

L9.1 Hierarchical Design Mon. February 25, 2002

Start with a one-block block diagram.

Expand to major blocks.

Repeat expansion until blocks are simple.

Implement these simple blocks and test.

Code them in VHDL and simulate.

Use structural instantiation in VHDL to wire the blocks together.

Test the design.

L9.2 One-block Block Diagram Mon. February 25, 2002

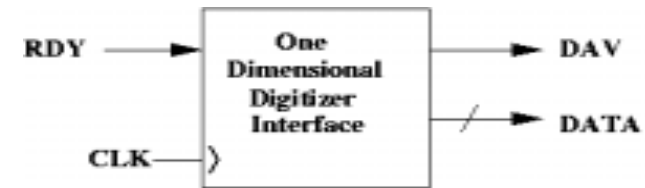
Global English Description

Describe the function.

Describe the input and output signals.

Example – Digitizer to Record Position

Simplify it so there is only one dimension.

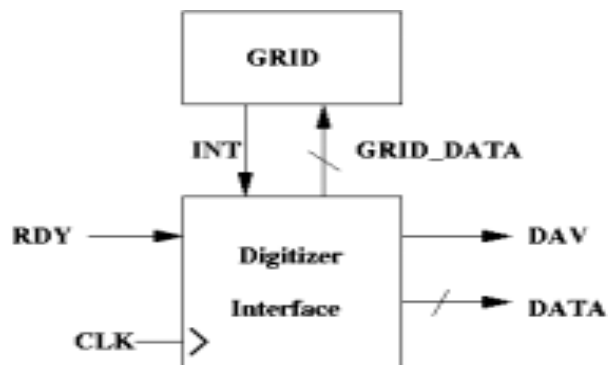


RDY – The receiver is ready.

DAV – Data is available to be read.

DATA – Represents the X position of the pen.

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.

L9.4 English Description Mon. February 25, 2002

Position detection using an array of wires

Generate magnetic field with a coil.

Count while sweeping over the array.

Detect position of a cursor:

By phase reversal

Or other artifact of signal detection

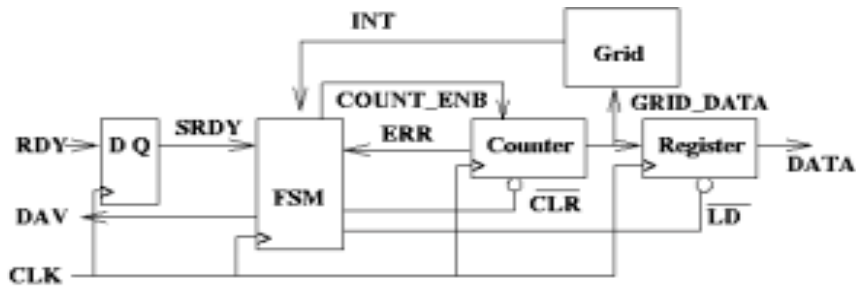
Put count into a register.

Implement a 'Handshake'.

Set handshake line (dav) when signal is ready.

Wait for ready signal (rdy) before counting.

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.

L9.6 FSM Specification Mon. February 25, 2002

Input:
 SRDY Synchronized version of RDY
 INT Cursor detected
 ERR Grid overflow (position not detected)

Output:
 DAV Data available
 /LD Load COUNT_DATA into the output register.
 /CLR Clear the counter.
 COUNT Enable the counter to count.



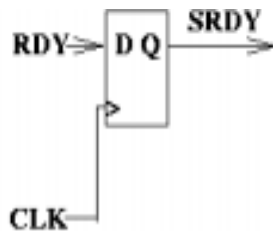
L9.7 Implement the Blocks with VHDL Mon. February 25, 2002

Don't forget to synchronize! Here is synchronizer.vhd.

```
library ieee;
use ieee.std_logic_1164.all;

entity synchronizer is
    port (rdy : in std_logic;
          srdy : out std_logic);
end synchronizer;
```

```
architecture behavioral of synchronizer is
begin -- behavioral
    sync:process(clk)
    begin
        if rising_edge(clk) then
            srdy <= rdy;
        end if;
    end process sync;
-- Why not use an internal signal and two flip-flops?
end architecture behavioral;
```



