Georgia Institute of Technology

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
- Software (cont.)
  - 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

Bibliography

- Elecia White, *Making Embedded Systems*, O'Reilly, 2011.
  - Not language-specific
  - Points to many other good references.
  - Includes interview-type questions

- E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*, LeeSeshia.org, 2011.
  - Available online

Overview

What Is An Embedded System?

- In your own words…
  - A system where <u>different things are brought together</u> 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 <u>designed for a specific set of purposes</u> (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 <u>operates under resource constraints</u>
  - A miniature computation system <u>developed for low power, high performance devices</u>

Georgia Institute of Technology

What Is An Embedded System?

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

Elecia White
Making Embedded Systems

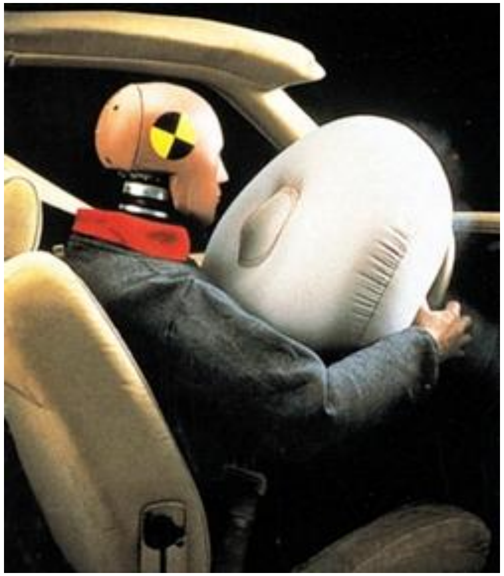
Georgia Institute of Technology

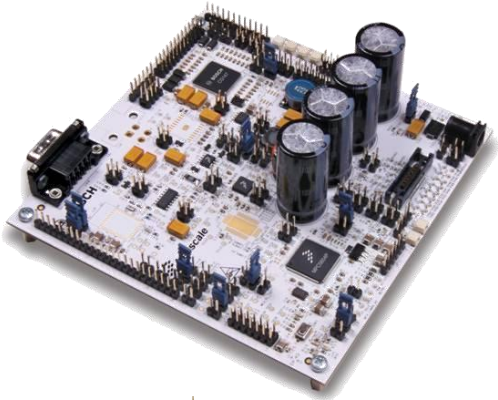
Application Examples

Georgia Institute of Technology

Application Examples

- 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

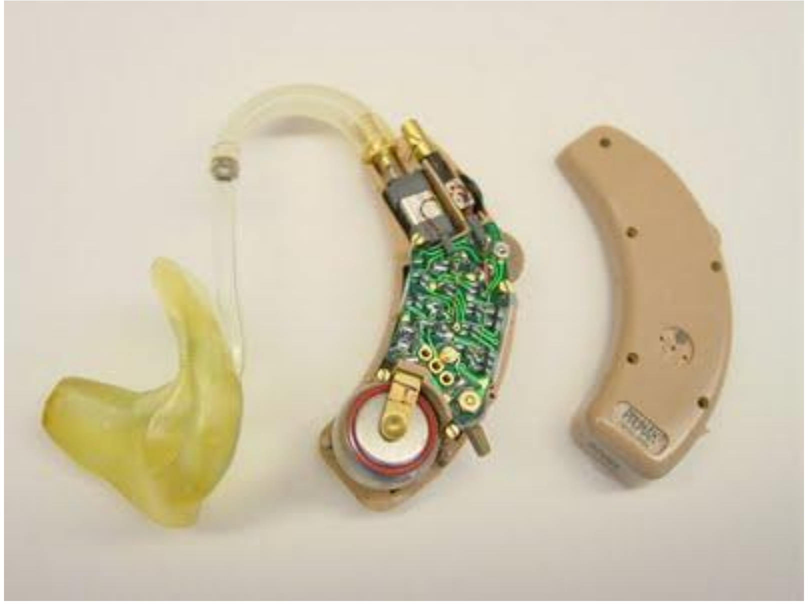
http://www.howstuffworks.com/car-computer.htm

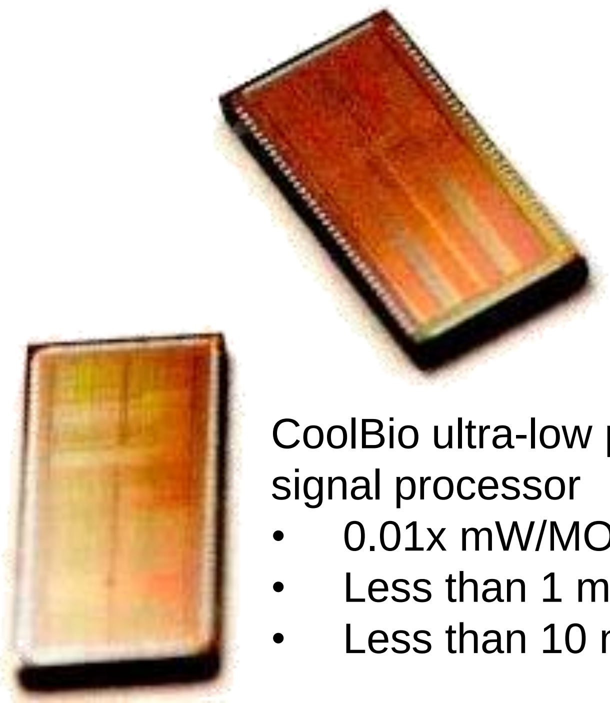
Georgia Institute of Technology

Application Examples

Kochkin's 2008 survey
Americans with hearing impairment:

- 35 million = 11.3% of population
- > 40 million by 2025





CoolBio ultra-low power biomedical signal processor

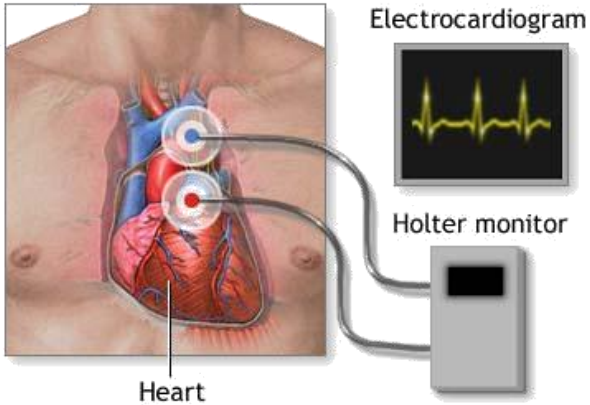
- 0.01x mW/MOPS
- Less than 1 mA @ 1 V
- Less than 10 mm^2 of Si

