

VHDL Design Flow

LAB 1

Objective:

Familiarize yourself with the Design Analyzer and Behavioral Compilation. The design we'll work on is a FIR with 9 TAPs described in VHDL.

An FIR is simply a sliding dot product.

$$O(i) = K1*I(i-8) + K2*I(i-7) + K3*I(i-6) + \dots + K9*I(i)$$

where O is the output vector, I is the input vector and Kx are the factors of the kernel (which are constants).

1. cd to your subdirectory day1.
2. Explore the VHDL code
3. To start the Design Analyzer, type: startsynopsys-3.4b
4. Analyze the design that we want to work on:
File/Analyses...
select the file fir.vhd and click OK (you can double-click on fir.vhd as a shortcut).

5. Go to Help/Commands...

Enter "elaborate" as Command Name then click Lookup Topic.

Which option specifies to include informations for the Behavioral Compiler?

Click Cancel to close the window.

6. We have to open a command window
because File/Elaborate... doesn't include the option you just mentioned.

Setup/Command Window

7. Now "elaborate" the design "fir" by entering:

elaborate -s fir

(-s is a shorthand for the needed option)

The icons represents the designs. [Y=A+B] indicates that it is in an equation-based format and the gate indicate that it is netlist-based.

8. Select 'fir' by clicking on the icon over the name and go down in that design by clicking the down arrow. (as a shortcut you can double-click on the icon). This is called the symbol view.

What are the inputs to the design?

What are the outputs?

Compare with the VHDL code

9. Now we will ask to estimate timings:

`bc_time_design`

What is the minimum clock rate if we want to chain the mul and add in one cycle? (it is the maximum delay to result_aut)

10. Select the input 'clk' and go to Attributes/Clocks/Specify...

Set Period to 50, click Apply then Cancel.

You will see a little wave over the 'clk' input to indicate that the input is driven by a clock. You can double click on it to change the clock.

11. Verify that chaining is enabled:

Setup/Variables...

Select the group "bc" and the variable "bc_enable_chaining" and make sure the value is "true". Click Apply, then Cancel.

12. We can now save the timed design so that we can try different mappings without recomputing timings:

File/Save As...

File Format: DB

File Name: timed_fir.db

Click OK.

13. Start the schedule process

(by default it optimizes for speed, with low effort). The "-io super" option specifies that we want to keep IO order but not necessarily the time between IOs.

`schedule -io super`

14. Ask to generate a report

(to know the different possible reports you can use Help/Commands...)

`report_schedule -summary`

How many cycles does the 'main' loop takes? _____

How many multipliers does the design takes?

How many adders does the design takes?

What is the total area?

15. Now we will try to reduce the cost of the design.

Reload the unscheduled design:

File/Read... and double-click fir.db

16. To reduce the cost we can ask to schedule to minimize area instead of delay:

schedule -io super -area

How many cycles does the 'main' loop takes?

How many multipliers does the design takes?

How many adders does the design takes?

What is the total area?

You can see that it is much smaller, but chances are that this design is not the smallest. You could have used the option "-effort high" to have better results but if you have an idea of what you want, you can give constraints to achieve desired results. We can say that we want a design that takes exactly 8 cycles and that we want the result to be written in the last cycle. These constraints will guide the compiler more directly to the solution we want.

17. Reload the unscheduled design

18. Set constraints on the number of cycles:

```
set_cycles 8 -from_begin fir_main/reset_loop/main -to_end fir_main/reset_loop/main
set_cycles 1 -from fir_main/reset_loop/main/result_aut -to_end fir_main/reset_loop/main
```

19. Schedule the design:

schedule -io super

How many multipliers does the design takes?

How many adders does the design takes?

20. Now if we're satisfied with this design we can write out the RTL description of it.

File/Save As...

File Format: DB

File Name: fir_schedule.db

21. Now we can compile the design to have it at gate level:

Tools/Design Optimization

22. Look at some reports using Analysis/Report...

23. Now go to the Schematic View (with the gate button on the left icon bar, when you are in Symbol View).

In this view you can ask to hilite the critical path with Analysis/Highlight/Critical Path. You can zoom using right-click/Zoom and selecting a region.

Lab1 VHDL code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
package coeffs is
  type coef_arr is array (0 to 8) of signed(7 downto 0);
  constant coefs: coef_arr := coef_arr'(
    "00000001", "00001000", "00011100", "00111000", "01000110",
    "00111000", "00011100", "00001000", "00000001");
end coeffs;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.coeffs.all;
```

