Architecture of Embedded Systems

José Costa

Software for Embedded Systems

Departamento de Engenharia Informática (DEI)
Instituto Superior Técnico

2015-09-22
Outline

- Embedded System Design and Development Process
  - 6 Stages of Creating an Embedded Architecture
    - Solid Technical Base
    - Architecture Business Cycles
    - Architectural Patterns & Reference Models
    - Architectural Structures
    - Document the Architecture
    - Analyze and Evaluate Architecture
Levels of abstraction

- Requirements
- Specification
- Architecture
- Component Design
- System Integration
Various models exist

Noergaard proposes model based on the Waterfall and Spiral models

- Creating the architecture
- Implementing the architecture
- Testing the system, and
- Maintaining the system
Embedded Systems Design and Development Lifecycle Model

1. **Phase 1: Creating The Architecture**
   - Product Concept
   - Preliminary Requirements Analysis
   - Creation of Architecture Design
   - Develop Version of Architecture
   - Deliver Final Version of Architecture
   - Deliver & Maintain the System

2. **Phase 2: Implementing the Architecture**
   - Incorporate Feedback
   - Review and Test the System
   - Review & Obtain Feedback
   - Deliver Version of Architecture

3. **Phase 3: Testing the System**
   - Incorporate Feedback
   - Review and Test the System
   - Review & Obtain Feedback
   - Deliver Version of Architecture

4. **Phase 4: Maintaining the System**
What is an Embedded Systems Architecture?

Embedded System Architecture

Is an abstraction of the embedded device that represents the embedded system as some combination of interacting elements.

- Typically doesn’t show detailed implementation information
- Represented as some composition of interlacing elements
An embedded architecture includes

- **Elements** of the embedded system

- **Elements interacting** with an embedded system

- The **properties** of each of the individual elements, and

- The **interactive relationships** between the elements
Elements of an Architecture

- Elements are representations of hardware and/or software
- Implementation details have been abstracted out
- Only behavioral and inter-relationship information

Examples
- Class
- Layers
- Kernel
- Client/Server
- Process
- Memory
- . . .
Major Types of Elements

- **Module**
  - hardware and/or software that the system needs to function correctly

- **Component and Connector**
  - main hw/sw processing units, such as processors, a Java Virtual Machine, etc.
  - communication mechanism that inter-connects components, such as a hardware bus, or software OS messages, etc.

- **Allocation**
  - relationships between sw and/or hw elements, and external elements in various environments
  - e.g. where the software resides in the hardware
Why Care About The Architecture of an Embedded System?

- Powerful tool used to understand an embedded systems design or to resolve challenges faced when designing a new system.
- Solid basis for analyzing and testing the quality of a device and its performance.
- Accurately estimates and reduces costs through its demonstration of the risks involved in implementing the various elements.
- Leveraged for designing future products with similar characteristics.
Importance of Architecture

- Every embedded system has an architecture, whether it is or is not documented.
- It is a useful tool in understanding all of the major elements:
  - why each component is there
  - why the elements behave the way they do
  - how they interact
  - how they behave in the real world
- Even if the architectural structures are rough and informal, it is still better than nothing!
- Many industry popular methodologies for creating architectures (adaptable to embedded systems):
  - Rational Unified Process (RUP), Attribute Driven Design (ADD), Object Oriented Process (OOP), ...
6 Stages of Creating an Embedded Architecture

- Stage 1: Having a Solid Technical Base
- Stage 2: Understanding the Architectural Business Cycle of Embedded Systems
- Stage 3: Defining the Architectural Patterns and Reference Models
- Stage 4: Creating the Architectural Structures
- Stage 5: Documenting the Architecture
- Stage 6: Analyzing & Evaluating the Architecture
Stage 1: Having a Solid Technical Base

Embedded Systems Model

- Application Layer
- System Software Layer
- Hardware Layer
Know Your Standards (1/3)

- Standards dictate:
  - how the components should be designed
  - what additional components are required in the system to allow for their successful integration and function

