SoC Design
Lab4-2 Optimization

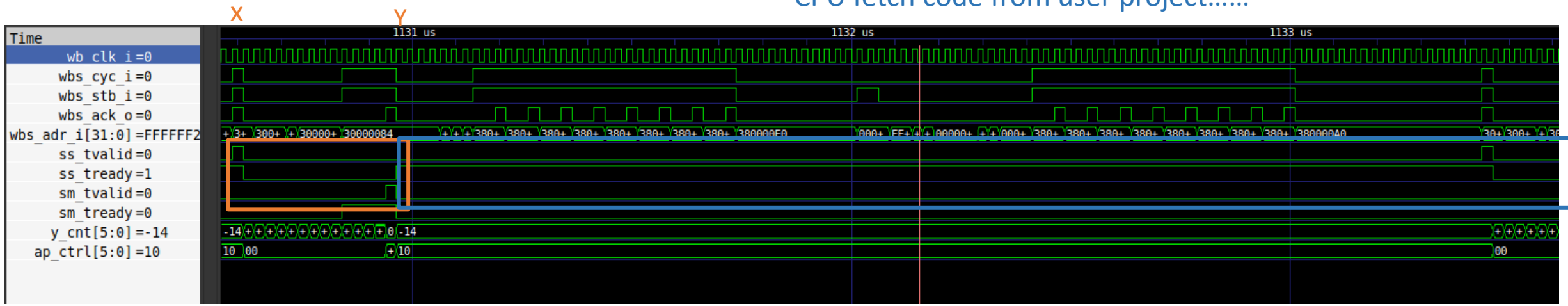
110590022 陳冠晰

Bottleneck:

ss_tvalid = CPU “Write X”
sm_tready = CPU “Read Y”

- Latency / throughput

CPU fetch code from user project……



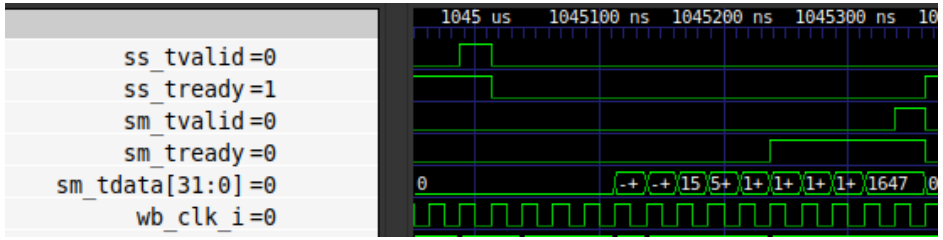
Hardware calculating FIR

```
----> Start FIR Test          1
Start latency-timer...
Final Y = 118
Finish processing...
Executes in        7497 cycles
```

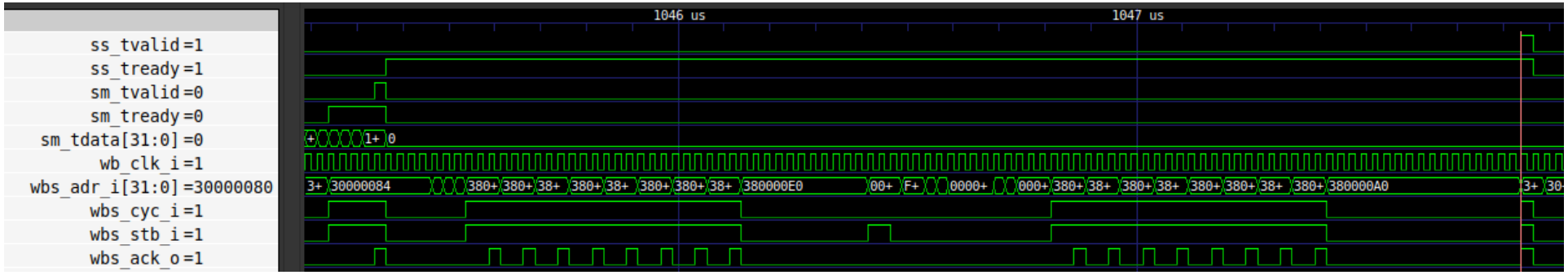
Final Y: 10614 = 0x2976
We only output 8-bit: 0x76 = 118

What can we improve?

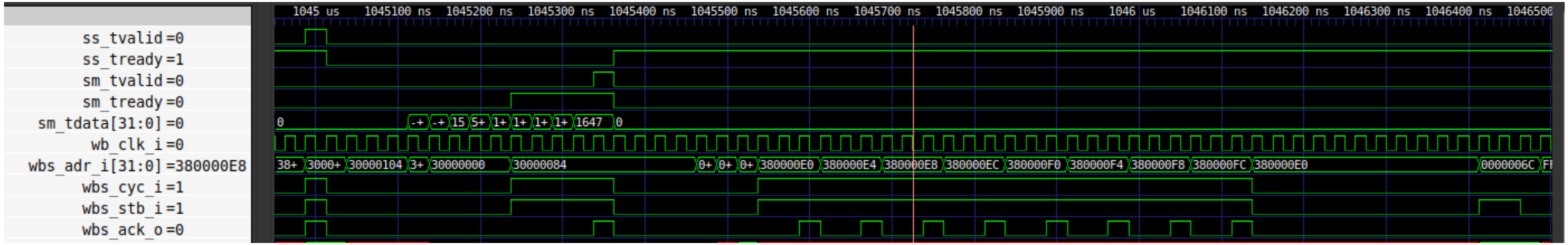
- Hardware (14T)