```
entity fir is
  port (clk, reset: in std_logic;
        sample: in signed (7 downto 0);
        result: out signed (9 downto 0));
end fir;
```

architecture beh of fir is

begin

 fir_main: process

 type shift_arr is array (8 downto 0)
 of signed (7 downto 0);

 variable tmp: signed (7 downto 0);

 variable pro: signed (15 downto 0);

 variable acc: signed (17 downto 0);

 variable shift: shift_arr;

 begin

 reset_loop: loop

 for i in 0 to 7 loop -- zero out the shift register

 shift(i) := (others => '0');

 end loop;

 result <= (others => '0');

 wait until clk'event and clk = '1';

 if reset = '1' then exit reset_loop; end if;

 main: loop

 tmp := sample;

 pro := tmp * coefs(0);

 acc := conv_signed(pro, 18);

 for i in 7 downto 0 loop

 pro := shift(i) * coefs(i + 1);

 acc := acc + conv_signed(pro, 18);

 shift(i + 1) := shift(i);

 end loop;

 shift(0) := tmp;

 result <= acc(17 downto 8); -- synopsys line_label aut

 wait until clk'event and clk = '1';

 if reset = '1' then exit reset_loop; end if;

 end loop main;

 end loop reset_loop;

 end process;

end beh;

LAB 2

Objective:

In this lab we will use a different IO mode and we will map an array to a RAM.

The design is a simple wave generator which uses a wave table and linear interpolation. We can change the wave and the playback frequency but not while the wave is played.

1. cd to your subdirectory lab2.
2. Take a look at the file .synopsys_dc.setup that, this time, also specify which synthetic libraries to use.
3. Also look at inter.vhd and notice that there are no clock specifications in the loops.
4. Edit the VHDL code to map the array to a RAM.
5. Start the Design Analyzer.
6. Analyze then elaborate the design 'inter' (as in lab1).
7. Now specify a clock period of 30 ns (on pin 'clk') and schedule the design like in lab1.

Are there any errors? _____

In the supersate_fixed IO mode, that we used in lab1, we must specify at least one clock cycle in each loop. In the free_floating IO mode, the scheduler can place IO operations at any clock cycle, so we don't have to specify clock boundaries. The wait specified at the end of the reset_loop is just so that Synopsys can infer a global synchronous reset.

8. Retry the scheduling with the free_floating IO mode.
schedule -io free

How many cycles does the read loop takes? _____

How many cycles does the write loop takes? _____

What is the estimated area? _____

9. Look at the states for the read loop in the abstract FSM.

report_schedule -abs

Which operations are in the:

first cycle of the loop? _____

second cycle of the loop? _____

third cycle of the loop? _____

Is there any chained operations? _____

We can see that the RAM is fast; it is made of d flip/flop, so it's also big. Synopsys doesn't recommend to use a DW03_ram with a size larger than 64 words.

10. Modify the VHDL source so that we can change the data in the RAM while the wave is playing.

To do this you must have only one loop and move the read out of the case. You must have one read and one write address; use the cmd that was used for read to change the write address. (a solution is in the file solutions/inter_fd.vhd)

11. Schedule the design with the same parameters as the last one.

How many cycles does the loop takes? _____

What is the estimated area? _____

12. Look at the operation schedule report.

What do you think is limiting the speed of the loop? _____

13. Now change the DW03_ram1 for a DW03_ram2 (which has dual port).

Does the loop takes less cycles? _____

In the next lab we will see how to do a faster design with pipelining.

VHDL code lab 2

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
library synopsys;
use synopsys.attributes.all;
library dwave;
use dwave.behavioral.all;

entity inter is
port(  clk:in bit;
      reset:in bit;
      din:in unsigned (15 downto 0);
      cmd:in integer range 0 to 3;
      dout:out signed (15 downto 0) );
end inter;

architecture beh of inter is
    type ram_type is array (0 to 2**8-1) of signed (7 downto 0);
begin
inter_main: process
    variable addr: unsigned (15 downto 0);
    variable incr: unsigned (15 downto 0);
    variable ss:ram_type; -- uncomment next 3 lines to map variable to ram
    -- constant ss_ram : resource := 0;
    -- attribute variables of ss_ram : constant is "ss";
    -- attribute map_to_module of ss_ram : constant is "DW03_ram1_s_d";
    -- Note: 1_s_d is sigle port synchronous d flip/flop

function interpolate(s1,s2: signed (7 downto 0); t: unsigned (7 downto 0))
    return signed is
begin
    return conv_signed(s1 + t*(s2-s1),16);
end interpolate;

begin
    reset_loop:loop

    case cmd is
    when 0 =>
        read:loop
            dout <= interpolate(ss(conv_integer(addr(15 downto 8))),
                               ss(conv_integer(addr(15 downto 8))+1),
                               addr(7 downto 0));
```

```
    addr := addr + incr;
  end loop;
when 1 =>
  write:loop
    ss(conv_integer(addr(15 downto 8))) := conv_signed(din(7 downto 0),8);
    addr(15 downto 8) := addr(15 downto 8)+1;
  end loop;
when 2 =>
  addr := din;
when 3 =>
  incr := din;
end case;
wait until clk'event and clk = '1';
if reset = '1' then exit reset_loop; end if;
end loop;
end process;
end beh;
```

LAB 3

Objective:

Experiment speed optimization techniques and in particular using pipelined loops.
The design is the same as lab2.

1. cd to your subdirectory lab3.

The file inter.vhd is like the one at the end of lab2.

2. Start the Design Analyzer.
3. Analyze and elaborate the design 'inter'.

4. Set the clock to 33 MHz (30 ns clock period).

We have seen in lab2 that the loop takes 4 cycles, now we will try to pipeline it to have better performances.

5. Pipeline the design

The initialization interval must be at least 2 because a memory write takes 2 cycles of the same resource.

We know that the latency must be at least 4 and it must be a multiple of the initialization interval so we can try 4.

```
pipeline_loop inter_main/the_loop -latency 4 -init 2
```

Try to schedule the design.

