Direct Memory Access Controller

Design Review

ECE 551 – SoC Design

Members (Team 4):

- Sk Hasibul Alam
- Milad Tanavardi Nasab
- Tanjina Sabrin

What is Direct Memory Access?

Feature that enables some hardware subsystems to access primary memory independent from the CPU.

From the CPU's perspective, it:

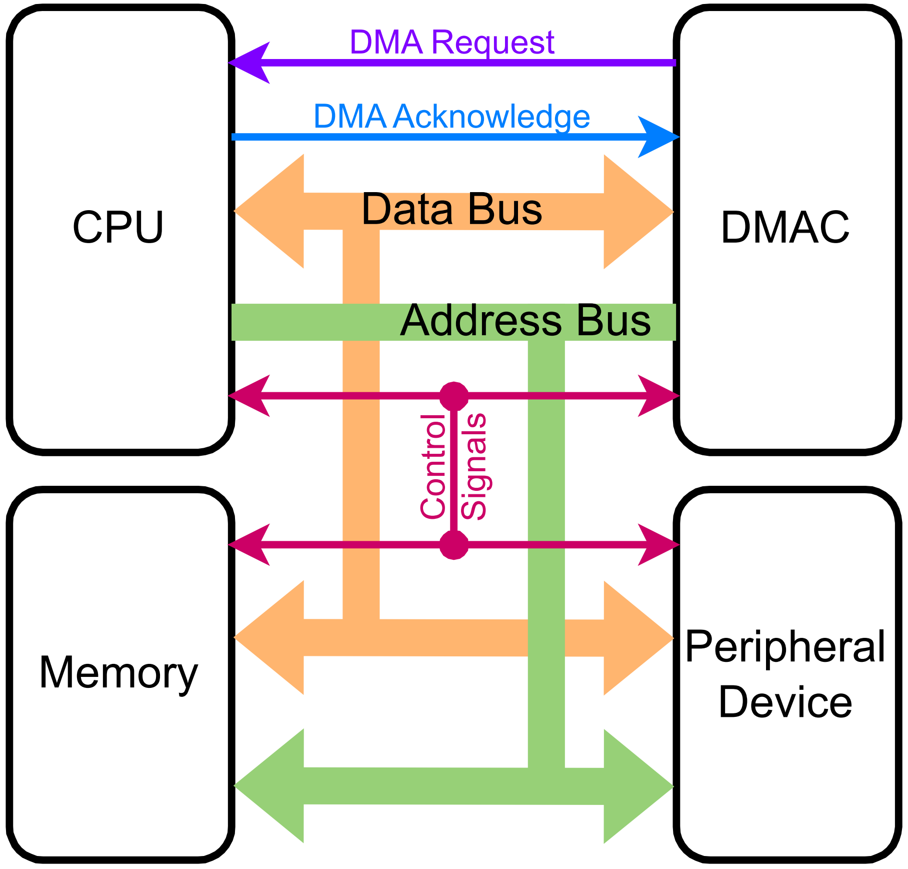
initiates the transfer first

↓

does other tasks while the transfer is ongoing

↓

waits for the DMAC to interrupt it when the operation is complete



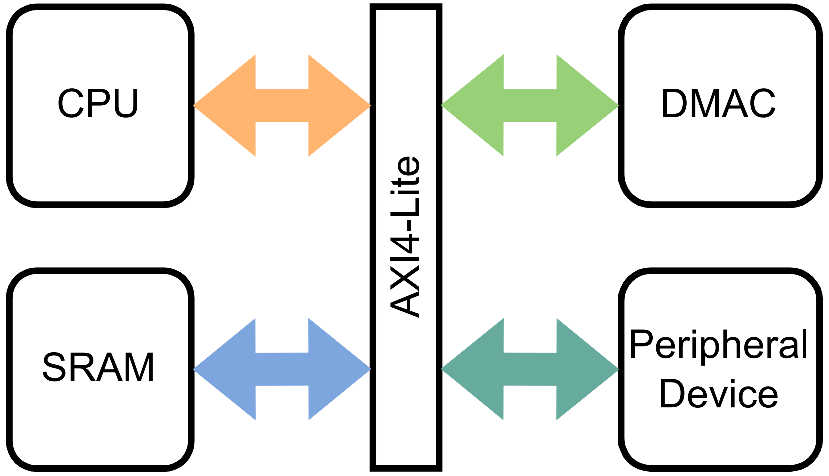
* DMAC = Direct Memory Access Controller

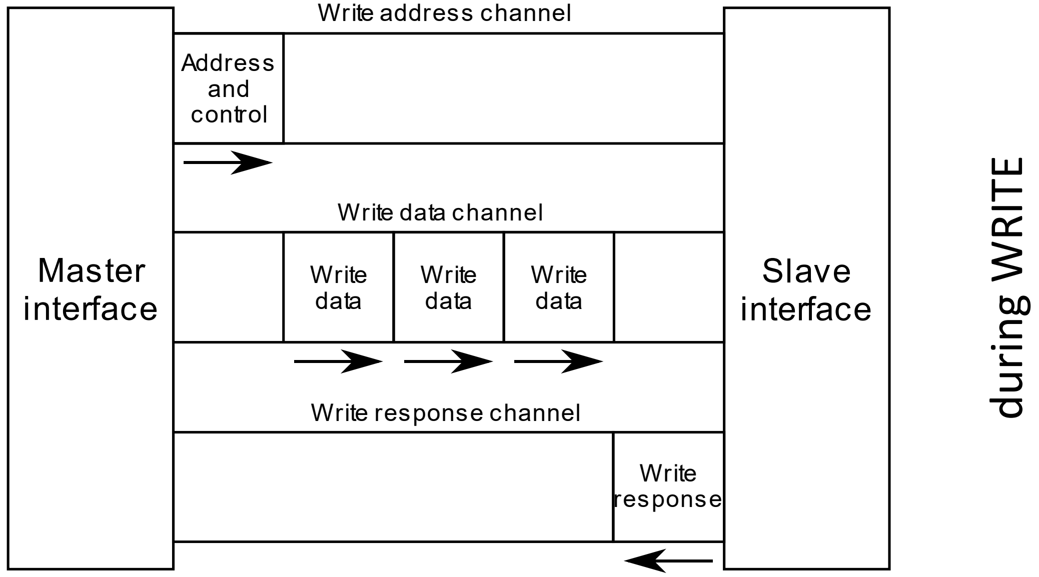
System with AXI4-Lite

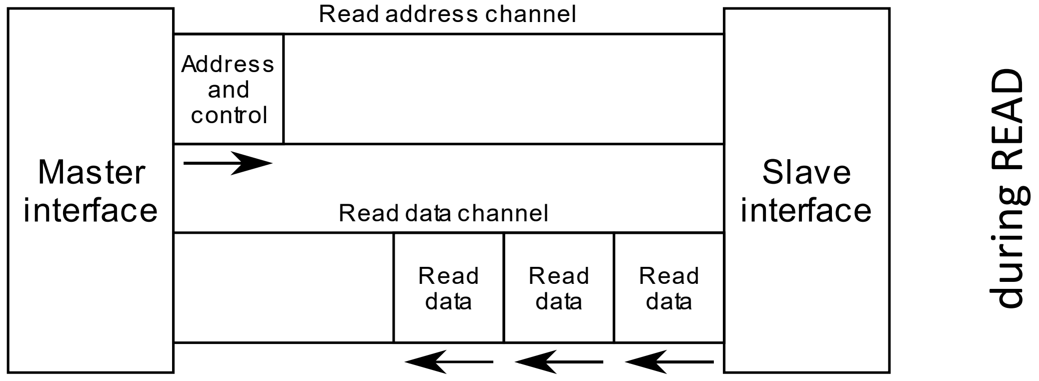
AXI = **A**dvanced e**X**tensible **I**nterface

Communicates using 5 independent channel sets:

