ECEN 449 – Microprocessor System Design

FPGAs and Reconfigurable Computing

Some of the notes for this course were developed using the course notes for ECE 412 from the University of Illinois, Urbana-Champaign
Objectives of this Lecture Unit

• Get a feel for the different technologies that can be used to implement a design
  – Flavors of hardware technologies
  – Flavors of implementation methods

• Understand the basics of how FPGAs work
  – So that the CAD tools in the lab make sense to you
Software, Custom Hardware or Reconfigurable Hardware?

- When should we use software, “custom” hardware, or reconfigurable hardware?
- Software based systems are easiest to implement
  - But there is a huge performance gap between software and hand-designed (custom) hardware systems
  - Often 100-to-1 ratio of performance (speed) or performance/area
- But custom hardware systems not so good for general computing
  - Big design effort (time, cost) are barriers to implementation
  - Not practical to buy a new machine every time you want to run a different program
- Reconfigurable systems offer best-of-both-worlds
  - Run-time programmability (in the field)
  - Hardware-level performance (although lower than custom hardware)
  - FPGAs and CPLDs are the vehicles for reconfigurable systems.
Why is Hardware Faster Than Software?

- Spatial vs. Temporal Computation
  - Processors divide computation across time, dedicated hardware divides across space.
  - But dedicated hardware is hardwired for a specific task.

\[ y = Ax^2 + Bx + C \]
Why is Hardware Faster Than Software?

• Specialization:
  – Instruction set may not provide the operations your program needs
  – Processors provide hardware that may not be useful in every program
    or in every cycle of a given program
    • Multipliers
    • Dividers

• Instruction Memory
  – Processors need lots of memory to hold the instructions that make up
    a program and to hold intermediate results.

• Bit Width Mismatches
  – In general, processors have a fixed bit width, and all computations are
    performed on that many bits
    • Multimedia vector instructions (MMX) a response to this
So why not just use Hardware?

• Dedicated hardware is
  – Dedicated (not flexible)
  – Takes long to design and develop (typical processor takes a handful of years to design, with design teams of a few hundred engineers)
  – This is expensive!
    – Only way to justify such an effort is if the customer demand guarantees high volume sales
• So there is a strong need for a design approach which has performance comparable to dedicated hardware, with ease-of-programmability comparable to software.
• Answer? Reconfigurable computing (FPGAs, CPLDs and their cousins)
Good Applications for Reconfigurable Computing

• Data Parallelism
  – Execute same computations on many independent data elements
  – Pipeline computations through the hardware
• Small and/or varying bit widths
  – Take advantage of the ability to customize the size of operators
• Low-volume applications which require rapid design turn-around time and hardware-like speeds
  – Several telecom, DSP (filters), radar, genomics (DNA sequence matching), processor emulation, neural network and similar applications.
Will FPGAs Defeat CPUs?

• **Capacity**: Instructions are very dense representation, logic blocks aren’t

• **Tools**: Compilers for reconfigurable logic aren’t very good
  – Some operations are hard to implement on FPGAs
  – C-for-FPGA technology is improving fast, though

One approach to capacity is to exploit the 90-10 rule of software
  – Run the 90% of code that takes 10% of execution time on a conventional processor
  – Run the 10% of code that takes 90% of execution time on reconfigurable logic

• But the temptation to merge the two worlds is real
  – Programmable-reconfigurable processors
A Peek Under the Hood

• In the next few slides, we will peek under the hood of some of competing hardware based digital system design platforms
• We will cover
  – Application Specific ICs (ASICs).
    • Examples are IP routing ICs
    • SSI/MSI/LSI/VLSI
  – Reconfigurable (also sometimes called programmable) ICs.
    • Examples are FPGAs, CPLDs
  – Full custom Integrated Circuits (ICs).
    • Examples are processors, GPUs, network processors, DSP processors.
Application Specific Integrated Circuits

• Very high capacity today -- 10-100M transistors
• Very high speed – 500MHz+
  – Integration
  – Specificity
• Can use any design style below (or a hybrid)
  – Full Custom
  – Standard-cell (synthesized) – dominating methodology due to manufacturing considerations
• Long fabrication time
  – Weeks-months from completed design to product
• Only economical for high-volume parts
  – Making the masks required for fabrication is becoming very expensive, in the order of $1M per design
Deep Submicron Design Challenges

