AARHUS
UNIVERSITY
SCHOOL OF ENGINEERING

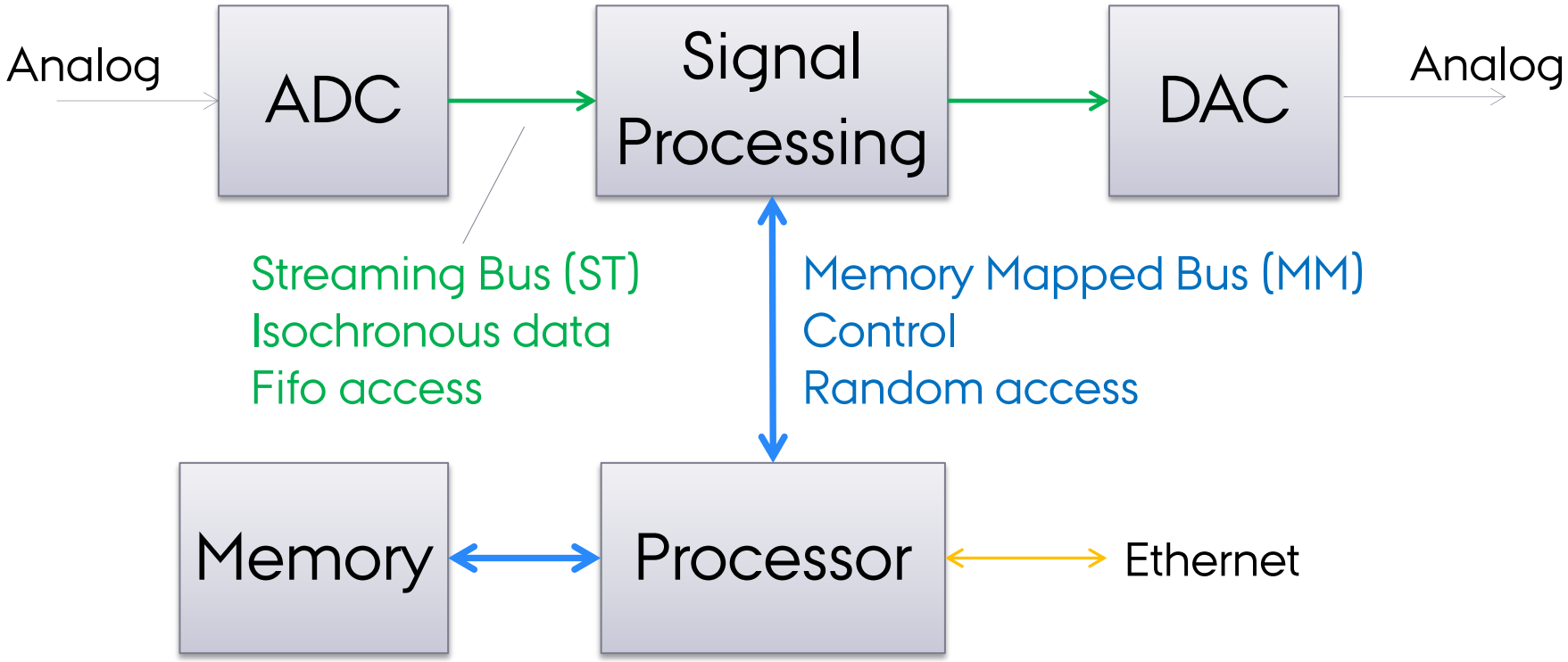
PETER HØGH MIKKELSEN
26. JANUARY, 2016

SOPC MEMORY MAPPED BUSSES

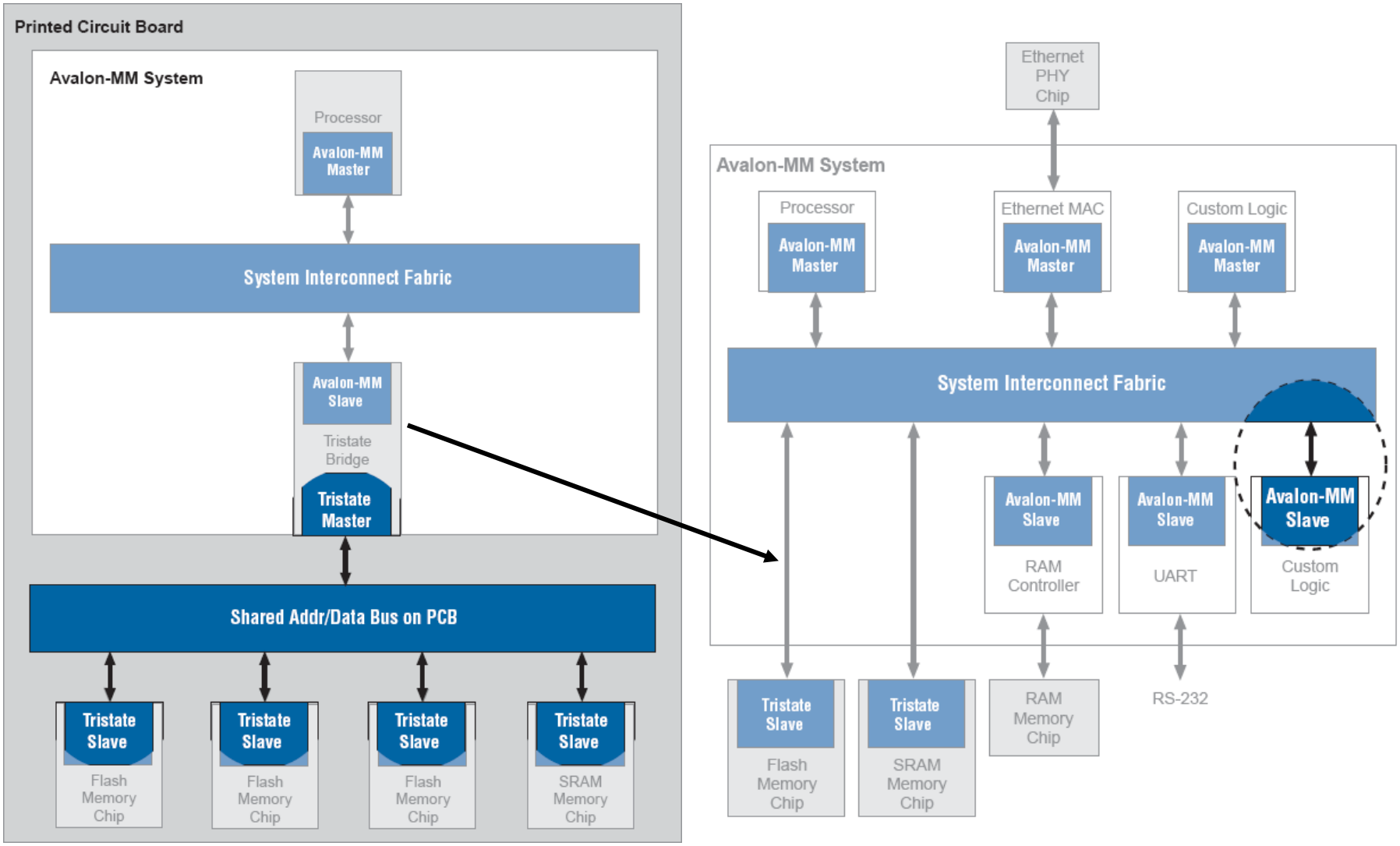
AGENDA

- Avalon Bus types
- System Interconnect Fabric

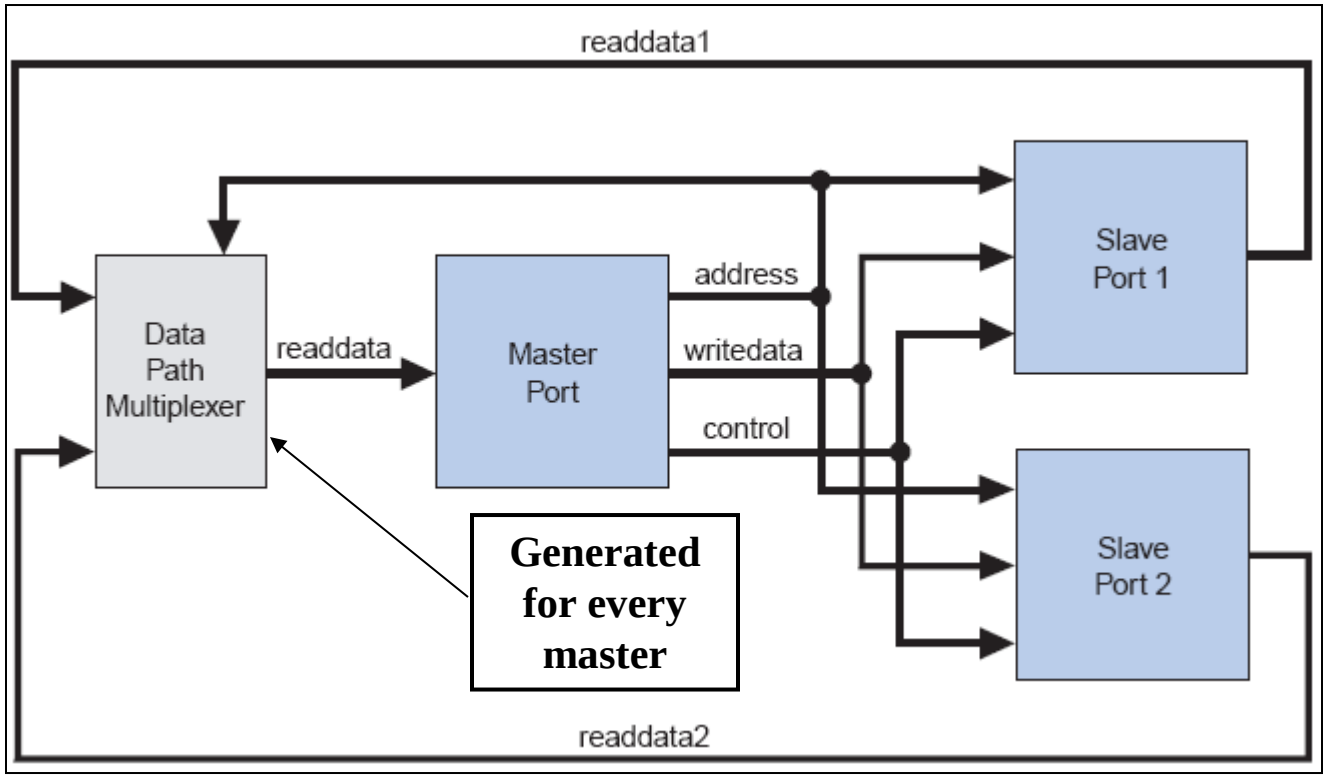
BUS TYPES


Analog
ADC
Signal Processing
DAC
Analog