Why does it fails? _____

The compiler assumes that there are dependencies between read and writes in our loop because it doesn't know that the addresses are not the same.

We can specify that we want to ignore these possibly false dependencies.

6. Now we ask to ignore precedencies on all memory operations:

```
labels = find("-hier", "cell", "MEM*")
ignore_memory_loop_precedences labels
ignore_memory_precedences -from labels -to labels
```

7. Retry to schedule.

It still fails (but with much less information on why).

8. We can try with an initialization interval of 3 cycles:

```
pipeline_loop inter_main/the_loop -latency 6 -init 3
```

9. Retry to schedule.

Now it works! And if you don't ignore memory precedences it fails also with this initialization interval.

To get the cycles down, because we always read consecutive locations, we could use interleaved memory.

10. Modify the VHDL source to have two RAMs, one for even and one for odd addresses.

(a solution is in the file solutions/inter_fd2i.vhd)

11. We can schedule the design with an interval of 2 cycles at 40 MHz (25 ns clock period) with a latency of 8 cycles.

What is the estimated area? _____

How does it compares to the area of the initial design (at start of lab2)? _____

VHDL code LAB 3

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
library synopsys;
use synopsys.attributes.all;
library dwave;
use dwave.behavioral.all;

entity inter is
port(  clk:in bit;
      reset:in bit;
      din:in unsigned (15 downto 0);
      cmd:in integer range 0 to 3;
      dout:out signed (15 downto 0) );
end inter;

architecture beh of inter is
    type ram_type is array (0 to 2**8-1) of signed (7 downto 0);
begin
inter_main: process
    variable raddr: unsigned (15 downto 0);
    variable waddr: unsigned (7 downto 0);
    variable incr: unsigned (15 downto 0);
    variable ss:ram_type; -- next 3 lines map variable to ram
    constant ss_ram : resource := 0;
    attribute variables of ss_ram : constant is "ss";
    attribute map_to_module of ss_ram : constant is "DW03_ram2_s_d";
    -- Note: 2_s_d is dual port synchronous d flip/flop

function interpolate(s1,s2: signed (7 downto 0); t: unsigned (7 downto 0))
    return signed is
begin
    return conv_signed(s1 + t*(s2-s1),16);
end interpolate;

begin
    the_loop:loop
        dout <= interpolate(ss(conv_integer(raddr(15 downto 8))),
                           ss(conv_integer(raddr(15 downto 8))+1),
                           raddr(7 downto 0));
        raddr := raddr + incr;

    case cmd is
```

```
when 0 =>
  waddr := din(15 downto 8);
when 1 =>
  ss(conv_integer(waddr)) := conv_signed(din(7 downto 0),8);
  waddr := waddr+1;
when 2 =>
  raddr := din;
when 3 =>
  incr := din;
end case;
wait until clk'event and clk = '1';
if reset = '1' then exit the_loop; end if;
end loop;
end process;
end beh;
```

LAB 4

Objective:

Familiarize yourself with RTL compiling and optimization of hierarchical designs. Two different modes for optimizing will be shown, one that optimizes each part locally (maintaining hierarchy) and the other first collapses the hierarchy to do a more global optimization. The design is a PCM coder and decoder that is similar to A-law. The encoder takes a 12 bit signed PCM signal and output an 8 bit (1 for sign, 3 for exponent and 4 for mantissa). This is a lossy coding: that is you can't recover the exact original data from the code.

1. Look at the files coder.vhd, decoder.vhd and codec.vhd.
2. Complete the decoder where there are comments describing what to do.
3. Start the Design Analyzer.

4. First we analyze the designs:

File/Analyze...

You can select multiple files by using the middle mouse button:

left click: coder.vhd

middle click: decoder.vhd

middle click: codec.vhd

Then click OK

5. Now we can elaborate the top design:

File/Elaborate...

Click on library WORK

Click on design CODEC(SCHEMATIC)

OK

Instead of Analyze then Elaborate, we can use Read which does that automatically.

You should see the icons for the three designs.

6. Select the CODEC and link the lower modules:

Analysis/Link Design...

OK

7. We will optimize and compile the CODEC for minimum area.

Select the CODEC
Attributes/Optimization Constraints/Design Constraints...
Max Area: 0
Apply
Cancel
Tools/Design Optimization...
OK

8. Take a look at the Compile Log window. Which designs were optimized?

This is known as "hierarchical compile".

Look at the area for each trial. Has the compiler been able to optimize? _____

9. Ask for report on the area, timing and references.

Analysis/Report...
Area, Timing, References
Apply
Cancel

Does the hierarchy still exist? _____

Total area: _____ CODER area: _____ DECODER area: _____

Time of critical path: _____

10. Go to the schematic view of the CODEC. (By using the down arrow and the gate button on left icon bar)

11. You can click on a sub-design and go down into it, walk around the hierarchy.

12. From the schematic view of the CODEC, save the design.

13. Collapse one level of the hierarchy:

Select the CODER and DECODER instances. (middle click to select more than one)

Edit/Ungroup...
Ungroup One Level
OK

14. Now reoptimize the design.

Total Area: _____ Critical Path: _____

Compare the results with hierarchical optimization.

Collapsing the hierarchy can lead to a better realization of the circuit because it doesn't try only to optimize each part individually without changing their compartment, it does optimization on the whole circuit.

This can be very time consuming for large circuits and thus it should be used with care.

We have seen in this example that some times it doesn't help to collapse. We will now try the same thing on another design.

15. Modify the codec.vhd source so that it decodes then encode instead of the other way.

16. Redo the whole process with this new design.

Hierarchical compile area: _____

Collapsed compile area: _____

What happend? _____

The coding used is lossy, so when you encode and then decode you don't have the same data in input and output. The encode-decode could be optimized but it is hard to find how by checking the circuits (the compiler can't guess that the decoder is approximately the inverse of the coder). On the other hand, if you decode then encode you'll have the same encoded stream as the original. When you collapse the hierarchy, the compiler can simplify the big function and find that the outputs are the same as the inputs. That's why there's no more circuit, only a little connection.

VHDL code LAB 4

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

