Lecture 2: Sensors, Actuators, and I/O
Discussion about labs
Review

Embedded systems are everywhere, have specific purposes, and unique challenges

Microcontrollers (MCUs) have CPU, I/O, memory on one chip

In lab you began working with an MCU and embedded hardware
In homework you previewed sensors, actuators, I/O
Today

- Sensors and actuators
- I/O
- Interrupts
Why would we as software engineers care about circuits, analog components, how I/O works, etc?
Voltage

Difference in electric potential

- Measured between two points (or one point and implicit ground)
- We say we measure voltage across a component

Voltage across resistor
Current

Rate of flow of charged particles through a circuit

Convention in circuits: imagine particles flowing from positive to negative terminal

We say we measure current through a component

Current through resistor
Parallel and series components

Series

Current is the same through both components
Voltage: ???

Parallel

Voltage is the same across both components
Current: ???
Kirchoff’s laws

Sum of **voltages** around a closed loop is zero

-\( V_{in} + V_1 + V_2 = 0 \)
-\( V_{in} = V_1 + V_2 \)

Sum of **currents** flowing *into* a node is the same as sum of currents flowing *out* of the node

-\( I = I_1 + I_2 \)
Ohm’s law & power law

Ohm’s law: \( V = IR \) (SI units: volts, amperes, ohms)

Power law: \( P = IV \) (SI units: watts, amperes, volts)

Useful for:

- Computing values needed to build circuit
- Figuring out the limits of what you can attach to your microcontroller
- Writing down accurate math for modeling your system
Thevenin equivalent circuit

Any linear electrical network containing current sources, voltage sources, and resistors can be replaced by an equivalent circuit with one voltage source and one resistor.
Thevenin for resistors in series

\[ V = IR \]

**Current is the same through both components**

Vin = V1 + V2

Vin = V1 + V2

V \_in = V1 + V2
Thevenin for resistors in parallel

\[ V = IR \]

Voltage is the same across both components

\[ I = I_1 + I_2 \]
Given Ohm’s law (V = IR) and the Power law (P = IV), what is the maximum power output of your Arduino pin (rated at 3.3 V, 7 mA)
What is the actual minimum resistance?
Interpreting device data sheets

Yellow LED data sheet
Digital devices

Leds are digital **output** devices

Things like push buttons are digital **input** devices

*(When connected correctly)* are driven by or produce a high/low signal
Can you give some examples of analog (produce/are driven by a continuous signal) peripherals?
Your book talks about:

- Accelerometers (measure acceleration of displaced mass)
- Anemometers (air flow for velocity)
- GPS (satellite for position)
- Gyroscopes (gimbals and modern)
- Microphones
- Engine controllers, thermometers, cameras, chemical sensors, etc
Other input components in your kits

- Photoresistor - resistance changes based on light
- Potentiometer - outputs voltage based on rotation of the dial
- Tilt sensor - Metal bearing completes circuit
Output components

Your book talks about:
- LEDs
- Motors (DC)

Your kits have:
- LCD screen (controlled digitally)
- Servo motor (controlled by lengths of high/low pulses)
- Piezo speaker (electricity displaces film to make sound)
How do you control a DC motor that requires an external power source?
Transistors

Basically an electric switch

Voltage applied to Gate connects Drain and Source

Come in different types (beyond the scope of this course)

The MOSFET transistor in your kits has a minimum gate voltage of 4.2V... what do we do?
Optocouplers

- Control one circuit using another, but they are completely electrically separate!
Motor driven using mosfet and optocoupler
Summary

Laws of physics help us compute properties of circuits, for design, understanding, and modeling

Devices can be driven/read with digital or analog signals, can be inputs (sensors) or outputs (actuators)
Break
Analog ↔ Digital I/O

Microcontrollers are digital

How do we read in analog input?

How do we produce or simulate analog output?
Modeling analog devices

Useful to have a mathematical model of sensors/actuators for verification and understanding
Linear and Affine Models, Range

Map *physical* value at time $t$ $x(t)$ to sensor input

Approximate with linear affine function, saturate if outside range $[L,H]$:

$$f(x(t)) = \begin{cases} 
ax(t) + b & \text{if } L \leq x(t) \leq H \\
AH + b & \text{if } x(t) > H \\
AL + b & \text{if } x(t) < L,
\end{cases}$$

(7.2)
Example: photoresistor

Physical value $x(t)$ is measure of light (e.g. illuminance).

MCU input voltage depends on physical properties of the sensor and how it’s wired.

First compute intermediate function $v(x(t)) = a'x(t) + b'$, which produces an input voltage between 0v and VCC.

Meaning $a'L + b' = 0$; $a'H + b' = VCC$

Can solve for L and H given $a', b', VCC$.
Quantization

Input to MCU is not a voltage, it is a number of some precision

For example, 10 bits precision: input is number from 0-1023

For our example,

\[ f(v(x(t))) = 1023 \times \frac{v(x(t))}{VCC} \]
If VCC = 3.3v and our precision is 10 bits, what is the smallest change in voltage that we can detect?
What is the smallest change in lux we can detect for our example?
Noise

Error of measured value

Comes from:

  Quantization

  Sensor imperfections

What is the noise of our photoresistor?
Analog to Digital Converter

Analog signal gets discretized to some number
Done via an *Analog to Digital Converter* (ADC)
MCUs have ADC built in

Different implementations - idea is to compare to reference voltages

Why it matters to embedded software developers: cost, timing, precision

Find this in the data sheet!
ADC sharing

**Mutex**
Up to individual task to check for ADC to be free and then ask for sample

**Time-triggered**
ADC runs in background and converts for all possible sources, storing the latest for each in a buffer
What tradeoffs do you see between mutex and time-triggered ADC sharing?
ADC sharing

Mutex

Time-triggered
What about outputs?

