



L8: Lab 2 Tristate, Glitches, Identifiers, and Catch Up





Lab 2 Assignment



Traffic Light Controller

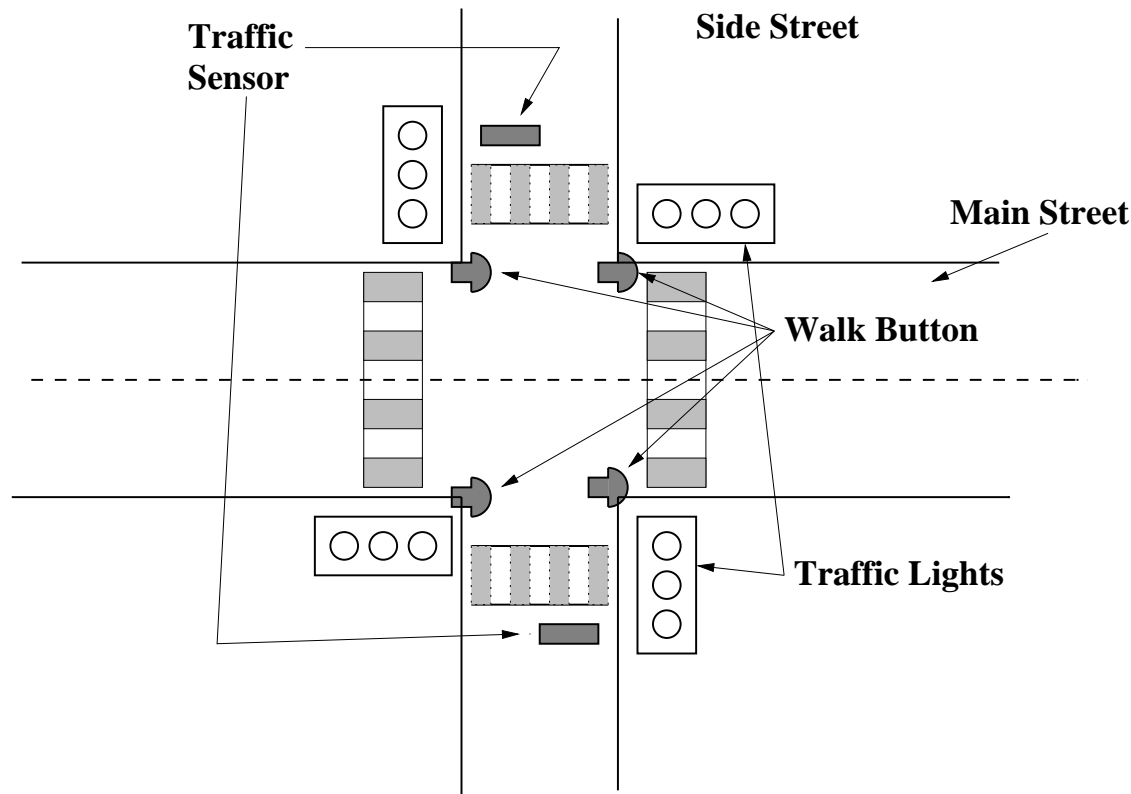
It has main and side streets
with a walk light button.

The main street part of the
cycle is longer than that of
the side street.
(TBASE + TEXT)

But the side street has a
sensor which keeps it green
a bit longer. (another TBASE)
The sensor **MUST** be
synchronized.

The walk button must be
latched and serviced at the
right time. It is to be
unlatched after it has been
serviced.

Walk is R Y
Blink is main Y and side R.
Blink interval (on or off) is
TBLINK.





Top Level Block Diagram



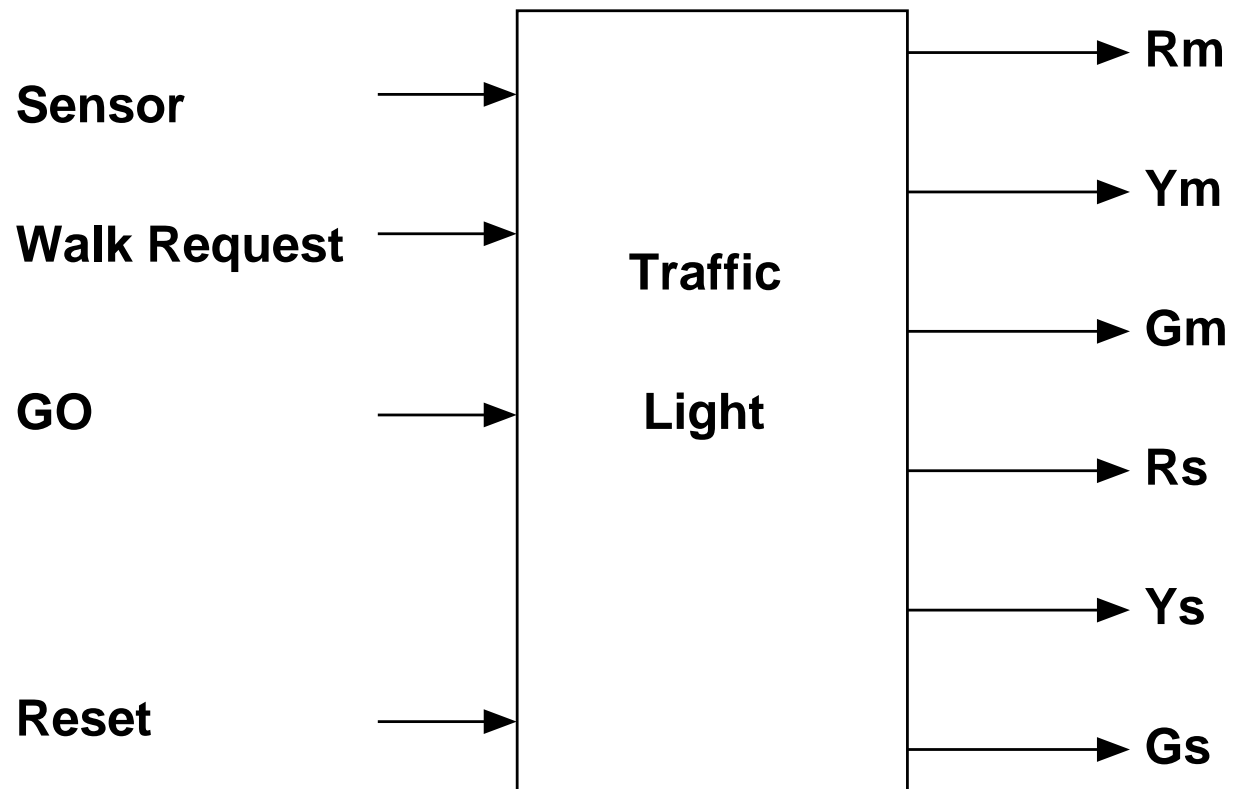
Design Procedure

Start with a simple block diagram.

Break the design down into more simple blocks.

Implement the simpler blocks and put it all together.

Note that this GO signal is similar to that discussed earlier, namely, a single pulse in response to a pushbutton (which could be of any length).

