Hardware Software Codesign with FPGAs

Instructor:
Prof. Jim Plusquellic

Text:

References:
• Hardware/Software Co-Design - Principles and Practice"
  ISBN: 0792380134

• "Co-Design for System Acceleration A Quantitative Approach"
  ISBN: 978-1-4020-5545-4

• "Embedded System Design, A Unified Hardware/Software Introduction"
  Frank Vahid and Tony Givargis, 2002.
  ISBN: 978-0-471-38678-0

Web: http://www.ece.unm.edu/~jimp/codesign
Introduction

Relevant Conferences/Symposia on Codesign:

• Conference on Formal Methods and Models for Codesign (MEMOCODE)

• CODES+ISSS: The premier conference for System-Level Design
  The CODES+ISSS Conference is the merger of two major international symposia on hardware/software codesign and system synthesis.

• DAC: Design Automation Conference

• ASP-DAC: Asia South Pacific Design Automation Conference

• CASES: International Conference on Compilers, Architecture, and Synthesis for Embedded Systems

• ICCAD: International Conference on Computer Aided Design
The Nature of Hardware and Software

What is H/S Codesign (Prof. Schaumont’s definition):

Hardware/Software Codesign is the design of cooperating hardware components and software components in a single design effort.

Other definitions

- HW/SW Codesign is a design methodology supporting the **concurrent** development of hardware and software (cospecification, codevelopment and coverification) in order to achieve *shared functionality and performance goals* for a combined system.
- HW/SW Codesign means *meeting system level objectives* by exploiting the **synergism** of hardware and software through their concurrent design.
  
- Codesign is the concurrent development of hardware and software.
The Nature of Hardware and Software

Choosing between implementing a design in HW or implementing it in SW may seem like a no-brainer -- clearly writing software is easier!

Proponents of hardware implementation will argue that performance is a big plus of hardware (an ASIC) over software (running on a microprocessor)

Unfortunately, the speed advantage of ASICs over software is fading

High-end processor have VERY high clk frequencies (much faster than ASICs)

Therefore, absolute performance is not a very good metric to compare hardware and software

A much better metric (that is independent of clk frequency) is energy efficiency, i.e., the amount of useful work done per unit of energy

This metric can be applied to all architectures
The Nature of Hardware and Software

Consider the energy consumption of an AES engine (encryption) on different architectures.

Encryption *throughput* using microprocessors is on order with throughput using dedicated ASICs.

This is true b/c of the shorter clock period of microprocessors.

When evaluated from a total energy perspective, dedicated hardware engines win hands-down.

NOTE log scale!
The Nature of Hardware and Software

This is true b/c there is a **large overhead** associated with executing software instructions in the microprocessor implementation

- Instruction and operand fetch from memory
- Complex state machine for control of the datapath, etc.

Flexibility comes with a significant energy cost -- one which energy optimized applications cannot tolerate

Therefore, you will **never find** a Pentium processor in a cell phone!

Specialized hardware architectures are usually also more efficient than software from a **relative perspective**

**Relative performance** means the amount of useful work done per clock cycle
The Nature of Hardware and Software

Highly parallel implementations are at an advantage b/c they do many things at the same time.

HW crypto implementations have a higher relative performance when compared to embedded processors.

Fig. 1.4 Cryptography on Small Embedded Platforms

HW crypto implementations have a higher relative performance when compared to embedded processors.
Why Hardware/Software Codesign?

System-on-chip (SoC) design integrates processors and peripherals on a single chip. The development of software for such a multi-component chip is inevitably tied to the hardware architecture.

It is common to have differing opinions about what portions of the design are to be implemented in HW and which are to be implemented in SW.

Arguments in favor of increasing the amount of on-chip dedicated HW:

- **Energy Efficiency:**
  
  Nearly every electronic consumer product today carries a battery, e.g., iPod, PDA, mobile phone, Bluetooth device, etc.

  HW/SW codesign plays an important role of trading function-specialization and energy-efficiency by moving parts of the flexible SW into fixed HW.

- **Power Densities:**
  
  Performance in modern high-end processors can no longer be scaled by speeding up the clk.
  
  Instead, there is a broad and fundamental shift towards parallel architectures.
Why Hardware/Software Codesign?

Arguments in favor of increasing the amount of on-chip dedicated HW (cont):

• **Power Densities** (cont.)
  