```
entity CODEC is
  port(DIN: in STD_LOGIC_VECTOR(11 downto 0);
        DOUT: out STD_LOGIC_VECTOR(11 downto 0));
end;
```

```
architecture SCHEMATIC of CODEC is
  signal CONNECT : STD_LOGIC_VECTOR(7 downto 0);
```

```
  component CODER
    port(DATA: in STD_LOGIC_VECTOR(11 downto 0);
          DOUT: out STD_LOGIC_VECTOR(7 downto 0));
  end component;
```

```
  component DECODER
    port(DATA: in STD_LOGIC_VECTOR(7 downto 0);
          DOUT: out STD_LOGIC_VECTOR(11 downto 0));
  end component;
```

```
begin
  CODE: CODER Port Map (DIN, CONNECT);
  DECODE: DECODER Port Map (CONNECT, DOUT);
end SCHEMATIC;
```

```
-- TEMP_DATA is the input data (if the sign is 1, TEMP_DATA is inverted)
```

```
--
-- TEMP_DATA =          gives as DOUT:
-- 0000000wxyz          s000wxyz
-- 0000001wxyz          s001wxyz
-- 000001wxyz           s010wxyz
-- 00001wxyzab          s011wxyz
-- 0001wxyzabc          s100wxyz
-- 001wxyzabcd          s101wxyz
-- 01wxyzabcde          s110wxyz
-- 1wxyzabcdef          s111wxyz
```

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

```

entity CODER is
  port(DATA: in STD_LOGIC_VECTOR(11 downto 0);
        DOUT: out STD_LOGIC_VECTOR(7 downto 0));
end;

architecture LOGIC of CODER is
begin
  process(DATA)
    variable TEMP_DATA : STD_LOGIC_VECTOR(10 downto 0);
    variable SIGN : STD_LOGIC;
    variable EXP : STD_LOGIC_VECTOR(2 downto 0);
  begin
    SIGN := DATA(11);
    if (SIGN = '0') then
      TEMP_DATA(10 downto 0) := DATA(10 downto 0);
    else
      TEMP_DATA(10 downto 0) := not DATA(10 downto 0);
    end if;

    EXP := "111";
    for I in 6 downto 0 loop
      if (TEMP_DATA(I+4) = '1') then-- check for the first 1
        EXP := CONV_STD_LOGIC_VECTOR(CONV_UNSIGNED(I,3),3);
        TEMP_DATA(3 downto 0) := SHR(TEMP_DATA, EXP)(3 downto 0);
        exit;
      end if;
    end loop;

    DOUT(7) <= SIGN;
    DOUT(6 downto 4) <= EXP;
    DOUT(3 downto 0) <= TEMP_DATA(3 downto 0);
  end process;