- Read Address channel (AR)
- Read Data channel (R)
- Write Address channel (AW)
- Write Data channel (W)
- Write Response channel (B)

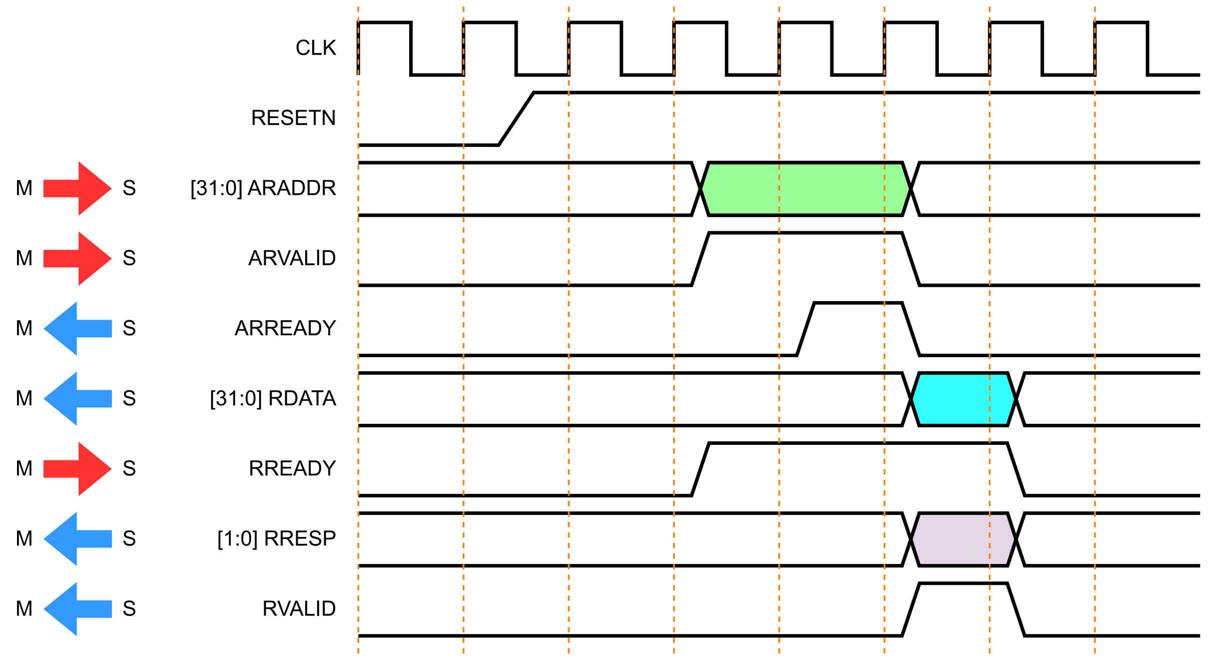




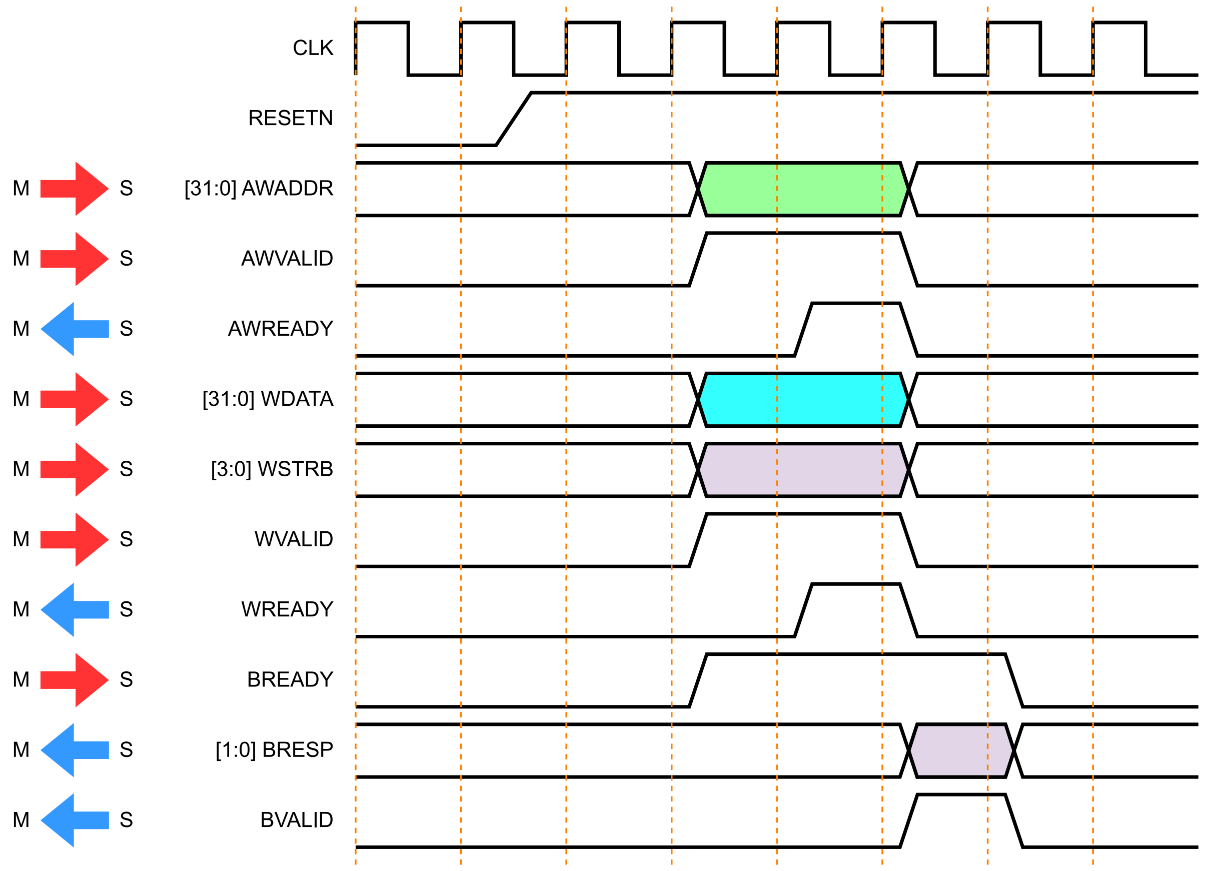


AXI4-Lite Transaction

