L24.1 Quiz 2 Review

Friday, April 12, 2002
- 1:00 to 2:00 (class time)
- Room 50-340 (Walker)
- 2 handwritten `crib sheets' allowed.

General Topics

- Finite State Machines
- PALs and CPLDs
- VHDL - Interpretation NOT generation
  - Entity and Architecture declarations
  - Concurrent Statements
    - Signal Assignment
    - Instantiation
    - when-else
    - with-select-when
    - Process as a wrapper around sequential statements
  - Sequential Statements - ONLY within a process
    - if-then, elsif-then, else, endif
      - case-when-end case
  - *** How to code FSMs

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.2 Quiz 2 Review –More General Topics

- SRAM (e.g., 6264)
  - Read/Write functions
  - Chip Enable
  - Write Pulse
- ROM (e.g., 28F256A Flash Memory)
- Microcoded Control Units (MCUs)
  - Use of a counter (e.g., '163) as a sequencer
  - 2-instruction machine
- *** FSM Hierarchy
  - Major/ Minor FSMs
- Chips (concepts) to be familiar with:
  - MSI stuff - AND, OR, NAND, NOR, XOR, ...
  - Clocked D, T, J-K flip-flops and registers
  - `163 and `393 counters
  - MUXs and Decoders
  - Shift Registers
  - Comparators
- SRAMs and ROMs
- PALs and 374i CPLD

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.3 Design Rules

- *** Use Modularity (hierarchical design).
  - Small subsystems are easier to design.
  - Subsystem definition is important.

- *** Design for testability.
  - Design subsystems so they will run alone.
  - Design in break-points.

- Use Don't Cares to simplify combinational logic.
  - Avoid trap states (check use of Don't Cares).
  - Decade counters and FSMs may have surprises for illegal states.

- *** Do your logic design carefully, and first.
  - Make up block (functional) and circuit diagrams (with pin numbers).
- Use logic symbols appropriate to algebraic representation.

- Use names appropriate to assertion level.

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.4 More Design Rules

- Avoid problems from `glitches'.
  - Gate delays can (and do) cause `glitches'.
  - Glitch-free combinational logic requires (but is not guaranteed by) single input changes.
  - CLK, G, /PR, /CL inputs must NOT have glitches.
  - Carry Output from counter (e.g., '163) glitches.
  - Ensure a stable combinational output before it is sampled by CLK.
- Use proper timing.
  - Clock period < MAX(FF delay, Input changes) + CL delay + Setup.
  - Obey FF timing restrictions - setup and hold times, clock width.
  - *** Use the SAME clock edge for all edge-triggered ffs.
  - Beware of clock skew.
    - Use a tree structure to expand the clock.
  - Change inputs only (just) after the clock edge.
- Be careful of multi-ended wiring paths.
  - Avoid tri-state bus contention.
  - Account for turn-off delays.
  - Don't overload outputs (observe fanouts).

L24.5 More Design Rules

Be careful about asynchronous events.
- Synchronize all external inputs.
- Asychronous event should only change one flip-flop.
- Consider pulse width carefully. Does your application need a narrow pulse or a sustained level?
- Beware of bouncing switches.
- Use monostables sparingly (if at all).

Use memory properly.
- Avoid high-Z address to SRAM when CE is asserted.
- Avoid address changes when write pulse is true.
- Make sure your write pulse is `clean` (glitch free).

Wire properly.
- Keep wires short.
- Wire all inputs (even unused ones).
- Use bypass (decoupling) capacitors. (They are already on your kit.)
- Alternate grounds with signals in flat cables.
- Use twisted pairs (different colors!) between kits.

L24.6 More Design Rules

Don't overload your power supply.
- Use a separate power supply for motors.

Use debugging strategy.
- Debug modules systematically.
- Write short test programs or implement test FSMs.
- Check for power supply and ground on every chip.
- Is every pin wired? Why not?
- Thermal debugging may help detect bus contention.
- Use a `scope`!
- Check valid logic levels and power supply.
- Use your logic analyzer for checking sequencing.