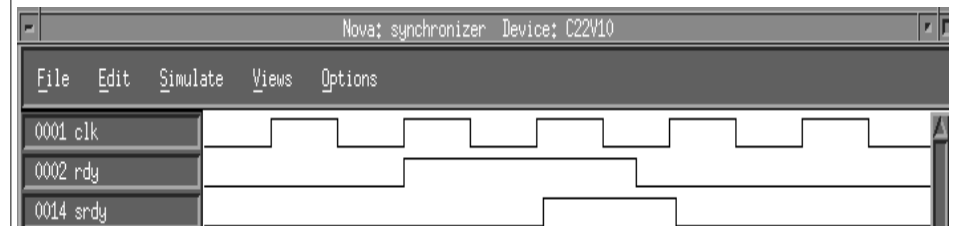
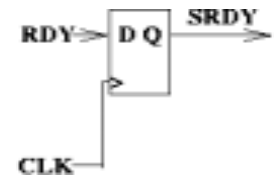
RDY – The receiver is ready.
SRDY – A synchronized version of RDY.

L9.8 Synchronizer Testing Mon. February 25, 2002

The synchronizer is simple, but important.

Should we also synchronize INT?

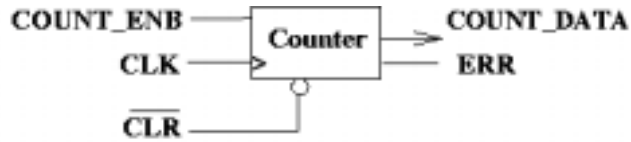
Do so if there is a doubt.
 But, INT only happens after COUNT_DATA has changed,
 So it is already synchronized.



L9.9 Ctr.vhd Mon. February 25, 2002

a clearable counter with a carry out
 library ieee;
 use ieee.std_logic_1164.all;
 use work.std_arith.all;

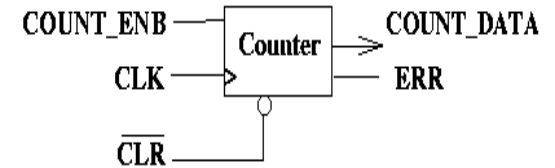
```
entity ctr is
    generic (size: integer := 4);
    port (count_enb, n_clr, clk : in std_logic;
          err : out std_logic;
          count_data : out std_logic_vector(size - 1 downto 0));
end ctr;
```



COUNT_DATA – This is counter output
COUNT_ENB – This enables the counter to count.
ERR – This indicates counter overflow without cursor detection.
CLR – This clears the counter.

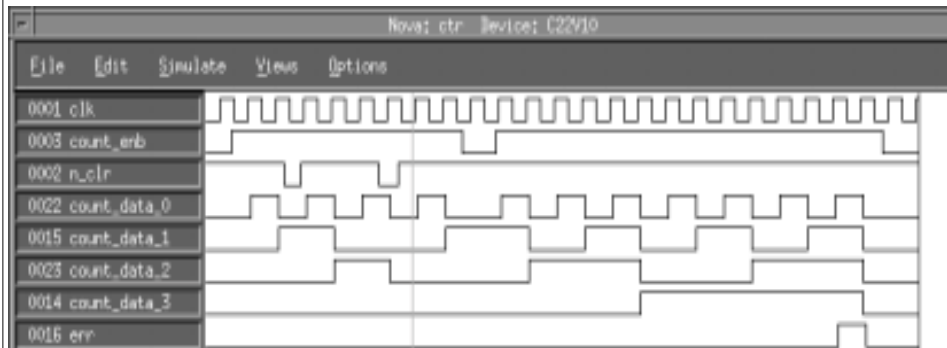
L9.10 ctr.vhd Architecture Mon. February 25, 2002

```
architecture behavioral of ctr is
    signal cnt_int : std_logic_vector(size - 1 downto 0);
    signal all_ones : std_logic_vector(size - 1 downto 0);
begin -- behavioral
    all_ones <= (others => '1');
    count_data <= cnt_int;
    err <= '1' when cnt_int = all_ones else '0';
    state_transition:process(clk)
    begin
        if rising_edge(clk) then
            if n_clr = '0' then
                cnt_int <= (others => '0');
            elsif count_enb = '1' then
                cnt_int <= cnt_int + 1;
            end if;
        end if;
    end process state_transition;
end behavioral;
```



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

ERR is the carry out signal.
 Try out n_clr and count control inputs.

