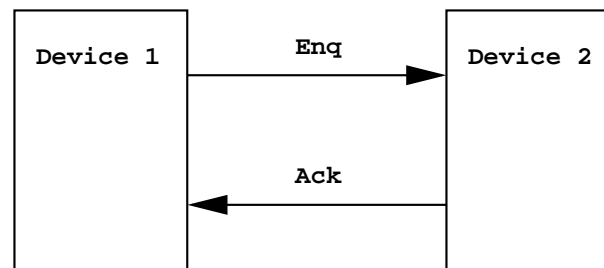


The CPU Bus : Structure

0

The following can be applied to both the internal CPU buses and the external system buses. This distinction becomes blurred when we discuss Systems on a single Chip (SoC).



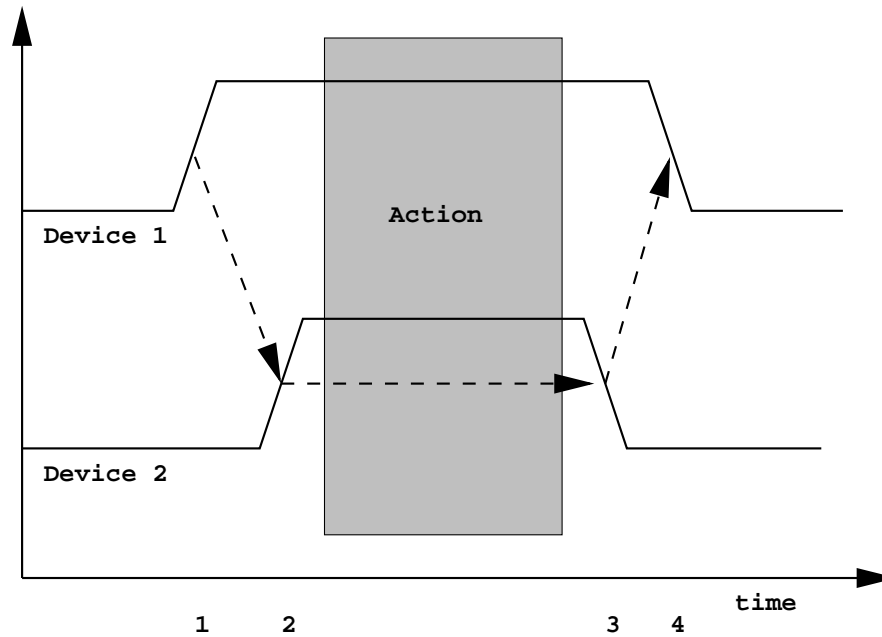
Buses require a protocol in order to function correctly.

Most bus protocols use a four-cycle protocol

- Device 1 raises enq
- Device 2 raises ack
- Device 2 lowers ack
- Device 1 lowers enq