Georgia Institute of Technology

Application Examples

Tongue Drive System

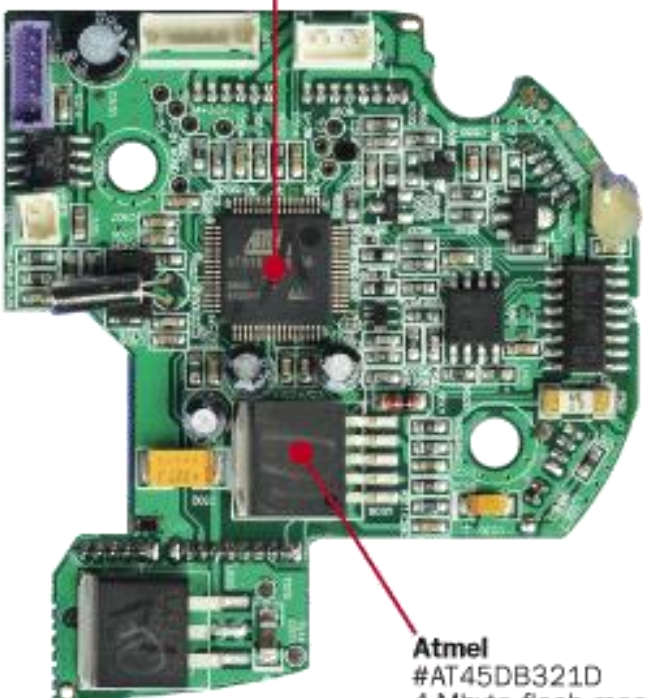
Prof. Maysam Ghovanloo, Georgia Tech



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

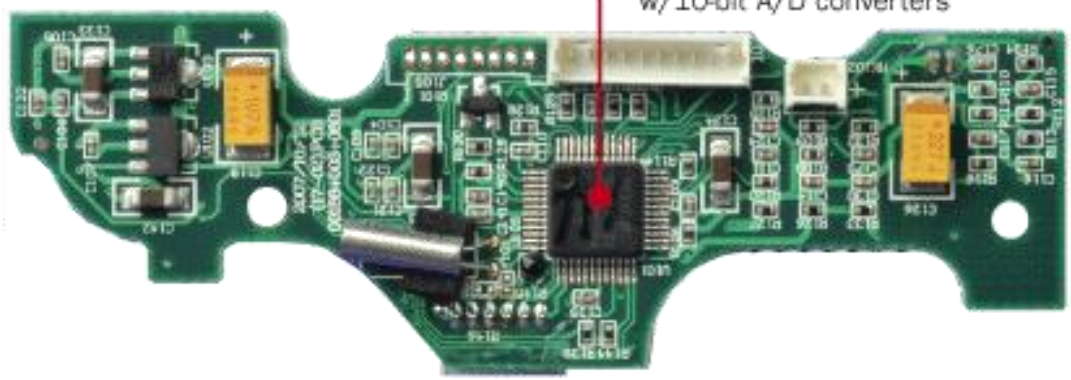
Application Examples

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

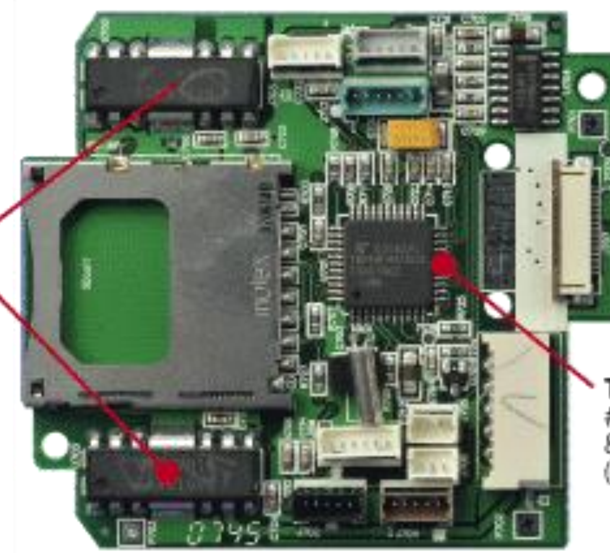


Atmel
#AT45DB321D
4-Mbyte flash memory

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



Fairchild
#FAN8100N
two-channel
motor driver



Toshiba
#TMP86FH47AUG
8-bit microcontroller
(one of four total in Pleo)

Application Examples

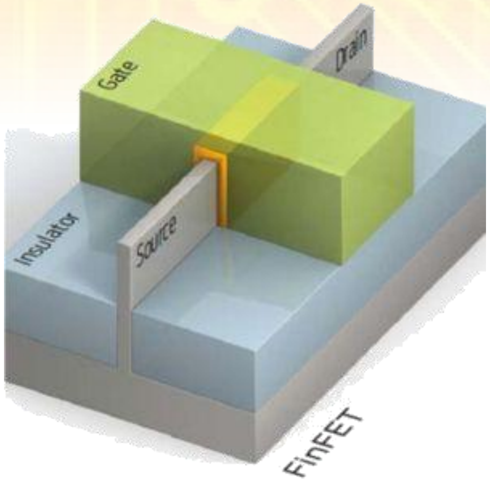
Georgia Institute of Technology

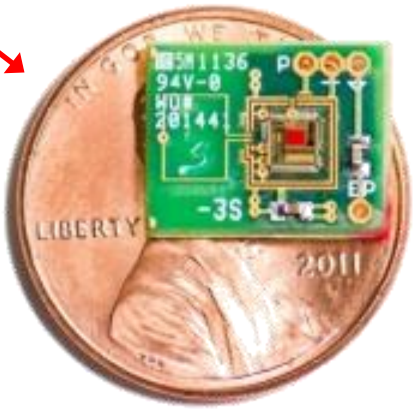
Key Characteristics

- 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

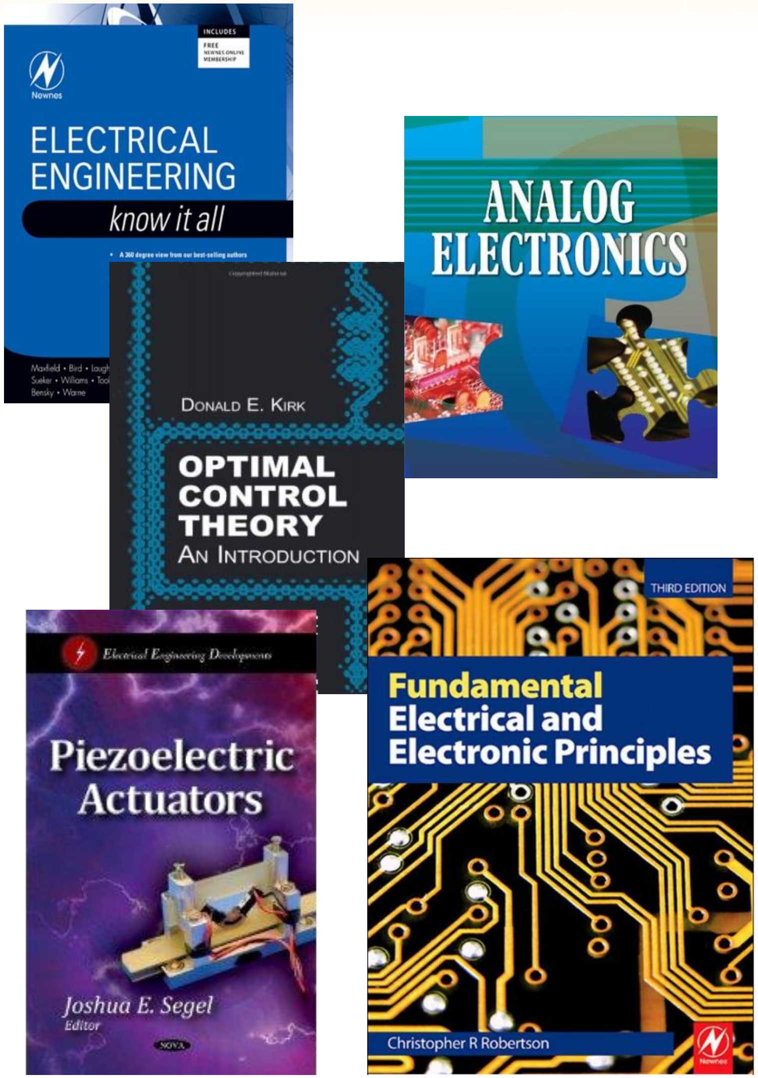




MIT-Harvard
Image: Patrick P. Mercier

Georgia Institute of Technology

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

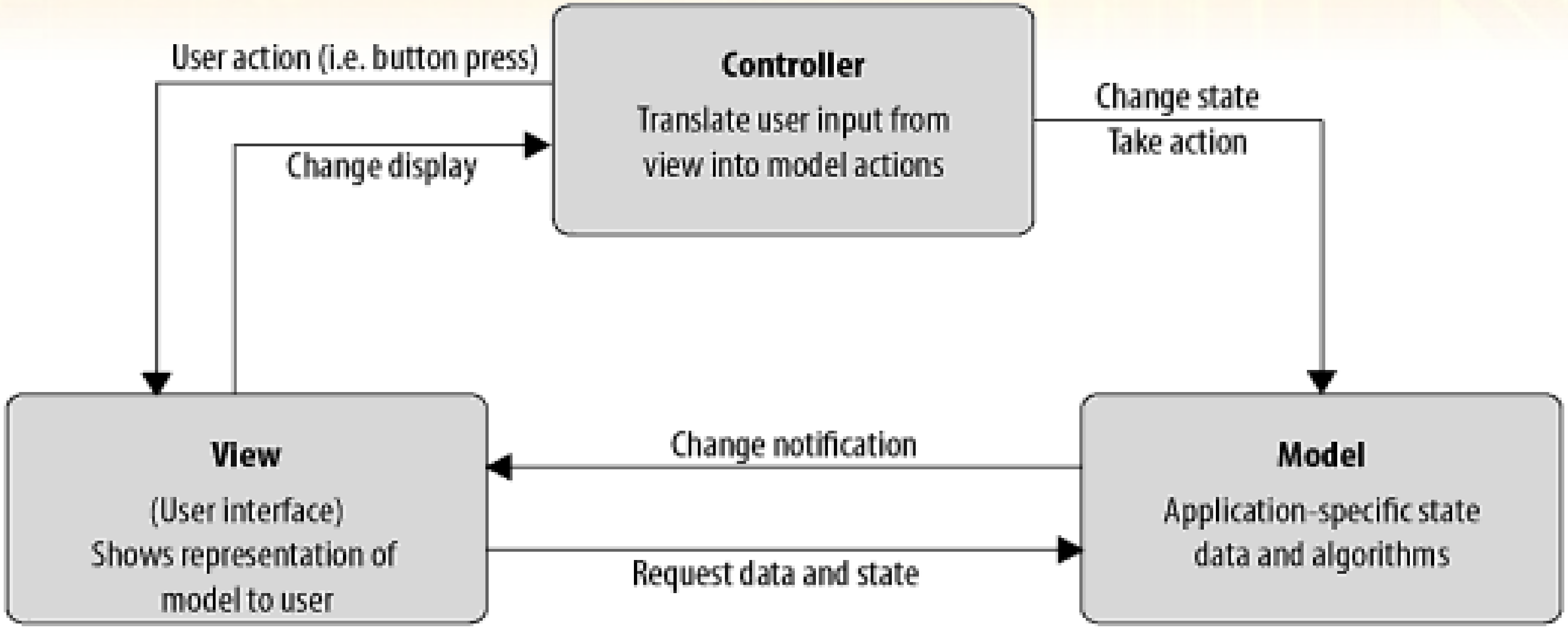
Approach Principles for S/W

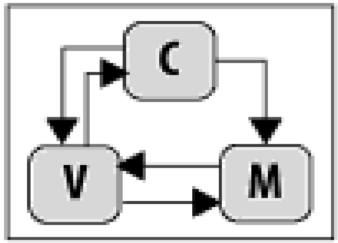
- Some challenges may be overcome by making use of the following principles
  - Flexibility
    - Allows to introduce changes in system design adapting to constraints found in different hardware configurations
    - Employs modularity and encapsulation to define functional software elements
  - Modularity
    - 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