Gating the clock
- Don't!
- If you must, do it carefully.

Account for clock skew.
- This increases effective setup and hold times.

L24.7 How to Make Your Project Work

Read (and heed) all of the handout.
It is `old' but all of it is good advice.

Sections that are particularly relevant are:

- Wiring Errors
- Care and Feeding of the Power Supply
- Unused Inputs
- Behavior of Ungrounded Parts
- Tri-State Logic Signals
- Handling CMOS Parts
- Wire Routing
- Clock Distribution
- Gating the Clock
- RAM Write Pulses
- Synchronizer Errors
- Testing Strategies
- Driving High Current Devices

L24.8 L7.10 Three Statements i1by5sv.vhd Wed. February 20, 2002

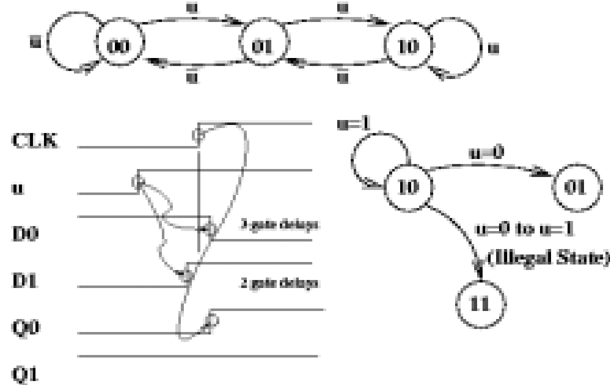
```
library ieee;
use ieee.std_logic_1164.all;
entity by5 is port (
  x, clk  : in std_logic;
  y       : out std_logic);
end by5;
architecture state_machine of by5 is
  signal p_s, n_s : std_logic_vector(2 downto 0);
  signal p_sx : std_logic_vector(3 downto 0);
  constant state0 : std_logic_vector(2 downto 0) := "000";
  constant state1 : std_logic_vector(2 downto 0) := "001";
  constant state2 : std_logic_vector(2 downto 0) := "010";
  constant state3 : std_logic_vector(2 downto 0) := "110";
  constant state4 : std_logic_vector(2 downto 0) := "100";
begin
  state_clocked:process(clk) -- register
  begin
    if rising_edge(clk) then p_s <= n_s;
    end if;
  end process state_clocked;
```

```
- - combinational output spec.
  y <= '1' when ((p_s = state4) or
       (p_s = state3) or
       ((p_s = state2) and (x = '1')))
       else '0';
- - combinational next state
specification
  p_sx <= p_s & x;
  with p_sx select
  n_s <= state0 when "0000",
        state1 when "0001",
        state2 when "0010",
        state3 when "0011",
        state4 when "0100",
        state0 when "0101",
        state2 when "0110",
        state1 when "0111",
        state3 when "1000",
        state4 when "1001",
        state0 when others;
end architecture state_machine;
```

L24.9 L7.12 Problem Transition Wed. February 20, 2002

Consider the transition from state 10 (2) to state 01 (1).

if u changes from 0 to 1 close to the clock edge:



u \ Q1 Q0	00	01	11	10
0	0	0	X	0
1	0	1	X	1

D1 = u*Q0 + u*Q1

u \ Q1 Q0	00	01	11	10
0	0	0	X	1
1	1	0	X	0

D0 = u*/Q0*/Q1 + /u*Q1

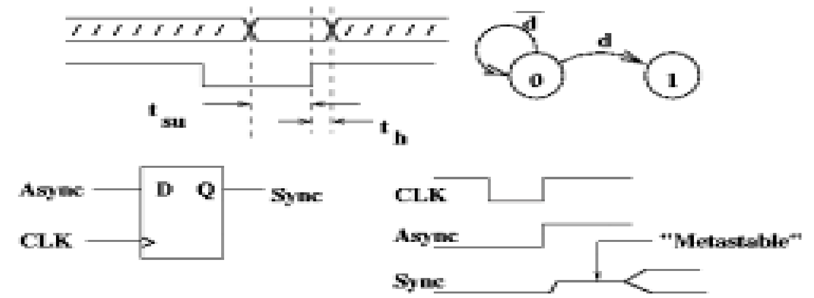
Most of the time, this circuit will work just fine. It makes a mistake and enters an illegal state ONLY when the input transition is close to a clock edge.

