EMB3
Advanced Programmable Electronics

Lecture 2

Thursday, February

Jørgen Christian Larsen

jcla@mmmi.sdu.dk

Lecture Overview

- Why use simulation?
- VHDL for synthesis / simulation
- Simulation
- Testbenches
- Quick VHDL recap
- ISim demonstration
- FSM
- Exercise
- Project
  - VGA Generator module
  - Groups
  - Blokdiagrams
- Next time

Why use simulation?

- No FPGA hardware is necessary
- Better access to signals
- Can simulate all situations
- Compilation for synthesis is fast

VHDL for synthesis / simulation

- **V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage
- Created by the US Department of Defense in the 1980s
- Intended as a description language for existing circuits
- Only a part of VHDL is synthesizable, much is simulation-only
  - VHDL is synthesized in constructs / templates
    - **Demonstration (Xilinx Templates)**

Simulators

- Several simulators exists
- For Xilinx FPGAs the two main contenders are:

- ISim
  - Highly integrated in the toolchain
  - Only for Xilinx FPGAs

- ModelSim
  - Standalone simulator
  - Supports multiple FPGA vendors

ISim

- Highly integrated in the toolchain
- Only for Xilinx FPGAs

- 2 basic simulation modes
  - Functional simulation
    - Behavioral (RTL simulation) [fast]
    - Post Translate (Post-NGDBuild) [medium]
  - Timing simulation
    - Post Map (Partial Timing / Block Delays) [slow]
    - Post Route (Post Place and Route) [very slow]

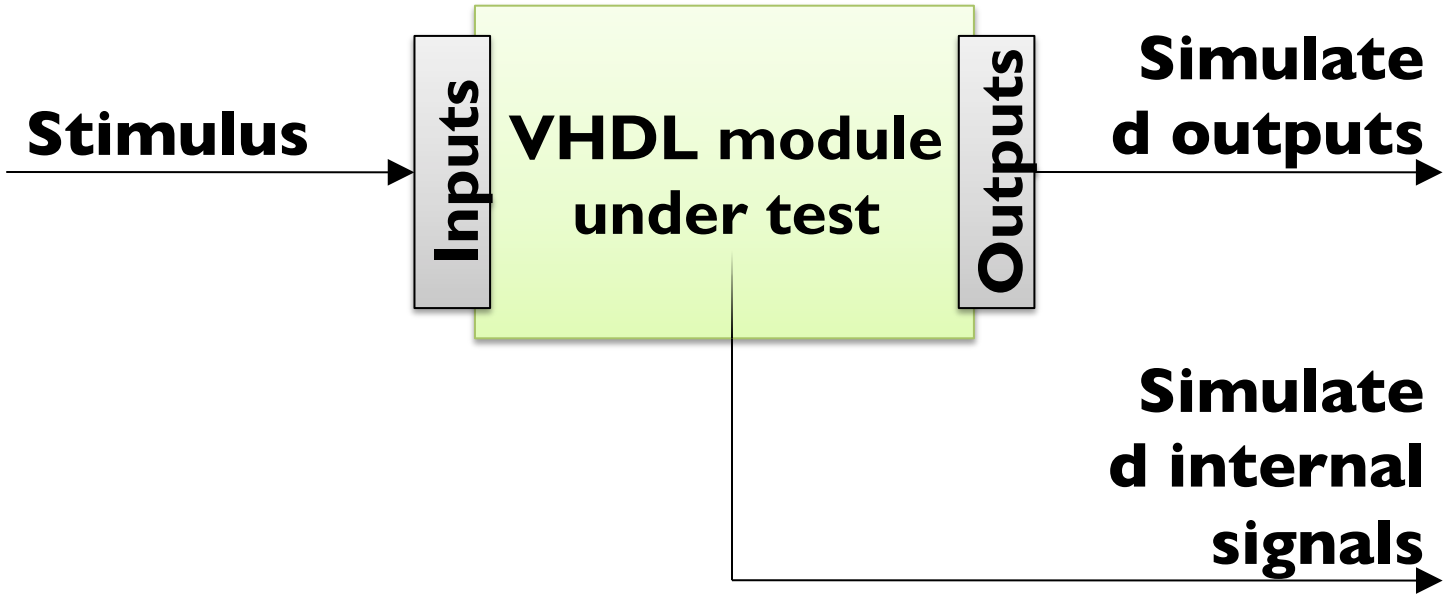
Creating stimulus

- **Manually inside the simulator**
  - Set the current value of specific signals
  - Either using the GUI or command line

