Scatter Gather DMA Example

Refer to Matchlib example 18_scatter_gather

Stuart Swan
Platform Architect
Siemens EDA

16 April 2024

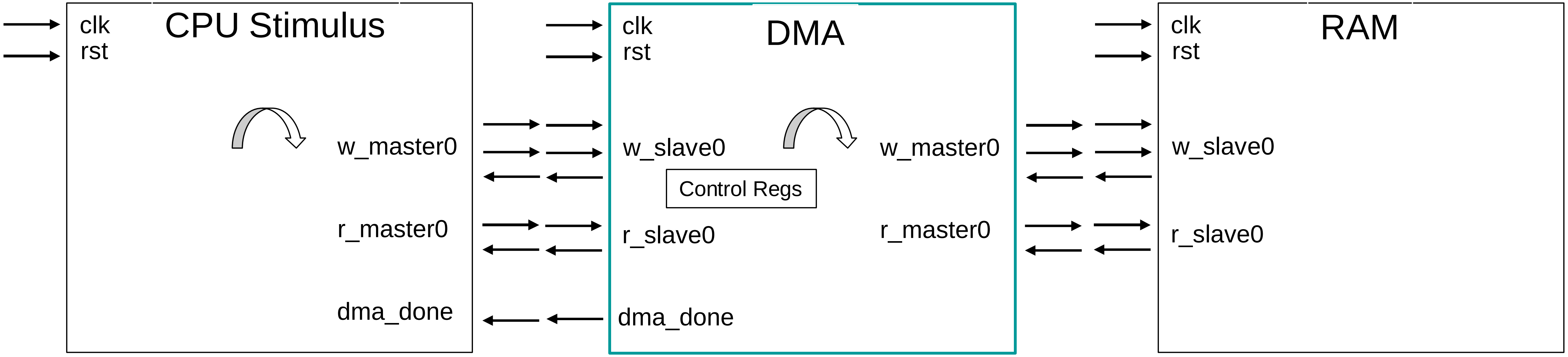
SIEMENS

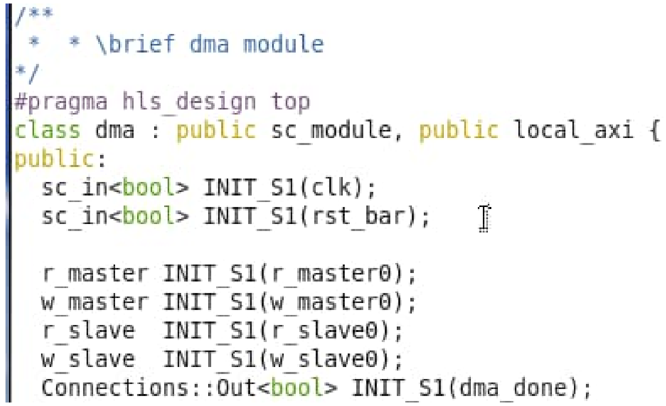
Scatter Gather DMA Introduction

- Before reading this presentation, read the “Matchlib AXI4 Interfaces” section within the Matchlib training slides in the Matchlib toolkit doc directory.

- DMA modules are common in SOCs to offload data movement tasks from other blocks
- The Scatter Gather DMA is a simple but representative example of common SOC blocks that use bus fabrics and which are not only datapath blocks but also include control
- Also represents typical DV aspects for SOC blocks
- The DMA fully uses the AXI4 features explained previously such as automatic burst segmentation
- The scatter gather DMA adds features to the simple DMA block presented previously
- Same top level interfaces as simple DMA
- Same RAM block

Scatter Gather DMA block diagram



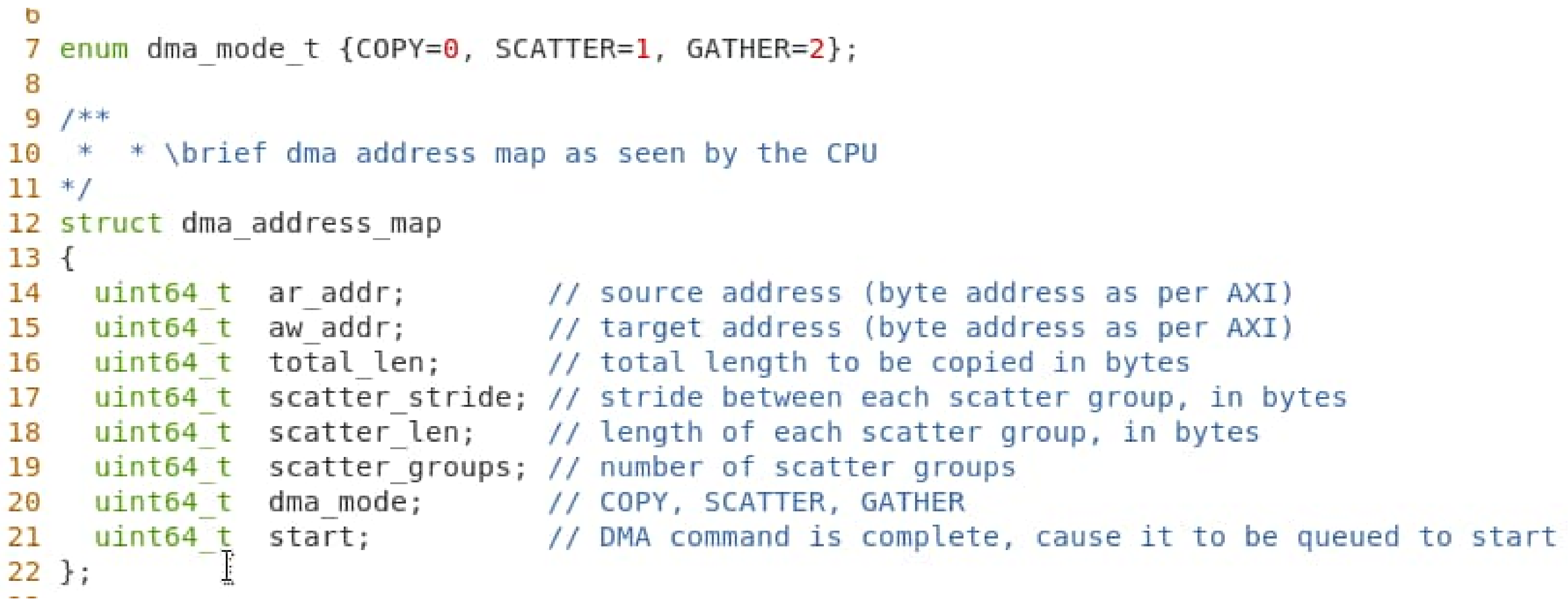


```
/**
 *   * \brief dma module
*/
#pragma hls_design top
class dma : public sc_module, public local_axi {
public:
  sc_in<bool> INIT_S1(clk);
  sc_in<bool> INIT_S1(rst_bar);

  r_master INIT_S1(r_master0);
  w_master INIT_S1(w_master0);
  r_slave  INIT_S1(r_slave0);
  w_slave  INIT_S1(w_slave0);
  Connections::Out<bool> INIT_S1(dma_done);
```

