CSCI1600: Embedded and Real Time Software
Lecture 7: Input Output Concepts (Nick)
Steven Reiss, Fall 2017
Input and Output

- I/O devices are attached to the computer
  - What does this mean?
  - How does the CPU interact with them?
  - What does this mean to the programmer?
Low-Level Details

- Allocate a portion of the address space to I/O
  - Read and write to those addresses talks directly to device
  - Read and write to those addresses talks to an I/O controller
- Special instructions to do this
  - No caching, ensure completion
- **Every device is different**

Lecture 7: Input Output Concepts 9/25/17
2. Overview

The ATmega48A/PA/88A/PA/168A/PA/328/P 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 ATmega48A/PA/88A/PA/168A/PA/328/P achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

2.1 Block Diagram

The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

- PORT C (7)
- PORT B (8)
- PORT D (8)
- USART 0
- 8bit T/C 2
- 8bit T/C 0
- 16bit T/C 1
- A/D Conv.
- SPI
- TWI
- SRAM
- Watchdog
- Oscillator
- Internal
- Bandgap
- Analog
- Comp.
- FIFO
- SRAM
- Flash
- EEPROM
- Watchdog
- Oscillator
- Circuits / Clock
- Generation
- Power
- Supply
- Voltage
- Comparator
- POR / BOD &
- Reset
- VCC
- GND
- AVCC
- AREF
- GND
- RESET
- XTAL[1..2]
- CPU
Device Controller

- Provides one or more I/O lines (Parallel)
- Input Lines
  - Can be connected to input
    - Directly
    - Pulled up
    - Schmitt trigger (threshold-based)
  - Can be latched
- Can be indirect
  - Opto-isolated
- Built into the Arduino
Device Controller

- **Output lines**
  - Pin connected through a driver for output
    - Provides some current (not too much)
  - Can connect through a driver for more power
  - Can be latched
Physical Hardware

- Peripheral Interface Adapter (PIA)
  - 16 lines, 4 control lines
  - Bus interface to CPU
- Built into the Arduino
  - Provides 14 digital lines
- Can interface to memory/IO bus
- Analog lines handled by separate hardware (ADC)
Pinball I/O

- **Problem**
  - 60 lights that must be controlled
  - 56 switches that must be sensed
  - Limited number of input/output ports

- **Solution**
  - Use an array of controls
Array of Controls
Alternative Solutions

- Use a multiplexor
- Use a latching shift register
- Use a resistor ladder
Multiplexer

![Multiplexer Diagram]

Lecture 7: Input Output Concepts
Demultiplexer (Demux)
Controlling 64 lights

- How many I/O lines are needed
- Problem
  - Can only drive a fraction at one time
  - Flash them fast enough and it will work
- Problem
  - Not enough current
Driving Outputs

- Arduino Uno output pins can provide a maximum of 20mA

Alternatives:
- Use a driver circuit
- Use a relay
Serial I/O

- Parallel inputs are limited
- Serial I/O: send multi-bit data with fewer pins
  - Send data over one line at a fixed rate
  - Used often for communication
  - Timing modes
    - Asynchronous: Must agree on timing
    - Synchronous: Shared clock signal
  - Can use with shift registers to simulate parallel I/O
Serial I/O with Latching

- How would you drive lights serially
  - Can you construct a circuit to drive the lights
  - Given which lights are on
- 74HC595: 8 bit shift register
  - Data is written to the chip serially
  - Latched in the storage register of the chip
  - Latched values then drive lights
Controlling 512 Lights

One of the 8 MOSFETs supply power to one of the 8x8 layer grids once the desired pattern for that layer has been setup in the shift registers.

Repeat the above circuit for a total of 8 layers.

All current limiting resistors are 220 ohm.

Eight shift registers independently control the "ground" for each of the 64 LED pillars. Note how pin 10 from previous shift register is used as input (14) for the next shift register. Arduino shifts all 64 bits for the layer and then supplies (+) on the desired layers via one of the 8 MOSFETs.
Resistor Ladder

- Read combinations of inputs based on voltages
- Creates different voltage dividers based on number of switches pressed
I/O Timing

- **Input comes at random points in time**
  - Handling timing issues is central to embedded computing
    - Must respond to an input within a certain time
    - Must deal with outputs in a timely fashion

- **Output should only be sent when the device is ready**
  - Device might not see output if it changes too fast

- **CPU can be responsible for timing**
  - Poll the inputs at the correct rate (input must be stable)
  - Turn outputs off and on at correct rate
  - Can complicate the program
Interrupts

- I/O device can have a ready line
  - Polling is not always desirable
  - Very accurate times might mean busy wait
  - Think of reading serial input at fixed rate
- Instead the I/O device can tell the CPU when it is ready
  - This is typically done using an interrupt
Device Perspective

- **When should a device interrupt**
  - Can be obvious (UART)
- **Can be flexible**
  - When a line turns on (or off)
    - Or voltage goes above/below a certain level
  - More complex conditions
    - When a set of lines gets to a certain state
    - When any of a set of lines changes
  - Set via hardware or software
CPU Perspective

- CPU has 2-16 interrupt request lines
  - IRQ : interrupt request
  - NMI : non-maskable interrupt
- If a device sets these lines
  - CPU stops what it is doing
  - Starts executing in a separate context at fixed address
Interrupt Priority and Masking

- Different lines have different priorities
  - Higher priority can interrupt lower priority processing
  - Lower priority wait for higher priority to finish
  - Can handle multiple interrupts at once

- CPU can mask off interrupts
  - All interrupts beyond a certain level
  - These pend until unmasked
  - Higher interrupts mask off those of lower priority
Interrupt Programming

- Interrupts introduce multiprocessing
  - Need to worry about race conditions, synchronization, deadlock
- They need to be coded very cautiously
  - Shared data can be problematic, esp. data structures
  - Note that even 16/32 bit integers might be multiple values
- Treat as multithreaded programming
  - Non-blocking synchronization techniques
  - Critical regions
    - Locks = masking interrupts
  - Interaction with real synchronization can be weird (later)
Arduino Interrupts

- User provide a routine to be called when an interrupt occurs
  - For either DIO or AIO line (small set are usable)
  - Specific line and condition is settable
- Interrupts can be masked
- Also have timer interrupts
Arduino Interrupt Coding

#include <PinChangeInt.h>
#include <PinChangeIntConfig.h>

int led = 13;
int sensor=7;

void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
  Serial.begin(9600);
  PCintPort::attachInterrupt(sensor,count,CHANGE);
}

void loop(){
void count()
{
  // Write your ISR
}
Interrupt Programming

- Do the minimum amount of work possible
- Small and fast interrupt routine
  - Set shared flags for processing later
  - Read input and save for future processing
- Protect access to these flags
For the rest of this class

- Continue working on Lab 1
- If you are done, optional task:
  - Create a circuit to drive 4 (or more!) LEDs in a matrix configuration
Homework

- Read LS Ch. 2.1-2.2
  - As much as you can follow
  - Come prepared with questions