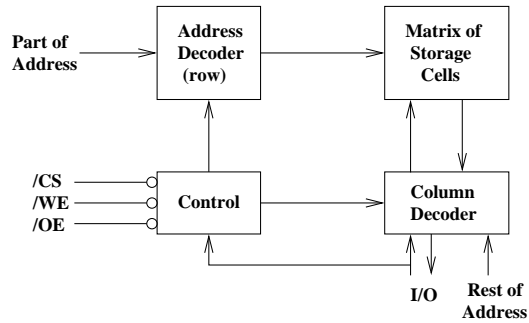


8.1 Memory

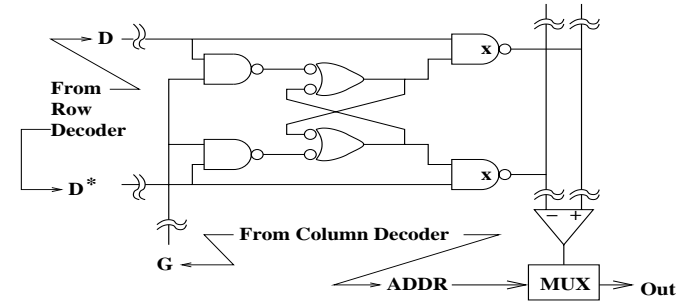
Friday, February 23, 2001

Memories are usually organized as 2-dimensional arrays of cells.



8.2 Conceptual Memory Cell

Note how this is like a 'D-Latch'.



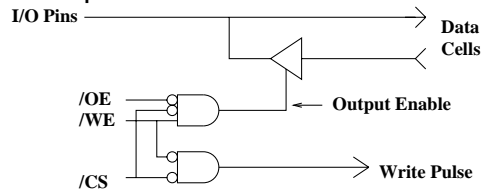
8.3 Control Lines:

Often Active Low

OE is 'Output Enable'.

WE is 'Write Enable'.

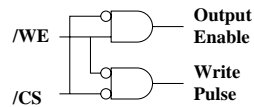
CS is 'Chip Select'.



To read: pull /OE and /CS low (active)

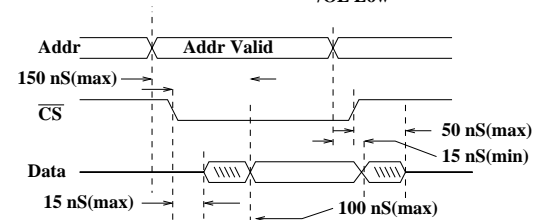
To write: pull all 3 control lines low

Some parts have 2 Control lines:

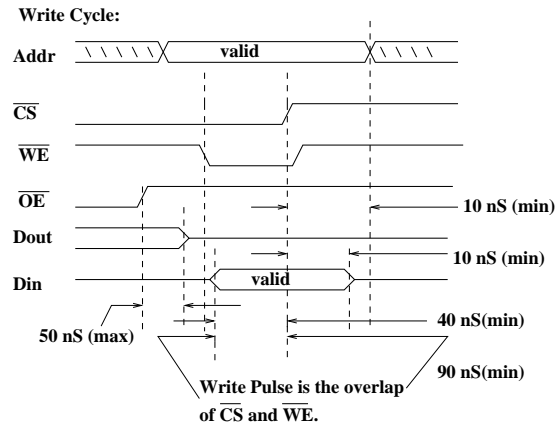


8.4 Read Timing

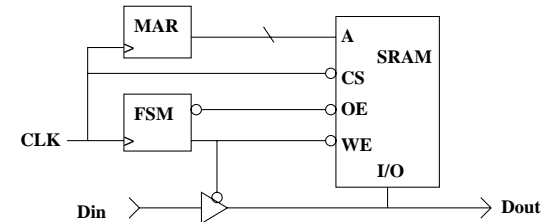
Read Cycle Timing: (6116-3) /WE High /OE Low