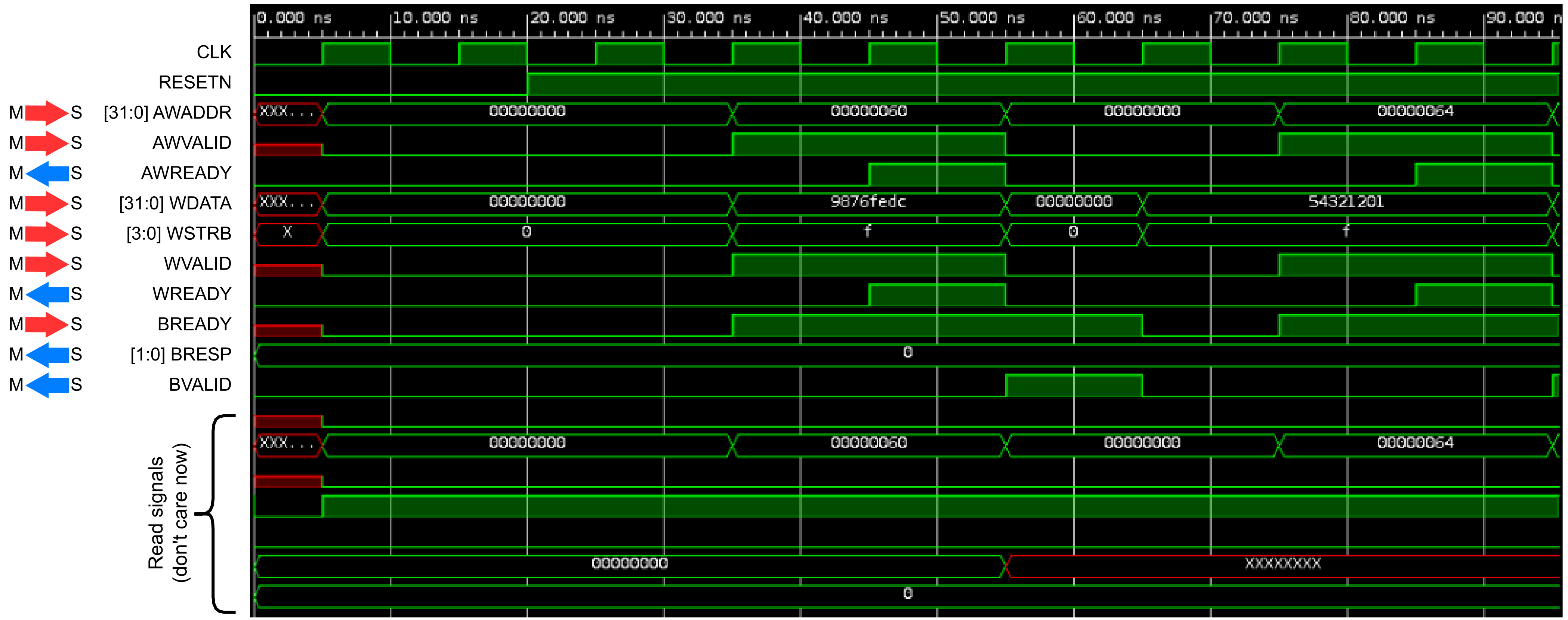
Read Cycle

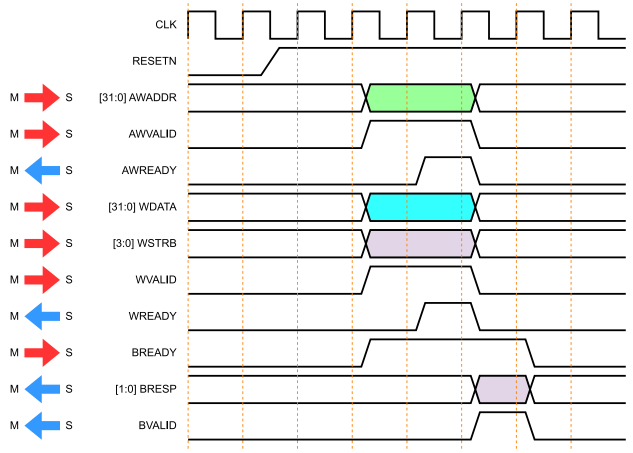


Write Cycle

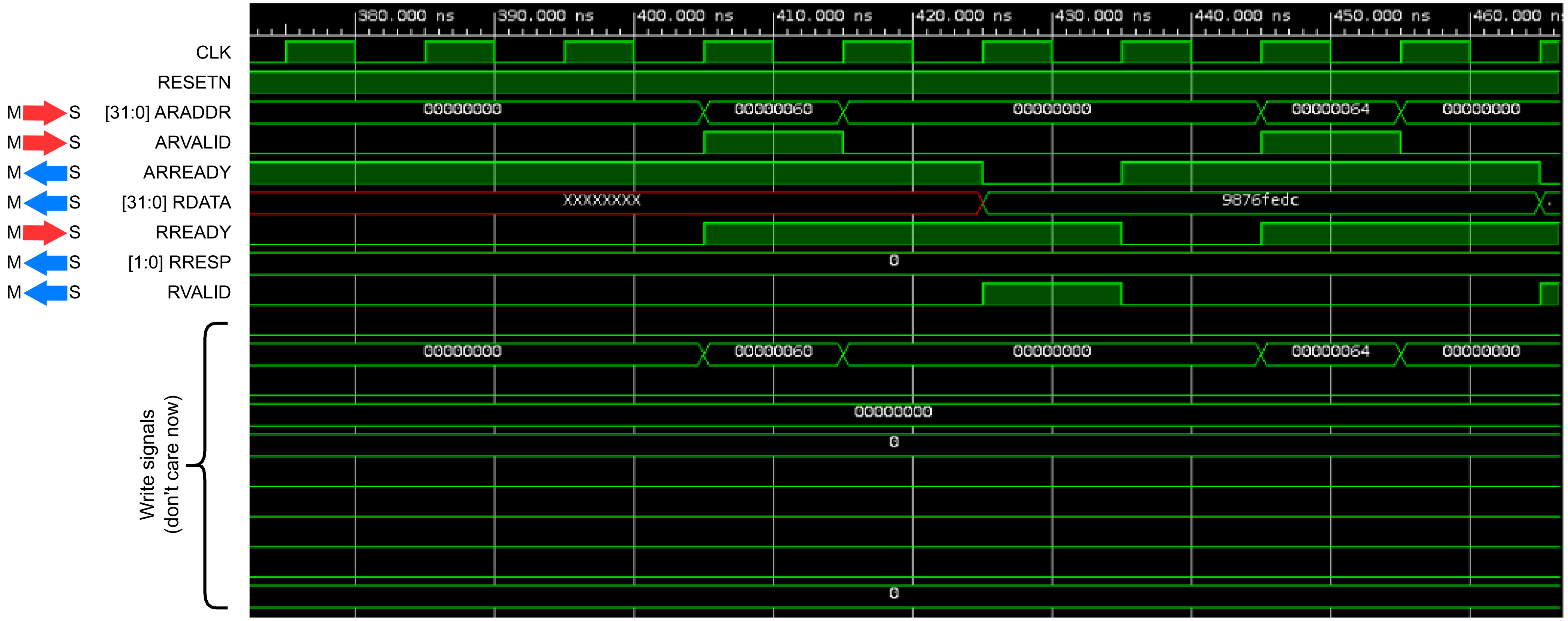


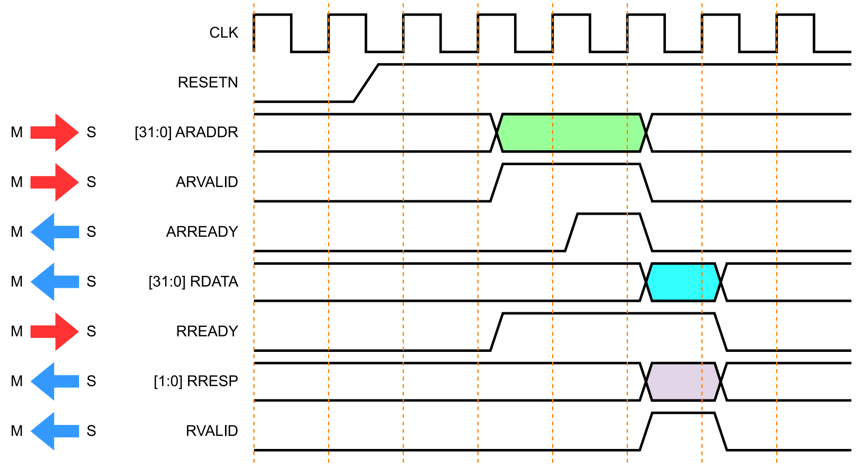
Vivado Simulation for AXI4-Lite Write Handshaking