Streaming Bus (ST)
Isochronous data
Fifo access
Memory Mapped Bus (MM)
Control
Random access
Memory
Processor
Ethernet

AVALON MM BUS



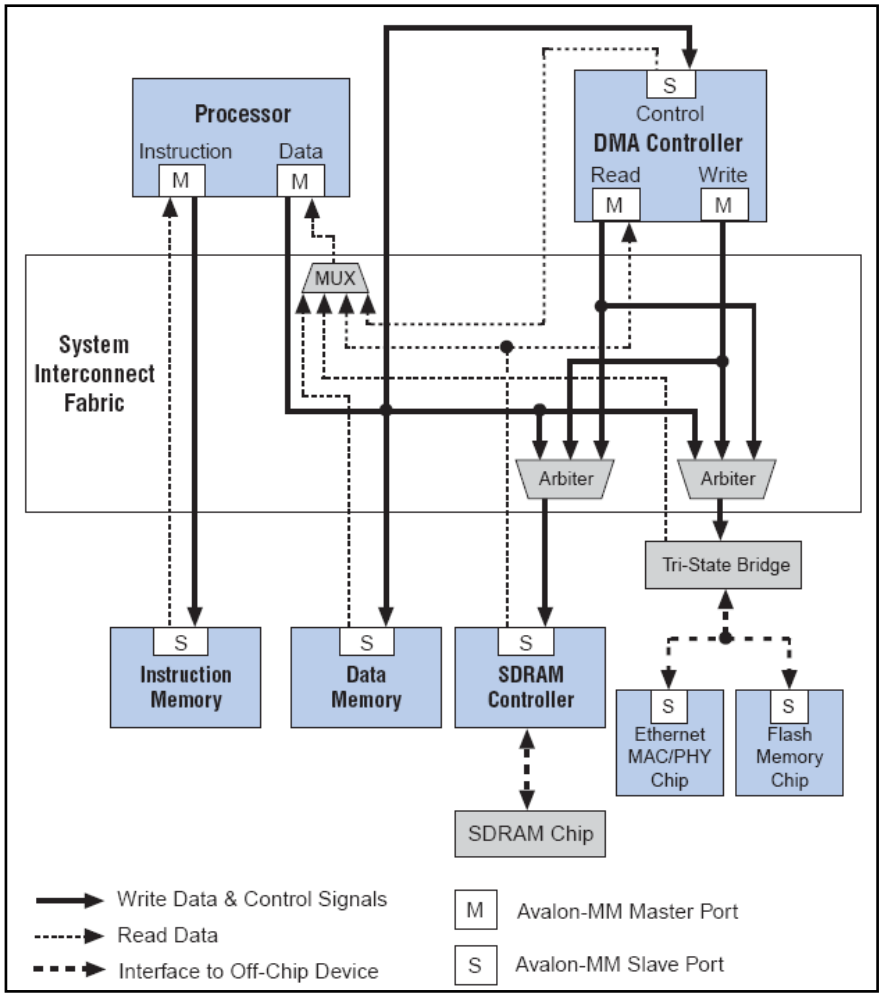
DATAPATH MULTIPLEXING



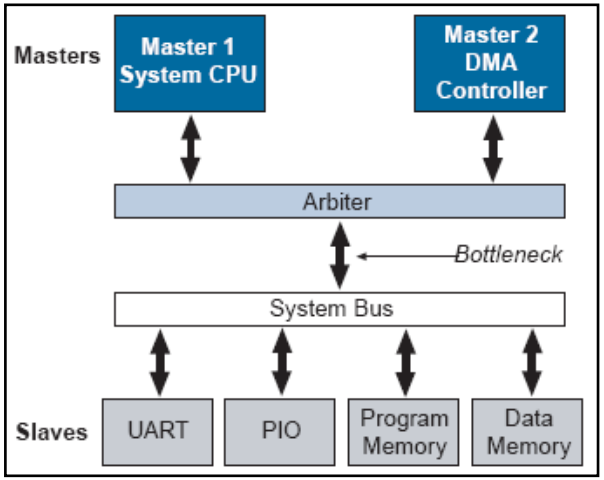
› Shows a block diagram of the data path multiplexing logic for one master and two slaves