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.10 L7.13 Important Design Rule Wed. February 20, 2002

DESIGN RULE:

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

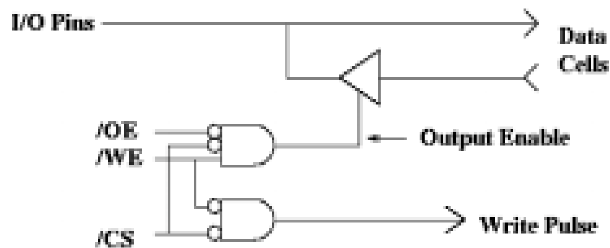


Any combinational logic with "Sync" as an input will be "glitchy" until after the metastable state has expired. In particular, do NOT use "Sync" as a CLK input.

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

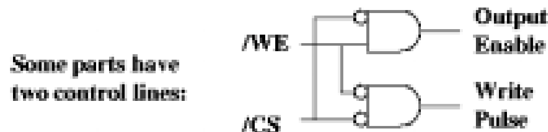
L24.11 L8.3 Control Lines Fri. February 22, 2002

The control lines are often active low. One must read the data sheet.



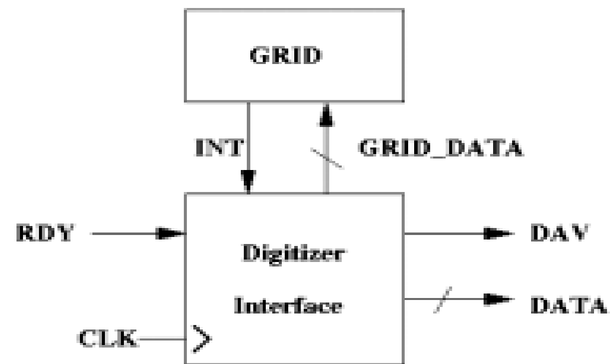
To read: pull /OE and /CS low (active)
To write: pull /WE and /CS low, /OE may be either value

Some parts have two control lines:



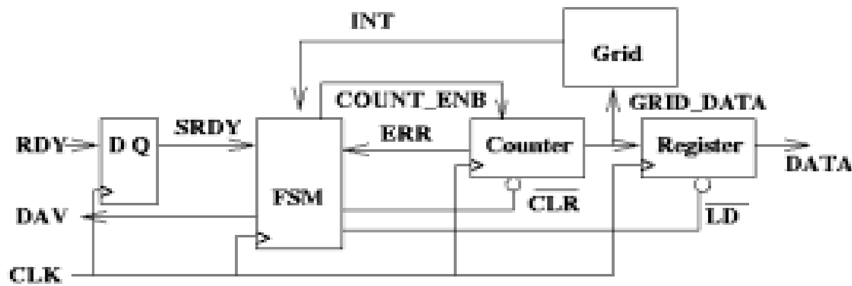
Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.12 L9.3 Go to Two Blocks Mon. February 25, 2002



RDY – The receiver is ready.
DAV – Data is available to be read.
DATA – Represents the X position of the pen.
INT – Indicates that the cursor was detected.
GRID_DATA – Specifies grid wire to be energized.

L24.13 L9.5 Enough Blocks Mon. February 25, 2002



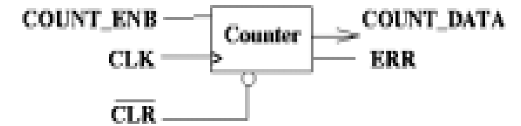
RDY – The receiver is ready.
DAV – Data is available to be read.
DATA – Represents the X position of the pen.
INT – Indicates that the cursor was detected.
GRID_DATA – Specifies the grid wire to be energized.
SRDY – A synchronized version of RDY.
COUNT_ENB – This enables the counter to count.
ERR – This indicates counter overflow without cursor detection.
CLR – This clears the counter.
LD – This loads the register with the counter data.