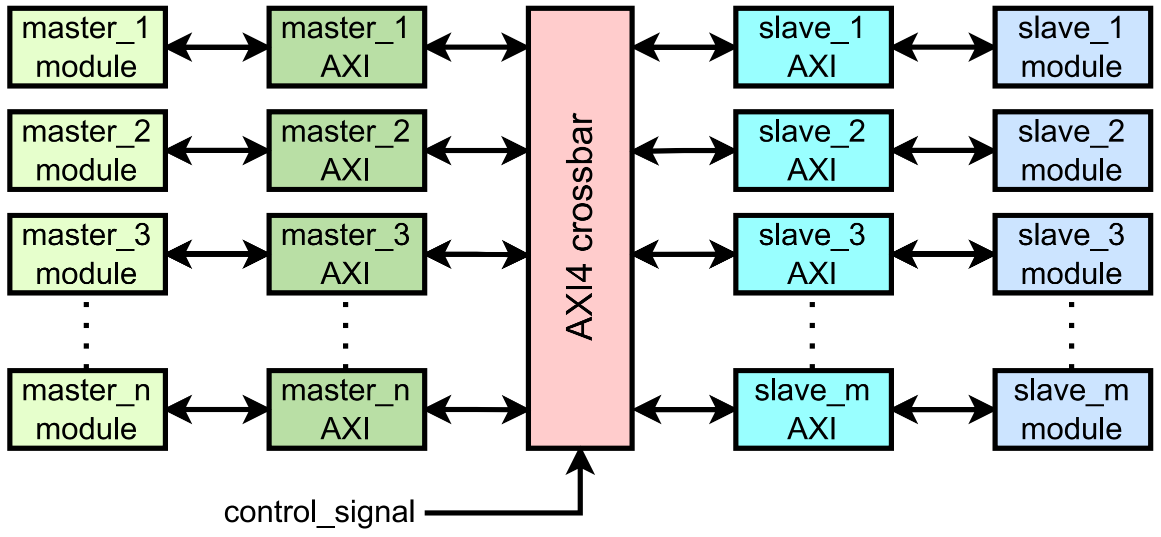
Vivado Simulation for AXI4-Lite Read Handshaking



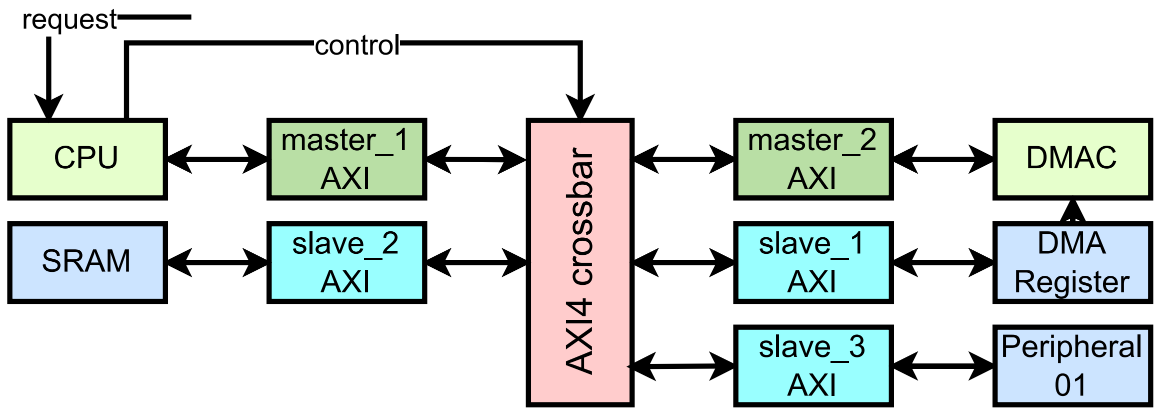


Crossbar Switch

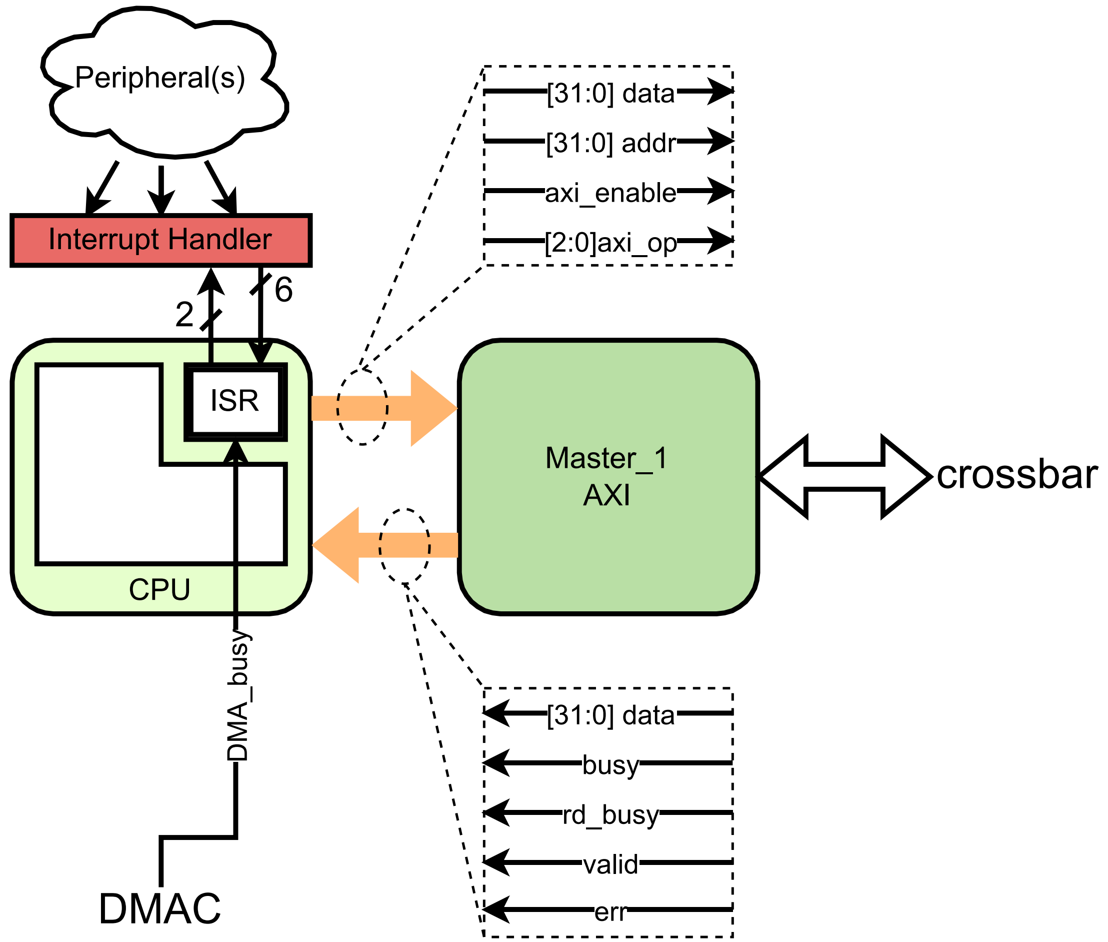
✓Needed when there are multiple masters or multiple slaves.



