Input and Output

CS 447– Wireless Embedded Systems
Outline

- I/O hardware
- PWM
- GPIO
- Serial interfaces
- Parallel interfaces
- Interrupts and exceptions
- Atomicity
- Modeling interrupts
I/O Hardware

In CPS, mechanisms in processors that support interaction with the physical world are critical!

• Require careful design

• Too much current to microcontroller (µC) could fry component
I/O Hardware

Also, in physical world, things happen at same time
• But software is sequential

Challenge of designing real-world CPS

• Incorrect interactions between sequential code and concurrent events in physical world => system failures

• Why it’s so important to carefully design CPS
I/O Hardware

Embedded processors typically include number of input and output mechanisms on chip
• Exposed as pins to designers: e.g., Raspberry Pi (ARM chip)
I/O Hardware

E.g., Arduino Uno pin out (Atmel chip)
I/O Hardware

E.g., Luminary Micro Stellaris μC (32-bit ARM)
I/O Hardware

Pin outs (aka breakout boards) are useful for prototyping
• Final product has only required components (nothing extra)
• E.g., standalone Atmel chip with components
I/O Hardware

• Developers use IDE (e.g., Arduino, Atmel Studio) to write software to the board
• Gets loaded / stored in FLASH memory
Outline

- I/O hardware
- PWM
- GPIO
- Serial interfaces
- Parallel interfaces
- Interrupts and exceptions
- Atomicity
- Modeling interrupts
Pulse Width Modulation

- PWM – technique to deliver variable amount of power to external hardware (e.g., motor, LED, heating coil)

- Uses digital circuits only (easy to integrate into μC)

- Programmer typically writes value to memory-mapped register to set duty cycle (e.g., 10%)

- Device then delivers power to external HW proportional to duty cycle
Outline

- I/O hardware
- PWM
- GPIO
- Serial interfaces
- Parallel interfaces
- Interrupts and exceptions
- Atomicity
- Modeling interrupts
General Purpose I/O

• Embedded system programmers often need to connect specialized or custom HW to µC

• Many processors have GPIOs, allow software to read or write voltage levels representing logical zero or one
General Purpose I/O

- **Active High Logic** - voltage close to Vdd (processor supply voltage) considered 1. Voltage close to 0 represents 0.

- **Active Low Logic** – interpretations are reversed: voltage near Vdd represents logical 0. Voltage close to 0 represents logical 1.
General Purpose I/O

GPIOs can be configured as **output**

- Enables software to write to a memory-mapped register to set output voltage to high or low
- Allows software to directly control external devices
- **NOTE:** designers must understand specifications of external device (e.g., voltage and current levels)
General Purpose I/O

GPIOs can be configured as **output**

- Important to maintain *electrical isolation* between processor circuit and external devices
- External devices may have noisy electrical characteristics
- May spill over to processor (make it unreliable)
- External device may operate at different voltage
- World of EEs. But important to be aware of
General Purpose I/O

GPIOs can be configured as **input**

- Software can read and react to external voltages

```c
pinMode(6, INPUT);
bool x = digitalRead(6);
```
Outline

- I/O hardware
- PWM
- GPIO
- Serial interfaces
- Parallel interfaces
- Interrupts and exceptions
- Atomicity
- Modeling interrupts
Serial Interfaces

CPS typically have design constraints
• Physically small packages
• Low power consumption

Consequence: limited number of pins on processor
• Each pin must be used efficiently
• Need to minimize number of wires b/t subsystems to decrease footprint and cost
Serial Interfaces

One way to use pins + wires efficiently: use **serial interface**

- Information sent serially as sequence of bits
- Many different standards (e.g., RS-232, UART)
- **UART** – Universal Asynchronous Receiver/Transmitter
  - Converts 8-bit register into sequence of bits

- **Baud rate** - how long it takes to send 1 bit of information
  - E.g., 57600 bits / second
  - How long to send 1 byte of information? 139 µs
  - (8 bits / 57600 bits per sec)
Serial Interfaces

Newer devices use **USB – Universal Serial Bus**
- USB 1.0 = 12 Mbits / sec
- USB 2.0 = 480 Mbits / sec
- USB 3.0 = 4.8 Gbits / sec

USB is electrically simpler than RS-232
- But requires more complex control logic to support
- Modern peripherals (e.g., printer) use a dedicated µC for this
Serial Interfaces

Another option: JTAG - Joint Test Action Group

- Usually provide debug interface to embedded processors
- Enables PC-hosted debug environment to examine and control processor
- E.g., read state of processor registers, set breakpoints, step
Serial Interfaces

I²C – Inter-integrated Circuit

- Allows one or more “slave” devices to communicate with master.
- Only requires two wires to communicate (SDA, SCL).
- Can have multiple slaves on bus (uses addressing scheme).
Serial Interfaces