Example: Model-View-Controller (MVC) Pattern

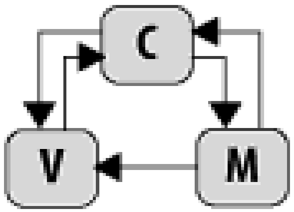
- Isolates the GUI center of the application from the user interface for independent testing
  - The Model
    - Contains the domain-specific data and logic
  - The View
    - Is the interface to the user (input and output)
  - The Controller
    - Bridges the Model and the View
- For example:
  - The 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

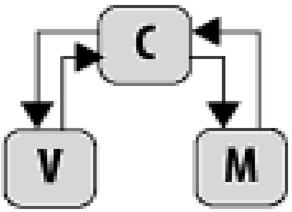




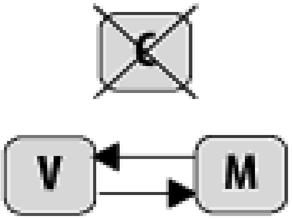
Shown here



Model receives data only from controller



Controller is translator

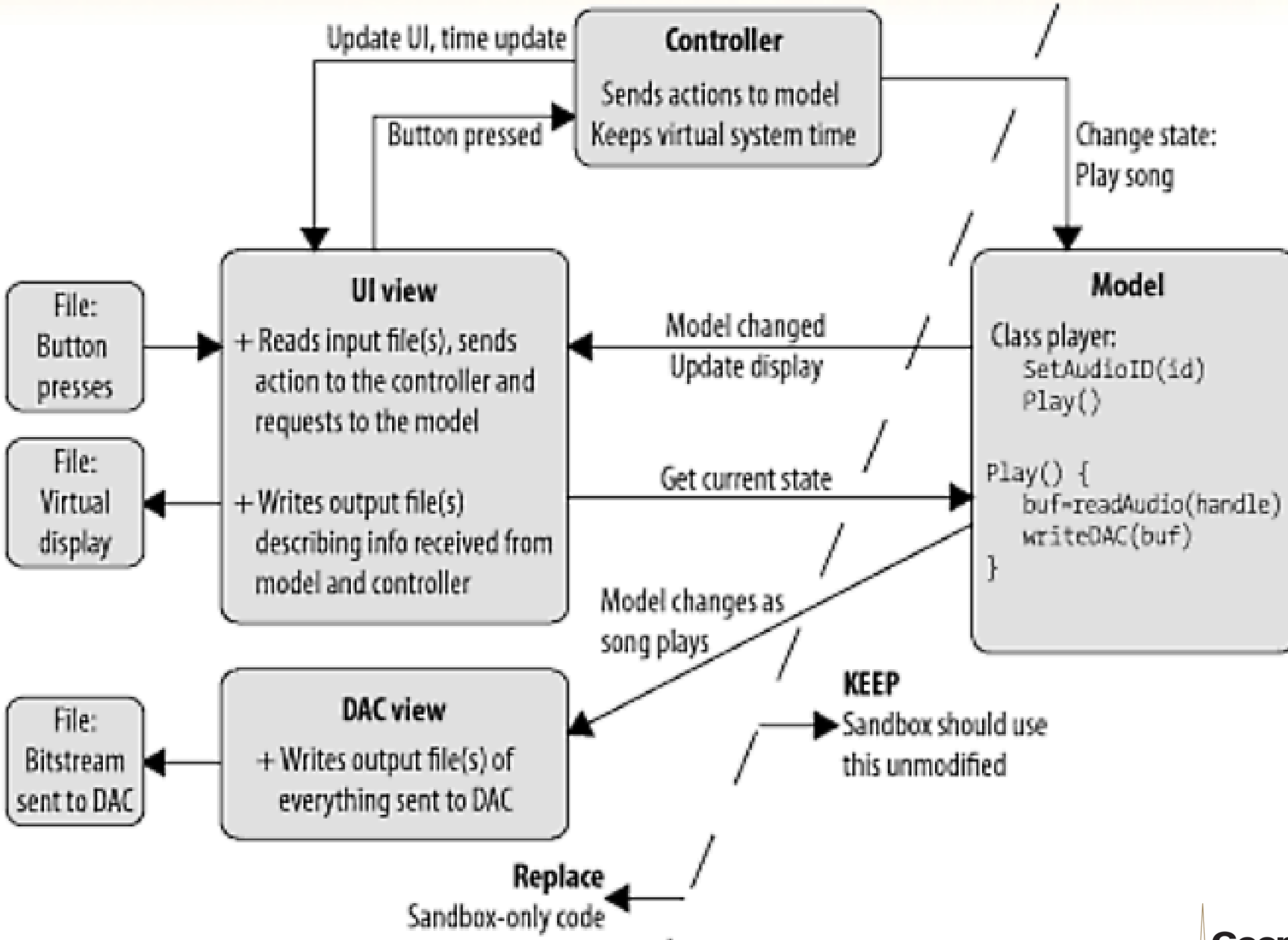


Model-View pattern

Elecia White, *Making Embedded Systems*, O'Reilly, 2011.

Example: The MVC Pattern

- Audio illustration

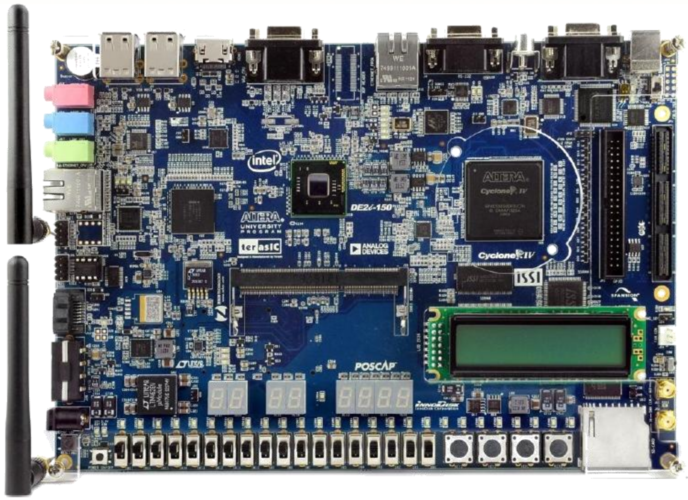


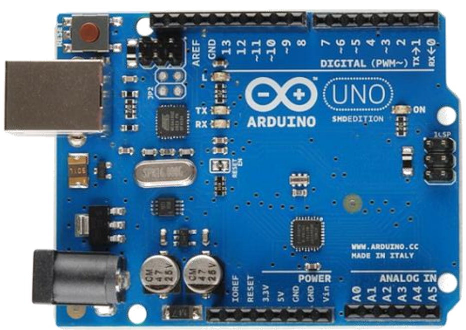
Elecia White, *Making Embedded Systems*, O'Reilly, 2011.