- **Scripted**
  - Scripted version of the manual way
  - Good for automated testing

- Testbench

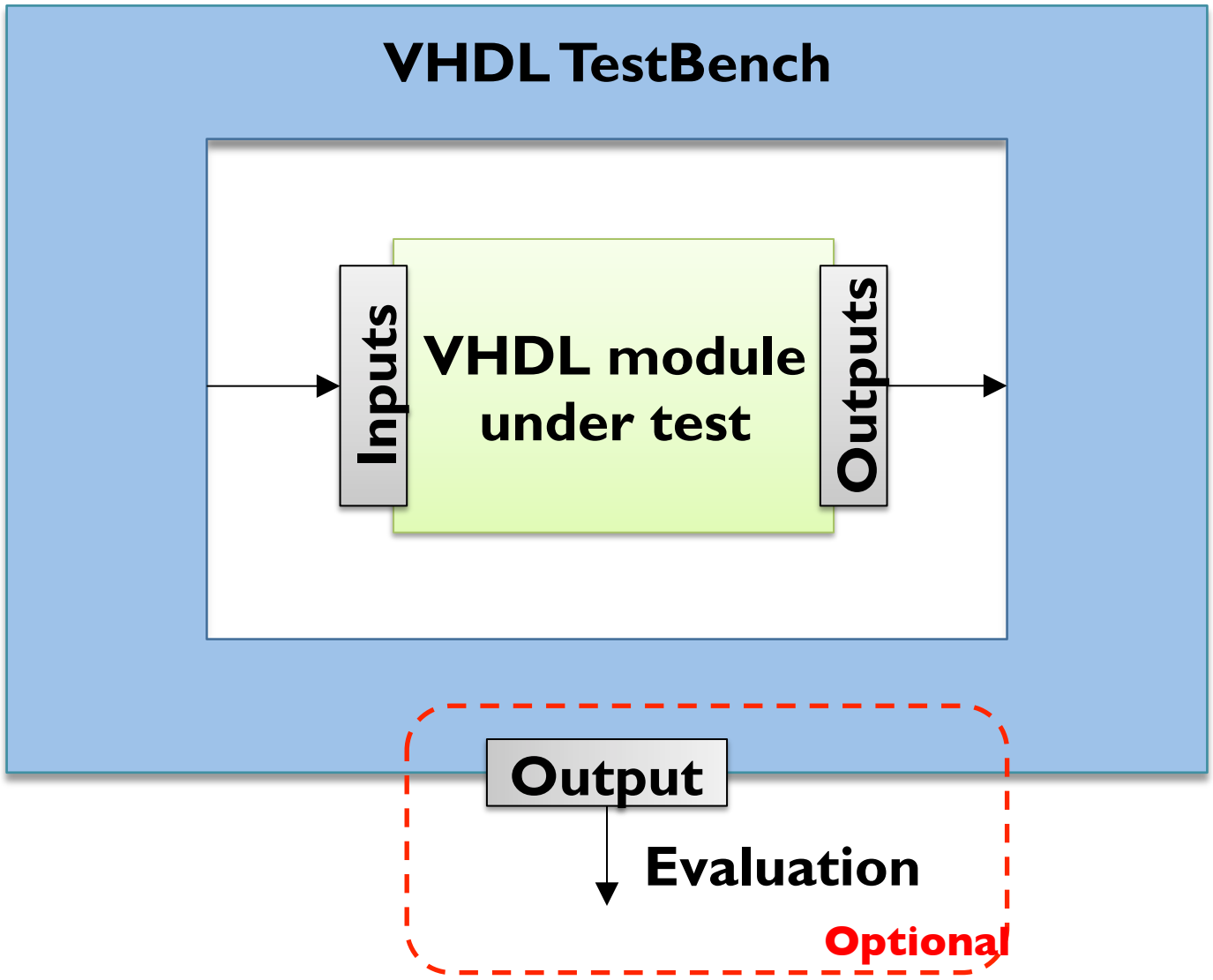
Simulation


Stimulus
Inputs
VHDL module under test
Outputs
Simulate d outputs
Simulate d internal signals

