The Configurable System-on-Chip Company

The Triscend E5 Configurable System Interconnect (CSI) Bus

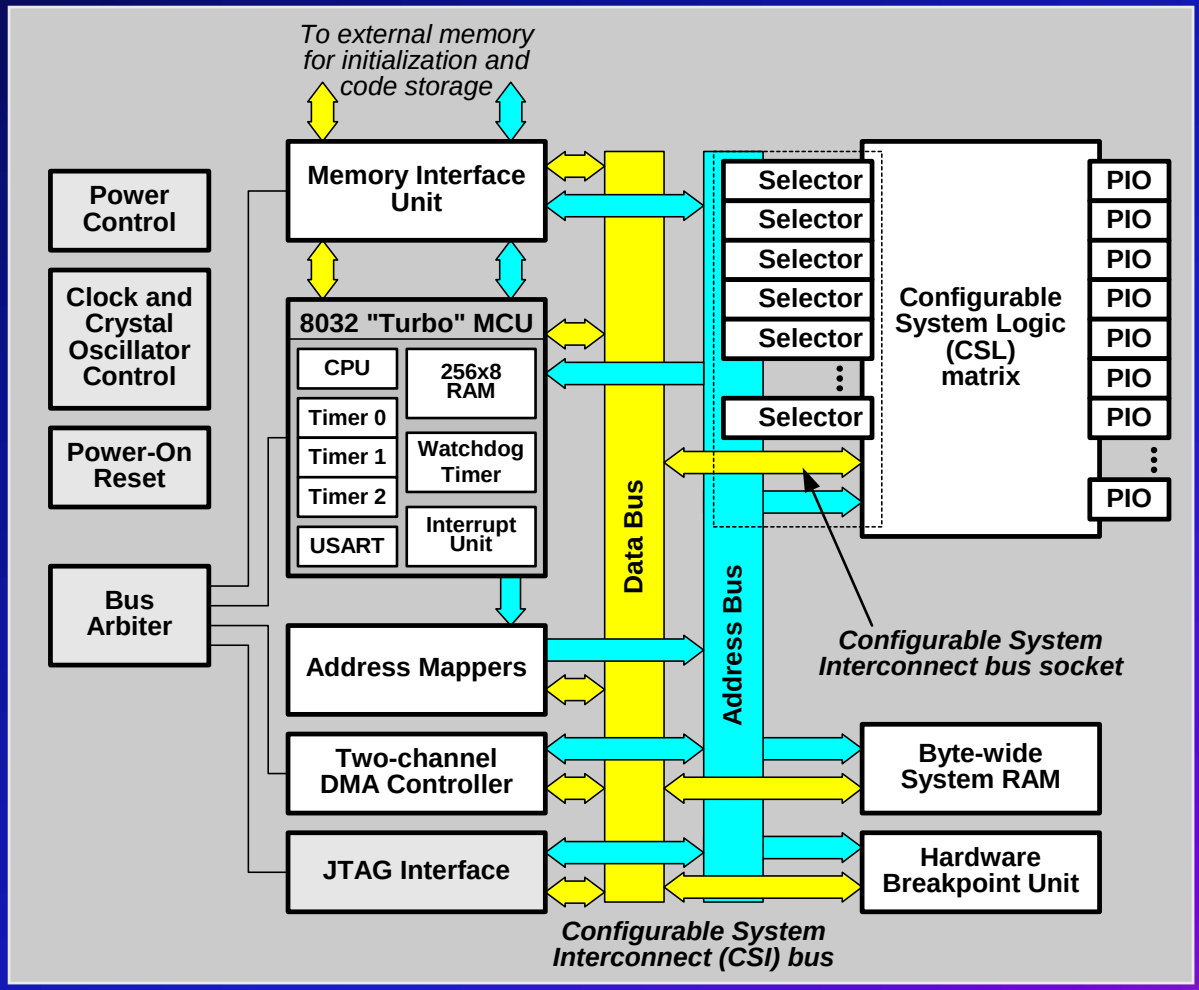
July 1999

Agenda

- The Basics
- Address Selectors
- Converting an Existing 8032 Design
- Creating Address Structures
- DMA Operations
- The CSI Bus Simulation Model
- Wait-State and Breakpoint
- Summary and Questions

Bus Basics

- Bus Master
  - 8032
  - DMA
  - JTAG
- Arbiter
- Bus Slave



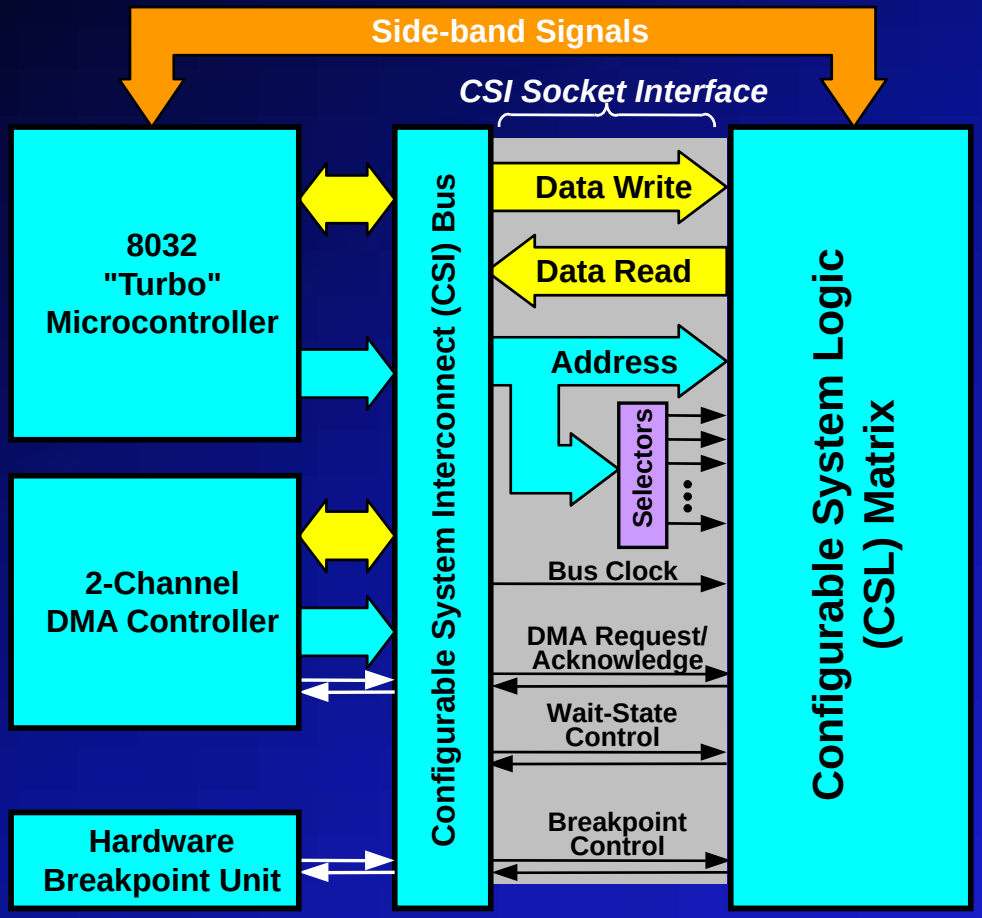
Triscend

Language Basics

- VHDL (VHSIC Hardware Description Language)
  - Widely used for logic synthesis and simulation
  - CSI Bus primitives cannot be inferred, must be instantiated
  - CSI Bus can be simulated using VHDL
- Verilog (no examples here)
- Chisel ('C'-like Hardware Specification Language)
  - Triscend proprietary language
  - Not intended for customer use

CSI Bus Basics

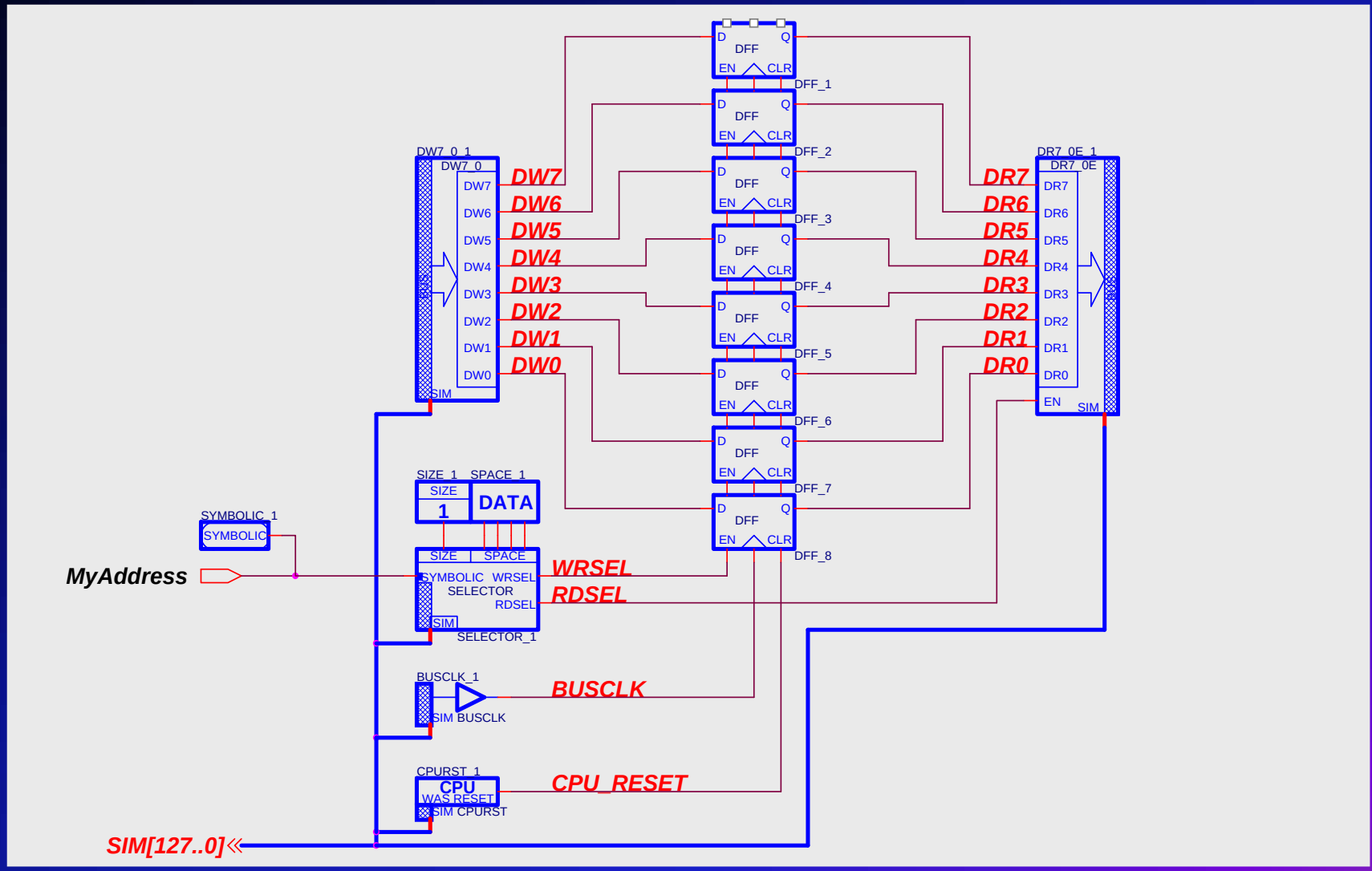
What is the CSI Bus?



- Distributes address, data, and control to functions in the CSL Matrix
- Predictable timing
- Future compatible

Triscend

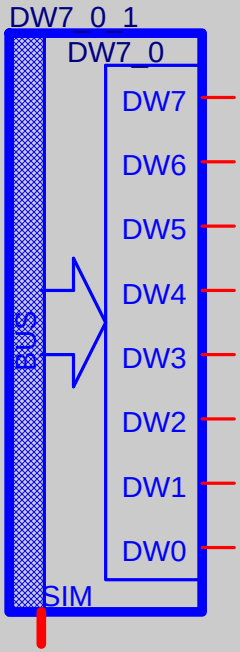
Example: Read/Write Register



Triscend

Data Write

Schematic



- Data provided by bus master
- Data available on every BusClock edge (assuming no wait-states)

Triscend

Data Write (VHDL)

Instantiate

```
architecture TRISCEND of MYDESIGN is

component DW7_0
  port (
        DW7    : out   std_logic;
        DW6    : out   std_logic;
        DW5    : out   std_logic;
        DW4    : out   std_logic;
        DW3    : out   std_logic;
        DW2    : out   std_logic;
        DW1    : out   std_logic;
        DW0    : out   std_logic;
        SIM    :  inout std_logic_vector(127 downto 0)
        );
 end component;
```

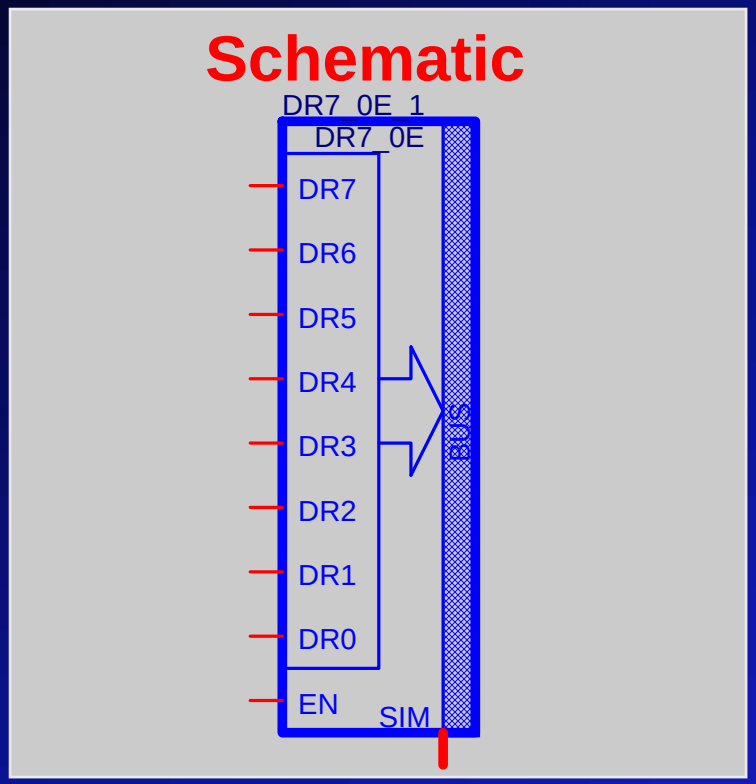
Use

```
begin
   MYDW7 : DW7_0 port map (
      DW7 => my_dw_bus(7),
      DW6 => my_dw_bus(6),
      DW5 => my_dw_bus(5),
      DW4 => my_dw_bus(4),
      DW3 => my_dw_bus(3),
      DW2 => my_dw_bus(2),
      DW1 => my_dw_bus(1),
      DW0 => my_dw_bus(0)
   );
```

Data Write (Chisel)

```
module example {

    parameter int component_width = 8;

    output [component_width] dw;

    for (int i=0; i < component_width; i++) {
        datain str("DW_",i) (.o=dw[i], :bit=i);
    }

}
```