□ = top level of design

Scatter Gather Slave Address Map

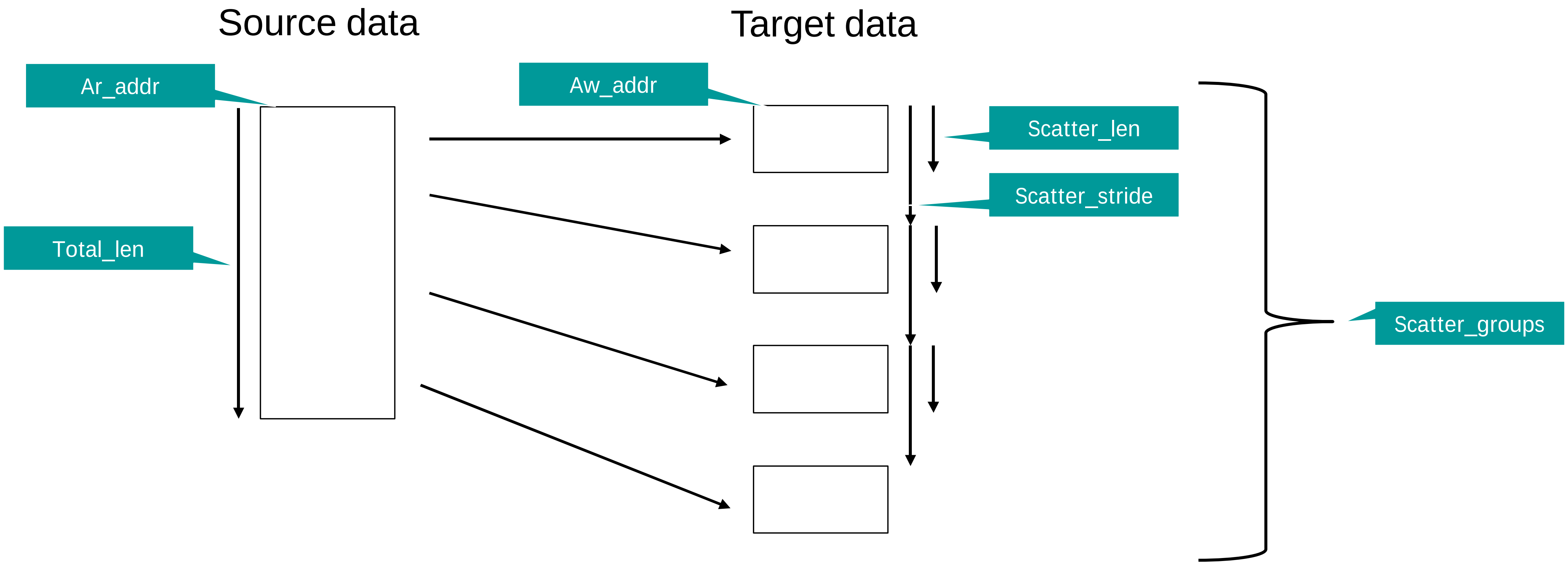


```
enum dma_mode_t {COPY=0, SCATTER=1, GATHER=2};

/**
 *  * \brief dma address map as seen by the CPU
*/
struct dma_address_map
{
  uint64_t  ar_addr;        // source address (byte address as per AXI)
  uint64_t  aw_addr;        // target address (byte address as per AXI)
  uint64_t  total_len;      // total length to be copied in bytes
  uint64_t  scatter_stride; // stride between each scatter group, in bytes
  uint64_t  scatter_len;    // length of each scatter group, in bytes
  uint64_t  scatter_groups; // number of scatter groups
  uint64_t  dma_mode;       // COPY, SCATTER, GATHER
  uint64_t  start;          // DMA command is complete, cause it to be queued to start
};
```

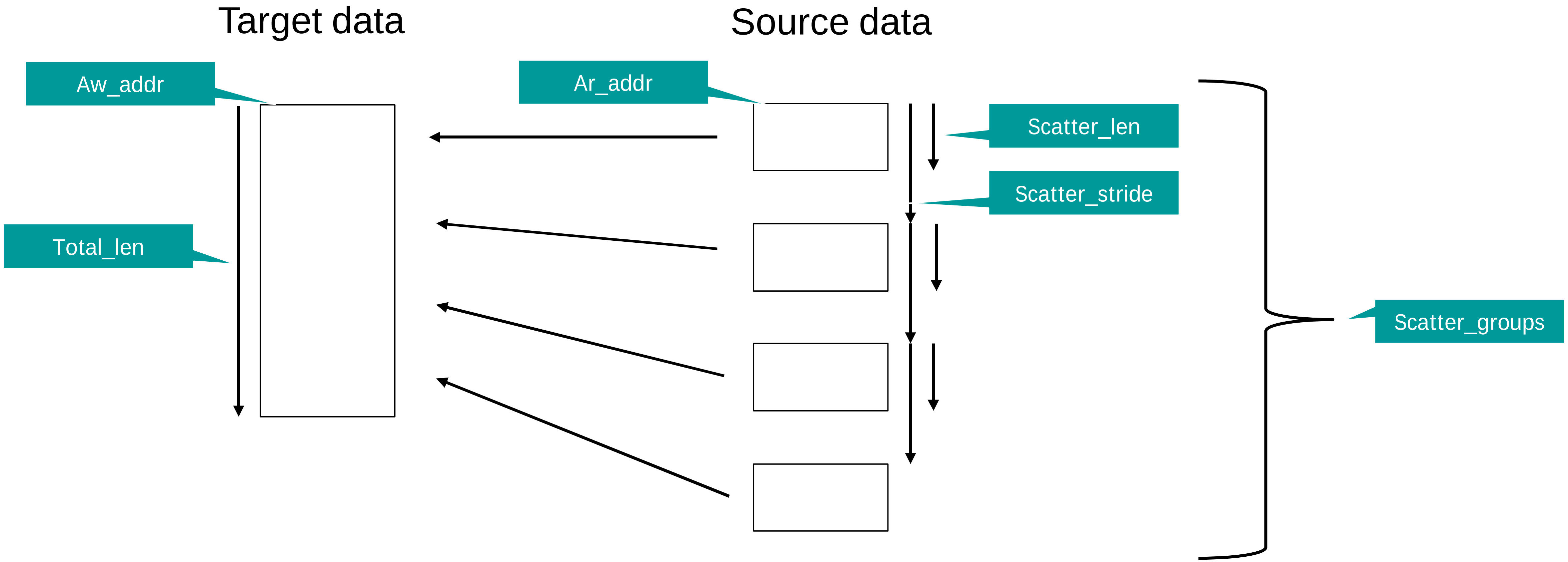
Note: In previous DMA (example 08*), total_len was in beats, but now it is a byte length