end LOGIC;

```

```

-----
-- TEMP_DATA=          for input DATA=
-- 0000000wxyz         s111wxyz
-- 0000001wxyz         s000wxyz
-- 000001wxyz          s001wxyz
-- 00001wxyzab         s010wxyz
-- 0001wxyzabc         s011wxyz
-- 001wxyzabcd         s100wxyz
-- 01wxyzabcde         s101wxyz
-- 1wxyzabcdef         s110wxyz
--

```

-- The result is inverted if the sign bit (s) is 1

Library IEEE;

use IEEE.std_logic_1164.all;

use IEEE.std_logic_arith.all;

use IEEE.std_logic_unsigned.all;

entity DECODER is

port(DATA: in STD_LOGIC_VECTOR(7 downto 0);

DOUT: out STD_LOGIC_VECTOR(11 downto 0));

end;

architecture LOGIC of DECODER is

begin

process(DATA)

variable TEMP_DATA : STD_LOGIC_VECTOR(10 downto 0);

variable SIGN : STD_LOGIC;

variable EXP : STD_LOGIC_VECTOR(2 downto 0);

begin

SIGN := DATA(7);

EXP := DATA(6 downto 4);

TEMP_DATA(3 downto 0) := DATA(3 downto 0);

-- Check the EXP

-- if it is 111...

-- if not, shift the data...

-- If SIGN = 1 complement the data

DOUT(11) <= SIGN;

DOUT(10 downto 0) <= TEMP_DATA;

end process;

end LOGIC;

LAB 5

Objective:

Experiment with finite state machines and different encoding styles.

The design is a finite state machine that controls the unlocking of a 3 button door lock system.

If you enter the correct code, an UNLOCK signal is sent.

1. First take a look at the lock.vhd file and complete it.

2. Start the Design Analyzer

3. Read in the design:

File/Read...

lock.vhd

4. Look at the log.

What is the width of the state register? _____

Before we can extract an FSM from the equation based representation of the VHDL code we must replace the synthetic components with mapped netlists.

5. To do that, open a command window and ask to:

replace_synthetic

6. Now we can extract the FSM:

Select the LOCK design

Tools/Finite State Machines...

Extract FSM from Netlist/Define the Legal States

OK

Cancel

In a more complex design, which have a little more than the FSM (like latched output), we should group the components of the FSM before the extraction.

The icon now says that the design is in state table format. In that format you can easily change the encoding style and ask to minimize the number of states.

Save the design in the state table format as lock.st:

File/Save As...

File Format: State Table

File Name: lock.st

7. You can look at the file to verify if it matches the described FSM.

Is it correct? _____

8. We will first compile the FSM using the default encoding generated by the analyzer:

Tools/Finite State Machines...

State Encodings

OK

Compile

OK

9. Now look at some reports:

Analysis/Report...

FSM

Area

Timing

Set Options...

Path Delay Type: Maximum

All Register Data Pins

OK

Apply

What is the encoding length? _____

What is the cell area? _____

How long is th critical path? _____

To specify a new encoding style we must return to the state table format. Reload the file you just saved (with Read).

As a second encoding we'll ask Synopsys to try to find by itself a good one.

10. We remove all predefined encodings and then recompile:

Select the LOCK design

Tools/Finite State Machines...

State Encodings

Clear All

OK

Compile

OK

What is the encoding length? _____

What is the cell area? _____

How long is th critical path? _____

11. Reload the file to try a last encoding style.

We'll now try one-hot:

Tools/Finite State Machines...

State Encodings...

One Hot

OK

Compile

OK

What is the encoding length? _____

What is the cell area? _____

How long is th critical path? _____

Which encoding style was the smallest? _____

Which encoding style was the fastest? _____

Now we will look a a bigger design. Look at the file proto.vhd, it is the description of a communication protocol.

12. Read the design.

13. Replace the synthetic components with mapped netlists.

14. Now we must separate the FSM from the rest of the design.

For simple designs we can use the "group -fsm" command which will try to do it by itself. This always find something that is an FSM but may have a lot of inputs.

In this design we will separate it by hand (which is not generally a good idea). The normal way to have a well separated design is to write the VHDL code for each part and to use hierarchy.

So here is presented away to separate a design if we don't have the high level description of it.

15. First we must find a cut that looks fine. For this design we have found a "good looking" cut.

Separate the design using the following commands:

```
find cell {bound* cnt* m* n* r* *cell* lte* U108 U194 U195 U196 U197 U198 U199 U200  
U216 U217 U218 U219 U235 U236 U240 U242 U243 U247 }
```

```
group dc_shell_status -design_name ET_data_path -cell_name data_path -except {*cell*40/  
U73/U1}
```

```
group -design_name ET_fsm -cell_name fsm -except {Logic0 Logic1 data_path}
```

Go down in the design ET_fsm.

What are the inputs of the FSM?

What are the outputs in general? _____

Does it look normal for a control FSM? _____

16. Now we can extract the FSM:

Tools/Finite State Machines...

Extract FSM from Netlist/Define the Legal States

17. Then we have to specify a valid state and ask to extract all the reachable states from this one.

State: S0

Encoding: 0000

Edit/Add

Also extract all reachable states

OK

Cancel

18. Then we can save it as a state table.

19. Look at the state table.

It is hard to recognize the protocol but in this format you can change the encoding and minimize the number of states.

LAB 5 VHDL code

```
Library IEEE;
use IEEE.std_logic_1164.all;

entity LOCK is
  port(X:          in STD_LOGIC_VECTOR(2 downto 0);
       CLOCK, RESET: in STD_LOGIC;
       UNLOCK:     out STD_LOGIC);
end;

architecture FSM of LOCK is
  type STATE_TYPE is (S0, S1, S2, GOOD, BAD);
  signal CURRENT_STATE, NEXT_STATE: STATE_TYPE;

  attribute state_vector : STRING;
  attribute state_vector of FSM: architecture is "CURRENT_STATE";
begin
  process(CURRENT_STATE, X)
  begin
    NEXT_STATE <= BAD;
    UNLOCK <= '0';
    case CURRENT_STATE is
      -- Add the transitions here
      -- Make the lock to recognize the sequence 010, 001, 100
      when GOOD =>
        UNLOCK <= '1';
        NEXT_STATE <= GOOD;
      when BAD =>
        NEXT_STATE <= BAD;
    end case;
  end process;

  process(CLOCK, RESET)
  begin
    if RESET = '0' then
      CURRENT_STATE <= S0;
    else
      if CLOCK'event and CLOCK = '1' then
        CURRENT_STATE <= NEXT_STATE;
      end if;
    end if;
  end process;
end FSM;
```