- This slide discusses why ASICs are becoming less popular in recent times (compared to reconfigurable ICs)
- Physical effects are increasingly significant
  - Parasitics, reliability issues, power management, process variation, etc.
- Design complexity is high
  - Multi-functionality integration
  - Design verification is a major limitation on time-to-market
- Cost of fabrication facilities and mask making has increased significantly
Rapid Increase in Manufacturing Cost

<table>
<thead>
<tr>
<th>Process (um)</th>
<th>2.0</th>
<th>...</th>
<th>0.8</th>
<th>0.6</th>
<th>0.35</th>
<th>0.25</th>
<th>0.18</th>
<th>0.13</th>
<th>0.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Mask cost ($K)</td>
<td>1.5</td>
<td>1.5</td>
<td>2.5</td>
<td>4.5</td>
<td>7.5</td>
<td>12</td>
<td>40</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td># of Masks</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>26</td>
<td>30</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>Mask Set cost ($K)</td>
<td>18</td>
<td>18</td>
<td>30</td>
<td>72</td>
<td>150</td>
<td>312</td>
<td>1,000</td>
<td>2,000</td>
<td></td>
</tr>
</tbody>
</table>

Source: EETimes
The Cost of Next Generation Product

Source: IBS Inc.

Product Cost

Wireless chip case
Networking chip case

Total Product Cost ($M)

$30M ~ $50M @ 90nm

Engineering Cost – 60% up
Manufacturing Cost – 40% up
NRE/Mask Cost – 100% up
Respin cost – 78% up
Programmable Logic Devices

• Early version: Mask-Programmable Gate Arrays
  – Build standard layout of transistors on chip
  – Customer specifies wiring to connect transistors into gates/system
  – Only has to go through last few mask steps of fabrication process, so faster than full chip fabrication
  – May become popular again in the near future

• Newer version: Programmable Logic Devices (PLD)
  – Use AND-OR array to implement arbitrary Boolean functions
  – Programmed by burning fuses that define connection from input wires to gates
  – Customer site programming allows rapid prototyping
  – Limited capacity, functionality
    • Generally have to be used in conjunction with other parts to hold state
    • Used to implement logic with moderate number of inputs (< 20)
Programmable Logic Device Advantages

- Short TAT (total turnaround time)
- No or very low NRE (non-recurring engineering) costs.
- Field-reprogrammable
- Platform-based design
Today - Two Major Types of Programmable Logic

• CPLD (complex programmable logic device)
  – coarse-grained two-level AND-OR programmable logic arrays (PLAs)
  – fast and more predictable delay
  – simpler interconnect structures

• FPGA (field programmable gate array)
  – fine-grained logic cells
  – high logic density
  – good design flexibility (field programmable), easy redesign (just reprogram the chip!)
  – arguably more popular

• Increasing ASIC design costs are making FPGAs more popular. This technology is therefore important to learn about. Hence this course.
  – Enables “garage” technology companies to thrive. This has a huge impact.

Texas A&M University
Evolution of the FPGA

- Early FPGAs used mainly for “glue logic” between other components
  - Simple CLBs, small number of inputs
  - Focus was on implementing “random” logic efficiently
- As capacities grew, other applications emerged
  - FPGAs as alternative to custom IC’s for entire applications
  - Computing with FPGAs
- FPGAs have changed to meet new application demands
  - Carry chains, better support for multi-bit operations
  - Integrated memories, such as the block RAMs in the devices we’ll use
  - Specialized units, such as multipliers, to implement functions that are slow/inefficient in CLBs
  - Newer devices incorporate entire CPUs: Xilinx Virtex II Pro has 1-4 Power PC CPUs (we will use such a device in our lab!!)
    - Devices that don’t have CPU hardware generally support synthesized CPUs
Full Custom ICs

• These have captured an important niche in hardware implementation of systems
• Microprocessors, GPUs, network processors, DSP processors are key examples.
  – HIGH sale volume (required to justify huge development cost and time)
  – These are often the flagship products of many semiconductor companies (Intel, IBM, AMD, TI, Freescale, etc)
  – These designs are “custom” designed, to do a specific task extremely fast, with minimum area and power.
FPGAs in Detail

• Now in the next few slides, we will look at the technology that is inside an FPGA IC.
• This will allow us to understand how the FPGA works
• After this, we will be able to make sense of the design flow that is used to design a FPGA based circuit.
Field-Programmable Gate Arrays

• Based on Configurable Logic Blocks (CLB)
A Generic FPGA Architecture

Programmable IO

Programmable Logic (CLB)

Programmable Routing
What’s in a CLB?