- Can be classified as:
  - market-specific standards
  - general-purpose standards, or
  - standards that are applicable to both categories
Market Specific

- Consumer Electronics
- Medical
- Industrial Automation & Control
- Networking & Communications
- Automotive
- Aerospace & Defense
- Office Automation, ...
General Purpose

- Networking
- Programming Language
- Security
- Quality Assurance, ...
### Hardware Layer: Many Many Many Embedded Processors To Choose From

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Processor</th>
<th>Main Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD</td>
<td>Au1xxx</td>
<td>Advanced Micro Devices</td>
</tr>
<tr>
<td>ARM</td>
<td>ARM7, ARM9, ...</td>
<td>ARM</td>
</tr>
<tr>
<td>ColdFire</td>
<td>5282, 5272, 5307, 5407, ...</td>
<td>Motorola</td>
</tr>
<tr>
<td>M Core</td>
<td>MMC2113, MMC2114, ...</td>
<td>Motorola</td>
</tr>
<tr>
<td>MIPS32</td>
<td>R3K, R4K, 5K, 16, ...</td>
<td>MTI4kx, IDT, MIPS Technologies</td>
</tr>
<tr>
<td>NEC</td>
<td>Vr55xx, Vr54xx, Vr41xx</td>
<td>NEC Corporation</td>
</tr>
<tr>
<td>PowerPC (PPC)</td>
<td>82xx, 74xx, 8xx, 7xx, 6xx, 5xx, 4xx</td>
<td>IBM, Motorola</td>
</tr>
<tr>
<td>68k</td>
<td>680x0, 683xx</td>
<td>Motorola</td>
</tr>
<tr>
<td>SuperH (SH)</td>
<td>SH3 (7702, 7707, 7708, 7709), SH4 (7750)</td>
<td>Hitachi</td>
</tr>
<tr>
<td>SHARC</td>
<td>SHARC</td>
<td>Analog Devices, Transtech DSP, Radstone</td>
</tr>
<tr>
<td>strongARM</td>
<td>strongARM</td>
<td>Intel</td>
</tr>
<tr>
<td>SPARC</td>
<td>UltraSPARC II</td>
<td>Sun Microsystems</td>
</tr>
<tr>
<td>TMS320C6xxx</td>
<td>TMS320C6xxx</td>
<td>Texas Instruments</td>
</tr>
<tr>
<td>x86</td>
<td>X86 [386, 486, Pentium (II, III, IV), ...]</td>
<td>Intel, Transmeta, National Semiconductor, Atlas, ...</td>
</tr>
<tr>
<td>TriCore</td>
<td>TriCore1, TriCore2, ...</td>
<td>Infineon</td>
</tr>
</tbody>
</table>

...
ISA Models

- Application Specific
  - Controller
  - Datapath
  - Finite State Machine with Datapath [FSMD]
  - Java Virtual Machine
  - ...

- General Purpose
  - Complex Instruction Set Computing [CISC]
  - Reduced Instruction Set Computing [RISC]

- Instruction Level Parallelism
  - Single Instruction Multiple Data [SIMD]
  - Superscalar Machine
  - Very Long Instruction Word (VLIW) Computing
  - ...

José Costa (DEI/IST)  Architecture of Embedded Systems
Implementing an ISA & Von-Neumann Architecture of Embedded Systems

controls usage and manipulation of data

data from cpu or input devices stored in memory until a cpu or output device request

brings data into the embedded system

system components commonly connected via buses

gets data out of the embedded system
Embedded Software: The System Software Layer (1/3)
Embedded Software: The System Software Layer (2/3)

Application Software

System Software

Operating System

Board Support

Device Driver

Hardware

Application Software

System Software

Middleware

Operating System

Device Driver

Hardware

Application Software

System Software

Middleware

Operating System

Device Driver

Hardware
Embedded Software: The System Software Layer (3/3)
Device drivers are **architecture-specific** or **generic**

### Architecture-specific device drivers
- Manages hardware integrated in the processor
- E.g. initialization and enable of on-chip memory, floating point hardware, ...

