Previous Lecture
The Evolution of FPGA (Xilinx)

Development Effort
Simplified FPGA Architecture

- **Functional Blocks**
  - Implement any logic function
    - \(F(a,b,c,d,e,f) = ab + cd + bde\)
    - \(F(a,b,c,d,e,f) = abcd + ef\)
    - \(F(a,b,c,d,e,f) = a + b + c + df\)
    - ....

- **Routing Network/Switchbox**
  - Connect blocks

- **I/O Blocks**
  - Communicate with the outside world

- **Everything is reconfigurable**
Xilinx Configurable Logic Block (CLB)

- Each CLB contains
  - 2 Slices
  - Carry-look-ahead unit
  - 8 Flip-flops
  - MUXes
- Each Slice contains 4 6-input LUTs
- Each LUT contains 2 5-input LUTs
Additional Resources

- **Block RAM**
  - Store large data

- **Carry-chain**
  - Faster Adder with carry-look-ahead

- **DSP (Digital Signal Processing)**
  - Multiplication, MAC

- **Processor**

---

Xilinx Zynq Routing Architecture Example
FPGA Design Flow

Architecture Independent

- Design the circuit
  - Design the circuit using VHDL or Verilog in Xilinx ISE or Vivado.
- Synthesis
  - Execute FPGA synthesys tools to convert the HDL codes into netlists.

Architecture Dependent

- Technology Mapping
  - Execute FPGA technology mapping tool to map the netlists into available FPGA resources.
- Placement and Routing
  - Execute FPGA Placement&Routing tools to place and connect the resources mapping done in the previous step onto the FPGA.
- Generate Bitstreams
  - The configuration data for the FPGA is generated in the form of bitstream.
- Reconfigure FPGA
  - Load the bitstream to the FPGA to run the circuit designed in the first step.
Partial Reconfiguration

- Reconfiguring part of the FPGA without affecting other components

- Benefits
  - Use smaller FPGA
  - If device is large enough, hide the reconfiguration latency by having cache-like mechanism to preload the accelerators to the available slots
“Homework”

- Why can one LUT implement $2^{2^n}$ functions? (n is the number of inputs)
- FPGA Vendors? What do they do?
- Which FPGA devices/families are being offered by Xilinx?
- How many ways are there to reconfigure Xilinx FPGA? What are the corresponding speeds?
- Can FPGA be used for IoT?
- Install Xilinx SDSoC (you may request for 30-day evaluation before a license server is setup for you to access)
Agenda

1. System Specification
2. Hardware Modeling and Description Language
3. Abstract Models
4. Compilation and Behavior Optimization
5. FPGA Design in Industry

Note: The first 4 topics are discussed quite extensively in Chapter 3 of the Ref Book.
Agenda

1. **System Specification**
2. **Hardware Modeling and Description Language**
3. **Abstract Models**
4. **Compilation and Behavior Optimization**
5. **FPGA Design in Industry**
Simplified and General Embedded System Design Methodology
System Specification

- Specification requirements
  - Functionality
  - Timing
  - Performance
  - External interface to other systems
  - Power consumption
  - Manufacturing cost
  - ...

![Diagram showing mapping and synthesis processes]
Types of Specification

- **Formal specification**
  - Based on mathematical modeling
  - A set of assertions about properties of a system
  - Formal method can be directly applied to prove that a specific implementation meets the requirements of the specification.

- **Informal specification**
  - Executable specification (simulation)
  - Focus on functional behavior of the system
  - Various stages are not logically connected.
  - May result in costly redesign
  - C, C++, SystemC, VHDL, etc.
Example

- Strict timing requirements
- Speed requirement is high (>= 200 MHz)
- Latency is money
Zynq7000 SoC Architecture
Computations

tmp0 = a + b;
tmp1 = c * d;
tmp2 = tmp0 * tmp1;
tmp3 = tmp0 + tmp2;
e = tmp2 * tmp3;
ZynqUltraScale SoC Architecture
Agenda

1. System Specification
2. Hardware Modeling and Description Language
3. Abstract Models
4. Compilation and Behavior Optimization
5. FPGA Design in Industry
Hardware Modeling