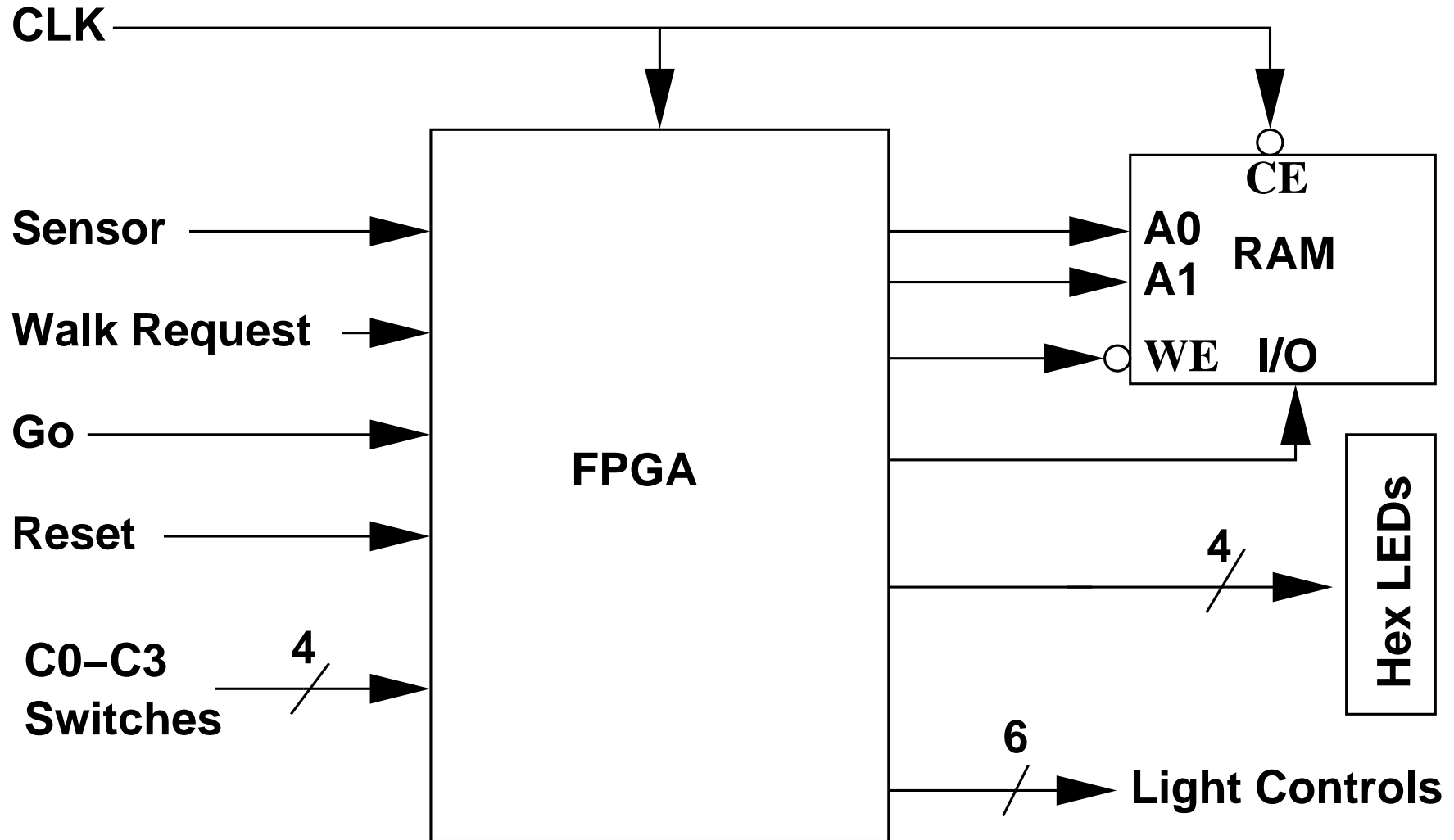




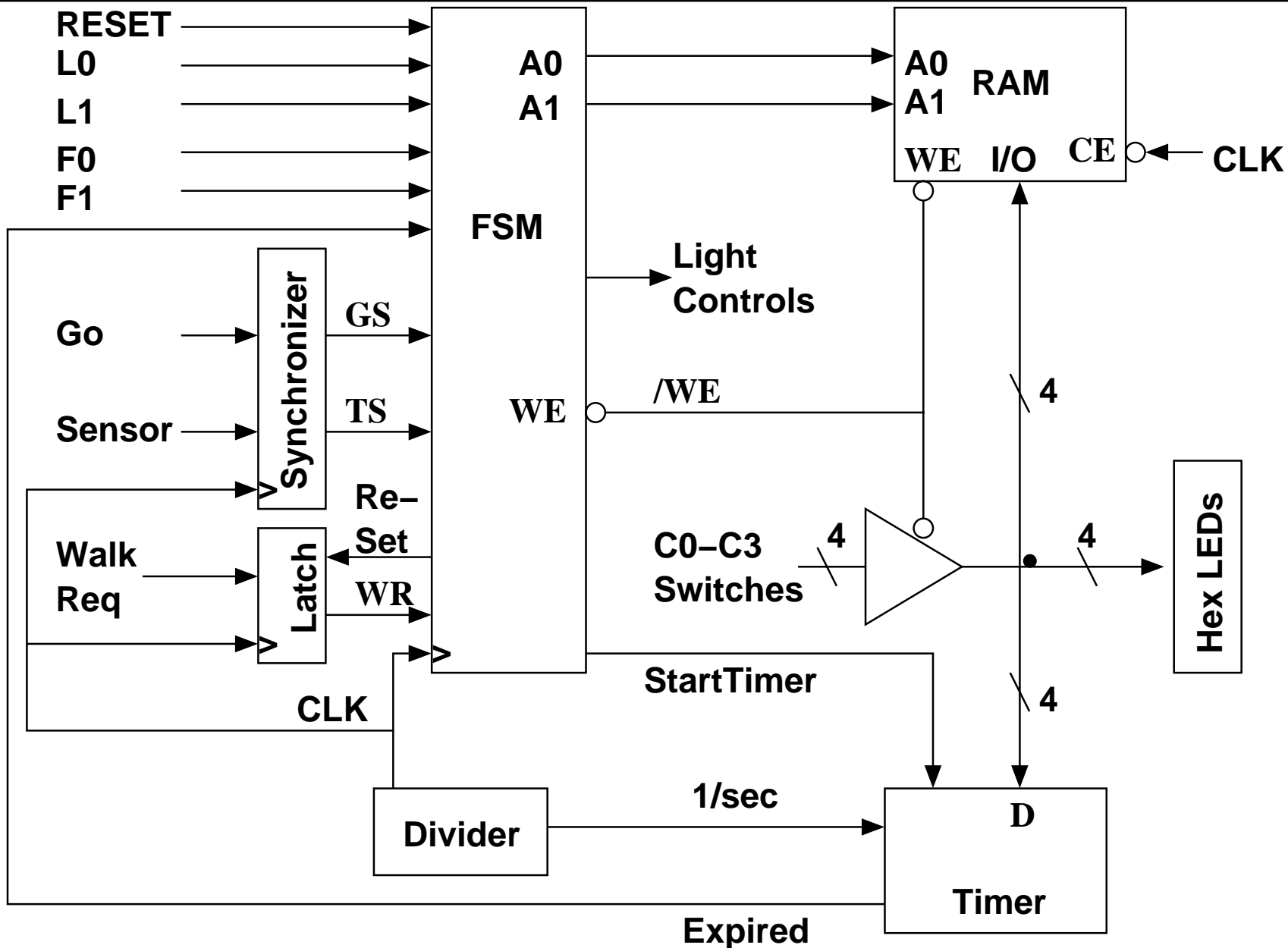
Implementation Details



We want you to use a RAM chip. Do not include RAM in your FPGA. Use the HEX LEDs on the kit to display memory contents.



Detailed Functional Diagram





FSM Inputs and Outputs



■ Inputs

- RESET** (from a switch)
- GS** Synchronized GO signal
- F1, F0** Function Selection (from switches)
- L0, L1** RAM Location Selection (from switches)
- TS** Synchronized sensor signal – This is wrong in the lab 2 handout. on page 4. Correct the handout. The web copy is corrected.
- WR** Walk Request (Re settable latch set by pushbutton)
- Expired** True when the timer has timed out

■ Outputs

- A0, A1** SRAM Address
- N_WE** SRAM Write Enable, also Gates Switches onto I/O
- StartTimer** Resets one second increment timer
- Light Controls**
 - Rm, Ym, Gm, Rs, Ys, Gs
- RST** Resets the walk request latch.