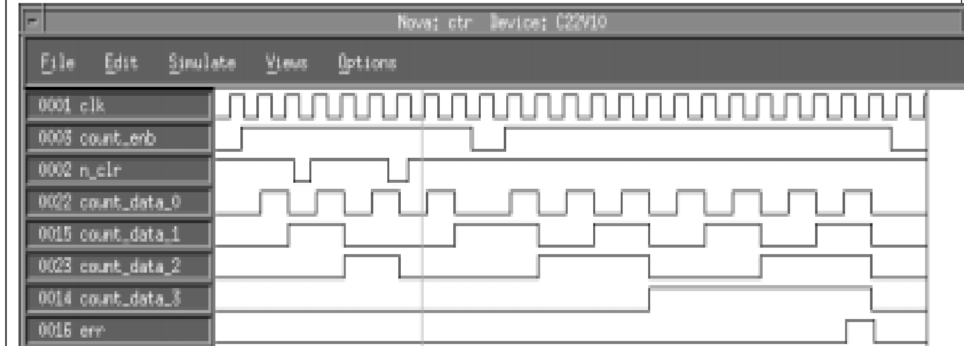
Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.14 L9.11 ctr.vhd Testing (Simulation) Mon. February 25, 2002

ERR is the carry out signal.

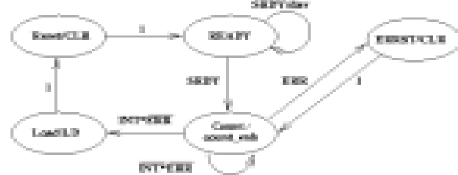
Try out n_clr and count control inputs.



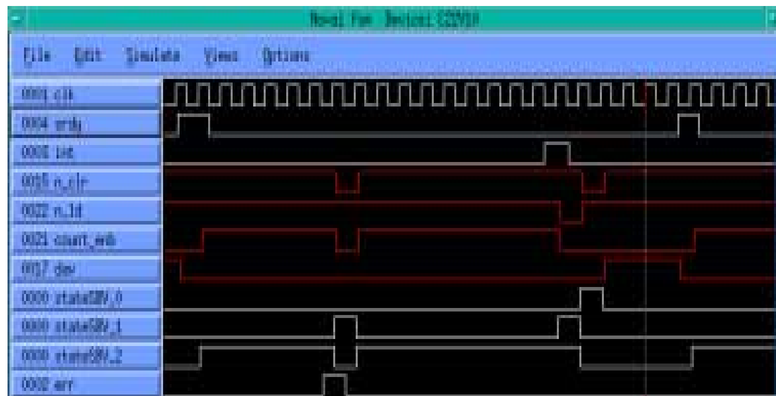


Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.15 L9.18 FSM Testing (Simulation) Mon. February 25, 2002



Exercise all state transitions.
An advantage of using constants rather than enumerated types is that the state names are visible.
Sometimes one has to poke around to see which jedec nodes encode the state!