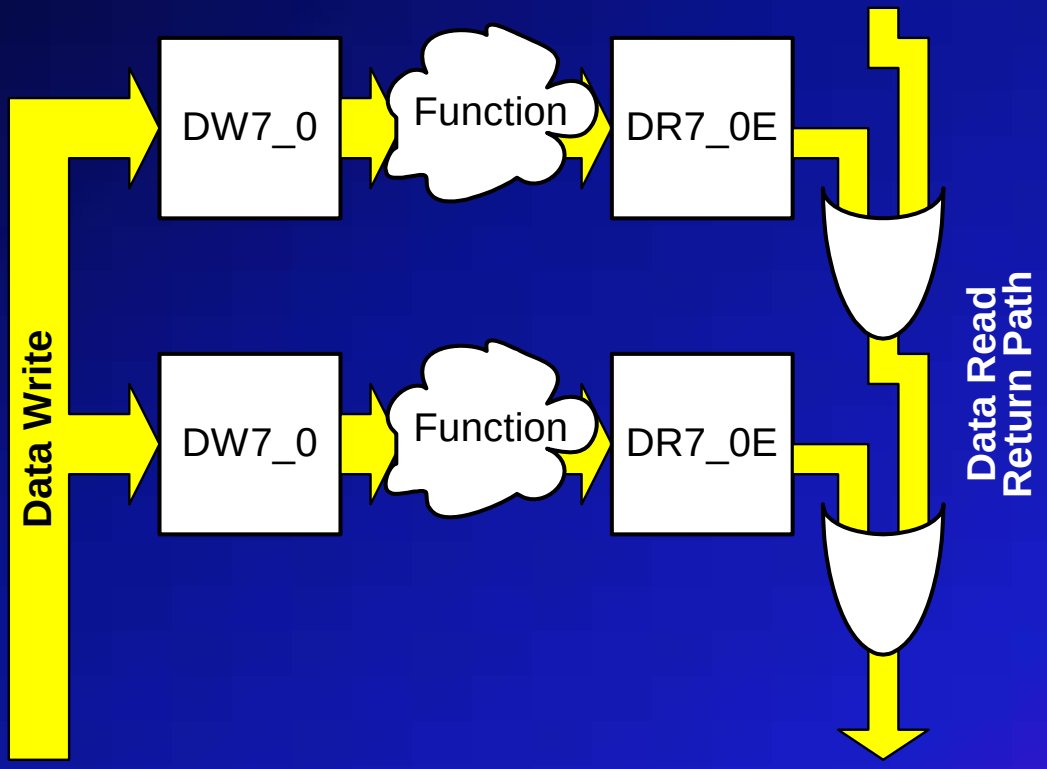
Data Read



- Data provided by currently addressed slave
- Assert EN to enable data onto the CSI bus
- Data Read is a contention-free wired-OR bus

Triscend

CSI Data Flow



- Return path is OR-ed
- Contention-free, not conflict-free

Triscend

Data Read (VHDL)

Instantiate

```
architecture TRISCEND of MYDESIGN is

component DR7_0E
  port (
        DR7    : in   std_logic;
        DR6    : in   std_logic;
        DR5    : in   std_logic;
        DR4    : in   std_logic;
        DR3    : in   std_logic;
        DR2    : in   std_logic;
        DR1    : in   std_logic;
        DR0    : in   std_logic;
        EN     : in   std_logic;
        SIM    : inout std_logic_vector(127 downto 0)
        );
end component;
```

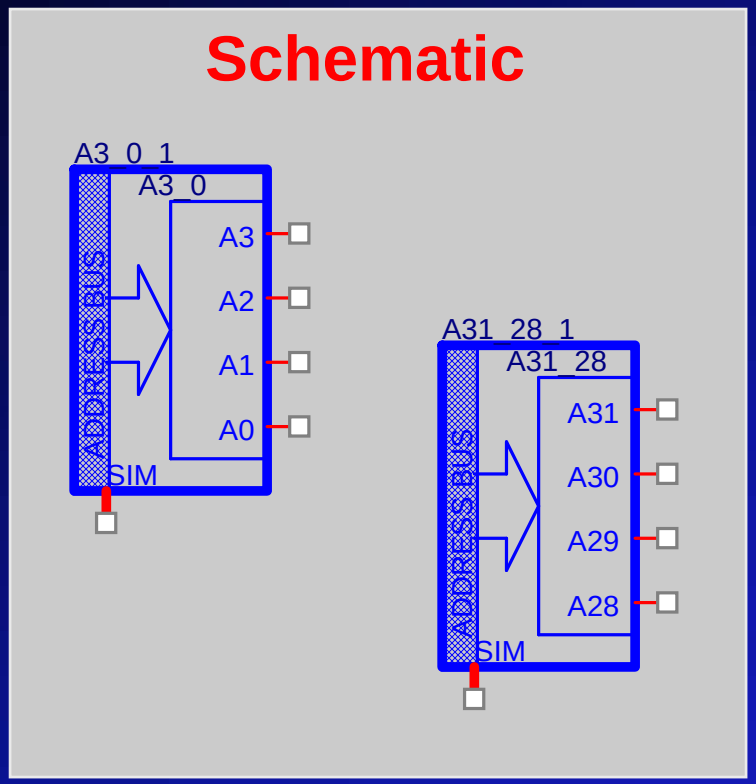
Use

```
begin
   MYDR7_0E : DR7_0E port map (
      DR7 => my_dr_bus(7),
      DR6 => my_dr_bus(6),
      DR5 => my_dr_bus(5),
      DR4 => my_dr_bus(4),
      DR3 => my_dr_bus(3),
      DR2 => my_dr_bus(2),
      DR1 => my_dr_bus(1),
      DR0 => my_dr_bus(0),
      EN  => my_read_select
   );
```

Data Read (Chisel)

```
module example {

     parameter int component_width = 8;

     input [component_width] dr;
     net [component_width] dr_dmux;
     input rdsel;

     for (int i=0; i < component_width; i++) {
         dmux str("DR_DMUX_",i)  (.d=dr[i], .s=rdsel, .o=dr_dmux);
     }
     for (int i=0; i < component_width; i++) {
         dataout str("DR_",i) (.i=dr_dmux[i], :bit=i);
     }

}
```

Address



- Address provided by bus master
- Address available on every BusClock edge (assuming no wait-states)
- Up to 32-bits in 4-bit chunks

Triscend

Address (VHDL)

Instantiate

```
architecture TRISCEND of MYDESIGN is

component A3_0
  port (
        A3    : out   std_logic;
        A2    : out   std_logic;
        A1    : out   std_logic;
        A0    : out   std_logic;
        SIM   : inout std_logic_vector(127 downto 0)
        );
 end component;
```

Use

```
begin
   MYA3_0 : A3_0 port map (
      A3 => my_addr(3),
      A2 => my_addr(2),
      A1 => my_addr(1),
      A0 => my_addr(0)
   );
```

Available Components

- A31_28
- A27_24
- A23_20
- A19_16
- A15_12
- A11_8
- A7_4
- A3_0

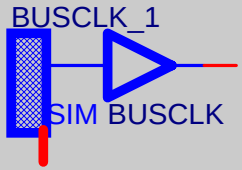
Triscend

Address (Chisel)

```
module example {

    parameter int addr_width= 8;

    output [addr_width] addr;

    for (int i=0; i < addr_width; i++) {
        addressin str("A_",i) (.o=addr[i], :bit=i);
    }

}
```

Bus Clock

Schematic



- Bus Clock drives the processor and all bus transactions
- Distributed globally within the device

Triscend

Bus Clock (VHDL)

Instantiate

```
architecture TRISCEND of MYDESIGN is

component BUSCLK
   port (
         O      : out   std_logic;
         SIM    : inout std_logic_vector(127 downto 0)
         );
end component;
```

Use

```
begin
   MYBUSCLK : BUSCLK port map (
      O => my_local_clock
   );
```

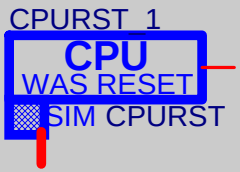
Bus Clock (Chisel)

```
module example {

   output my_local_clock;
   busclk (.o=my_local_clock);

}
```


Triscend

CPU Reset

Schematic



- Indicates that CPU was reset
  - Power-on
  - By application reset (RSTC)
  - By watchdog timer

Triscend

CPU Reset (VHDL)

Instantiate

```
architecture TRISCEND of MYDESIGN is

component CPURST
  port (
        CPURST  : out   std_logic;
        SIM  : inout std_logic_vector(127 downto 0)
        );
end component;
```

Use

```
begin
   MYRST : CPURST port map (
      CPURST => my_local_reset
   );
```

CPU Reset (Chisel)

```
module example {

   output my_reset;

   cpurst (.cpurst=my_reset);

}
```

Address Selectors

What is an Address?

- Three types of addresses for the E5
  - 8032 "**Logical**" address (16-bits, 64K)
  - E5 "**Physical**" address (32-bits, 4G)
    - 8032's 64K address spaces are mapped into regions of the 4G physical address space
    - The E5's mappers translate the 8032's logical address into an E5 physical address
  - Technology-independent "**Symbolic**" address
    - Re-useable, "plug and play" approach

How to Define an Address

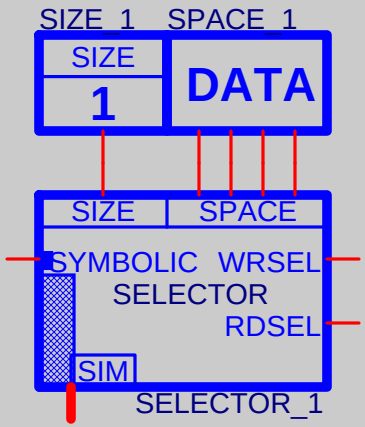
- **SIZE** of the address to decode
  - Ranges from a single byte to a 4G region
- Which address **SPACE** to decode
  - The 8032 has three user spaces
- The **SYMBOLIC** name for the decoded address
  - Technology independent
  - Designed for module re-use
  - Forward compatible, “plug and play”

CSI Address Selectors

- Distributed, integrated address decoders
- Up to 200 per device
- Multi-purpose
  - SELECTOR - decode read and write
  - CHIPSEL - decode read and chip select
  - DMACTRL - distributed DMA control
- “Plug and Play” versatility

Selector

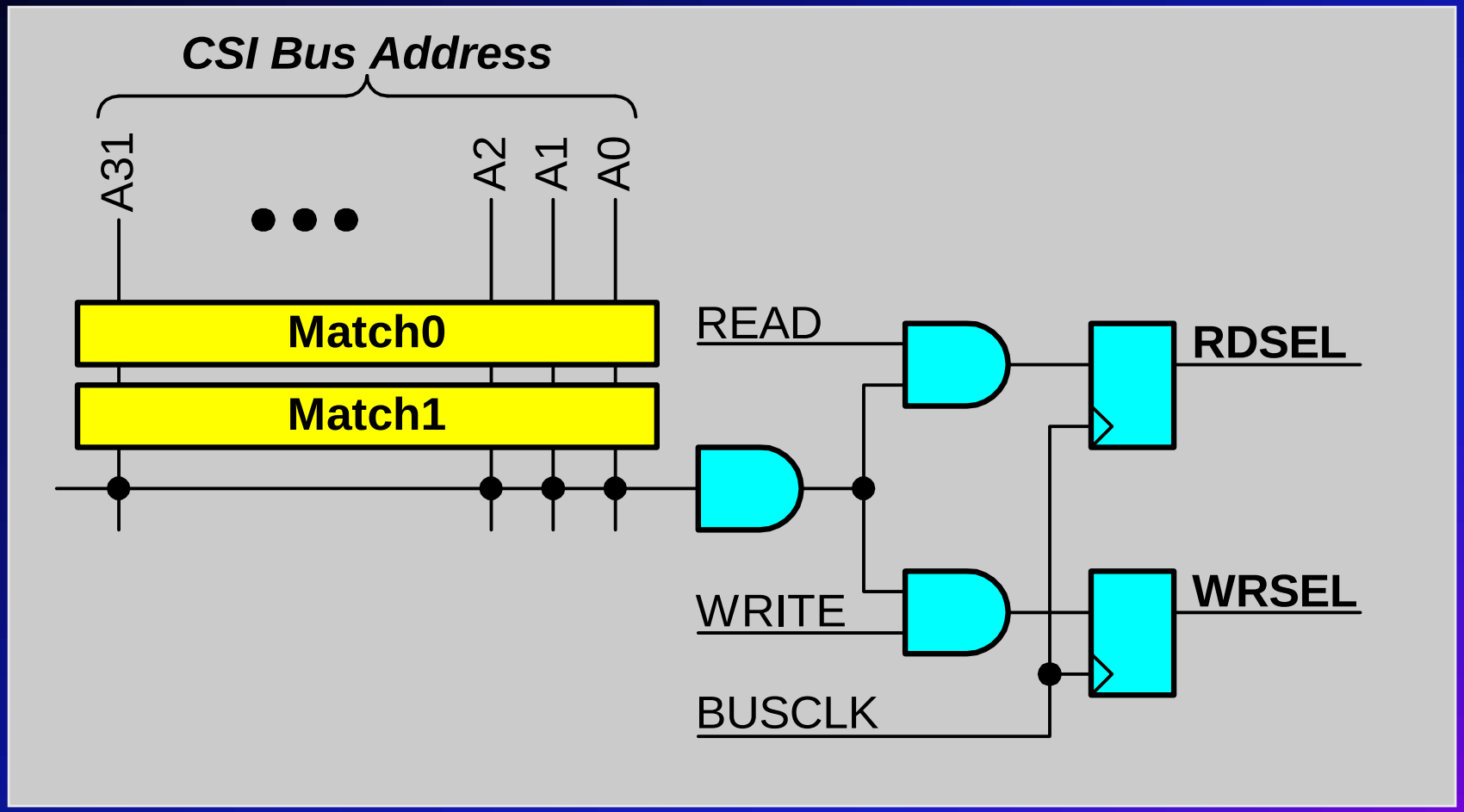
Schematic



- Separately decode read and write transactions
- Use SELECTORW if you need a wait-state

Triscend

Selector (Hardware)



Triscend

Selector Programming Example

Match1	Match0	Purpose
0	0	Disabled
0	1	An=1
1	0	An=0
1	1	Don't care

- Each address bit individually decoded
  - Match on High
  - Match on Low
  - Match on either (don't care)
- Final decode is the AND of all 32 lines

Selector (VHDL)

Instantiate

```
architecture TRISCEND of MYDESIGN is

component SELECTOR
   port (
      SYMBOLIC: in std_logic;
      SIZE          : in std_logic;
      DATA          : in std_logic;
      CODE          : in std_logic;
      SPC1          : in std_logic;
      SPC0          : in std_logic;
      WRSEL         : out std_logic;
      RDSEL         : out std_logic;
      SIM   : inout std_logic_vector(127 downto 0)
      );
end component;
```

Use

```
begin
   MYSEL : SELECTOR port map (
      SYMBOLIC  => my_symbolic,
      SIZE   => my_size,
      DATA => my_data,
      CODE => my_code,
      SPC1  => my_spc1,
      SPC0  => my_spc0,
      WRSEL => my_wrsel,
      RDSEL => my_rdsel
   );
```

Ports vs. Parameters

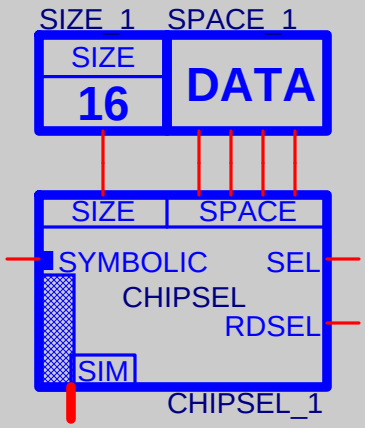
- No easy way to define parameters in schematic editors
- Limited parameterization in VHDL and Verilog
- Parameters are notoriously difficult to simulate
- Chisel supports both ports and parameters
  - A variable beginning with a period is a ***port***
  - A variable beginning with a colon is a ***parameter***

Selector (Chisel)

```
module example {

   parameter string my_symbolic = "MY_SYMBOLIC";

   output my_wrsel;
   output my_rdsel;

   selector MYSEL (
            .wrsel=my_wrsel,
            .rdsel=my_rdsel,
            :symbolic=my_symbolic,
            :size="1",
            :isdataspace=true
   );

}
```

