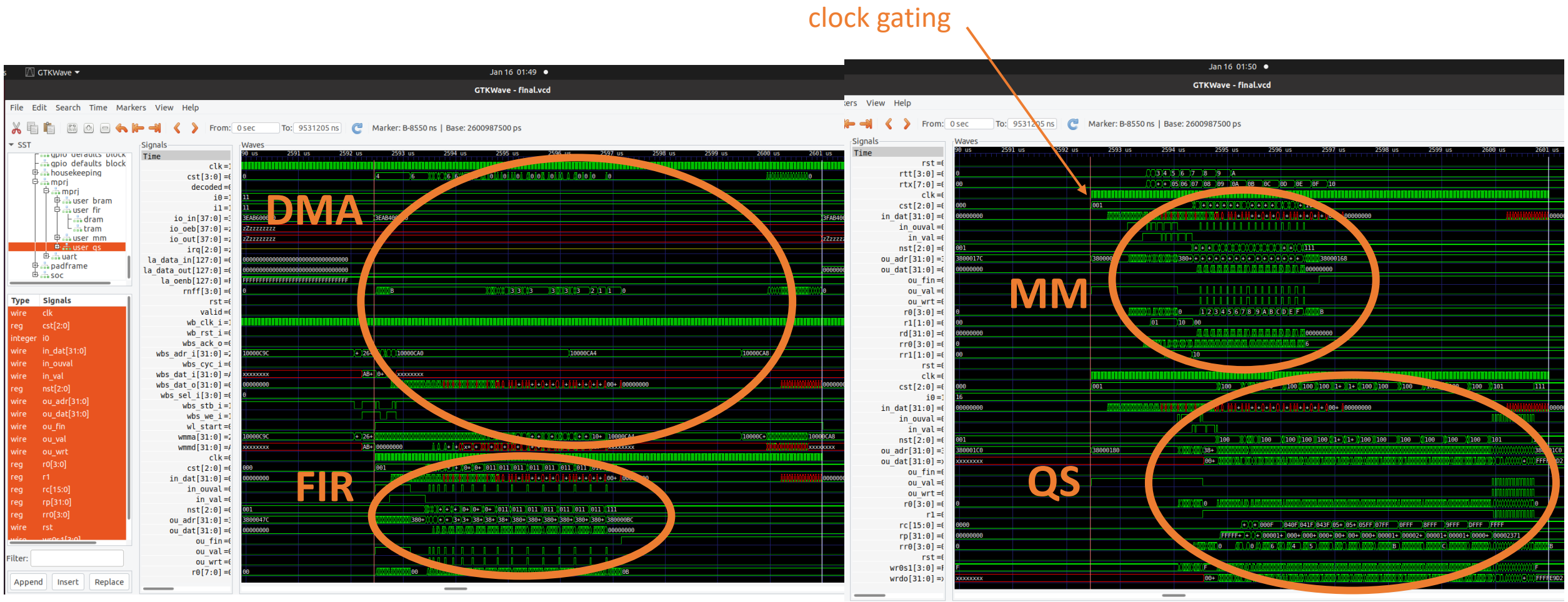
- V[16]:

893	40	…	0	-1

- Latency: 342 cycles; 8550 ns



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.