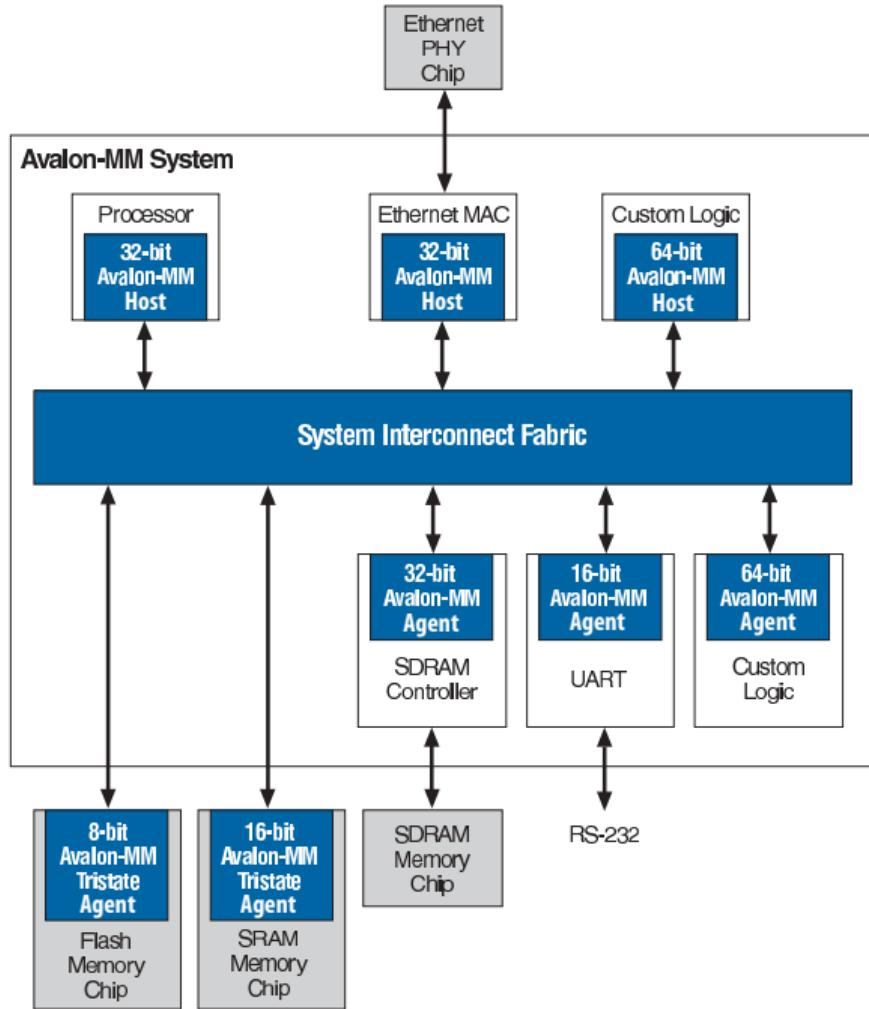


Altera's Avalon Interface

Stephen A. Edwards

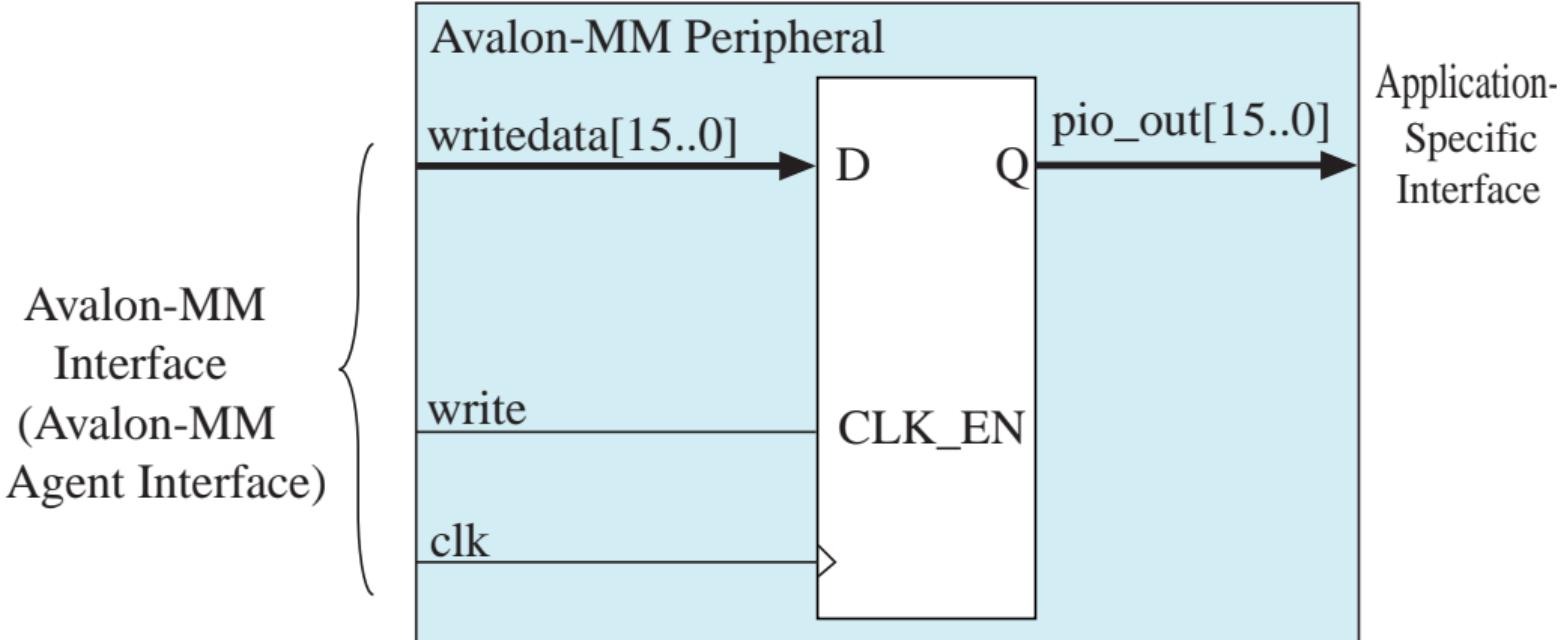
Columbia University

Spring 2026



- ▶ **Avalon Memory Mapped Host:** Initiates transactions (e.g., processor)
Complex protocol requests access first
- ▶ **Avalon Memory Mapped Agent:** Responds to hosts (e.g., peripheral, memory)
Simpler protocol: just responds
Also manager/subordinate, M/S, initiator/target, requester/responder
See *Avalon Interface Specifications*