```

package proto is
type state is (idle, w_conn, connected, w_send, blocked,
sending,send2,send3, w_disc);
type message_in is (send_req,cc,data_req,token_g,resume,bloc,ackn,dis_req);
type message_out is (cr1,send_conf,dt,tok_rel, monit_cpl,dis_req,dis_ind);
end proto;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.proto.all;

entity ET is
    port (command: in message_in; clk: in bit; reset: in bit;
        m_out: out message_out ; n,b: in signed(7 downto 0); n_out: out
signed(7 downto 0));
end ET;
architecture Rec of ET is
    signal currentS, nextS: state;
    attribute state_vector: string;
    attribute state_vector of Rec: architecture is "currentS";
begin
    COMB: process( currentS, command)
        variable bound, nb_seg,nb,cnt: signed(7 downto 0);
        begin
            case currentS is
                when idle    => if command = send_req then m_out <= cr1; nextS <=
w_conn;
                    else nextS <= currentS; end if;

                when w_conn  => if command = cc then m_out <= send_conf; nextS <=
connected;
                    elsif (command = dis_req) then nextS <= idle;
m_out <=dis_ind;
                    else nextS <= currentS; end if;

                when connected => if command = data_req then m_out <= send_conf;
                    nb_seg := n; bound:= b; nb:=conv_signed(0,8);
cnt:=conv_signed(0,8);
                    nextS <= w_send;
                    else nextS <= currentS; end if;

                when w_send  => if command = token_g then m_out <= dt; n_out <= nb;
                    nb := nb+1; nextS <= sending;
                    else nextS <=currentS;
            end case;
        end process;
    end architecture;

```

```

        end if;
when sending => if (command = ackn) and (nb <nb_seg) then
    m_out<= dt; n_out <= nb;
    nb := nb+1; nextS <= sending;
elseif(command = ackn) and (nb = nb_seg) then
    m_out <= monit_cpl; n_out <= cnt; nextS <= send2;
elseif(command = bloc) then
    cnt:= cnt+1; nextS <= blocked;
else nextS <= currentS; end if;

when send2 => m_out <= tok_rel; nextS <= send3;
when send3 => m_out <= dis_req; nextS <= w_disc;

when blocked => if (command = resume) and (cnt <= bound) then nextS
<= sending;
    elseif cnt > bound then
        m_out <= monit_cpl; n_out <= cnt; nextS <= send2;

        else nextS <= w_send; end if;
when w_disc => if (command = dis_req) then m_out <=dis_ind; nextS
<= idle;
    else nextS <= currentS; end if;
when others => nextS <= idle;
end case;
end process; -- Outputs not registered