Functions and RAM Locations



■ Function Specifications

| | |
|--------------|--|
| F1 F0 | Are provided by function control switches. |
| 0 0 | Examine memory location provided by the switches. |
| 0 1 | Store value in memory location specified by switches. |
| 1 0 | Run traffic lights. |
| 1 1 | Blink. |

■ Meaning of locations of the SRAM

| | |
|--------------|--|
| L1 L0 | Location specification is provided by switches. |
| 0 0 | TYEL Time for yellow light |
| 0 1 | TBASE Base Interval |
| 1 0 | TEXT Extension Interval |
| 1 1 | TBLINK Blink Interval |



Lab 2 Schedule



| | | |
|--------------|-----------------|---|
| Wed. | Sept. 24 | Lab Assignment (done today) |
| Mon. | Sept. 29 | Design Conference (oral, with TA or LA) |
| Mon. | Oct. 6 | Lab 2 check off by 5:00 P.M. (in lab). |
| Tues. | Oct. 7 | Lab 2 report due by 5:00 P.M. (in lab). |
| Tues. | Oct. 28 | Commented Lab 2 Report returned (in recitation). |
| Wed. | Nov. 5 | Revised Lab 2 report due at lecture. |



Tristate Loadable Counter



```
-- Use tri state logic to multiplex IO pins.
library ieee;
use ieee.std_logic_1164.all;
-- needed for integer + signal
use ieee.std_logic_unsigned.all;
entity Idcntc is
-- can pass in parameter width if
-- instantiated as a component.
-- Otherwise 3 is the value of width.
generic (width : integer := 3);
port(clk, ld, oe, cnt_enb : in std_logic;
      data : inout
        std_logic_vector(width- 1 downto 0));
end Idcntc;
-- purpose: count with an output enable
```

```
architecture archldcnt of Idcntc is
signal counter :
  std_logic_vector(width- 1 downto 0);
begin
  data <= counter when oe = '1'
    else (others => 'Z');
    -- N.B. Z must be UPPER CASE!
cnt: process(clk)
begin
  if rising_edge(clk) then
    if ld = '1' and oe = '0' then
      counter <= data;
    elsif cnt_enb = '1' then
      counter <= counter + 1;
    end if;
  end if; -- rising_edge(clk)
end process cnt;
end architecture archldcnt;
```



Tristate Simulation



```

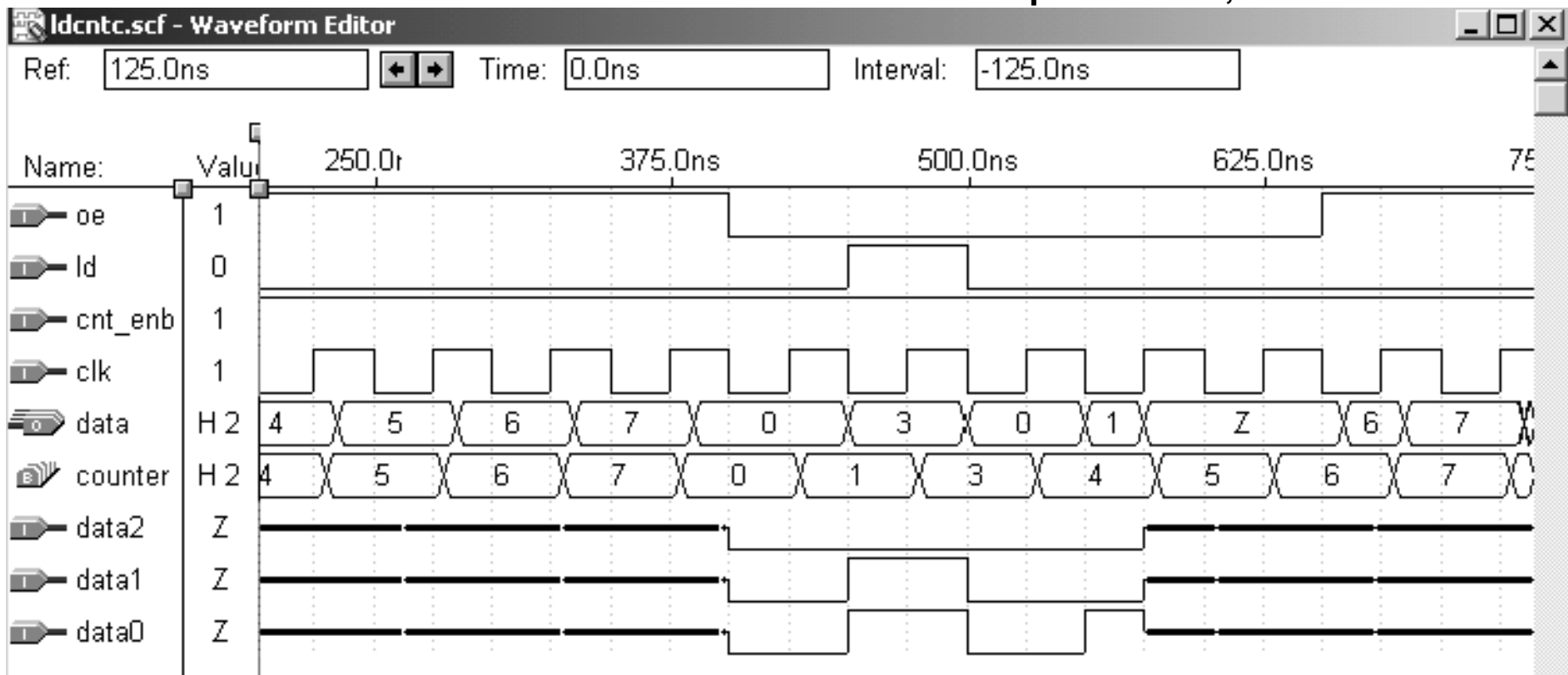
data <= counter when oe = '1'
  else (others => 'Z');
  -- N.B. Z must be UPPER CASE!
cnt: process(clk)

```

```

begin
  if rising_edge(clk) then
    if ld = '1' and oe = '0' then
      counter <= data;
    elsif cnt_enb = '1' then
      counter <= counter + 1;
    end if;
  end if; -- rising_edge(clk)
end process cnt;

```



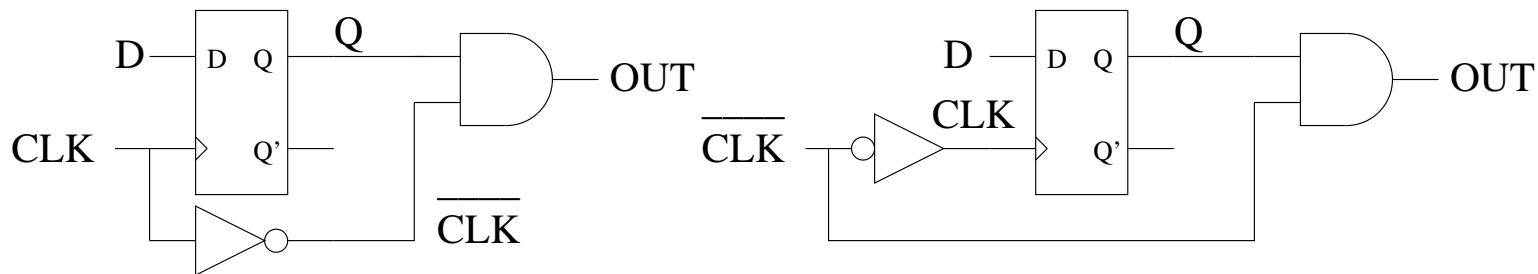


Avoiding Glitches- Gating



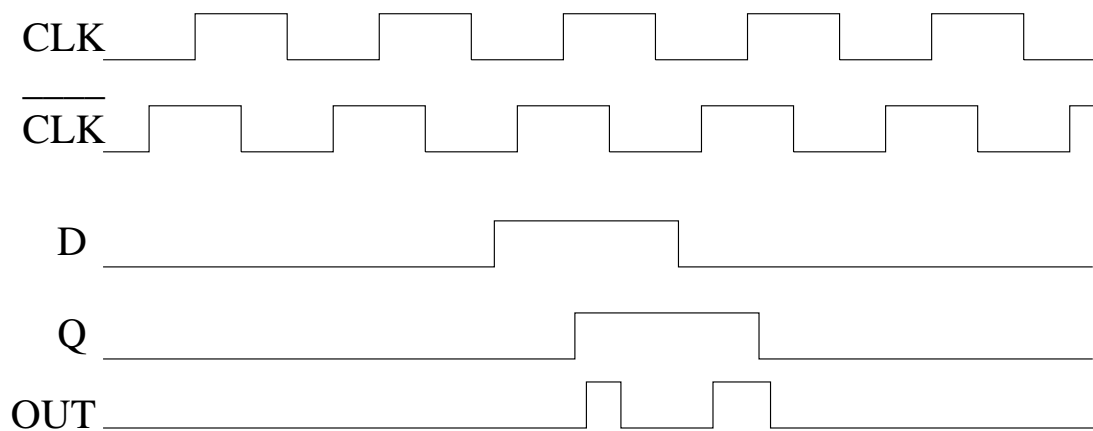
■ Gate the output with the clock (carefully).

- This is another way of saying “Don’t look at a combinational output until the output has settled down into its final state”.
- Make sure that the gated pulse does NOT have a glitch.



Suppose the inverter delay is large compared to the CLK to Q delay.