The CPU Bus : Behaviour

1



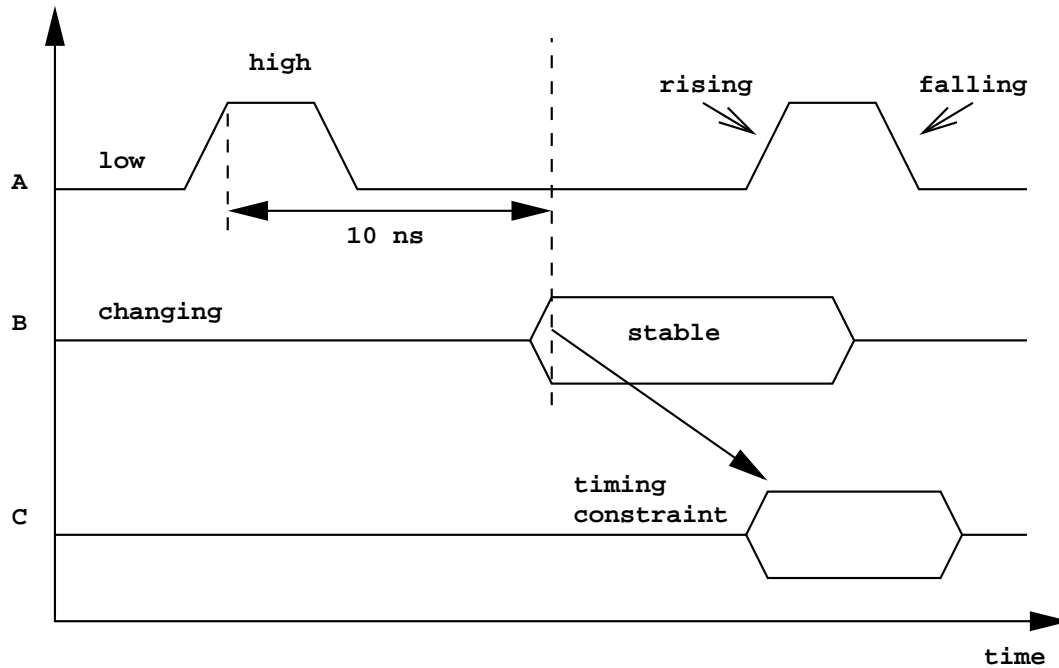
from 'Computers as Components'
W. Wolf

Most bus operations are reading and writing. A typical bus will require the following to support this :

- Clock
 - synchronizes the bus components.
- R/W'
 - controls whether a read or write should take place.
- Address
 - an m-bit set of signals used to identify the location for an access.
- Data
 - an n-bit set of signals used to carry the data to or from the CPU.

Under most circumstances the bus will be controlled by the CPU. The main exceptions are

- DMA transfers, where the CPU relinquishes control of the bus.
- Bus bridges.