This project



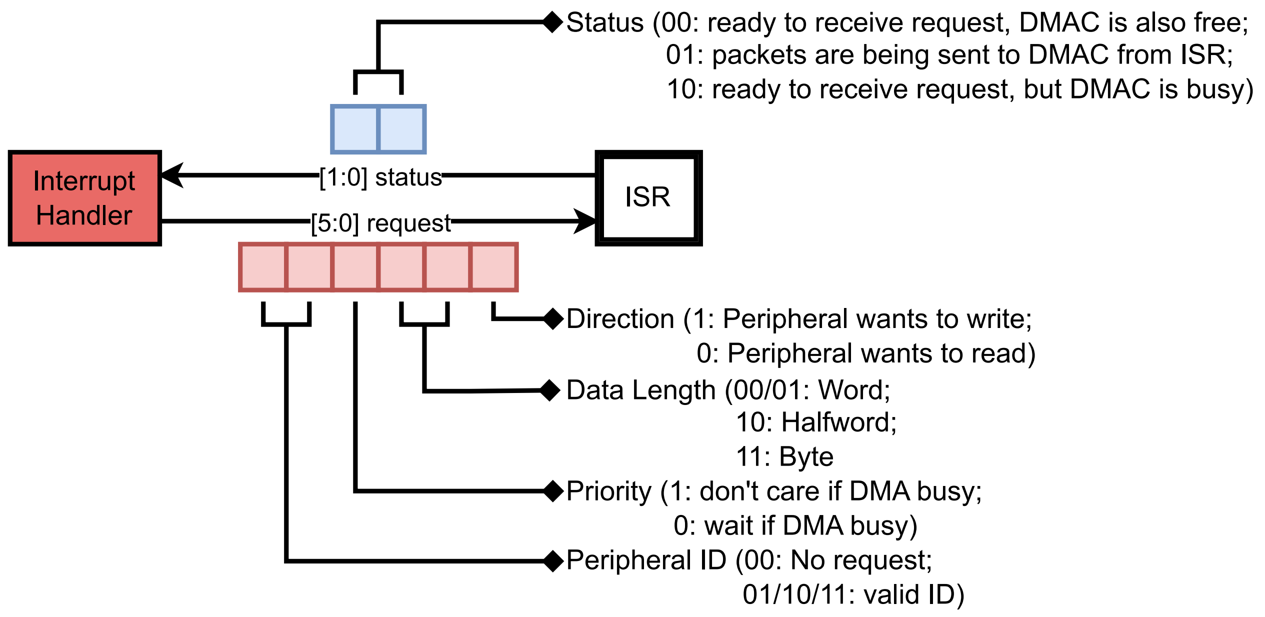
Interrupt Service Routine (ISR)



Part of the CPU

- Receives 6-bit service request from the `Interrupt Handler`
- Sends 2-bit status to the `Interrupt Handler`
- Gets DMA status directly from the `DMAC`
- Sends data and address to the `DMA Register` module when suitable
- Controls the `crossbar`

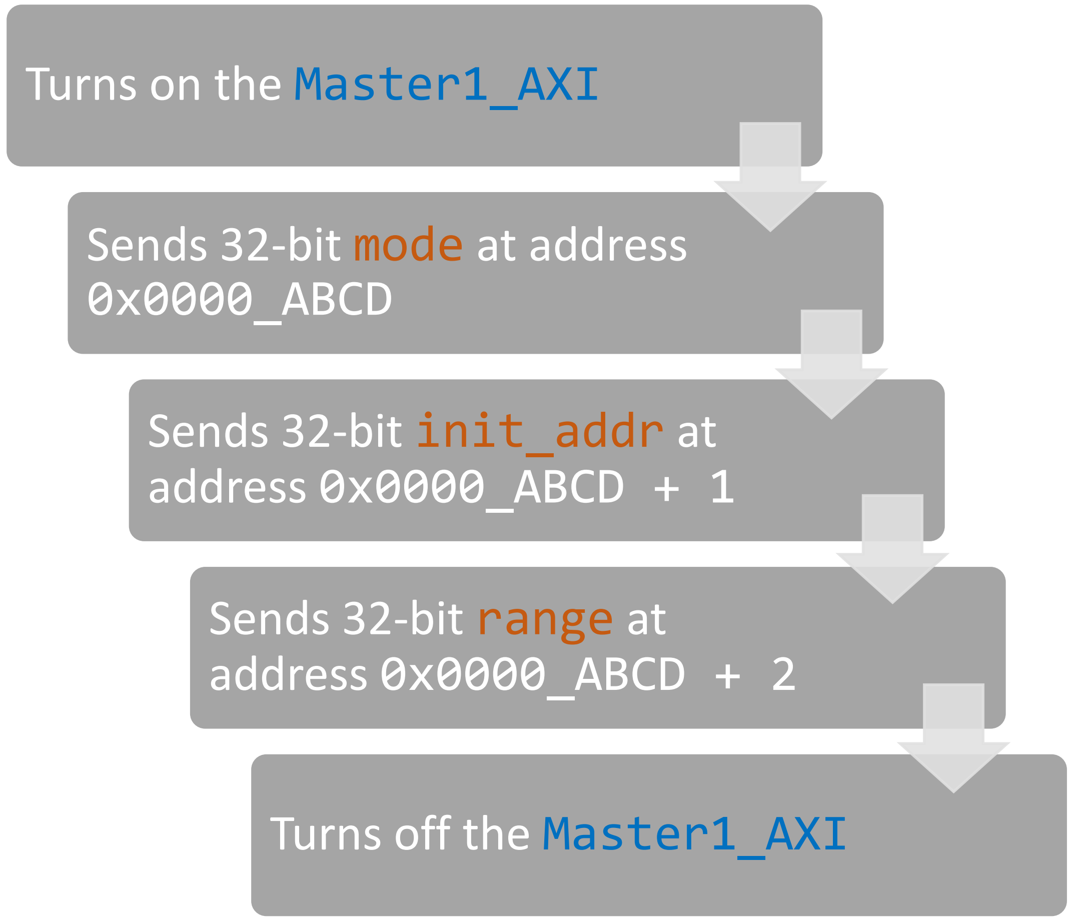
continued (ISR) ...


Status (00: ready to receive request, DMAC is also free;
01: packets are being sent to DMAC from ISR;
10: ready to receive request, but DMAC is busy)
Interrupt Handler
[1:0] status
[5:0] request
ISR
Direction (1: Peripheral wants to write;
0: Peripheral wants to read)
Data Length (00/01: Word;
10: Halfword;
11: Byte
Priority (1: don't care if DMA busy;
0: wait if DMA busy)
Peripheral ID (00: No request;
01/10/11: valid ID)

Three packets to be sent to DMAC (*via* DMA Reg) from ISR:

- mode: encapsulates the 6-bit request from the Interrupt Handler
- init_addr: the starting address of the SRAM available for the DMAC to read or write
- range: how many transaction the DMAC would do with the SRAM

With appropriate request, the ISR:


Turns on the Master1_AXI

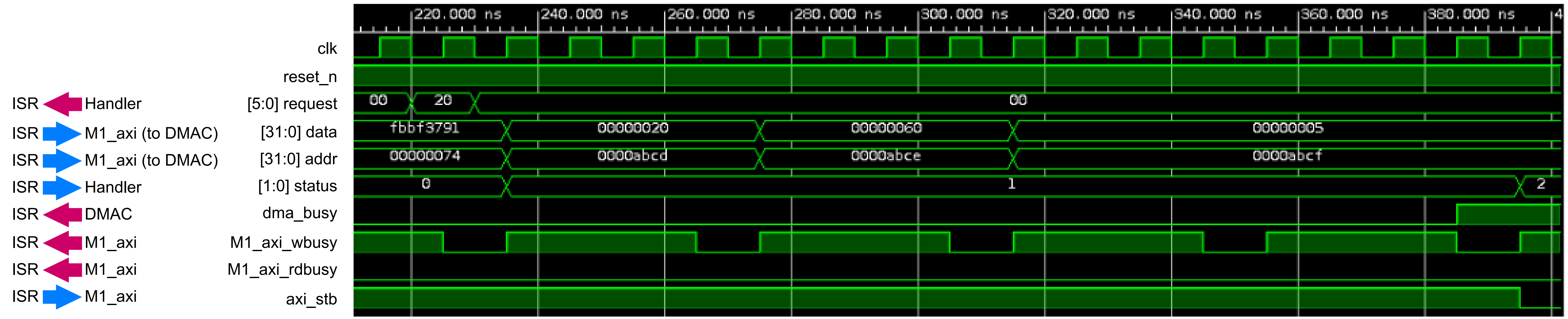
Sends 32-bit mode at address 0x0000_ABCD

Sends 32-bit init_addr at address 0x0000_ABCD + 1

Sends 32-bit range at address 0x0000_ABCD + 2

Turns off the Master1_AXI

continued (ISR) …



220 New request is coming from ID: 2 (‘b10_0000) via Handler.

225 Latch the request.

M1_AXI is free to write.

Prepare to load data and addr at next posedge clk.