Y-Chart

Behavioral
- Algorithm (D.S.)
- Control/Data Flow (Word)
- FSM, Boolean Equation (Logic)
- Circuit (Transfer Function)

Structural
- Processor, Memory, Switch
- ALU, MUX, Register, Control
- Gates
- Transistors

Physical
- Cell Layout
- Module Floor Plan
- Processor Floor Plan
- System Floor Plan
Hardware Modeling
Abstraction Levels and Synthesis Flow with HDL

<table>
<thead>
<tr>
<th>language models</th>
<th>abstract models</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDL</td>
<td>operations &amp; dependencies (control/data-flow graph)</td>
</tr>
<tr>
<td>HDL</td>
<td>architectural synthesis/optimization</td>
</tr>
<tr>
<td>HDL</td>
<td>FSMS and logic functions (state tables &amp; logic networks)</td>
</tr>
<tr>
<td>HDL</td>
<td>logic synthesis/optimization</td>
</tr>
<tr>
<td>HDL</td>
<td>interconnected logic blocks (logic networks)</td>
</tr>
</tbody>
</table>

© Akash Kumar
Hardware Description Language Characteristics

- Hardware (vs. software)
  - Concurrency (parallel computer)
  - Structural information as well as behavioral information
  - Timing (real time software)
Hardware Description Language Simulation

- Event-driven
- Each simulation time (ns, ps) is decomposed into a large number of $\Delta$ time slots
- The dependencies (input, output, propagation delay, etc.) between components are resolved sequentially based on $\Delta$
- Simulation speed depends on the level of abstraction
Hardware Description Language
Different Languages

- VHDL
- Verilog
- SystemVerilog
- SystemC
- Bluespec
- Chisel
- Gem5
Agenda

1. System Specification
2. Hardware Modeling and Description Language
3. Abstract Models
4. Compilation and Behavior Optimization
5. FPGA Design in Industry
Abstract Model

- Structure
- Logic Network
- State Diagram
- CDFG – Control/Data Flow Graph
Abstract Model – Structure (1)

- Can be modeled in terms of incidence structures
  - modules and/or pins + nets + incidence relation
- Hypergraph
- Bipartite graph
Abstract Model – Structure (2)

- Incidence matrix
- Netlist: when the corresponding incidence matrix is sparse
  - module-oriented, net-oriented
- Hierarchy: leaf module is a primitive

\[
\begin{pmatrix}
  n1 & n2 & n3 \\
  m1 & 1 & 1 & 1 \\
  m2 & 1 & 1 & 0 \\
  m3 & 0 & 1 & 1 \\
\end{pmatrix}
\]

\[
\begin{align*}
  m1 &: n1, n2, n3 \\
  m2 &: n1, n2 \\
  m3 &: n2, n3 \\
\end{align*}
\]
Abstract Model – Logic Network

- **Combinational**
  - No feedback (partial order)
  - Boolean network:
    - multi-input/multi-output
    - vertex describes multi-input/single-output leaf module

- **Sequential**
  - Not necessarily partial order

\[
\begin{align*}
  d &= ab' + a'b \\
  e &= dc' + d'c \\
  f &= dc + ab
\end{align*}
\]
Abstract Model – State Diagram

- **State diagram**
  - A set of primary inputs $X$
  - A set of primary outputs $Y$
  - A set of states $S$
  - State transition function
    - $\delta : X \times S \rightarrow S$
  - Output function:
    - $\lambda : X \times S \rightarrow Y$ (Mealy)
    - $\lambda : S \rightarrow Y$ (Moore)
  - Initial state

https://en.wikipedia.org/wiki/State_diagram
Abstract Model – State Diagram

Architectural abstract level
Behavioral view

Architecture abstract level
Structural view

https://www.emoze.com/@ALILIRFQ

© Akash Kumar
Abstract Model – CDFG (1)

- Control/Data Flow Graph
- Operations + data dependency

\[
\begin{align*}
\text{xl} &= x + dx; \\
\text{ul} &= u - (3xu dx) - (3y dx); \\
\text{yl} &= y + (u dx); \\
\text{c} &= \text{xl} < a;
\end{align*}
\]
Abstract Model – CDFG (2)