- Firmware (???T)



Waveform (Before)



- Hardware calculates FIR (X->Y)
- CPU fetch code from user project (Y->X)

Original Assembly Code

```
while (t < data_length) {

        wb_write(reg_fir_x_in, t); // write x into fir

        outputsignal[t] = wb_read(reg_fir_y_out);  // read Y from fir

        t = t + 1;
}
```

Assembly code of the while loop in firmware code is very long!

CPU Cache:

```
4412        wire                when_Fetcher_l398;
4413        (* ram_style = "block" *) reg [31:0] banks_0 [0:15];
4414        (* ram_style = "block" *) reg [27:0] ways_0_tags [0:1];
4415
```

CPU has 16 instructions cache, but loop code has 17 instructions.

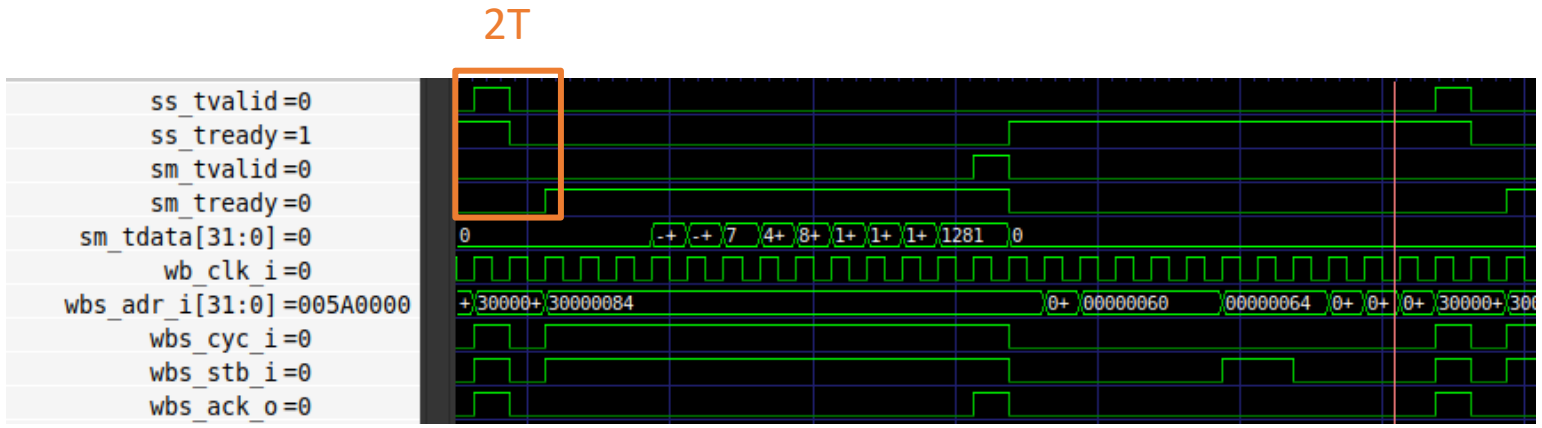
```
38000080 <fir_excute>:
38000080:       ff010113                addi    sp,sp,-16
38000084:       00812623                sw      s0,12(sp)
38000088:       00912423                sw      s1,8(sp)
3800008c:       01010413                addi    s0,sp,16
38000090:       260007b7                lui     a5,0x26000
38000094:       00c78793                addi    a5,a5,12 # 2600000c <_esram_rom+0x15fffcec>
38000098:       00a50737                lui     a4,0xa50
3800009c:       00e7a023                sw      a4,0(a5)
380000a0:       300007b7                lui     a5,0x30000
380000a4:       00100713                li      a4,1
380000a8:       00e7a023                sw      a4,0(a5) # 30000000 <_esram_rom+0x1ffffce0>
380000ac:       00000493                li      s1,0
380000b0:       0400006f                j       380000f0 <fir_excute+0x70>
380000b4:       300007b7                lui     a5,0x30000
380000b8:       08078793                addi    a5,a5,128 # 30000080 <_esram_rom+0x1ffffd60>
380000bc:       00048713                mv      a4,s1
380000c0:       00e7a023                sw      a4,0(a5)
380000c4:       300007b7                lui     a5,0x30000
380000c8:       08478793                addi    a5,a5,132 # 30000084 <_esram_rom+0x1ffffd64>
380000cc:       0007a783                lw      a5,0(a5)
380000d0:       00048613                mv      a2,s1
380000d4:       00078693                mv      a3,a5
380000d8:       03400713                li      a4,52
380000dc:       00261793                slli    a5,a2,0x2
380000e0:       00f707b3                add     a5,a4,a5
380000e4:       00d7a023                sw      a3,0(a5)
380000e8:       00148793                addi    a5,s1,1
380000ec:       0ff7f493                zext.b  s1,a5
380000f0:       03f00793                li      a5,63
380000f4:       fc97f0e3                bgeu    a5,s1,380000b4 <fir_excute+0x34>
380000fc:       0007a783                lw      a5,0(a5) # 30000000 <_esram_rom+0x1ffffce0>
38000100:       03400793                li      a5,52
38000104:       0fc7a783                lw      a5,252(a5)
38000108:       01879713                slli    a4,a5,0x18
3800010c:       005a07b7                lui     a5,0x5a0
38000110:       00f76733                or      a4,a4,a5
38000114:       260007b7                lui     a5,0x26000
38000118:       00c78793                addi    a5,a5,12 # 2600000c <_esram_rom+0x15fffcec>
3800011c:       00e7a023                sw      a4,0(a5)
38000120:       00000013                nop
38000124:       00c12403                lw      s0,12(sp)
38000128:       00812483                lw      s1,8(sp)
3800012c:       01010113                addi    sp,sp,16
```

