VHDL
Simulation vs. Synthesis

EL 310
Erkay Savaş
Sabancı University
Two Models for VHDL Programs

- Two models
- Simulation
  - Describe the behavior of the circuit in terms of input signals, the output signals, knowledge of delays
  - Behavior described in terms of occurrences of events and waveforms on signals
- Synthesis
  - Reverse process - inference of hardware from description
  - The synthesis tool will infer a hardware architecture from the VHDL model
  - When writing a VHDL program, think of the hardware that synthesis tool would infer from it
Simulation and synthesis are complementary processes
Recall, in digital systems
- Events on signals take place at discrete points in time
- Some events cause other events after some delay
- Many events may be generated concurrently
Discrete Event Simulation

- A programming-based methodology for accurately modeling the generation of events in physical systems.
  - A program that specifies how and when events on specific signals are generated
  - Discrete event simulator then executes this program, modeling the passage of time and the occurrence of events at various points in time
  - Simulator often manages millions of events and keep accurate track of correct order of events
  - The way the simulator manages events and their orders, that is called discrete event simulation model, must be very well understood to write, understand and debug VHDL codes
Discrete Event Simulation Model

- Discrete event simulation utilizes a sorted event list structure.
  - Each event is described by the type of event (e.g. 1→0, 0→1 transitions) and the time it is to occur.
  - Transition could be in various forms (Z→1, 0→X)
  - Timestamp of an event is the time at which this event happens.
  - Event list is ordered according to increasing timestamp value.
  - Simulator clock records the passage of simulated time.
  - The value of the clock is called current timestep.
  - A snapshot of the values of all signals at a timestep is called the state of the simulation.
Discrete Event Simulation: Example 1

- Half Adder

\[ a \rightarrow \text{sum} \rightarrow \text{carry} \]

\[ b \rightarrow \text{sum} \rightarrow \text{carry} \]

\[ \text{event} \]

\[ a, b, \text{sum}, \text{carry} \]

\[ 5 \text{ ns} 10 15 20 25 30 \]
Discrete Event Simulation: Example 2

- Discrete event simulation of half adder

For example, \( \text{sum} \) is scheduled to take value of 0 at simulation time 5 ns. Prior to this time, \( \text{sum} \) and \( \text{carry} \) are uninitialized.

- Simulator removes these events from the list in the specified order and updates the values of signals accordingly.

- The global clock is first updated with the current time, then the event scheduled to this time are executed and then removed.
Steps of the Simulation Model

1. Advance the simulation time to that of the event with the smallest timestamp in the event list. This is the event at the head of the sorted event list.
2. Execute all events at this timestep by updating signal values
3. All components affected by the new signal values must be re-evaluated (new events that may be generated as a result of changes in signal values must be scheduled by placing them in event list in the order of their timestamp)
4. Schedule future events
5. Repeat until the event list is empty or preset simulation time has expired.
   - The simulation model is very general such that it is applicable to different levels of abstraction.
Logical Model vs. Physical Model

- Discrete event simulation model is just an approximation of behavior of real systems.
  - In fact, there are only analog devices where we interpret analog voltage levels as logical 0 or 1.
  - We refer discrete event simulation model as logical model while the real device as physical system
  - For better accuracy with which to capture the behavior of physical system some complimentary techniques or models must be employed
  - VHDL simulators provide facilities for setting the duration of a simulation timestep and query the contents of the event queue during simulation
  - User can examine the event at the head of the event queue and force signals to specific values prior to next timestep.
Accuracy vs. Speed

- Think of a 32-bit adder
- **Low accuracy:**
  - In VHDL, a 32-bit adder may be described in one line: two inputs and a single output. Inputs generate a value that appears at the output after some propagation delay.
  - One step corresponds to the delay of adder
  - Easy, fast, and not too accurate
- **High accuracy:**
  - We might describe the adder at gate level
  - More realistic simulation
  - Has to handle many events
  - One step in the simulation corresponds to one gate delay.
  - Naturally slow
Synthesis Model