235 Load mode into data.

Load 0x0000_ABCD into addr.

Send status as ‘b01.

Wait until M1_AXI is free to write.

275 Load init_addr into data.

Load 0x0000_ABCD+1 into addr.

Wait until M1_AXI is free to write.

315 Load range into data.

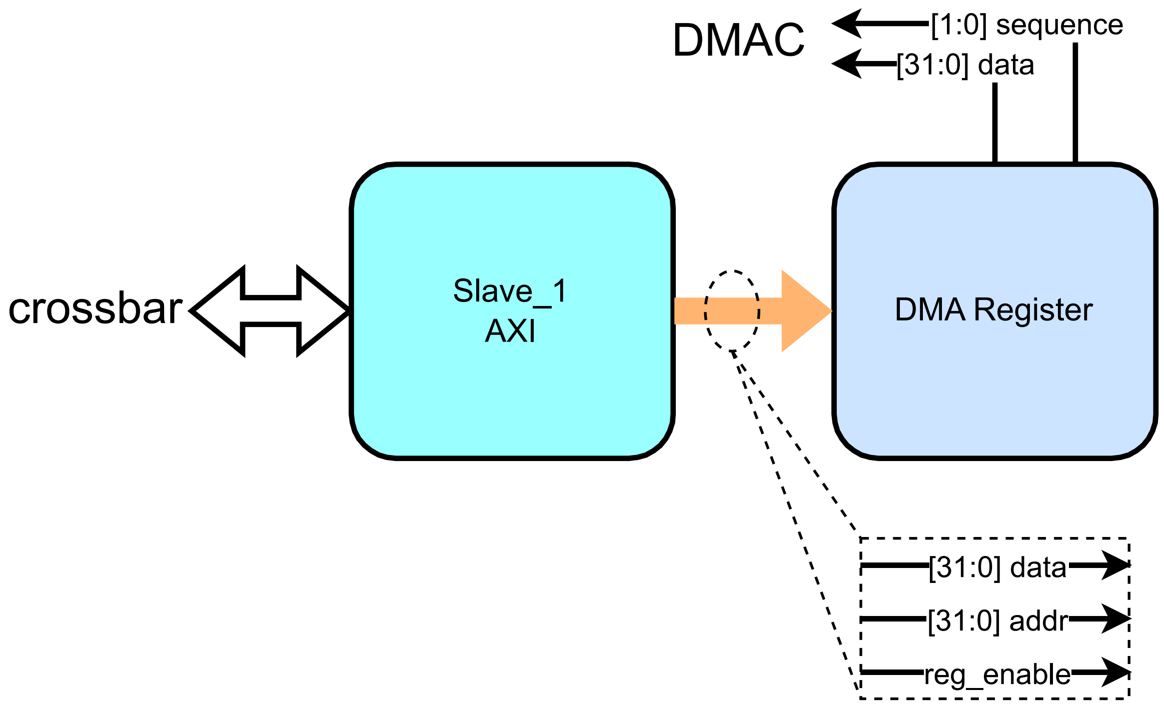
Load 0x0000_ABCD+2 into addr.

385 DMAC is now busy.

Send status as ‘b10 to Interrupt Handler at next posedge clk.

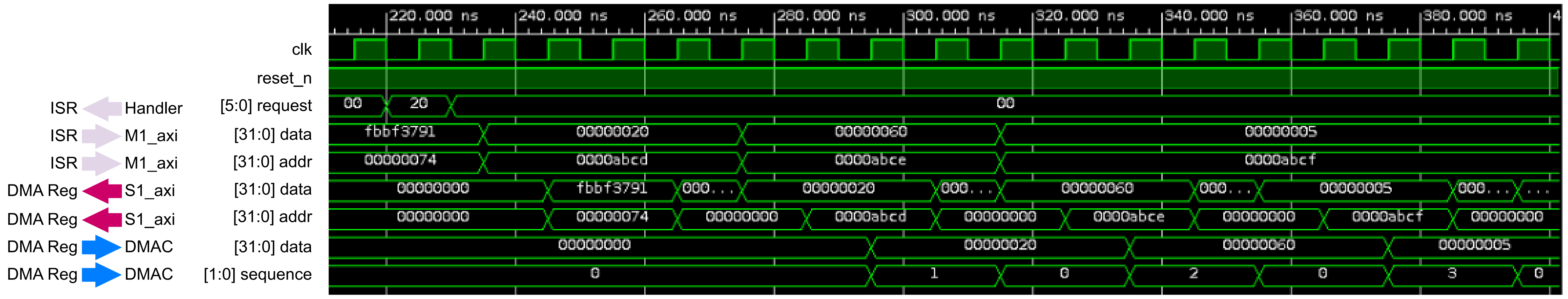
Stop accessing M1_AXI at next posedge clk.

DMA Register



- Verifies the address (`0x0000_ABCD`) from `ISR`
- Transfers `mode`, `init_addr`, `range` packets from `ISR` to `DMAC`
- Sends 2-bit `sequence` info to `DMAC` for discerning those three 32-bit packets

continued (DMA Reg) …



285 Valid addr (0x0000_ABCD) detected from S1_AXI.

295 Copy data from S1_AXI to DMAC.

Tell DMAC this is the 1st packet.

325 Valid addr (0x0000_ABCD+1) detected from S1_AXI.

335 Copy data from S1_AXI to DMAC.

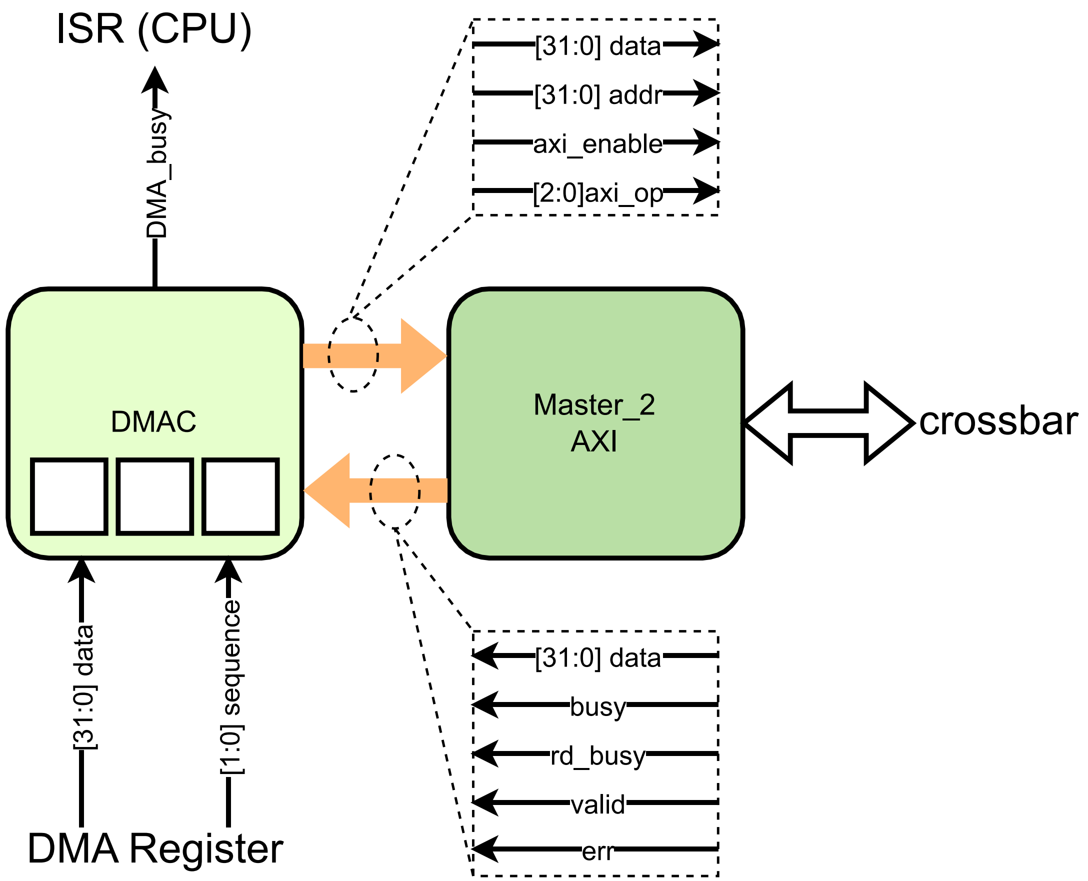
Tell DMAC this is the 2nd packet.