The Simplest Agent Peripheral



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

Agent Signals

For a 16-bit connection that spans 32 halfwords,

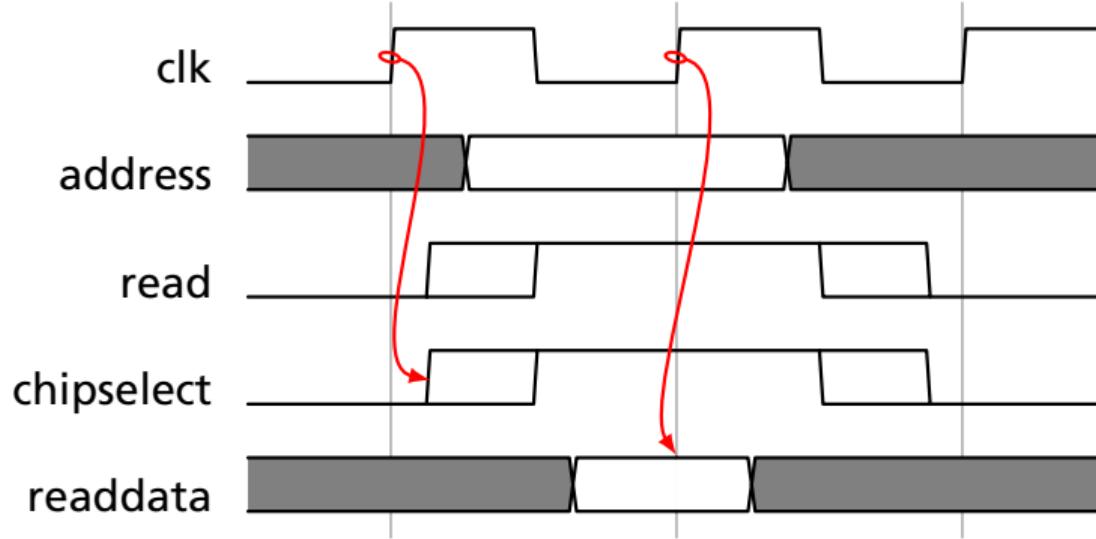
Agent	Avalon
← clk	Clock to Agent
← reset	Reset signal to Agent
← chipselect	Asserted when bus accesses Agent
⇐ address[4:0]	Register address (in words)
← read	Bus is reading from Agent
← write	Bus is writing to Agent
⇐ writedata[15:0]	Data from bus to Agent
⇐ byteenable[1:0]	Which bytes are being transferred
readdata[15:0] ⇒	Data from Agent to bus
irq →	Interrupt request to processor