SWITCH FABRIC FOR AVALON MM

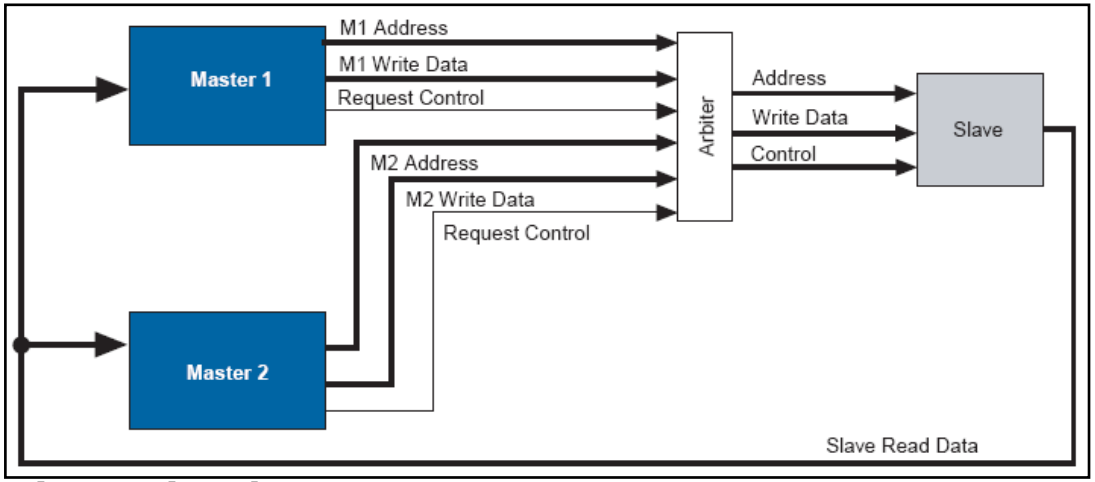
- Any number of master and slave components
- Interface to off/on chip devices.
- Different data width of master and slaves.
- Components operating in different clock domains
- Components using multiple Avalon-MM ports.



ABITRATION FOR MULTIMASTER



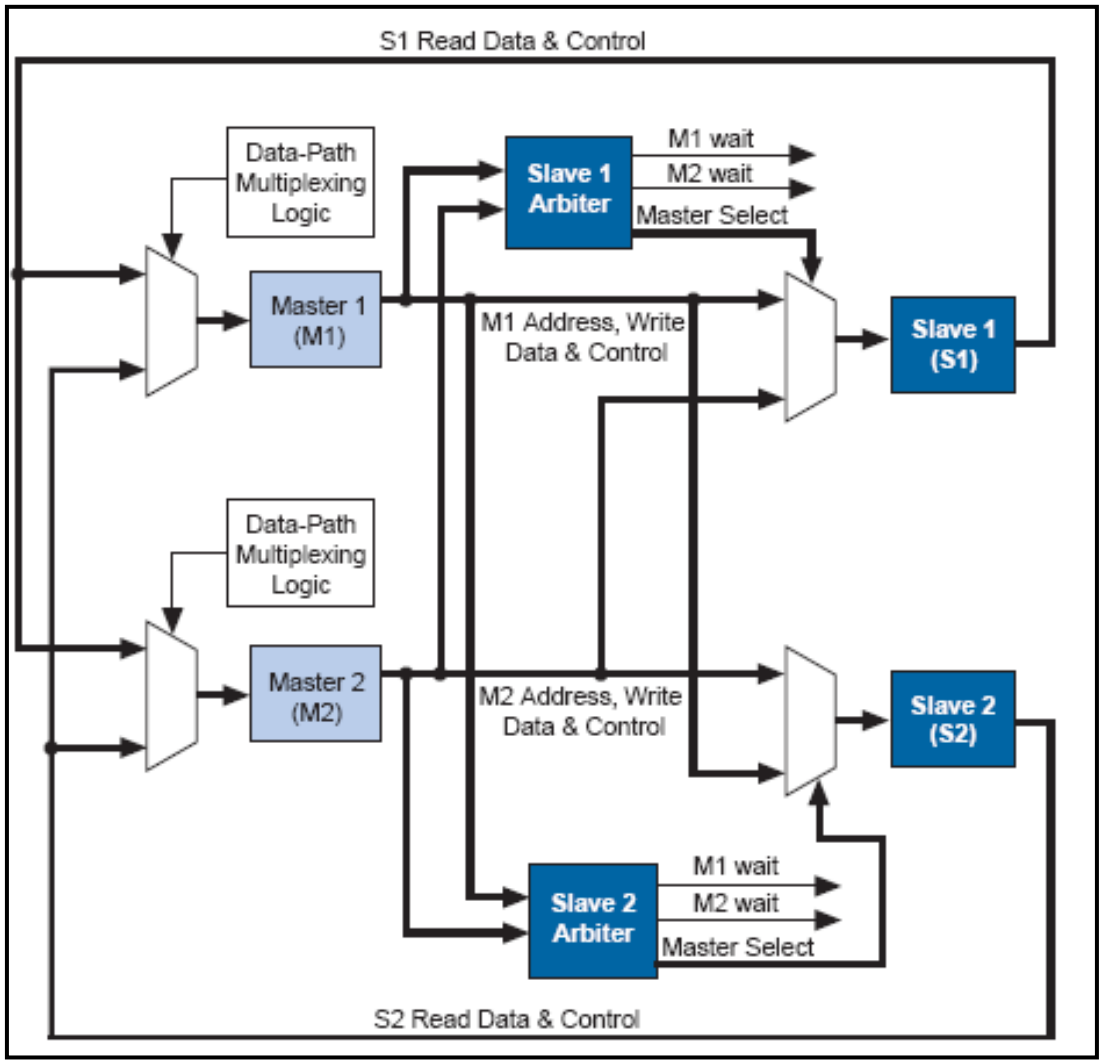
Traditionally arbitration



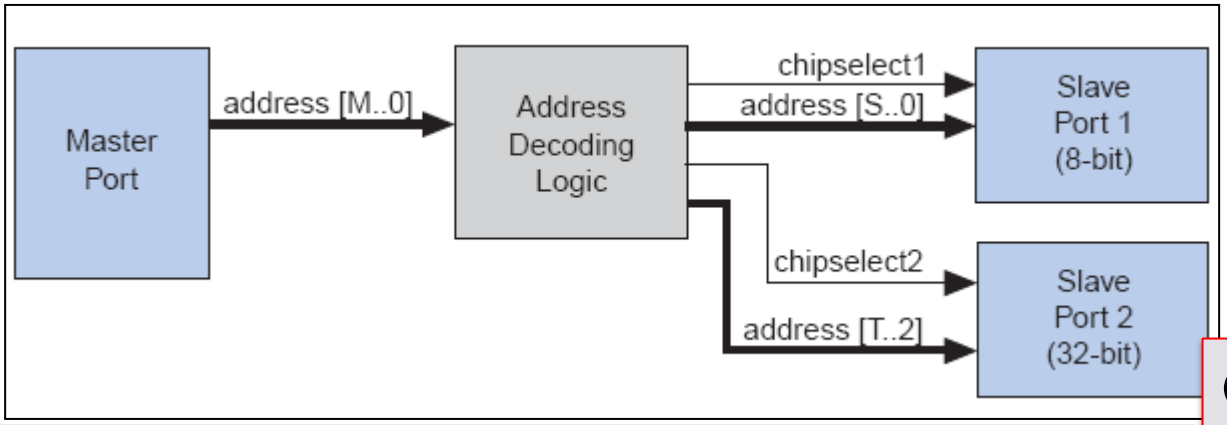
Slave-Side Arbitration

- QSys generates an arbiter for every slave, that:
- Evaluates the address and control signals from each master and determines which master, if any, gains access to the slave next.
- Grants access to the chosen master and forces all other requesting masters to wait.
- Use multiplexers to connect address, control, and data paths between the multiple masters and the slave.