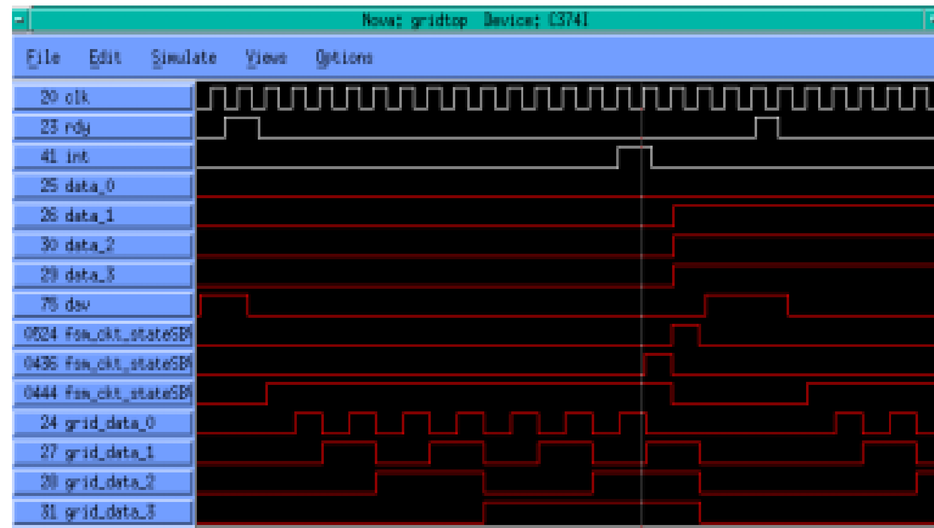
Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.16 L9.19 Pkg Declaration – gridpkg.vhd Mon. Feb.25, 2002

```
library ieee;
- - Take generic and port declarations from the entity.
use ieee.std_logic_1164.all;
package gridpkg is
  component synchronizer
  port (rdy, clk : in  std_logic;
        srdy     : out std_logic); end component;
  component fsm
  port (srdy, int, err, clk : in  std_logic;
        dav, count_enb, n_clr, n_ld  : out std_logic); end component;
  component ctr
  generic (size: integer := 4);
  port (count_enb, n_clr, clk : in  std_logic;
        err : out std_logic;
        count_data : out std_logic_vector(size - 1 downto 0)); end component;
  component reg
  generic (size: integer := 4);
  port (n_ld, clk : in  std_logic;
        count_data : in std_logic_vector(size - 1 downto 0);
        data : out std_logic_vector(size - 1 downto 0)); end component;
end gridpkg;
```

L24.17 L9.23 Overall Testing (Simulation) Mon. Feb. 25, 2002

Exercise all functions.



Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.18 L10.18 Avoiding Glitches – Register Wed. February 27, 2002

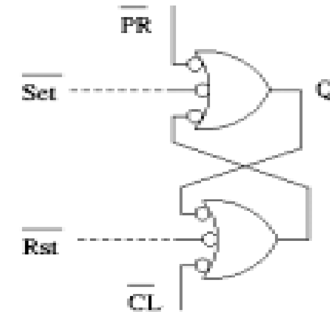
Creating glitch free outputs:

Register the output.

If a flip–flop does NOT change state upon 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, of course, that the setup and hold times are honored.

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

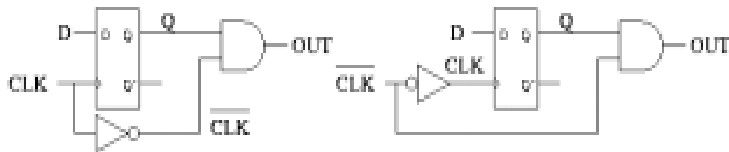
The set or reset pulses (negative true) are full pulses and only one occurs if the setup and hold times are adhered to.



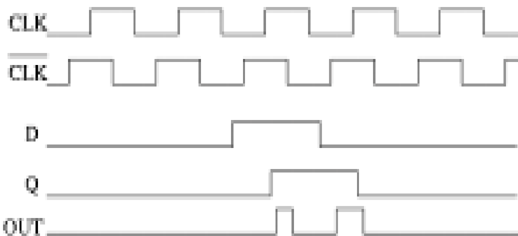
Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.19 L10.19 Avoiding Glitches – Gating Wed. February 27, 2002

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.



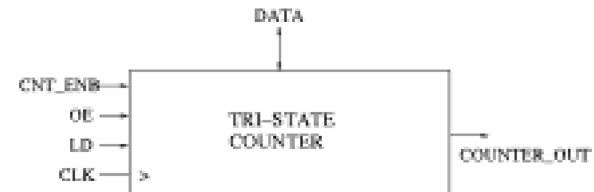
Remember – Do NOT use glitchy signals for CLK, PR, CLR, S, or R.

