Overview and Introduction to Embedded Systems

(Module 1)

Vertically Integrated Projects (VIP) Program
Contents

Overview
- What is an Embedded System?
- Application examples
- Key characteristics
- Recent trends
- Embedded System Designer
- Role of the Design Team

Software
- Compilers and Languages
- System Development
  - Debugging
  - Resource scarcity
  - Approach principles
- System Architecture
  - System sketches (diagrams)
  - From diagrams to architecture
  - The Model-View-Controller (MVC) pattern

Hardware
- Examples
- Datasheets
- Schematics
- Debugging tools

H/S Integration
- System development
- Dealing with errors
- Not language-specific
- Points to many other good references.
- Includes interview-type questions

- Available online
Overview
What Is An Embedded System?

In your own words…

- A system where different things are brought together to perform a particular application
- An electronic device with computing capability, but whose main purpose isn’t computing (i.e. cellphone, appliance,...., not a laptop)
- Combination of h/w and s/w designed for a specific set of purposes (as opposed to a PC which can be programmed to close to anything)
- A system that contains a micro computer controller
- A computerized system that operates under resource constraints
- A miniature computation system developed for low power, high performance devices
What Is An Embedded System?

- Is a computerized system that is purpose-built for its application.

Elecia White
Making Embedded Systems
Application Examples

- Parking aid (in boot)
- Climate control (behind dashboard)
- Alarm & immobiliser (behind glovebox)
- Engine management system (under bonnet)
- Automatic wiper control (under bonnet)
- Airbag control unit (behind central console)
- Electric window & central locking (in footwell)
- Anti-lock braking system (under cover)
- Automatic transmission (under cover)
Modern Cars

- Use ~100 processors
- Complex software for
  - Engine & emissions control
  - Stability & traction control
  - Diagnostics
  - Gearless automatic transmission

Qorivva MPC560xP MCU family (32-bit)
For Chassis and Safety Applications

http://www.howstuffworks.com/car-computer.htm
CoolBio ultra-low power biomedical signal processor
- 0.01x mW/MOPS
- Less than 1 mA @ 1 V
- Less than 10 mm² of Si

Kochkin’s 2008 survey
Americans with hearing impairment:
- 35 million = 11.3% of population
- > 40 million by 2025
Application Examples

Tongue Drive System

Prof. Maysam Ghovanloo, Georgia Tech

Samsung S3C2410
- 16/32-bit ARM920T processor.
- Clocked up to 203MHz
- Instruction and Data: 16KB each
Application Examples

Atmel
#AT91SAM7S256
32-bit ARM7TDMI RISC processor w/ 256-kbyte flash

Atmel
#AT45DB321D
4-Mbyte flash memory

Fairchild
#FAN8100N
two-channel motor driver

NXP
#LPC2103F
32-bit ARM7TDMI RISC processor w/ 10-bit A/D converters

Toshiba
#TMP86FH47AUG
8-bit microcontroller (one of four total in Pleo)
Application Examples
Key Characteristics

- Not a personal computer
- Real-time processing
  - Reactive to changes in the environment
- Never terminates the program
- Not general purpose – specific
  - Application known a priori
- A computing device of a larger system
- Integrated with sensors and actuators (cyberphysical)
- Interacts with the external world
- Its operation is time-constrained
- Increasingly high performance and networked
Recent Trends

- Multimedia demands increasing computation
  - E.g. HDTV, cellphones, mp3 players, tablets

- Low power demand enables higher efficiency
  - Reducing current consumption in devices (e.g. FinFET’s)
  - Idle time becomes more important than active

- Energy harvesting alternatives are critical
  - Could the ear generate energy to power a cochlear implant?