Creating stimulus

- Manually inside the simulator
- Scripted
- **Testbench**
  - Basically an extra VHDL module
  - May use all VHDL constructs (including simulation-only constructs)
  - Allows advanced functionality/stimulus, that reacts to the VHDL module under test.
  - Simulator independent

Testbench


VHDL TestBench
Inputs
VHDL module under test
Outputs
Output
Evaluation
Optional

Best Practices

- Create testbenches for all modules
- Create testbenches based on requirement specification
- Cover all special cases
- Run tests whenever a module is modified
- Good investment of time, especially in medium to large projects

Hardware Co-Simulation

- Complementary to the tool-based HDL simulation
- Enables simulation of a design or portion of a design
- Offloads the computational work of running a simulation to the FPGA hardware
  - Accelerates simulation
  - In-Hardware functional verification
  - Bit-and-cycle accurate

  - Requires a Board Support File
    - Non Xilinx boards need a custom file (Easy to make though)

Further Reading

- Xilinx ISim manual:
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_4/plugin_ism.pdf

- Xilinx ISim Tutorial:
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/ug682.pdf

- Xilinx ISim Hardware Co-Simulation Tutorial:
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/ug817_fft_sim_tutorial.pdf

Quick VHDL recap

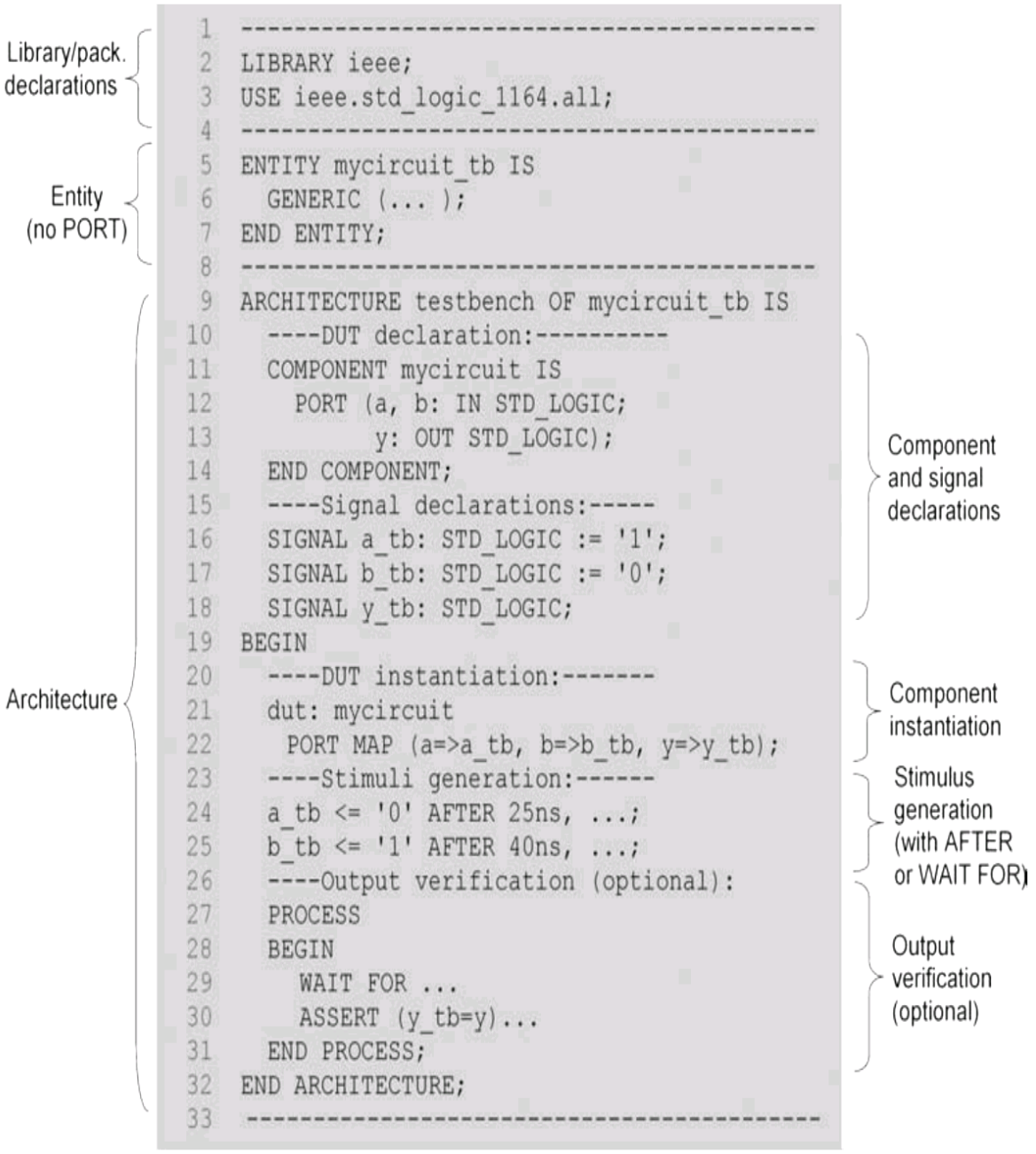
- Hierarchical design
  - VHDL modules as "black" box components
- Old-school method
  - Necessarry when building libraries with cross dependencies