Clock data into a register AFTER signals are stable.

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

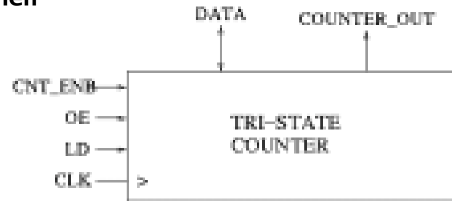
L24.20L11.9 Tri–State to Multiplex I/O pins Mon. March 4, 2002

```
-- Use tri-state logic to multiplex IO pins.
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;  -- needed for integer + signal
entity ldcnt is
  generic (width : integer := 3);
  port(clk, ld, oe, cnt_enb : in std_logic;
       -- Use counter_out_out only for simulation.
       counter_out : out std_logic_vector(width -1
downto 0);
       data : inout std_logic_vector(width - 1 downto
0));
end ldcnt;
```



L24.21 L11.10 Tri–State Architecture Mon. March 4, 2002

```
-- purpose: count with an output enable
architecture archldcnt of ldcnt is
signal counter : std_logic_vector(width - 1 downto 0);
begin
  counter_out <= counter;
  data <= counter when oe = '1' else (others => 'Z';
  -- N.B. Z must be UPPERCASE!
  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;
```

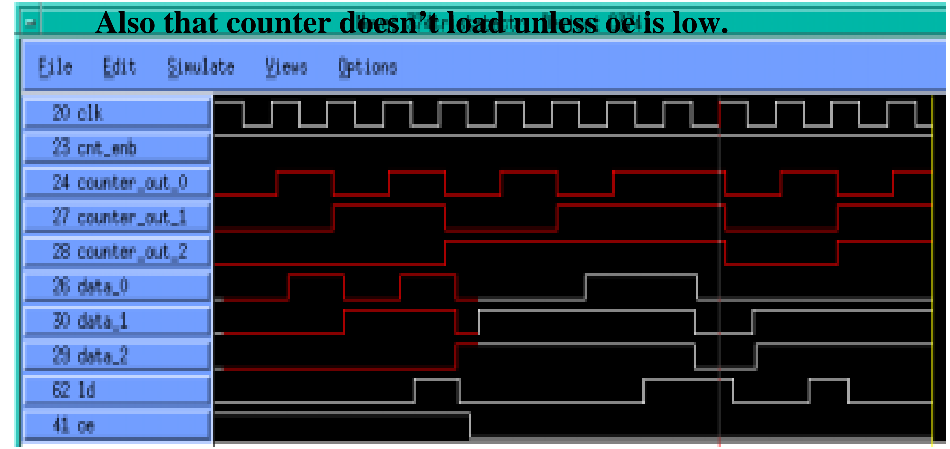


Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.22 L11.11 Simulation of Tri–State Ctr Mon. March 4, 2002

Note that data(2 downto 0) are white (meaning an input) when oe is low.

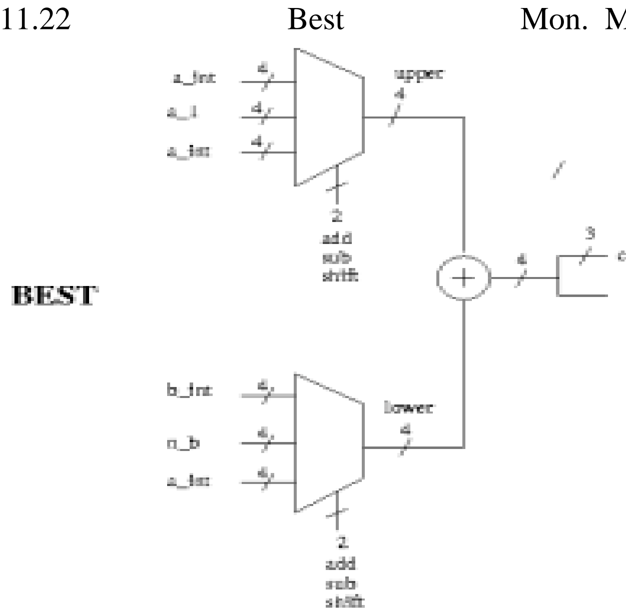
Also that counter doesn't load unless oe is low.



Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.23 L11.22 Best Mon. March 4, 2002

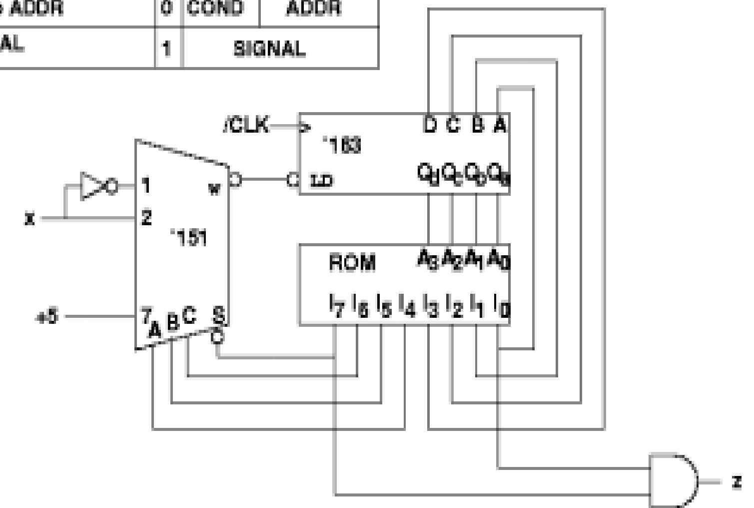
BEST



Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.24 L12.10 MSI Implementation Wed. March 6, 2002

	I7	I6 I5 I4	I3 I2 I1 I0
if COND jmp ADDR	0	COND	ADDR
assert SIGNAL	1	SIGNAL	



L24.25 L12.14 Specification File Wed. March 6, 2002

Extract of /mit/6.111/prom/examples/encr.sp

```
op<7:0>;          /*defines instruction length and location*/
address op<4:0>;  /*defines bit field the address*/
value op<4:0>;    /*allows integer values in ASSEM_FILE*/
if op<7> = 0;     /*used in making conditional jumps*/
do op<7:6> = 3;   /*this is used for making assertions*/
/*definitions of conditional signals*/
go op<6:5> = 0;
encrypt op<6:5> = 1;
a_b op<6:5> = 2;
test op<6:5> = 3;
uncond op<7:5> = 4; /*this is for an unconditional jump*/
/*definitions of assertions*/
assert_low op<5> op<4>;  /*makes default for field <5:4> high*/
select_b op<5> = 0;
select_a op<4> = 0;  /*Note that they are asserted low*/
load     op<3> = 1;
select_157 op<2:0> = %b001;
incmar     op<2:0> = %b010;
clr        op<2:0> = %b011;
shift      op<2:0> = %b110;
bit0       op<0> = %b1;
select_257 op<2> = 1 op<1> = %b1 bit0;
```

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.26 L12.15 Assembly File (Instructions) Wed. March 6, 2002