EXAMPLE - MULTIMASTER



ADDRESS DECODING (1:2)



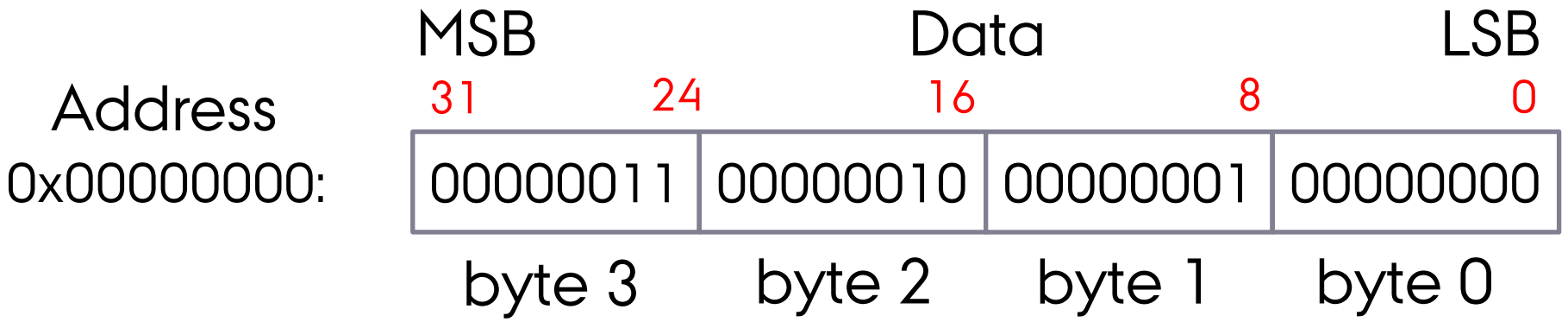
Module Name	Description	Base	End	IRQ
cpu	Nios II Proces...			
instruction_master	Master port			
data_master	Master port	IRQ 0	IRQ 31	
jtag_debug_mod...	Slave port	0x02120000	0x021207FF	
ext_flash	Flash Memory...	0x0000 0000	0x007F FFFF	
ext_ram	IDT71V416 S...	0x02000000	0x020FFFFF	
ext_ram_bus	Avalon Tri-St...			
button_pio	PIO (Parallel I/O)	0x02120860	0x0212086F	2
high_res_timer	Interval timer	0x02120820	0x0212083F	3

Chip Select active in this range

Local addr range

QSys handles both address decoding and different slave size

ADDRESSING (1:2)



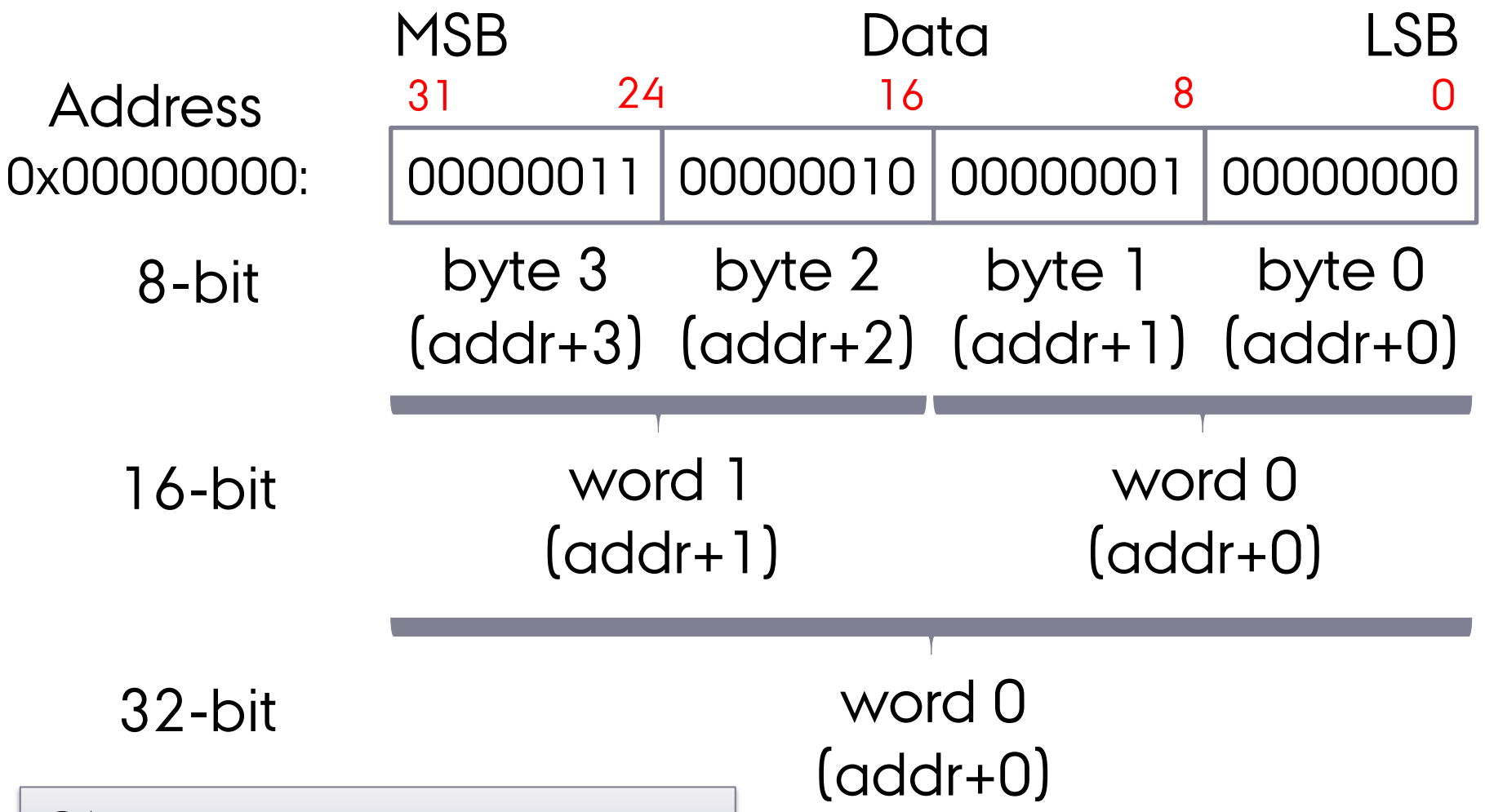
LITTLE ENDIAN:

```
u8 *ptr_8 = 0x00000000;
*ptr_8 = 0x00;
*(ptr_8+1) = 0x01;
*(ptr_8+2) = 0x02;
*(ptr_8+3) = 0x03;
```

BIG ENDIAN:

```
u8 *ptr_8 = 0x00000000;
*ptr_8 = 0x03;
*(ptr_8+1) = 0x02;
*(ptr_8+2) = 0x01;
*(ptr_8+3) = 0x00;
```