• Synthesis is a process
  - Where a physical system is constructed from an abstract description using a pre-defined set of basic building blocks (e.g. logic gates, flip-flops, latches, small blocks of memory, LUTs in FPGA)

• Synthesis compilers operate on three types of information
  - Model of the circuit (e.g. VHDL model)
  - Set of constraints (speed, area, etc)
  - Set of components (2-input NAND gates, flip-flops with asynchronous set and reset, etc.)
Design Process 1

- We use
  - Boolean expressions, truth tables, state diagrams to design a digital system.
  - With synthesis tools, we can design digital systems at higher level of abstractions such as addition or multiplication of integers or real numbers, masking and shifting operations on Boolean strings, etc.
  - The higher the level of abstraction you design at the less control over what is being synthesized by the compiler.
  - High-level synthesis problem is more challenging for the compiler since it has to choose between alternatives.
  - The one chosen by the compiler may not be optimal.
Design Process 2

• However,
  - working at higher level of abstraction we will design the hardware in less time (shorter design cycle)
  - It also facilitate design re-use and react to the future changes in the design
  - For example, think of 32-bit adder
  - By specifying the design constraints the compiler may choose between carry ripple adder (which is area optimized) and carry lookahead adder (which is time optimized)
Hardware Inference

• Just as simulation process is based on discrete event simulation, synthesis process from VHDL model is based on the process of inference.
  - Synthesis compilers must infer typical hardware components and their interconnection from the VHDL code.
  - Inference is followed by optimization to reduce the size or increase the speed of the inferred circuit.
  - If you follow certain rules while writing VHDL programs, you will have significant control over the inferred hardware.
Hardware Inference Example

Inferring Combinational Logic

```plaintext
--
-- pseudo code for a single-bit arithmetic/logic unit
--
s1 <= in1 and in2;
s2 <= in1 or in2;
s3 <= in1 xor in2 xor c_in; -- perform the sum operation
c_out <= (in1 and c_in) or (in1 and in2) or (in1 and c_in);
z <= s1 when sel = "00" else
   s2 when sel = "01" else
   s3 when sel = "10" else
   '0';
```
Inferring Combinational Logic

- Single-bit ALU

\[ \text{c_in} \]

\[ \text{in1} \]

\[ \text{in2} \]

\[ \text{sel(1)} \]

\[ \text{sel(0)} \]

\[ (\text{in1} \cdot \text{in2}) \cdot \text{sel(1)}' \cdot \text{sel(0)}' \]

\[ (\text{in1} \oplus \text{in2} \oplus \text{c_in}) \cdot \text{sel(1)} \cdot \text{sel(0)}' \]

\[ \text{c_out} \]

\[ z \]
Inferring Sequential Logic

```vhdl
--
-- a simple conditional code block
--
if (sel = '0') then
    z <= in1 nor in2;
end if;
```

- Inference of sequential components is more subtle.
- Whether a latch of flip-flop is inferred depends on the way we write your VHDL codes.
- By careful coding, we can avoid unnecessary sequential components (stay away from conventional programming techniques; stick to “hardware aware” approach)
Target Primitives 1

- Target primitives are the building blocks synthesis compiler use.
  - Example: Boolean equation, $z = a'bc + b'c + db$

Simple fact: Circuit produced by the synthesis compiler depends on the set of available building blocks
Target Primitives 2

- Implementation given the description is not necessarily unique.
- The way the synthesis compiler choose building blocks is also dependent on selected performance metrics such as area and speed.
- Fortunately, CAD tools offers optimization facilities given performance metrics for target devices.
- If you know the target primitives, you can anticipate the logic that the compiler infer for a given language construct.
Field Programmable Gate Arrays (FPGA)

• Reprogrammable hardware devices
  - (Re)configurable hardware, reprogrammable hardware, etc.
  - We are interested in one kind of such reprogrammable hardware: FPGA.