Chip Select

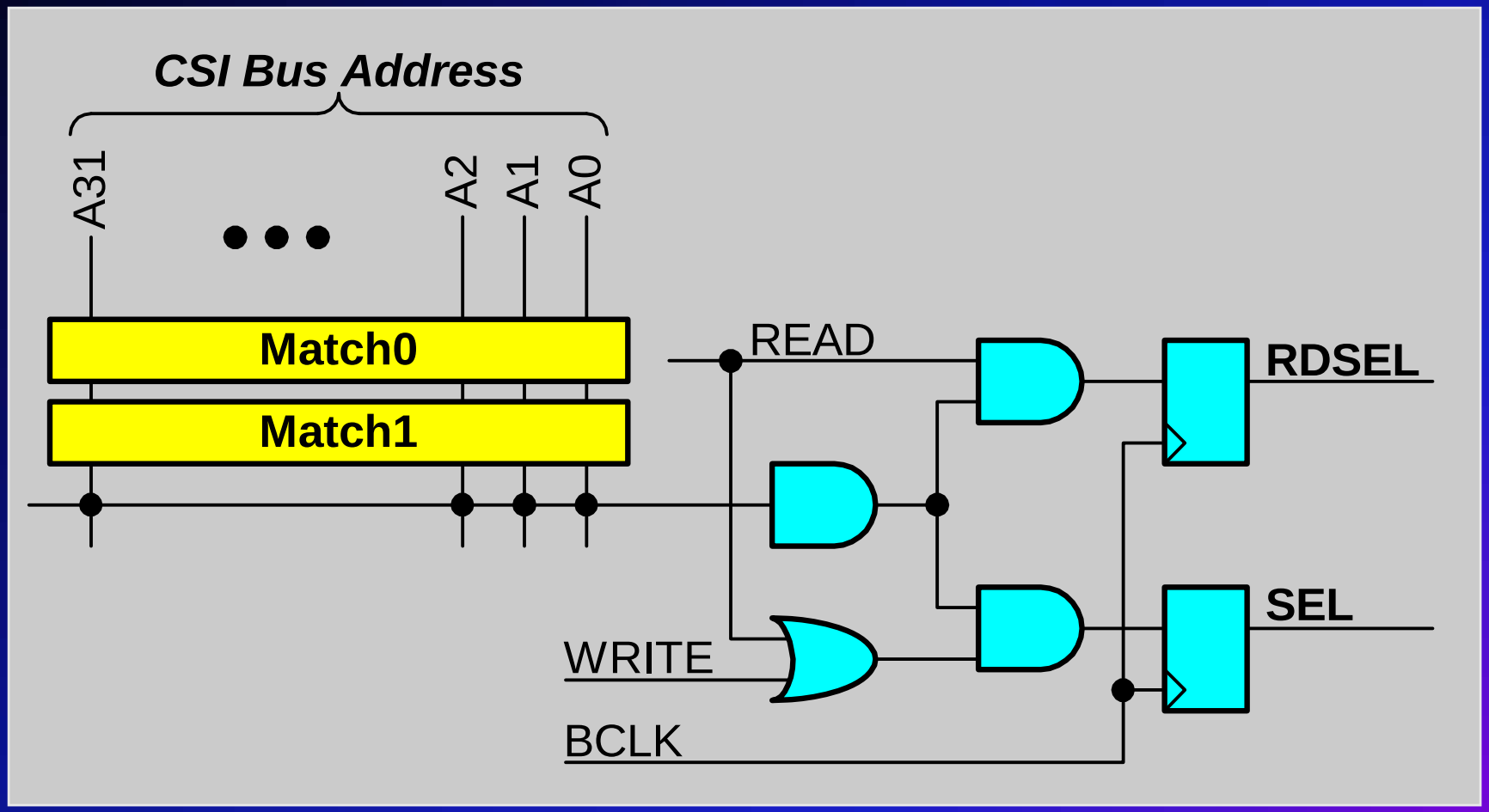
Schematic



- SEL indicates any transaction to selected address space
- RDSEL asserted only on read to selected address space (indicates direction of transfer)

Triscend

Chip Select (Hardware)



Triscend

Chip Select (VHDL)

Instantiate

```
architecture TRISCEND of MYDESIGN is

component CHIPSEL
   port (
      SYMBOLIC: in std_logic;
      SIZE          : in std_logic;
      DATA          : in std_logic;
      CODE          : in std_logic;
      SPC1          : in std_logic;
      SPC0          : in std_logic;
      SEL           : out std_logic;
      RDSEL         : out std_logic;
      SIM   : inout std_logic_vector(127 downto 0)
      );
end component;
```

Use

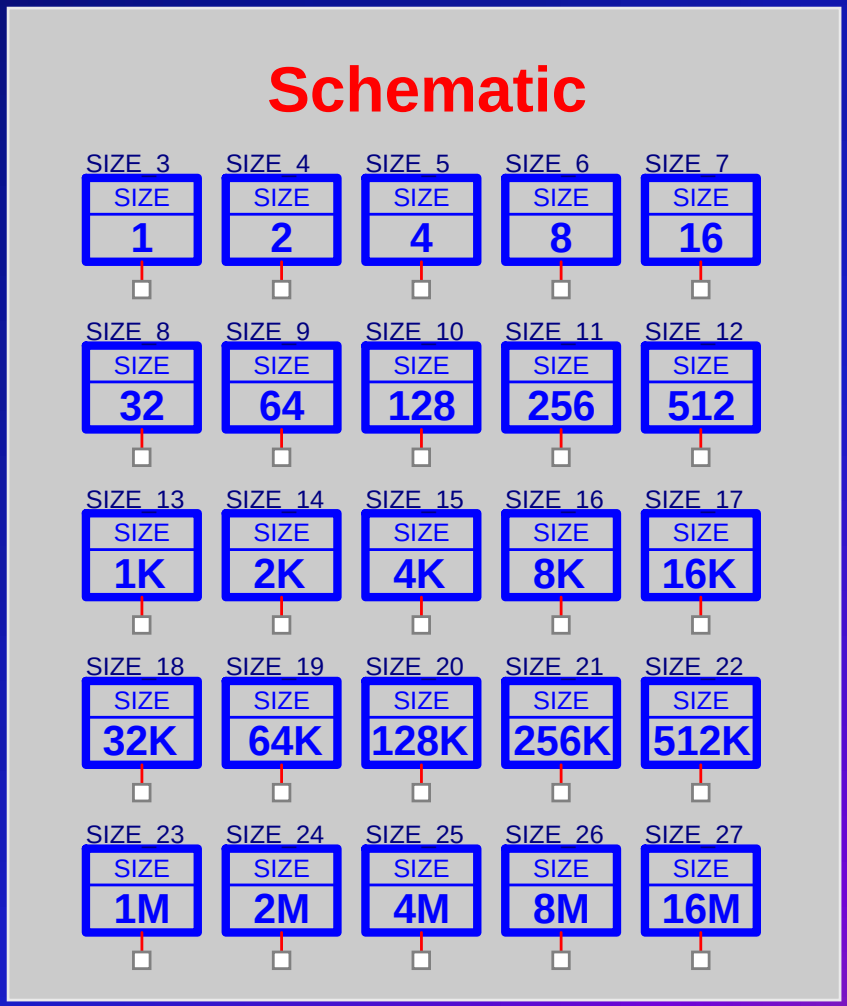
```
begin
   MYSEL : CHIPSEL port map (
      SYMBOLIC  => my_symbolic,
      SIZE   => my_size,
      DATA => my_data,
      CODE => my_code,
      SPC1  => my_spc1,
      SPC0  => my_spc0,
      SEL     => my_sel,
      RDSEL => my_rdsel
   );
```

Chip Select (Chisel)

```
module example {

   parameter string my_symbolic = "MY_SYMBOLIC";

   output my_sel;
   output my_rdsel;

   chipsel MYSEL (
      .sel=my_sel,
      .rdsel=my_rdsel,
      :symbolic=my_symbolic,
      :size="16",
      :isdataspace=true
   );

}
```

Defining the SIZE

- Defines number of bytes selected
  - Defines number of “don’t care” bits for low-order address lines
- Always a power of 2
  - SIZE1 = single byte
  - SIZE1K = 1 Kbyte
  - SIZE1M = 1 Mbyte
- Up to 16M in E5 family



Triscend

Size (VHDL)

Instantiate

```
architecture TRISCEND of MYDESIGN is

component SIZEx
   port (
      O    : out std_logic
      );
end component;
```

SIZE1, SIZE1K, SIZE1M, etc.

Use

```
begin
   MYSIZE : SIZEx port map (
      O => my_size
   );
```

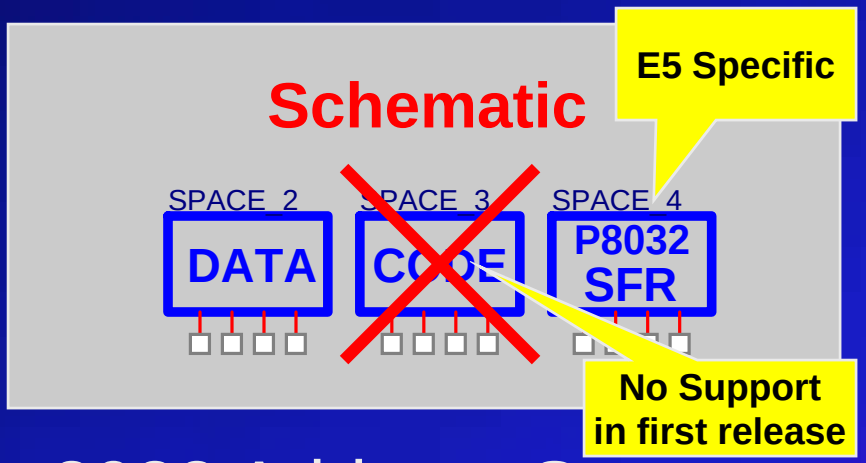
Size (Chisel)

- Parameter on various Selector functions
- Example: `:size="16K"`

Triscend

Defining the SPACE

- Defines the address space for the decode
  - Data (XDATA)
  - Code
  - Special Function Registers (SFRs)
- Defines the upper byte presented on the address bus
  - Mappers



- 8032 Address Spaces
  - XDATA is 64K
  - CODE is 64K
  - SFR is about ~90 bytes

Triscend

SPACE (VHDL)

Instantiate

```
architecture TRISCEND of MYDESIGN is

component DATA
   port (
      DATA        : out std_logic;
      CODE        : out std_logic;
      SPC1        : out std_logic;
      SPC0        : out std_logic
   );
end component;
```

DATA, CODE, SFR, etc.

Use

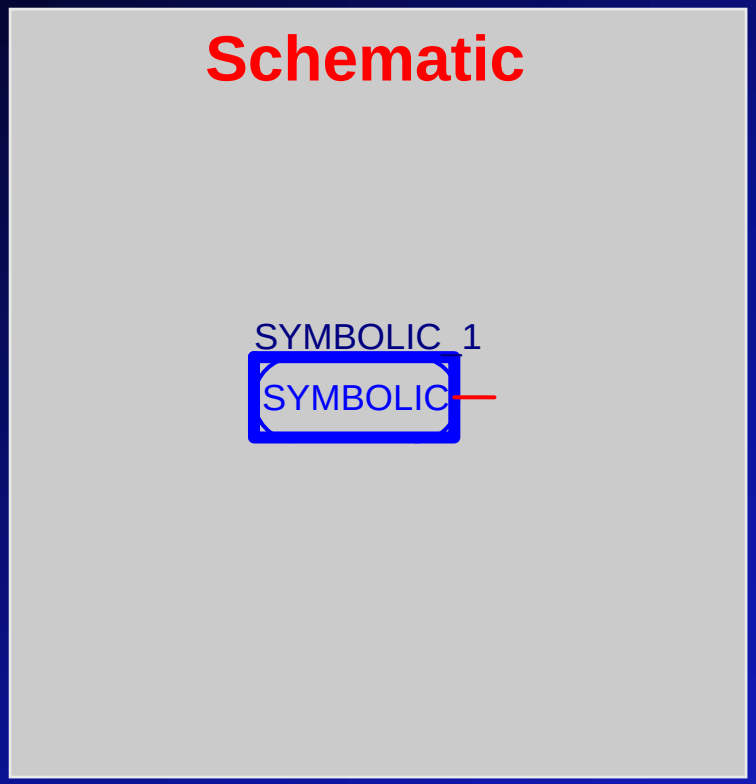
```
begin
   MYSPACE : DATA port map (
      DATA => my_data,
      CODE => my_code,
      SPC1  => my_spc1,
      SPC0  => my_spc0
   );
```

SPACE (Chisel)

- Parameter on various Selector functions
- Example: **`:isdataspace=true`**

Triscend

SYMBOLIC



- Defines a net as a symbolic name
- Symbolic inputs are treated separately from normal input nets
- Maintains simulation capability (no parameters)

Triscend

SYMBOLIC (VHDL)

Instantiate

```
architecture TRISCEND of MYDESIGN is

component SYMBOLIC
   port (
      I   : in std_logic
   );
end component;
```

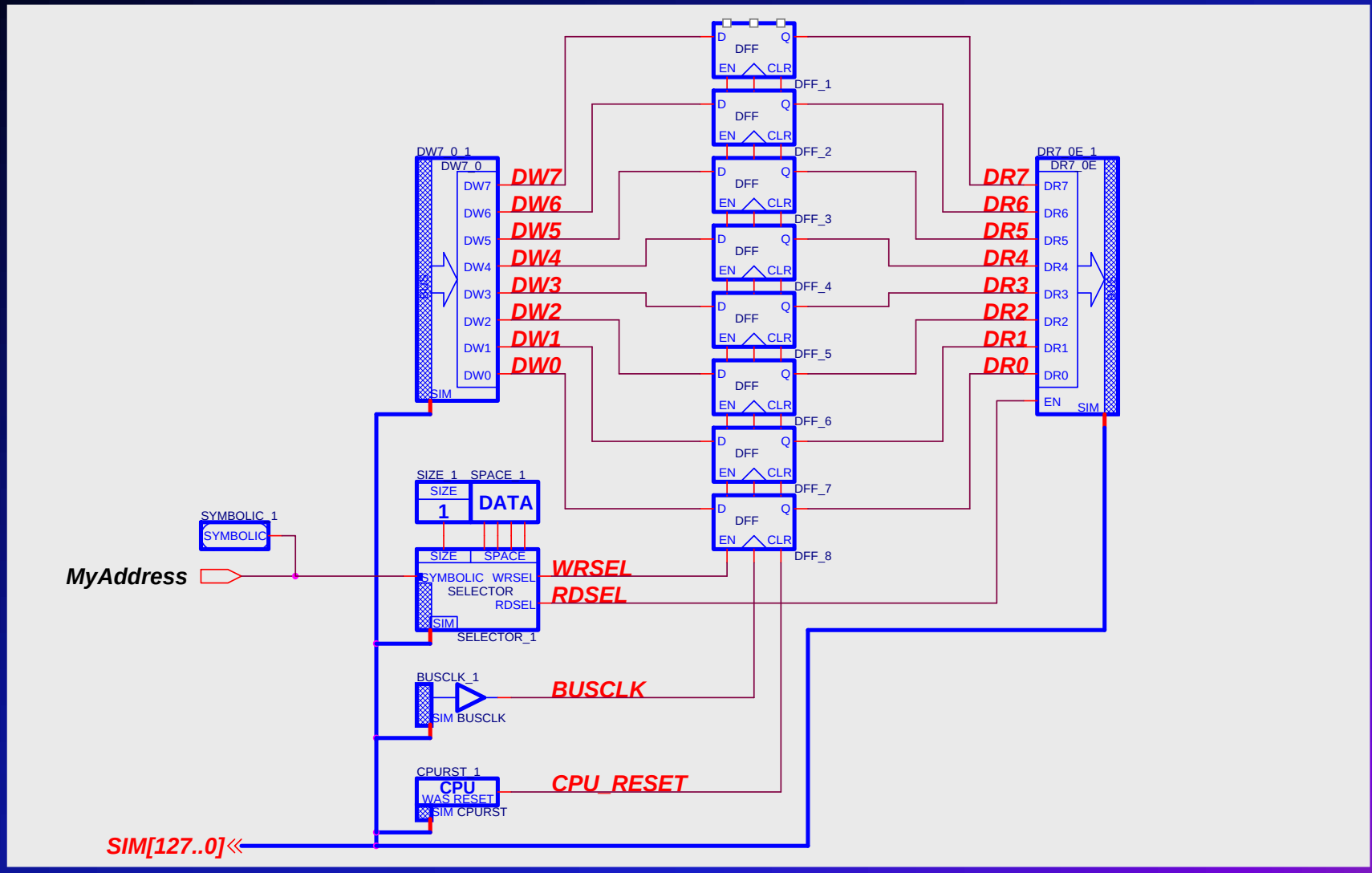
Use

```
begin
   MYADR : SYMBOLIC port map (
      I => my_addess
   );
```