Declare first

```
COMPONENT encoder is
PORT(
     clk_i   : IN  std_logic;
     reset_i : IN  std_logic;
     enc_a_i : IN  std_logic;
     enc_b_i : IN  std_logic;
     count_o : OUT SIGNED(7 downto 0)
    );
END COMPONENT;
```

- then instantiate

```
<instance_name>: encoder
PORT MAP(
       clk_i => clk,
       reset_i => reset,
       enc_a_i => enc_a,
       enc_b_i => enc_b,
       count_o => count
       );
```




```
1  ----------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------
5  ENTITY mycircuit_tb IS
6    GENERIC (... );
7  END ENTITY;
8  ----------------------------------------
9  ARCHITECTURE testbench OF mycircuit_tb IS
10   ----DUT declaration:----------
11   COMPONENT mycircuit IS
12     PORT (a, b: IN STD_LOGIC;
13           y: OUT STD_LOGIC);
14   END COMPONENT;
15   ----Signal declarations:-----
16   SIGNAL a_tb: STD_LOGIC := '1';
17   SIGNAL b_tb: STD_LOGIC := '0';
18   SIGNAL y_tb: STD_LOGIC;
19 BEGIN
20   ----DUT instantiation:-------
21   dut: mycircuit
22     PORT MAP (a=>a_tb, b=>b_tb, y=>y_tb);
23   ----Stimuli generation:------
24   a_tb <= '0' AFTER 25ns, ...;
25   b_tb <= '1' AFTER 40ns, ...;
26   ----Output verification (optional):
27   PROCESS
28   BEGIN
29     WAIT FOR ...
30     ASSERT (y_tb=y)...
31   END PROCESS;
32 END ARCHITECTURE;
33 ----------------------------------------
```

Figure 10.9
VHDL template for testbenches.

Quick VHDL recap

- Libraries
  - Use:
    - std_logic_1164 (standard)
    - numeric_std (contains SIGNED, UNSIGNED, etc.)
    - Others
  - Do **NOT** use:
    - std_logic_arith (implementations vary!)
    - std_logic_signed
    - std_logic_unsigned

    } (may be used for backwards-compatibility if absolutely necessary)

VHDL for simulation

- Simulation-only example
  - wait:

```
process
begin
  CLK <= '1';
  wait for 10 ns;
  CLK <= '0';
  wait for 10 ns;
end process;
```