365 Valid addr (0x0000_ABCD+2) detected from S1_AXI.

375 Copy data from S1_AXI to DMAC.

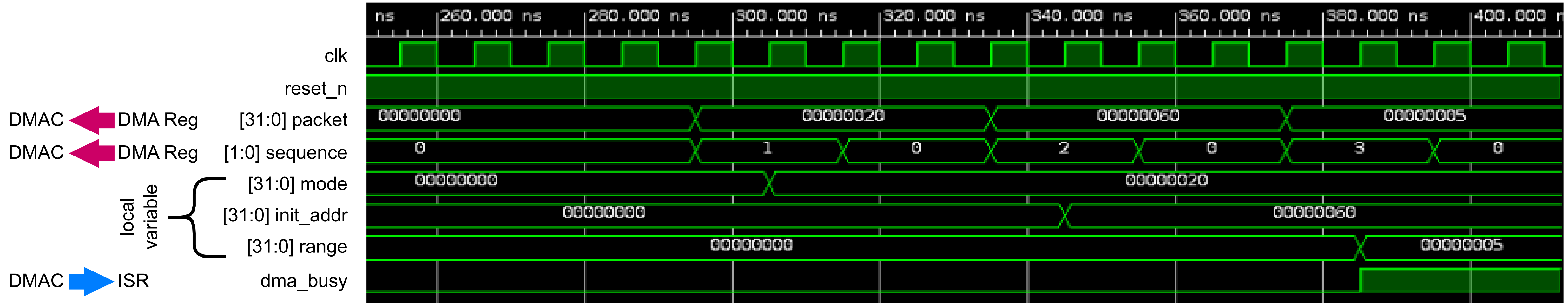
Tell DMAC this is the 3rd packet.

DMAC



- Has 3 registers to save mode, init_addr, range packets from DMA Register module
- Starts transacting with SRAM after saving all three packets
- Keeps DMA_busy HIGH during transaction with SRAM
- Continuously polls if the priority bit (within mode) is HIGH

continued (DMAC) …



295 Input sequence is ‘d1, so the packet must be mode.

305 Save the packet into mode var.

335 Input sequence is ‘d2, so the packet must be init_addr.

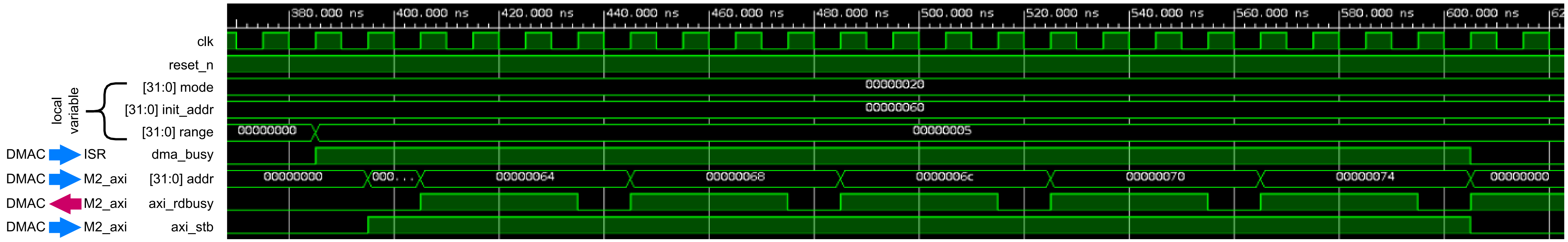
345 Save the packet into init_addr var.

375 Input sequence is ‘d3, so the packet must be range.

385 Save the packet into range var.

Assert the dma_busy high.

continued (DMAC) …



385

DMAC is ready to access SRAM. Output dma_busy is high.

M2_AXI is free to read.

Load addr 0x0000_0060 at next posedge clk.

395

M2_AXI is free to read.

Load addr 0x0000_0060+(4) at next posedge clk.

435

M2_AXI is free to read.

Load addr 0x0000_0060+(4×2) at next posedge clk.

475, 515, 555

Repeat for addr 0x0000_0060+(4×3), 0x0000_0060+(4×4), 0x0000_0060+(4×5)

595

M2_AXI is free to read.

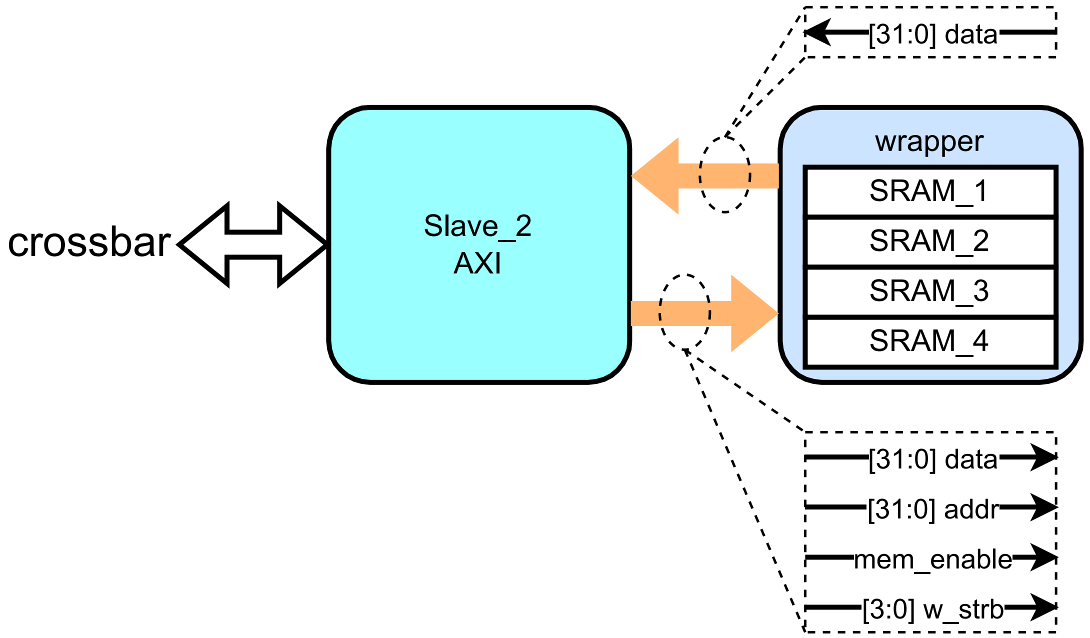
But range is exhausted.

Assert the dma_busy low at next posedge clk.

Stop accessing M2_AXI at next posedge clk.