• Why Reprogrammable hardware?
  - ASIC offers the highest performance
  - But performs only one function
  - Development cost cannot be spread across multiple applications
  - **Ideal solution**: change the interconnection between millions of transistors comprising the chip in order to compute another function.
• Think of a cell (or block)
  - That can be programmed to implement any small combinational or sequential circuit
  - Tile the surface of a silicon die with these cells.
  - Wires running vertically and horizontally between these cells.

Any block can connect itself to horizontal or vertical wires that are adjacent to it.

Interconnect of the wires are called switching matrix.
Design with FPGA

• Two types of blocks
  - **CLB** - configurable logic blocks (inside the chip)
  - **IOB** - I/O blocks on the periphery (to drive the signals off the chip and read the incoming signals)

• Design process
  1. Partition the “big” design into small sub-circuits.
  2. Each sub-circuit can be implemented in a CLB
  3. Establish the connections between sub-circuits through the switching matrix
  4. If another design to be implemented, go to Step 1.
Inside the XC 4000 CLB

• Many FPGAs look similar
  - But they differ in how CLBs are implemented.
  - CLB is actually Xilinx terminology
• Main components in XC 4000 CLB
  - Two 16x1 RAM blocks
  - One 8x1 RAM block
    • RAM can be used as LUT to implement combinational logic through truth tables
  - Two edge-triggered D flip-flops
    • Can also be configured as latches
  - Several multiplexors to configure the interconnection within the CLB.
**Lookup Tables (LUT)**

- Combinational circuits are implemented using LUT

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Combinational functions in LUT

- Four inputs serve as the 4-bit address to the 16x1 memory.
- Single bit memory hold the output value corresponding to the input value that is used as an address to access to this memory cell.
- 16x1 LUT can implement any Boolean function of four variables.
- LUT is also referred as function generator.
- Each CLB has two 4-bit function generators: F and G.
- The third generator, H, can implement any Boolean function of three variables.
  - Two inputs come from the outputs of F and G.
  - The other input comes from outside.
LUT

- Using three function generators F, G, and H
  - A single CLB can implement any Boolean function of five variables and some functions of up to nine variables
  - It has nine logic inputs: \( F_1, F_2, F_3, F_4, G_1, G_2, G_3, G_4, \) and \( H_1. \)
  - General form of Boolean function of five variables:
    \[
    H = F(F_1, F_2, F_3, F_4) \cdot H_1' + G(F_1, F_2, F_3, F_4) \cdot H_1
    \]
- In some 4000 CLBs, LUTs can be configured as memory block
  - One 32x1 module
  - Dual ported 16x1 memory module
  - Synchronous-edge-triggered and asynchronous-memory interfaces are supported
Odds and Ends

- Two D flip-flops with enable clock (EC) input
  - Edge-triggered flip-flop or
  - Can also be configured as latches with a common clock and clock enable inputs
- Four additional inputs and a clock
  - \((C_1 \ldots C_4)\) and \(K\) (clock)
- CLB has four outputs
  - Two from the combinational function generators.
  - Two from the flip-flops
- Bypass signals
  - To facilitate placement and routing, there are bypass signals that pass inputs to outputs without changing it: \(XQ\) and \(YQ\)
Odds and Ends

• Many vendors typically supply implementation of common building blocks that are optimized to the structure of their hardware components.
  - Xilinx, in fact, has a core generator utility that can create common building blocks from parameterized descriptions provided by the user
  - Adders, subtractors, multipliers, memories, etc. are such building blocks

• FPGA as a sea of LUTs and flip-flops
  - A gate-level design can be placed on the array by mapping combinational components to function generators, sequential to flip-flops
Each cell contains carry logic for two bits.
Carry Logic 2

- F and G generators can be used to generate the sum bits.
- Thus, each cell can be programmed to implement both carry and sum for two bits of a full adder.
- The carry lines between cells are hardwired (not programmable) to provide for fast propagation of carries.
- The carries can propagate up or down between cells and programmable MUX (M1) selects the direction.
- MUXes M2, M3, and M4 allow some of F and G generator inputs to come from carries instead of normal inputs.
- The carry logic can be programmed to implement subtracter, incrementer/decrementers, 2’s complementers, and counters.
A CLB as Two-Bit Full Adder