FSM

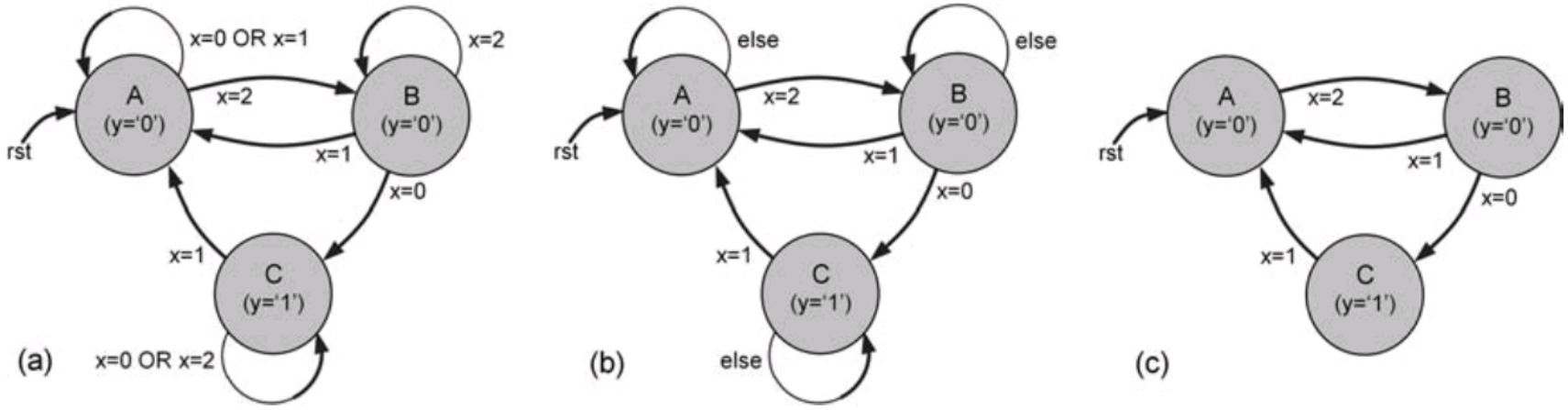


Figure 11.1
Three equivalent state transition diagrams: (a) Explicitly specified; (b) Using the "else" keyword; (c) With implicit "else" conditions.

FSM

```
1  ----------------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------------------------
5  ENTITY <entity_name> IS
6     PORT (clk, rst: IN STD_LOGIC;
7           input: IN <data_type>;
8           output: OUT <data_type>);
9  END <entity_name>;
10 ----------------------------------------------------------
11 ARCHITECTURE <architecture_name> OF <entity_name> IS
12    TYPE state IS (A, B, C, ...);
13    SIGNAL pr_state, nx_state: state;
14    ATTRIBUTE ENUM_ENCODING: STRING; --optional attribute
15    ATTRIBUTE ENUM_ENCODING OF state: TYPE IS "sequential";
16 BEGIN
```

#1 : Page 279-280

FSM

```
17    ------Lower section of FSM:------------
18    PROCESS (clk, rst)
19    BEGIN
20       IF (rst='1') THEN
21          pr_state <= A;
22       ELSIF (clk'EVENT AND clk='1') THEN
23          pr_state <= nx_state;
24       END IF;
25    END PROCESS;
```

#1 : Page 279-280

```
26    ------Upper section of FSM:------------
27    PROCESS (pr_state, input)
28    BEGIN
29       CASE pr_state IS
30          WHEN A =>
31             output <= <value>;
32             IF (input=<value>) THEN
33                nx_state <= B;
34             ...
35             ELSE
36                nx_state <= A;
37             END IF;
38          WHEN B =>
39             output <= <value>;
40             IF (input=<value>) THEN
41                nx_state <= C;
42             ...
43             ELSE
44                nx_state <= B;
45             END IF;
```

#1 : Page 279-280

```
42              ...
43              ELSE
44                 nx_state <= B;
45              END IF;
46           WHEN ...
47        END CASE;
48     END PROCESS;
49     ------Output section (optional):-------
50     PROCESS (clk, rst)
51     BEGIN
52        IF (rst='1') THEN
53           new_output <= <value>;
54        ELSIF (clk'EVENT AND clk='1') THEN --or clk='0'
55           new_output <= output;
56        END IF;
57     END PROCESS;
58  END <architecture_name>;
59  ------------------------------------------------------
```

#1 : Page 279-280

Exercise

- Build a FSM that switches the diodes on the board in a sequential order.
- It should have a button for switching to the next state
- It should have a button for reset.