- Trend enables novel applications
  - Computing
  - Communications
  - Sensors
  - Controls

- Devices are increasingly networked
  - Cars with web servers
  - Buildings with networked environmental control

- Increasing need for flexibility and modularity
  - Reduce time-to-market under ever changing standards
What is an Embedded System Designer?
Role of the Design Team

- Interdisciplinary learning
  - Hardware and software skill sets must be integrated

- Diverse background in team members and teamwork make the job in embedded systems easier

- Team skills need to include the ability to:
  - Read a datasheet
  - Understand the components of a new processor
  - Get to know a new processor
  - Go through schematics
  - Put together a debugging toolbox
  - Test hardware (and software)
Software
Compilers and Languages

- Embedded systems use cross compilers
  - Creates code that can run on the specified target platform
  - Larger processors make use of Unix-compatible cross compilers

- Embedded software compiler’s languages
  - C, or C/C++ (only a subset of C++)
  - Java may become popular, but only works on systems with larger memory storage capacity
Resource Scarcity in Embedded Systems

- Memory (RAM)
- Code space (ROM or Flash)
  - May be traded for processor cycles, more space but faster
- Processor cycles or speed
  - Tradable for battery life, i.e. lower power consumption
- Power consumption (battery life)
  - Usually a design driver in stand-alone applications
- Processor peripherals
  - May be created using I/O lines and processor cycles
Additional Challenges

- Some “bugs” during the debugging process are caused by resource scarcity

- Other are only expressed during board-bring up
  - Introduces uncertainty on sources of the bugs
    - Is the bug a problem on hardware or software?
  - Bugs may damage hardware – application specific
    - Requires paying attention to details and learning fast
    - Same challenges found in one system may not apply to a different system

- Consider the function of the final product
  - Bugs may result in catastrophes
    - Consider aviation, medicine, or other critical fields of application
Some challenges may be overcome by making use of the following principles:

- **Flexibility**
  - Allows to introduce changes in system design adapting to constraints found in different hardware configurations
  - Employs modularity and encapsulation to define functional software elements

- **Modularity**
  - Separates the functionality of a system into subsystems
  - Hides the data used by subsystems and defines classes of objects
    - Such is the case in object-oriented programming
    - Enables code changes with minimal or no impact to other modules

- **Encapsulation**
  - Establishes the interfaces (inputs, outputs, properties) of modules
  - Isolates software elements
    - In object-oriented design it defines classes
Isolates the GUI center of the application from the user interface for independent testing

- **The Model**
  - Contains the domain-specific data and logic
- **The View**
  - Is the interface to the user (input and output)
- **The Controller**
  - Bridges the Model and the View

For example:
- The View-Controller modules may allow to exchange displays and inputs (e.g. keyboard and screen in a PC for a touchscreen in a tablet)
Example: The MVC Pattern

- **Controller**: Translate user input from view into model actions.
- **View**: (User interface) Shows representation of model to user.
- **Model**: Application-specific state data and algorithms.

User action (i.e. button press) leads to change state and take action. Change display leads to change notification and request data and state.

- **Shown here**
- **Model receives data only from controller**
- **Controller is translator**
- **Model-View pattern**
Example: The MVC Pattern

- Audio illustration
Hardware
Hardware Examples

☑ DE2i-150 FPGA Development Kit
☑ Snapdragon™ S3-based Dragonboard™

☑ Arduino R3 SMD
☑ Raspberry Pi Model B
☑ Beagleboard
Arduino R3 SMD

- Microcontroller: ATmega328
  - Maximum operating frequency = 20 MHz

- Memory
  - Flash Memory: 32 KB (ATmega328)
    - 0.5 KB used by bootloader
  - SRAM: 2 KB (ATmega328)
  - EEPROM: 1 KB (ATmega328)

- Operating Voltage: 5V

- Input Voltage: 7-12V

- Input Voltage (limits): 6-20V

- Digital I/O Pins: 14 (6 provide PWM output)