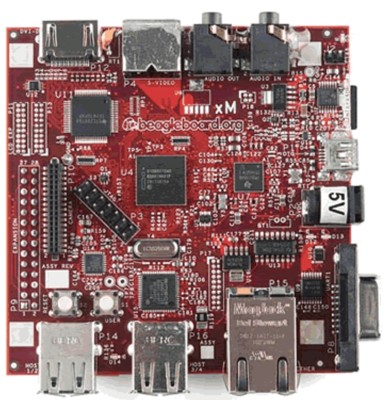
Hardware

Hardware Examples

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



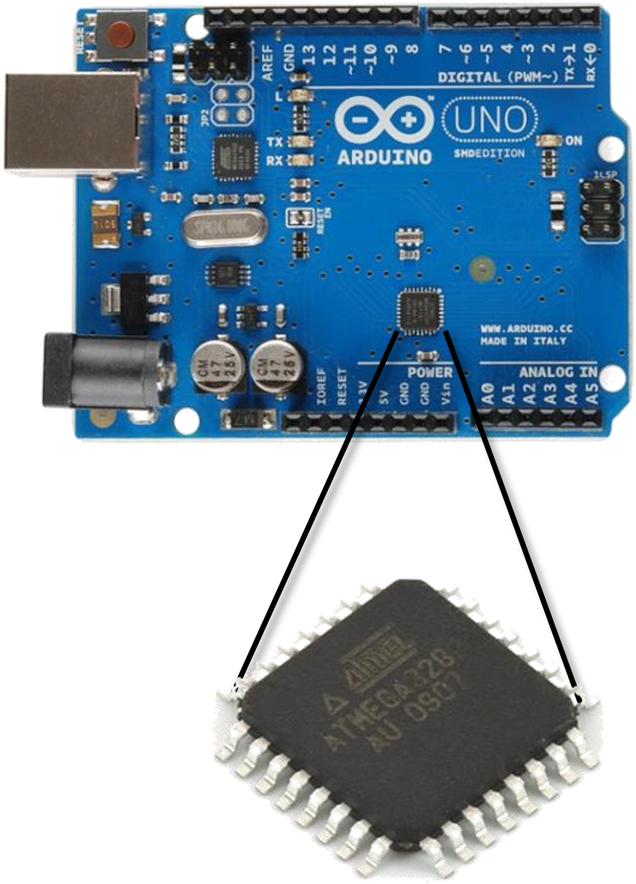




- ✔ Arduino R3 SMD
- Raspberry Pi Model B
- Beagleboard

Arduino R3 SMD

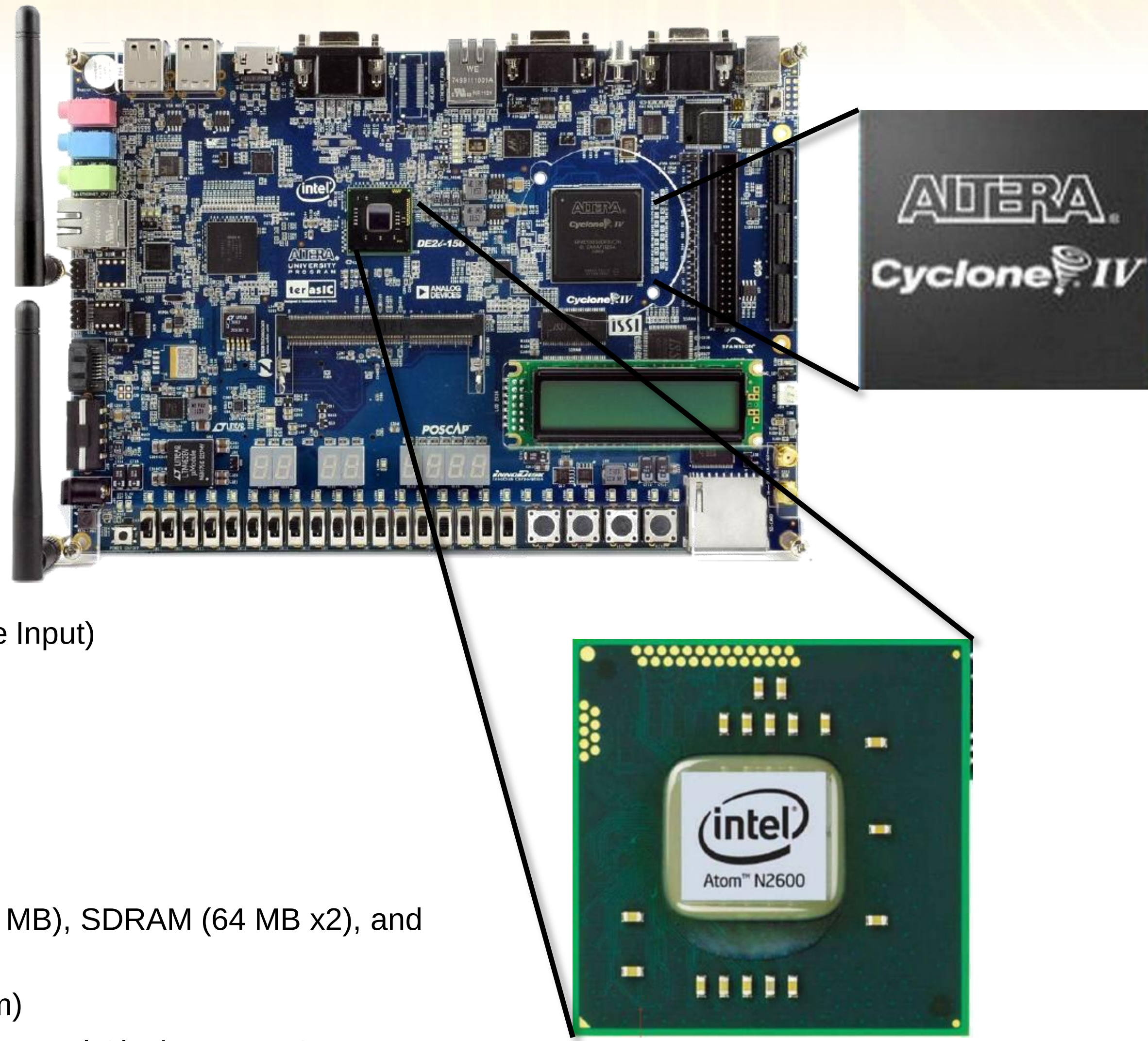
- Microcontroller: ATmega328
  - Maximum operating frequency = 20 MHz
- Memory
  - Flash Memory: 32 KB (ATmega328)
    - 0.5 KB used by bootloader
  - SRAM: 2 KB (ATmega328)
  - EEPROM: 1 KB (ATmega328)
- Operating Voltage: 5V
- Input Voltage: 7-12V
- Input Voltage (limits): 6-20V
- Digital I/O Pins: 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



Georgia Institute of Technology

DE2i-150 FPGA Development Kit

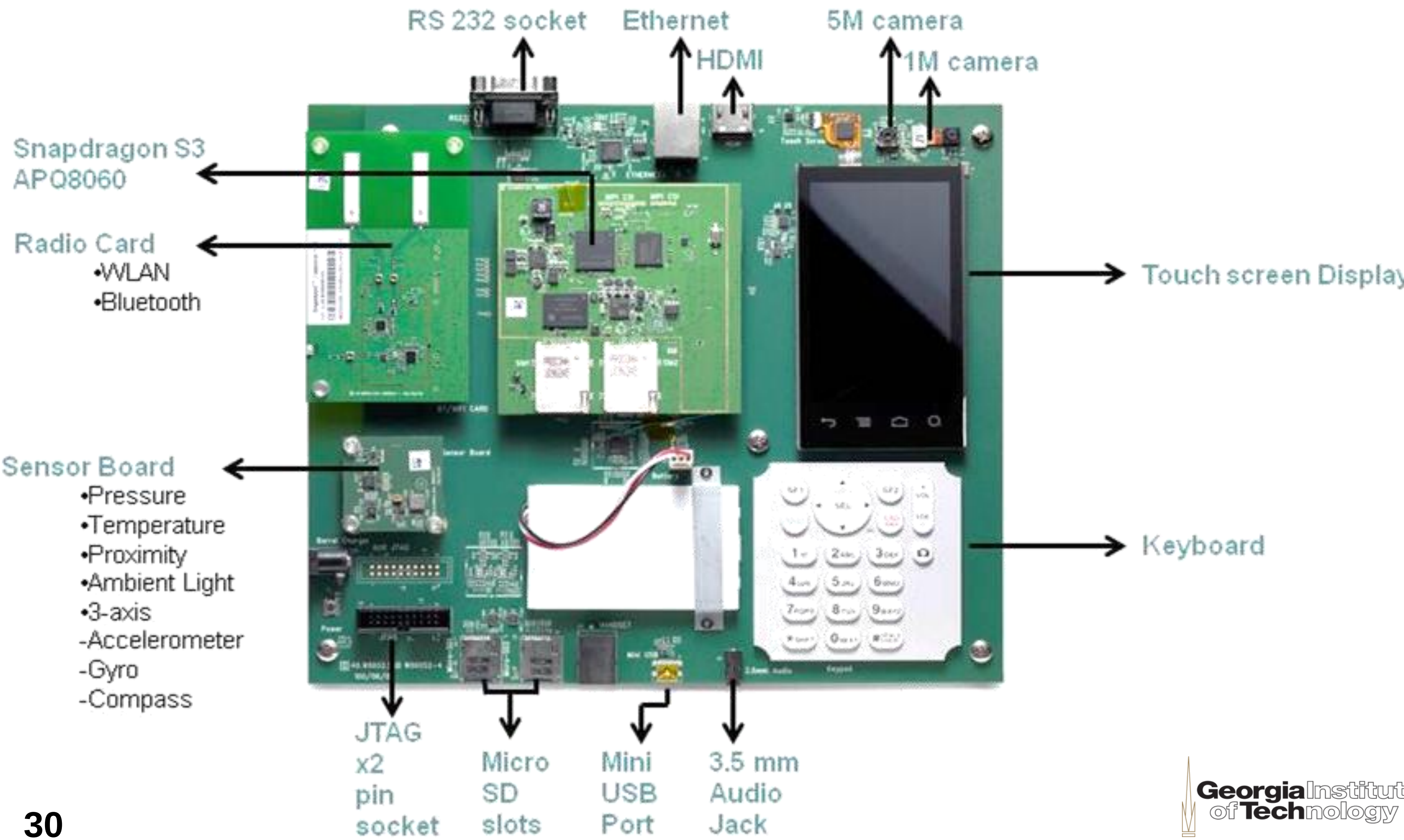
- Processor: Intel Atom N2600
- FPGA: Altera Cyclone IV GX
- Intel® Chipset NM10
- Audio Input & Output
- HDMI 1.3a
- VGA
- PCIe Mini Card (Half-Size)
- mSATA Card (Full-Size)
- USB 2.0 Host x4
- 10/100/1000 M Ethernet
- SATA Gen2
- DDR3 SO-DIMM Socket
- VGA Display, TV Decoder (Composite Input)
- Gigabit 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 telphony
- 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™



Georgia Institute of Technology

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)

13. External Interrupts

The External Interrupts are triggered by the INT0 and INT1 pins or any of the PCINT23...0 pins. Observe that, if enabled, the interrupts will trigger even if the INT0 and INT1 or PCINT23...0 pins are configured as outputs. This feature provides a way of generating a software interrupt. The pin change interrupt PCI2 will trigger if any enabled PCINT[23:16] pin toggles. The pin change interrupt PCI1 will trigger if any enabled PCINT[14:8] pin toggles. The pin change interrupt PCI0 will trigger if any enabled PCINT[7:0] pin toggles. The PCMSK2, PCMSK1 and PCMSK0 Registers control which pins contribute to the pin change interrupts. Pin change interrupts on PCINT23...0 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than Idle mode.