SYMBOLIC (Chisel)

- Parameter on various Selector functions
- Example: **`:symbolic="my_addr"`**

Example: Read/Write Register



Triscend

Interface to Compilers/Assemblers

- FastChip Address Allocation
  - Examines information from each Selector function
    - SIZE, SPACE, SYMBOLIC
  - Creates address assignment for each address space
- Allocation file (.al) created by FastChip
- Header file contains address information for compiler/assembler

Example Allocation File

- Located in ‘data/ProjectRegistry/<proj>’
- File is named ‘<proj>.al’

```
// Allocation written 7/10/99 5:27 AM
// By FastChip Version Preview 4 Release 061999 Build 08
// Summary of zones:
// -----------------
// CRU_DATA (DMAP3) L:0xff00:0xffff P:0x2_0e00:0x2_0eff F:0 T:0x98 A:0x0
// USER_DATA_C (DMAP2) L:0x0000:0x1fff P:0x1_8000:0x1_9fff F:0 T:0x90 A:0x0
// USER_DATA_D (DMAP1) L:0x0000:0xffff P:0x10_0000:0x10_ffff F:1 T:0x88 A:0x0
// SRAM_DATA (DMAP0) L:0x0000:0xffff P:0x2_0000:0x2_ffff F:0 T:0x80 A:0x0
// USER_CODE_B (CMAP1) L:0x0000:0x7fff P:0x1_0000:0x1_7fff F:0 T:0x28 A:0x48
// ROM_CODE (CMAP0) L:0x0000:0x03ff P:0x0000:0x03ff F:0 T:0x20 A:0x40
// SFR_DATA (XMAP) L:0x0080:0x00ff P:0x1f_ff80:0x1f_ffff F:1 T:0xc0 A:0xe0
S E520JTAG;
A "Result.CMDREG" "Result" "USER_DATA_D" 0x2000;      // 0x10_2000
  "Switch.STATREG" "Switch" "SFR_DATA" 0x009a;        // 0x1f_ff9a
```

Mapper programming information

Allocation style

Assigned user locations

Triscend

Generated Header File

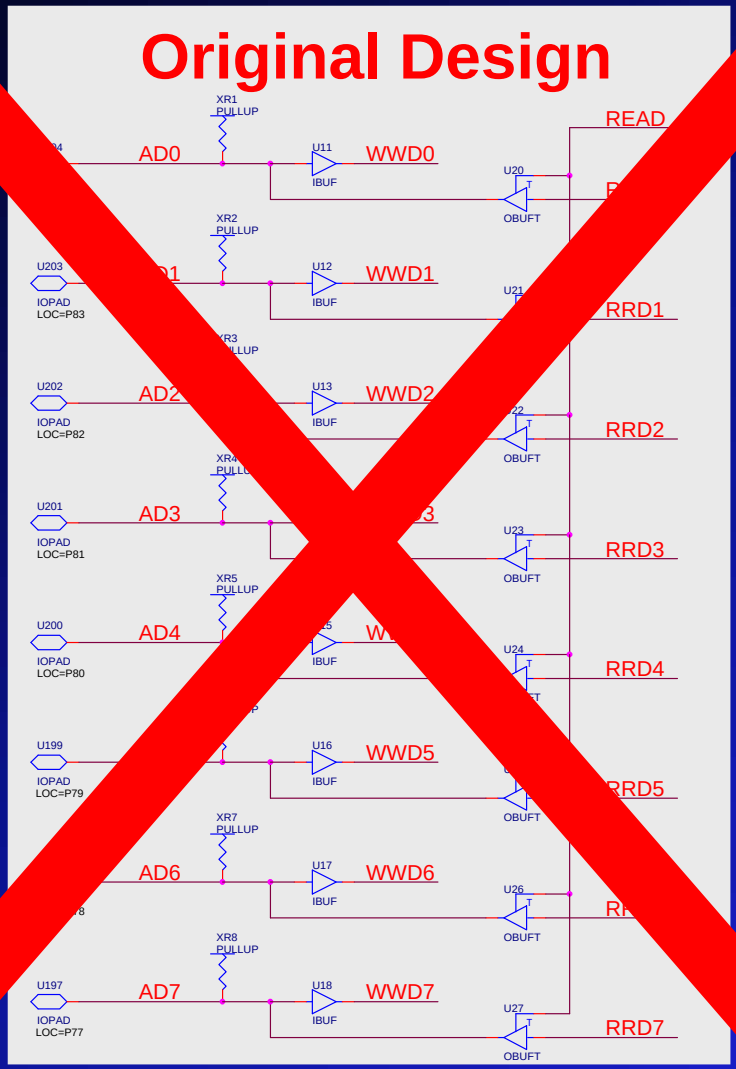
- Header file created during Generate
- Saved in either A51 (.inc) or C51 (.h) format, targeted to Keil
- Contains compiler/assembler view of allocation (plus much more)

```
//========= BEGIN SOFT MODULE REGISTER DECLARATIONS ======

//------------------------------- Module Result
    CHAR_XDATA (Result,0x2000)

//------------------------------- Module Switch
    sfr Switch = 0x9a;

//========== END SOFT MODULE REGISTER DECLARATIONS =======
```


Triscend

Converting an Existing 8032 Design to CSI Bus

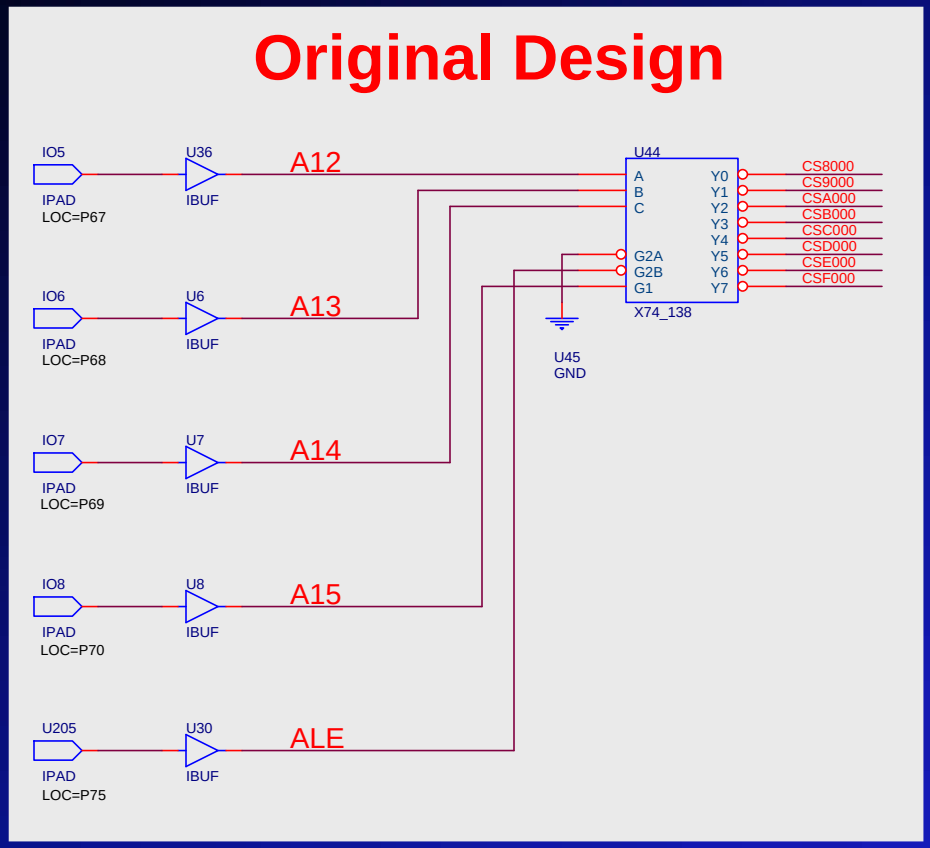
Data Interface



- 8032 data interface embedded on CSI bus
- Saves I/O pin count
- Reduces power consumption and EMI
- Replaced by CSI Bus Data Write and Data Read primitives

Triscend

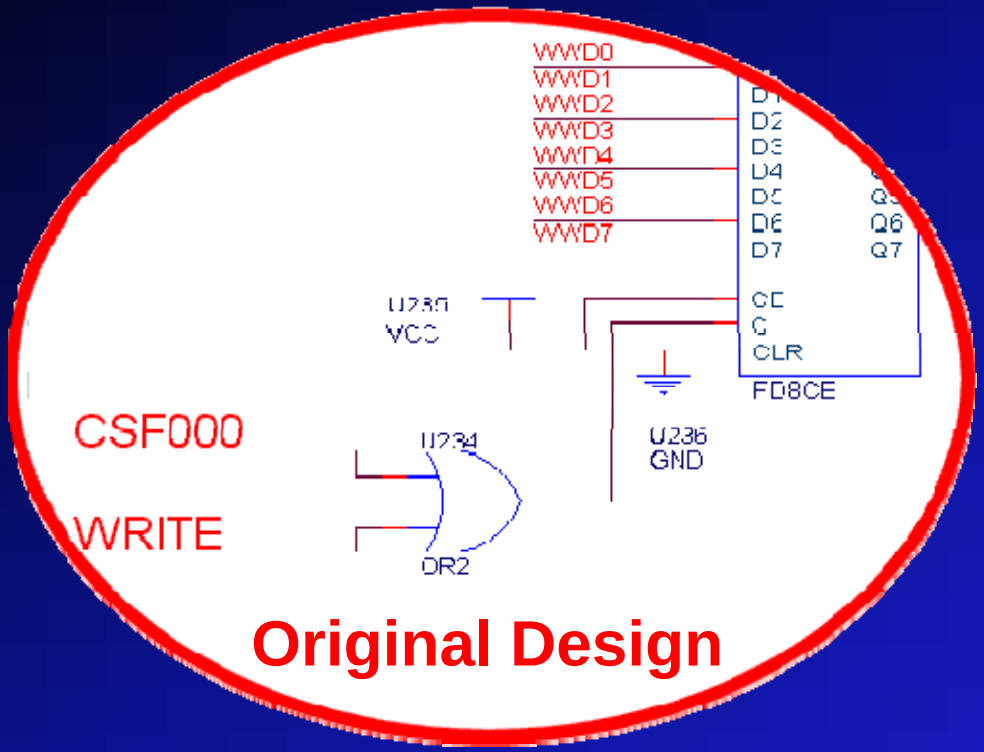
Address Decoding



- Original 8032 had multiplexed address/data (ALE)
- Expensive to decode full address bus
- Address lines embedded on CSI bus
- Saves I/O pin count
- Reduced power, EMI

Triscend

Addressing Registers



- CSI bus Selectors replace address decoding logic in the application
- See Read/Write Example
- CSI approach is safe, synchronous design

Triscend

Address Decoding (VHDL)

```
CE_ADDR <= TRUE when   ((ADDR = "10000000") or
                        (ADDR = "10000001") or
                        (ADDR = "10000010") or
                        (ADDR = "10000011") or
                        (ADDR = "10000100") or
                        (ADDR = "10000101") or
                        (ADDR = "10000110") or
                        (ADDR = "10000111") or
                        (ADDR = "10001000") or
                        (ADDR = "10001001") or
                        (ADDR = "10001010") or
                        (ADDR = "10001011") or
                        (ADDR = "10001100") or
                        (ADDR = "10001101") or
                        (ADDR = "10001110") or
                        (ADDR = "10001111"))
                  else FALSE;

CE <= '0' when ((CE_ADDR) and (ALE = '0')) else '1';
```

Decodes 16 bytes

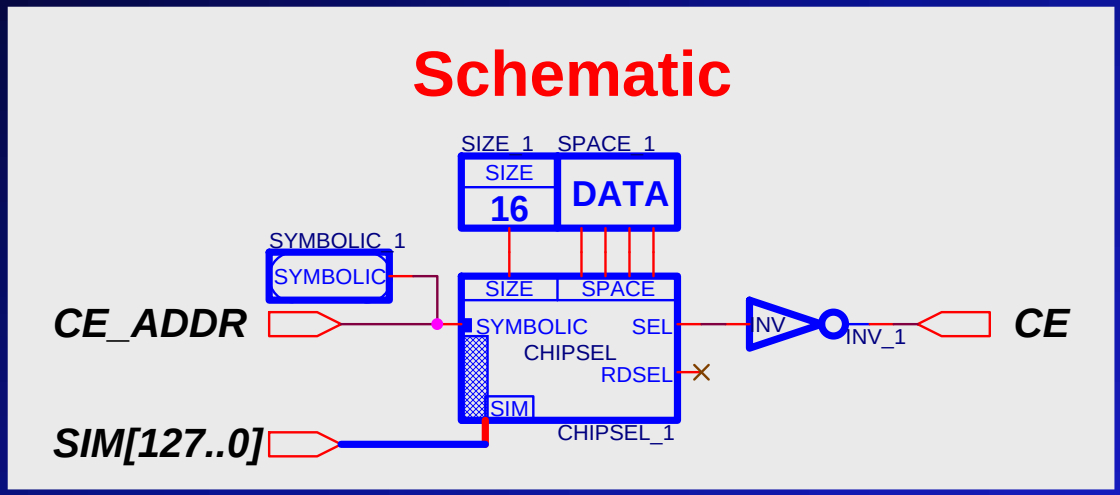
Specifying logical address, not portable

Final chip select signal is active-Low

8032 signal indicating address cycle

Triscend

CSI Bus Equivalent



- Function is a chip select (decode independent of Read or Write)
- For existing applications, most peripherals are in DATA space