- Analog Input Pins: 6

- DC Current per I/O Pin: 40 mA

- DC Current for 3.3V Pin: 50 mA

- Clock Speed 16 MHz
DE2i-150 FPGA Development Kit

- Processor: Intel Atom N2600
- FPGA: Altera Cyclone IV GX
- Intel® Chipset NM10
- Audio Input & Output
- HDMI 1.3a
- VGA
- PCIe Mini Card (Half-Size)
- mSATA Card (Full-Size)
- USB 2.0 Host x4
- 10/100/1000 M Ethernet
- SATA Gen2
- DDR3 SO-DIMM Socket
- VGA Display, TV Decoder (Composite Input)
- Gigabit Ethernet
- SD Card Socket
- IR Receiver, RS232
- Accelerometer
- HSMC & GPIO Expansion Connector
- EEPROM, Flash (64 MB), SSRAM (2 MB), SDRAM (64 MB x2), and EPCS64 (for FPGA Configure)
- Two PCIe x1 (Connected to Intel Atom)
- On board Oscillator and SMAx2 for External Clock Input & Output
- LED, 2x16 LCD, Button, Switch & 7-Segment
- On-board USB Blaster
Snapdragon™ S3-based Dragonboard™

- APQ8060 dual core processor
- Adreno 220 Graphics
- 1500 mAH battery
- 3.61” WVGA Display
  - Cap Sense Multi-touch screen
- 5MP main camera
- 2MP camera for video telephony
- BT/WiFi expansion card
- Sensors expansion card
  - Pressure and temperature
  - 3-axix accelerometer
  - 3-axis gyro
  - Proximity and ambient light
  - 3-axis compass
Snapdragon™ S3-based Dragonboard™

- Snapdragon S3 APQ8060
- Radio Card: WLAN, Bluetooth
- Sensor Board: Pressure, Temperature, Proximity, Ambient Light, 3-axis, Accelerometer, Gyro, Compass
- RS 232 socket
- Ethernet
- 5M camera
- 1M camera
- Touch screen Display
- Keyboard
- JTAG x2 pin socket
- Micro SD slots
- Mini USB Port
- 3.5 mm Audio Jack
Datasheets

Sections to explore

- First: driver-useful information
  - Operation information
  - Initialization
  - Communication
  - Timing diagrams
    - Describe digital states
    - Show transition relationships
    - Start on left hand side
    - Time progresses from left to right

- Next: Other sections
  - Find example applications
    (may give hints on implementations)
Timing Diagrams

SN74HC595
Shift Register (8-bit)

NOTE: **///** implies that the output is in 3-State mode.
Represent devices and their connections

Include

- Chips
  - Microcontrollers
  - Processors
  - Peripherals
- Circuit elements
  - Passive: resistors, capacitors, etc.
  - Active: inverters, op-amps, etc.
- Logical components
  - And, or, not, nand, nor
- Connections
  - Power, ground, wiring, pull-up, etc.
Schematics - Example

- Arduino Uno ATmega8
Equip your station with

- **Handtools**
  - Needle-nose pliers
  - Tweezers
    - Include mini-pliers
  - Screwdrivers
  - Box cutter
- **Measurement devices**
  - Oscilloscope
  - Digital multimeter
- **Vision support/protection**
  - Magnifying glass
  - Safety glasses
  - Flashlight
- **Miscellaneous**
  - Electrical tape
  - Sharpies
  - Cable ties
    - Velcro
    - Zip ties
Hardware-Software Integration
System Development

- Conception
- Prototyping
- Board bring-up
- Debugging
- Testing
- Release
Three different diagrams are recommended (White 2011)

- Architecture block diagram
  - Helps define software modules

- Hierarchy of control organization chart
  - Establishes relationships of modules (i.e. which module calls which other one)

- Software layering view
  - Allows to size modules by their complexity
  - Helps identify modules to be combined
Conception - Example

- Architecture block diagram

- Hardware block diagram

- Software architecture

A more detailed software architecture block diagram

- Continue adding modules as required by design elements
Hierarchy of control diagram

- “Main” defines the top level
- Lower levels are called by those higher in the hierarchy
- Helps document shared resources
  - Robustness may be compromised when sharing resources
Conception - Example