Digital to analog converter - pin produces voltage from 0 to VCC

Pulse Width Modulation - simulates analog output
Digital to Analog Converter (DAC)

Divide voltage based on digital number

Actuator driven by DAC has similar quantization error to ADC

Some MCUs don’t have DACs

- Expensive
- Fewer applications
- Unlike ADC, cannot share
Pulse Width Modulation (PWM)

Rapidly switch digital pin on and off
Creates perception of analog output
Increasing/decreasing duty cycle increases/decreases perception of power output level
Many microcontrollers provide PWM peripherals

Image source
Digital Signal Processing (DSP)

ADC/DAC/PWM combined with computational power of an MCU has enabled the explosion of digital applications

- Audio, video, robotics, medical...

MCU lets you take in an analog signal, do computations on it, and produce a new analog signal

DSP is a cool area but (mostly) beyond the scope of this course
These things are not perfect

Quantization, non-linearity, error in components all contribute to imprecision.

DSP can help alleviate some sources of error.

Design and models that take sources of error into account are vital for some applications.
Example of very simple DSP: debouncing

What do we do here?

Image source
Break
How did we read inputs from sensors in the first lab? What are the upsides and downsides to this method?
Sensors
Polling vs interrupts

Polling: reading input periodically, keep track of changes

Interrupt: be alerted when input changes/does something we are watching for

Real life examples: push notification vs checking texting app, rice cooker alerting you vs manually checking doneness, pet begging for food instead of you checking their bowl...
Types of interrupts

Software interrupts - function is called or some bits are set in a specific memory location to tell the software to go to an interrupt service routine (ISR)

Hardware interrupts - external trigger (voltage change on pin) tells software to go to an ISR

Exceptions - internal trigger (like writing to a protected memory location) triggers a fault
Interrupt process

1. Program executing normally
2. Interrupt triggered
3. Processor saves program state
4. Processor enters ISR
5. Processor restores program state
6. Program resumes executing
Main function

ISR

PC

Stack

SP
Main function

Interrupt happens

ISR

Code in memory

Stack

PC

ISR

Interrupt happens

SP
Main function

ISR

Code in memory

Stack

Old PC

Program state (local variables, etc)*

*architecture/MCU-dependent. Sometimes it is up to the programmer to save specific state such as registers
Main function
ISR
Code in memory

Stack
Program state (local variables, etc)
Old PC

Interrupt type 1: code memory location 1
Interrupt type 2: code memory location 2
....

Interrupt vector table
Main function

 ISR

 Code in memory

 Stack

 Program state
(local variables, etc)

 Old PC

 Interrupt vector table

 Interrupt type 1: code memory location 1
 Interrupt type 2: code memory location 2
 ....

 ISR

 PC

 SP

 Code in memory

 Stack

 Program state
(local variables, etc)

 Old PC

 Interrupt vector table

 Interrupt type 1: code memory location 1
 Interrupt type 2: code memory location 2
 ....

 ISR

 PC

 SP

 Code in memory

 Stack

 Program state
(local variables, etc)

 Old PC

 Interrupt vector table

 Interrupt type 1: code memory location 1
 Interrupt type 2: code memory location 2
 ....

 ISR

 PC

 SP
Main function

ISR

Code in memory

Stack

Program state (local variables, etc)

Old PC

PC

SP
Main function
ISR

Stack
Old PC
SP

PC

Code in memory
Main function
ISR

Stack
PC
SP
What if multiple interrupts happen?

Often interrupts are **prioritized**

Higher priority interrupt is allowed to interrupt lower priority interrupt

Ties broken by position in vector interrupt table

Programmer configures this when setting up code
Implementing ISRs

Often only one handler for some type of interrupt

Example on SAMD21 (your Arduino MCU): external interrupts have one ISR

Why?

Check flags that are set by MCU to see which pin/peripheral triggered the interrupt -- you will see this in lab
What is the difference between an interrupt and a subroutine call?
Interrupts can happen at any time

Subroutine call: you know exactly when in the code you call it

Interrupt: can happen at any point, even “inside” of a command

   Even \( x = x + 1 \) is made up of multiple machine instructions (load \( x \) from memory, increment \( x \), write value back to memory)

Atomic instructions: values being read/changed in atomic instruction cannot be read/changed by anyone else
Homework problem 10.7

(a) Is it possible for the ISR to update the value of sensor1 while the main function is checking whether sensor1 is faulty? Why or why not?

(b) Suppose a spurious error occurs that causes sensor1 or sensor2 to be a faulty value for one measurement. Is it possible for that this code would not report “Sensor1 faulty” or “Sensor2 faulty”?

(c) Assuming the interrupt source for ISR() is timer-driven, what conditions would cause this code to never check whether the sensors are faulty?

(d) Suppose that instead being interrupt driven, ISR and main are executed concurrently, each in its own thread. Assume a microkernel that can interrupt any thread at any time and switch contexts to execute another thread. In this scenario, is it possible for the ISR to update the value of sensor1 while the main function is checking whether sensor1 is faulty? Why or why not?
**Why do we care?**

Interrupts are powerful and used widely in embedded programming.

Understanding how interrupts affect program state/variables aids in design and debugging.

Interrupts complicate modeling and timing analysis of a system (more on this later).
Summary

Physical devices (sensors and actuators) obey the laws of physics.

Can be controlled by digital or analog signals.
   - Can model this conversion using math.
   - Conversion between digital/analog: ADC, DAC, PWM.

Interrupts allow us to detect changes in inputs instead of just polling for them.