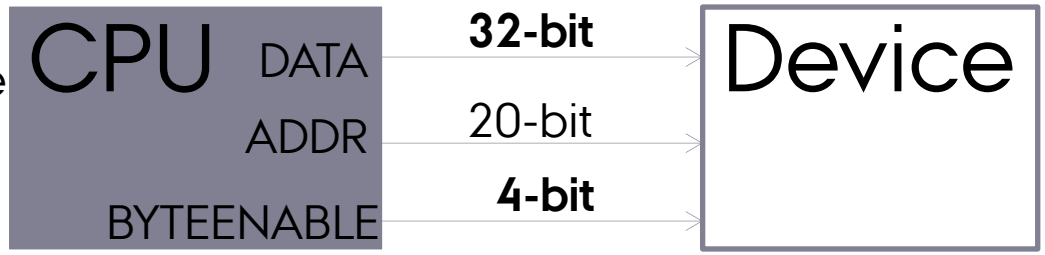
ADDRESSING (2:2)

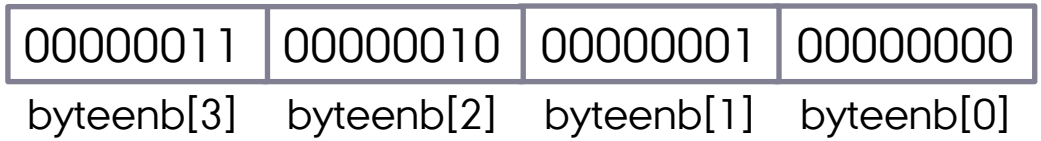


ONLY Little Endian shown

BYTE ENABLE (1:2)

Example:
32-bit Interface
LITTLE ENDIAN



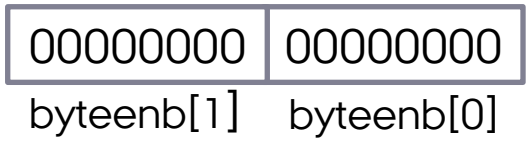



```
u32 *ptr_32 = 0x00000000;  // 32-bit pointer
ptr_32++; // byteaddr +=4;
value = *ptr_32; // BYTEENABLE = b1111, ADDR = 0x00000001

u16 *ptr_16 = 0x00000000; // 16-bit pointer
ptr_16++; // byteaddr +=2
value = *ptr_16; // BYTEENABLE = b1100, ADDR = 0x00000000

u8 *ptr_8 = 0x00000000; // 8-bit pointer
ptr_8++; // byteaddr +=1
value = *ptr_8; // BYTEENABLE = b0010, ADDR = 0x00000000
```

BYTE ENABLE (2:2)

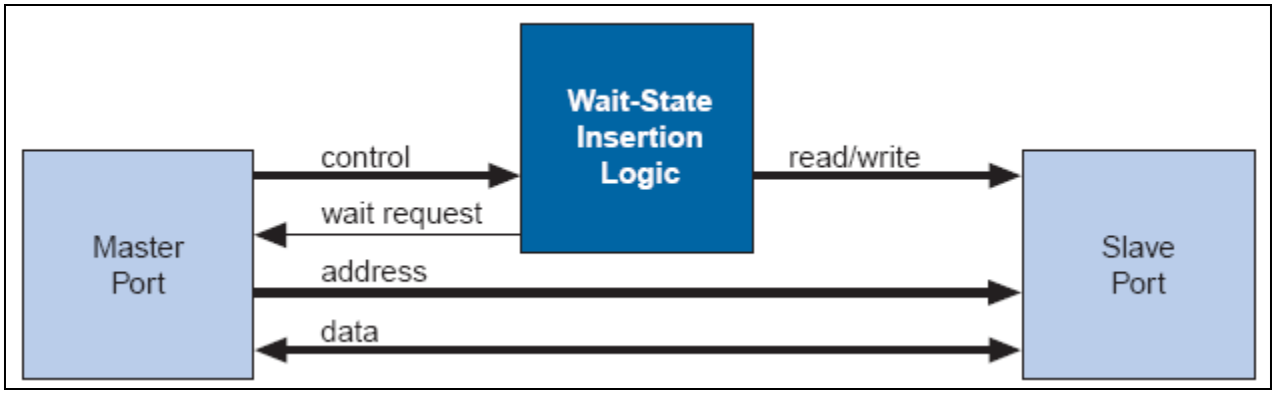
Example:
16-bit Interface
LITTLE ENDIAN

0x00000000:



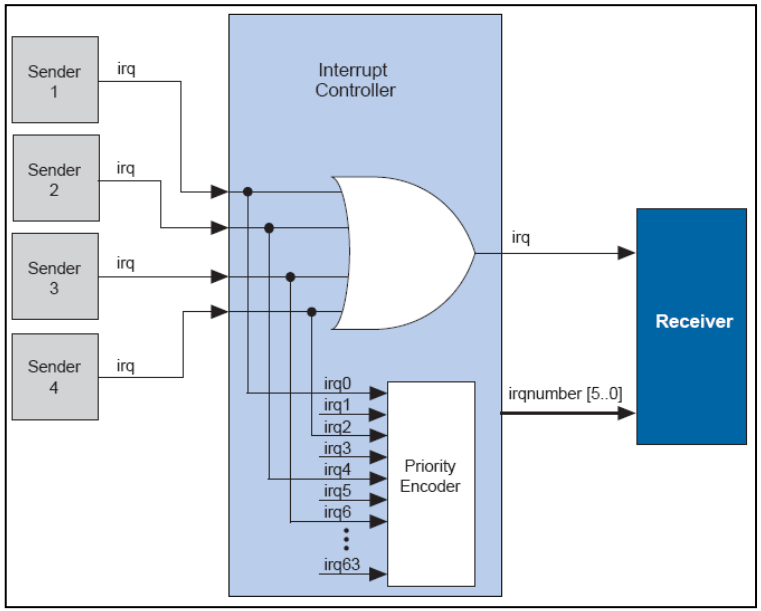

```
u32 *ptr_32 = 0x00000000;  // 32-bit pointer
ptr_32++; // byteaddr +=4;
value = *ptr_32; // BYTEENABLE = b11, ADDR = 0x00000004&2 (two reads)

u16 *ptr_16 = 0x00000000; // 16-bit pointer
ptr_16++; // byteaddr +=2
value = *ptr_16; // BYTEENABLE = b11, ADDR = 0x00000002

u8 *ptr_8 = 0x00000000; // 8-bit pointer
ptr_8++; // byteaddr +=1
value = *ptr_8; // BYTEENABLE = b10, ADDR = 0x00000000
```

