Outline

1. Introduction to embedded systems
2. Design metrics
3. Embedded system technologies
4. Design processes
5. Computation models and languages
6. Design technologies
7. Summary

References

Embedded System Design: A Unified Hardware/Software Introduction
Frank Vahid and Tony Givargis, John Wiley & Sons, 2002

Slides are mainly based on the lecture slides available at the book homepage:
http://www.cs.ucr.edu/~vahid/courses/122a_f99/

Draft version online available at:
http://www.cs.ucr.edu/~vahid/courses/122a_f99/

Further references:
• Vorlesung „Eingebettete Systeme“. Bernd Kleinjohann und Lisa Kleinjohann, C-LAB, Fänsterallee 11, Raum FZU14, Tel. 606101 oder 606102, e-mail: bernd@c-lab.de lisa@c-lab.de
• Embedded Systems Internet Resources:
  http://www.compapp.dcu.ie/~cdaly/embed/embedsys.html

A “short list” of embedded systems

And the list goes on and on

Some common characteristics of embedded systems

• Single-functioned
  • Executes a single program, repeatedly
• Tightly-constrained
  • Low cost, low power, small, fast, etc.
• Reactive and real-time
  • Continually reacts to changes in the system’s environment
  • Must compute certain results in real-time without delay

Embedded systems overview

• Embedded computing systems
  • Computing systems embedded within electronic devices
  • Hard to define. Nearly any computing system other than a desktop computer
  • Billions of units produced yearly, versus millions of desktop units
  • Perhaps 50 per household and per automobile
Design challenge – optimizing design metrics

• Obvious design goal:
  • Construct an implementation with desired functionality

• Key design challenge:
  • Simultaneously optimize numerous design metrics

• Design metric
  • A measurable feature of a system’s implementation
  • Optimizing design metrics is a key challenge

Design challenge – optimizing design metrics

• Common metrics
  • Unit cost: the monetary cost of manufacturing each copy of the system, excluding NRE cost
  • NRE cost (Non-Recurring Engineering cost): The one-time monetary cost of designing the system
  • Size: the physical space required by the system
  • Performance: the execution time or throughput of the system
  • Power: the amount of power consumed by the system
  • Flexibility: the ability to change the functionality of the system without incurring heavy NRE cost

• Common metrics (continued)
  • Time-to-prototype: the time needed to build a working version of the system
  • Time-to-market: the time required to develop a system to the point that it can be released and sold to customers
  • Maintainability: the ability to modify the system after its initial release
  • Correctness, safety, many more

Design metric competition -- improving one may worsen others

• Expertise with both software and hardware is needed to optimize design metrics
  • Not just a hardware or software expert, as is common
  • A designer must be comfortable with various technologies in order to choose the best for a given application and constraints

The performance design metric

• Widely-used measure of system, widely-abused
  • Clock frequency, instructions per second – not good measures
  • Digital camera example – a user cares about how fast it processes images, not clock speed or instructions per second

• Latency (response time)
  • Time between task start and end
  • e.g., Camera A and B process images in 0.25 seconds
  • Throughput
    • Tasks per second, e.g. Camera B may process 8 images per second (by capturing a new image while previous image is being stored).

• Speedup of B over S = B’s performance / A’s performance
  • Throughput speedup = 8/4 = 2

An embedded system example -- a digital camera

• Single-functioned – always a digital camera
• Tightly-constrained – Low cost, low power, small, fast
• Reactive and real-time – only to a small extent

Digital camera chip

lens

CCD

Microcontroller
CCD preprocessor
Pixel coprocessor
A2D
D2A
JPEG codec
DMA controller
Memory controller
ISA bus interface
UART
LCD ctrl
Display ctrl
Multiplier/Accum

Hardware
Software

Throughput
Profit
NRE cost
Participation

Widely-used measures of system, widely-abused
  • Clock frequency, instructions per second – not good measures
  • Digital camera example – a user cares about how fast it processes images, not clock speed or instructions per second

• Latency (response time)
  • Time between task start and end
  • e.g., Camera A and B process images in 0.25 seconds
  • Throughput
    • Tasks per second, e.g. Camera A processes 4 images per second
    • Throughput can be more than latency seems to imply due to concurrency, e.g. Camera B may process 8 images per second (by capturing a new image while previous image is being stored).