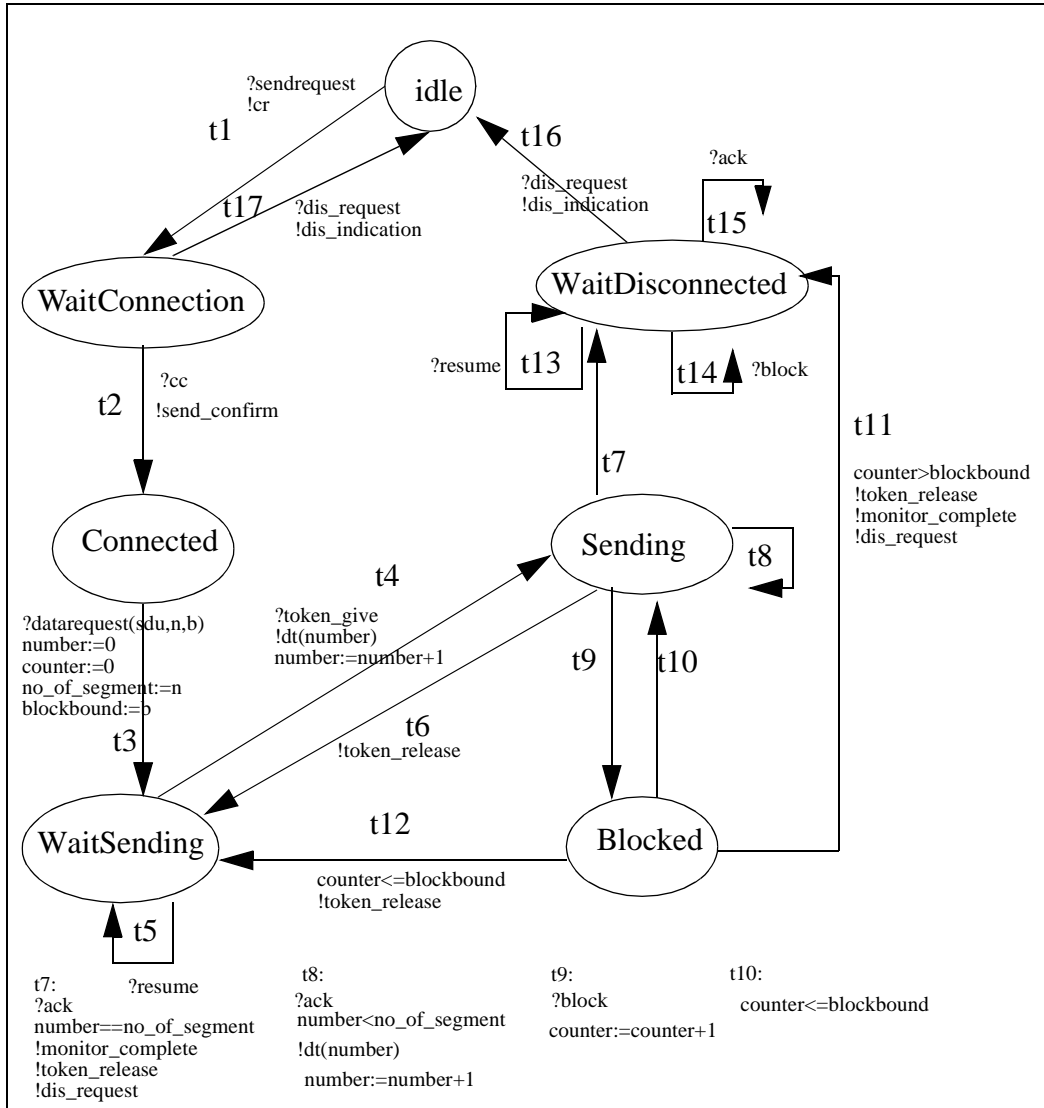
```

```

SYNC: process(clk, reset)
begin
    if reset = '0' then
        currentS <= idle; -- Async reset
    else
        if clk'event and clk = '1' then
            currentS <= nextS;
        end if;
    end if;
end process;
end Rec;

```

Protocol case study



```

package types is
  type state is (idle, w_conn, connected, w_send, blocked, sending, send2, send3, w_disc);
  type message_in is (send_req, cc, data_req, token_g, resume, bloc, ackn, dis_req);
  type message_out is (no_cr1, send_conf, dt, tok_rel, monit_cpl, dis_req, dis_ind);
  type manip is (none, assign, inc_nb, inc_cnt);
  type return_value is (nothing, ret_cnt, ret_nb);
end types;

```

```

-----
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.types.all;

```

```

entity TOP is
  port (clk, rst: in bit; command: in message_in;
        n, b: in signed(7 downto 0);
        m_out: out message_out; n_out: out signed(7 downto 0));
end TOP;

```

```

architecture schematic of TOP is
  signal nb_eq_nb_seg, nb_less_nb_seg, cnt_le_bound: boolean;
  signal order : manip;
  signal ret_order: return_value;

```

```

component fsmct
  port (clk: in bit; command: in message_in;
        nb_eq_nb_seg, nb_less_nb_seg, cnt_le_bound: in boolean;
        order : out manip; ret_order: out return_value;
        m_out: out message_out);
end component;

```

```

component Data_path
  port (clk, rst: in bit; order: in manip; ret_order: in return_value;
        n, b: in signed(7 downto 0);
        nb_eq_nb_seg, nb_less_nb_seg, cnt_le_bound: out boolean;
        n_out: out signed(7 downto 0));
end component;

```

```

begin
  fsm_c: fsmct
    port map (clk, command,
             nb_eq_nb_seg, nb_less_nb_seg, cnt_le_bound,

```

```

        order, ret_order,
        m_out);
data_c:Data_path
    port map (clk,rst, order, ret_order,
             n,b,
             nb_eq_nb_seg, nb_less_nb_seg, cnt_le_bound,
             n_out);
end schematic;

```

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.types.all;

```

```

entity fsmct is
    port (clk: in bit; command: in message_in;
          nb_eq_nb_seg, nb_less_nb_seg, cnt_le_bound: in boolean;
          order : out manip; ret_order: out return_value;
          m_out: out message_out);
end fsmct;
architecture Rec of fsmct is
    signal currentS, nextS: state;
    attribute state_vector: string;
    attribute state_vector of Rec: architecture is "currentS";
begin
    COMB: process( currentS, command, nb_eq_nb_seg, nb_less_nb_seg, cnt_le_bound)
        variable bound, nb_seg,nb,cnt: signed(7 downto 0);
        begin
            -- preventing latches
            order <= none; ret_order <= nothing; m_out <= no_sign;

            case currentS is
                when idle =>
                    if command = send_req then m_out <= cr1; nextS <= w_conn;
                    else nextS <= currentS; end if;

                when w_conn =>
                    if command = cc then m_out <= send_conf; nextS <= connected;
                    elsif (command = dis_req) then nextS <= idle; m_out <=dis_ind;
                    else nextS <= currentS; end if;

                when connected =>
                    if command = data_req then m_out <= send_conf;
                    order <= assign;
            end case;
        end process;
end architecture Rec;

```

```

        nextS <= w_send;
    else nextS <= currentS; end if;

when w_send => if command = token_g then m_out <= dt; ret_order <= ret_nb;
    order <= inc_nb; nextS <= sending;
    else nextS <=currentS;
    end if;
when sending => if (command = ackn) and nb_less_nb_seg then
    m_out <= dt; ret_order <= ret_nb;
    order <= inc_nb; nextS <= sending;
elseif(command = ackn) and nb_eq_nb_seg then
    m_out <= monit_cpl; ret_order <= ret_cnt; nextS <= send2;
elseif(command = bloc) then
    order <= inc_cnt; nextS <= blocked;
else nextS <= currentS; end if;

when send2 => m_out <= tok_rel; nextS <= send3;
when send3 => m_out <= dis_req; nextS <= w_disc;

when blocked => if (command = resume) and cnt_le_bound then nextS <= sending;
    elseif (not cnt_le_bound) then
        m_out <= monit_cpl; ret_order <= ret_cnt; nextS <= send2;
    else nextS <= w_send; end if;
when w_disc => if (command = dis_req) then m_out <=dis_ind; nextS <= idle;
    else nextS <= currentS; end if;
when others => nextS <= idle;
end case;
end process;

SYNC: process
begin
    wait until clk'event and clk = '1';
    currentS <= nextS;
end process;
end Rec;