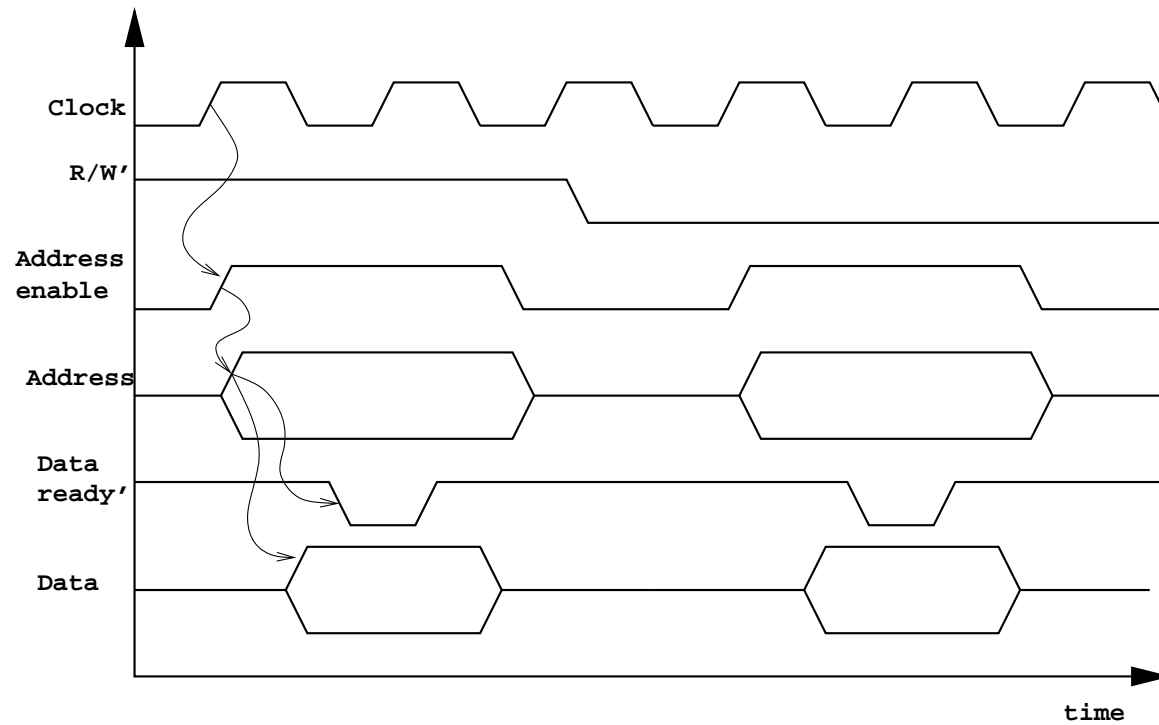


from 'Computers as Components'
W. Wolf

Timing diagram notation

The CPU Bus : Read - Write cycles

4



from 'Computers as Components'
W. Wolf

A typical timing diagram for read and write cycles

Buses will be described as either

- an interface (port description).

or

- a set of signals.

The control signals or ports will normally be unidirectional.

The bus(es) will be bi-directional and will need to be described with a resolution function to resolve the bus access.

When a signal can be driven by more than one source, there must be a way of resolving what happens if multiple drivers are active at the same time.

	U	X	0	1	Z	W	L	H	-
U	U	U	0	U	U	U	0	U	U
X	U	X	0	X	X	X	0	X	X
0	0	0	0	0	0	0	0	0	0
1	U	X	0	1	X	X	0	1	X
Z	U	X	0	X	X	X	0	X	X
W	U	X	0	X	X	X	0	X	X
L	0	0	0	0	0	0	0	0	0
H	U	X	0	1	X	X	0	1	X
-	U	X	0	X	X	X	0	X	X

U	Uninitialised
X	Forcing unknown
0	Forcing 0
1	Forcing 1
Z	High impedance
W	Weak unknown
L	Weak 0
H	Weak 1
-	Don't care

std_u_logic

The resolution table for the AND function when the inputs are defined as std_u_logic. Note that the high impedance value, 'Z', is not treated as a special case.

The CPU Bus : VHDL descriptions

0b

	U	X	0	1	Z	W	L	H	-
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
-	U	X	X	X	X	X	X	X	X

Possible values for
3.3-v LVTTTL

'X' 1.5v
'0' < 0.4v
'1' > 2.4v
'W' > 0.8v & < 1.5v
< 2.0v & > 1.5v
'L' < 0.8v & > 0.4v
'H' > 2.0v & < 2.4v

std_logic

The resolution function for std_logic. This shows what the result will be if 2 values are driving the same signal. For example driving a signal with a strong '1' and a strong '0' results in a strong unknown 'X'.

The Alliance toolset provides the resolution functions mux and wor.

`mux: mux_bit mux_vector(...) bus`

- A resolved subtype of bit.
- May only have a single active driver.
- must be declared with the keyword bus.

`wor: wor_bit wor_vector(...) bus`

- A resolved subtype of bit.
- May have multiple active drivers.
- All drivers must have the same value.
- must be declared with the keyword bus.

```
port (  
  clk      : in std_logic;  
  reset    : in std_logic;  
  enable   : in std_logic;  
  preload  : in std_logic_vector(n-1 downto 0);  
  bus_enable : in std_logic;  
  reg_out  : out unsigned(n-1 downto 0));
```

Port description (vhd) for a generic register connected to a data-bus. At this level we rely on the resolution function provided by std_logic.

This is then converted to a data-flow (vbe) model using vasy ready for simulation and mapping to a technology.

```
port(  
  clk   : in bit;  
  reset : in bit;  
  enable      : in bit;  
  preload    : in bit_vector(7 downto 0);  
  bus_enable  : in bit;  
  reg_out    : out mux_vector(7 downto 0) bus;  
  vdd   : in bit;  
  vss   : in bit  
);
```

Note that the output from the register has been identified as connected to a bidirectional bus and declared as subtype `mux_vector`.