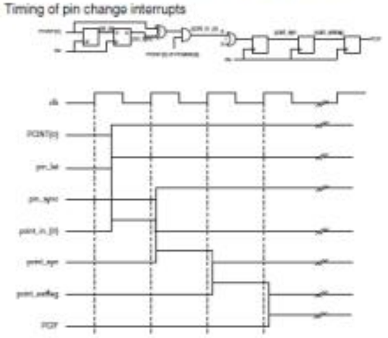
The INT0 and INT1 interrupts can be triggered by a falling or rising edge or a low level. This is set up as indicated in the specification for the External Interrupt Control Register A – EICRA. When the INT0 or INT1 interrupts are enabled and are configured as level triggered, the interrupts will trigger as long as the pin is held low. Note that recognition of falling or rising edge interrupts on INT0 or INT1 requires the presence of an I/O clock, described in "Clock Systems and their Distribution" on page 26. Low level interrupt on INT0 and INT1 is detected asynchronously. This implies that this interrupt can be used for waking the part also from sleep modes other than Idle mode. The I/O clock is halted in all sleep modes except Idle mode.

Note: Note that if a level triggered interrupt is used for wake-up from Power-down, the required level must be held long enough for the MCU to complete the wake-up to trigger the level interrupt. If the level disappears before the end of the Start-up Time, the MCU will still wake up, but no interrupt will be generated. The start-up time is defined by the SUT and CKSEL Fuses as described in "System Clock and Clock Options" on page 26.

13.1 Pin Change Interrupt Timing

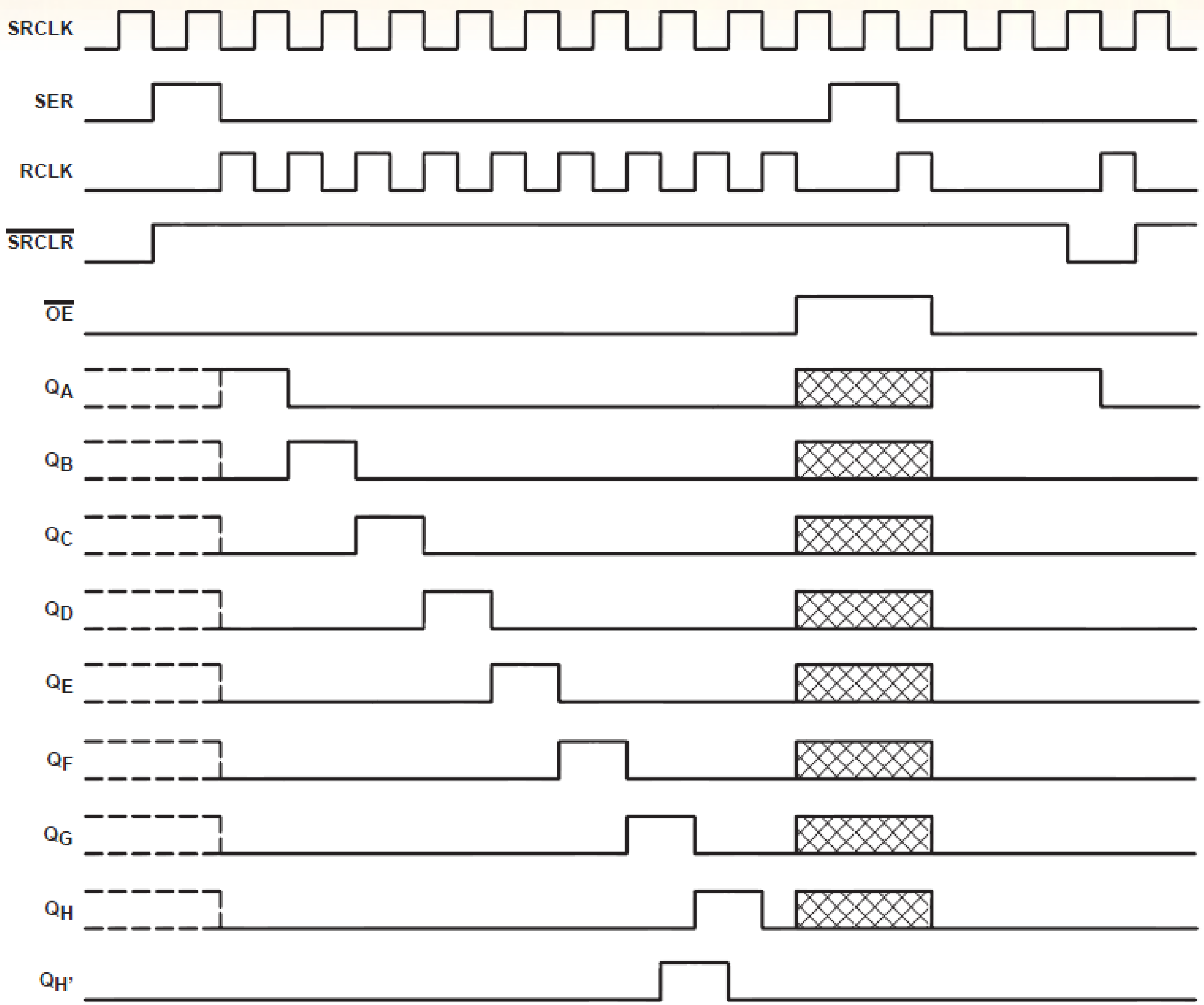
An example of timing of a pin change interrupt is shown in Figure 13-1.

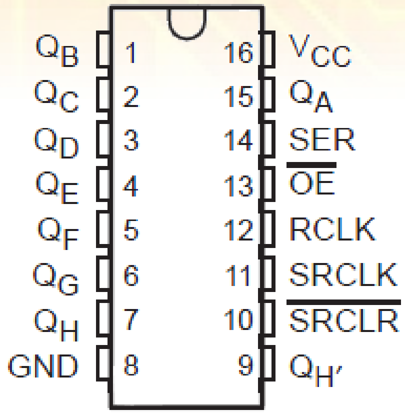
Figure 13-1. Timing of pin change interrupts



Georgia Institute of Technology

Timing Diagrams





SN74HC595
Shift Register (8-bit)

Texas Instruments

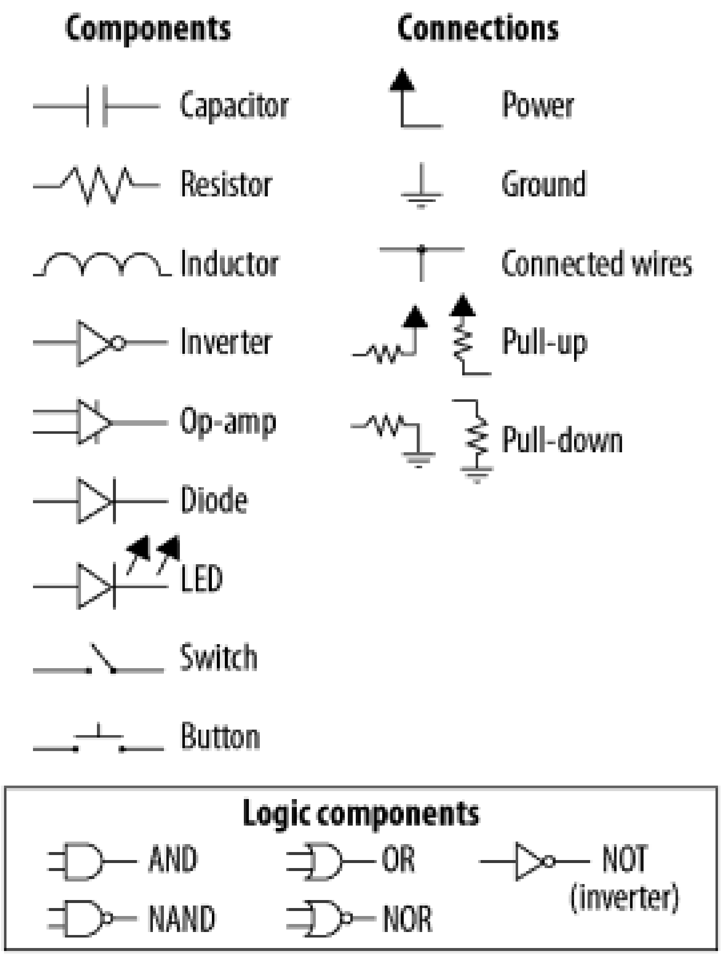
Georgia Institute of Technology

NOTE: implies that the output is in 3-State mode.

Schematics

- 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.