- Control-flow graph
  - Conditional branching, iteration, model call

- Control/Data-Flow Graph (CDFG)
  - How to merge the two graphs
  - Simple approach: data-flow graph + branching vertices

- Sequencing graph
  - Hierarchical CDFG
  - Vertices: operations, links
  - Acyclic (partial order)
  - Polar: source and sink, model NOP (no operation)
  - Link vertex:
    - model call,
    - branching,
    - iteration
Abstract Model – CDFG – CALL

\[ x = a \times b; \]
\[ y = x \times c; \]
\[ z = a + b; \]
\[ \text{submodel}(a, z) \]

\[ \text{submodel}(m, n) \{ \]
\[ p = m + n; \]
\[ q = m \times n; \]
\[ \} \]
Abstract Model – CDFG - BRANCHING

\[
x = a \times b \\
y = x \times c \\
z = a + b \\
\text{if} \ (z \geq 0) \ \{ \\
\quad p = m + n; \\
\quad q = m \times n; \\
\}
\]
Abstract Model – CDFG - Iteration

diffeq {
  read (x, y, u, dx, a);
  repeat {
    xl = x + dx;
    ul = u - (3*x*u*dx) - (3*y*dx);
    yl = y + (u*dx);
    c  = xl < a;
    x = xl; u = ul; y = yl;
  } until (c);
  write (y);
}
- Data independent
- Data dependent
  - Bounded (min., max.): loop, conditional branch
  - Unbounded: external synchronization
- Latency: overall delay of a graph
  - Bounded-latency graph
  - Unbounded-latency graph
Agenda

1. System Specification
2. Hardware Modeling and Description Language
3. Abstract Models
4. Compilation and Behavior Optimization
5. FPGA Design in Industry
Hardware Compiler v.s. Software Compiler

Front End
- lex
- parse

Intermediate Form
- optimization
  - behavioral optimization

Back End
- codegen
  - a-synthesis
  - l-synthesis
  - P&R

Machine Code
- Mask layout
- FPGA bitstream
Compiler Front-end

- Architecture-level description
  - CDFG
  - Control-flow, data-flow analysis

- CDFG
- Control-flow, data-flow analysis

- parse tree
- control/data-flow analysis
- CDFG
- Behavioral optimization

- Front end
- Back end
Behavioral Optimization

- Data-flow-based transformation
  - Tree height reduction
  - Constant propagation
  - Variable propagation
  - Common subexpression elimination
  - Dead code elimination
  - Operator strength reduction
  - Etc.

- Control-flow-based transformation
  - Model expansion
  - Conditional expansion
  - Loop expansion (unrolling)
Data-flow-based Transformation
Tree Height Reduction

\[(a + (b \times c)) + d\]  
\[(a + d) + (b \times c)\]
Data-flow-based Transformation
Tree Height Reduction

(a * b) * (c * d) + a * e

a * (b * c * d + e)
Other Data Flow Transformation

- **Constant propagation**
  
  \[
  \begin{align*}
  &a = 0; \quad a = 0; \\
  &b = a + 1; \quad ---> \quad b = 1; \\
  &c = 2 \times b; \quad c = 2;
  \end{align*}
  \]

- **Variable propagation**
  
  \[
  \begin{align*}
  &a = x; \quad a = x; \quad --- \text{ can be removed} \\
  &b = a + 1; \quad ---> \quad b = x + 1; \\
  &c = 2 \times a; \quad c = 2 \times x;
  \end{align*}
  \]

- **Common subexpression elimination**
  
  \[
  \begin{align*}
  &a = x + y; \quad a = x + y; \\
  &b = a + 1; \quad ---> \quad b = a + 1; \\
  &c = x + y; \quad c = a;
  \end{align*}
  \]

- **Dead code elimination**
  
  \[
  \begin{align*}
  &a = x; \quad --- \text{ can be removed} \\
  &b = x + 1; \\
  &c = 2 \times x;
  \end{align*}
  \]
Other Data Flow Transformation (cont)