Triscend

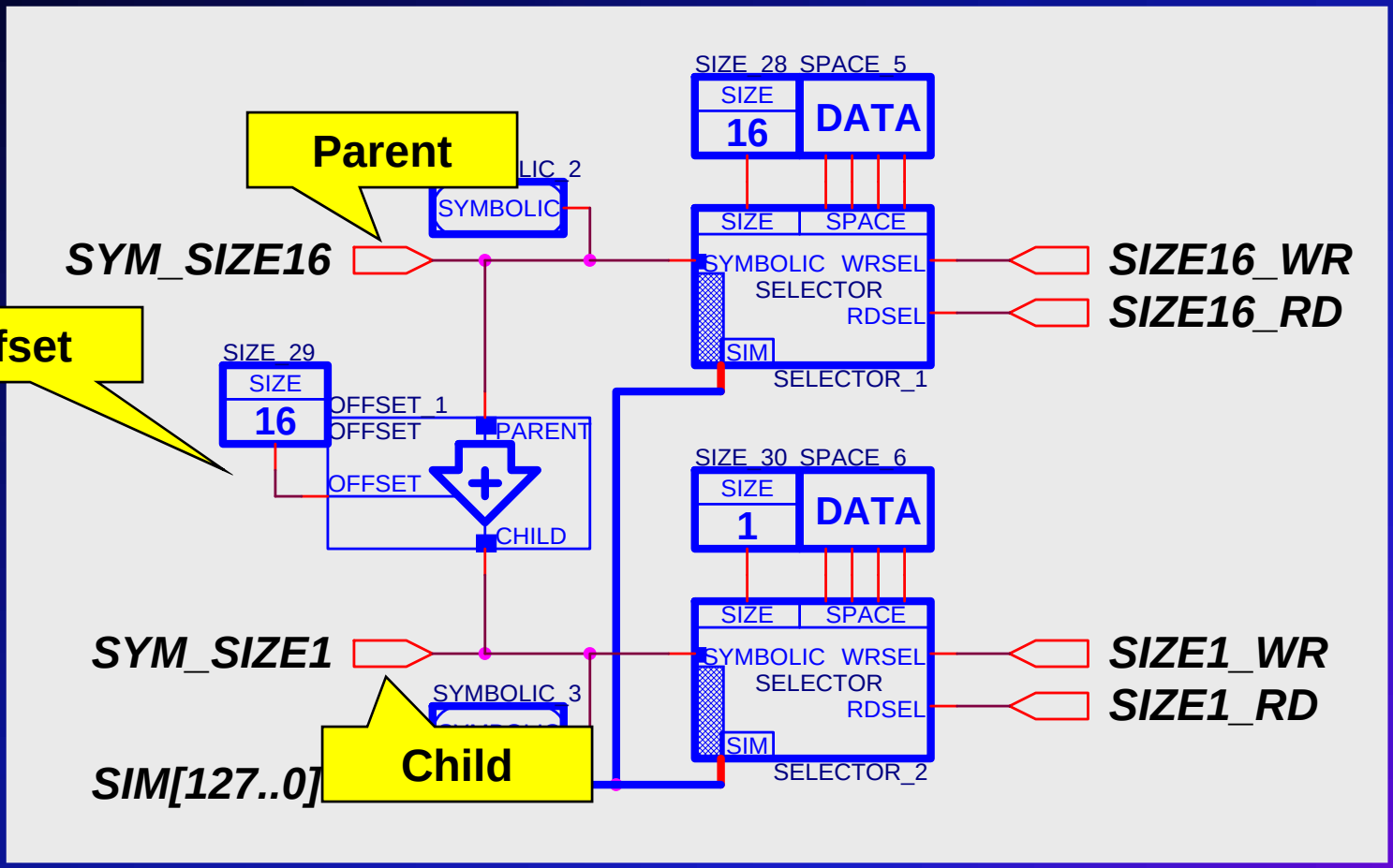
Creating Address Structures

Relative Assignments and Offsets

Creating Memory Structures

- Symbolic addressing allows technology-independent design
- How do you create a memory structure?
  - Very important for DMA operations
  - DMA requires contiguous addresses
- Need ability to specify relationships
  - Parent-child relationships
  - Offset between addresses

Relative Addressing Example



Triscend

OFFSET (VHDL)

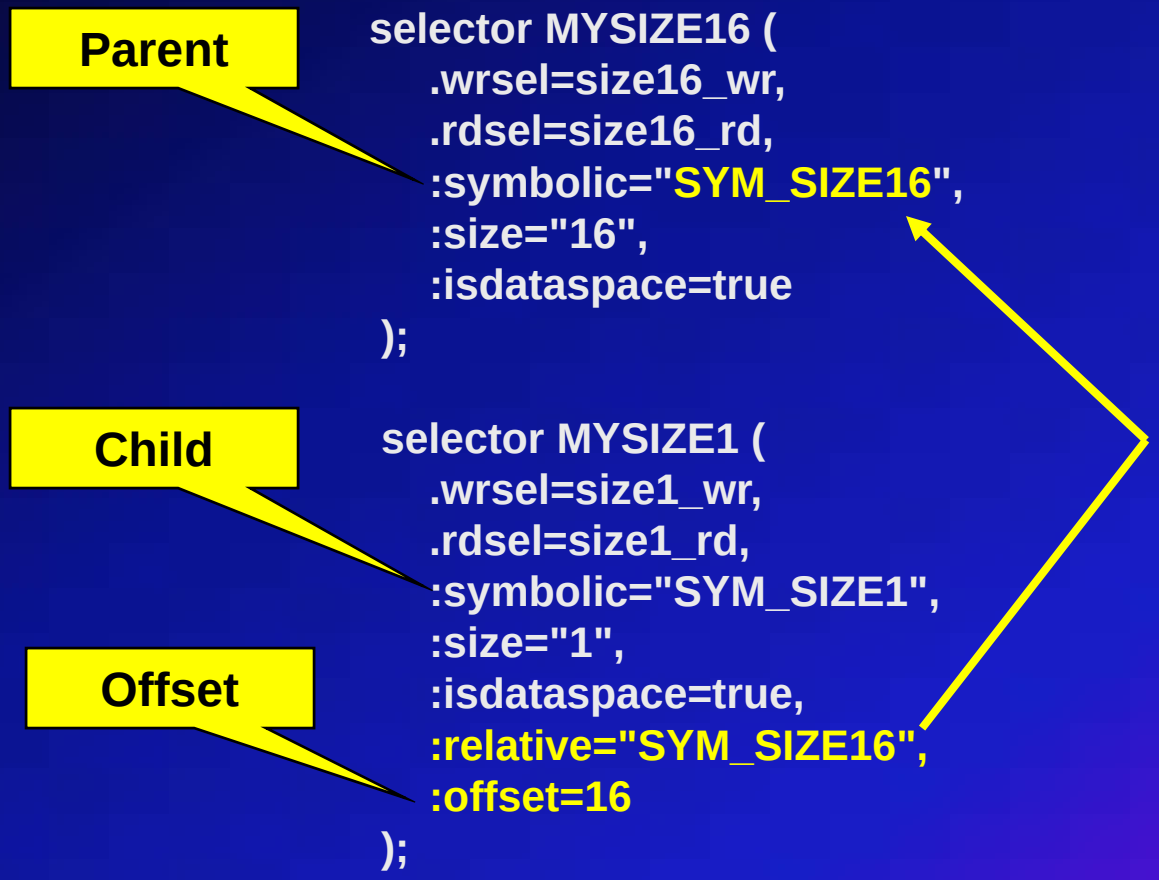
Instantiate

```
architecture TRISCEND of MYDESIGN is

component OFFSET
  port (
    CHILD    : in std_logic;
    PARENT : in std_logic;
    OFFSET  : in std_logic
  );
end component;
```

Use

```
begin
  MYOFFSET : OFFSET port map (
    CHILD    => my_child,
    PARENT => my_parent,
    OFFSET  => my_offset
  );
```

OFFSET (Chisel)



Triscend

Relative Addressing Rules

- Child must be aligned to a power-of-2 of its size
  - Byte aligned for SIZE1
  - Aligned to 1K boundary for SIZE1K
- Parent and child must be in same address space (no mixing)

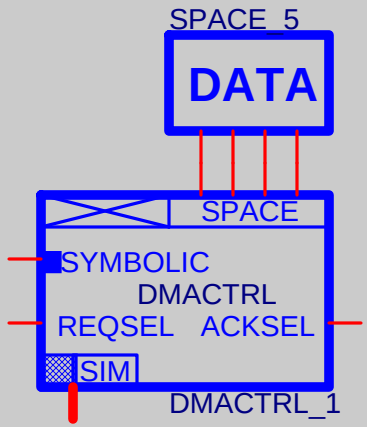
DMA Operations

DMA Control Registers

- Selectors alter ego: The DMA Control Register
- Provides distributed DMA services to functions within the CSL logic
- DMA controller provides “proxy” bus mastering services for CSL peripherals
- DMA Control Registers work in conjunction with DMA controller

DMA Control Register

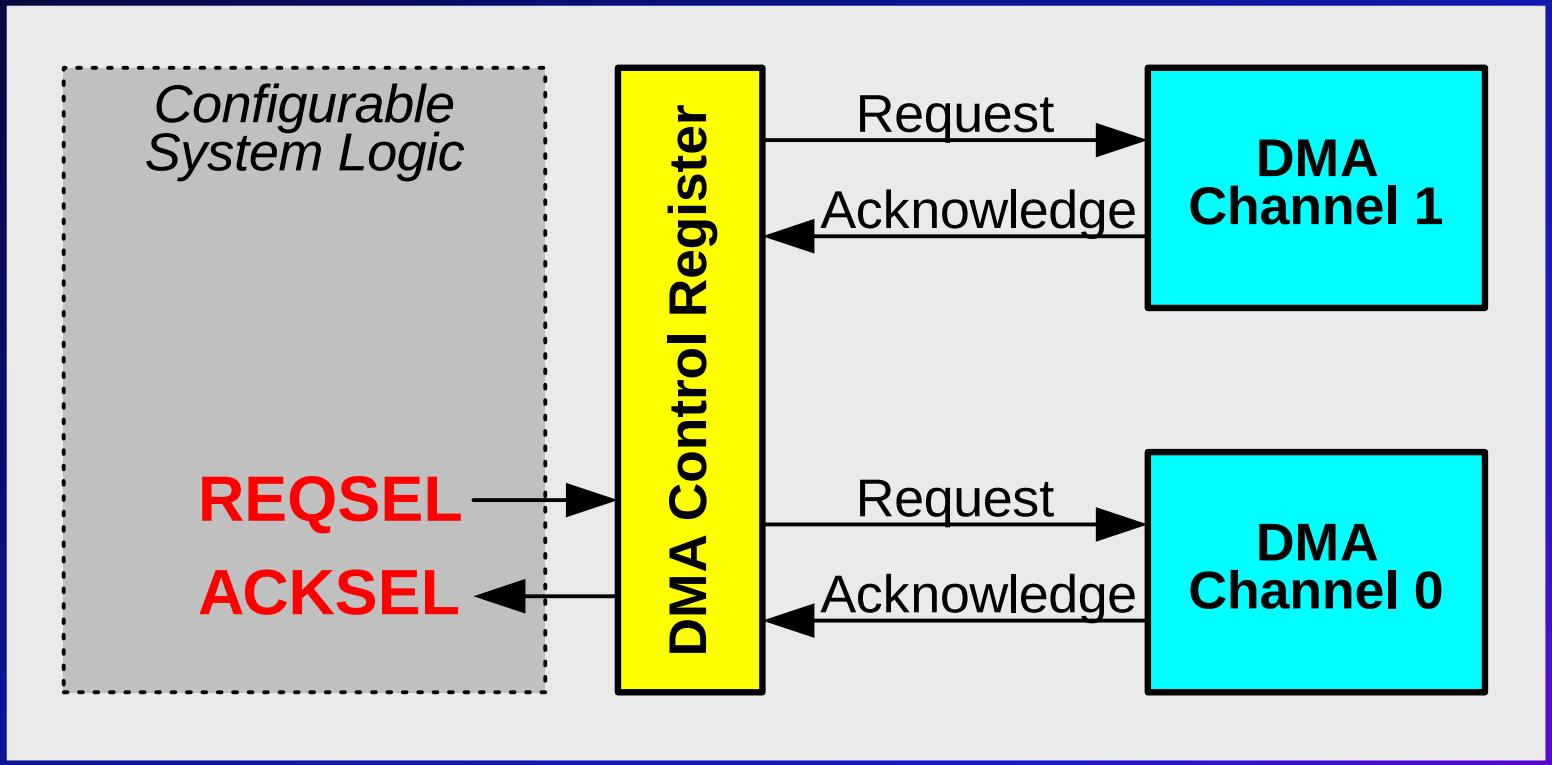
Schematic



- Provides Request and Acknowledge steering to selected DMA channel
- Always one byte
- Exists in either Data or SFR space
- Control register located at SYMBOLIC address

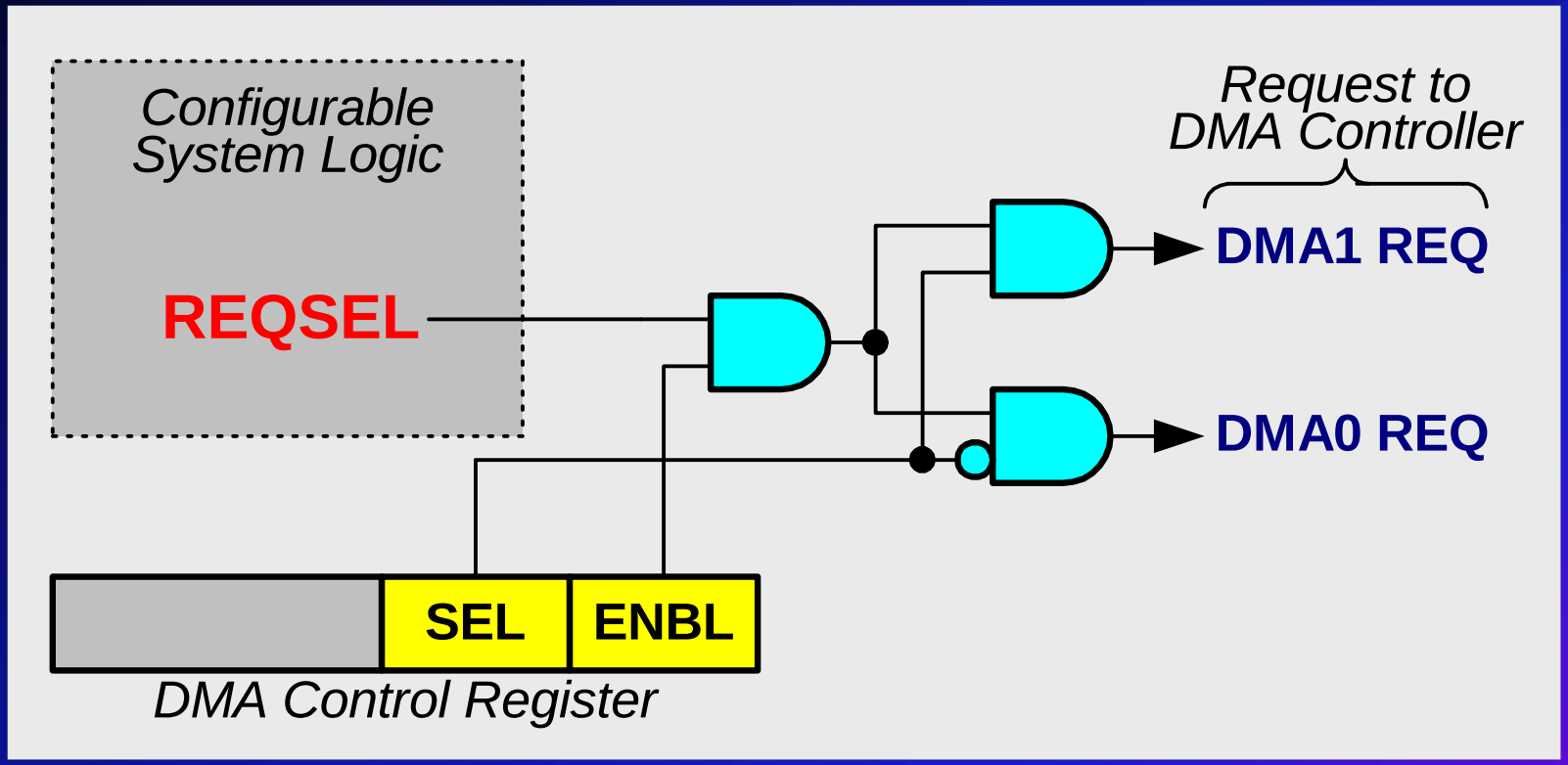
Triscend

DMA Control Register (Physical Model)