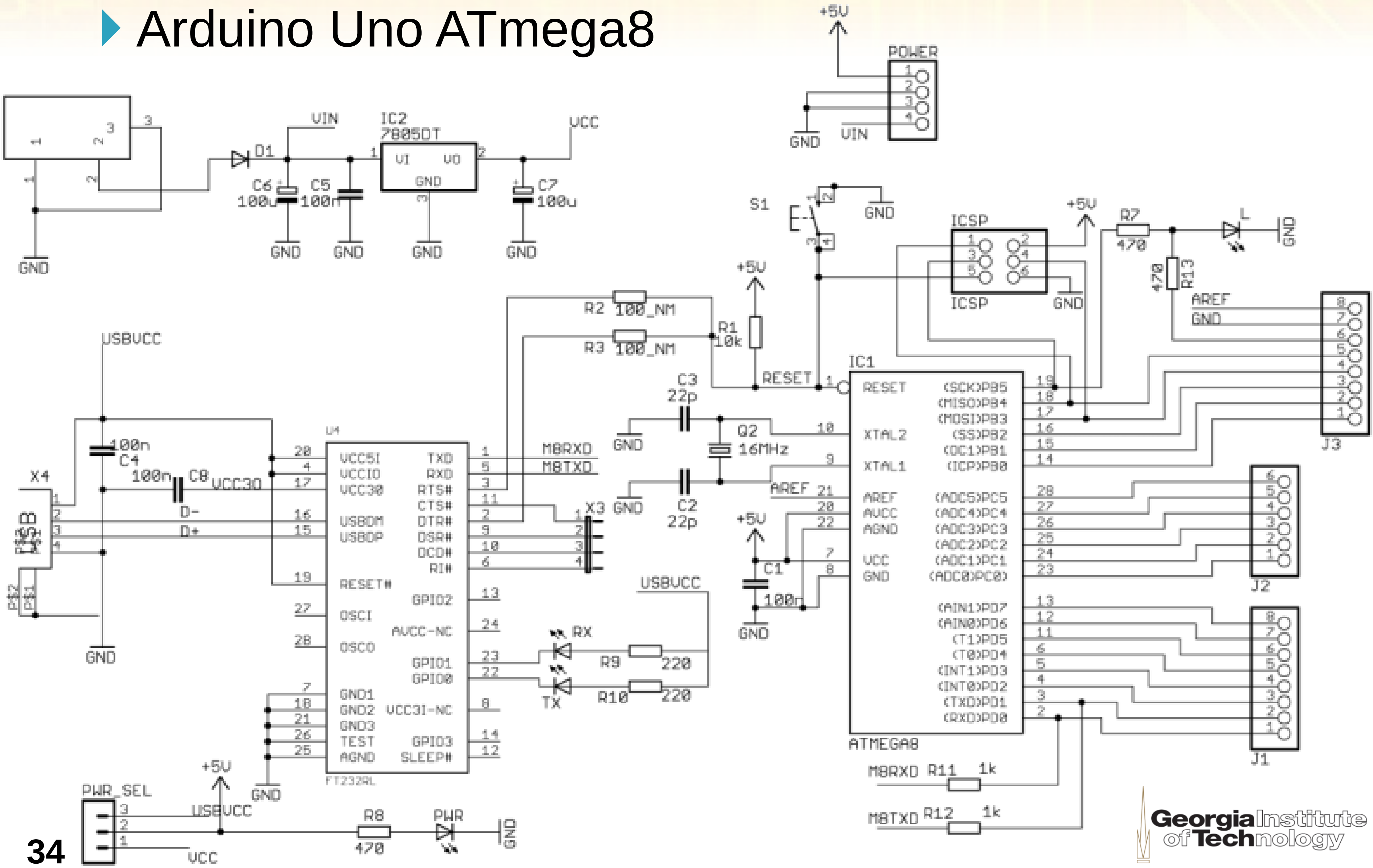
Common Schematic Components



Georgia Institute of Technology

Schematics - Example

- Arduino Uno ATmega8



Debugging Tools and Hardware

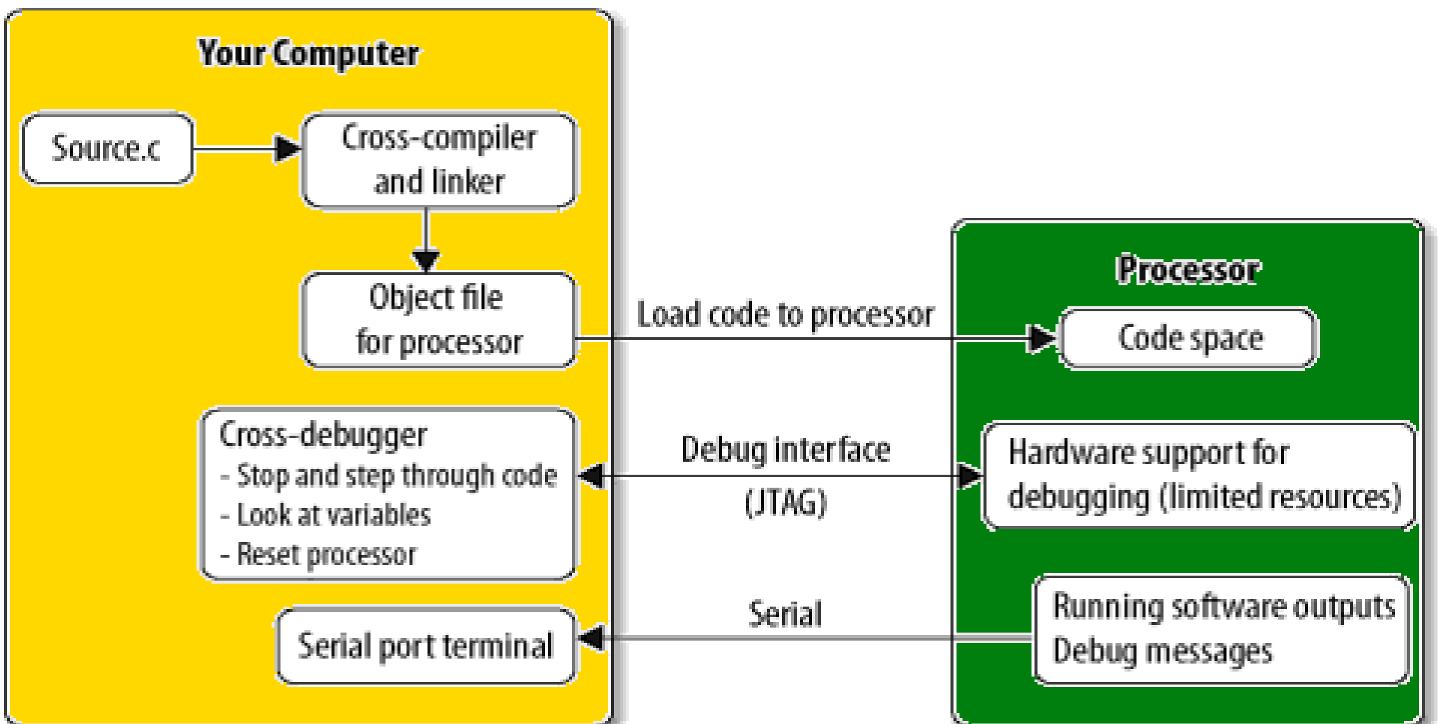
- Equip your station with
  - Handtools
    - Needle-nose pliers
    - Tweezers
      - Include mini-pliers
    - Screwdrivers
    - Box cutter
  - Measurement devices
    - Oscilloscope
    - Digital multimeter
  - Vision support/protection
    - Magnifying glass
    - Safety glasses
    - Flashlight
  - Miscellanous
    - Electrical tape
    - Sharpies
    - Cable ties
      - Velcro
      - Zip ties

Georgia Institute of Technology

Hardware-Software Integration

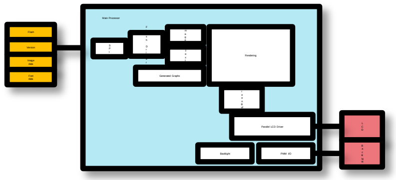
System Development

- Conception
- Prototyping
- Board bring-up
- Debugging
- Testing
- Release

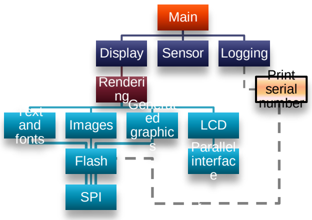


Conception

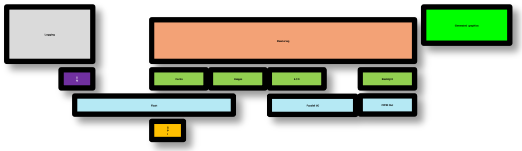
- Three different diagrams are recommended (White 2011)
  - Architecture block diagram
    - Helps define software modules