INSERTING WAITSTATES

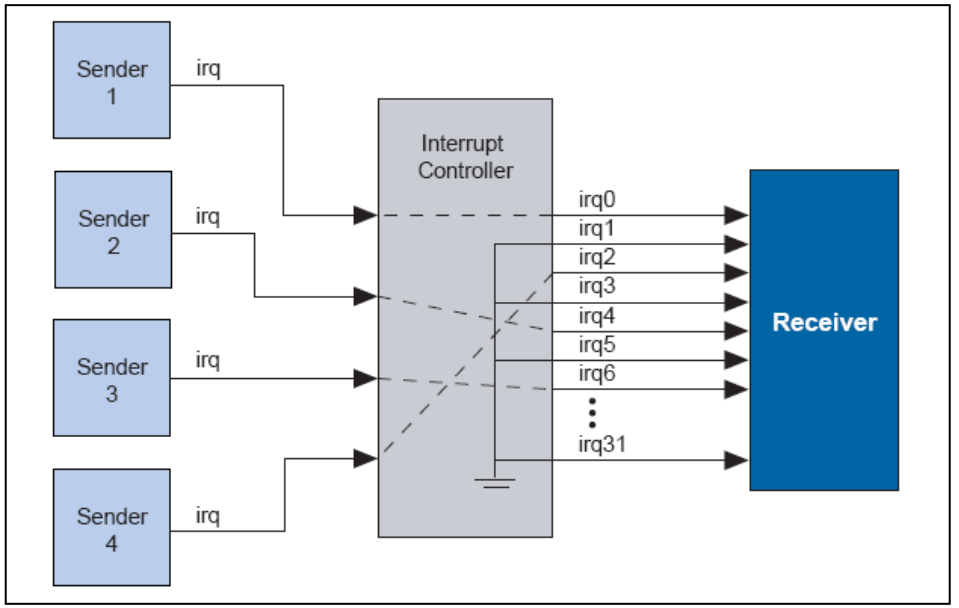


› QSys generates wait state insertion logic based on the properties of all slaves in the system.

INTERRUPT



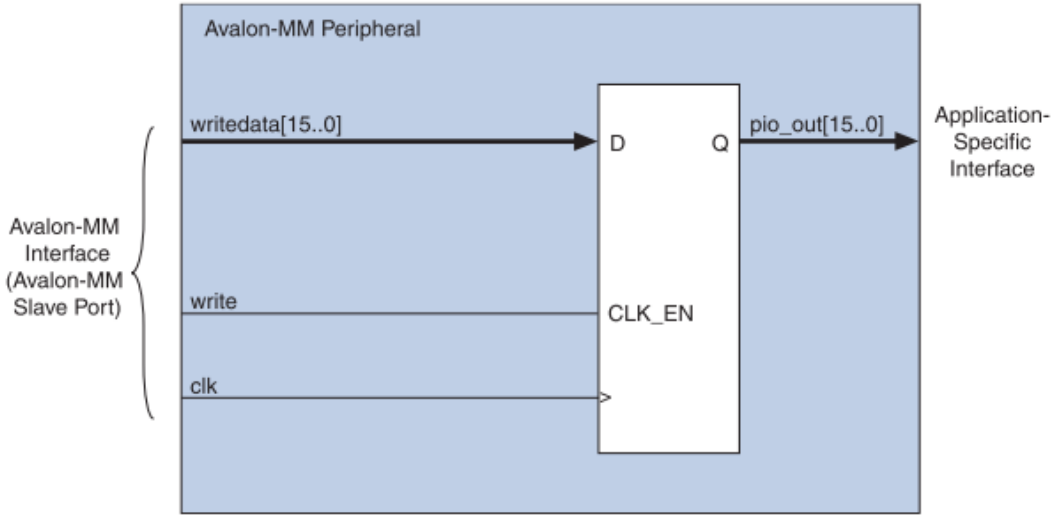
IRQ Mapping Using Hardware Priority



IRQ Mapping Using Software Priority

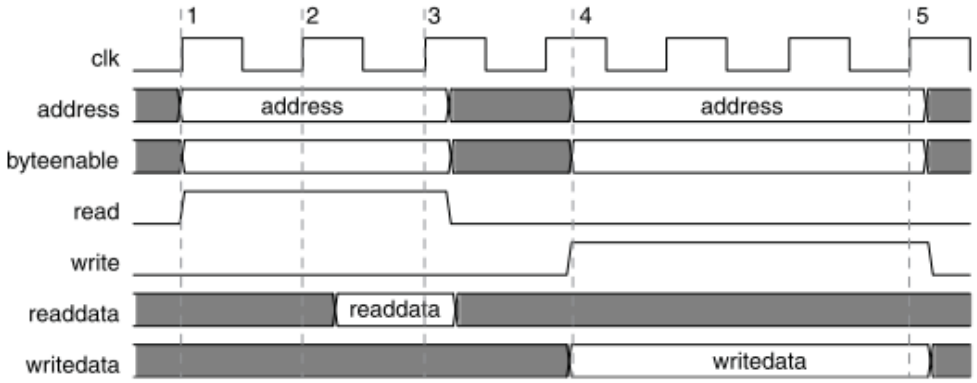
- A separate interrupt controller is generated for each interrupt *receiver*.
- An Avalon-MM slave can only include one interrupt *sender*.
- Interrupt receivers can be either of the two illustrated above depending of the *irqScheme* property setting.

SIMPLEST (OUTPUT) PERIPHERAL

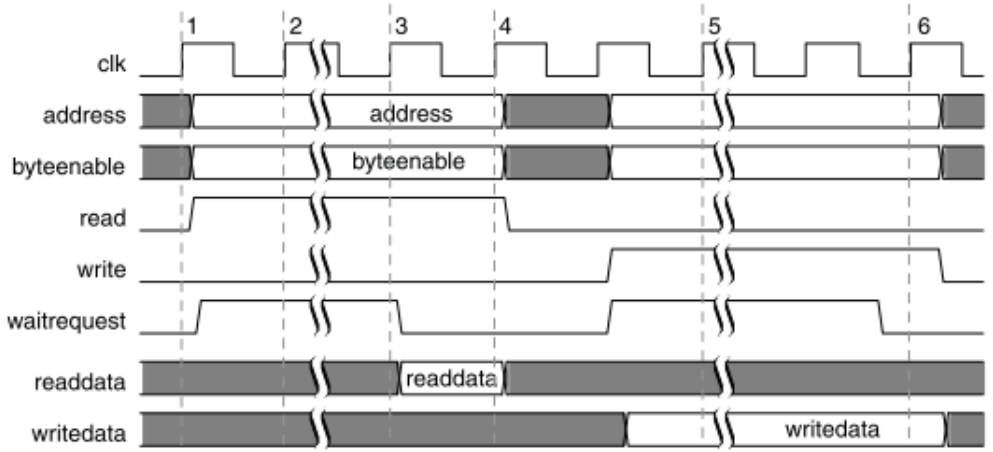


READ- /WRITE TRANSFERS

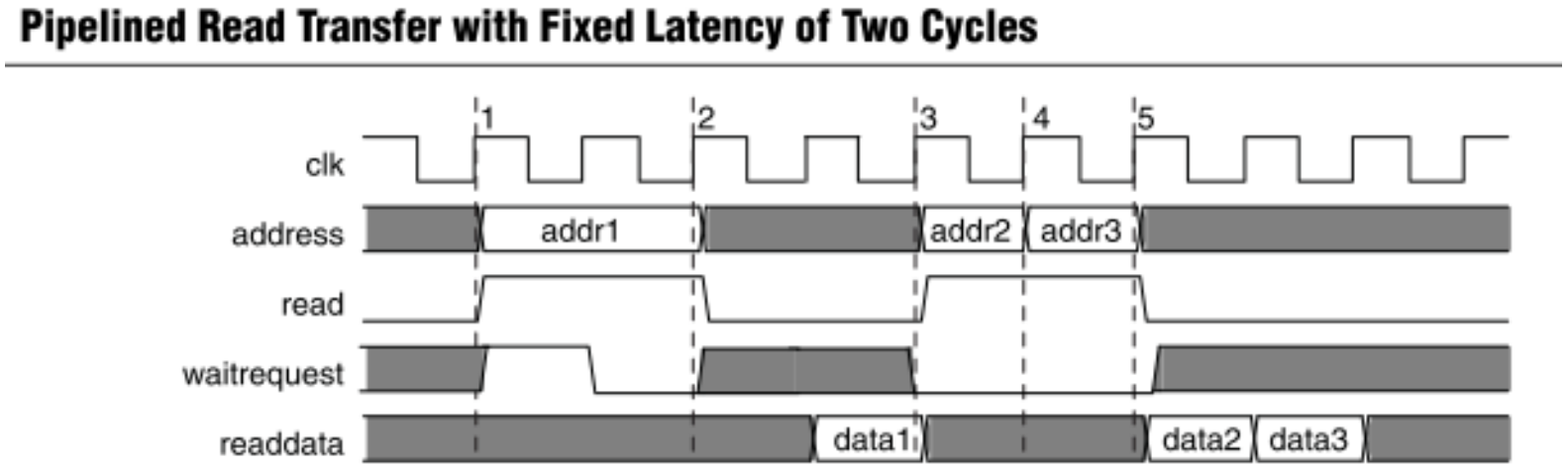
Read and Write Transfer with Fixed Wait-States at the Slave Interface