8.5 Write Timing



8.6 A Possible Control Structure

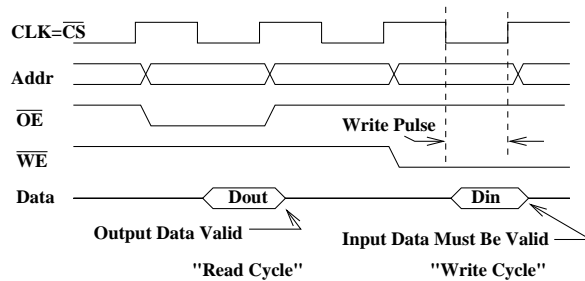


Here is a suggested setup for controlling memory.

FSM is the control.

Note: if the data bus is driven ONLY by D and SRAM, you can just ground \overline{OE} and not bother with it in the FSM!

8.7 Read and Write Cycles

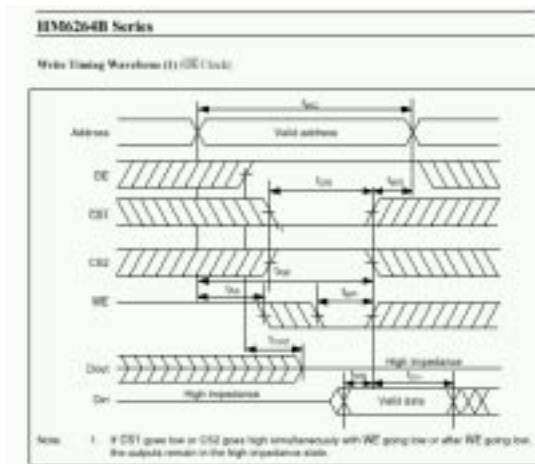


8.8 HM6264

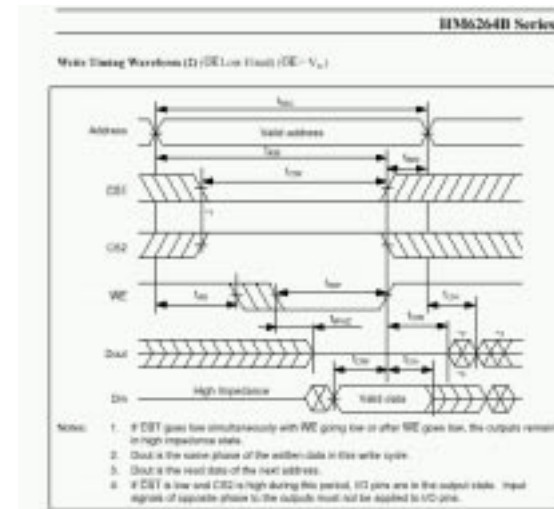
HM6264 has TWO Chip Select Lines:

\overline{WE} / $\overline{CS1}$ / $\overline{CS2}$ / \overline{OE}	Mode	I/O
X H X X	Standby (low pwr)	High Z
X X L X	Standby (low pwr)	High Z
H L H H	Output disabled	High Z
H L H L	Read	Data Out
L L H H	Write (1)	Data In
L L H L	Write (2)	Data In

8.9 6264B Write Cycle 1



8.10 6264B Write Cycle 2



8.11 Finite State Machines

□ A key decision is how the outputs are to be encoded.

- Function of state only
- Function of input and state
- Registered - glitch free
- State flip-flop - glitch free

□ Generally, it is more efficient to have an output flip-flop also be a state flip-flop.

□ However, there may be situations where an output flip-flop is not a state flip-flop, for example, when one doesn't want noise on an output to possibly cause an incorrect state transition.