### Generic device drivers
- Manages hardware on the board
- E.g. initialization and enable of off-chip memory, board buses, off-chip I/O, ...
What are Device Drivers? (2/2)
Most Common Types of Device Drivers Routines

- **Hardware Startup** - initialization of the hardware upon power-on or reset
- **Hardware Shutdown** - configuring hardware into its power-off state
- **Hardware Disable** - allowing other software to disable hardware on-the-fly
- **Hardware Enable** - allowing other software to enable hardware on-the-fly
- **Hardware Acquire** - allowing other software gain singular (locking) access to hardware
- **Hardware Release** - allowing other software to free (unlock) hardware
- **Hardware Read** - allowing other software to read data from hardware
- **Hardware Write** - allowing other software to write data to hardware
- **Hardware Install** - allowing other software to install new hardware on-the-fly
- **Hardware Uninstall** - allowing other software to remove installed hardware on-the-fly
Embedded Operating Systems

- Process Management
  - Process Implementation
  - Scheduling
  - Intertask Communication & Synchronization
  - ...

- Memory Management
  - Segmentation
  - Paging
  - Virtual Memory
  - System Security
  - ...

- I/O System Management
  - File System
## Evolution of Programming Languages

<table>
<thead>
<tr>
<th>Generation</th>
<th>Language</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>Machine code</td>
<td>Binary (0,1) and hardware dependent.</td>
</tr>
<tr>
<td>2nd</td>
<td>Assembly language</td>
<td>Hardware-dependent representing corresponding binary machine code.</td>
</tr>
<tr>
<td>3rd</td>
<td>HOL (high-order languages)/procedural languages</td>
<td>High-level languages with more English-like phrases and more transportable, such as C, Pascal, etc.</td>
</tr>
<tr>
<td>4th</td>
<td>VHLL (very high level languages)/non-procedural languages</td>
<td>“Very” high-level languages: object-oriented languages (C++, Java,...), database query languages (SQL), etc.</td>
</tr>
<tr>
<td>5th</td>
<td>Natural languages</td>
<td>Programming similar to conversational languages, typically used in artificial intelligence (AI). Still in the research and development phases in most cases—not yet applicable in mainstream embedded systems.</td>
</tr>
</tbody>
</table>
Translation of Code on Host (1/2)

Host (Development System)
- Preprocessor
- Compiler
- Linker
- System Software Layer
- Hardware Layer

Target (Embedded System)
- Application Layer
- System Software Layer
- Hardware Layer

José Costa (DEI/IST)
Architecture of Embedded Systems
Translation of Code on Host (2/2)

- C SourceFile(s)
- C Compiler
  - Preprocessing
  - Compiling
    - C Object File(s)
      - Linker
        - C Executable File
          - Host Computer
            - Embedded System
Translation of Code on Target (1/2)

Interpretation

- Byte Code 1 Parsing
- Byte Code 1 Interpreting
- Byte Code 2 Parsing
- Byte Code 2 Interpreting
- ...

Host

- Source File(s)
- Byte Code File(s)
- Source L1
- Source L2
- Source L3
- Source L4
- Source L5
- Source L6
- ...

Just in Time (JIT)

- Byte Code 1 Parsing and Interpreting
- Byte Code 1 JIT Compiling
- Byte Code 2 Parsing and Interpreting
- Byte Code 2 JIT Compiling
- ...