Shorten the Assembly Code

```
rm -f counter_la_fir.hex

riscv32-unknown-elf-gcc -Wl,--no-warn-rwx-segments -g \
        --save-temps \
        -Xlinker -Map=output.map \
        -I../../firmware \
        -march=rv32i -mabi=ilp32 -D__vexriscv__ \
        -Wl,-Bstatic,-T,../../firmware/sections.lds,--strip-discarded \
        -ffreestanding -nostartfiles -O2 -o counter_la_fir.elf ../../firmware/crt0_vex.S ../../firmware/isr.c fir.c fir_control.c

# -nostartfiles
riscv32-unknown-elf-objcopy -O verilog counter_la_fir.elf counter_la_fir.hex
riscv32-unknown-elf-objdump -D counter_la_fir.elf > counter_la_fir.out

# to fix flash base address
sed -ie 's/@10/@00/g' counter_la_fir.hex

iverilog -Ttyp -DFUNCTIONAL -DSIM -DUNIT_DELAY=#1 \
        -f./include.rtl.list -o counter_la_fir.vvp counter_la_fir_tb.v

vvp counter_la_fir.vvp
rm -f counter_la_fir.vvp counter_la_fir.elf counter_la_fir.hexe
```

Add O1, O2, O3 or Ofast to optimize the assembly code

Result:



X->Y: 14T

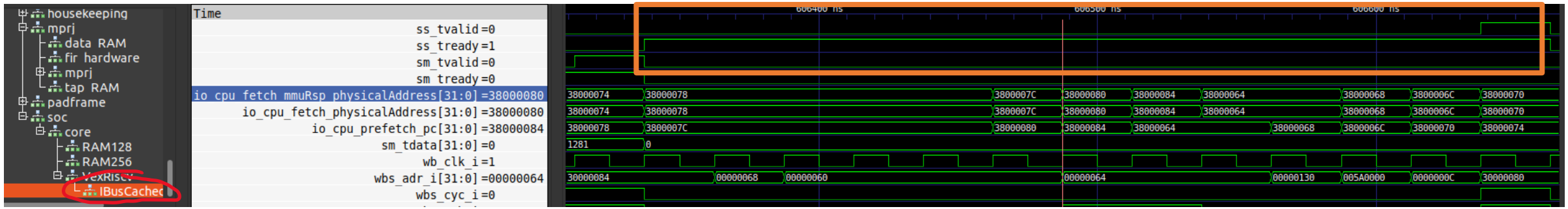
Y->X: 13T

```
----> Start FIR Test            1
Start latency-timer...
Final Y = 118
Finish processing...
Executes in        1701 cycles
--------------------------------
```

Why?

CPU fetch code from Cache

Takes 13T




```
38000038 <fir_excute>:
38000038:       260007b7                lui     a5,0x26000
3800003c:       00a506b7                lui     a3,0xa50
38000040:       00d7a623                sw      a3,12(a5) # 2600000c <_esram_rom+0x15fffde4>
38000044:       300007b7                lui     a5,0x30000
38000048:       00100693                li      a3,1
3800004c:       00d7a023                sw      a3,0(a5) # 30000000 <_esram_rom+0x1ffffdd8>
38000050:       03400513                li      a0,52
38000054:       00000793                li      a5,0
38000058:       03400713                li      a4,52
3800005c:       300006b7                lui     a3,0x30000
38000060:       04000593                li      a1,64
38000064:       08f6a023                sw      a5,128(a3) # 30000080 <_esram_rom+0x1ffffe58>
38000068:       0846a603                lw      a2,132(a3)
3800006c:       00178793                addi    a5,a5,1
38000070:       00470713                addi    a4,a4,4
38000074:       fec72e23                sw      a2,-4(a4)
38000078:       feb796e3                bne     a5,a1,38000064 <fir_excute+0x2c>
3800007c:       0fc52783                lw      a5,252(a0)
38000080:       005a0737                lui     a4,0x5a0
38000084:       01879793                slli    a5,a5,0x18
38000088:       00e7e7b3                or      a5,a5,a4
3800008c:       0006a703                lw      a4,0(a3)
38000090:       26000737                lui     a4,0x26000
38000094:       00f72623                sw      a5,12(a4) # 2600000c <_esram_rom+0x15fffde4>
38000098:       00008067                ret
```