Then there can be a glitch on the circuit to the left but not on the circuit to the right.



**Remember –
Do NOT use glitchy
signals for CLK,
PR, CLR, S, or R.
Clock data into a
register AFTER
signals are stable.**



Avoiding Glitches- Register

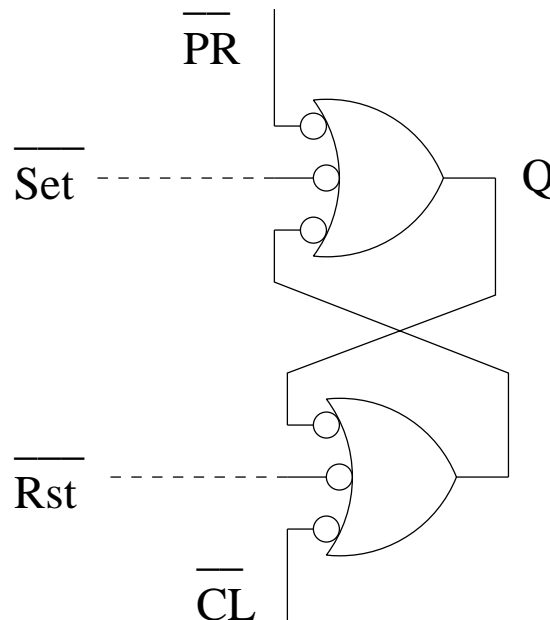


■ Register the output.

- If a flip flop does NOT change state upon the occurrence of a clock pulse then it has no glitches, i.e.,
 - 0 to 0 is guaranteed not to glitch to 1 in between and
 - 1 to 1 is guaranteed not to glitch to 0 in between.
- We assume that setup and hold times are honored.
- Some high performance flip flops do not adhere to this. They are only concerned that the flip flop end up in the “right” state and not whether they have glitches on the output.

This is the output latch of an edge triggered flip-flop.

The set or reset pulses (negative true) are full pulses and only one occurs if the setup and hold times are adhered to.





VHDL Identifiers & Reserved Words



- **Case InSenSitive (but best not to rely on this)**
 - **First character must be a letter.**
 - **Letters, Digits, and Underscores (only)**
 - **Two underscores in succession are not allowed.**
 - **The last character cannot be an underscore.**
 - **Using reserved words as identifiers is NOT allowed.**
 - **Reserved words are AQUA in emacs.**
 - **Using reserved words usually provokes an understandable error comment.**
 - **Legal examples are CLK, Three_state_enable, h23. Reg_12.**
 - **Illegal examples are _clk, 3_state_enable, large#num, clk__.**
 - **Some reserved words are**
 - **Abs, access, impure, postponed**
 - **In other words, there are too many to remember!**
 - **To see what is a reserved word, notice the color in your editor.**
 - **This is another good reason for “incremental” compilation.**
 - **Start with code that compiles and add a block at a time.**



Miscellaneous



- **s'event** is read as “s tick event” where s is a signal name.
- **Rising_edge(s)** is the same as (s'event and s = '1')
 - as synthesis only worries about 1 and 0.
- **s'event-** true if an event occurred in the current delta cycle
- **Don't cares** are represented by a – (hyphen).
 - ' ' for a character “ - - ” for a string (vector)
 - (others => '1') for something independent of length
- **& (ampersand)** to concatenate strings or signals
 - “01” & “111” is the same as “01111”.
 - '0' & “1111” is the same as “01111”.
- **A + B** is valid only if A and B are of the same length.
 - The result is of the same length as A (or B).
 - If you want the result to be one bit longer then use
 - $C \leq ('0' \& A) + ('0' \& B)$ - of course, C must be a BIT longer.



Array Attributes



■ One dimensional array

```
signal s : std_logic_vector ( 7 downto 3);
```

```
s'left      = 7  
s'right     = 3  
s'length    = 5  
s'high      = 7  
s'low       = 3
```

■ Two dimensional array

```
type rom is array (0 to 6, 3 downto 0) of std_logic;
```

```
r'left(1)    = 0           r'right(1)   = 6  
r'left(2)    = 3           r'right(2)   = 0  
r'high(1)    = 6           r'low(1)    = 0  
r'high(2)    = 3           r'low(2)    = 0  
r'length(1)  = 7  
r'length(2)  = 4
```

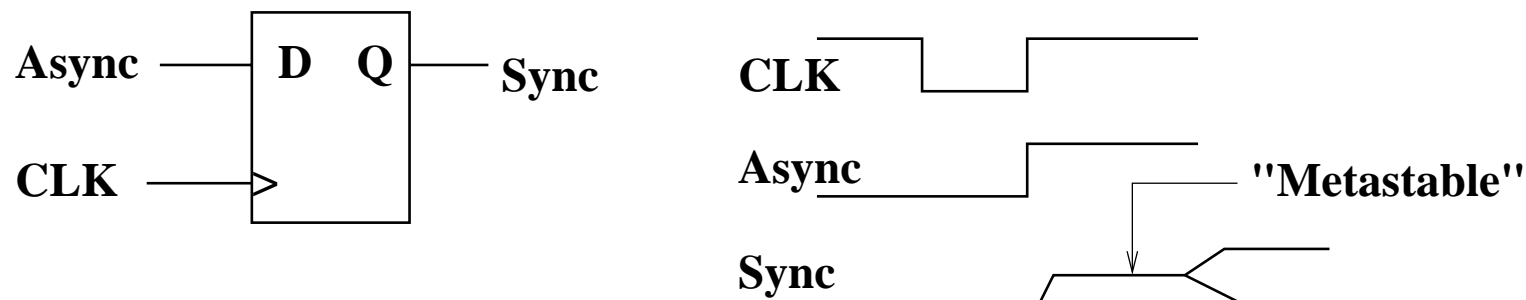
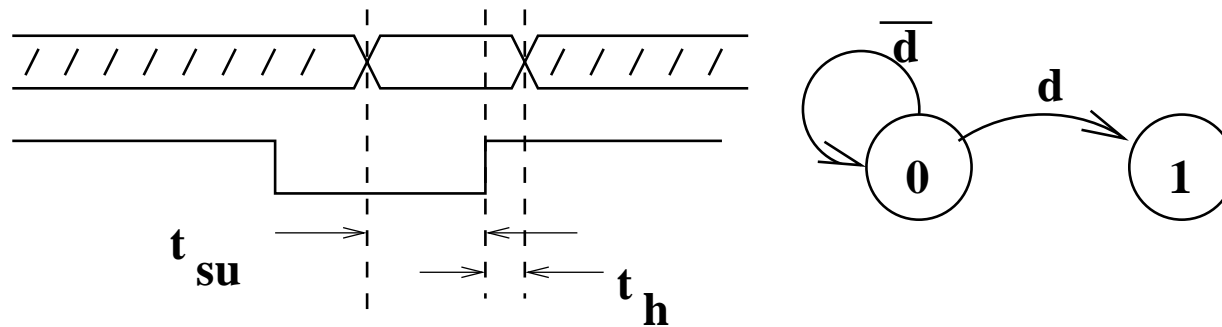


Important Design Rule



DESIGN RULE:

1. Synchronize ALL external signals.
2. Any asynchronous input must affect ONLY ONE flip-flop.



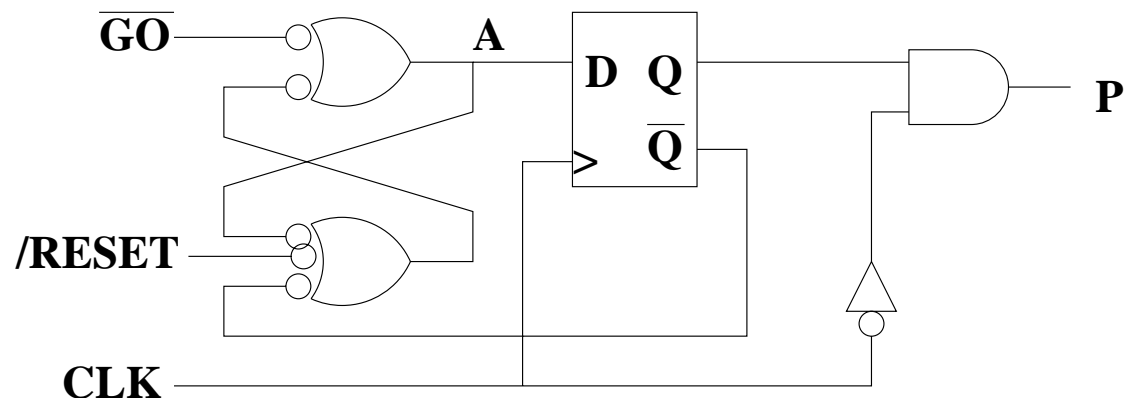
Any combinational logic with "Sync" as an input will be "glitchy" until after the metastable state has expired.
In particular, do NOT use "Sync" as a CLK input.