Way-Ahead-of-Time /Ahead-of-Time (WAT/AOT)

- .class File
  - Byte Code 1
  - Compiled Byte Code 1
  - Byte Code 2
  - Compiled Byte Code 1
  - Byte Code 3
  - Compiled Byte Code 1
  - ...
- JVM WAT Compiler
- object File
- JVM Linker
- Runtime Libraries
- executables

1st Pass of Processing Byte Code

2nd and Additional Passes of Processing Byte Code
Translations of Code on Target (2/2)

- Translating Code
  - Interpretation
  - Just-in-Time (JIT)
  - Way-Ahead-of-Time/Ahead-of-Time (WAT/AOT)

- Garbage Collection
  - Copying
  - Mark & Sweep
  - Generational
Garbage Collection: Copying

Memory Before GC
- Object 1
- Object 2
- Object 3
- Object 4

Copying Garbage Collector

Memory After GC
- Object 1
- Object 2
- Object 4
Garbage Collection: Mark & Sweep

Memory Before GC

- Object 1 ✓
- Object 2 ✓
- Object 3
- Object 4 ✓

Mark & Sweep Garbage Collector

Memory After GC

- Object 1
- Object 2
- Object 4

Mark

Sweep
Garbage Collection: Generational

Youngest Generation (Nursery)

Copying Garbage Collector

Memory Before GC
Object 1
Object 2
Object 3
Object 4

Memory After GC

Object 1
Object 2
Object 4

Mark & Sweep

Older Generation

Mark, Sweep & Compact Garbage Collector

Memory Before GC
Object 1
Object 2
Object 3
Object 4

Memory After GC

Object 1 ✓
Object 2 ✓
Object 4 ✓

Mark

Sweep

Compaction

Memory After GC

Object 1
Object 2
Object 4

Mark, Sweep & Compact

Garbage Collector

Mark

Sweep

Compaction

José Costa (DEI/IST) Architecture of Embedded Systems
How can Java Add to An Embedded System’s Architecture?

- Standards (NanoVM, JStamp, MicroEJ, Java ME, ...)
- Processing Bytecode ( Interpretation, JIT, WAT/AOT )
- Garbage Collection ( Copying, Mark&Sweep, ... )
Scripting Languages

- Perl, JavaScript, HTML, ...
- Processing Bytecode (Interpretation)
Stage 2: Understanding the ABCs
(Architecture Business Cycles)

System Stakeholders
- Customers
- Engineers
- Managers
- ...

Technical Requirements
Quality Assurance Requirements
Industry Standard Requirements
Marketing Requirements

Architecture
Embedded System
Stage 2: Understanding the ABCs (Architecture Business Cycles)

1. ABC influences drive the requirements of an embedded system
   - not limited to technical ones.

2. Identify all the ABC influences on the design
   - technical, business, political and/or social

3. Engaging the various influences as early as possible in the design and development lifecycle and gathering the requirements of the system

4. Determining the possible hardware and/or software elements that would meet the gathered requirements.
Stage 3: Defining the Architectural Patterns & Reference Models

- Determine the components that meet
  - deadlines,
  - time-to-market,
  - cost, ...

- Select a programming language

- Select a OS

- Select a master processor
Stage 4: Define the Architectural Structures

- Decomposing the structures into smaller and smaller elements

- These decompositions are represented as some combination of various types of elements

- It is for the architects to decide which structures to select and how many to implement
Stage 5: Document the Architecture

- At least two documents
- A document outlining the entire architecture
  - an overview of the embedded system
  - the actual requirements supported by the architecture
  - the definitions of the various structures
  - the inter-relationships between the structures
  - ...

- A document for each structure
  - which requirements are being supported by the structure
  - how these requirements are being supported by the design
  - any relative constraints, issues, or open items
  - representation of each of the various elements within the structure
Stage 6: Analyze and Evaluate Architecture

- Architecture is analysed by an evaluation team.

- Architects and evaluation team agree on the different scenarios for the architecture.

- Results of the evaluation should be produced:
  - list of requirements/scenarios
  - return on investment (ROI)
  - risks
  - strengths
  - problems
  - recommended changes to the architecture
Outline

- Embedded System Design and Development Process
- 6 Stages of Creating an Embedded Architecture
  - Solid Technical Base
  - Architecture Business Cycles
  - Architectural Patterns & Reference Models
  - Architectural Structures
  - Document the Architecture
  - Analyze and Evaluate Architecture
References

Next class

• The Embedded Computing Platform: Input/Output Interfaces