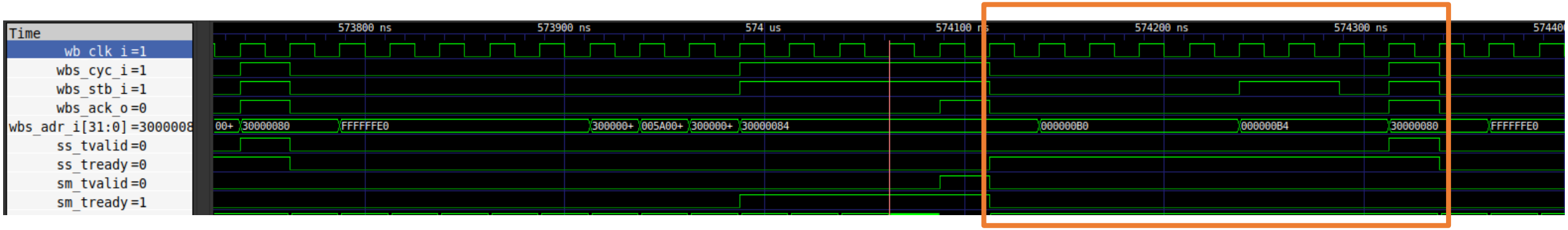
In hardware write-X to read-Y takes latency for HW calculation

Read-Y to write-X: HW does nothing, simply wait for X

Let see if we start the while loop with reading Y-out

Start with Y:

Reduce to 9T

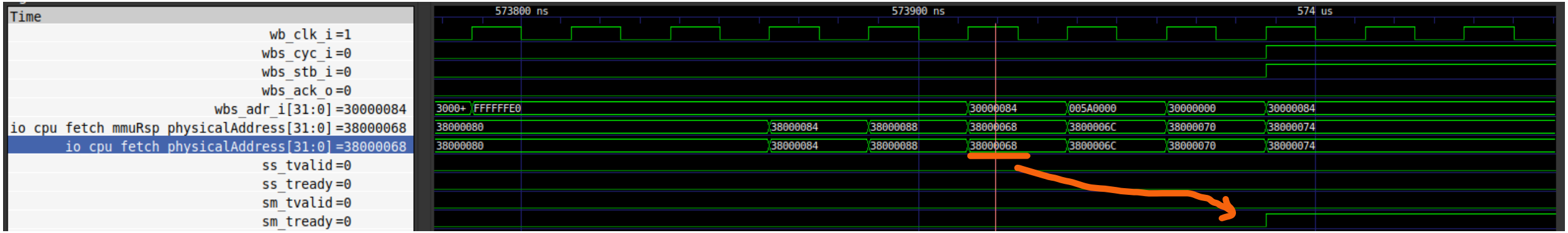


Write 1st X outside the loop

Loop start with reading Y

```
wb_write(reg_fir_x_in, t);

while (t < data_length-1) {

        [redacted]

        outputsignal[t] = wb_read(reg_fir_y_out);  // read Y from fir

        t = t + 1;

        wb_write(reg_fir_x_in, t);
}

outputsignal[N-1] = wb_read(reg_fir_y_out);
```

Take a look at Cache addr when X->Y



CPU physical address to wishbone bus cycle takes 3T

Corresponding assembly code:

```
38000068: 0846a603          lw  a2,132(a3) # 30000084 <_esram_rom+0x1ffffe5c>
3800006c: 00470713          addi  a4,a4,4 # a50004 <_fstack+0xa4fa04>
38000070: fec72e23          sw  a2,-4(a4)
38000074: 08f6a023          sw  a5,128(a3)
38000078: 00178793          addi  a5,a5,1
3800007c: feb796e3          bne a5,a1,38000068 <fir_excute+0x30>
```

Further Optimization:

Re-ordering some of the instructions

```
38000068: 0846a603          lw  a2,132(a3) # 30000084 <_esram_rom+0x1ffffe5c>
3800006c: 00470713          addi  a4,a4,4 # a50004 <_fstack+0xa4fa04>
38000070: fec72e23          sw  a2,-4(a4)
38000074: 08f6a023          sw  a5,128(a3)
38000078: 00178793          addi  a5,a5,1
3800007c: feb796e3          bne a5,a1,38000068 <fir_excute+0x30>
```

lw a2, 132(a3) => get the Y from 3000_0084 and load into reg **a2**

outputsignal[a4] = 32-bit y from hardware reg

addi a4, a4, 4 => increment the value in **a4** by 4

sw a2, -4(a4) => store Y-out in reg **a2** into the memory location that is four bytes before the address stored in reg **a4**