Note that the input has not been modified (other than to the subtype `bit_vector`) as it does not drive the bus.

architecture vbe of Id_reg_8 is

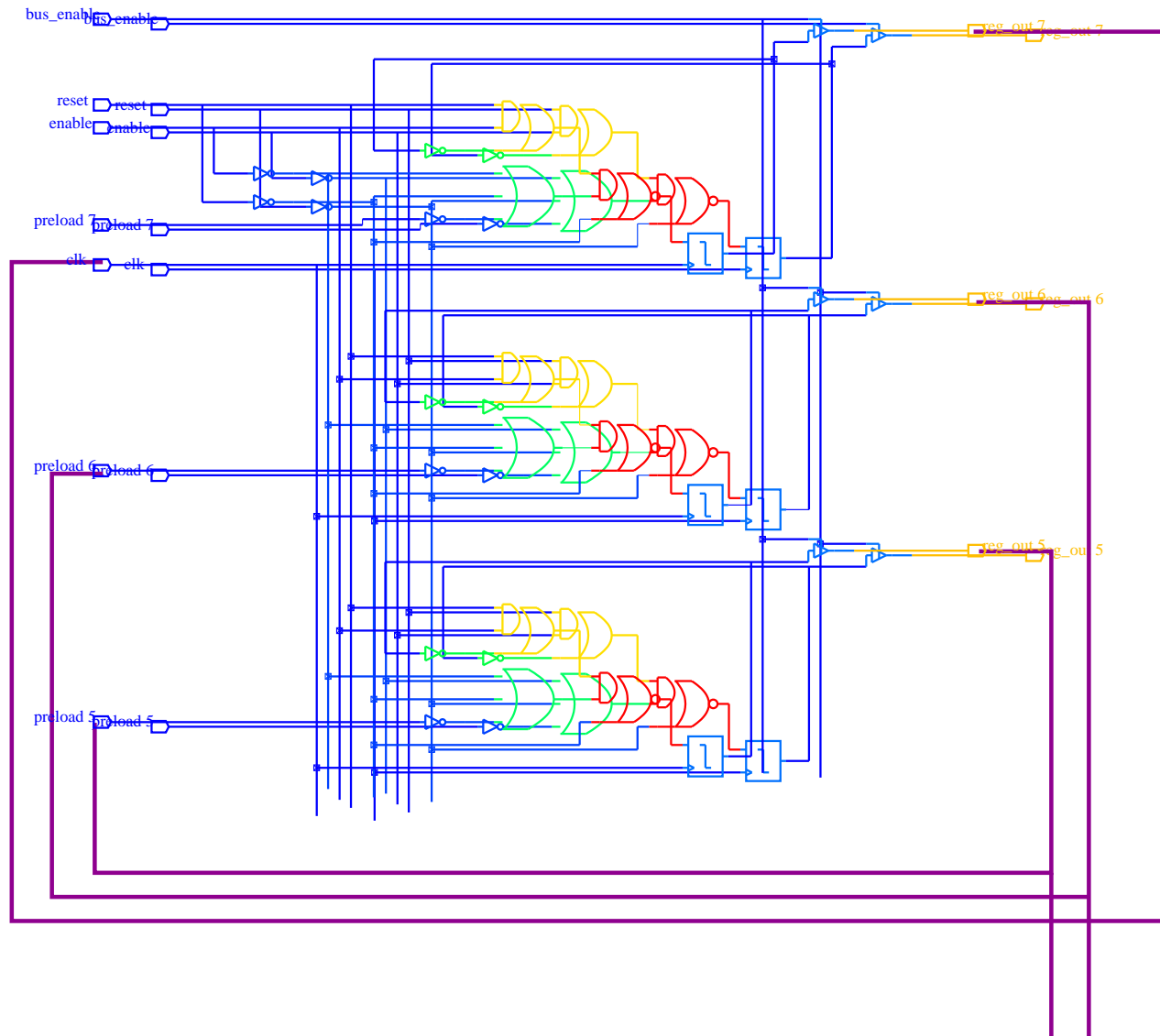
```
    signal reg_val : reg_vector(7 downto 0) register;
begin
    label0 : block (bus_enable = '1')
    begin
        reg_out <= guarded reg_val;
    end block label0;
    label1 : block ((clk = '1') and not(clk'stable) )
    begin
        reg_val <= guarded "00000000" when not(reset) else
            preload          when (reset and enable)
            else reg_val;
    end block label1;
end vbe;
```

The guarded signal provides a resolution function by only permitting assignment of the signal or output under controlled conditions.