- **Operator strength reduction**
  
  \[
  \begin{align*}
  a &= x^2; \\
  b &= 3 \times x; \\
  \rightarrow \\
  a &= x \times x; \\
  t &= x \ll 1; \\
  b &= x + t;
  \end{align*}
  \]

- **Code motion (hoisting)**
  
  \[
  \begin{align*}
  \text{for} \ (i = 1; i \leq a \times b; i++) \rightarrow \ t &= a \times b; \\
  &\quad \text{for} \ (i = 1; i \leq t; i++) \{\ldots\}
  \end{align*}
  \]

- **Code motion (lowering)**
  
  \[
  \begin{align*}
  z1 &= x + y; \\
  z2 &= x - y; \\
  \text{if} \ (c = 1) \{
  &\quad a = f(z1); \\
  &\quad \rightarrow \\
  &\quad \text{else} \{
  &\quad b = f(z2);
  &\quad \}
  \}
  \quad \text{if} \ (c = 1) \{
  &\quad z1 = x + y; \\
  &\quad a = f(z1); \\
  &\quad \}
  \quad \text{else} \{
  &\quad z2 = x - y; \\
  &\quad b = f(z2); \\
  &\quad \}
  \end{align*}
  \]
Control-flow-based Transformation

- **Model expansion**
  
  \[
  x = a + b; \\
  y = a \times b; \\
  z = \text{foo}(x, y);
  \]

  \[
  \text{foo}(p, q) \{
  t = q - p; \\
  \text{return } t;
  \}
  \]

- **Conditional expansion**

  \[
  y = ab; \\
  \text{if } (a) \\
  x = b + d; \quad \rightarrow \quad x = y + d (a + b);
  \]

  \[
  \text{else} \\
  x = bd;
  \]

  \[
  x = a(b + d) + a'bd \\
  = a(b + d + bd) + a'bd \\
  = ab + ad + abd + a'bd \\
  = ab + ad + bd \\
  = ab + d (a + b)
  \]
Control-flow-based Transformation (cont)

- Loop expansion (unrolling)

```c
x = 0;
for (i = 1; i <= 3; i++)
    x = x + a[i];
```

```
x = a[1] + a[2] + a[3];
```
Summary

- HDL is used at different levels of abstraction
- In HDL, everything is in parallel with timing and delay information
- Abstract models can be represented in different forms
- HW compilation has some resemblance to SW
- CDFG is the basis for optimization and transformation
Agenda

1. System Specification
2. Hardware Modeling and Description Language
3. Abstract Models
4. Compilation and Behavior Optimization
5. FPGA Design in Industry
FPGA Design in Industry

Sub-microsecond latency
FPGA Design Flow

**Architecture Independent**
- Design the circuit
  - Design the circuit using VHDL or Verilog in Xilinx ISE or Vivado.
- Synthesis
  - Execute FPGA synthesys tools to convert the HDL codes into netlists.

**Architecture Dependent**
- Technology Mapping
  - Execute FPGA technology mapping tool to map the netlists into available FPGA resources.
- Placement and Routing
  - Execute FPGA Placement & Routing tools to place and connect the resources mapping done in the previous step onto the FPGA.
- Generate Bitstreams
  - The configuration data for the FPGA is generated in the form of bitstream.
- Reconfigure FPGA
  - Load the bitstream to the FPGA to run the circuit designed in the first step.
FPGA Design Flow

Design
- HDL
- Block design

Simulate
- Testbench (Verilog)

Generate bitstream
- 1-button click
Generate Bitstream

Logic Synthesis

Technology Mapping

Placement & Routing

Generate bitstream
PnR

- The most time-consuming step
- Memory killer
- CPU killer
- In normal cases with “easy” designs, don’t have to care much about this
What is an “easy” design?

- No strict timing requirements
  - Not dealing with timing-sensitive IOs such as DDR, Ethernet, PCIE, etc.
- The minimum speed requirement is slow (50, 100 MHz)
- Latency is not a concern
Path #1: Setup slack is -0.264 (VIOLATED)
Path #2: Setup slack is -0.063 (VIOLATED)
Path #3: Setup slack is -0.041 (VIOLATED)
Path #4: Setup slack is -0.020 (VIOLATED)
Path #5: Setup slack is -0.122 (VIOLATED)
What do we need to do?