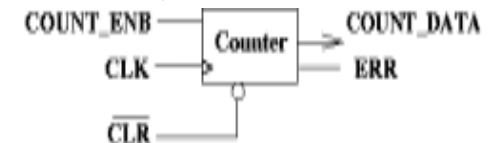


L9.12 reg.vhd Mon. February 25, 2002

- This is a loadable register whose width is a generic.
- Size has a default - one number to change.
- Instantiation as a component can define size.

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is
    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 reg;
```

```
architecture behavioral of reg is
begin -- behavioral
    regff:process(clk)
    begin
        if rising_edge(clk) then
            if n_ld = '0' then
                data <= count_data;
            end if;
        end if;
    end process;
end architecture behavioral;
```



COUNT_DATA – This is the counter output.
COUNT_ENB – This enables the counter to count.
ERR – This indicates counter overflow without cursor detection.
CLR – This clears the counter.

L9.13 regng.vhd – This Doesn't Work Mon. February 25, 2002

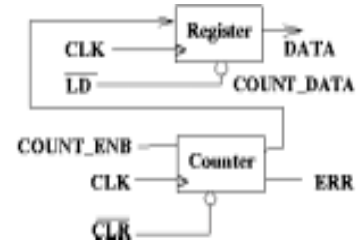
```

- - One can't combine other conditions with rising_edge(clk).
library ieee;
use ieee.std_logic_1164.all;
entity reg is
    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 reg;
architecture behavioral of reg is
begin -- behavioral
    regff:process(clk)
    begin
        if rising_edge(clk) and n_ld = '0' then -- This is the offending code.
            data <= count_data;
        end if;
    end process;
end architecture behavioral;
- - this produces
- - Warning: 'n_ld' should be referenced in the sensitivity list.
- - Warning: 'count_data' should be referenced in the sensitivity list.
- - Abort: Can't handle expression 's'event' in final equations.
    
```

L9.14 Test program for reg: uses ctr Mon. February 25, 2002

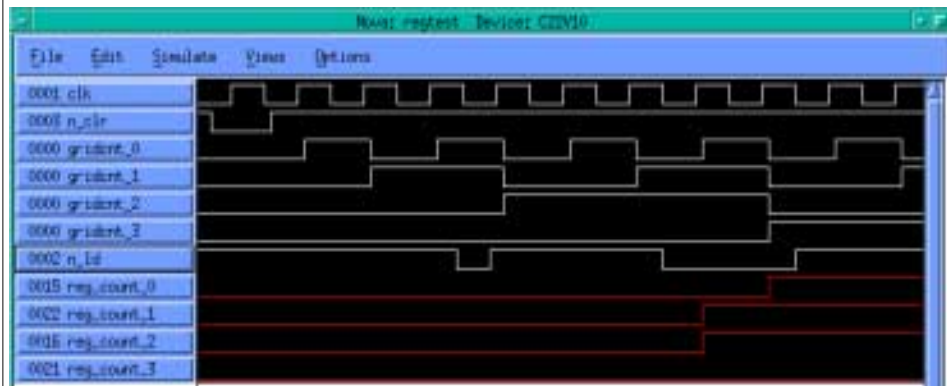
```

- - Test of register using counter as input
library ieee;
use ieee.std_logic_1164.all;
use work.gridpkg.all;
entity testreg is -- to see if the register works
    generic (gridsize : integer := 4); -- adjustable
    port (n_clr, n_ld, clk : in std_logic;
          reg_count : out std_logic_vector(gridsize-1 downto 0));
end testreg;
- - purpose: assemble counter and register
architecture test of testreg is
- - internal count
    signal gridcnt : std_logic_vector(gridsize-1 downto 0);
- - gridcnt is the internal count.
    signal err, one : std_logic; -- counter overflow
begin -- test
    one <= '1';
    count_circuit: ctr
        port map (count_enb => one, n_clr => n_clr, clk => clk,
                 err=> err, count_data => gridcnt);
    reg_circuit: reg
        port map (n_ld => n_ld, clk => clk, count_data => gridcnt,
                 data => reg_count);
end test;
    