VHDL for a Short Pulse Catcher



```
library ieee;
use ieee.std_logic_1164.all;
entity spc is
    port(n_go, clk, n_reset : in std_logic;
         p : out std_logic);
end spc;
-- purpose: catch a short pulse
architecture behavioral of spc is
    signal a, n_a, q, n_q, n_clk: std_logic;
begin -- behavioral
    a <= (not n_go) or (not n_a);
    n_a <= (not a) or (not n_q) or (not n_reset);
    n_q <= (not q);
    p <= q and (not clk);
ff: process(clk)
begin
    if rising_edge(clk) then
        q <= a;
    end if;
end process ff;
end behavioral;
```



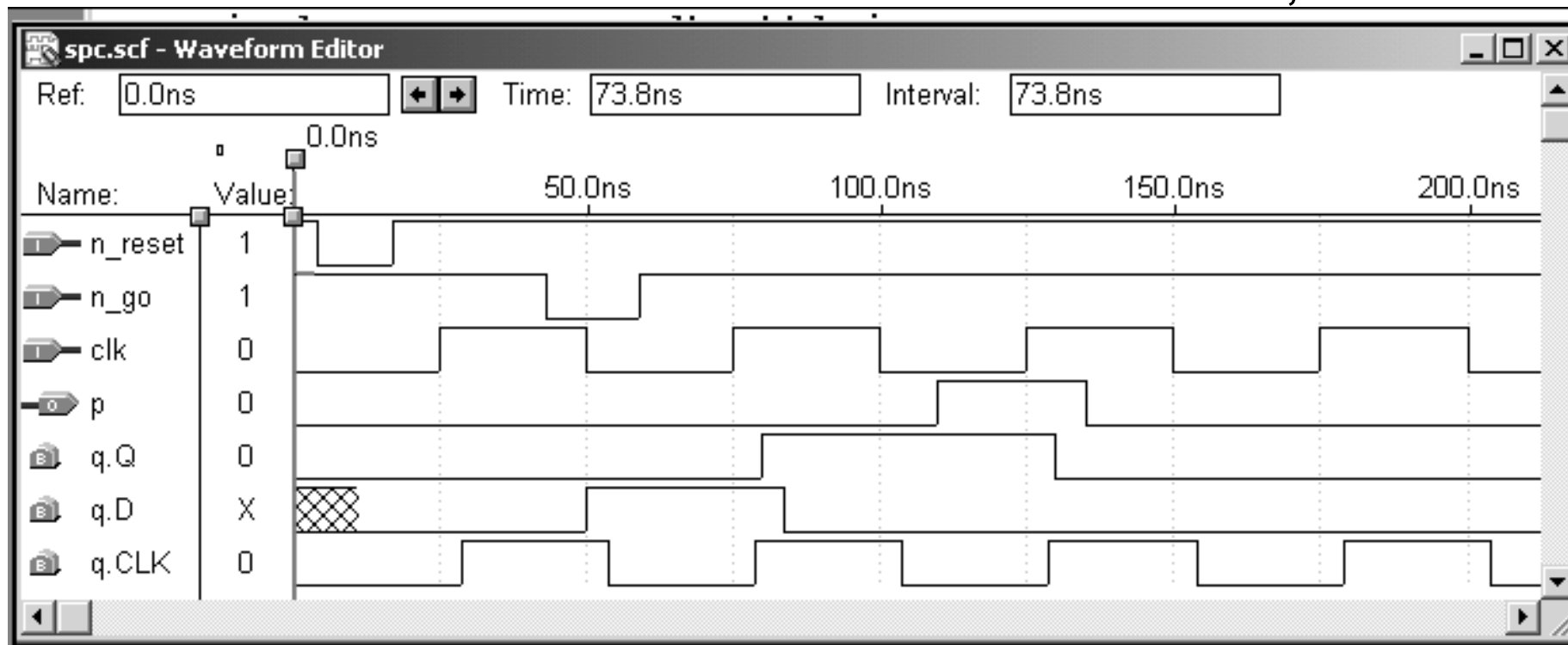


Simulation of a Short Pulse Catcher



```
architecture behavioral of spc is
  signal a, n_a, q, n_q, n_clk: std_logic;
begin -- behavioral
  a <= (not n_go) or (not n_a);
  n_a <= (not a) or (not n_q) or (not n_reset);
  n_q <= (not q);
  p <= q and (not clk);
```

```
ff: process(clk)
begin
  if rising_edge(clk) then
    q <= a;
  end if;
end process ff;
end behavioral;
```



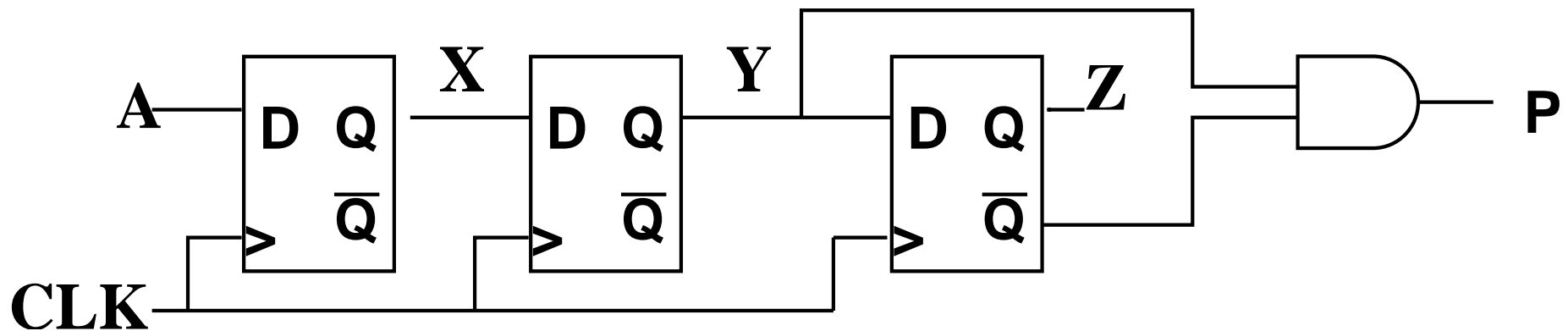


Level to Pulse



```
library ieee;
use ieee.std_logic_1164.all;
entity pform is
  port(A, CLK: in std_logic;
        X, Y, Z: buffer std_logic;
        P: out std_logic);
end pform;
-- purpose: turn a level into a
-- finite width pulse
```

```
architecture behavioral of pform is
begin -- behavioral
ff: process(CLK)
begin
  if rising_edge(CLK) then
    X <= A;
    Y <= X;
    Z <= Y;
  end if;
end process ff;
P <= (Y AND (not Z));
end behavioral;
```