Inputs

Look-Up Table (LUT)

Clock

Enable

Out

State
Xilinx CLB – a.k.a. “Slice”

Figure 1: Simplified Block Diagram of XC4000-Series CLB (RAM and Carry Logic functions not shown)

Page 4-12, Xilinx XC400 Series Field Programmable Gate Arrays Product Specification
An Implementation of a 4-input Look-up Table (4-LUT)

\[ \text{Out} = f (\text{in0}, \text{in1}, \text{in2}, \text{in3}) \]
Input-Output Blocks

• One IOB per FPGA pin
  – Allows pin to be used as input, output, or bidirectional (tri-state)

• Inputs
  – Direct
  – Registered
  – Drive dedicated decoder logic for address recognition

• IOB may also include logic for boundary scan (JTAG)
Figure 16: Simplified Block Diagram of XC4000E IOB

Figure 16, Xilinx XC4000 Series Field Programmable Gate Arrays Product Specification
Interconnect

• 2-Dimensional mesh of wires, with switching elements at wire crossings to control routing
  – Bit patterns stored into the switch FFs determine routing
  – Switch connections programmed as part of configuring array

• To optimize for speed, many designs include multiple lengths of wire
  – Single-length (connect adjacent switches)
  – Double-length (connect to switches two hops away)
    • Long lines (run entire length/width of array)
Interconnect Diagram

Figure 29: Single- and Double-Length Lines, with Programmable Switch Matrices (PSMs)

Figure 29, Xilinx XC400 Series Field Programmable Gate Arrays Product Specification
One Commercial FPGA, Altera Stratix II
Chip Shot - Xilinx Spartan-3 die image

• Note the regularity…
Design Variables

- The following issues are something that the company which designs the FPGA needs to worry about. The user of the FPGA is agnostic to these issues.
- # of inputs to LUT
  - Trade off number of CLBs required vs. size of CLB and routing area
- How is logic implemented
  - Switch based? Gate based?
  - SRAM configuration? Fuse burning configuration?
- Flip-flop in CLB?
- Additional Functionality
  - Carry chains, CPU’s, block RAM files
Design Flow for Programmable Logic

RTL design
- RTL elaboration and optimization
  - Architecture-independent optimization
  - Technology mapping & Architecture-specific optimization
- Clustering & placement
  - Placement-driven optimization & incremental placement
- Routing
  - Bitstream generation

Texas A&M University
FPGAs – Pros (recap)

• Reasonably cheap at low volume
  – Good for low-volume parts, more expensive than IC for high-volume parts
  – Can migrate from SRAM based to fuse based when volume ramps up
• Short Design Cycle
• Reprogrammable (~1sec programming time)
  – Can download bug fix into units you’ve already shipped
• Large capacity (100 million gates or so, though we won’t use any that big)
  – FPGAs in the lab are “rated” at ~1M gates for 30K LE’s
• More flexible than PLDs -- can have internal state
• More compact than MSI/SSI
FPGAs – Cons (recap)

- Lower capacity, speed and higher power consumption than building an ASIC
  - Sub-optimal mapping of logic into CLB’s – often 60% utilization
  - Much lower clock frequency that max CLB max toggle rate – often 40%
  - Less dense layout and placement and slower operation due to programmability
    - Overhead of configurable interconnect and logic blocks
- PLDs may be faster than FPGA for designs they can handle
- Need sophisticated tools to map design to FPGA. But the FPGA vendor typically provides these tools (at a cost).
FPGA Design Flow

• Now that we know what the circuit structure inside an FPGA is, let's see how we go about programming an FPGA.
• In other words, we will briefly cover the steps that we undertake between
  – The conception of a design idea
  – The decision-making step of whether an FPGA will be the correct hardware platform for the design
  – The design flow we follow to obtain an FPGA based hardware realization of this design.
From Concept to Circuit

• Need to specify your design and then implement it as a functioning system
• Trade-offs between time/cost and efficiency
  – Performance of final system
  – Amount of silicon area required (manufacturing cost)
  – Time to manufacture
  – Power consumption
• Need to think about a number of factors to make decision
  – Sales volume
  – Profit margin and how performance affects it
  – Time-to-market concerns, particularly if trying to be the first product in a new area
High-Level Design

• Problem 1: modern designs are just too complex to keep in your head at one time
  – Custom chips approaching 100M transistors
  – FPGA designs approaching 1M gates
• Problem 2: Even if you could design complex systems by hand, it would take too long
  – 100M transistors at 10s/transistor = 133.5 person-years
  – Transistor counts and system speed increasing at 50% or so/year
  – Design time is critical
• # transistors per chip increasing at 50%/year
• # transistors per engineer-day increasing at 10%/year