```



L9.15 Test of reg.vhd (Simulation) Mon. February 25, 2002

BEWARE – Busses would seem to help, but the values may not display – probably a font problem.



L9.16 fsm.vhd Mon. February 25, 2002

There are multiple ways of defining states:
 Does one use constants or enumerated types?
 In some cases, one doesn't need the "efficiency" of making the state

```

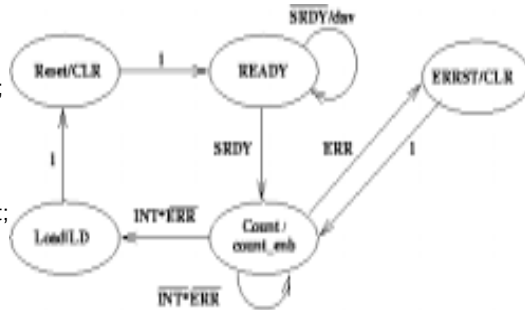
library ieee;
use ieee.std_logic_1164.all;
entity fsm is
    port (srdy, int, err, clk : in std_logic;
          dav, count_enb, n_clr, n_ld : out std_logic);
end fsm;
architecture behavioral of fsm is
    type StateType is (READY, Count, Load, ERRST, Reset);
    attribute enum_encoding of StateType: type is
        "000 001 011 010 100";
    signal state : StateType;
begin -- behavioral
    n_clr <= '0' when (state = Reset) or (state = ERRST) else '1';
    n_ld <= '0' when state = Load else '1';
    count_enb <= '1' when state = Count else '0';
    dav <= '1' when (state = READY) and (srdy = '0') else '0';
    
```

State Assignment		
	READY	Count
Count	000	001
Load	011	010
ERR	010	100
Reset	100	

L9.17 fsm/vhd Architecture Mon. February 25, 2002

```
state_transitions:process(clk)
begin
```

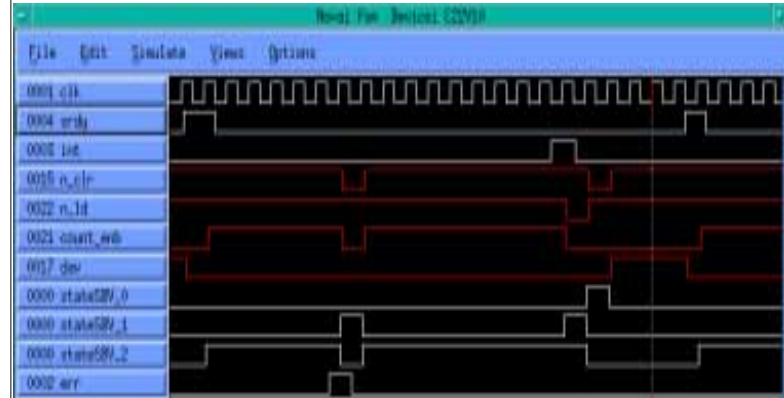
```
if rising_edge(clk) then
case state is
when READY =>
if srdy = '0' then state <= READY;
else state <= Count;
end if;
when Count =>
if err = '1' then state <= ERRST;
elsif int = '0' then state <= Count;
else state <= Load;
end if;
when Load =>
state <= Reset;
when Reset =>
state <= READY;
when ERRST =>
state <= Count;
-- don't need "when others" as all cases guaranteed
end case;
end if;
end process state_transitions;
end architecture behavioral;
```



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!



L9.19 Package Declaration – gridpkg.vhd Mon. February 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;
```

L9.20 Putting it All Together Mon. February 25, 2002

First, test all the components.

Often do this with a simpler PLD (as in a PAL) as the computation time is shorter.

After testing the components individually, put them all in a single file.

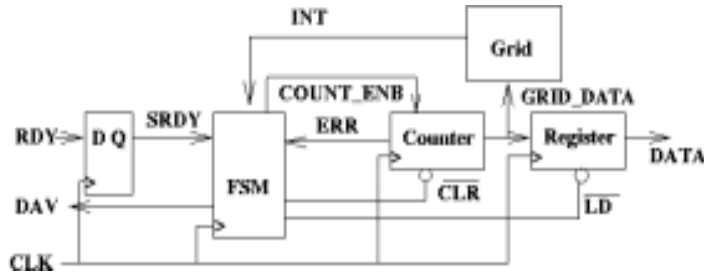
```
cat gridpkg.vhd synchronizer.vhd reg.vhd ctr.vhd fsm.vhd > all.vhd
```

Set the device to the target device, C374i.