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

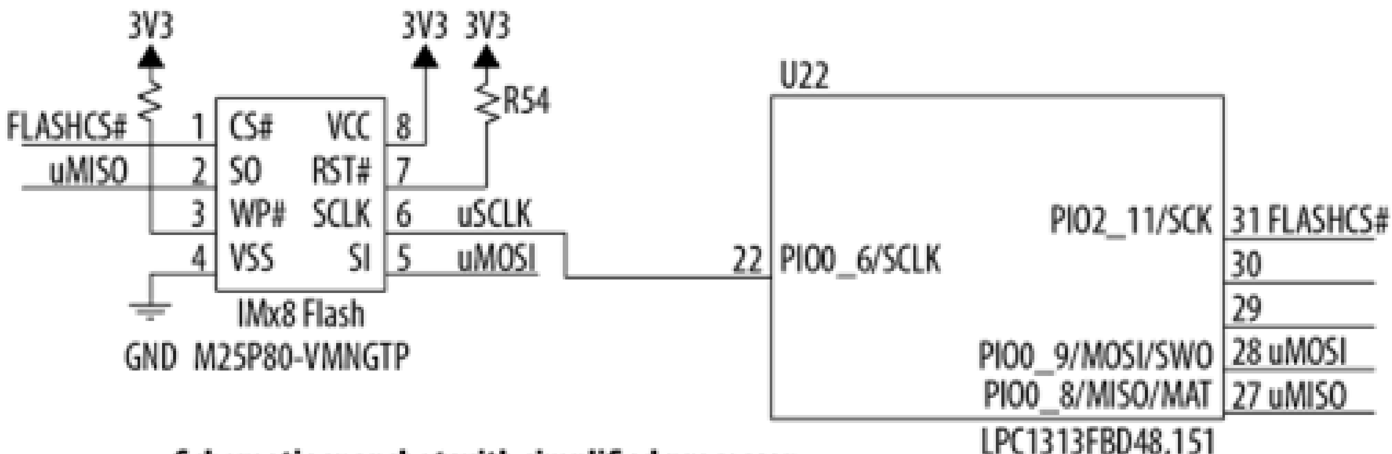


  - Software layering view
    - Allows to size modules by their complexity
    - Helps identify modules to be combined



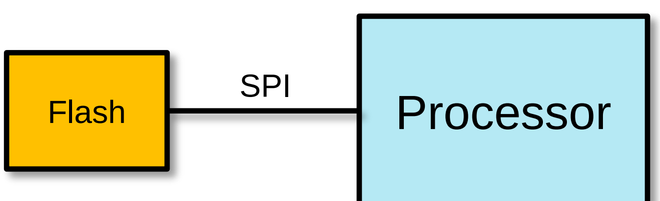
Conception - Example

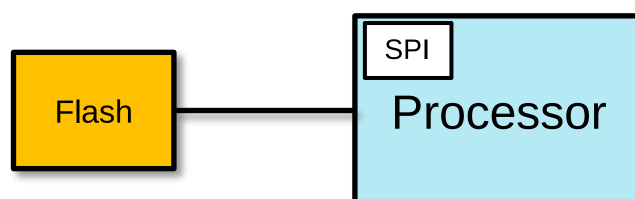
- Architecture block diagram



Schematic snapshot with simplified processor

- Hardware block diagram
- Software architecture

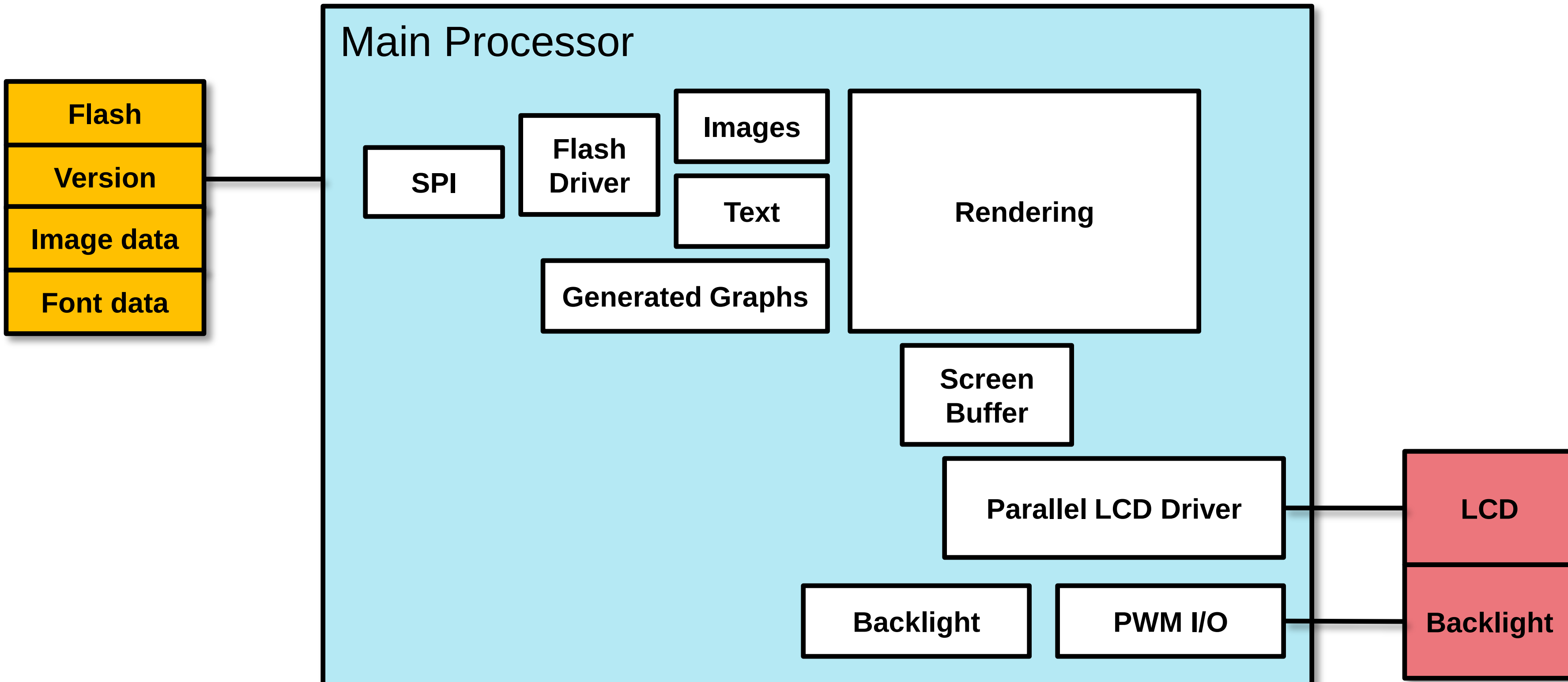




Elecia White, *Making Embedded Systems*, O'Reilly, 2011.

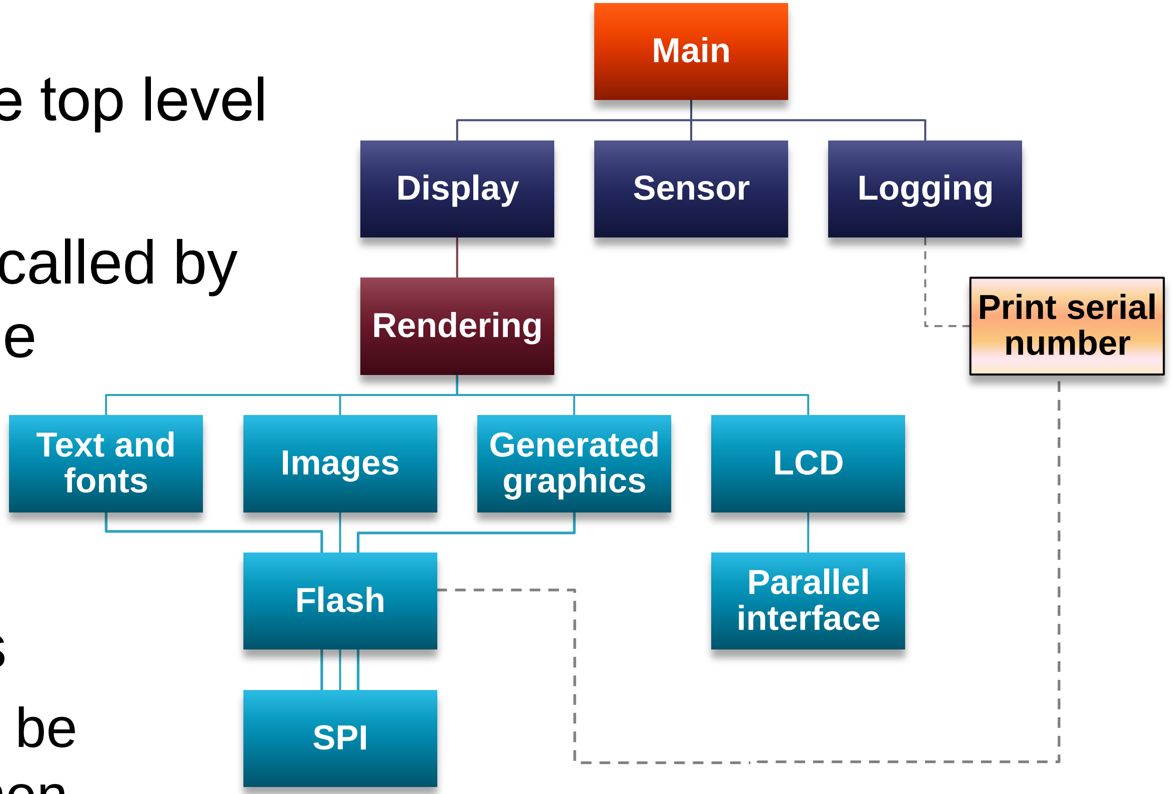
Conception - Example

- A more detailed software architecture block diagram
  - Continue adding modules as required by design elements