Scatter Operation



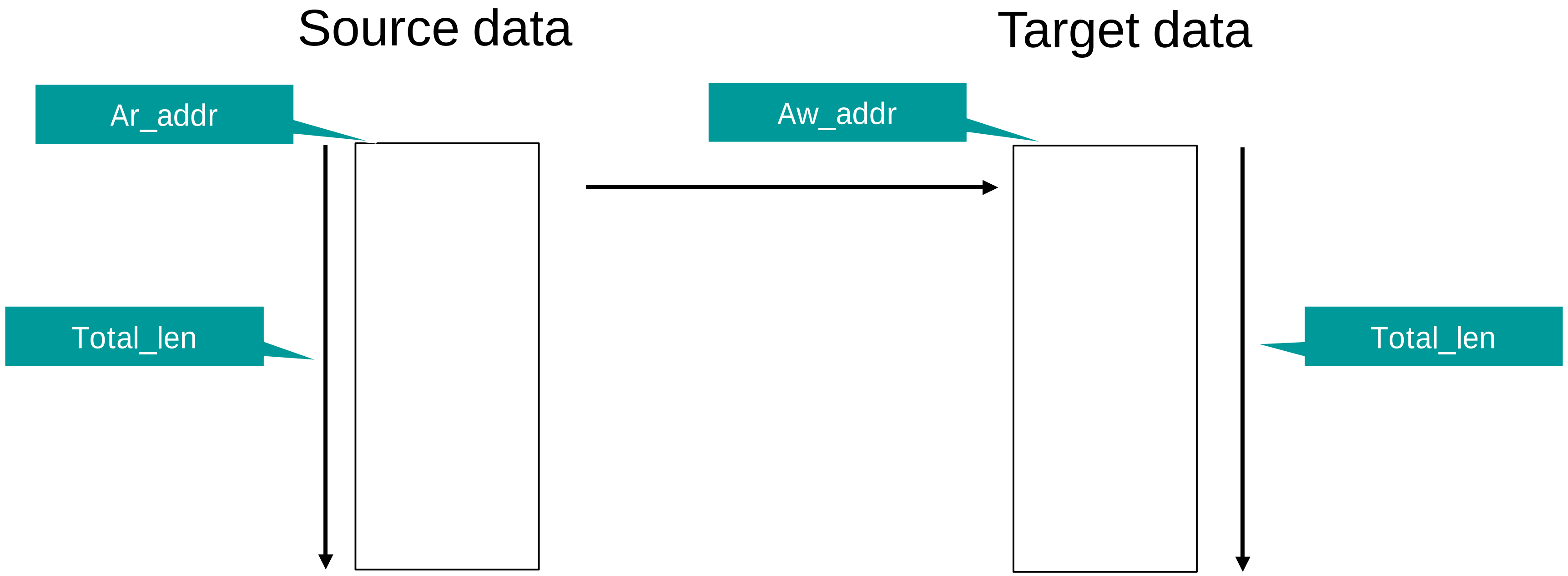
All addresses are byte addresses
All lengths are byte lengths
total_len = scatter_groups * scatter_len

Gather Operation



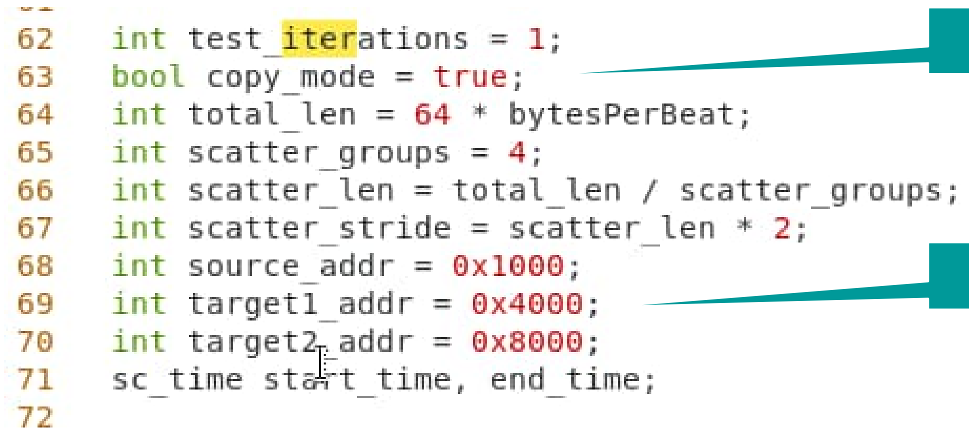
All addresses are byte addresses
All lengths are byte lengths
total_len = scatter_groups * scatter_len

Copy Operation



All addresses are byte addresses
All lengths are byte lengths
For copy operation, scatter_len, scatter_stride, scatter_groups are ignored

SC Testbench

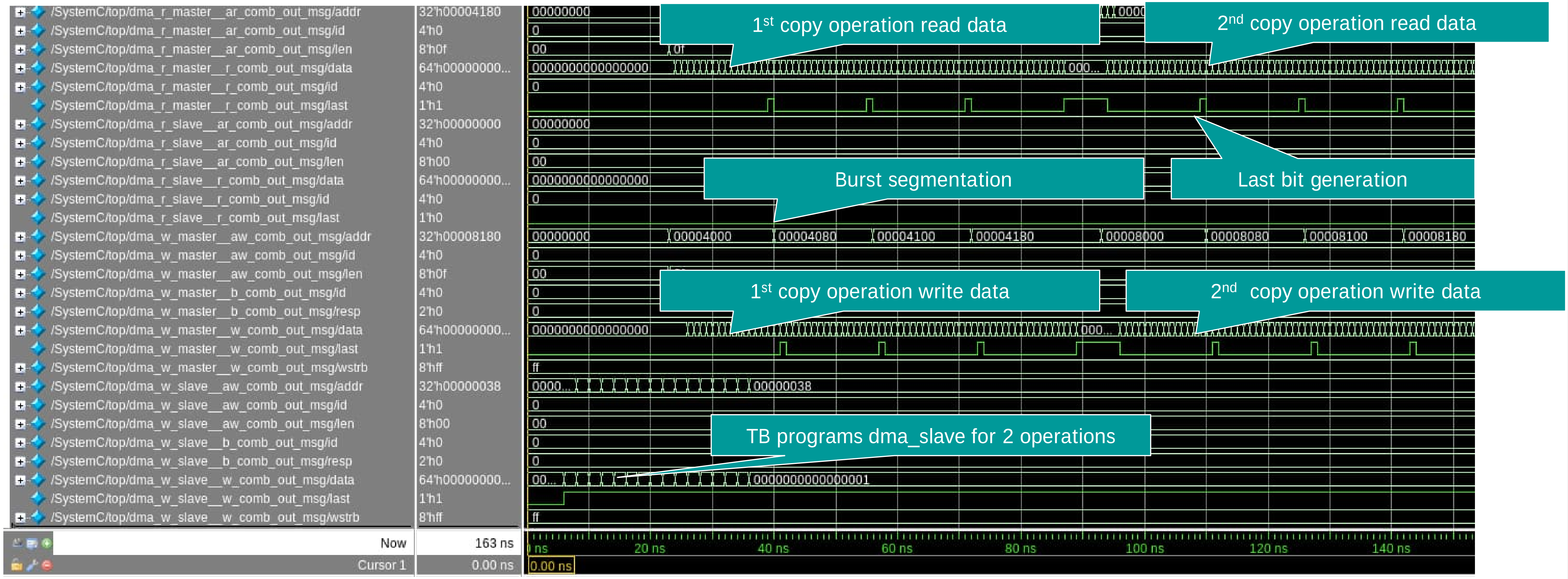


```
int test_iterations = 1;
bool copy_mode = true;
int total_len = 64 * bytesPerBeat;
int scatter_groups = 4;
int scatter_len = total_len / scatter_groups;
int scatter_stride = scatter_len * 2;
int source_addr = 0x1000;
int target1_addr = 0x4000;
int target2_addr = 0x8000;
sc_time start_time, end_time;
```

