CSCI1600: Embedded and Real Time Software

Lecture 4: Introduction to the Arduino

Steven Reiss, Fall 2017
What is an Arduino
Arduino Design: USB
Arduino Design: CPU
2. Overview

The ATmega8U2/16U2/32U2 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega8U2/16U2/32U2 achieves throughputs approaching 1 MIPs per MHz, allowing the system designer to optimize power consumption versus processing speed.

2.1 Block Diagram

Figure 2-1. Block Diagram
Arduino CPU
Embedded Processors

- Arduino: a microcontroller
  - Others exist as well
- General purpose processors
  - ARM processor (low power)
  - Raspberry Pi
- Special purpose processors
  - DSP for sound, video processing
  - Graphics processors
Other alternatives

- **Field-Programmable Gate Array (FPGA)**
  - Less expensive than creating chip
  - Like having a bucket of parts to wire together
    - Except the wiring is done programmatically
    - And the whole thing fits on a chip
    - Easy and cheap to make in quantity

- **Programmed in a high-level language**
  - Event-based, parallel
Arduino Restrictions

- Simple CPU
  - Single core, single threaded
  - No pipelining
- No floating point processor
  - How is floating point done
Arduino CPU

- Clock Speed: 16 MHz
- 32KB flash, 2KB SRAM, 1KB EEPROM
- 16 bit processing
What you need to know

Power/USB Input
Ground and power outputs
Digital input/outputs
Analog input/outputs
Reset button
Digital Input/Outputs

- **Inputs:**
  - Must be declared (default is output)
  - Should have value connected
  - Should provide input
  - Pull-up input might be enabled

- **Outputs**
  - Provide up to 40ma
  - Connect LEDs through 470 or 1K ohm resistor
  - Pins 0,1 shared with USB; Pin 13 with LED
Analog Inputs/Outputs

- Can also be used as digital inputs/outputs
- Must be declared
- Inputs (A1-A6)
  - Can measure 0-3, 0-5 volts
  - 10-bit A/D converter (0-1023)
- Outputs (on pins 3,5,6,9,10,11)
  - Uses pulse-width modulation
  - Values 0-255 (0-100% of 5v during cycle)
  - Pins 5,6 shared for clocks
Pulse Width Modulation

- 0% Duty Cycle - analogWrite(0)
- 25% Duty Cycle - analogWrite(64)
- 50% Duty Cycle - analogWrite(127)
- 75% Duty Cycle - analogWrite(191)
- 100% Duty Cycle - analogWrite(255)
Clapper

- **Input circuit**
  - Analog based on sound intensity

- **Output Circuit**
  - Two lights
  - Digital (on or off)
Clapper Input Circuit

Visualizing Sound

Diagram showing stages of signal processing:
- Input Signal
- Amplification
- Signal Balancing
- Digitization

LM386 IC shown on the right side of the diagram.
How does this work

- Microphone generates a weak signal
  - [https://www.jameco.com/Jameco/Products/ProdDS/136574.pdf](https://www.jameco.com/Jameco/Products/ProdDS/136574.pdf)

- LM386 is an amplifier chip
  - Amount controlled by resistor, capacitor

- Output balanced between 0 and 5 volts
Output circuit

- Connect Digital Output Pin to LED
- Through a 1K resistor (why)
- Other side of LED is grounded
How should the clapper work?

- Read the input
  - How often
- Check if we have a clap
  - What does a clap look like
  - How would you find out
- Check if we have a clap sequence
- Turn on lights based on inputs
Clapper requirements

- Detect claps
  - Based on sound input
- Handle flashing lights
  - How often to check
Arduino Operating System

- No real operating system per se
- Libraries
  - Handling I/O, timers, interrupts
- Single program is run
  - setup() run once
  - loop() run continually
Tasks

- The basic concept in RT/EM software
  - Break your project into tasks
- Tasks are independent
  - Communicate via shared variables
  - Communicate via network, etc.
- Tasks are run periodically or on demand
  - Periodic – run every k microseconds
  - Demand – run on event, interrupt, condition
Clapper Tasks

- Listen for sound
- Handle clapper logic
- Handle first light (on/off/flash)
- Handle second light (on/off/flash)
Detecting a clap
Finite State Models

- Annotate states
  - What they mean, what they do
- Annotate arcs
  - When they occur (timeout, error, signal detect)
- We’ll cover how to do this formally
Clapper Program

- Each task is broken up
  - Initialization for the task
    - Set up input pins, precompute tables, ...
  - What is done repeatedly
    - The real work of the task
- First part goes in (or called from) setup()
- Second part called from loop()
Scheduling Tasks

- **loop() needs to invoke all tasks**
  - Tasks that should run every k ms
  - Tasks that should run on event

- **Typical loop code:**
  - Get time
  - For each timed task
    - Check if it should run, call routine for it if so
  - For each event-based task
    - Check flag, call routine for it if set
  - Checks can be done in the task code itself
Clocks

- Arduino provides timers
  - 16 or 32 bit, clock (16MHz) resolution
    - Can be scaled by power of 2
- There are also internal clocks
  - millis(), microseconds()
  - Time since board last reset / started
    - These will wrap (how often?)
- User clocks as well
  - Tick based on internal clock
  - Task actions based on user clock
Homework

- Think about what the clapper code should look like logically