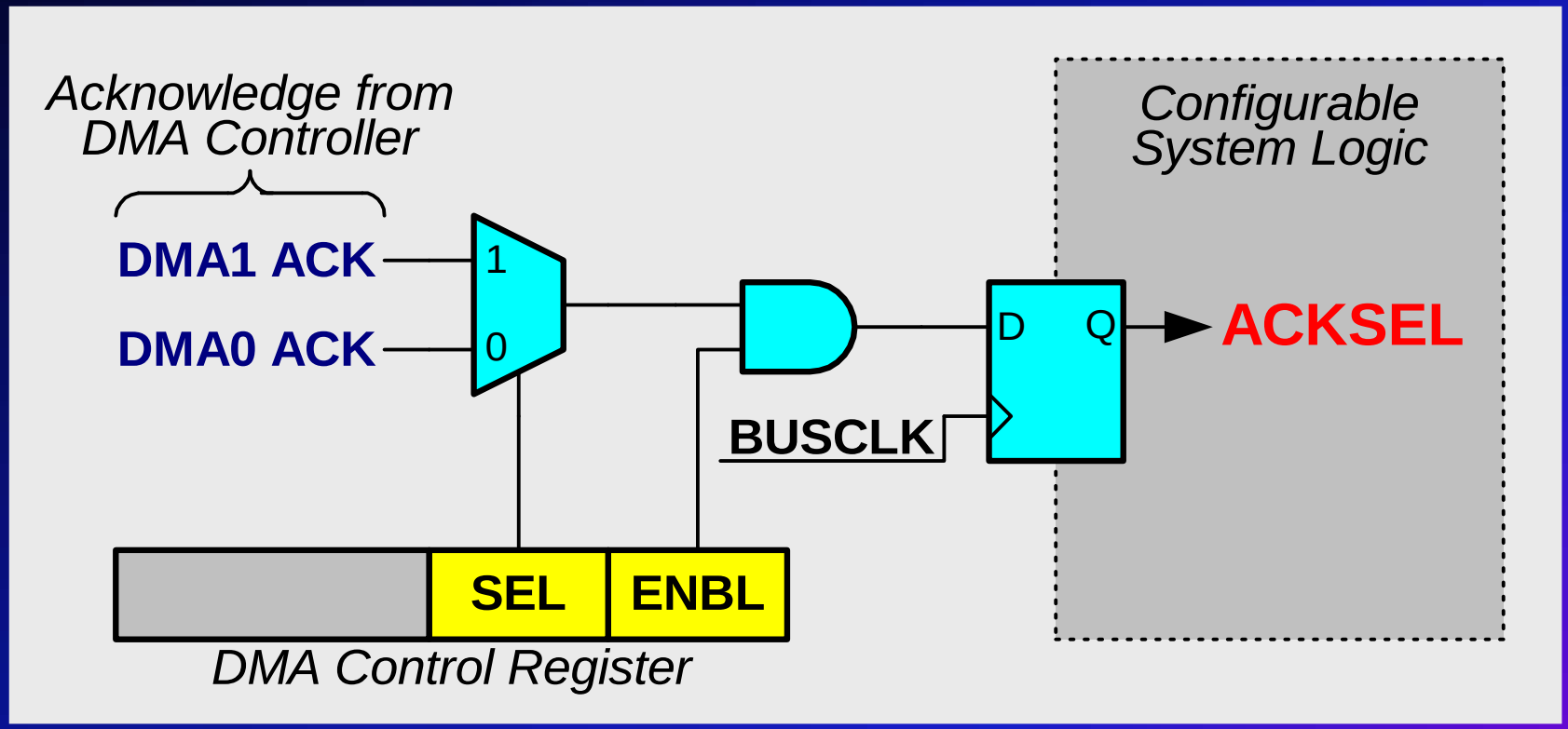
Triscend

Request Steering (REQSEL)



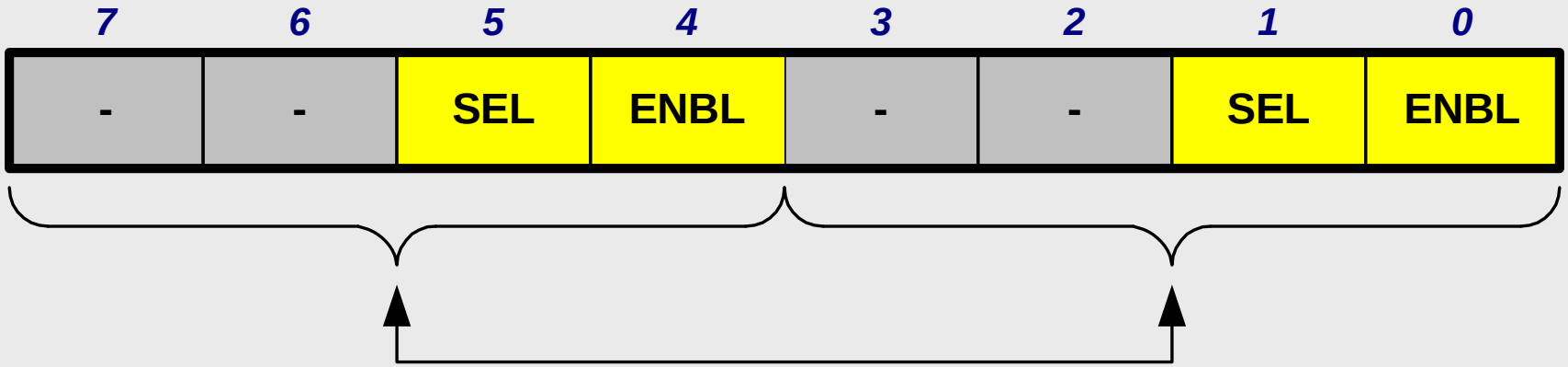
Triscend

Acknowledge Steering (ACKSEL)



Triscend

DMA Control Register Details



Write both nibbles with duplicate data.

Read register and OR bits from each nibble.

- Due to data distribution to Selectors, watch out for these details!

Triscend

DMA Control Register (VHDL)

Instantiate

```
architecture TRISCEND of MYDESIGN is

component DMACTRL
  port (
    SYMBOLIC : in std_logic;
    DATA     : in std_logic;
    CODE     : in std_logic;
    SPC1     : in std_logic;
    SPC0     : in std_logic;
    REQSEL   : in std_logic;
    ACKSEL   : out std_logic;
    SIM      : inout std_logic_vector(127 downto 0)
    );
end component;
```

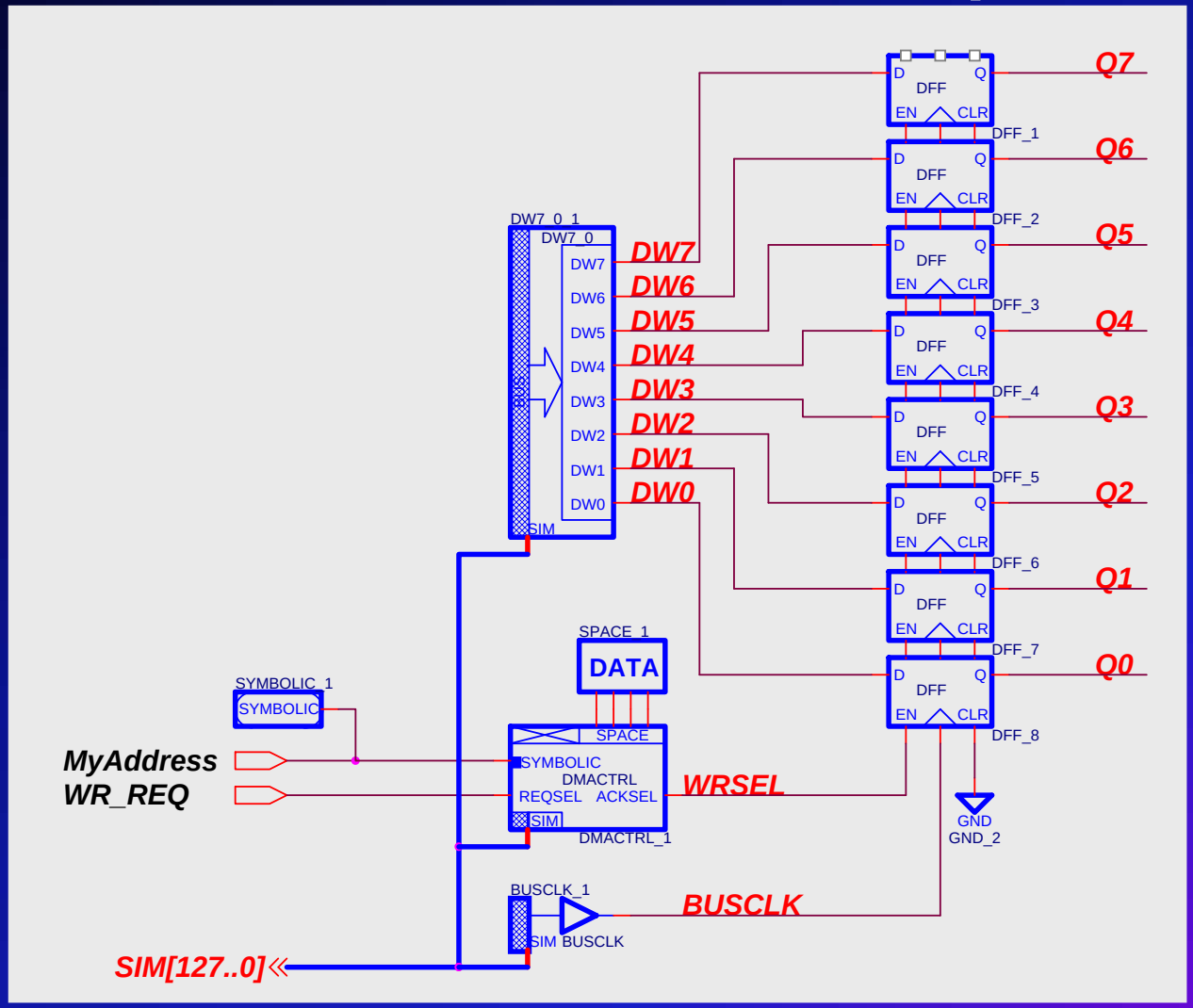
Use

```
begin
  MYSEL : DMACTRL port map (
    SYMBOLIC  => my_symbolic,
    DATA => my_data,
    CODE => my_code,
    SPC1  => my_spc1,
    SPC0  => my_spc0,
    REQSEL  => my_reqsel,
    ACKSEL  => my_acksel
  );
```

DMA Control Register (Chisel)

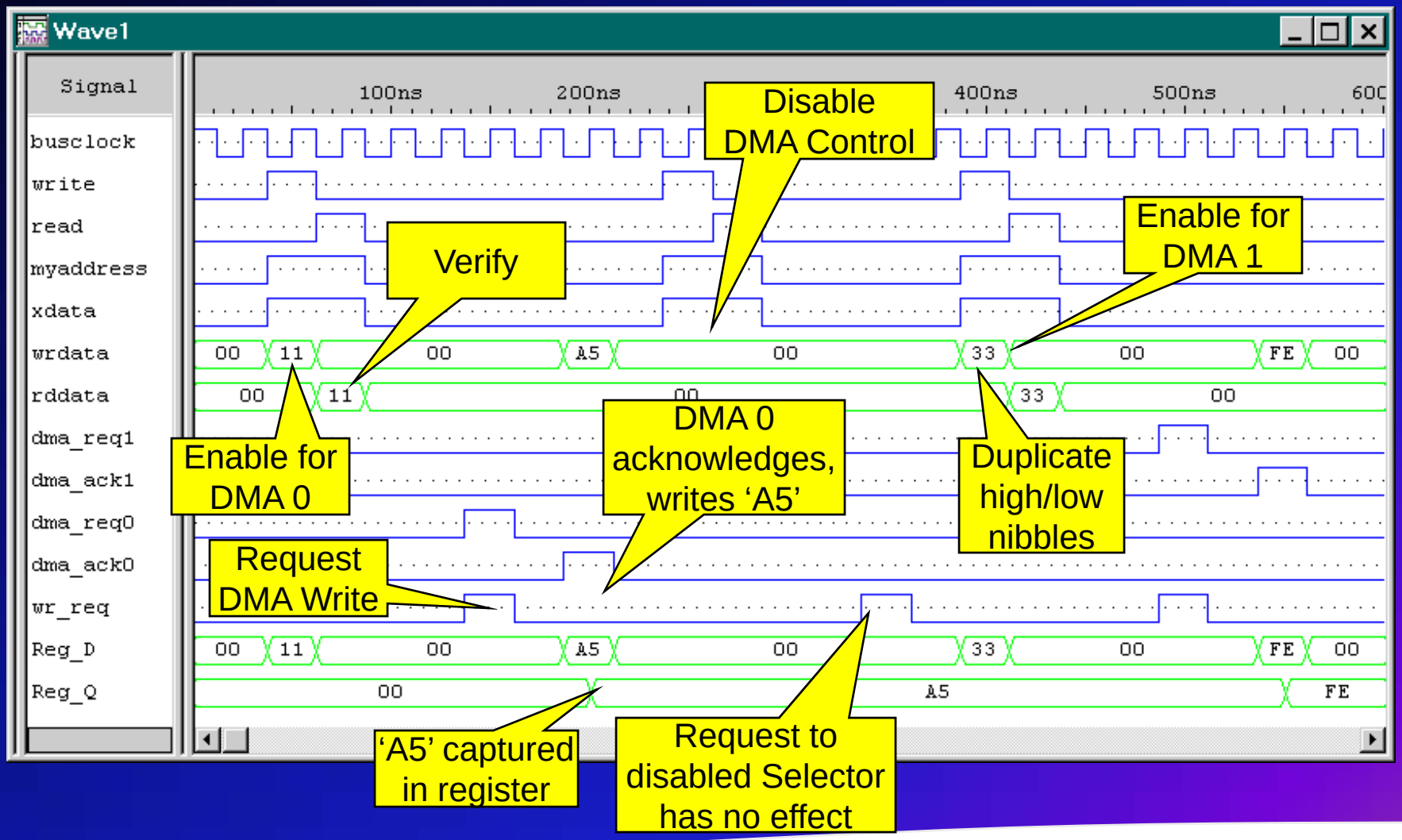
```
module example {

   parameter string my_symbolic = "MY_SYMBOLIC";

   input my_reqsel;
   output my_acksel;

   dmactrl MYSEL (
      .reqsel=my_reqsel,
      .acksel=my_acksel,
      :symbolic=my_symbolic,
      :isdataspace=true,
      :enabled=false,
      :channel=0
   );


}
```