sw a5, 128(a3) => store X from 3000_0080 into reg **a5**

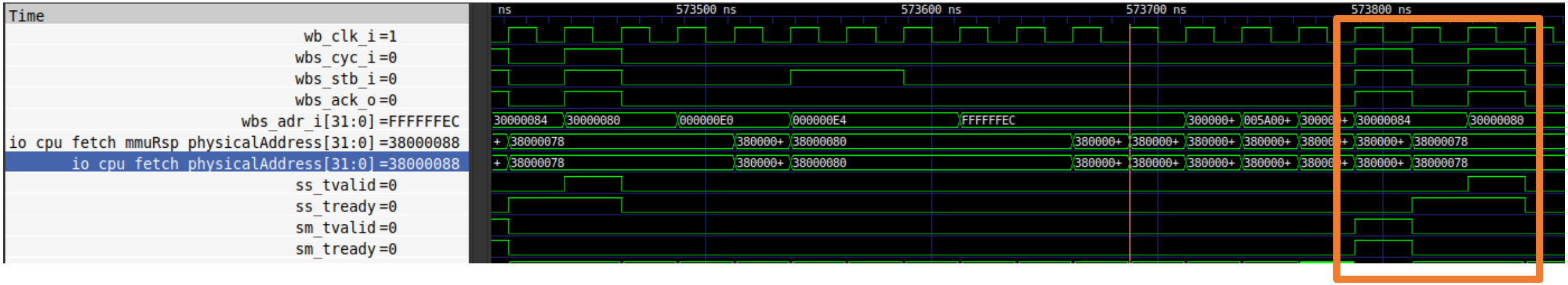
addi a5, a5, 1 => t = t + 1

```
38000068: 0846a603          lw  a2,132(a3) # 30000084 <_esram_rom+0x1ffffe5c>
3800006c: 00470713          addi  a4,a4,4 # a50004 <_fstack+0xa4fa04>
38000070: 08f6a023          sw  a5,128(a3)
38000074: fec72e23          sw  a2,-4(a4)
38000078: 00178793          addi  a5,a5,1
3800007c: feb796e3          bne a5,a1,38000068 <fir_excute+0x30>
```

The (sw a2, 0(a4)) stalled thus delays the following write x (sw a5, 128 (a3)).

Re-ordering them like the screenshot above.

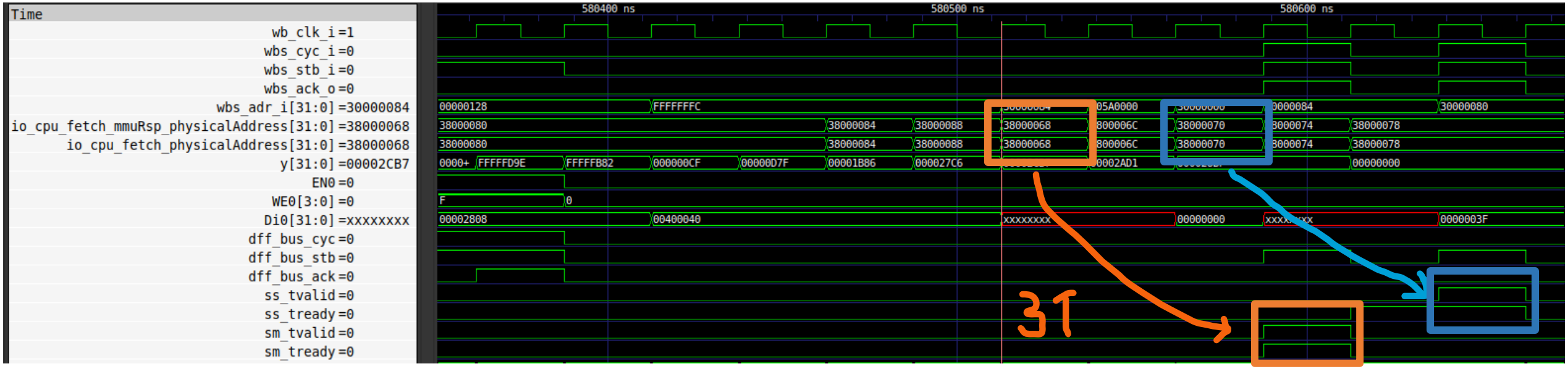
Result:



Y->X reduce to 2T

Read Y

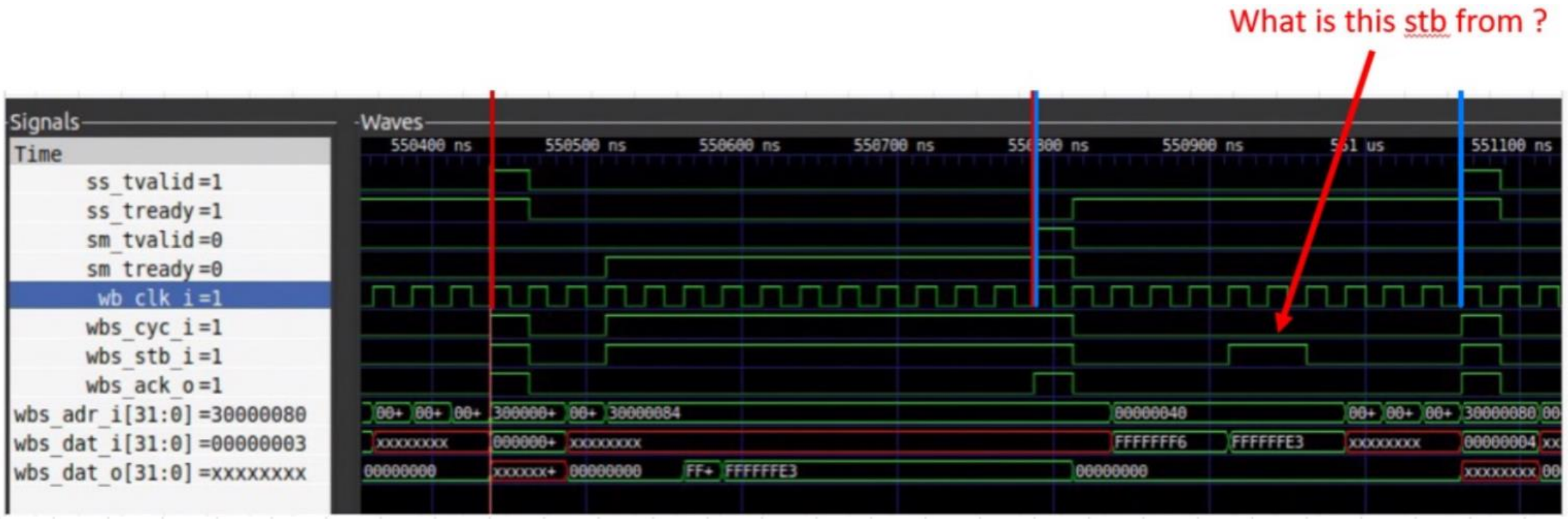
Write X



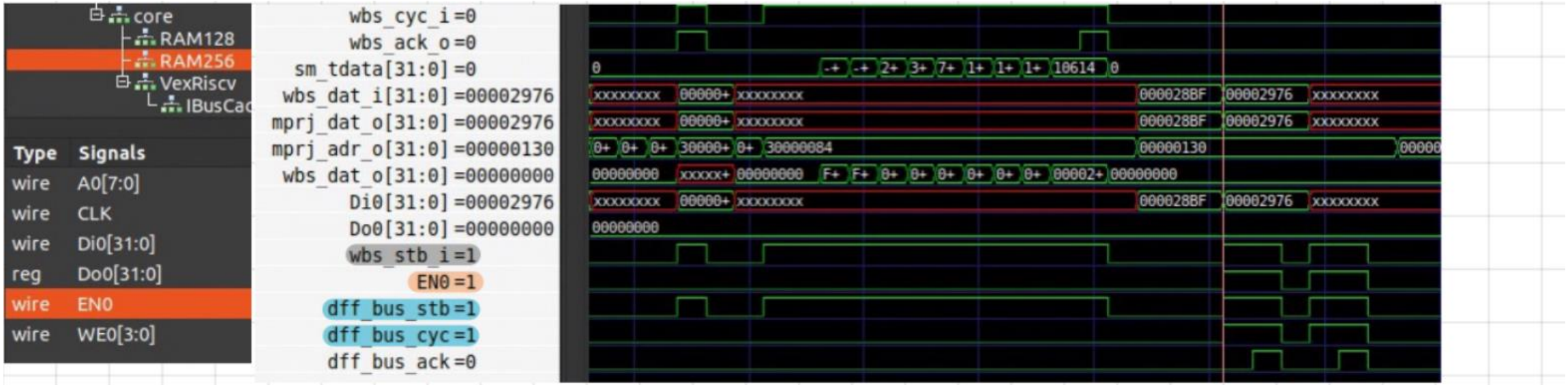
CPU physical address to wishbone bus cycle takes 3T

```
38000068: 0846a603          lw  a2,132(a3) # 30000084 <_esram_rom+0x1ffffe5c>
3800006c: 00470713          addi  a4,a4,4 # a50004 <_fstack+0xa4fa04>
38000070: 08f6a023          sw  a5,128(a3)
38000074: fec72e23          sw  a2,-4(a4)
38000078: 00178793          addi  a5,a5,1
3800007c: feb796e3          bne a5,a1,38000068 <fir_excute+0x30>
```

Where the STB from?