Placement & Routing tool needs some help!
Floorplanning
Example

Path #1: Setup slack is -0.264 (VIOLATED)

Path #1: Setup slack is -0.066 (VIOLATED)
Example

Path #1: Setup slack is -0.066 (VIOLATED)
Example

Path #1: Setup slack is -0.001 (VIOLATED

Random Seed for Simulated-Annealing Placement Algorithm
FPGA Design Flow

- **Design**
  - HDL
  - Block design

- **Simulate**
  - Testbench (Verilog)

- **Generate bitstream**
  - 1-button click
Basic Structure of a Testbench

- **Stimulus**
- **Driver**
- **Design Under Test (Black Box)**
- **Model**
- **Expected**
- **Actual**

==?
Early days of Testbench

- A testbench is developed to evaluate a specific design (DUT = design under test)
- Normally, it’s as simple as unit-test
- The purpose is to quickly identify errors in
  - Wiring (connecting things together)
  - The basic functions of the module with simple input
- The output is verified by either:
  - Hand
  - External scripts/tools to generate the expected results
UVM-based Verification

- UVM = Universal Verification Methodology
- First released: Feb 2011, based on OVM (Open Verif. Method.)
- Developed by Accellera (accellera.org)
  - AMD
  - ARM
  - Intel
  - Qualcomm
  - Cadence, Mentor, Synopsys
  - ....
Language

- SystemVerilog = Verilog + OOP
- Verilog is Hardware Description Language (HDL)
- OOP: C++ -like syntax but with limited features such as a class is allowed to inherit from at most one parent.
UVM-based Testbench

Testcase

Stimulus

Driver

Design Under Test (Black Box)

Environment

Model

Expected

Compare

Scoreboard

Actual

TLM

Monitor

Un-timed

Timed

Un-timed
More Complicated

- 200 ./config/config unit_tb.sv
- 220 ./parser/itch_parser_tb.sv
- 283 ./parser/itch_self_check.sv
- 136 ./matcher/read_parameters.sv
- 719 ./matcher/matcher_tb.sv
- 189 ./matcher/matcher_self_check.sv
- 202 ./common/read_pcap.sv
- 1949 total

```
top
ghp_top_agent.sv
ghp_top_pkg.sv
ghp_top_seq_base.sv
top_tb
  file_list
  top_instantiation.sv
top_tb.sv
10 directories, 84 files
```

```
  25 ./top/ghp_top_pkg.sv
  416 ./top/ghp_top_agent.sv
  170 ./top/ghp_top_seq_base.sv
  235 ./top/ghp_top_cfg.sv
  112 ./top/ghp_top_env.sv
  8787 total
```
Benefits of UVM/SV

- OOP
  - Inheritance offers flexibility and scalability
  - Factory
    - Any verification component can be replaced by a new one with different behaviors.
      - The new component inherits the old one:
        \[
        new\_class\text{ extends old\_class}
        \]
      - In a testcase, the factory is setup to use the new_class:
        \[
        base\_class::type\_id::set\_type\_override(new\_class::get\_type())
        \]
      - The instance of the new component is created instead of the old one
        \[
        m\_inst = base\_class::type\_id::create("m\_inst", this);
        \]
Benefits of UVM/SV

- Constrained Randomization

```c
rand int arr[];
constraint c_arr{
    arr.size() < 10;
    arr.sum() inside {[1:1000]}; // if you want a positive result
    foreach(arr[i]) arr[i] inside {[0:1000]};
}
```

- Assertions

```verilog
assert property (@(posedge Clock) Req |-> ##[1:2] Ack);
```

(this assertion checks that whenever Req is asserted, Ack must be asserted on the next clock, or the following clock)

- Coverage

```verilog
covergroup memory @ (posedge ce);
    address : coverpoint addr {
        bins low  = {0,50};
        bins med  = {51,150};
        bins high = {151,255};
    }
```
Debug

- Log files
- Waveform