  However, there is no dominant parallel computer architecture that can handle all applications:
  
  • Symmetric multiprocessors attached to the same memory
  • FPGAs used as accelerator engines for classic processors
  • Multi-core and many-core architectures such as Graphics Processing Engines

This means that the software designer will not be able to ’ignore’ or ’abstract’ the computer architecture in next generation systems

**Architecture-awareness** comes natural with HW/SW co-design

Arguments for flexibility and thus for increasing the amount of on-chip SW:

• **Design Complexity**
  
  High-end SoCs are extremely complex -- they contain multiple processors, large embedded memories, multiple peripherals and input-output devices
Why Hardware/Software Codesign?
Arguments for flexibility and thus for increasing the amount of on-chip SW (cont.):

• **Design Complexity** (cont.)
  Software bugs are easier to address than hardware bugs

• **Design Cost**
  New chips are very expensive to design -- designers make chips **programmable**
  so they can be reused over multiple products or product generations

• **Shrinking Design Schedules**
  Each new technologies is exponentially more complex than the previous generation,
  and the move to the next generation happens more quickly

  For the designer, this means that each new product generation brings more work
  that needs to be completed in a shorter amount of time

  Shrinking design schedules require engineering teams to work on multiple tasks
  at once: HW and SW are developed concurrently
Why Hardware/Software Codesign?

- **Deep-submicron Effects**
  Designing new hardware from-scratch in high-end silicon processes is difficult b/c of technology-related second-order effects
  - Increased *variability*
  - Decreased *reliability*

  It is easier to leverage tried-and-true embedded cores and implement functionality in SW

  The above drivers have made programmable systems **more popular** than dedicated, hardcoded designs

  This forces the designer to deal with the programmable part (SW) as well as the fixed part (HW) of an application

  These concepts provide a **traditional** definition of HW/SW design:
  Hardware/Software Codesign is the design of **cooperating** HW components and SW components in a single design process
**Why Hardware/Software Codesign?**

For example, if you would design the architecture of a processor and at the same time develop a program that runs on that processor, then you are using HW/SW codesign.

This definition does NOT define what SW and HW mean, however -- there are many forms and the distinction is becoming blurred.

- An FPGA is a hardware circuit that can be reconfigured.
  The **program** for an FPGA is a bitstream, and it is used to configure its logic function.

  Writing software for an FPGA really looks like hardware development - even though it is software.

- A soft-core is a processor configured within an FPGA’s reconfigurable fabric.
  Subsequently, the soft-core processor itself can be programmed!

  Software executing on top of other software!
Why Hardware/Software Codesign?

- A **digital signal processor** (DSP) has instructions which are optimized for signal-processing applications
  
  Although C programs can be written, efficient programs require detailed knowledge of the DSP HW architecture, connecting the efficiency of the software to capabilities of the hardware.

- An **application-specific instruction-set processor** (ASIP) is a processor with a customizable instruction set
  
  The hardware of such a processor can be extended, and these hardware extensions can be encapsulated as new instructions for the processor

  Thus, an ASIP designer will develop a hardware implementation for these custom instructions, and subsequently write software that uses those instructions

- The **CELL processor**, used in the Playstation-3, contains one control processor and 8 slave-processors, interconnected through a high-speed on-chip network
  
  Writing software for a CELL requires the description of the computation of 9 concurrent communicating programs and of their on-chip network config.
Why Hardware/Software Codesign?
A common characteristic of all these examples is that creating the SW requires intimate familiarity the HW

A better definition of HW/SW codesign
HW/SW codesign is the partitioning and design of an application in terms of fixed and flexible components

The Hardware-Software Codesign Space
For a given application, there are many different possible solutions as exemplified by the previous discussion

The collection of all possible implementations is called the HW/SW codesign space

The following figure represents the design space symbolically
At the top is the specification, a description of the desired application

A designer will then choose a platform to use in the implementation (a platform is a collection of programmable components)
The Hardware-Software Codesign Space
Once the platform is chosen, the designer maps the application onto the platform

The format of the software varies according to the components of the platform
For example, a RISC processor may be programmed in C, while an FPGA could be programmed starting from a HDL program
The Hardware-Software Codesign Space

In this section, we will investigate the following three activities:

- Platform selection
- Application mapping
- Platform programming

Specification (can start out as informal ideas):

For example, a new application can be a novel way of encoding audio in a more economical format than current encoding methods.

Designers then write C programs to render their idea in detail -- it becomes a specification of the application -- not an actual implementation.

Very often, a specification is just a piece of English text, typically resulting in ambiguity and open questions.