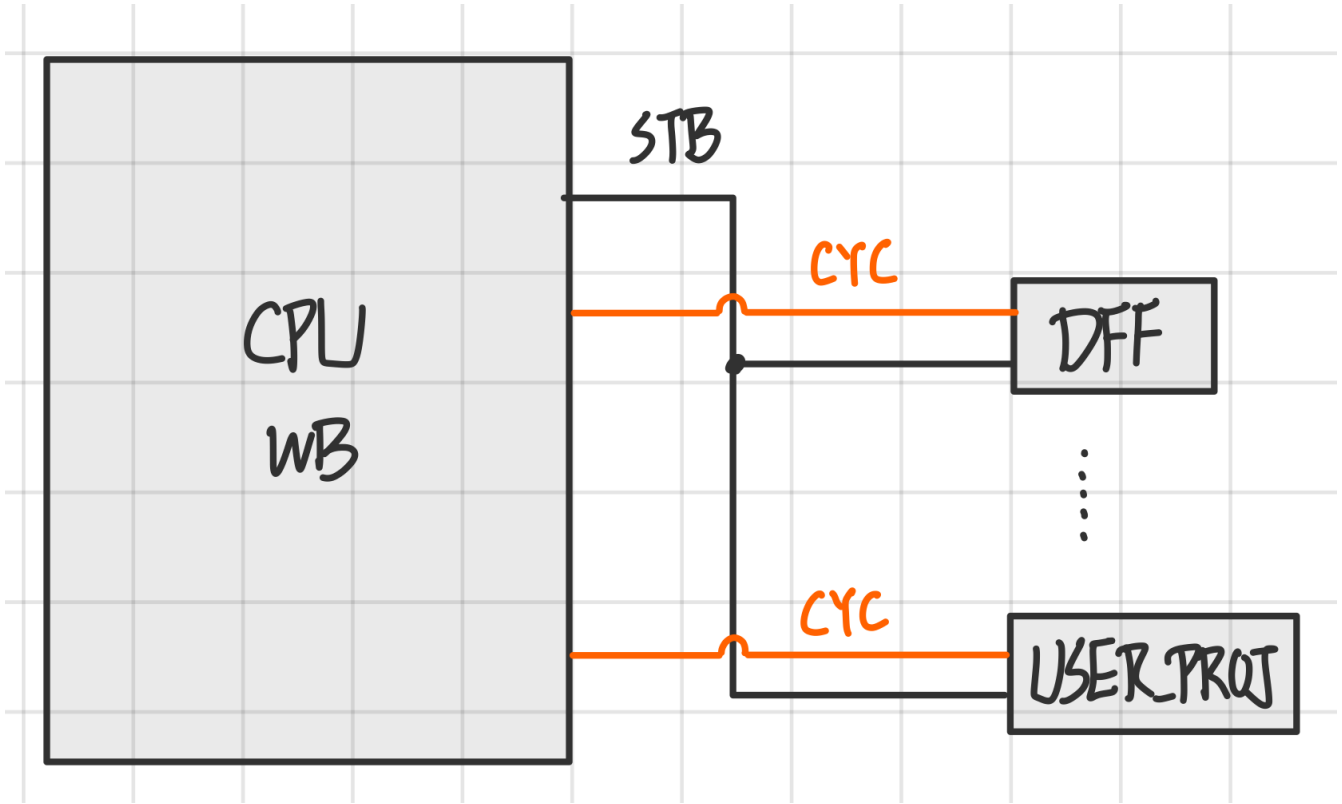


Caravel core:

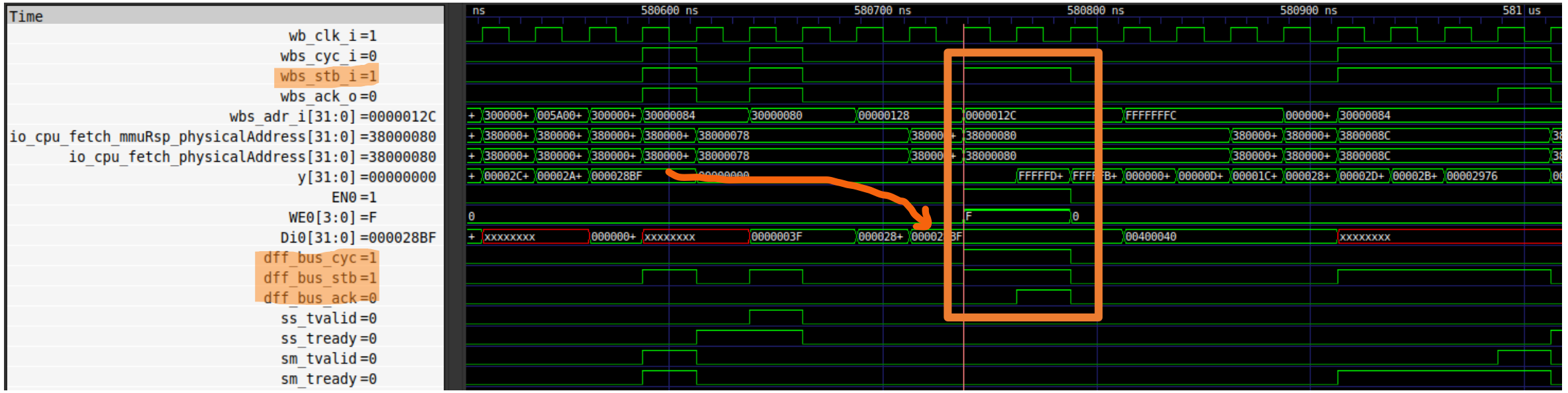


We can know that Caravel core has it own DFF Bus, different from Wishbone Bus, but has some relative characteristics.
From the waveform above, dff_bus_stb and wbs_stb_i are the same.
EN0 is the enable of RAM256 of RISC-V CPU, which is driven by dff_bus_cyc & dff_bus_stb. (VALID)

Although they share the same STB, but their CYC are independent.