- Software layering view
  - Represents objects by their estimated size
  - Draw from the bottom, from processor
  - Facilitates grouping resources
    - Horizontally or vertically

Diagram:
- Logging
- Rendering
- Flash
- SPI
- Parallel I/O
- LCD
- Images
- Fonts
- Backlight
- PWM Out
- Generated graphics
- SN

Courses:
- Rendering
- Logging
- Images
- LCD
- Flash
- SPI
- Parallel I/O
- PWM Out
What is a prototype?
- It is a physical model of the product that is tested to validate conceptual design decisions

Objective
- To demonstrate that the concept performs the functions that satisfy the design specifications (customer needs)

It may include a succession of proof-of-concept models

It is not intended to look like the final product
- Layout, size, connections, structure, and packaging
What is *board bring-up*?
- Is the process of electrically and functionally validating hardware components in a printed circuit board assembly (PCBA)

Objective
- To power up the hardware and verify every testable component in the PCBA

How is it done?
- Taking small steps first; e.g. testing an I/O device with an LED or oscilloscope
- With *in-detail understanding* of how the processor and peripherals work
  - Reading their datasheets
Debugging

- Works different from computer programming debugging
  - For an embedded system it makes use of dedicated ports and demands system resources.

- For a cross-compiler, need a cross-debugger

- The cross-debugger
  - Makes use of a dedicated debug interface
    - Emulator
    - In-circuit emulator (ICE)
    - JTAG standard ("jay-tag")
  - Communicates with target processor
  - Makes use of processing capacity

- Limited debugging operations on processors
  - Reduces production cost
  - Maximum number of hardware breakpoints = 2

- Debugging alternative: Use \texttt{printf} (most commonly used)
Testing

- **Types of test**
  - Power-on self test (POST)
    - Verifies that all components run properly
  - Unit tests
    - May require to test all possible software paths *(time consuming!)*
    - Aims to detect all bugs **before** deployment
    - Alternative: test cases likely to occur (!)
  - Bring-up tests
    - Developed earlier for components that may not have worked as expected
    - Sometimes built upon for more comprehensive tests, or added to unit tests

- **Test software should make hardware testing easier**
  - Think about a production line

- **Proper s/w documentation**
  - Promotes better quality control
  - Facilitates s/w certification
Release

- Ends the design stage

- Should involve s/w certification
  - Expensive (!)
  - Time consuming (again, expensive)

- Delivers design data to manufacturing
  - Engineering drawings, design notebooks
  - Bill of materials
  - Software (source code, compiled files)
  - Documentation (datasheets, specs, reports)
Applications to keep in mind

- Medical
  - ICU at home for life support monitoring
- Assistive technology for
  - senior citizens
  - individuals with disabilities
- Automation in transportation systems
  - Motor vehicles
  - Aircraft
- Home-automation

Why software certification is important
Dealing with Errors

Possible sources of errors
- Written code
- Environmental conditions

Options of error handling
- “Graceful degradation”: The system does not collapse while the software does the best it can
  - Example: A long-term sensor system for data logging
- Immediate stop: The system triggers an alarm and enters safe mode
  - Example: A non-life-critical medical system with redundancy
Dealing with Errors

Some options

- `assert()`: if the argument is false (equals to zero) `abort` is called and a message is printed out to the standard error device.
- `printf()` prints a message to a system console or log.
- An LED that blinks on error conditions.
- An error handling library
  - Make each function return an error code
  - Include error functions:
    - `ErrorSet()`
    - `ErrorGet()`
    - `ErrorPrint()`
    - `ErrorClear()`
After this presentation you should know about:

- Basics
  - What is an embedded system
  - Key characteristics
  - Recent trends
  - Makeup of a design team
- Challenges for software development
  - System development and architecture
- Skills and tools needed to approach hardware
  - Reading a datasheet and schematics
  - Debugging tools
- Hardware/software integration
  - The cycle of system development
In the Next Module...

- I/O Software Interface
- Outputs
- Inputs
- Timers
- Runtime uncertainty