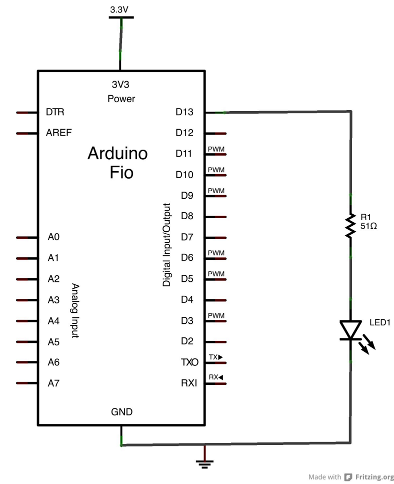
CS 5422: Physical Computing

I/O and Interrupts

Slides will be available through Blackboard. There is no textbook.

Interacting with the Physical World

We've seen the following schematic.



Interacting with the Physical World

How do we change the I/O pin?

- `digitalWrite(pin, HIGH);` (or `LOW`)
- ... but what does this actually do?

Short answer: there is an electrical circuit that is *controlled* by the microcontroller.

All programming languages get translated into *assembly language* by the compiler.

Assembly Language

The "native" language of the microcontroller

- MIPS: `add $10, $15, $13`
- MSP430: `add.w R5, R6`
- AVR: `add r2, r0`

The "processor family" or "processor architecture" specifies the details of the language.

- Native operations
- Operand sizes/types
- Addressing modes (how operands are specified)

How does I/O work?

Various options in terms of the assembly-language:

- Special instructions
- Special registers
- Special memory locations

. . . and sometimes combinations

I/O locations/groups are typically called ports

I/O Instructions

Special instruction in the ISA for input/output

Example: AVR ISA

- `out $18, r16`
  - Output value stored in register `16` to port 18
- `in r25,$16`
  - Read the value in port 16 and store it in register `25`

Ports are special operands

I/O Instructions

Special register in the ISA for input/output

Example: SNAP ISA

- `add $15,$1,$2`
  - Register 15 is mapped for output operations
- `add $1,$15,$2`
  - Register 15 is also mapped for input operations

I/O operation determined by writing specific values to `$15`.

I/O Instructions

Memory mapped I/O: Reads and writes to specific memory locations correspond to I/O operations

Example: MSP430 ISA

- `bis.b #0x01,&P1OUT`
  - Modify memory location specified by `P1OUT`
- The same operation also *reads* from `P1OUT`

AVR also uses this approach.

Memory-mapped I/O

If a variable `var` is located at an *I/O address* then:

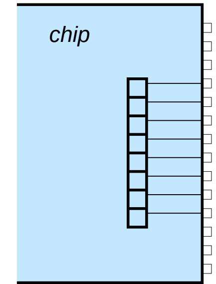
- `x = var`
- `var = y`

Standard assignment statements can be used to access I/O ports.

How do we know `var` is located at a certain address?

- Special pre-defined (external) variables
- Linker knows that those variables are located at certain fixed addresses

Output



On-chip registers connected to I/O pins

Implementing the instructions:

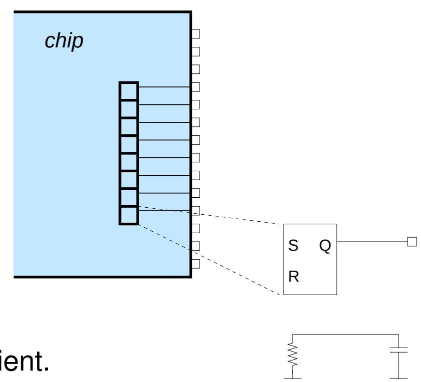
- Write register for output values
- Change in state appears on the pins
  - . . . after a small delay

How do we analyze the delay?

How do we determine if the output can even be set?

Output

- Simplified effective circuit
- Model should include:
  - Register
  - Package
  - Board
  - Other components

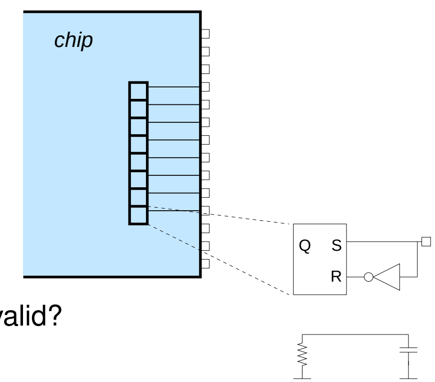


Typically RLC models are sufficient.