• Speedup of B over S = B’s performance / A’s performance
  • Throughput speedup = 8/4 = 2
Moore’s law

• The most important trend in embedded systems
  • Predicted in 1965 by Intel co-founder Gordon Moore
  • IC transistor capacity has doubled roughly every 18 months for the past several decades

Design productivity gap

• While designer productivity has grown at an impressive rate over the past decades, the rate of improvement has not kept pace with chip capacity

Design process model

• Describes order that design steps are processed
  • Behavior description step
  • Behavior to structure conversion step
  • Mapping structure to physical implementation step

  Waterfall model
  • Proceed to next step only after current step completed

  Spiral model
  • Proceed through 3 steps in order but with less detail
  • Repeat 3 steps gradually increasing detail
  • Keep repeating until desired system obtained
  • Becoming extremely popular (hardware & software development)

Waterfall method

• Not very realistic
  • Bugs often found in later steps that must be fixed in earlier step
  • E.g., forgot to handle certain input condition
  • Prototype often needed to know complete desired behavior
  • E.g., customer adds features after product demo
  • System specifications commonly change
  • E.g., to remain competitive by reducing power, size
  • Certain features dropped
  • Unexpected iterations back through 3 steps cause missed deadlines
  • Lost revenues
  • May never make it to market

Spiral method

• First iteration of 3 steps incomplete
  • Much faster, though
  • End up with prototype
  • Use to test basic functions
  • Get idea of functions to add/remove
  • Original iteration experience helps in following iterations of 3 steps
  • Must come up with ways to obtain structure and physical implementations quickly
  • E.g., FPGAs for prototype
  • silicon for final product
  • May have to use more tools
  • Extra effort/cost
  • Could require more time than waterfall method
  • If correct implementation first time with waterfall

Modelling embedded systems

• Describing embedded system’s processing behavior
  • Can be extremely difficult
  • Complexity increasing with increasing IC capacity
  • Past: washing machines, small games, etc.
  • Hundreds of lines of code
  • Today: TV set-top boxes, Cell phone, etc.
  • Hundreds of thousands of lines of code
  • Desired behavior often not fully understood in beginning
  • Many implementation bugs due to description mistakes/omissions
  • English (or other natural language) common starting point
  • Precise description difficult to impossible
  • Example: Motor Vehicle Code – thousands of pages long...
**Models and languages**

- **How can we (precisely) capture behavior?**
  - We may think of languages (C, C++, Java), but computation model is the key
- **Common computation models:**
  - **Sequential program model**
    - Statements, rules for composing statements, semantics for executing them
  - **Communicating process model**
  - **Multiple sequential programs running concurrently**
  - **State machine model**
    - For control dominated systems, monitors control inputs, sets control outputs
  - **Dataflow model**
    - For data dominated systems, transforms input data streams into output streams
  - **Object-oriented model**
    - For breaking complex software into simpler, well-defined pieces

**Finite-state machine (FSM) model**

- Nodes with more complex transformations
- Nodes with arithmetic transformations
- Nodes with more complex arithmetic

**Concurrent process model**

- Describes functionality of system in terms of two or more concurrently executing subtasks
- Many systems easier to describe with concurrent process model because inherently multitasking
- E.g., simple example:  
  - Read two numbers X and Y
  - Display "Hello world." every X seconds
  - Display "How are you?" every Y seconds
- More effort would be required with sequential program or state machine model

**Dataflow model**

- **Derivative of concurrent process model**
- **Nodes represent transformations**
  - May execute concurrently
  - Edges represent flow of tokens (data) from one node to another
  - May or may not have token at any given time
  - When all of node’s input edges have at least one token, node may fire
  - When node fires, it consumes input tokens processes transformation and generates output token
  - Nodes may fire simultaneously
  - Several commercial tools support graphical languages for capture of dataflow model
  - Can automatically translate to concurrent process model for implementation
  - Each node becomes a process

**Role of appropriate model and language**

- **Finding appropriate model to capture embedded system is an important step**
  - Model shapes the way we think of the system
  - Originally thought of sequence of actions, wrote sequential program
  - Then wait for requested floor to differ from target floor
  - Then, we close the door
  - Then, we open the door
  - Then, we repeat this sequence
  - To create state machine, we thought in terms of states and transitions among states
  - When system must react to changing inputs, state machine might be best model
- **Language should capture model easily**
  - Ideally should have features that directly capture constructs of model
  - Other factors may force choice of different model
  - Structured techniques can be used instead
  - E.g., Template for state machine capture in sequential program language
Three key embedded system technologies