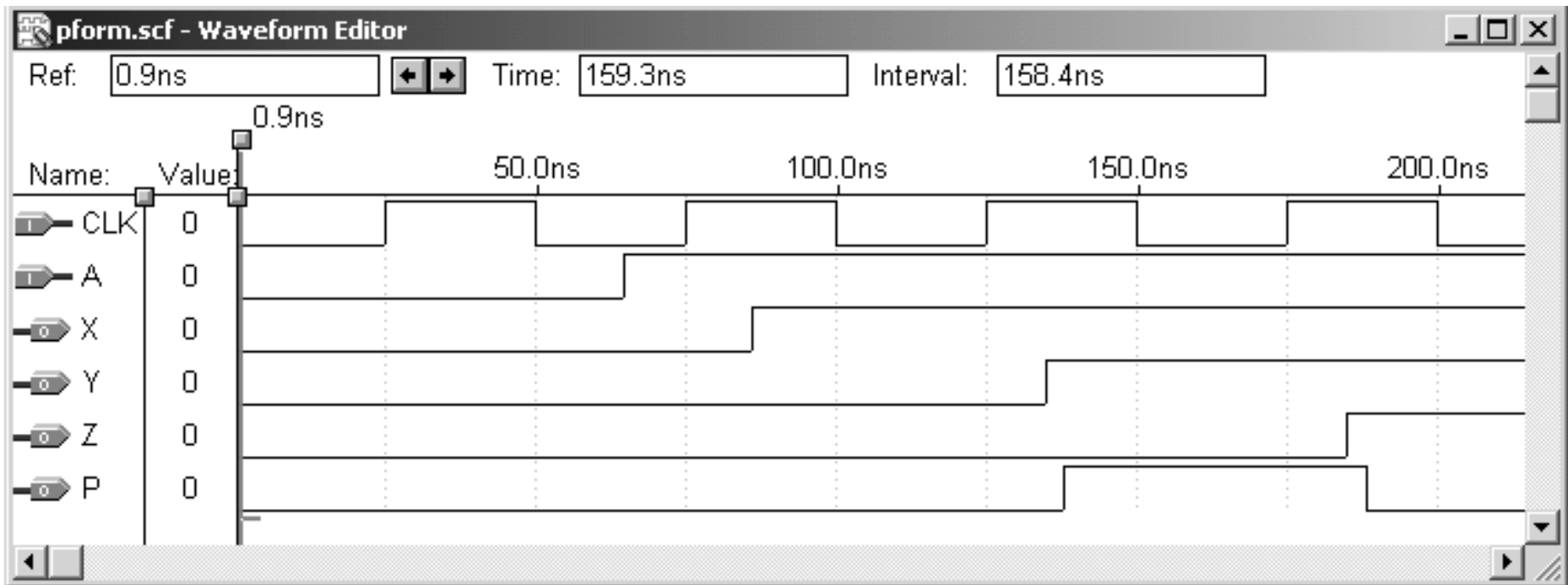


Simulation of Level to Pulse



```
architecture behavioral of pform is
begin -- behavioral
ff: process(CLK)
begin
```

```
    if rising_edge(CLK) then
        X <= A;
        Y <= X;
        Z <= Y;
    end if;
end process ff;
P <= (Y AND (not Z));
end behavioral;
```

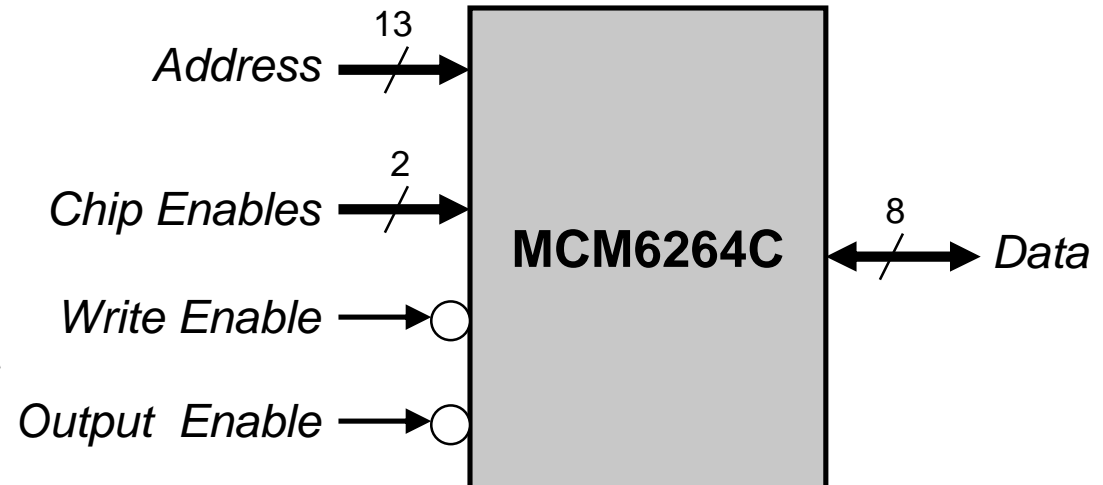




MCM6264C 8k x 8 Static RAM



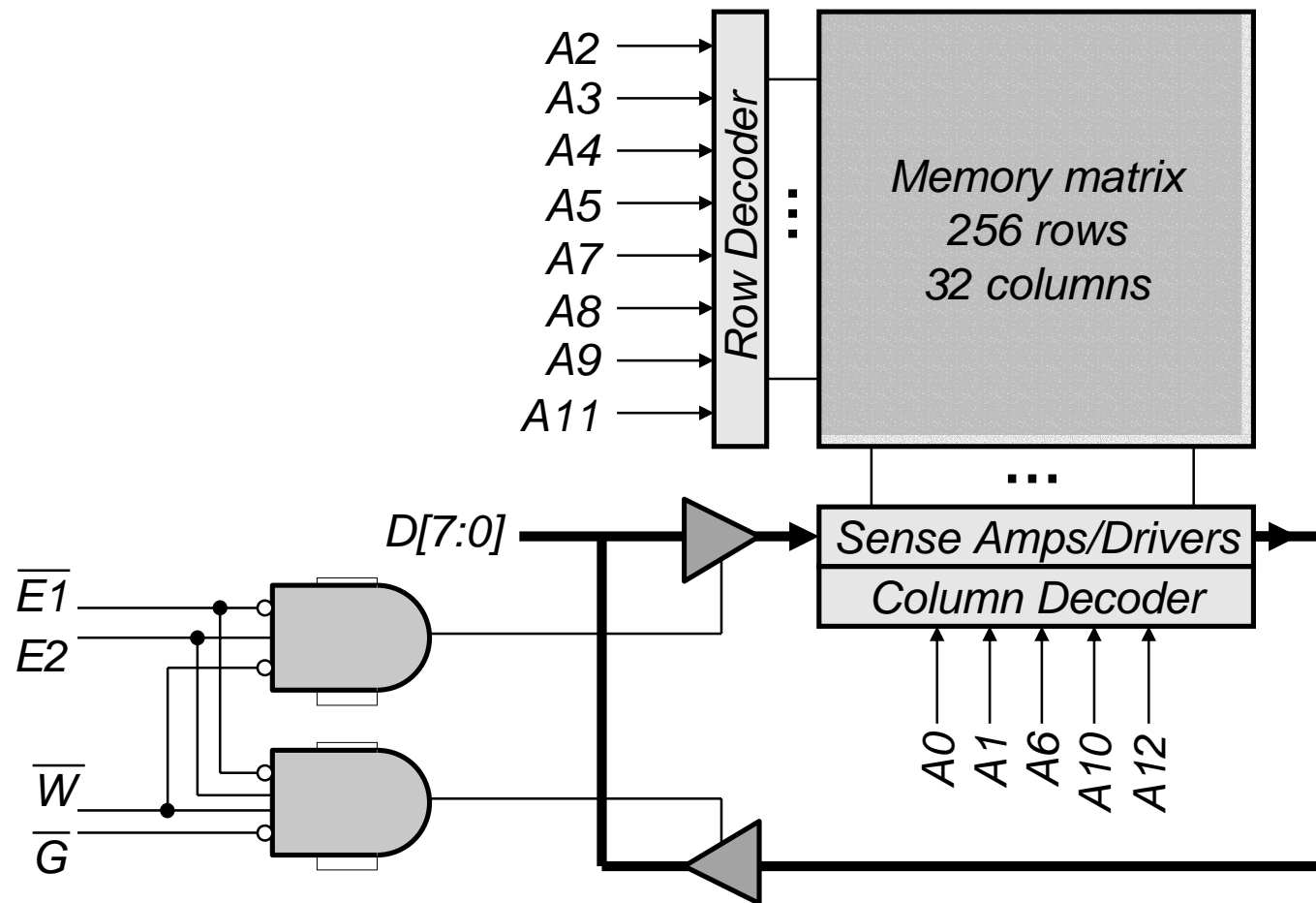
- Same (bidirectional) data bus used for reading and writing
- Chip Enables ($\overline{E1}$ and $E2$)
 - $\overline{E1}$ must be low and $E2$ must be high to enable the chip.
- Write Enable (\overline{W})
 - When low (and chip is enabled), the values on the data bus are written to the location selected by the address bus.
- Output Enable (\overline{G})
 - When low (and chip is enabled), the data bus is driven with the value of the selected memory location.