Extract of /mit/6.111/prom/examples/encr.as

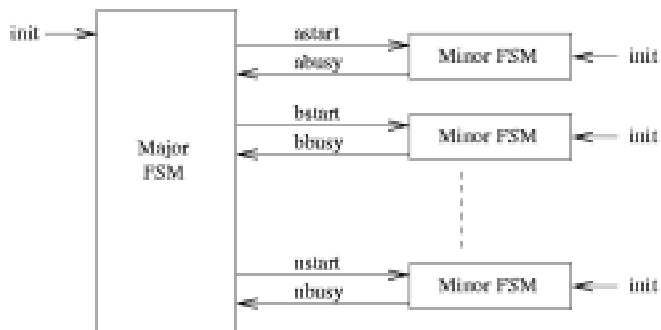
```
 # SPEC_FILE = encr.sp;   /*gives the specification filename*/
 # LIST_FILE = encr.list; /*specifies the list filename*/
 # SET_ADDRESS = 0;       /*start assembling here*/
start      : if go start;  /*The semicolon delimits one
   instruction from the next semicolon. Everything between
   semicolons is put in the same instruction */
rdata      : do load clr select_a select_b;
encheck1   : if encrypt memcheck;
applyalg   : do load select_b shift;
switch     : do load select_a select_b select_257;
             do load select_b;
donecheck  : if encrypt readyl;
memcheck   : if a_b multrep;
             do incmar;
lastloc    : if test memcheck;
             if encrypt applyalg;
             do 14; /*Note the use of integer values. a value*/
             do 11; /*field was specified in the spec file */
readyl     : do ready;
             uncond start;
multrep    : if encrypt a_multrep;
             uncond readyl;
```

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.27 L20.1 FSM Hierarchy

Problem Statement:

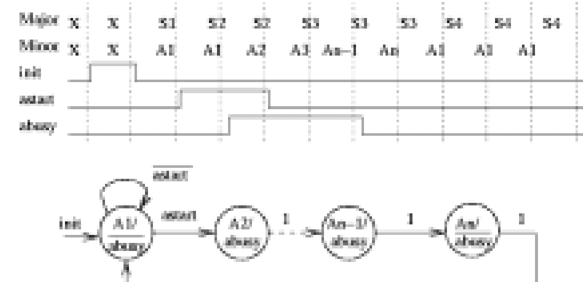
We want to coordinate (control) a sequence of operations performed by other FSMs. To do this we divide the problem into multiple FSMs which may take multiple (sometimes an unknown number) clock cycles. These minor FSMs are controlled by a major FSM. Minor FSMs may also be decomposed into multiple FSMs. All FSMs operate on the rising edge of the same clock.



Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 Introductory Digital Systems Laboratory
Wednesday
April 10, 2002

L24.28 L20.2 Minor FSMs

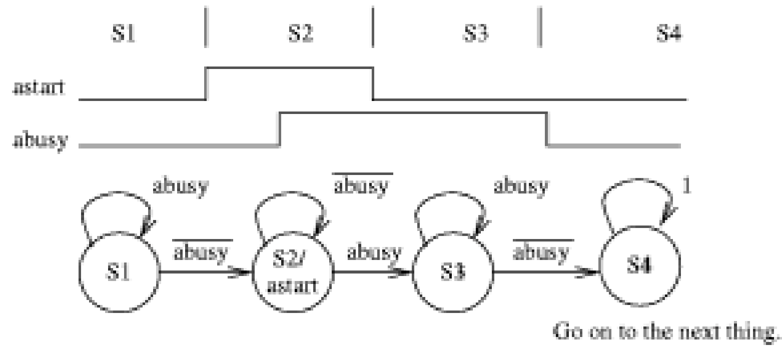
Minor FSMs are started by a control signal (start) generated by the major FSM which controls this particular minor FSM.
All minor FSMs are initialized by a signal (init) which is synchronized to the clock and asserted for a single clock period.
The makeup of minor FSMs is arbitrary, but with some constraints. The shortest path back to the starting state (reached by init) must be at least two clock cycles.



L24.29 L20.3

Major FSMs

The major FSM is initialized by the same init signal supplied to the minor FSMs.
Major FSMs generate a control signal (start) for the minor FSMs. They test the busy signal generated by the minor FSM to determine both when the minor FSM is available and when it is done.



L24.30 L20.9

Different Clocks

Suppose one or more minor FSMs use different clocks.
What then? Suppose the clocks are at different frequencies?

The answer is deceptively simple.
It is easy to implement and extremely difficult to debug.

Simply synchronize the start and busy signals BEFORE they are used to control transitions in the receiving state machine.
Make sure you synchronize ALL signals that change asynchronously with your clock.
If you are not sure, synchronize the signal.