8.12 rygtype2.vhd

```
library ieee; --This is "best" (for this example).
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity ryg is port (
    t, clk : in std_logic;
    r, y, g : out std_logic);
--attribute pin_avoid of ryg: entity is "23 14";
end ryg;
architecture state_machine of ryg is
    type StateType is (red, yel, grn, walk);
    attribute enum_encoding of StateType: type is
        "100 010 001 110";
    signal p_s, n_s : StateType;
begin
    r <= '1' when (p_s = red) or (p_s = walk) else '0';
    y <= '1' when (p_s = yel) or (p_s = walk) else '0';
    g <= '1' when (p_s = grn) else '0';
    state_clocked:process(clk)
    begin
        if rising_edge(clk) then p_s <= n_s;
        end if;
    end process state_clocked;
--More architecture in the next slide.
```

8.13 More Architecture for rygtype2.vhd

```
fsm:process(p_s, t) -- combinational
begin -- case
  case p_s is
    when red =>
      if t = '1' then n_s <= grn;
      else n_s <= red;
      end if;
    when grn =>
      if t = '1' then n_s <= yel;
      else n_s <= grn;
      end if;
    when yel =>
      if t = '1' then n_s <= walk;
      else n_s <= yel;
      end if;
    when walk =>
      if t = '1' then n_s <= red;
      else n_s <= walk;
      end if;
    when others => n_s <= walk;
  end case;
end process fsm;
end architecture state_machine;
```

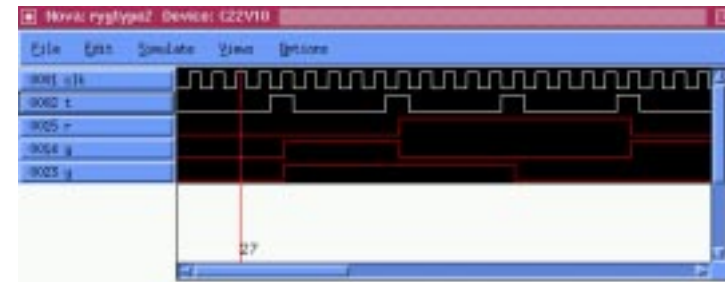
8.14 Results of rygtype2.vhd

```
g.D = t * /y.Q * /g.Q + /t * g.Q
r.D = t * y.Q + /t * r.Q
y.D = /t * y.Q + t * /r.Q
```

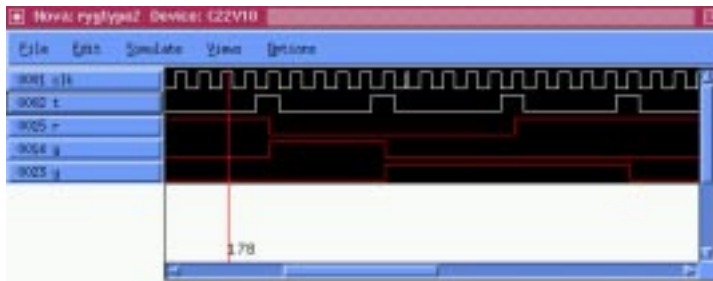
Description	Used	Max
Dedicated Inputs	1	11
Clock/Inputs	1	1
I/O Macrocells	3	10

5 / 22 = 22 %

□ Whoops! Are both green and yellow on?



8.15 But This One is OK!



The initial state was "000" for the previous simulation display which is illegal. Look at a portion of the simulation window after it has settled into its routine.

8.16 rygsv.vhd

```
--This is very close to the "best" (for this example).
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity ryg is port (
  t, clk : in std_logic;
  r, y, g : out std_logic);
end ryg;
architecture state_machine of ryg is
  signal p_s, n_s : std_logic_vector(2 downto 0);
  constant red   : std_logic_vector(2 downto 0) := "100";
  constant grn   : std_logic_vector(2 downto 0) := "001";
  constant yel   : std_logic_vector(2 downto 0) := "010";
  constant walk  : std_logic_vector(2 downto 0) := "110";
begin
  r <= p_s(2);
  y <= p_s(1);
  g <= p_s(0);
  state_clocked:process(clk)
  begin
    if rising_edge(clk) then p_s <= n_s;
    end if;
  end process state_clocked;
```