- The block statement provides the guard signal.
- The guarded keyword defines the controlled signal. The signal being assigned to will be disconnected from its driver(s) when the guard is false.
- Note that the two block statements are concurrent statements.
- There is an implied tri-state output for the assignment to `reg_out`, as will be seen in the netlist.

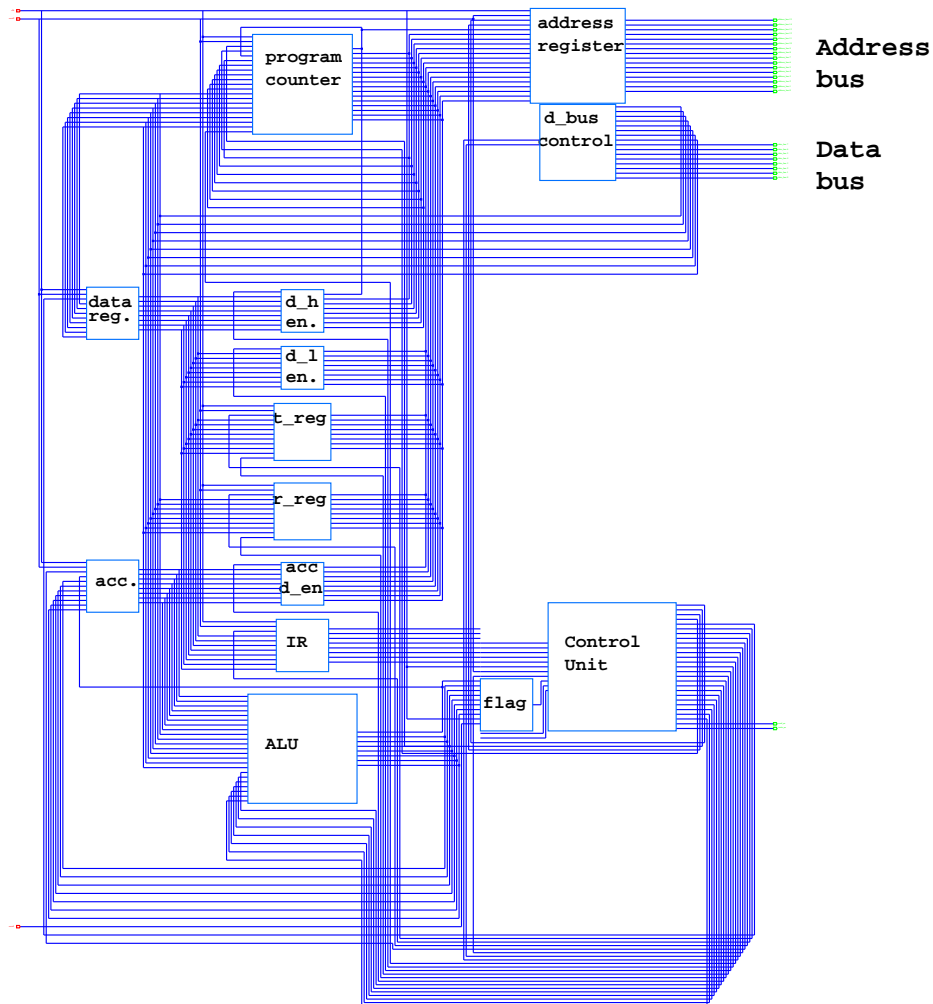
Register to Bus : hardware (vst) view

6



CPU Buses : hardware (vst) view

7



Structure of the r-s cpu showing internal busses and the external connection to the address and data buses