If true, each iteration is a copy then a copy, else it is a scatter then a gather

Target address for first operation, source address for second operation

Copy_mode = true, test_iterations = 1



```
/SystemC/top/dma_r_master__ar_comb_out_msg/addr   32'h00004180
/SystemC/top/dma_r_master__ar_comb_out_msg/id     4'h0
/SystemC/top/dma_r_master__ar_comb_out_msg/len    8'h0f
/SystemC/top/dma_r_master__r_comb_out_msg/data    64'h00000000...
/SystemC/top/dma_r_master__r_comb_out_msg/id      4'h0
/SystemC/top/dma_r_master__r_comb_out_msg/last    1'h1
/SystemC/top/dma_r_slave__ar_comb_out_msg/addr    32'h00000000
/SystemC/top/dma_r_slave__ar_comb_out_msg/id      4'h0
/SystemC/top/dma_r_slave__ar_comb_out_msg/len     8'h00
/SystemC/top/dma_r_slave__r_comb_out_msg/data     64'h00000000...
/SystemC/top/dma_r_slave__r_comb_out_msg/id       4'h0
/SystemC/top/dma_r_slave__r_comb_out_msg/last     1'h0
/SystemC/top/dma_w_master__aw_comb_out_msg/addr   32'h00008180
/SystemC/top/dma_w_master__aw_comb_out_msg/id     4'h0
/SystemC/top/dma_w_master__aw_comb_out_msg/len    8'h0f
/SystemC/top/dma_w_master__b_comb_out_msg/id      4'h0
/SystemC/top/dma_w_master__b_comb_out_msg/resp    2'h0
/SystemC/top/dma_w_master__w_comb_out_msg/data    64'h00000000...
/SystemC/top/dma_w_master__w_comb_out_msg/last    1'h1
/SystemC/top/dma_w_master__w_comb_out_msg/wstrb   8'hff
/SystemC/top/dma_w_slave__aw_comb_out_msg/addr    32'h00000038
/SystemC/top/dma_w_slave__aw_comb_out_msg/id      4'h0
/SystemC/top/dma_w_slave__aw_comb_out_msg/len     8'h00
/SystemC/top/dma_w_slave__b_comb_out_msg/id       4'h0
/SystemC/top/dma_w_slave__b_comb_out_msg/resp     2'h0
/SystemC/top/dma_w_slave__w_comb_out_msg/data     64'h00000000...
/SystemC/top/dma_w_slave__w_comb_out_msg/last     1'h1
/SystemC/top/dma_w_slave__w_comb_out_msg/wstrb    8'hff
Now        163 ns
Cursor 1   0.00 ns
```

1st copy operation read data

2nd copy operation read data

Burst segmentation

Last bit generation

1st copy operation write data

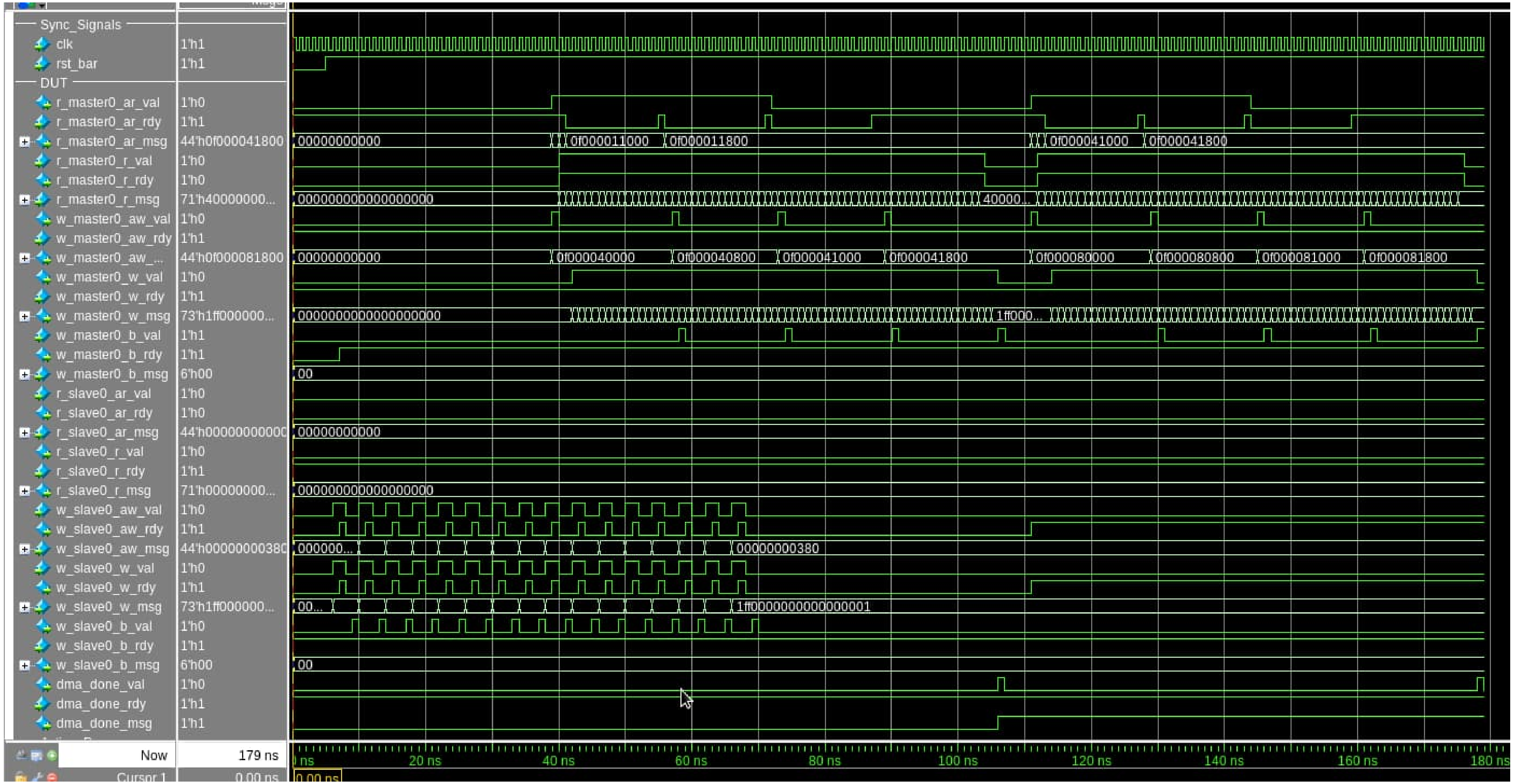
2nd copy operation write data

TB programs dma_slave for 2 operations

Note: SystemC Waveforms, (not RTL)