DMA Write Example



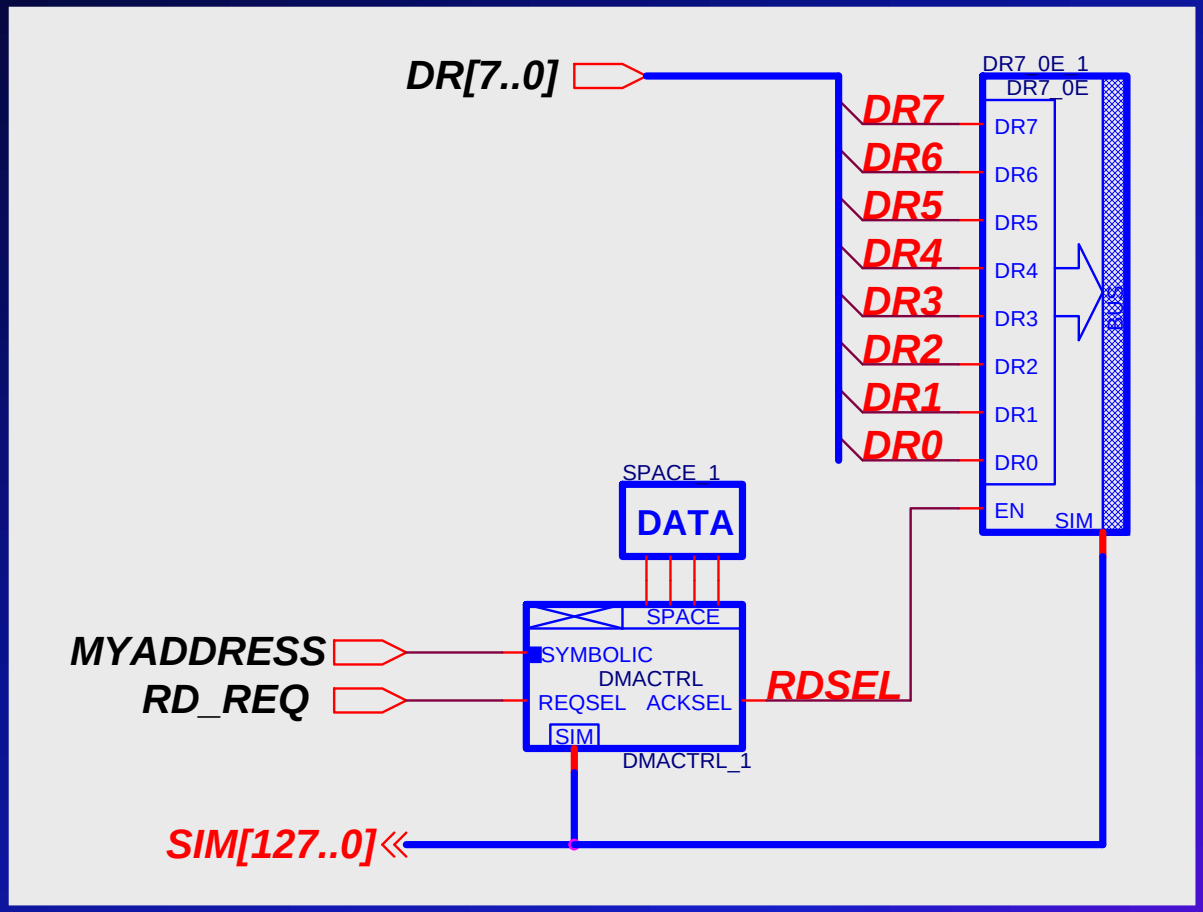
Triscend

DMA Write Simulation



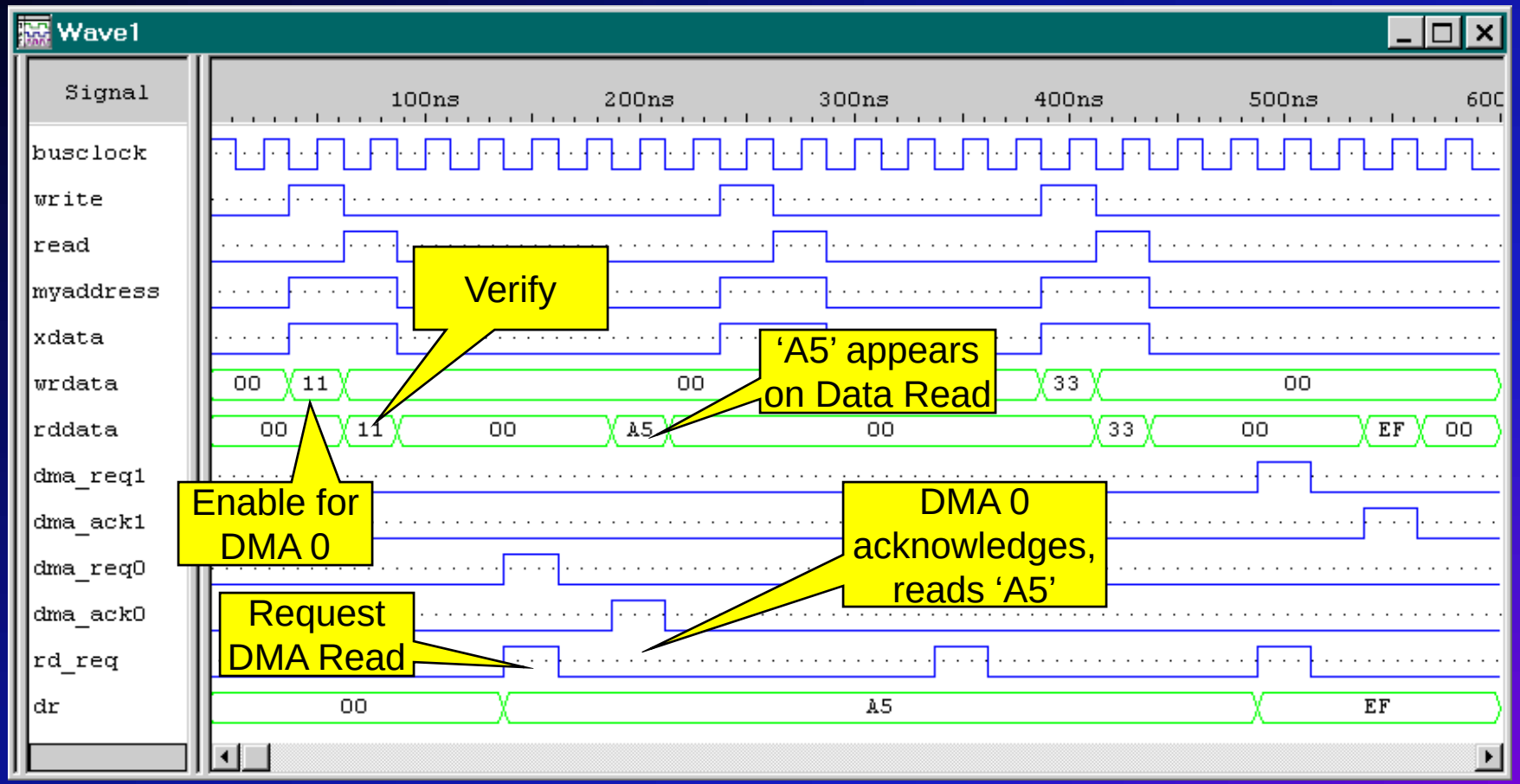
Triscend

DMA Read Example



Triscend

DMA Read Simulation



Triscend

CSI SIM Bus Models

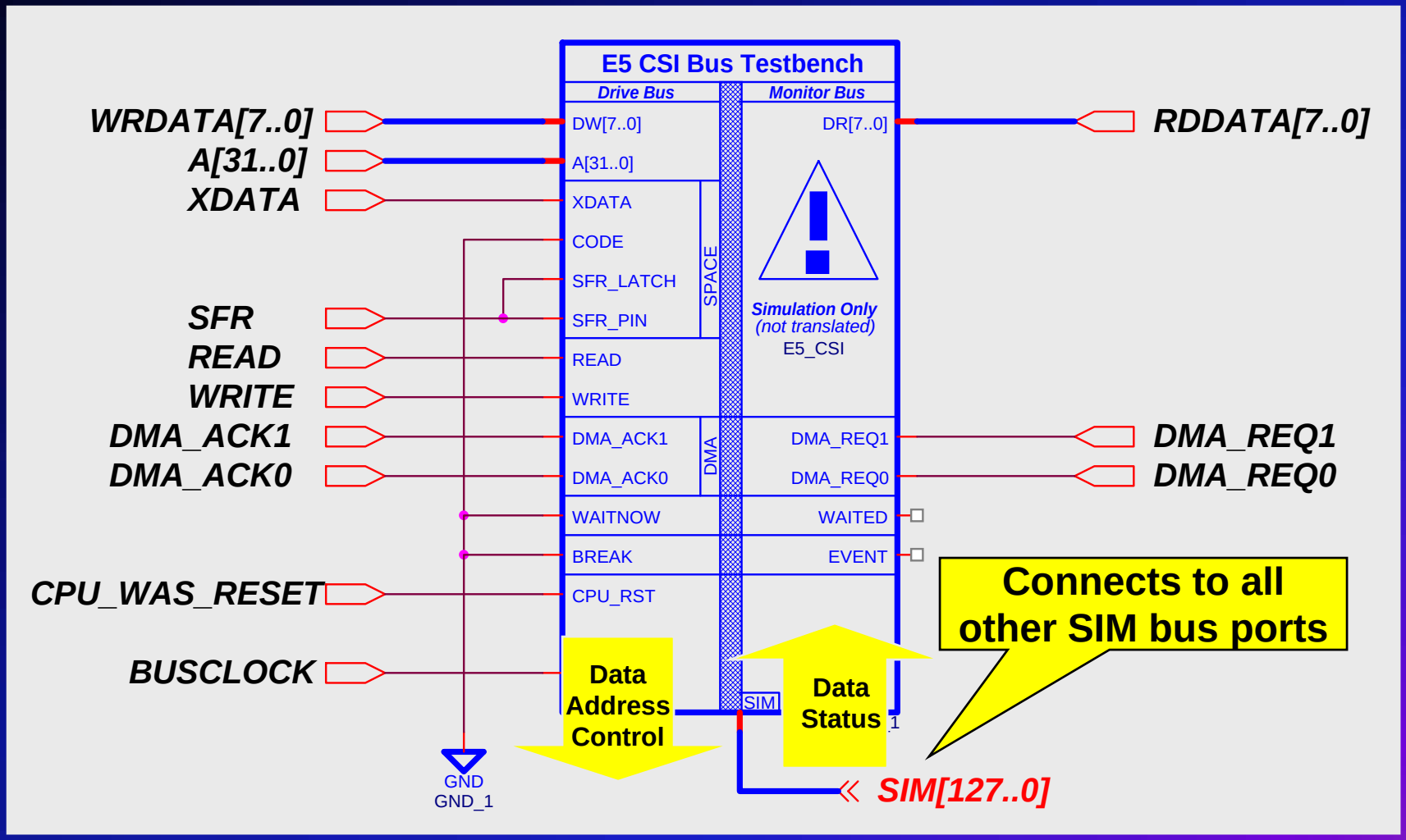
The CSI Bus Simulation Model

- What's that 128-bit bus connected to my bus symbols?
  - 128-bit bidirectional bus
  - Contains all CSI bus signals, forward compatible with 32-bit architectures
  - Simplifies simulation for CSI bus peripherals
  - Easy connection to CSI bus primitives
- E5_CSI top-level symbol drives SIM bus

CSI SIM Bus Signals

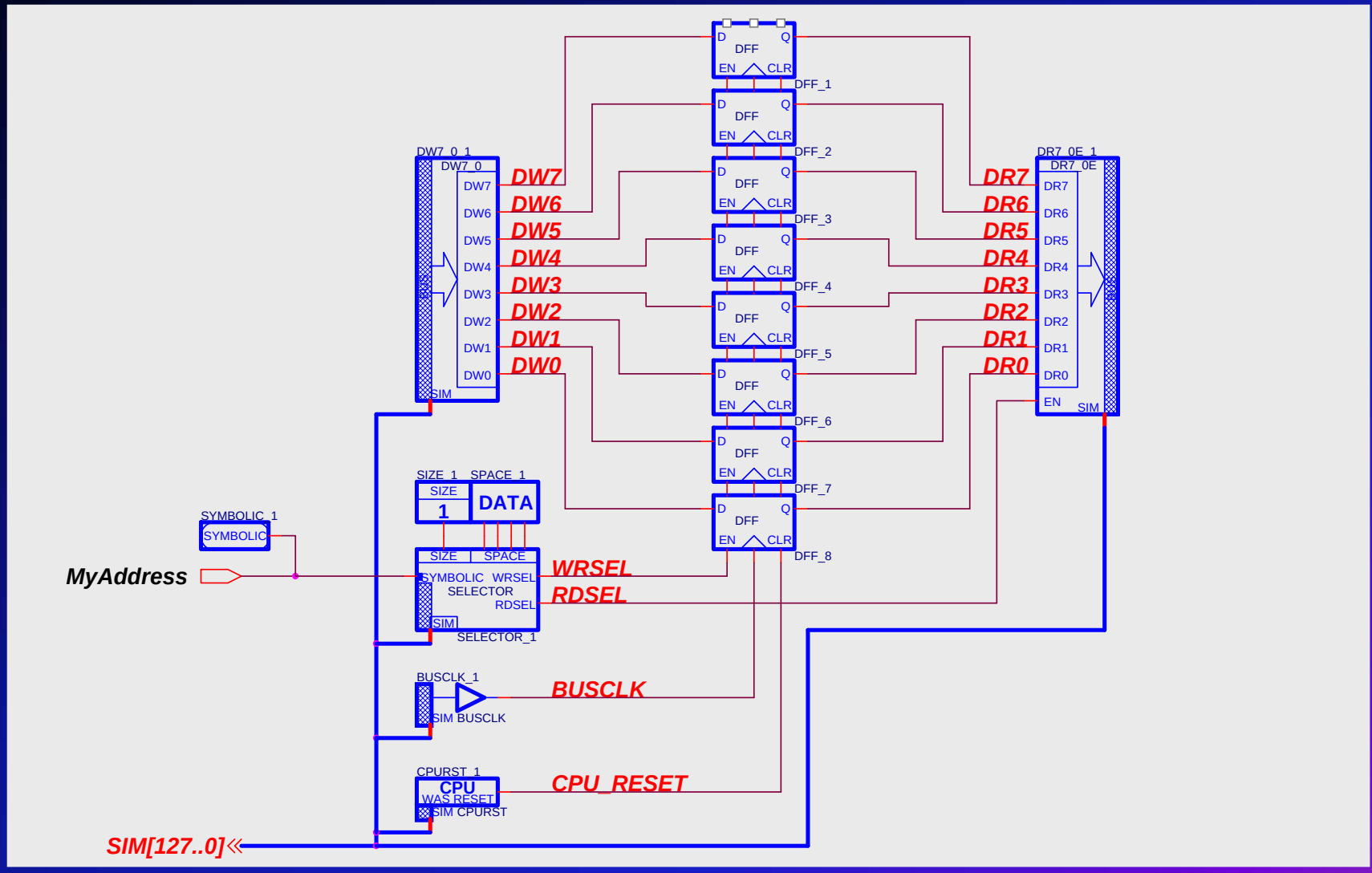
CSI Bus Signal	Simulation Bus Signal
Data Write[31:0]	SIM[31:0]
Data Read[31:0]	SIM[63:32], Wired-OR
Address[31:0]	SIM[95:64]
Bus Clock	SIM[96]
Read	SIM[97]
Write	SIM[98]
Wait	SIM[99], Wired-OR
Reserved	SIM[100]
Breakpoint	SIM[101], Wired-OR
Reserved	SIM[102]
DMAREQ[7:0]	SIM[109:103], Wired-OR
DMAACK[7:0]	SIM[117:110]
DATA access	SIM[118] (8032 – XDATA)
CODE access	SIM[119]
SPC1 access	SIM[120] (8032 – SFR latch)
SPC2 access	SIM[121] (8032 – SFR pin)
Reserved	SIM[127:122]