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

In SystemVerilog

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

Basic Agent Read Transfer

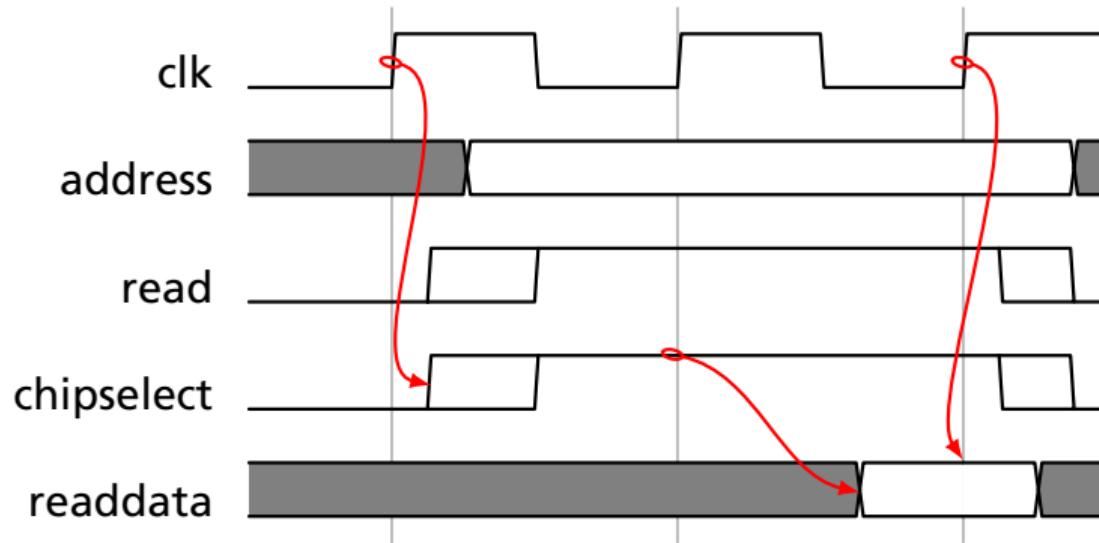


Bus cycle starts on rising clock edge

Data latched at next rising edge

Such a peripheral must be purely combinational

Agent Read Transfer w/ 1 Wait State

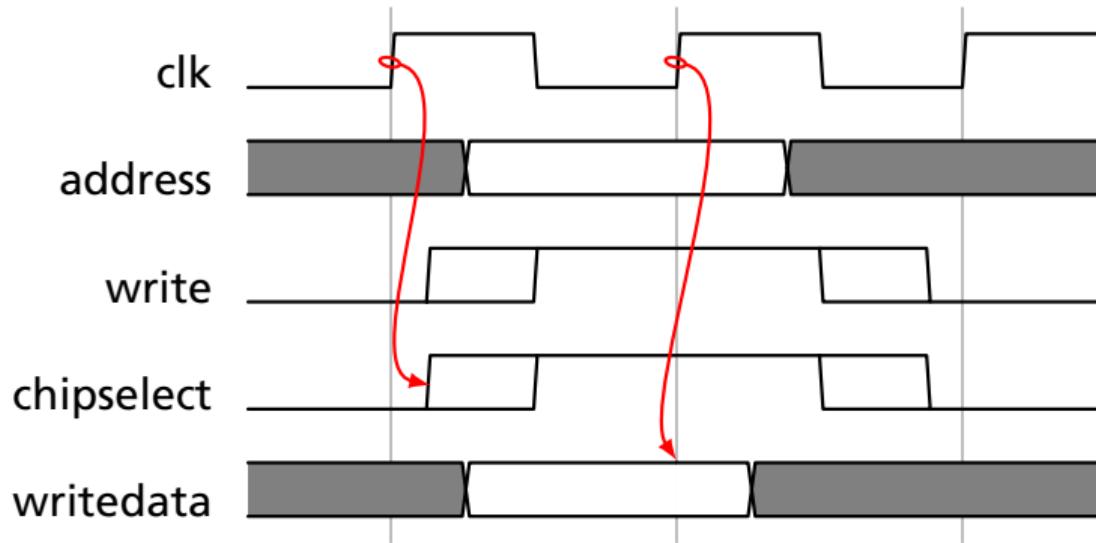


Bus cycle starts on rising clock edge

Data latched two cycles later

Approach used for synchronous peripherals

Basic Async. Agent Write Transfer

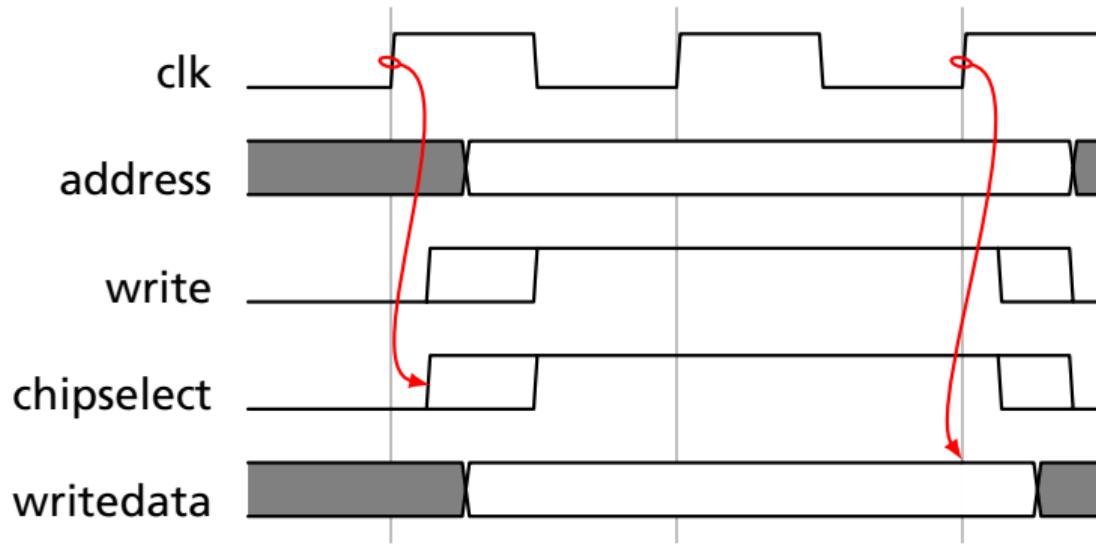


Bus cycle starts on rising clock edge