Makefile sets segmentation size to 16 rather than 256 so easier to see in waveforms

Same scenario, but in RTL



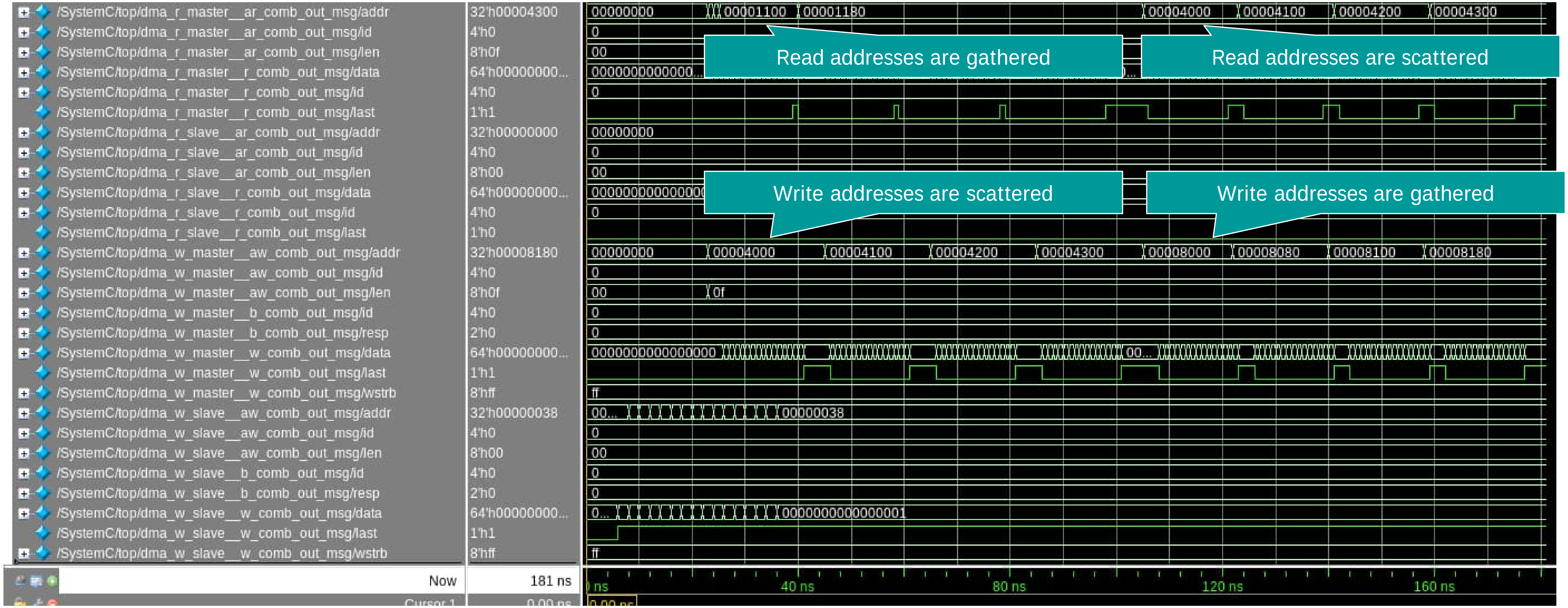
Sync_Signals

clk	1'h1
rst_bar	1'h1

DUT

r_master0_ar_val	1'h0
r_master0_ar_rdy	1'h1
r_master0_ar_msg	44'h0f000041800
r_master0_r_val	1'h0
r_master0_r_rdy	1'h0
r_master0_r_msg	71'h40000000...
w_master0_aw_val	1'h0
w_master0_aw_rdy	1'h1
w_master0_aw_...	44'h0f000081800
w_master0_w_val	1'h0
w_master0_w_rdy	1'h1
w_master0_w_msg	73'h1ff000000...
w_master0_b_val	1'h1
w_master0_b_rdy	1'h1
w_master0_b_msg	6'h00
r_slave0_ar_val	1'h0
r_slave0_ar_rdy	1'h0
r_slave0_ar_msg	44'h00000000000
r_slave0_r_val	1'h0
r_slave0_r_rdy	1'h1
r_slave0_r_msg	71'h00000000...
w_slave0_aw_val	1'h0
w_slave0_aw_rdy	1'h1
w_slave0_aw_msg	44'h00000000380
w_slave0_w_val	1'h0
w_slave0_w_rdy	1'h1
w_slave0_w_msg	73'h1ff000000...
w_slave0_b_val	1'h0
w_slave0_b_rdy	1'h1
w_slave0_b_msg	6'h00
dma_done_val	1'h0
dma_done_rdy	1'h1
dma_done_msg	1'h1
Now	179 ns
Cursor 1	0.00 ns

Copy_mode = false, test_iterations = 1



/SystemC/top/dma_r_master__ar_comb_out_msg/addr	32'h00004300
/SystemC/top/dma_r_master__ar_comb_out_msg/id	4'h0
/SystemC/top/dma_r_master__ar_comb_out_msg/len	8'h0f
/SystemC/top/dma_r_master__r_comb_out_msg/data	64'h00000000...
/SystemC/top/dma_r_master__r_comb_out_msg/id	4'h0
/SystemC/top/dma_r_master__r_comb_out_msg/last	1'h1
/SystemC/top/dma_r_slave__ar_comb_out_msg/addr	32'h00000000
/SystemC/top/dma_r_slave__ar_comb_out_msg/id	4'h0
/SystemC/top/dma_r_slave__ar_comb_out_msg/len	8'h00
/SystemC/top/dma_r_slave__r_comb_out_msg/data	64'h00000000...
/SystemC/top/dma_r_slave__r_comb_out_msg/id	4'h0
/SystemC/top/dma_r_slave__r_comb_out_msg/last	1'h0
/SystemC/top/dma_w_master__aw_comb_out_msg/addr	32'h00008180
/SystemC/top/dma_w_master__aw_comb_out_msg/id	4'h0
/SystemC/top/dma_w_master__aw_comb_out_msg/len	8'h0f
/SystemC/top/dma_w_master__b_comb_out_msg/id	4'h0
/SystemC/top/dma_w_master__b_comb_out_msg/resp	2'h0
/SystemC/top/dma_w_master__w_comb_out_msg/data	64'h00000000...
/SystemC/top/dma_w_master__w_comb_out_msg/last	1'h1
/SystemC/top/dma_w_master__w_comb_out_msg/wstrb	8'hff
/SystemC/top/dma_w_slave__aw_comb_out_msg/addr	32'h00000038
/SystemC/top/dma_w_slave__aw_comb_out_msg/id	4'h0
/SystemC/top/dma_w_slave__aw_comb_out_msg/len	8'h00
/SystemC/top/dma_w_slave__b_comb_out_msg/id	4'h0
/SystemC/top/dma_w_slave__b_comb_out_msg/resp	2'h0
/SystemC/top/dma_w_slave__w_comb_out_msg/data	64'h00000000...
/SystemC/top/dma_w_slave__w_comb_out_msg/last	1'h1
/SystemC/top/dma_w_slave__w_comb_out_msg/wstrb	8'hff
Now	181 ns
Cursor 1	0.00 ns