E5_CSI Simulation Driver



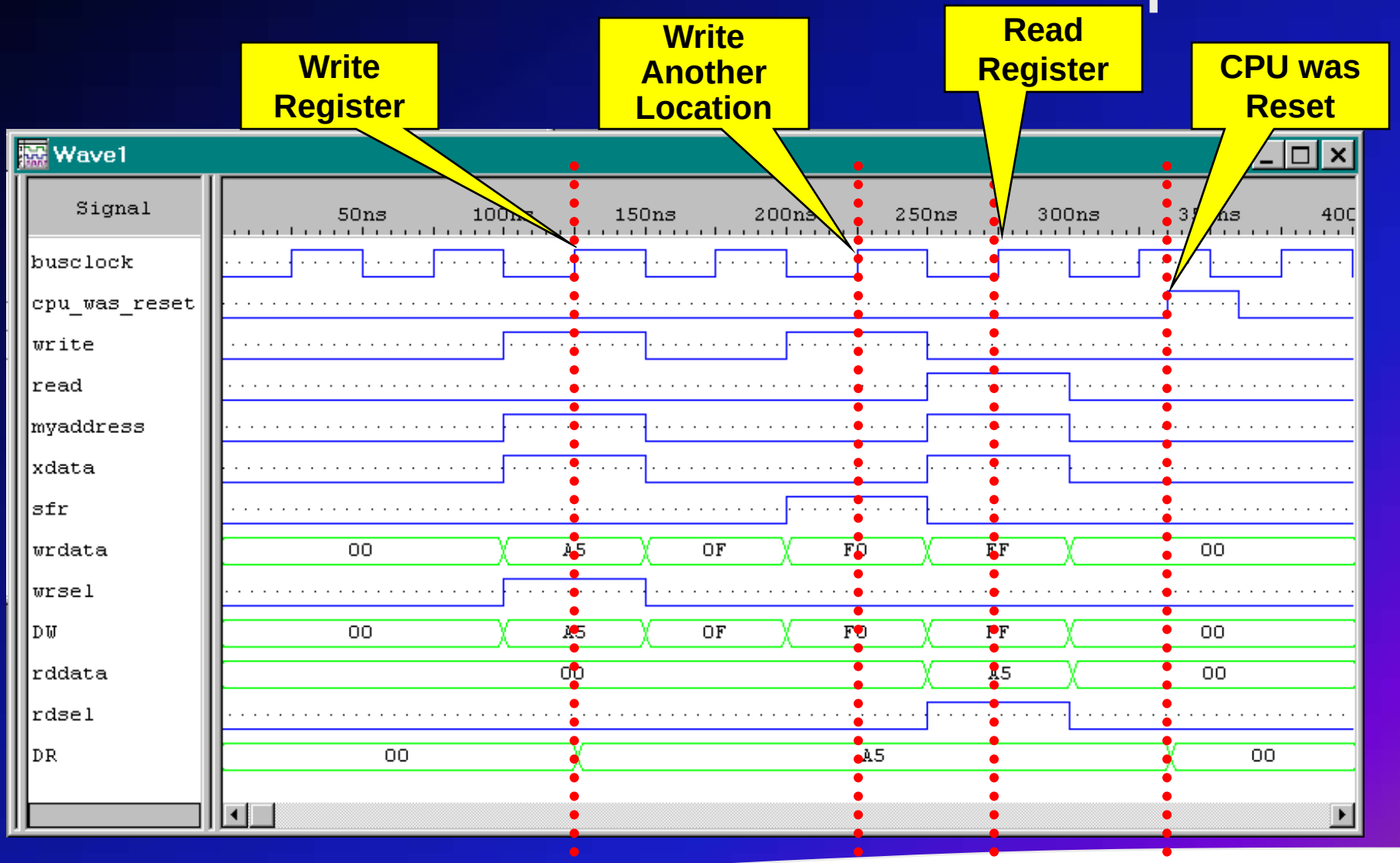
Triscend

Example: Read/Write Register



Triscend

CSI Bus Simulation Example

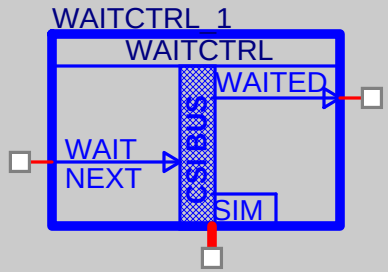


Triscend

Wait-State and Breakpoint

Wait-State Control

Schematic



- WAITNEXT holds the next bus cycle
- WAITED indicates that a wait-state happened during the last bus cycle

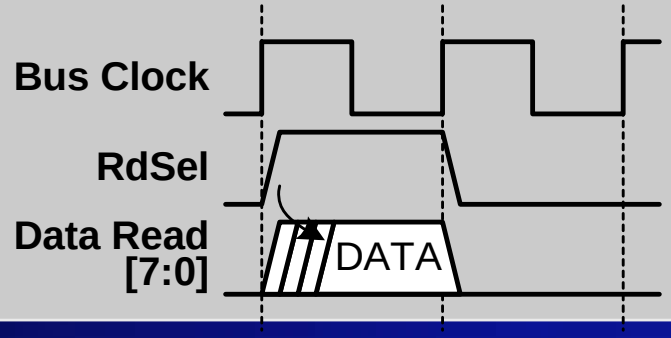
Triscend

Wait-State Rules

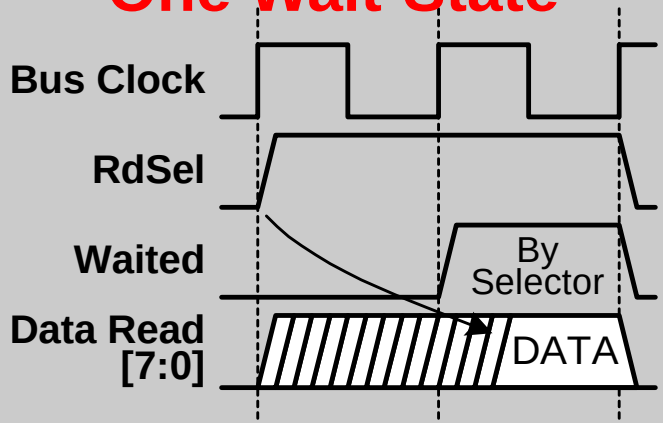
- The CSI Bus Supports Wait-States, but with a few restrictions
- If a peripheral asserts a wait-state, the first wait-state is asserted by the associated Selector
  - SELECTORW
  - CHIPSELW
  - There are no wait-states on DMA transfers

Wait-State Examples

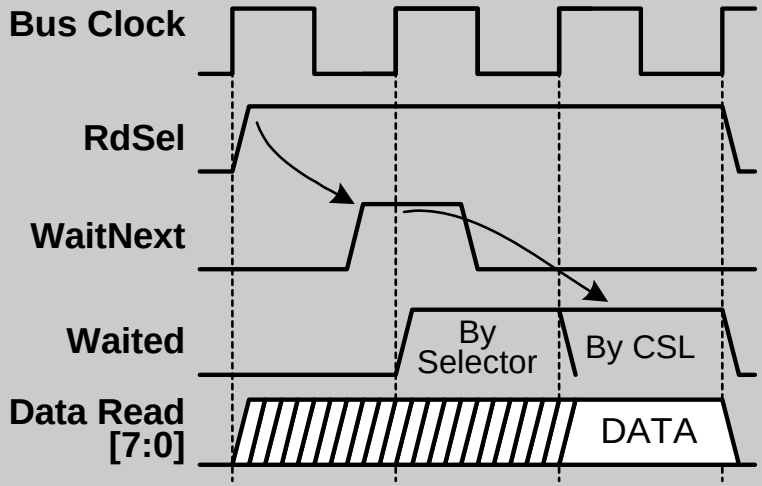
Zero Wait-States



One Wait-State



Multiple Wait-States



Triscend

Wait-State Control (VHDL)

Instantiate

```
architecture TRISCEND of MYDESIGN is

component WAITCTRL
  port (
        WAITNEXT  : in      std_logic;
        WAITED    : out     std_logic;
        SIM       : inout std_logic_vector(127 downto 0)
        );
end component;
```

Use

```
begin
   MYWAIT : WAITCTRL port map (
      WAITNEXT => my_waitnext,
      WAITED   => my_waited
   );
```

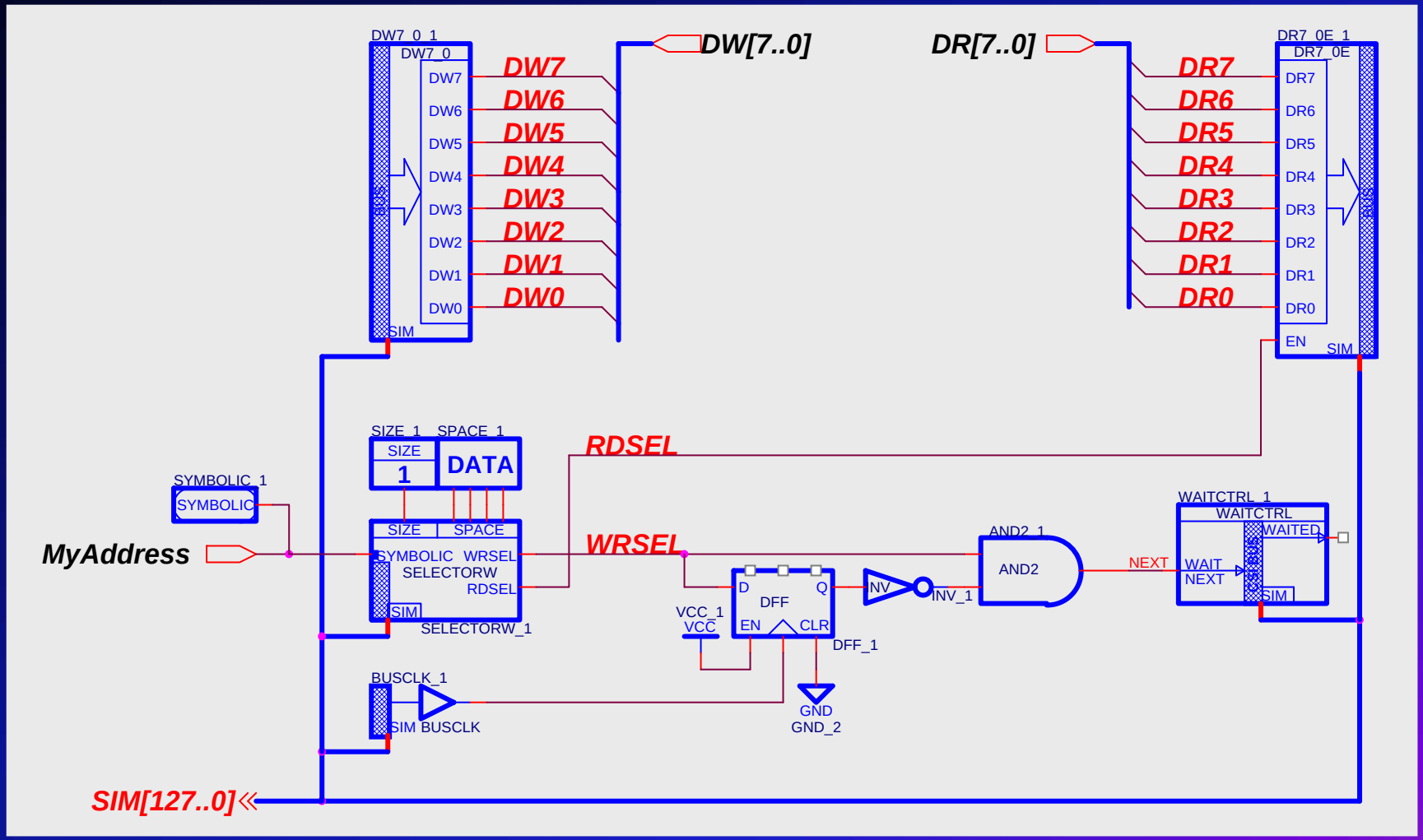
Wait-State Control (Chisel)

```
module example {

   input my_waitnext;
   output my_waited;

   waitnext (.i=my_waitnext);
   waited (.o=my_waited);

}
```

Example Wait-State Design

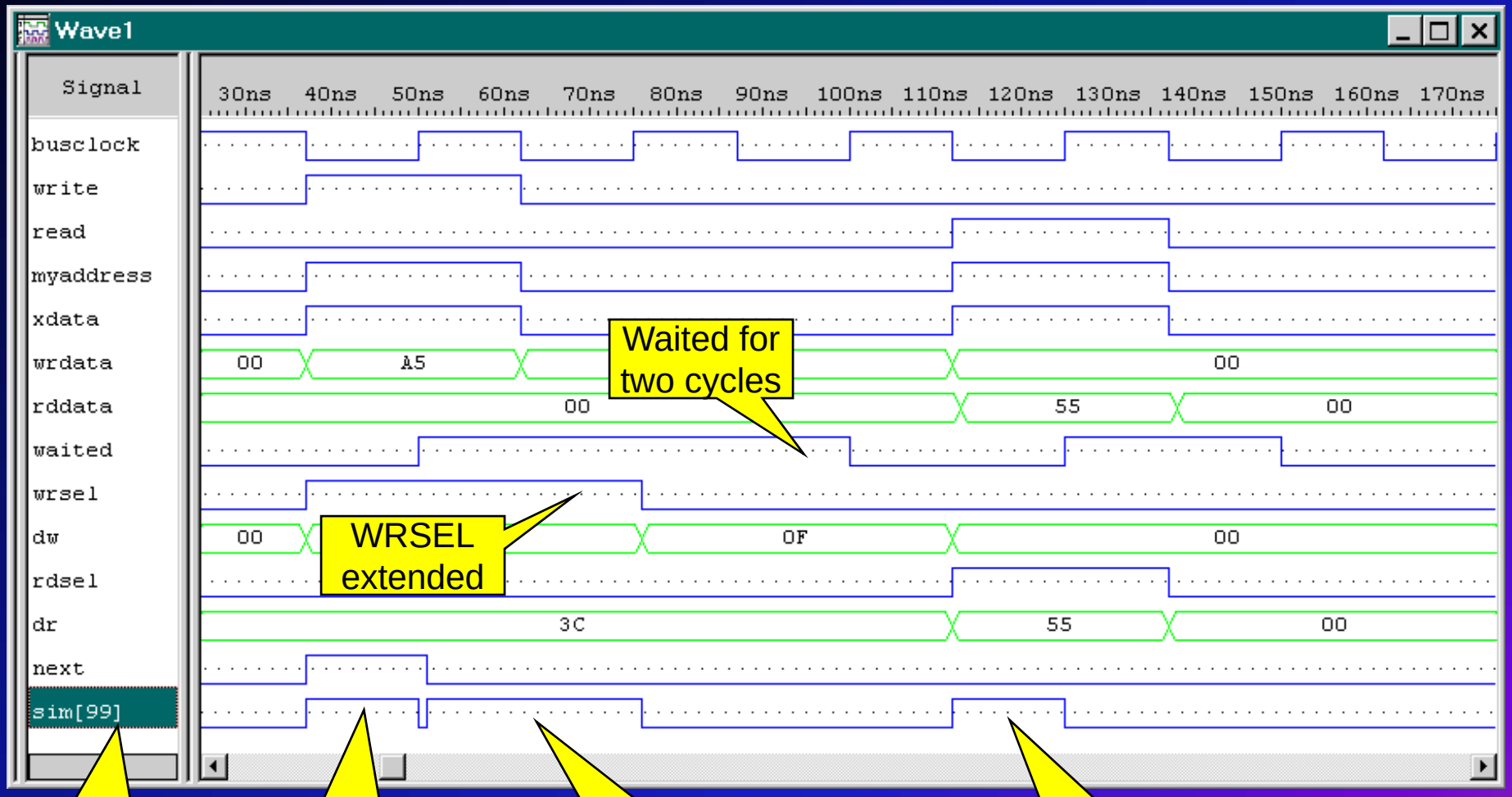
- Peripheral supports single-cycle reads but requires a three-cycle write
- CSI wait-state mechanism requires at least one wait-state if you have any wait-states

Example Wait-State Schematic



Triscend

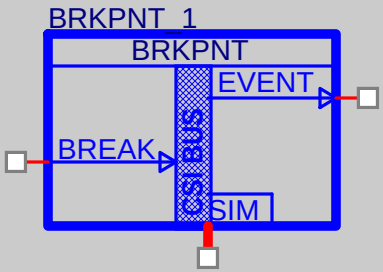
Example Wait-State Simulation



Triscend

Hardware Breakpoint

Schematic



- BREAK causes a hardware breakpoint
- EVENT indicates that a hardware breakpoint is happening

Triscend

Hardware Breakpoint (VHDL)

Instantiate

```
architecture TRISCEND of MYDESIGN is

component BRKPNT
  port (
        BREAK  : in     std_logic;
        EVENT  : out    std_logic;
        SIM    : inout std_logic_vector(127 downto 0)
        );
end component;
```

Use

```
begin
   MYBRK : BRKPNT port map (
      BREAK  => my_break,
      EVENT  => my_event
   );
```

Hardware Breakpoint (Chisel)

```
module example {

   input my_break;
   output my_event;

   bpctl BREAK (.i=my_break);
   bpevt EVENT (.o=my_event);
}
```

Summary

- The Configurable System Interconnect bus provides easy access to “soft” peripherals
  - Data, Address, Control
  - Address Decoding
  - Simple, synchronous protocol

Questions?
Comments?
Feedback?