Input

What about input?

- Analog v/s digital
- Value must be stable



When do we read the input?

How do we know the value is valid?

Input

Need some communication *discipline*

Examples:

- Use a valid bit
  - 9-bit input, with 8-bits of data
  - Toggle 9th bit to indicate new data
- Use encoded data
  - One-hot encoding
  - `01` = *false*, `10` = *true*, `00` = *no data*

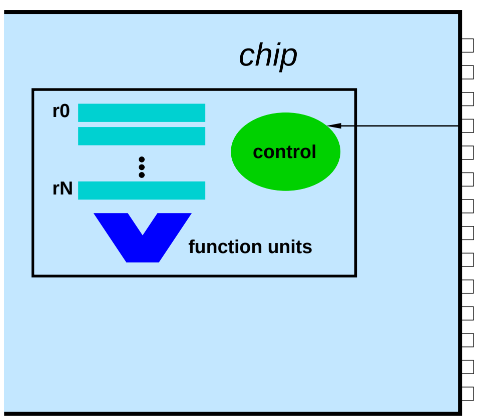
More on this later ...

Input

When do we read the input?

- Polling
  - Keep reading the value in a loop
- Interrupts
  - Notify processor that there is a new input

Interrupts



Interrupts

What happens on an interrupt?

- Control flow modified
  - Interrupt *vector*
  - Normally multiple, based on type of interrupt
- Record information about the interrupt
  - Flags that specify which interrupt occurred
  - Sometimes additional registers with extra information
- Information saved to permit execution to be resumed

... like a special procedure call.

Interrupts

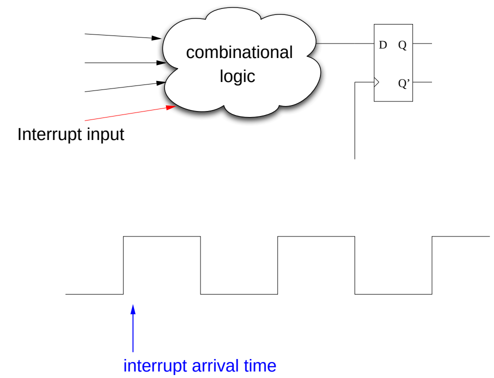
Two major types:

- Non-maskable
  - Fixed function interrupts
  - Can't disable them
  - Example: reset
- Maskable
  - User-controlled
  - Can selectively activate them

Multiple interrupt priorities to resolve conflicts.

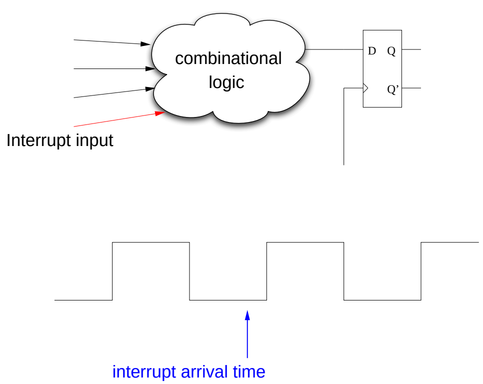
Interrupts

Subtle issue: what happens on an interrupt?



Interrupts

Subtle issue: what happens on an interrupt?

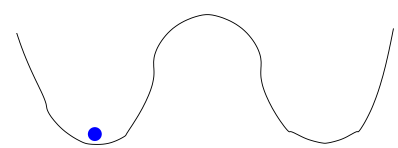


Interrupts

The flip-flop can enter a *metastable* state.

Examples:

- Inverted pendulum
- Potential well



Interrupts

Metastable states are inherently unstable.

For flip-flops, the probability of staying in a metastable state for at least t time units

$$p(t) \propto e^{-t/\tau}$$

Standard way to resolve this?

Interrupts

The Arduino library handles most of this for you.

- `attachInterrupt (interrupt, function, mode);`
  - `mode`: low value, rising edge, falling edge, change in value
- `detachInterrupt (interrupt);`

Interrupts can also be globally disabled:

- `interrupts();`
- `noInterrupts();`