I²C – Inter-integrated Circuit

• Arduino has robust I²C support (i.e., Wire library)
Serial Interfaces

SPI – Serial Peripheral Interface
• A bit more complicated than I²C
• Requires 4 wires for communication:
  • **MOSI** – Master Out Slave In
  • **MISO** – Master In Slave Out
  • **SCLK** - Clock
  • **CS** – Chip Select (aka SS – Slave Select)
Serial Interfaces

SPI – Serial Peripheral Interface

• Master “selects” Slave by pulling CS HIGH*
• Master and Slave communicate using MOSI and MISO
• MOSI – from Master to Slave
• MISO – from Slave to Master
• Master “deselects” Slave by pulling CS LOW*
Serial Interfaces

SPI – Serial Peripheral Interface

• Can handle multiple Slaves on same bus
• Master selects particular Slave by pulling CS pin HIGH
Serial Interfaces

SPI – Serial Peripheral Interface
• Arduino has a robust SPI library (i.e., SPI)

Basic operation of SPI:
• Configure SPI parameters (clock rate, data mode, bit order)
• Begin SPI
• Select particular Slave to communicate with
• Transfer / receive bytes simultaneously (1 byte out = 1 byte in)
• Deselect Slave
Serial Interfaces

Other serial interfaces

- PCI Express
- FireWire
- MIDI
- SCSI
- Networked
- ...

January 11, 2018
Outline

• I/O hardware
• PWM
• GPIO
• Serial interfaces
• Parallel interfaces
• Interrupts and exceptions
• Atomicity
• Modeling interrupts
Parallel Interfaces

• Serial interface: Tx / Rx a sequence of bits over single line

Parallel interface: multiple lines simultaneously
• Each line is serial
• Logical grouping and coordination make it parallel
  • E.g., DB-25 connector
Parallel Interfaces

Group of GPIO pins can be used together to implement parallel interface

• Often used to emulate interface not supported by HW

Intuition: parallel is faster

• But.. Requires more wires
• More control
• Serial comm is most common!
Outline

- I/O hardware
- PWM
- GPIO
- Serial interfaces
- Parallel interfaces
- Interrupts and exceptions
- Atomicity
- Modeling interrupts

January 11, 2018
Interrupts and Exceptions

**Interrupt** – mechanism for pausing processor’s current execution and execute a predefined code sequence

- Code sequence – **Interrupt Service Routine (ISR)** aka handler

Three types of interrupts:
- Hardware
- Software
- Exception
Interrupts and Exceptions

**Hardware Interrupt** - external hardware changes voltage level on an interrupt request line

- Arduino provides hardware interrupts via digital pins 2, 3

- Search `attachInterrupt()` for more details
Interrupts and Exceptions

Hardware Interrupts can be level triggered or edge triggered

- **Level triggered** – HW must hold specific voltage for some predefined amount of time
- **Edge triggered** – HW changes voltage
Interrupts and Exceptions

**Software Interrupt** – program triggers interrupt by executing some special instruction

- E.g., timer interrupt
- Arduino has many timer libraries (e.g., MsTimer2)
Interrupts and Exceptions

Exception – interrupt triggered by internal hardware that detects a fault

• E.g., segmentation fault
• NOTE: Arduino does NOT handle exceptions
  • x[10000] is totally legal! (even though Arduino only has ~2k RAM)

• Your program will just behave in undefined manner
Interrupts and Exceptions

- For **hardware** and **software** interrupts, code will resume execution after ISR completes.

- For **exceptions**, code does NOT typically resume execution after ISR completes.
  - Instead: program counter set to some fixed location.
Outline

• I/O hardware
• PWM
• GPIO
• Serial interfaces
• Parallel interfaces
• Interrupts and exceptions
• Atomicity
• Modeling interrupts
Atomicity

ISR can be invoked between any two instruction of main program

• Challenge of embedded software development: reasoning about possible interleaving of instructions
Atomicity

Timers
• Most μC have timers
• Count down from some number to zero
• On zero, interrupt raised
• Counter resets

Much more precise than restarting timer in each ISR
• Why? Because time between count = 0 and timer restart is non-deterministic
• Avoid using Arduino delay( ) as a repeating timer! Why?
Atomicity

In 8-bit µC, atomicity is extremely important!

• E.g., single line of code: “x = 2000;”

• How does 2000 fit into an 8-bit instruction?
• Requires two separate 8-bit instructions

• ISR could occur between two instructions!
Atomicity

Such bugs are **really** hard to track down
• Also, problematic interleaving are uncommon
• May not show up in testing..

But what about safety critical applications that cannot fail?
• Must take great care to avoid such bugs
• Important to carefully *model* and *design* system first!
Outline

• I/O hardware
• PWM
• GPIO
• Serial interfaces
• Parallel interfaces
• Interrupts and exceptions
• Atomicity
• Modeling interrupts

January 11, 2018
Modeling Interrupts

Behavior of interrupts can be difficult to understand

• Many catastrophic system failures caused by unexpected behaviors

• E.g., ADC chip interrupts during write to SD card

• One way to make logic more precise? model it as a Finite State Machine...