Altera's Avalon Communication Fabric

Stephen A. Edwards

Columbia University

Spring 2022

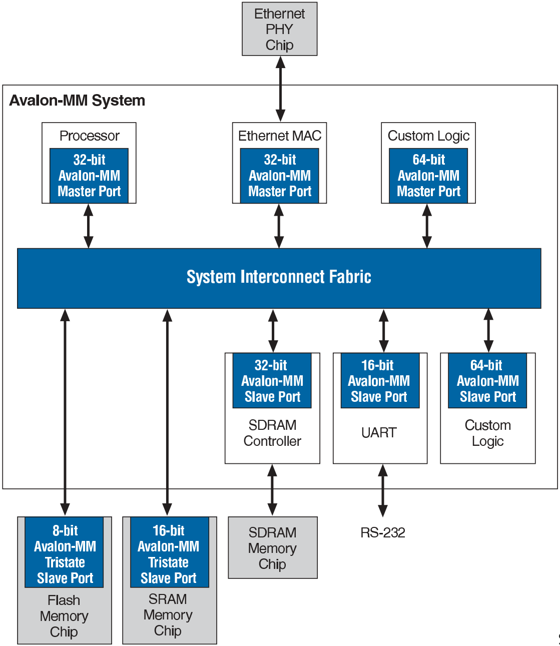
Altera’s Avalon Bus

Something like “PCI on a chip”

Described in Altera’s *Avalon Memory-Mapped Interface Specification* document.

Protocol defined between peripherals and the “bus” (actually a fairly complicated circuit).

Intended System Architecture



Source: Altera

Masters and Slaves

Most bus protocols draw a distinction between

Masters: Can initiate a transaction, specify an address, etc. E.g., the Nios II processor

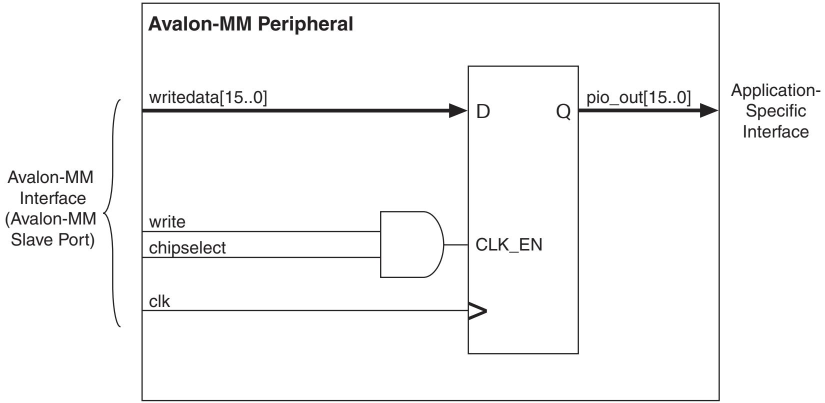
Slaves: Respond to requests from masters, can generate return data. E.g., a video controller

Most peripherals are slaves.

Masters speak a more complex protocol

Bus arbiter decides which master gains control

The Simplest Slave Peripheral



Basically, “latch when I’m selected and written to.”

Slave Signals

For a 16-bit connection that spans 32 halfwords,

Slave	$\leftarrow$	clk		**Avalon**
	$\leftarrow$	reset		
	$\leftarrow$	chipselect		
	$\Leftarrow$	address[4:0]		
	$\leftarrow$	read		
	$\leftarrow$	write		
	$\Leftarrow$	byteenable[1:0]		
	$\Leftarrow$	writedata[15:0]		
	readdata[15:0]		$\Rightarrow$	
	irq		$\rightarrow$	

Avalon Slave Signals

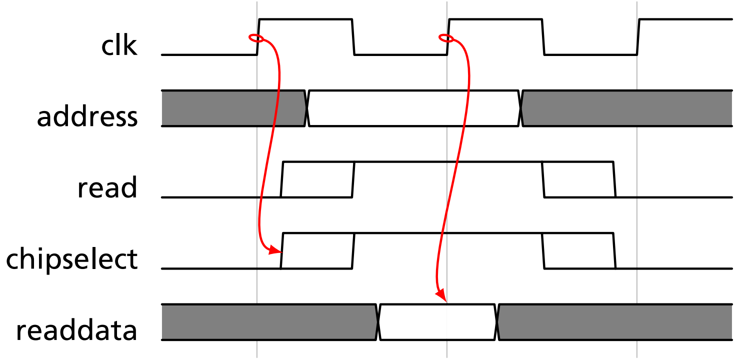
clk	Master clock
reset	Reset signal to peripheral
chipselect	Asserted when bus accesses peripheral
address[..]	Word address (data-width specific)
read	Asserted during peripheral$\rightarrow$bus transfer
write	Asserted during bus$\rightarrow$peripheral transfer
writedata[..]	Data from bus to peripheral
byteenable[..]	Indicates active bytes in a transfer
readdata[..]	Data from peripheral to bus
irq	peripheral$\rightarrow$processor interrupt request

All are optional, as are many others for, e.g., flow-control and burst transfers.