8.17 More Architecture for rygsv.vhd

```
fsm:process(p_s, t) -- combinational
begin -- case
  case p_s is
    when red =>
      if t = '1' then n_s <= grn;
      else n_s <= red;
      end if;
    when grn =>
      if t = '1' then n_s <= yel;
      else n_s <= grn;
      end if;
    when yel =>
      if t = '1' then n_s <= walk;
      else n_s <= yel;
      end if;
    when walk =>
      if t = '1' then n_s <= red;
      else n_s <= walk;
      end if;
    when others => n_s <= walk;
  end case; end process fsm;
end architecture state_machine;
```

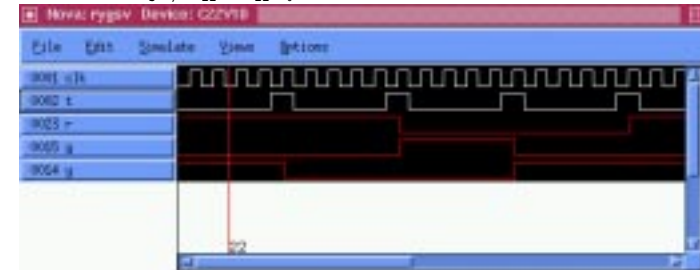
8.18 Results for rygsv.vhd

The logic is a little more complex, but it fits!

```
/r.D = t * r.Q * /y.Q * /g.Q + /t * /r.Q * y.Q * /g.Q
+ /r.Q * /y.Q * g.Q
/y.D = /t * /r.Q * /y.Q * g.Q + t * r.Q * /g.Q
+ r.Q * /y.Q * /g.Q
g.D = t * r.Q * /y.Q * /g.Q + /t * /r.Q * /y.Q * g.Q
```

Description	Used	Max
Dedicated Inputs	1	11
Clock/Inputs	1	1
I/O Macrocells	3	10

5 / 22 = 22 %



8.19 rygone.vhd

```
--This is "worst".
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity ryg is port (
  t, clk : in std_logic;
  r, y, g : out std_logic);
end ryg;

architecture state_machine of ryg is
  type StateType is (red, yel, grn, walk);
  signal p_s : StateType;
begin
  fsm:process(clk) -- clocked
  begin -- case
    if rising_edge(clk) then
      r <= '0';
      y <= '0';
      g <= '0';
    end if;
  end process fsm;
end architecture state_machine;
```

xxxxxxx
 --More of the process architecture in the next slide.

8.20 More Architecture for rygone.vhd

```
case p_s is
  when red => r <= '1';
    if t = '1' then p_s <= grn;
    else p_s <= red; end if;
  when grn => g <= '1';
    if t = '1' then p_s <= yel;
    else p_s <= grn; end if;
  when yel => y <= '1';
    if t = '1' then p_s <= walk;
    else p_s <= yel; end if;
  when walk => r <= '1'; y <= '1';
    if t = '1' then p_s <= red;
    else p_s <= walk; end if;
  when others => p_s <= walk;
end case;
end if;
end process fsm;
end architecture state_machine;
```

8.21 Results for rygone.vhd

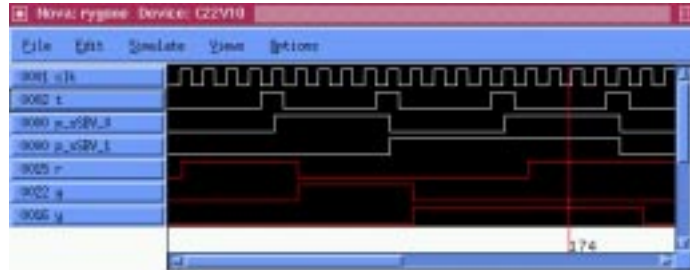
```

y.D = p_ssbv_1.Q
g.D = p_ssbv_0.Q * /p_ssbv_1.Q
r.D = /p_ssbv_0.Q * /p_ssbv_1.Q + p_ssbv_0.Q * p_ssbv_1.Q
p_ssbv_0.D = p_ssbv_0.Q * /t + /p_ssbv_0.Q * t
p_ssbv_1.D = p_ssbv_0.Q * /p_ssbv_1.Q * t
          + p_ssbv_1.Q * /t + /p_ssbv_0.Q * p_ssbv_1.Q