Read addresses are gathered

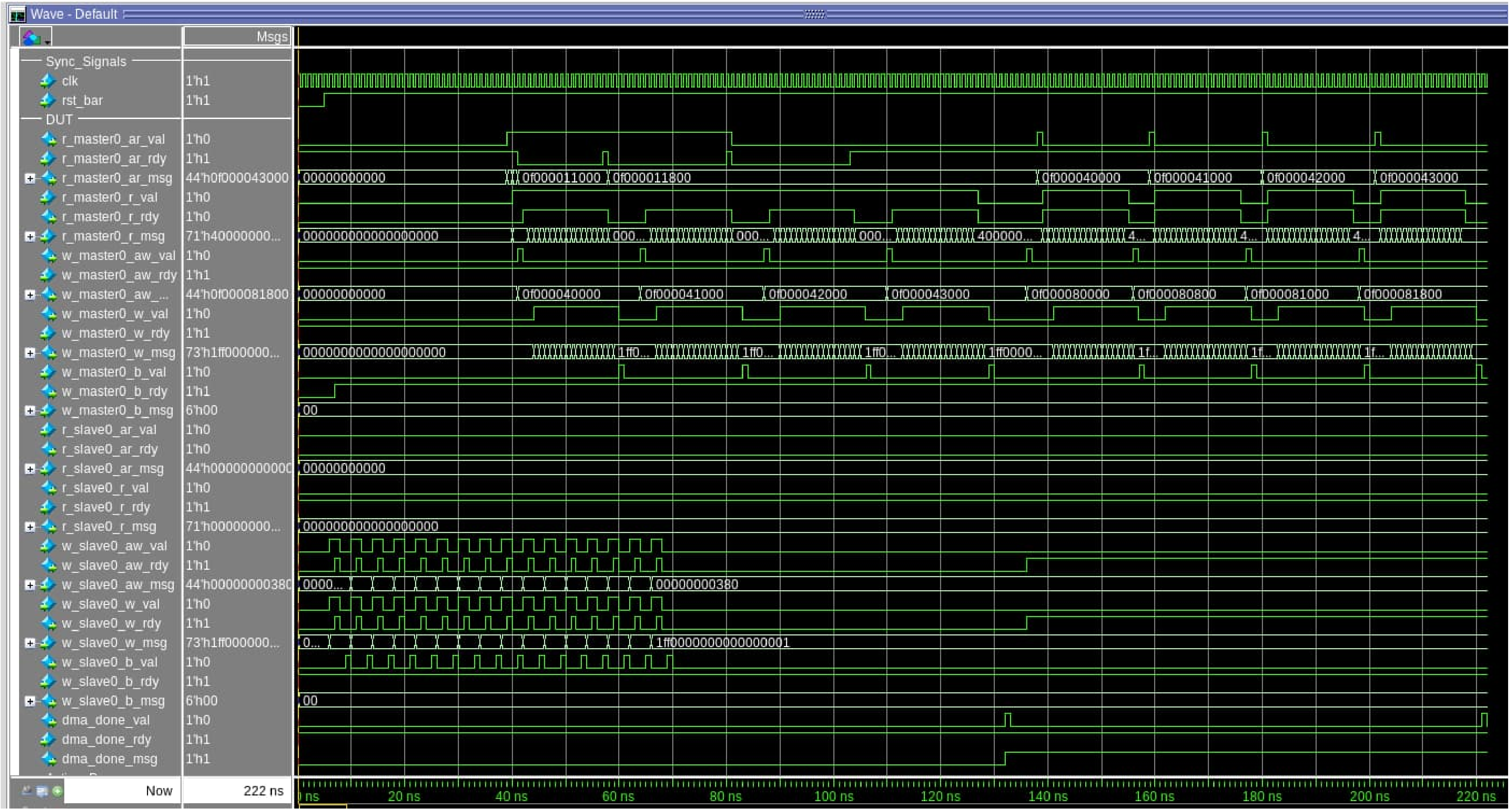
Read addresses are scattered

Write addresses are scattered

Write addresses are gathered

Note: SystemC Waveforms, (not RTL)