Read and Write Transfers with Waitrequest



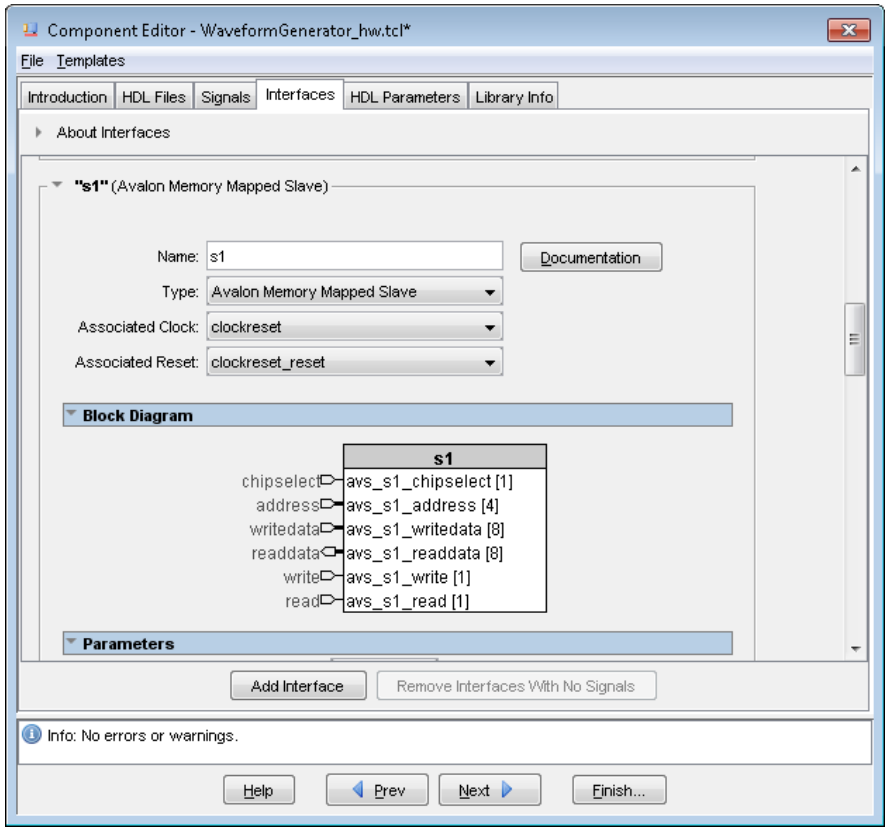
PIPELINED TRANSFER

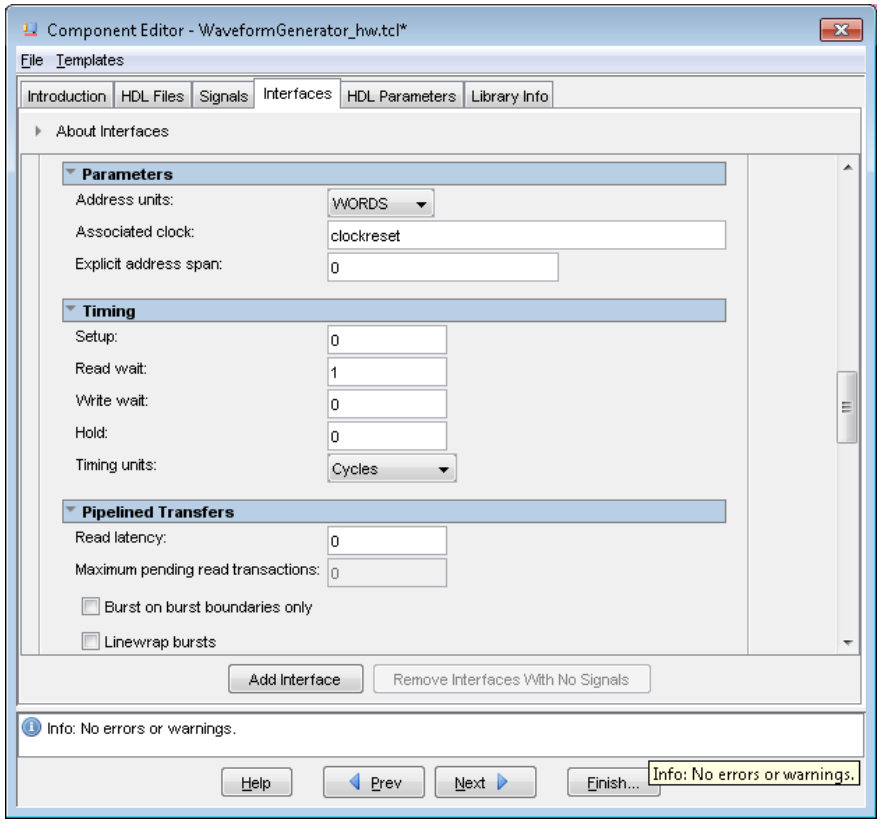


SOPC MM EXAMPLE (1:3)