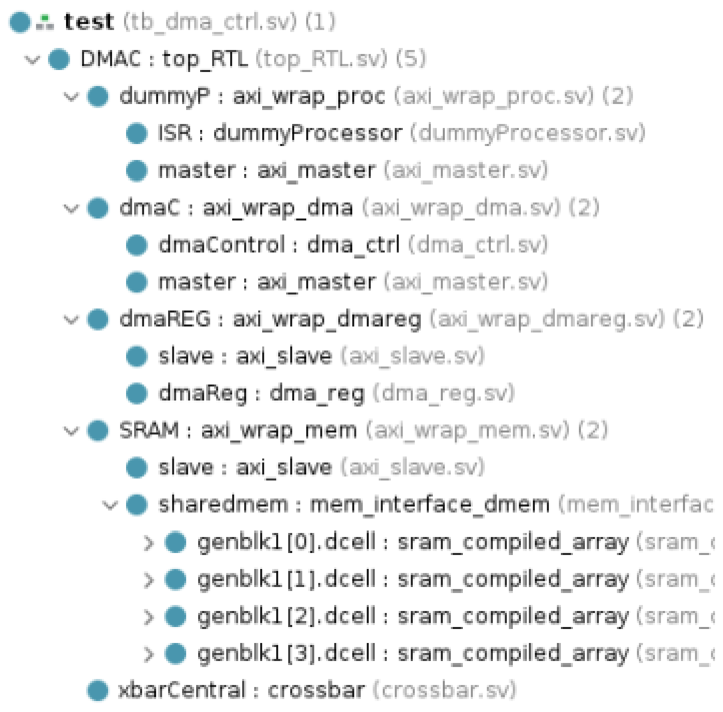
But why increment by 4 (instead of 1)?

SRAM Wrapper

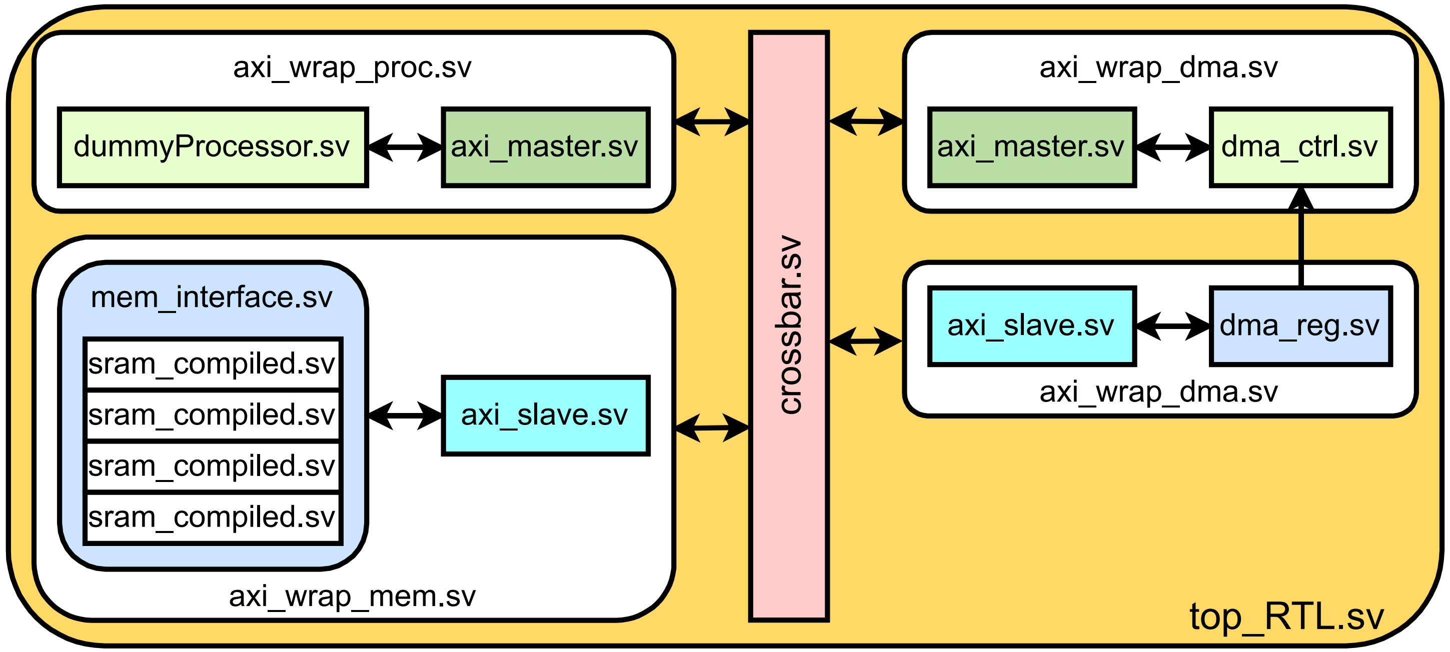


- Encapsulates 4 banks of SRAM block, each sized 16×32×8

Overall Hierarchy

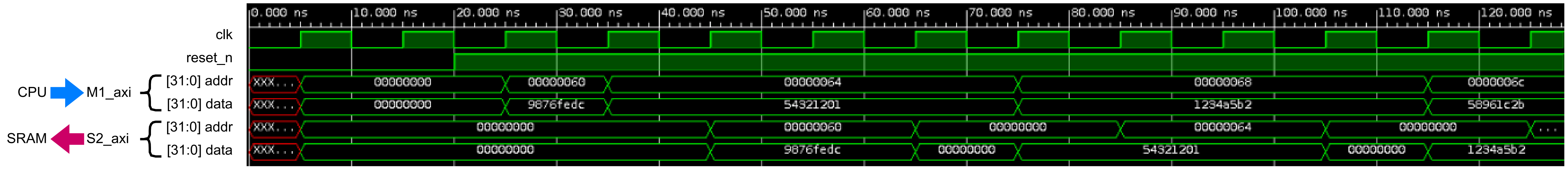


```
● test (tb_dma_ctrl.sv) (1)
 ⌄ ● DMAC : top_RTL (top_RTL.sv) (5)
    ⌄ ● dummyP : axi_wrap_proc (axi_wrap_proc.sv) (2)
         ● ISR : dummyProcessor (dummyProcessor.sv)
         ● master : axi_master (axi_master.sv)
    ⌄ ● dmaC : axi_wrap_dma (axi_wrap_dma.sv) (2)
         ● dmaControl : dma_ctrl (dma_ctrl.sv)
         ● master : axi_master (axi_master.sv)
    ⌄ ● dmaREG : axi_wrap_dmareg (axi_wrap_dmareg.sv) (2)
         ● slave : axi_slave (axi_slave.sv)
         ● dmaReg : dma_reg (dma_reg.sv)
    ⌄ ● SRAM : axi_wrap_mem (axi_wrap_mem.sv) (2)
         ● slave : axi_slave (axi_slave.sv)
       ⌄ ● sharedmem : mem_interface_dmem (mem_interfac
          › ● genblk1[0].dcell : sram_compiled_array (sram_
          › ● genblk1[1].dcell : sram_compiled_array (sram_
          › ● genblk1[2].dcell : sram_compiled_array (sram_
          › ● genblk1[3].dcell : sram_compiled_array (sram_
      ● xbarCentral : crossbar (crossbar.sv)
```

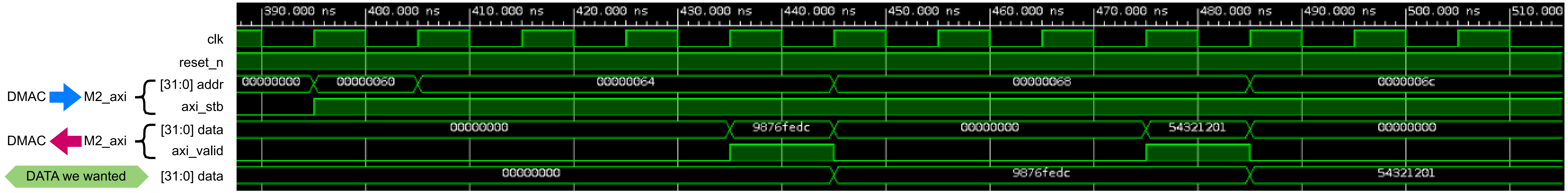


SRAM Transactions

CPU writes to SRAM



DMAC reads from SRAM



Upcoming Plans

- ➢Add cycle-stealing mode
  - We’ve only introduced burst mode.
- ➢Attach three dummy peripherals
  - with the Interrupt Handler
- ➢Complete the physical layout
  - with partitioning

Questions?
Comments?
Concerns?