Same scenario, but in RTL



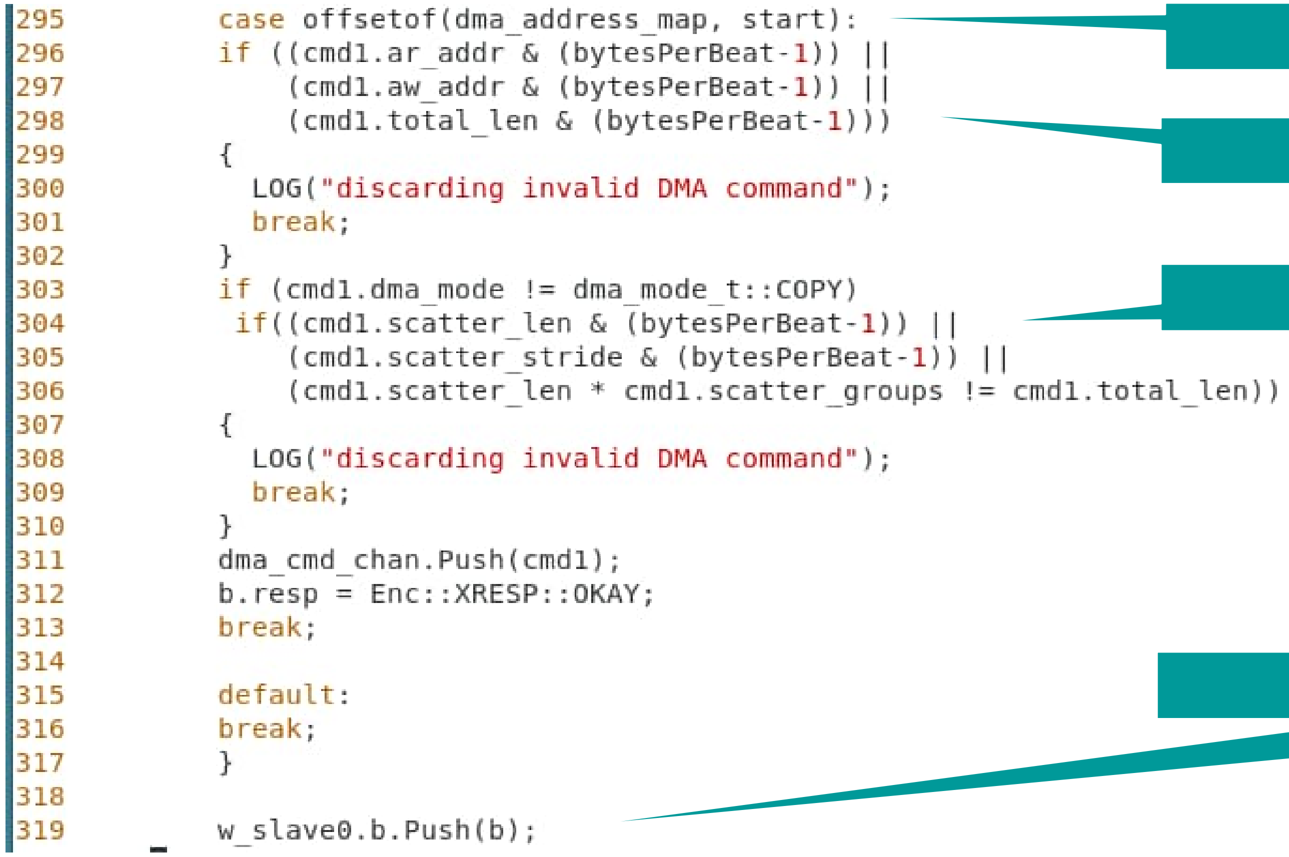
Wave - Default

	Msgs
Sync_Signals	
clk	1'h1
rst_bar	1'h1
DUT	
r_master0_ar_val	1'h0
r_master0_ar_rdy	1'h1
r_master0_ar_msg	44'h0f000043000
r_master0_r_val	1'h0
r_master0_r_rdy	1'h0
r_master0_r_msg	71'h40000000...
w_master0_aw_val	1'h0
w_master0_aw_rdy	1'h1
w_master0_aw_...	44'h0f000081800
w_master0_w_val	1'h0
w_master0_w_rdy	1'h1
w_master0_w_msg	73'h1ff000000...
w_master0_b_val	1'h0
w_master0_b_rdy	1'h1
w_master0_b_msg	6'h00
r_slave0_ar_val	1'h0
r_slave0_ar_rdy	1'h0
r_slave0_ar_msg	44'h00000000000
r_slave0_r_val	1'h0
r_slave0_r_rdy	1'h1
r_slave0_r_msg	71'h00000000...
w_slave0_aw_val	1'h0
w_slave0_aw_rdy	1'h1
w_slave0_aw_msg	44'h00000000380
w_slave0_w_val	1'h0
w_slave0_w_rdy	1'h1
w_slave0_w_msg	73'h1ff000000...
w_slave0_b_val	1'h0
w_slave0_b_rdy	1'h1
w_slave0_b_msg	6'h00
dma_done_val	1'h0
dma_done_rdy	1'h1
dma_done_msg	1'h1
Now	222 ns

Keeping things simple..

To simplify both the DUT and the TB, the DUT enforces that all addresses and lengths are aligned to bus datawidth boundaries

When TB writes to “start” CSR, checks are enforced and a AXI4 error response is returned if there any errors.



```
case offsetof(dma_address_map, start):
if ((cmd1.ar_addr & (bytesPerBeat-1)) ||
    (cmd1.aw_addr & (bytesPerBeat-1)) ||
    (cmd1.total_len & (bytesPerBeat-1)))
{
  LOG("discarding invalid DMA command");
  break;
}
if (cmd1.dma_mode != dma_mode_t::COPY)
 if((cmd1.scatter_len & (bytesPerBeat-1)) ||
    (cmd1.scatter_stride & (bytesPerBeat-1)) ||
    (cmd1.scatter_len * cmd1.scatter_groups != cmd1.total_len))
{
  LOG("discarding invalid DMA command");
  break;
}
dma_cmd_chan.Push(cmd1);
b.resp = Enc::XRESP::OKAY;
break;

default:
break;
}

w_slave0.b.Push(b);
```