| | | | |
|-----------------|----|----|-----------------|
| NC | 1 | 28 | V _{CC} |
| A12 | 2 | 27 | \overline{W} |
| A7 | 3 | 26 | E2 |
| A6 | 4 | 25 | A8 |
| A5 | 5 | 24 | A9 |
| A4 | 6 | 23 | A11 |
| A3 | 7 | 22 | \overline{G} |
| A2 | 8 | 21 | A10 |
| A1 | 9 | 20 | $\overline{E1}$ |
| A0 | 10 | 19 | DQ7 |
| DQ0 | 11 | 18 | DQ6 |
| DQ1 | 12 | 17 | DQ5 |
| DQ2 | 13 | 16 | DQ4 |
| V _{SS} | 14 | 15 | DQ3 |

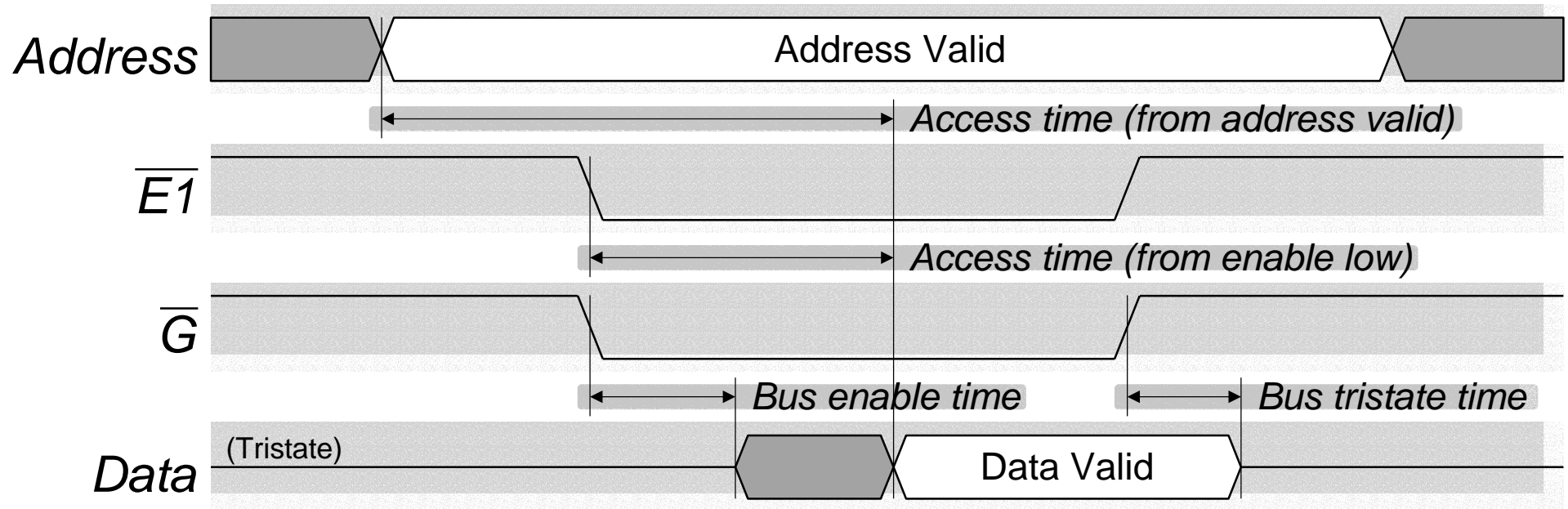


Inside the '6264C





Reading From SRAM

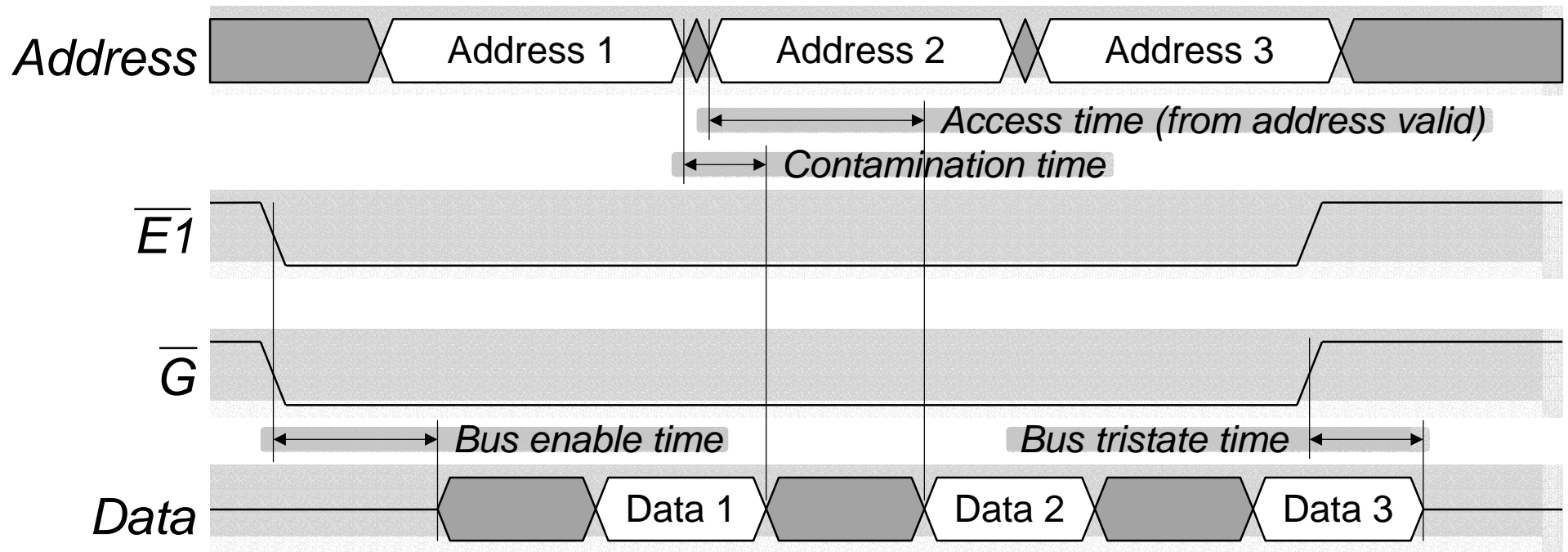


$E2$ assumed high (enabled), $\overline{W} = 1$ (read mode)

- Read cycle begins when all enable signals ($\overline{E1}$, $E2$, \overline{G}) are active.
- Data is valid after read access time.
 - Access time is indicated by full part number: *MCM6264CP* $t_2 \rightarrow 12ns$.
- Data bus is tristated shortly after \overline{G} or $\overline{E1}$ goes high.



Address Controlled Reads

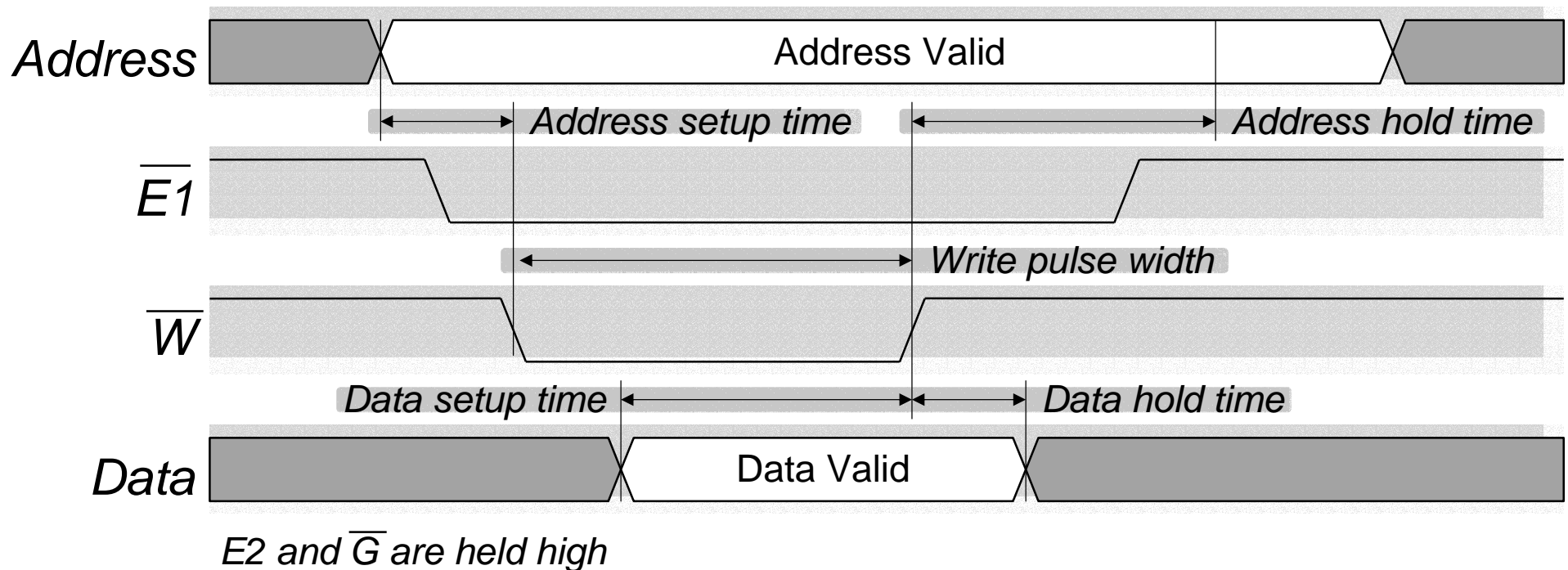


$E2$ assumed high (enabled), $\overline{W} = 1$ (read mode)