In SystemVerilog

```
module myslave(input logic        clk,
               input logic        reset,
               input logic [7:0]  writedata,
               input logic        write,
               input logic        chipselect,
               input logic [2:0]  address);
```

Basic Slave Read Transfer

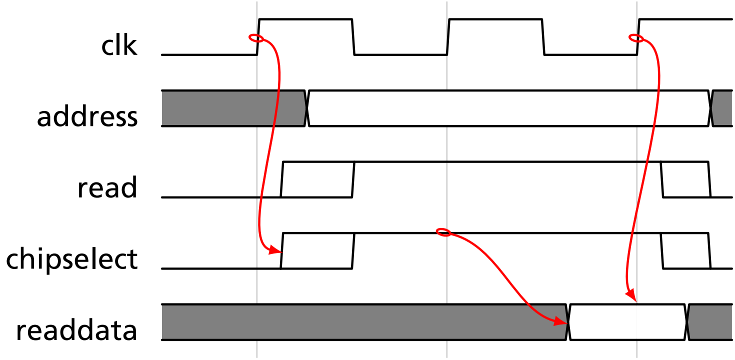


Bus cycle starts on rising clock edge

Data latched at next rising edge

Such a peripheral must be purely combinational

Slave Read Transfer w/ 1 Wait State

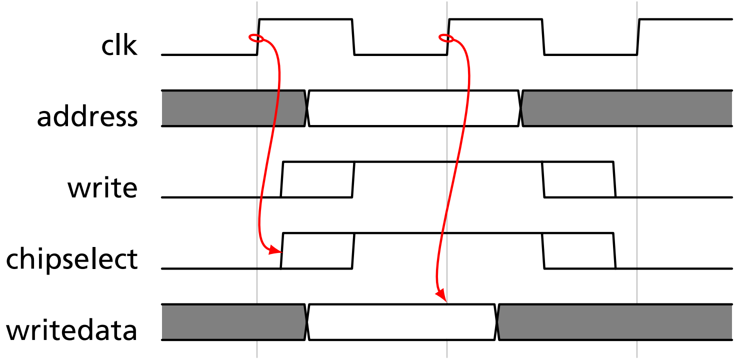


Bus cycle starts on rising clock edge

Data latched two cycles later

Approach used for synchronous peripherals

Basic Async. Slave Write Transfer

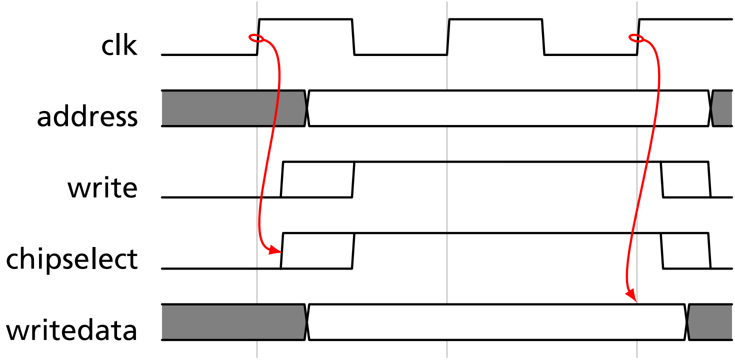


Bus cycle starts on rising clock edge

Data available by next rising edge

Peripheral may be synchronous, but must be fast

Basic Async. Slave Write w/ 1 Wait State



Bus cycle starts on rising clock edge

Peripheral latches data two cycles later

For slower peripherals

The Vga_ball Peripheral

```
module vga_ball(input logic        clk,
                input logic        reset,
                input logic [7:0]  writedata,
                input logic        write,
                input              chipselect,
                input logic [2:0]  address,

                output logic [7:0] VGA_R, VGA_G, VGA_B,
                output logic       VGA_CLK, VGA_HS, VGA_VS,
                                   VGA_BLANK_n,
                output logic       VGA_SYNC_n);

   logic [10:0]    hcount;
   logic [9:0]     vcount;

   logic [7:0]     background_r, background_g, background_b;

   vga_counters counters(.clk50(clk), .*);
```

The Vga_ball Peripheral

```
always_ff @(posedge clk)
  if (reset) begin
     background_r <= 8'h0;
     background_g <= 8'h0;
     background_b <= 8'h80;
  end else if (chipselect && write)
    case (address)
      3'h0 : background_r <= writedata;
      3'h1 : background_g <= writedata;
      3'h2 : background_b <= writedata;
    endcase

always_comb begin
   {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
   if (VGA_BLANK_n )
     if (hcount[10:6] == 5'd3 &&
         vcount[9:5] == 5'd3)
       {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
     else
       {VGA_R, VGA_G, VGA_B} =
          {background_r, background_g, background_b};
end
```