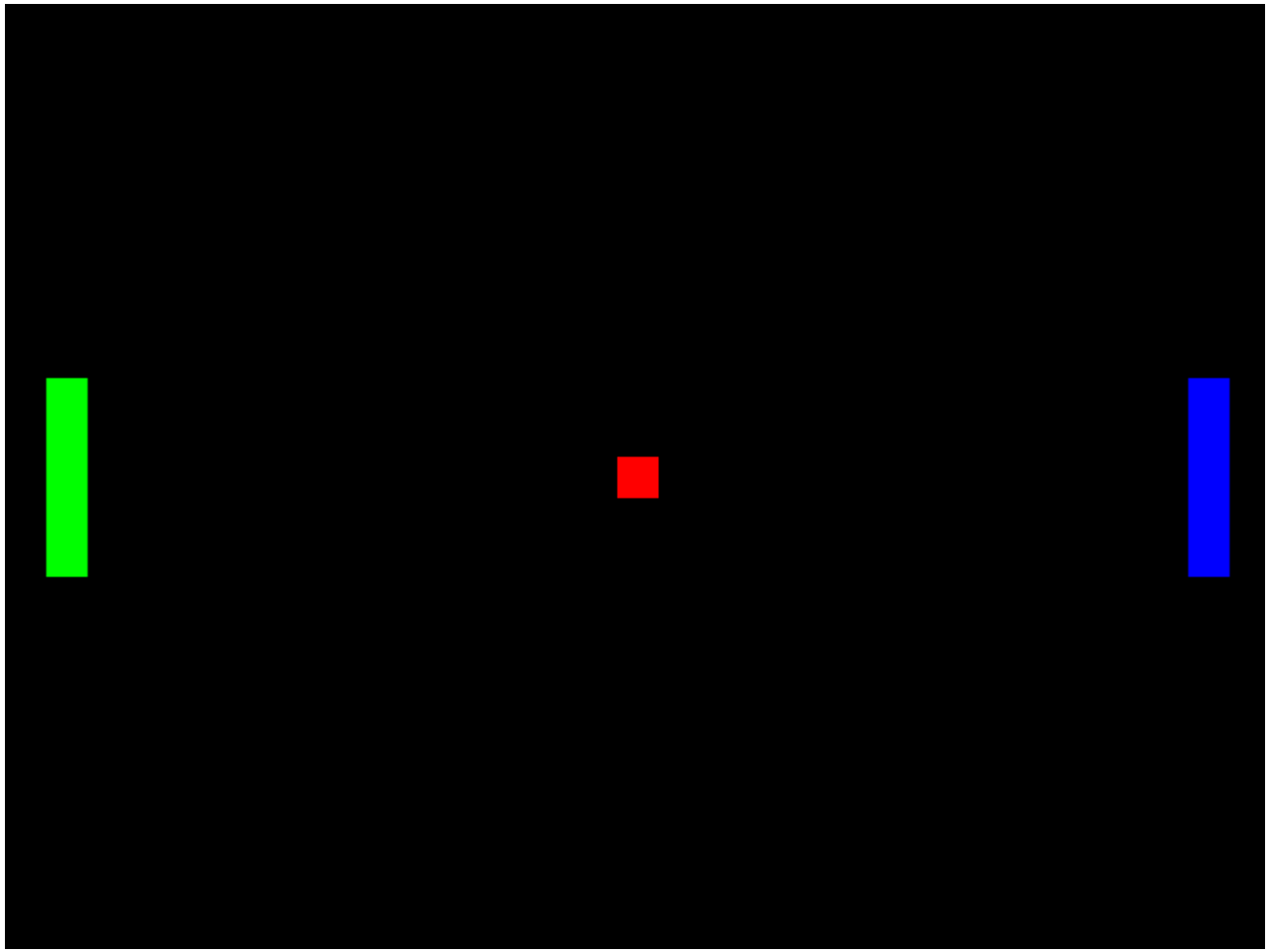
Project

- Groups
- Blokdiagrams
- VGA generator

VGA generator

- VGA Generator module to instantiate in a testbench to use for your project.
- Can NOT be synthesized
- Generates a single, static image (The pong game start-screen)
  - R, G and B signals
  - H- and V-sync signals
  - N.B. the generated image is in color, the image in the game is only in black and white.

VGA generator



Tasks for next time

- Read ALL the mandatory material:
  - See ”Reading Materiale” on BlackBoard
- Work on / Update your blockdiagram
- Play with the VGA-Generator module
  - Create a VHDL module that can locate the ball and the left+right bat’s
  - Create a testbench
  - Instantiate your Object-locator module and the VGA generator module and test if your code works!