Data available by next rising edge

Peripheral may be synchronous, but must be fast

Basic Async. Agent Write w/ 1 Wait State

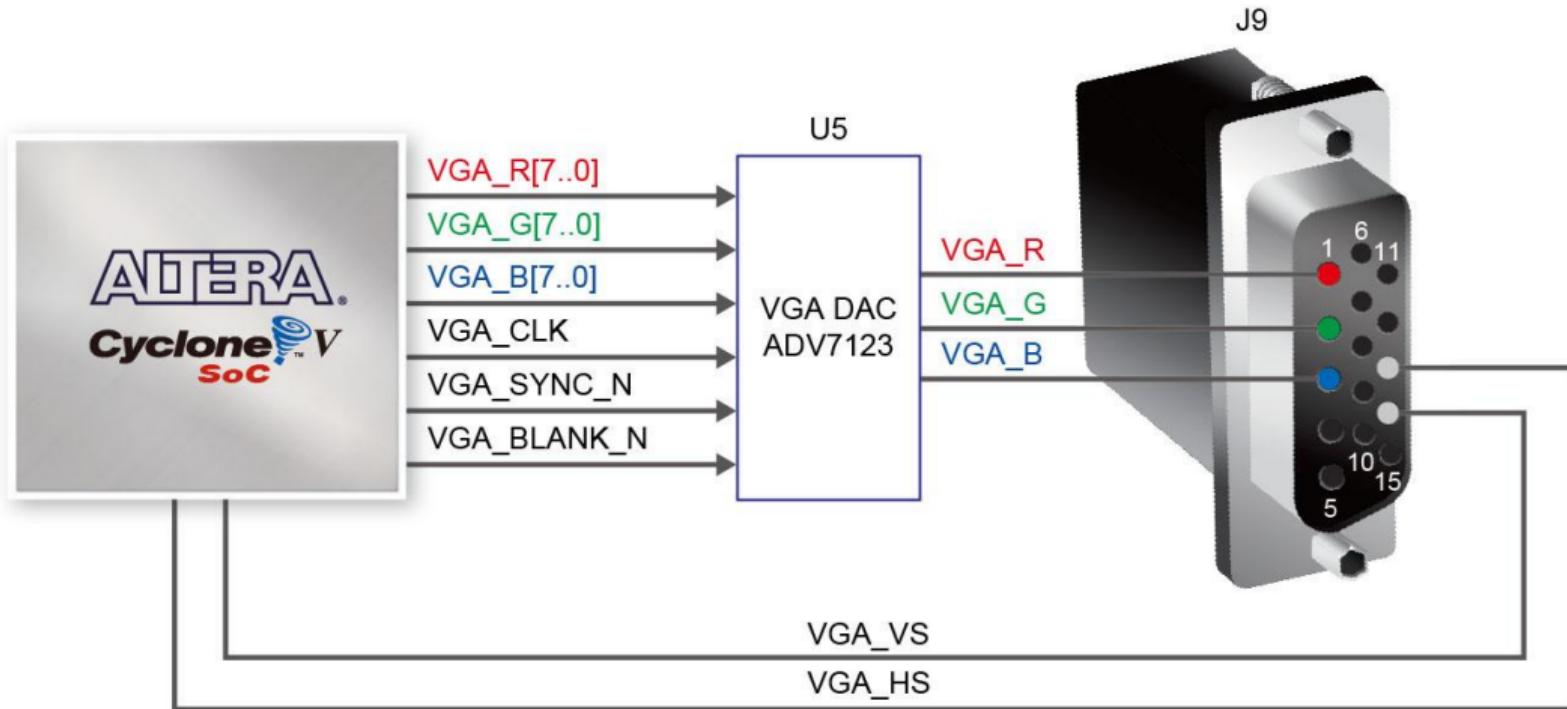


Bus cycle starts on rising clock edge

Peripheral latches data two cycles later

For slower peripherals

VGA on the DE1-SoC



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), .*);
```

Register Map

Offset	7	...	0	Meaning
0	Red			Red component of background color (0–255)
1	Green			Green component of background color (0–255)
2	Blue			Blue component of background color (0–255)

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
```