```

Description	Used	Max
Dedicated Inputs	1	11
Clock/Inputs	1	1
I/O Macrocells	5	10

7 / 22 = 31 %



8.22 rygonec.vhd

```

--A three way tie for the "middle".
--Also rygtype.vhd and rygtype1.vhd
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity ryg is port (
    t, clk : in std_logic;
    r, y, g : out std_logic);
end ryg;
--
architecture state_machine of ryg is
    type StateType is (red, yel, grn, walk);
    signal p_s : StateType;
begin
    fsm:process(clk) -- clocked
    begin -- case
        if rising_edge(clk) then
            r <= '0';
            y <= '0';
            g <= '0';
        end if;
    end process;
end architecture;
--More of the process architecture in the next slide.

```

8.23 More Architecture for rygonec.vhd

```

case p_s is
    when red => r <= '1';
        if t = '1' then p_s <= grn;
        else p_s <= red; end if;
    when grn => g <= '1';
        if t = '1' then p_s <= yel;
        else p_s <= grn; end if;
    when yel => y <= '1';
        if t = '1' then p_s <= walk;
        else p_s <= yel; end if;
    when walk => r <= '1'; y <= '1';
        if t = '1' then p_s <= red;
        else p_s <= walk; end if;
    when others => p_s <= walk;
end case;
end if;
end process fsm;
end architecture state_machine;

```

8.24 rygtype.vhd

```

--Another in the three way tie for the "middle".
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity ryg is port (
    t, clk : in std_logic;
    r, y, g : out std_logic);
end ryg;
architecture state_machine of ryg is
    type StateType is (red, yel, grn, walk);
    signal p_s, n_s : StateType;
begin
    state_clocked:process(clk)
    begin
        if rising_edge(clk) then p_s <= n_s;
        end if;
    end process state_clocked;
    fsm:process(p_s, t) -- combinational
    begin -- case
        r <= '0';
        y <= '0';
        g <= '0';
    end process;
end architecture;
--More of the process architecture in the next slide.

```

8.25 More Architecture for rygtype.vhd

```

case p_s is
  when red => r <= '1';
    if t = '1' then n_s <= grn;
    else n_s <= red; end if;
  when grn => g <= '1';
    if t = '1' then n_s <= yel;
    else n_s <= grn; end if;
  when yel => y <= '1';
    if t = '1' then n_s <= walk;
    else n_s <= yel; end if;
  when walk => r <= '1'; y <= '1';
    if t = '1' then n_s <= red;
    else n_s <= walk; end if;
  when others => n_s <= walk;
end case;
end process fsm;
end architecture state_machine;

```

8.26 rygtype1.vhd

```

--The last in a three way tie for the "middle"
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity ryg is port (
  t, clk : in std_logic;
  r, y, g : out std_logic);
end ryg;
--
architecture state_machine of ryg is
  type StateType is (red, yel, grn, walk);
  signal p_s, n_s : StateType;
begin
  fsm:process(p_s, t) -- combinational
  begin -- case
    r <= '0';
    y <= '0';
    g <= '0';

--More of the process architecture in the next slide.