Step 1 of implementation: select a target platform

For HW/SW codesign, we select a programmable component(s) as shown in the previous figure, e.g., RISC micro, FPGA, etc.
Target Platforms

- Microprocessor
- FPGA
- SoC

Microcontroller

DSP
SoC Examples

Example System-on-Chip (SoC) with IP cores

- Transreflective monochrome backlit display
- Hynix #HY57V641629 SDRAM 8MB
- Fujitsu #MBM29D1323 Flash 4MB
- Motorola #MC68VZ328 DragonBall Proc.
- Xilinx #XCR3064 CPLD
- Analog Devices #AD7873 Screen digitizer
- Agere #I2R50INE POM baseband proc.
- Philips #PDIUBD12 USB Interface
- Maxim #MAX4472 Pow. Amp contrl
- Maxim #MAX3386 Transceivers
- RF Micro #RF2173 Pow Amp
- Motorola #MC1376VF Dig. Transceivers
- Motorola #MC1376VF Dig. Transceivers
- Maxim #MAX3386 Transceivers
- Maxim #MAX4472 Pow. Amp contrl
- RF Micro #RF2173 Pow Amp
- Memory
- Processor
- RF
- DSP
- Interface
- Universal Connector
- TCXO
- K001 VCO
- FPGA
- Manual inputs
- MMC-fermat memory card slot
Codesign Examples
Video Codec (H261)

Camera

Grabber
VLD
MSQ
IDCT

Display

Unframer
Framer
DCT
Pred. Filter
M.E.
VLC

MSQ bus
MCC bus

ISDN Line

uP+code
SW Processors

HW
HW Processors
The Hardware-Software Codesign Space

(Note: An ASIC is normally not considered a programmable component, but instead represents a boundary in the codesign space)

The definitions of SW and HW have **very different meanings** depending on the platform:

- For a RISC processor, software is written in C, while the hardware is a general-purpose processor
- For FPGAs, software is written in HDL
  
  When the FPGA contains a soft-core processor, we will also write additional platform software in C
- For a DSP, we use a combination of C and assembly code for software while the hardware is a specialized processor architecture
- For an ASIP, the processor can be specialized to a particular application domain by adding new instructions and by extending the processor datapath
  
  The software of an ASIP can contain C code as well as a hardware description of the processor extensions
- For an ASIC, the application is written in HDL, and converted into a hardcoded netlist -- here the application and the platform are merged to a single entity
The Hardware-Software Codesign Space

Each of the above platforms presents a trade-off between application flexibility and platform efficiency.

The wedge-shape of Figure 1.1 expresses this idea:

**Flexibility** refers to how well the platform can be adapted to different applications.

Flexibility in platforms is desired because it allows designers to make changes to the application after the platform is fabricated.

Note that when a platform becomes more specialized, the programming tends to become more specialized too.

**Efficiency** refers to either absolute performance (i.e. time-efficiency) or to the energy efficiency.

A specialized platform tends to be more efficient than a general platform, because its HW components are optimized for the application.
The Hardware-Software Codesign Space

**Platform Programming** is the task of mapping SW onto HW

This can be done automatically, e.g., using a C compiler or an HDL synthesis tool

However, many platforms are not just composed of simple components, but rather require multiple pieces of software, possibly in different programming languages

For example, the platform may consist of a RISC processor and a specialized hardware coprocessor

Here, the software consists of C (for the RISC) as well as dedicated coprocessor instruction-sequences (for the coprocessor).

Therefore, the reality of platform programming is more complicated, and automated tools and compilers and NOT always available
The Hardware-Software Codesign Space

Another concept reflected in Figure 1.3 is the **domain-specific platform**

*General-purpose* platforms, such as RISC and FPGA, are able to support a broad range of applications

*Application-specific* platforms, such as the ASIC, are optimized to execute a single application

In the middle is a class called **domain-specific** platforms that are optimized to execute a **range of applications** in a particular application domain

- Signal-processing, cryptography, networking, are examples of domains

And domains can have *sub-domains*, e.g., voice-signal processing vs. video-signal processing

- Optimized platforms can be designed for each of these cases

DSPs and ASIPs are two examples of domain-specific platforms
The Hardware-Software Codesign Space

Difficult questions

- How does one select a platform for a given specification (harder problem of two)
- How can one map an application onto a selected platform

Seasoned designers choose based on their previous experience with similar applications.

The second issue is also challenging, but can be addressed in a more systematic fashion using a design methodology.

A design method is a systematic sequence of steps to convert a specification into an implementation.

Design methods cover many aspects of application mapping:

- Optimization of memory usage
- Design performance
- Resource usage
- Precision and resolution of data types, etc.

A design method is a canned sequence of design steps.