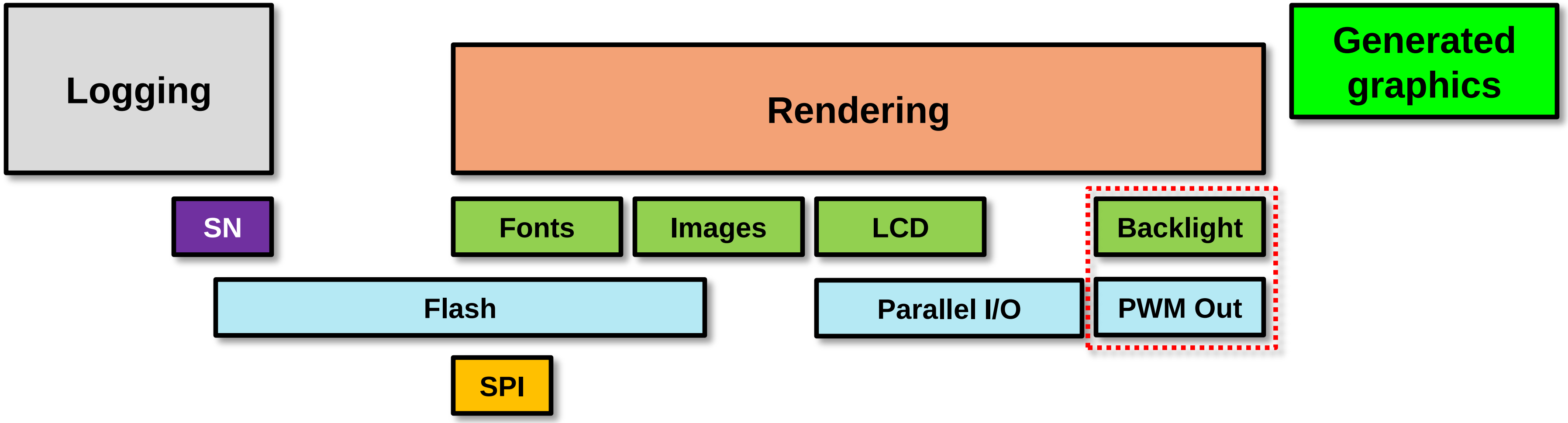
Conception - Example

- 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



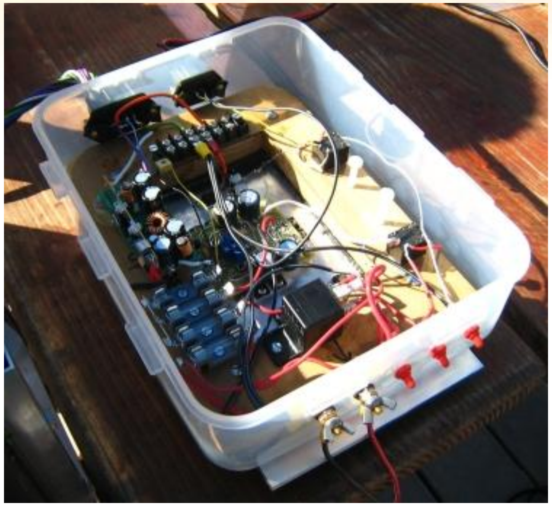
Conception - Example

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

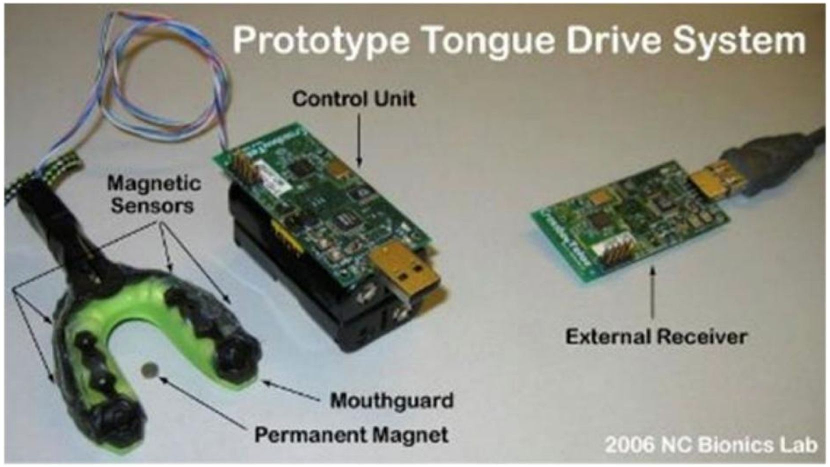


Prototyping

- 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



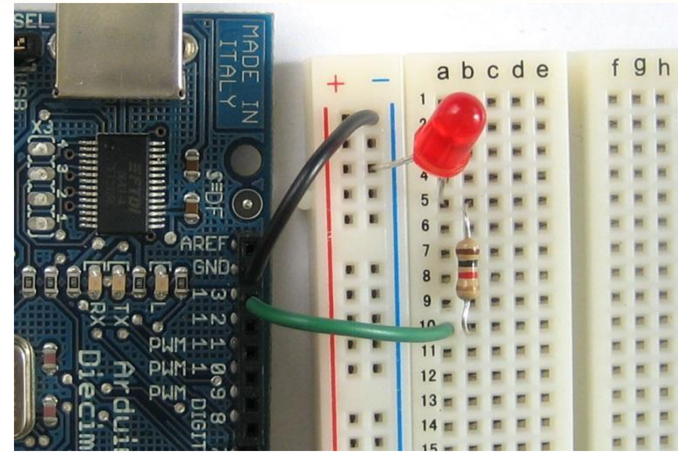
UCSD Aquanode Prototype



Georgia Institute of Technology

Board bring-up

- 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



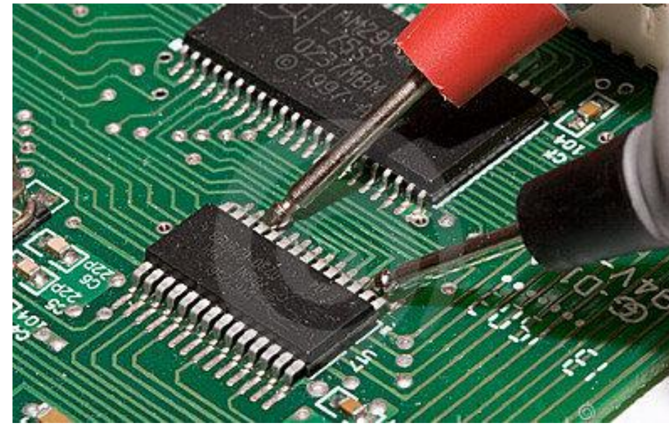
Georgia Institute of Technology

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 `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

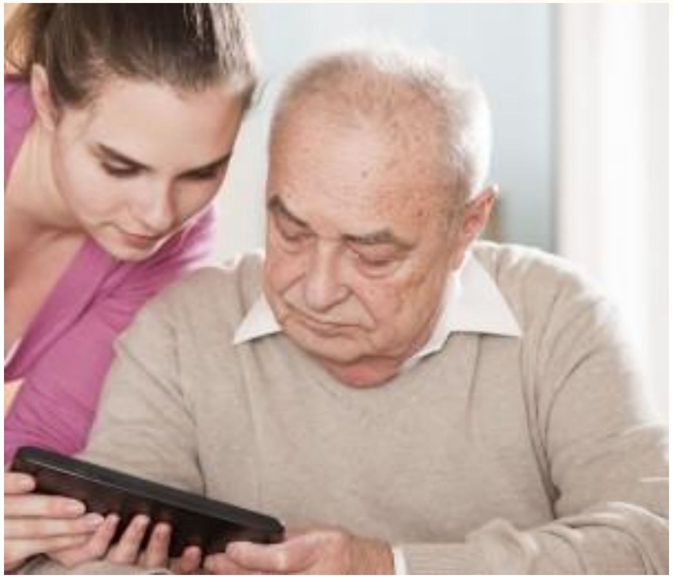


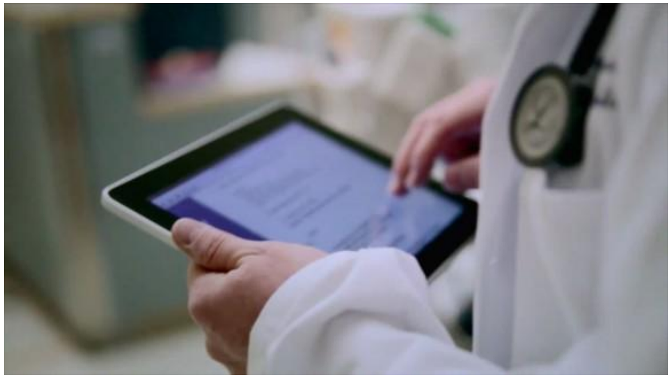
Release

- 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)

Release

- 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

Georgia Institute of Technology

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()`

Summary

- 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