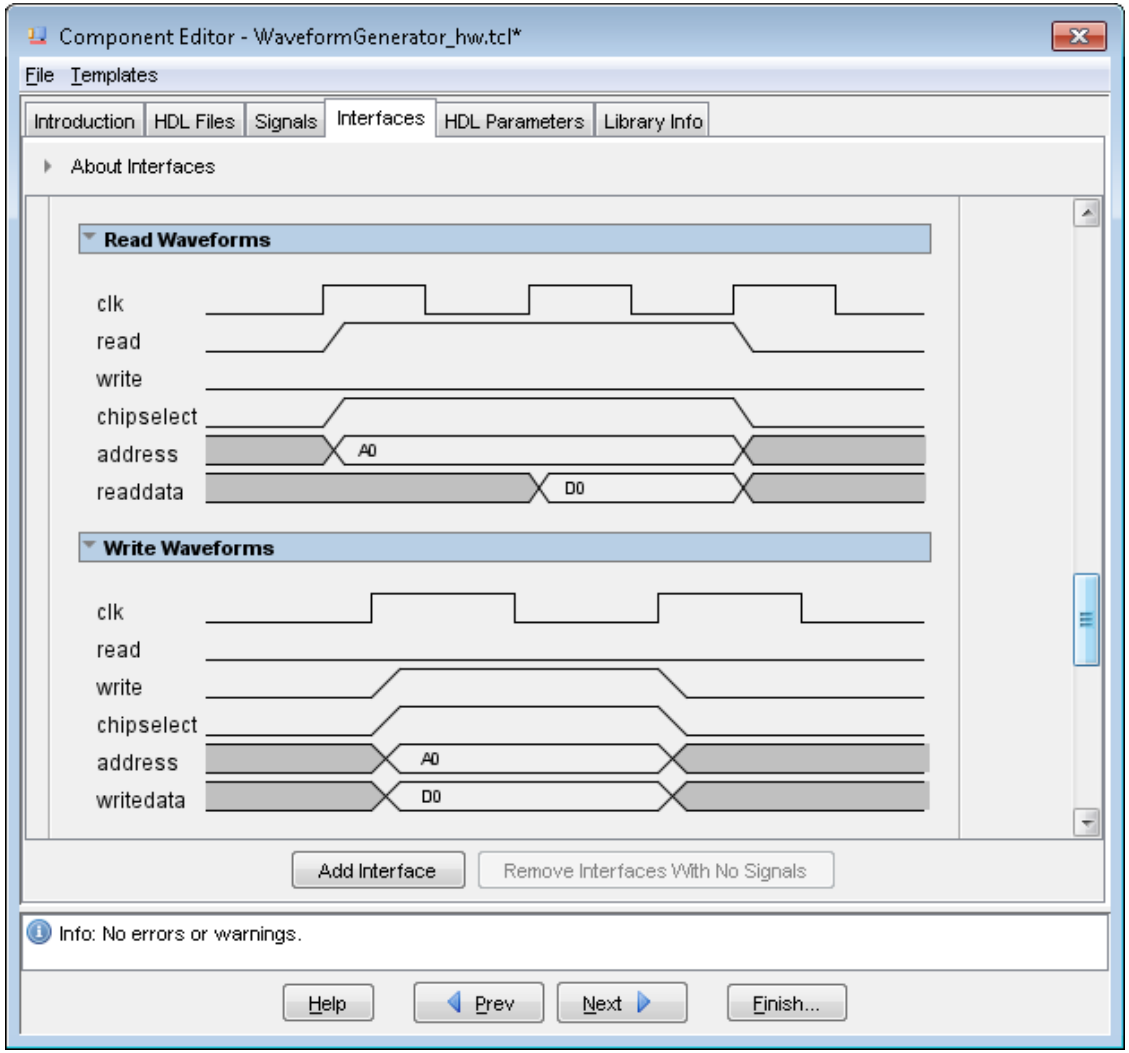
```
entity my_avalon_mm_slave is

  port (
    -- Avalon Interface
    csi_clockreset_clk     : in  std_logic;  -- Avalon Clk
    csi_clockreset_reset_n : in  std_logic;  -- Avalon Reset
    avs_s1_write           : in  std_logic;  -- Avalon wr
    avs_s1_read            : in  std_logic;  -- Avalon rd
    avs_s1_chipselect      : in  std_logic;
    avs_s1_byteenable      : in  std_logic_vector(3 downto 0);
    avs_s1_address         : in  std_logic_vector(3 downto 0);  -- Avalon address
    avs_s1_writedata       : in  std_logic_vector(7 downto 0);  -- Avalon wr data
    avs_s1_readdata        : out std_logic_vector(7 downto 0));  -- Avalon rd data

end my_avalon_mm_slave;

architecture behaviour of my_avalon_mm_slave is

  -- VHDL BODY CODE
...

end behavior;
```

SOPC MM EXAMPLE (2:3)





SOPC MM EXAMPLE (3:3)

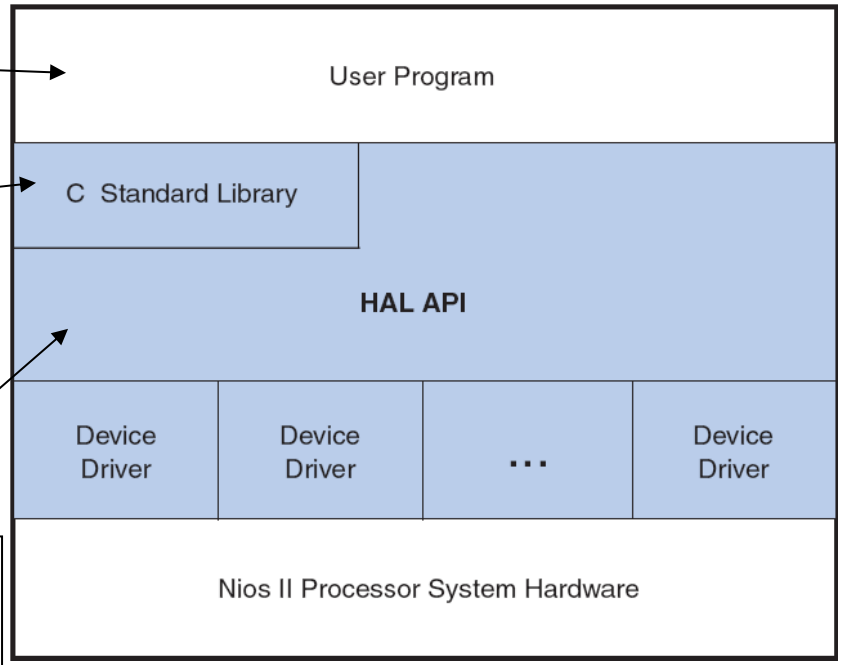


HAL

Application code

Unix functions like:

fopen, fprintf, printf etc.



The **_regs.h** header file defines the following access macros for the component:

- Register access macros that provide a read and/or write macro for each register in the component that supports the operation. The macros are:
  - IORD_*<component name>_<register name>* *(<component base address>)*
  - IOWR_*<component name>_<register name>* *(<component base address>, <data>)*

  For example, **altera_avalon_jtag_uart_regs.h** defines the following macros:
  - `IORD_ALTERA_AVALON_JTAG_UART_DATA()`
  - `IOWR_ALTERA_AVALON_JTAG_UART_DATA()`
  - `IORD_ALTERA_AVALON_JTAG_UART_CONTROL()`
  - `IOWR_ALTERA_AVALON_JTAG_UART_CONTROL()`

EXCERPTS FROM SYSTEM.H

```
/*
 * sys_clk_timer configuration
 *
 */

#define SYS_CLK_TIMER_NAME "/dev/sys_clk_timer"
#define SYS_CLK_TIMER_TYPE "altera_avalon_timer"
#define SYS_CLK_TIMER_BASE 0x00920800
#define SYS_CLK_TIMER_IRQ 0
#define SYS_CLK_TIMER_ALWAYS_RUN 0
#define SYS_CLK_TIMER_FIXED_PERIOD 0

/*
 * jtag_uart configuration
 *
 */

#define JTAG_UART_NAME "/dev/jtag_uart"
#define JTAG_UART_TYPE "altera_avalon_jtag_uart"
#define JTAG_UART_BASE 0x00920820
#define JTAG_UART_IRQ 1
```

PIO

```
while(1) {
  /* Output a 8-bit value to the LEDs */
  IOWR ALTERA AVALON PIO DATA( LEDS BASE, (led val & 0xFF) );
  switches = IORD_ALTERA_AVALON_PIO_DATA( SWITCHES_BASE );
  if( led_val == 0x80 )
    led_val = 1;
  else
    led_val = led_val << 1;

  /* Wait for 0.5 seconds */
  usleep( 500000 );
}
```