- Technology
  - A manner of accomplishing a task, especially using technical processes, methods, or knowledge
  - Three key technologies for embedded systems
    - Processor technology
    - IC technology
    - Design technology

Communication among processes

- Processes need to communicate data and signals to solve their computation problem
- Processes that don’t communicate are just independent programs solving separate problems
- Basic example: producer/consumer
  - Process A produces data items, Process B consumes them
  - E.g., A decodes video packets, B display decoded packets on a screen
- How do we achieve this communication?
  - Two basic methods
    - Shared memory
    - Message passing
- Important mechanisms: Mutual exclusion, Synchronization => Deadlocks!

Implementation

- Mapping of system’s functionality onto hardware processors:
  - captured using computational model(s)
  - written in some language(s)
- Implementation choice independent from language(s) choice
- Implementation choice based on power, size, performance, timing and cost requirements
- Final implementation tested for feasibility
  - Also serves as blueprint/prototype for mass manufacturing of final product

Concurrent process model: implementation

- Can use single and/or general-purpose processors
  - (a) Multiple processors, each executing one process
    - True multitasking (parallel processing)
    - General purpose processors
      - Use programming language like C and compile to instructions of processor
      - Expensive and in many cases not necessary
    - Custom single-purpose processors
      - More common
  - (b) One general-purpose processor running all processes
    - Most processes don’t use 100% of processor time
    - Can share processor time and still achieve necessary execution rates
  - (c) Combination of (a) and (b)
    - Multiple processes run on one general-purpose processor while one or more processes run on own single purpose processor

Processor technology

- The architecture of the computation engine used to implement a system’s desired functionality
- Processor does not have to be programmable
  - “Processor” not equal to general-purpose processor
IC technology

- The manner in which a digital (gate-level) implementation is mapped onto an IC
- IC: Integrated circuit, or “chip”
- IC technologies differ in their customization to a design
- IC’s consist of numerous layers (perhaps 10 or more)
- IC technologies differ with respect to who builds each layer and when
- Types: Full-custom VLSI, Semi-custom ASIC (gate array and standard cell), PLD (Programmable Logic Device)

Design Technology

- The manner in which we convert our concept of desired system functionality into an implementation

The co-design ladder

- In the past:
  - Hardware and software design technologies were very different
  - Recent maturation of synthesis enables a unified view of hardware and software
- Hardware/software “codesign”

The choice of hardware versus software for a particular function is simply a tradeoff among various design metrics, like performance, power, size, NRE cost, and especially flexibility; there is no fundamental difference between what hardware or software can implement.

Independence of processor and IC technologies

- Basic tradeoff:
  - General vs. custom
  - With respect to processor technology or IC technology
  - The two technologies are independent

Design technology

- Design task:
  - Define system functionality
  - Convert functionality to physical implementation while satisfying constrained metrics
  - Optimizing other design metrics
  - Designing embedded systems is hard
  - Complex functionality
  - Millions of possible environment scenarios
  - Competing, tightly constrained metrics
  - Productivity gap
  - As low as 10 lines of code or 100 transistors produced per day

Improving productivity

- Design technologies developed to improve productivity
  - We focus on technologies advancing hardware/software unified view
    - Automation
    - Program replaces manual design
    - Synthesis
    - Reuse
      - Predesigned components
      - Cores
      - General-purpose and single-purpose processors on single IC
    - Verification
      - Ensuring correctness/completeness of each design step
      - Hardware/software co-simulation
Automation: synthesis

- Early design mostly hardware
- Software complexity increased with advent of general-purpose processor
- Different techniques for software design and hardware design
  - Caused division of the two fields
- Design tools evolve for higher levels of abstraction
  - Different rate in each field
  - Hardware/software design fields rejoining
- Both can start from behavioral description in sequential program model
  - Many more design dimensions
  - Optimization critical

Implementation

- Assembly instructions
- Machine instructions
- Logic gates
- Logic equations / FSM's
- Register transfers
- Sequential program code (e.g., C, VHDL)
- Compilers (1960s, 1970s)
- Assemblers, linkers (1950s, 1960s)
- Behavioral synthesis (1990s)
- RT synthesis (1980s, 1990s)
- Logic synthesis (1970s, 1980s)
- Microprocessor plus program bits
- VLSI, ASIC, or PLD implementation