G4: Ai+1
G1: Bi+1
F2: Ai
F1: Bi
Ci

G Function

Si+1
Ci+1
Ci+2

F Function

Si
Connections for 4-bit Adder

- If we want to detect a possible overflow, we add the 4th CLB.
- The 3rd CLB outputs $C_3$ instead of $C_4$ (How?)
- In the 4th CLB, $C_4$ can be re-computed using the carry logic.
- Overflow is computed using the $G$ function generator in the 4th CLB
  - Overflow: $V = C_3 \oplus C_4$
- 4-bit adders can easily be expanded 8 or 16-bit adders
- Adder modules are available in Xilinx library
F and G as RAM Memory: 16x2

$F_i = G_i$

$C_1$ is write enable (WE)

$C_2$ and $C_3$ are data inputs

Can it be configured as 32x1 memory?
• 4000 IOB
  - Implements I/O functionality in the chip
  - \( I_1 \) and \( I_2 \) signals carry input to the FPGA directly from the pin or from an IOB flip-flop that can be used to capture the input data.
  - Additional latch for fast capture of data
  - Output signals can pass directly to the pin or can be stored in a flip-flop
  - Output multiplexor enables two signals (output data and clock enable) to share a pin (higher number of effective outputs).
  - Separate clocks for input and output
  - can be programmed to invert input, output, tristate buffer, and clock input
Switch Matrix

- Different levels of interconnections
  - Direct interconnects between CLBs. Short wires to connect adjacent CLBs in both directions.
  - Switch matrix

• PSM: Programmable Switch Matrix
• PSM can be configured to connect a horizontal wire to a vertical one.
• One wire can be connected to multiple wires
• This way output of a CLB can routed through multiple PSMs to the input of another CLB.
Types of Lines

- **Single-length**: connects adjacent PSMs
- **Double-length**: connects every other PSM
- **Quad-length**: traverse four CLBs before passing through a PSM.
- **Long**: runs entire chip.
  - Using tri-state buffers within the CBLSs, long lines can be configured as buses.
- Local connections use direct interconnect or single length lines in order to avoid to many switching points
- **Global Nets**: Low skew signal paths used to distribute high fan-out signals such as clock and reset signals.
Double Length Lines

[Diagram showing a grid with CLB and PSM blocks arranged in a double-length line pattern]
Example Interconnects

Direct interconnects between adjacent CLBs

General-purpose interconnects
Tri-state Buffers

- **Bus**
  - Each CLB has tri-state buffers that connect to long lines.
  - Bus contention must be avoided
    \[ Z = D_A A' + D_B B' + D_C C' + ... + D_N N' \]

- **Wired-AND implementation**

\[ Z = D_A D_B D_C ... D_N \]
Configuration 1

- **Configuration data for CLBs**
  - Provide the contents of the LUTs (16+16+8 = 40 bits)
  - Configure multiplexors within the CLB
    - To determine the input sources of the flip-flops, and the inputs to the H function etc.
    - Bits for configuring flip-flops
- **Configuration data for PSM**
  - Connecting horizontal and vertical lines
  - Tristate devices within the CLBs
Configuration 3

• Configuration data for IOB
  - IO clocks
  - Storage vs. direct access to the pin.

• Size of the configuration file
  - Each of the configuration requires at least a bit of memory
  - Configuration data from several Kbits to several Mbits
  - Rule of Thumb: 20 configuration bits per available user gates

  - CAD tools translate VHDL code to bit stream to configure FPGA device
Configuration 4

- Output clock
- Input clock
- Clock enable
- Output
- Input

Components:
- Slew Rate control
- Passive Pull-Up Pull-Down
- MUX
- Output Buffer
- Input Buffer
- Fast Capture Latch
- Delay

Signals:
- TS
- I1
- I2
- CE
- D
- Q
- G

Diagram shows the connections and flow of signals through the configuration.
FPGA Design Flow 2

- **Model development:**
  - VHDL code
  - State-machines may be described in a graphical manner and translated into VHDL code.
  - Traditional schematic capture can be translated into VHDL source.