Compile this file, all.vhd, (without it being the top design).

L9.21 gridtop.vhd Mon. February 25, 2002

```
library ieee;
use ieee.std_logic_1164.all;
use work.gridpkg.all;
entity grid is
generic (gridsize: integer := 4);
port (rdy, int, clk : in std_logic;
      dav : out std_logic;
      data : out std_logic_vector(gridsize - 1 downto 0);
      grid_data : out std_logic_vector(gridsize - 1 downto 0));
end grid;
```



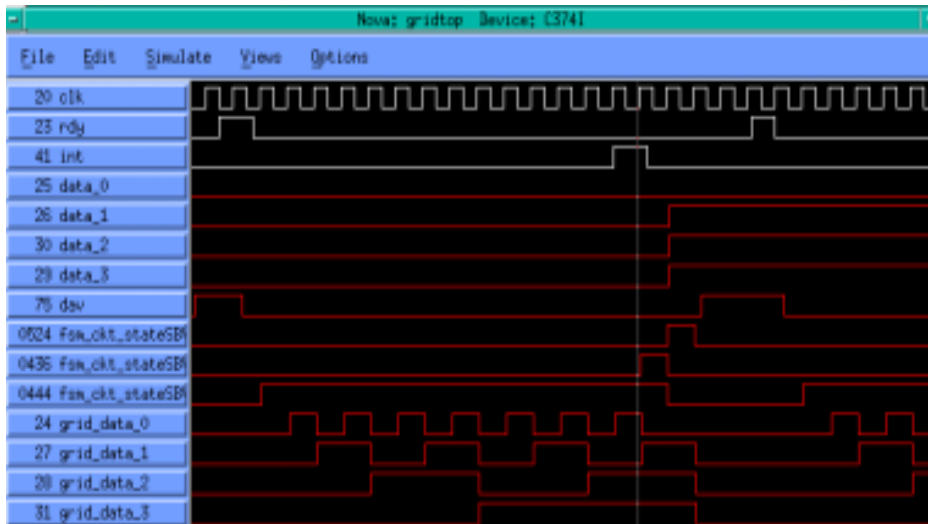
L9.22 gridtop.vhd Architecture Mon. February 25, 2002

```
architecture top of grid is
- - Note the use of generic map to specify the width of the ctr and reg.
signal srdy, err : std_logic;
signal count_enb, n_clr, n_ld : std_logic;
signal count_data : std_logic_vector(gridsize - 1 downto 0);
begin
sync_ckt: synchronizer
port map (clk => clk, rdy => rdy, srdy => srdy);
fsm_ckt: fsm
port map (srdy => srdy, int=> int, err => err,
         clk => clk, dav => dav, count_enb => count_enb,
         n_clr => n_clr, n_ld => n_ld);

ctr_ckt: ctr
generic map(size => gridsize)
port map (count_enb => count_enb, n_clr => n_clr, clk => clk,
         err => err, count_data => count_data);
grid_data <= count_data;
reg_ckt: reg
generic map(size => gridsize)
port map (n_ld => n_ld, clk => clk, count_data => count_data,
         data => data);
end top;
```

L9.23 Overall Testing (Simulation) Mon. February 25, 2002

Exercise all functions.



L9.24 Design Rules Mon. February 25, 2002

Hierarchical design

Test everything, modules and the whole system.

Synchronize all external inputs.

An asynchronous event must change ONLY one flop-flop.

Use the same clock edge for all edge triggered flip-flops. Beware of clock skew!
 Clock period \geq Max(FF delay, Input changes) + CL delay + Setup time.

CLK, /PR, /CLR, and G must NOT have glitches.

Most combinational logic is glitchy. Carry from a counter is glitchy.

Keep wires short.

Wire all inputs (even unused ones).

Avoid tri-state bus contention. Account for turn-off delays.

Avoid High-Z address top SRAM when CE is true.

Avoid address changes when write pulse is true.

Use don't cares to simplify combinational logic.

Use names and logic symbols appropriate for algebra and assertion levels.

Bypass (decoupling) capacitors are already on your kit.

Use monostables sparingly (if at all).

Gate clock pulse CAREFULLY (more on this later).