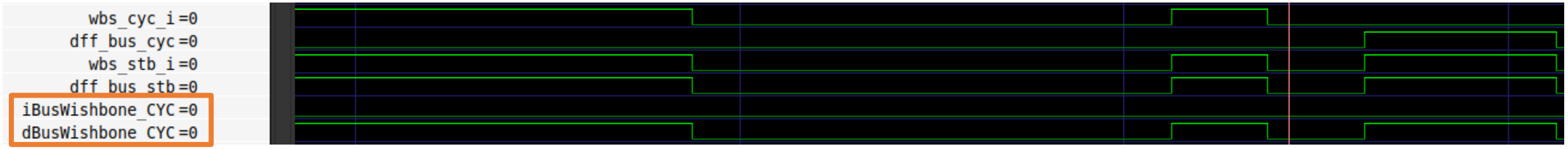


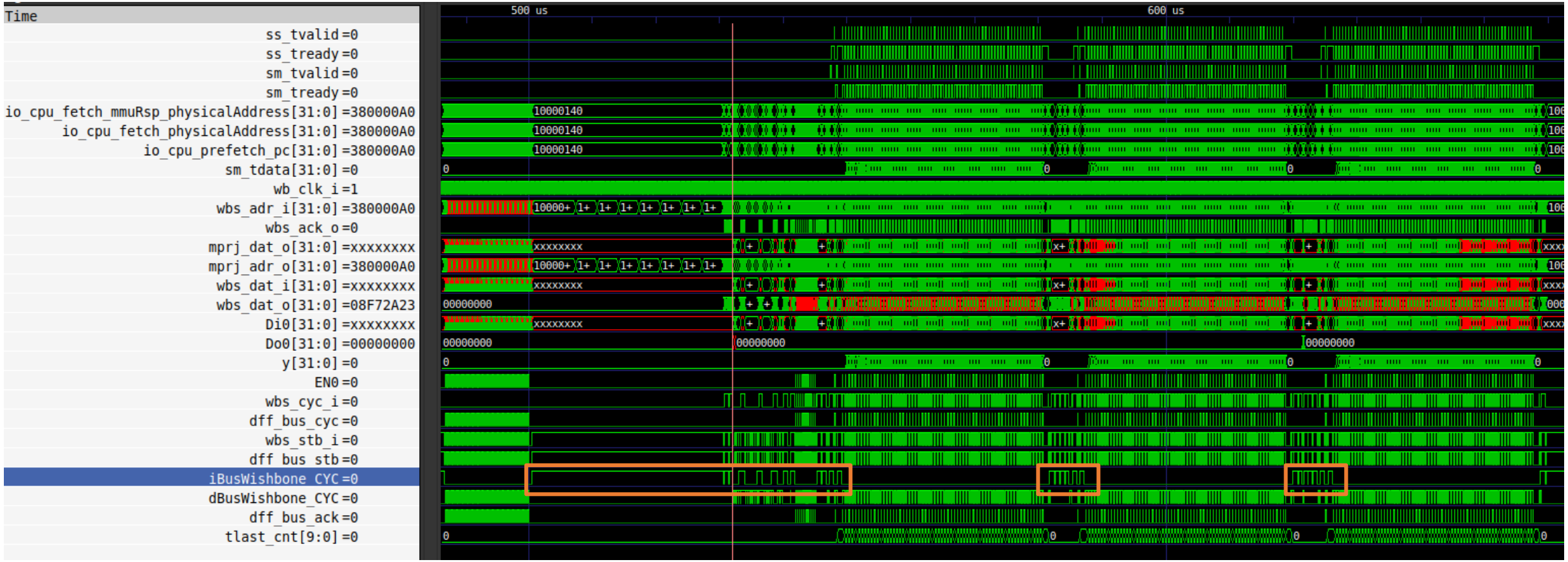
Write Y-out into DFF




```
MEMORY {
        vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
        dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
        dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
        flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
        mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000
        mprjram : ORIGIN = 0x38000000, LENGTH = 0x00400000
        hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
        csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
```

Instruction / Data BUS
Wishbone CYC in Caravel core





CPU transfer instructions

Relative: Data_WB_CYC <- DFF_WB_CYC <- DFF_WB_STB = WBS_STB_I

We can know that when STB in user project asserted without CYC, it probably because CPU is using wishbone bus to transfer data in different place.

Summarize:

- Original cycle count: about 100T
- Cycle count after shorten the assembly code: 27T
- Cycle count after start loop with Y->X: 23T
- Cycle count after re-order the instruction: 16T