- **Behavioral Simulation**
  - Before synthesis; for testing functional correctness

- **Synthesis**
  - The design is synthesized to a library of primitive components such as gates, flip-flops, and latches

- **Functional Simulation**
  - To find out preliminary performance estimates
  - For example, timing information can be obtained from known properties of FPGA components
  - Still not too accurate
**FPGA Design Flow 3**

- **Place and Route:**
  - The design is mapped to the primitives in the target chip.
  - In FPGA, there are function generators, flip-flops, and latches.
  - Each primitive must be assigned to a specific CLB (Placement).
  - Connections between CLBs that implement the primitives must be established (routing).
  - Accurate timing can be obtained in Verification step (Post-placement and routing simulation).
  - The configuration bits are generated.
FPGA Design Flow 4

• **Programming:**
  - The configuration data (bit stream) is finally loaded into the target FPGA chip.

• These steps are fairly generic although the terminology used here is adopted from Xilinx.
Xilinx Foundation Tools: Design Flow
Efficient Design

• Generally, we try to reduce the number of flip-flops in the design
  - X4000 (like many other FPGAs) has two flip-flops in a CLB
  - Instead of trying to reduce the number of flip-flops, try to reduce the number of CLBs and interconnects between them

• One-Hot State Assignment
  - Use one flip-flop for each state (N state N flip-flops)
One-Hot State Assignment: Example

- Four states: $T_0$, $T_1$, $T_2$, and $T_3$.
- Four flip-flops: $Q_0$, $Q_1$, $Q_2$, and $Q_3$.
- Assignment:
- $T_0$: (1000), $T_1$: (0100), $T_2$: (0010), $T_3$: (0001).

\[
Q_3^+ = X_1Q_0Q_1'Q_2'Q_3' + X_2Q_0'Q_1Q_2'Q_3' + X_3Q_0'Q_1'Q_2Q_3' + X_4Q_0'Q_1'Q_2'Q_3
\]
\[
= X_1Q_0 + X_2Q_1 + X_3Q_2 + X_4Q_3
\]

\[
Z_1 = X_1Q_0 + X_3Q_2
\]
\[
Z_2 = X_2Q_1 + X_4Q_3
\]
One-Hot State Assignment: Example

- With One-Hot state assignment, the next state and output equations will contain one term for each arc leading into the corresponding state.
- Resetting the system requires one flip-flop be set to 1 instead of resetting all flip-flops to 0.
  - If flip-flops do not have a preset input (as in the case for the Xilinx 3000 series).
  - New assignment:

\[ T_0: (0000), T_1: (1100), T_2: (1010), T_3: (1001). \]

- \( Q_3^+ = X_1Q_0' + X_2Q_1 + X_3Q_2 + X_4Q_3 \)
- \( Z_1 = X_1Q_0' + X_3Q_2 \) and \( Z_2 = X_2Q_1 + X_4Q_3 \)
Typical FPGA

• **XC4003 FPGA**
  - Aprox. 3000 gates
  - 100 CLBs (10 x 10)
  - 80 user I/Os
  - 360 flip-flops (200 in CLBs and 160 in the IOB)
  - 45536 configuration bits

• **XC4010E/XL**
  - 7000 - 20000 Gates
  - 400 CLBs (20 x 20)
  - 160 User I/O
  - 1120 flip-flops
  - Several hundred bits of configuration data per CLB
Summary

- **Simulation Model**
  - Discrete event execution model for VHDL programs
  - Accuracy vs. time

- **Synthesis Model**
  - Need for inference from language constructs
  - Basic principles behind the use of FPGAs
  - Challenges in the design flow