- Can perform multiple reads without disabling chip
- Data bus follows address bus, after some delay.



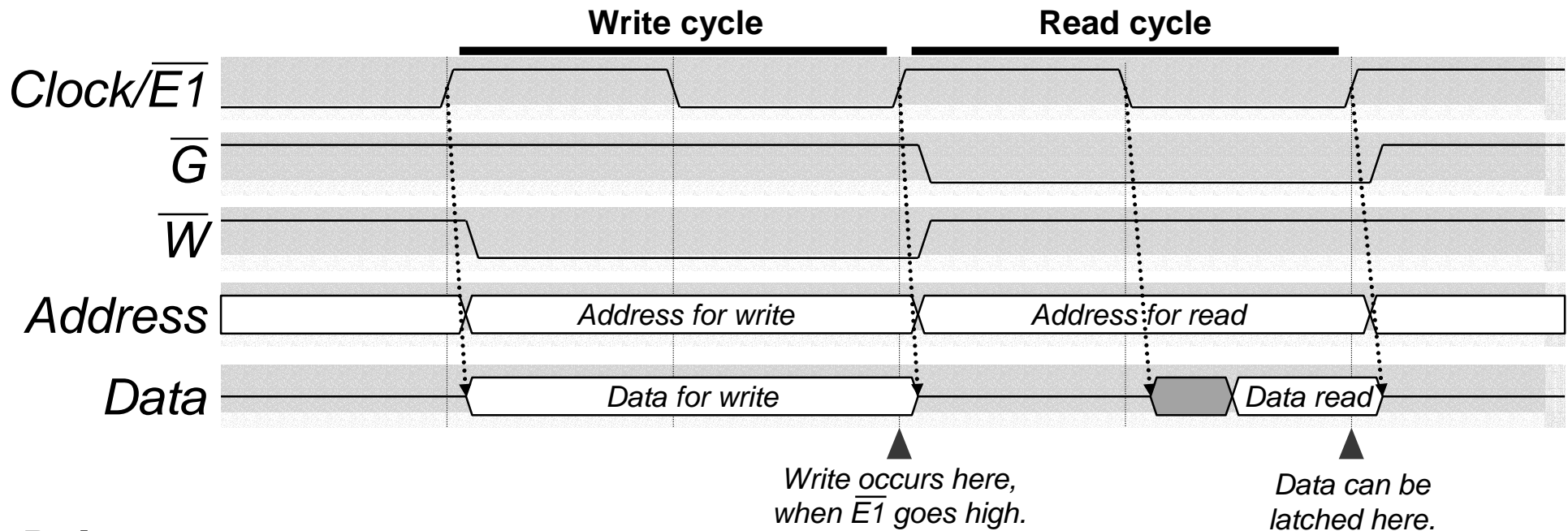
Writing to SRAM



- Data latched when \overline{W} or $\overline{E1}$ goes high (or $E2$ goes low)
 - Data must be stable at this time.
 - Address must be stable before \overline{W} goes low.
- Write waveforms are more important than read waveforms.
 - Glitches can cause writes to random addresses!

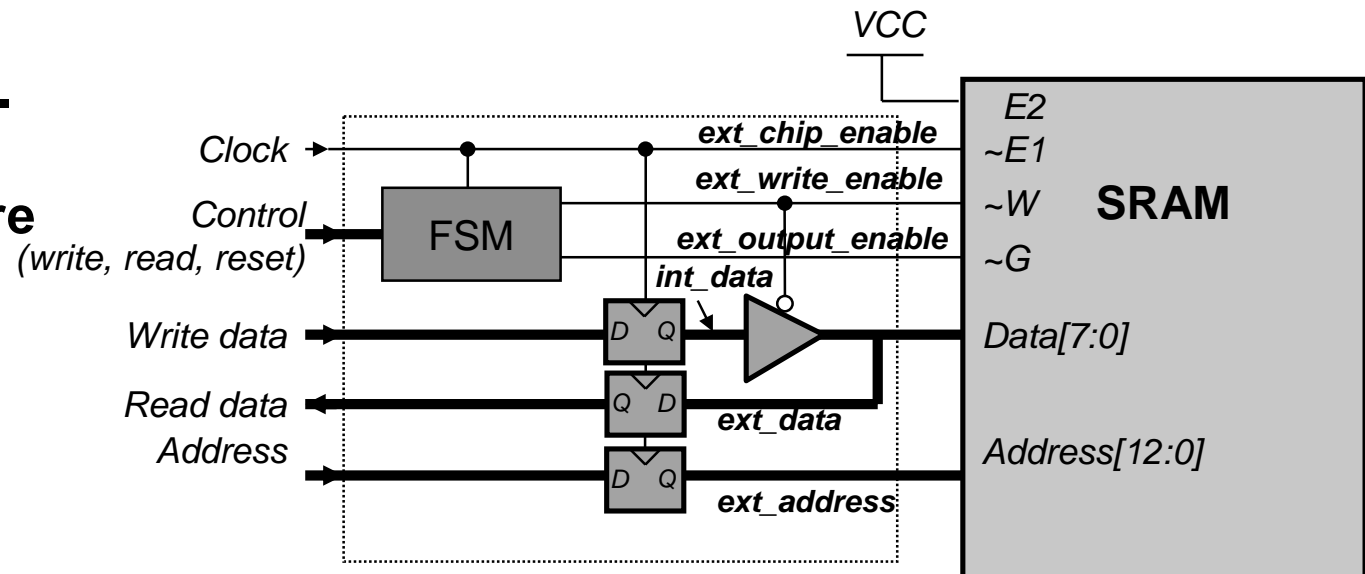


Sample Memory Interface Logic



■ Drive memory enable with clock.

- Ensures data and memory busses are stable for writes
- Minimum clock period is twice memory access time.





Toy Memory Interface in VHDL (I)



```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity mem_int is  
  port (clock : in std_logic;  
        reset : in std_logic;  
        write : in std_logic;  
        read : in std_logic;  
        address : in std_logic_vector(12 downto 0);  
        write_data : in std_logic_vector(7 downto 0);  
        read_data : out std_logic_vector(7 downto 0);  
        ext_chip_enable : out std_logic;  
        ext_write_enable : out std_logic;  
        ext_output_enable : out std_logic;  
        ext_address : out std_logic_vector(12 downto 0);  
        ext_data : inout std_logic_vector(7 downto 0));  
end mem_int;  
  
architecture behavioral of mem_int is  
  
  signal int_data : std_logic_vector(7 downto 0);
```

(cont. on next viewgraph)



Toy Memory Interface in VHDL (II)



```
begin
ext_chip_enable <= clock;
ext_data <= int_data when ext_write_enable = '0' else (others => 'Z');

process (clock, reset)
begin
  if reset = '0' then
    ext_write_enable <= '1';
    ext_output_enable <= '1';
  elseif clock'event and clock = '1' then
    ext_address <= address;
    read_data <= ext_data;
    int_data <= write_data;
    if write = '1' then
      ext_write_enable <= '0';
    elsif read = '1' then
      ext_output_enable <= '0';
    else
      ext_write_enable <= '1';
      ext_output_enable <= '1';
    end if;
  end if;
end process;

end behavioral;
```