The codesign ladder

Gajski’s Y-chart

- Each axis represents type of description
  - Behavioral
    - Defines outputs as function of inputs
    - Algorithms but no implementation
  - Structural
    - Implement behavior by connecting components with known behavior
  - Physical
    - Given specifications and sizes on chip/board
- Synthesis converts behavior at given level to structure at same level or lower
  - E.g.,
    - FSM ? gates, flip-flops (same level)
    - FSM ? transistors (lower level)
    - FSM X registers, FUs (higher level)
    - FSM X processors, memories (higher level)

Verification

- Ensuring design is correct and complete
  - Correct
    - Implements specification accurately
  - Complete
    - Describes appropriate output to all relevant input
- Formal verification
  - Hard
    - For small designs or verifying certain key properties only
- Simulation
  - Most common verification method
  - Controllability
    - Control time
      - Stop/start simulation at any time
    - Control data values
      - Inputs or internal values
  - Observability
    - Examine system/environment values at any time
  - Debugging
    - Can stop simulation at any point and:
      - Observe internal values
      - Modify system/environment values before restarting
    - Can step through small intervals (i.e., 500 nanoseconds)

Disadvantages

- Simulation setup time
  - Often has complex external environments
  - Could spend more time modeling environment than system
- Models likely incomplete
  - Some environment behavior undocumented if complex environment
  - May not model behavior correctly
- Simulation speed much slower than actual execution
  - Sequentializing parallel design
    - IC: gates operate in parallel
    - Simulation: analyze inputs, generate outputs for each gate 1 at time
  - Several programs added between simulated system and real hardware
    - 1 simulated operation:
      - = 10 to 100 simulator operations
      - = 100 to 10,000 operating system operations
      - = 1,000 to 100,000 hardware operations

Advantages over physical implementation

- Controllability
  - Control time
    - Stop/start simulation at any time
  - Control data values
    - Inputs or internal values
- Observability
  - Examine system/environment values at any time
- Debugging
  - Can stop simulation at any point and:
    - Observe internal values
    - Modify system/environment values before restarting
    - Can step through small intervals (i.e., 500 nanoseconds)

Simulation speed

- Relative speeds of different types of simulation/emulation
  - 1 hour actual execution of SOC
    - = 1.2 years instruction-set simulation
    - = 10,000,000 hours gate-level simulation
Emulators

- General physical device system mapped to
  - Microprocessor emulator
    - Microprocessor IC with some monitoring, control circuitry
    - FPGA (10s to 100s)
    - Usually supports debugging tasks
  - Created to help solve simulation disadvantages
    - Mapped relatively quickly
    - Hours, days
    - Can be placed in real environment
    - No environment setup time
    - No incomplete environment
    - Typically faster than simulation
    - Hardware implementation

Reuse: intellectual property cores

- Commercial off-the-shelf (COTS) components
  - Predesigned, prepackaged ICs
  - Implements GPP or SPP
  - Reduces design/debug time
  - Have always been available
- System-on-a-chip (SOC)
  - All components of system implemented on single chip
  - Made possible by increasing IC capacities
  - Changing the way COTS components sold
    - As intellectual property (IP) rather than actual IC
      - Behavioral, structural, or physical descriptions
      - Processor-level components known as cores
    - SOC built by integrating multiple descriptions

Summary

- Embedded systems are everywhere
- Key challenge: optimization of design metrics
  - Design metrics compete with one another
- A unified view of hardware and software is necessary to improve productivity
- Three key technologies
  - Processor: general-purpose, application-specific, single-purpose
  - IC: Full-custom, semi-custom, PLD
  - Design: Compilation/synthesis, libraries/IP, test/verification
- Computation models are distinct from languages
- Sequential program model is popular
  - Most common languages like C support it directly

Summary

- State machine models good for control
- Concurrent process model for multi-task systems
  - Communication and synchronization methods exist
  - Scheduling is critical
- Dataflow model good for signal processing
- Design technology seeks to reduce gap between IC capacity growth and designer productivity growth
- Synthesis has changed digital design
- Increased IC capacity means sw/hw components coexist on one chip
- Design paradigm shift to core-based design
- Simulation essential but hard
- Spiral design process is popular