```

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.types.all;

```

```

entity Data_path is
    port (clk,rst: in bit; order: in manip; ret_order: in return_value;
        n,b: in signed(7 downto 0));

```

```

        nb_eq_nb_seg, nb_less_nb_seg, cnt_le_bound: out boolean;
        n_out: out signed(7 downto 0));
end Data_path;

```

```

architecture high of Data_path is
    signal bound, nb_seg,nb,cnt: signed(7 downto 0);

```

```

begin

```

```

    nb_eq_nb_seg <= nb = nb_seg;
    nb_less_nb_seg <= nb < nb_seg;
    cnt_le_bound <= (cnt <= bound);

```

```

    process(ret_order,cnt,nb)
    begin
        if ret_order = ret_cnt then n_out <= cnt;
        elsif ret_order = ret_nb then n_out <= nb;
        else n_out <= (others => '0');
        end if;
    end process;

```

```

    process

```

```

    begin
        wait until clk'event and clk = '1';
        if rst = '1' then
            bound<=(others=>'0');
            nb_seg<=(others=>'0');
            nb<= (others=>'0');
            cnt <= (others=>'0');
        end if;
        if order = assign then
            nb_seg <= n; bound<= b; nb<=conv_signed(0,8);
            cnt<=conv_signed(0,8);
        elsif order = inc_nb then nb <= nb+1;
        elsif order = inc_cnt then cnt<= cnt+1;
        end if;
    end process;
end high;

```


LAB 6

Objective:

In this lab we will see how to compile a technology file to do gate level simulation with that technology.

The design used is the coder/decoder of lab4.

1. Take a look at the file test.vhd, it describes the simulation.

It feeds the input of a coder with values from the file dec.tv, the output of the coder is sent to a decoder.

We will start with a simple simulation of the source VHDL files.

2. In your shell window analyze the vhd files:

```
vhdlan coder.vhd
vhdlan decoder.vhd
vhdlan test.vhd
```

3. You can then start the simulation in text mode:

```
vhdsim CFG_TB
```

4. Then you just have to type "run" to start executing the test.vhd file.

The format used by test.vhd for the output is: Input => Coded => Decoded

Does the circuit do what was specified? _____

(Look at the description of the coding and decoding at the top of the .vhd files)

Type "quit" to return to the shell.

Now we are going to simulate it at gate level with the class technology.

5. First we can extract simulation components from the synthesis library:

```
liban -arch FTGS $SYNOPTSYS/libraries/syn/class.db
```

This generates two files: class_components.vhd and class_FTGS.vhd.E (encrypted vhd).

Analyze these two files with vhdlan (the FTGS file will take time to analyze and generates about 6MB of data).

6. Now compile the coder.vhd and decoder.vhd using Design Analyzer and save the result in

VHDL format in the files coder_gl.vhd and decoder_gl.vhd.

7. Edit the files to give different names to the entities (CODER_S and DECODER_S).
8. Analyze these two files
9. Modify the test.vhd file to have the two circuits run in parallel (before and after synthesis) and compare the results.

(a solution is in solutions/test2.vhd)

Analyze the file and start the simulator.

Use "run > mesg" to start the simulation.

Do the two circuits give the same results? _____

10. Look at the file mesg that contains all the warnings that occurred in the simulation. As you can see there are glitches but the output really is correct after 20 ns.

LAB 6 VHDL CODE

```
-- VHDL Test Bench

use STD.textio.all;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_textio.all;

entity TB is
end TB;

Architecture TESTBENCH of TB is
    signal CIN : STD_LOGIC_VECTOR(11 downto 0);
    signal DIN : STD_LOGIC_VECTOR(7 downto 0);
    signal DOUT : STD_LOGIC_VECTOR(11 downto 0);

    component DECODER
        port(DATA: in STD_LOGIC_VECTOR(7 downto 0);
             DOUT: out STD_LOGIC_VECTOR(11 downto 0));
    end component;
    component CODER
        port(DATA: in STD_LOGIC_VECTOR(11 downto 0);
             DOUT: out STD_LOGIC_VECTOR(7 downto 0));
    end component;

begin
    DEC : DECODER
        Port Map ( DIN, DOUT );

    COD : CODER
        Port Map ( CIN, DIN );
    process

    file TV : TEXT is in "dec.tv";
    variable L : LINE;
    variable ARR : STRING (1 to 4) := " => ";
    variable C_IN : STD_LOGIC_VECTOR (11 downto 0);

begin

--Get a vector
    readline (TV,L);
    read (L,C_IN);
```

```

--Assign input values
  CIN <= C_IN;

--Wait
  wait for 20 ns;

--Check output values
  write(L,CIN);
  write(L,ARR);
  write(L,DIN);
  write(L,ARR);
  write(L,DOUT);
  writeline(OUTPUT,L);

  if endfile(TV) then wait; end if;
end process;

end TESTBENCH;

configuration CFG_TB of TB is
  for TESTBENCH
    for DEC: DECODER
      use entity work.DECODER(LOGIC);
    end for;
    for COD: CODER
      use entity work.CODER(LOGIC);
    end for;
  end for;
end CFG_TB;

```