```

8.27 More Architecture for rygtype1.vhd

```

case p_s is
  when red => r <= '1';
    if t = '1' then n_s <= grn;
    else n_s <= red; end if;
  when grn => g <= '1';
    if t = '1' then n_s <= yel;
    else n_s <= grn; end if;
  when yel => y <= '1';
    if t = '1' then n_s <= walk;
    else n_s <= yel; end if;
  when walk => r <= '1'; y <= '1';
    if t = '1' then n_s <= red;
    else n_s <= walk; end if;
  when others => n_s <= walk;
end case;
end process fsm;
state_clocked:process(clk)
begin
  if rising_edge(clk) then p_s <= n_s;
  end if;
end process state_clocked;
end architecture state_machine;

```

8.28 Results for the Three Which are Tied

They produce IDENTICAL results!

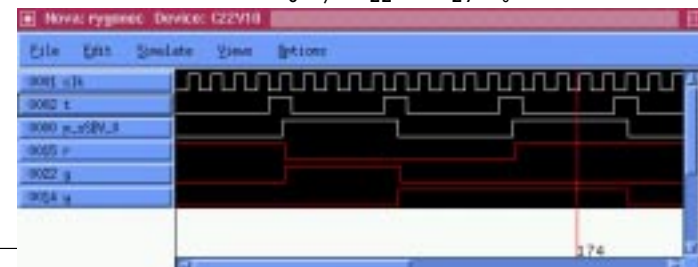
```

g = /y.Q * p_sSBV_0.Q
r = /y.Q * /p_sSBV_0.Q + y.Q * p_sSBV_0.Q
p_sSBV_0.D = p_sSBV_0.Q * /t + /p_sSBV_0.Q * t
y.D = /y.Q * p_sSBV_0.Q * t
      + y.Q * /t + y.Q * /p_sSBV_0.Q

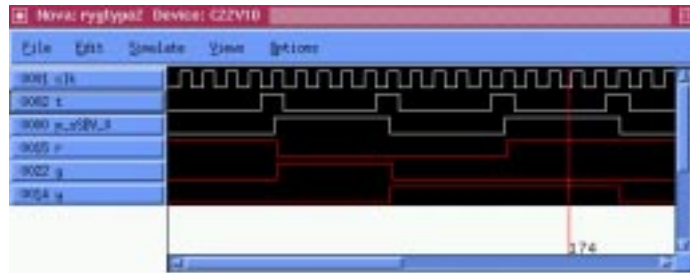
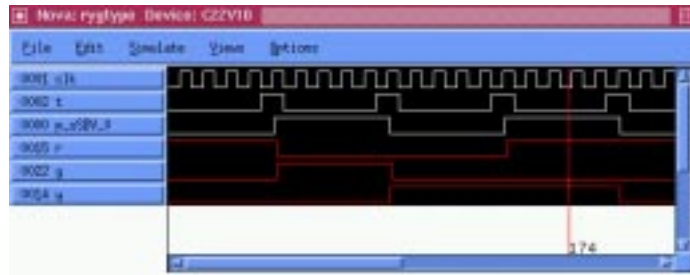
```

Description	Used	Max
Dedicated Inputs	1	11
Clock/Inputs	1	1
I/O Macrocells	4	10

6 / 22 = 27 %



8.29 The Remaining Two Results



8.30 Lab 2, Phase II Schedule

From perelman@MIT.EDU Thu Feb 22 20:37:22 2001
To: troxel@MIT.EDU
From: "Leslie C. Perelman" <perelman@MIT.EDU>
Subject: 6.111 Coop

I will be in charge of all 6.111 papers this term.

Les
Leslie C. Perelman
Director of Writing Across the Curriculum
Program in Writing and Humanistic Studies

Room 14N-233
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139
Phone: (617) 253-3375

The 6.111 secretary will transmit reports to Les Perelman and they will be returned to Anne Hunter's Office, 36-476.

8.31 Lab 2, Phase II Schedule

An Added Step (Optional)

Feb 21: Lab assignment (already done).

Feb 26: Lab 2 Design Due (oral).

Mar 7: Lab 2 Demonstration (working hardware).

Mar 12: Final report due.

Mar 15: Revised Report Due for Students using
Lab 2 as their Phase II paper.