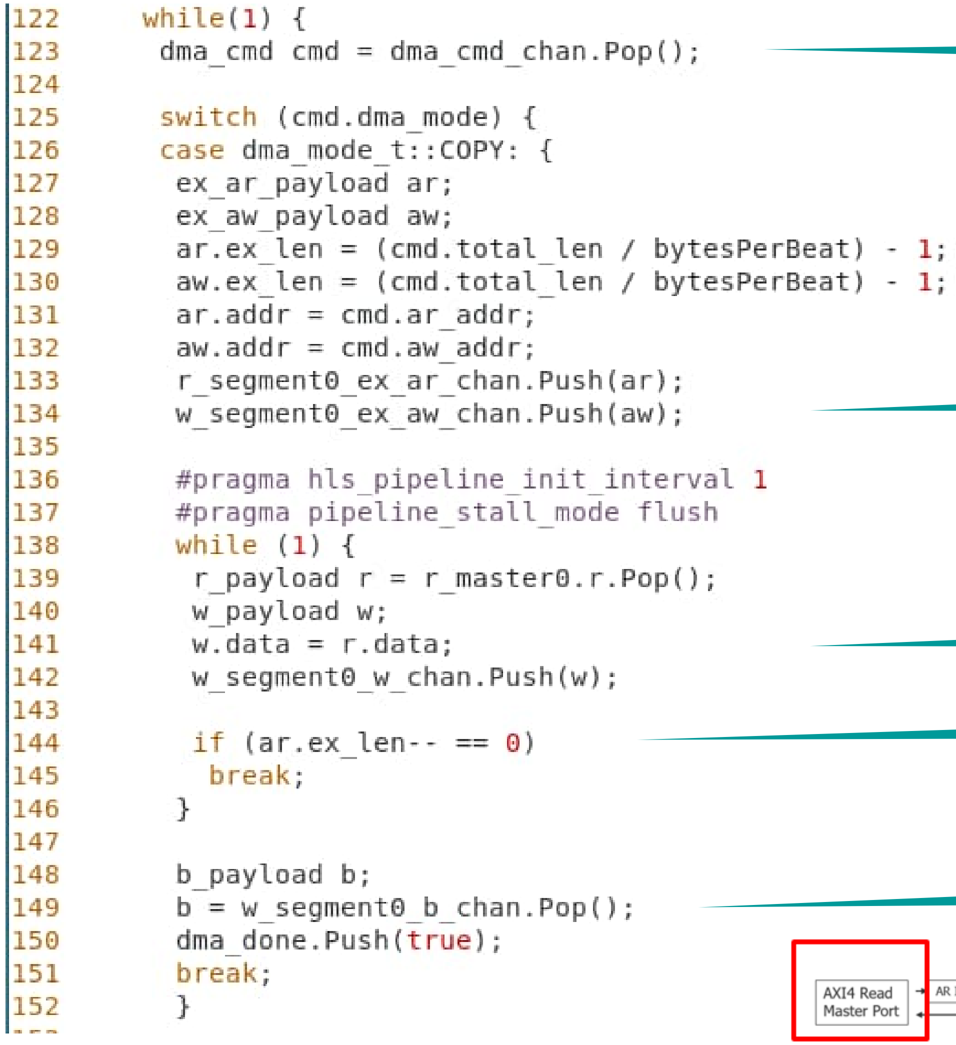
When TB writes to “start” CSR

Make sure addresses and lengths are aligned

Do additional checks if not in COPY mode

Send AXI4 error code to TB if any checks fail

COPY implementation in DMA



```
while(1) {
 dma_cmd cmd = dma_cmd_chan.Pop();

 switch (cmd.dma_mode) {
 case dma_mode_t::COPY: {
  ex_ar_payload ar;
  ex_aw_payload aw;
  ar.ex_len = (cmd.total_len / bytesPerBeat) - 1;
  aw.ex_len = (cmd.total_len / bytesPerBeat) - 1;
  ar.addr = cmd.ar_addr;
  aw.addr = cmd.aw_addr;
  r_segment0_ex_ar_chan.Push(ar);
  w_segment0_ex_aw_chan.Push(aw);

  #pragma hls_pipeline_init_interval 1
  #pragma pipeline_stall_mode flush
  while (1) {
   r_payload r = r_master0.r.Pop();
   w_payload w;
   w.data = r.data;
   w_segment0_w_chan.Push(w);

   if (ar.ex_len-- == 0)
    break;
  }

  b_payload b;
  b = w_segment0_b_chan.Pop();
  dma_done.Push(true);
  break;
  }
```

Pop next DMA command

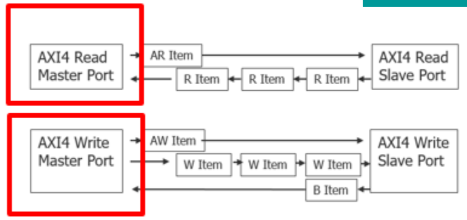
Convert byte length to beat length (AXI4 beat length encoding)

Push out AR and AW bursts

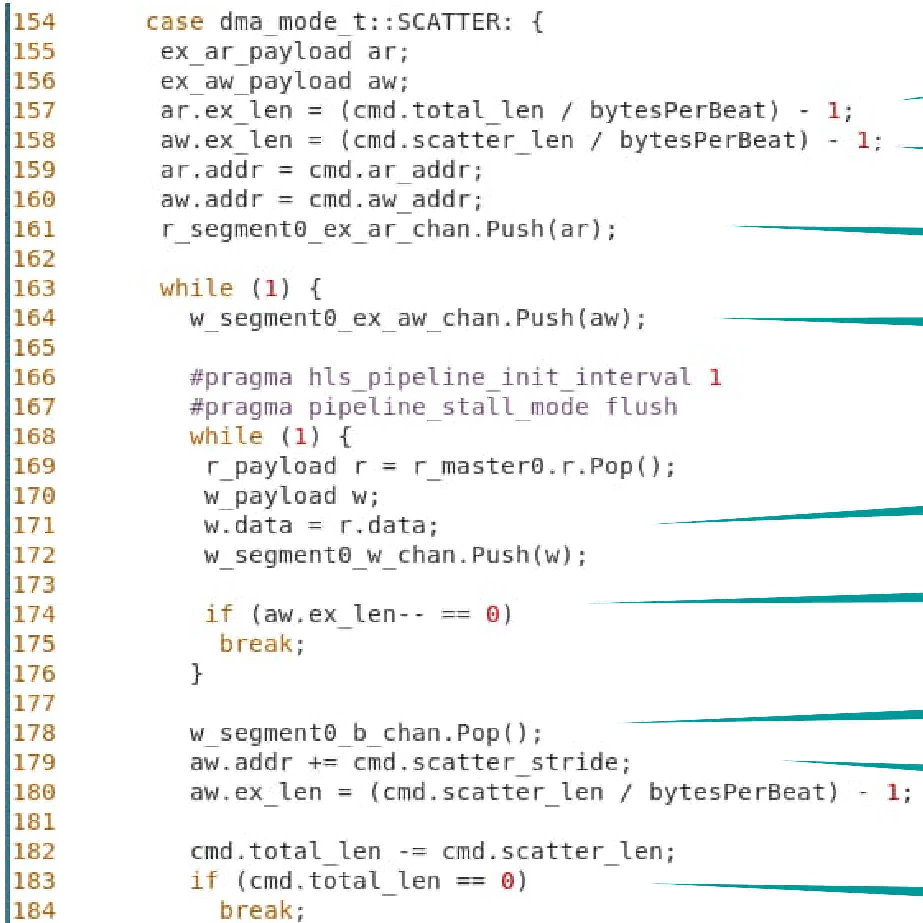
Copy data from R to W

We are done when length is currently 0 (AXI4 beat encoding)

Pop write response from b channel



SCATTER implementation in DMA



```
    case dma_mode_t::SCATTER: {
     ex_ar_payload ar;
     ex_aw_payload aw;
     ar.ex_len = (cmd.total_len / bytesPerBeat) - 1;
     aw.ex_len = (cmd.scatter_len / bytesPerBeat) - 1;
     ar.addr = cmd.ar_addr;
     aw.addr = cmd.aw_addr;
     r_segment0_ex_ar_chan.Push(ar);

     while (1) {
       w_segment0_ex_aw_chan.Push(aw);

       #pragma hls_pipeline_init_interval 1
       #pragma pipeline_stall_mode flush
       while (1) {
        r_payload r = r_master0.r.Pop();
        w_payload w;
        w.data = r.data;
        w_segment0_w_chan.Push(w);

        if (aw.ex_len-- == 0)
          break;
       }

       w_segment0_b_chan.Pop();
       aw.addr += cmd.scatter_stride;
       aw.ex_len = (cmd.scatter_len / bytesPerBeat) - 1;

       cmd.total_len -= cmd.scatter_len;
       if (cmd.total_len == 0)
         break;
     }

     dma_done.Push(true);
     break;
    }
```

Convert byte length to beat length (AXI4 beat length encoding)

aw.ex_len is computed based on scatter_len, not total_len

Start the AR burst, but not the AW yet

Start the AW burst for current region in the scatter groups

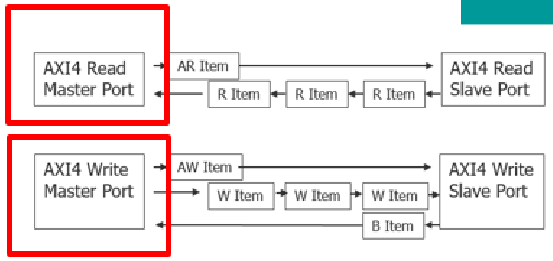
Copy data from R to W

We are done when length is currently 0 (AXI4 beat encoding)

Need to do b.Pop for every Push(aw)

Increment aw.addr for next scatter group region

We are done when the remaining total_len is zero



Homework #1 - GATHER implementation in DMA

Steps:

1. Edit testbench.cpp and change copy_mode to be false
2. Edit scatter_gather.h and write the code for GATHER
    - This is strictly confined to the GATHER branch of the case statement.
3. Compile and run your SC code and make sure the TB self-check passes
4. View SC and RTL waveforms – see README file in same dir.

Homework #2 – Optimize the beat rate

- For copy_mode = false and test_iterations = 10, make small modifications to model and synthesis directives to optimize throughput (as reported in log output as "beat rate").
- Measure in both SC sim and RTL sim.