• Need techniques that reduce the amount of human effort required to design systems
  – Let humans work at higher levels, rely on software to map to low-level designs
Increasing Design Abstraction

• Old way: specify/layout each device by hand
  – Early chips were laid out by cutting patterns in rubylith with knives

• Current State of the Art: Combination of synthesis and hand design
  – Specify entire system in HDL (Verilog or VHDL), simulate, and test
  – Use synthesis tools to convert non-performance-critical parts of the design to transistors/gates
  – Human designs critical components by hand for performance

• Where Things are Going: System-on-a-Chip Design
  – Specify design out of high-level components (cores)
  – Integrate sensors, transmitters, actuators, computers on a chip
  – Rely very heavily on tools to map design to software and hardware.
  – \textit{XUP (the board we will use in the lab) is an SoC design vehicle}
(FPGA) Design Flow

Design Entry

Simulation

Implementation

Physical Device
Design Entry

Two main methods:

• Text entry (VHDL/Verilog):
  – Compact format, no special tools required
  – Good for high-level designs and control logic

• Schematic Capture: Draw pictorial representation of circuit, tool converts into design (typically HDL description)
  – Traditionally used for low-level (transistor) designs, regular structures
  – Commonly used today in conjunction with text entry to provide visual viewing of overall structure of a design
Simulation

- Two types of HDL simulators
  - Interpreted: runs slower but more versatile and no compilation time
  - Compiled: runs faster but require compilation time and often not as versatile partly due to needs to compile all library components used.
- Both typically use Discrete-Event techniques
  - Divide time into discrete steps
    - User can select time step to trade accuracy vs. run-time
  - Keep lists of events that have to be resolved at each time step.
    - At each time step, resolve all events for the time step and schedule events for later time steps
- Output:
  - Text from output/print statements in your design
  - Errors from assert statements
  - Waveform traces
- Like any testing, the key is having good tests. The designer creates these!
Implementation of an FPGA Design

Going from simulated Verilog design to circuits

• 5 Phases
  – Synthesis
  – Timing Analysis
  – Technology Mapping
  – Place and Route
  – Bitstream Generation

(Sometimes do additional timing analysis after place and route just to make sure that the timing is good)
Synthesis

Transforms program-like VHDL into hardware design (netlist)
- Inputs
  - HDL description
  - Timing constraints (When outputs need to be ready, when inputs will be ready, data to estimate wire delay)
  - Technology to map to (list of available blocks and their size/timing information)
  - Information about design priorities (area vs. speed)

For big designs, will typically break into modules and synthesize each module separately
  - 10K gates/module was reasonable size 5 years ago, tools can 50-100K gates now
Timing Analysis

Static timing analysis is the most commonly-used approach
• Calculate delay from each input to each output of all devices
• Add up delays along each path through circuit to get critical path
• Works as long as no cycles in circuit
  – Tools let you break cycles at registers to handle feedback
• Also, ignores *false paths* in the design.

• Trade off some accuracy for run time
  – Simulation tools like SPICE will give more accurate numbers, but take much longer to run
• If the netlist passes timing analysis tests, we proceed further
Technology Mapping

- Technology mapping converts a given Boolean circuit (a netlist) into a functionally equivalent network comprised only of LUTs or PLAs
  - Basically, can divide logic into n-input functions, map each onto a CLB.
- Technology mapping is a crucial optimization step in the programmable logic design flow
- Direct impact on
  - Delay (number of levels of logic)
  - area/power (number of LUTs or PLAs)
  - Interconnects (number of edges)
- Harder problem: Placing blocks to minimize communication, particularly when using carry chains
Place and Route

Synthesis generates netlist -- list of devices and how they’re interconnected

Place and route determines how to put those devices on a chip and how to lay out wires that connect them

Results not as good as you’d like -- 40-60% utilization of devices and wires is typical for FPGA
  – Can trade off run time of tool for greater utilization to some degree, but there are serious limits
  – Beyond 80% utilization, there is a good chance that routing will fail.
Bitstream Generation

• A bitstream in FPGA-speak is a sequence of bits, which
  – Determines how the FPGA fabric is customized in order to implement
    the design.
  – It determines how IOs, CLBs, and wiring are configured.

• This bitstream is loaded (serially) into the FPGA in a final step. Now the FPGA is customized to implement the design we wanted.
  – This loading typically takes a few seconds at most.

• Reprogramming simply means that we load a new bitstream on to the FPGA.