04 - Interrupt Programming

CEG 4330/6330 - Microprocessor-Based Embedded Systems
Max Gilson

Polling

- Polling is when the computer is forced to wait for something to happen
    - Technically involves input/output devices
    - Waiting in general is wasteful and bad design/programming
- Waiting wastes processing power, time, electrical power, and is generally a bad idea
- Examples:
    - Waiting inside a loop until a pin goes HIGH
    - Using delay()
        - The delay function polls the millis() function until the desired amount of time has passed

Alternatives to Polling

- You can almost always avoid polling and delay()
- Use millis()
  - millis() returns the total amount of milliseconds of how long the Arduino has been running
  - Check millis() once in your loop function
  - If the millis() exceed a threshold, execute some code
  - This prevents your program for waiting long periods of time to do something useful
  - Not as useful for *some* external devices (not precise timing)

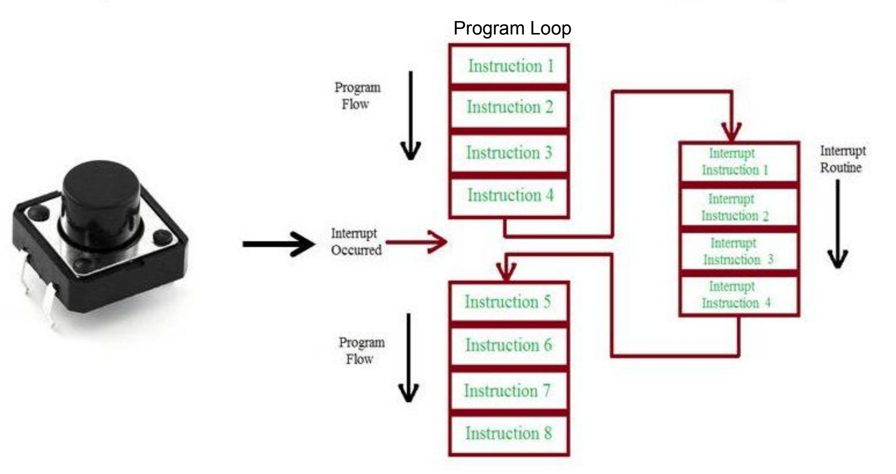
millis() Example Code

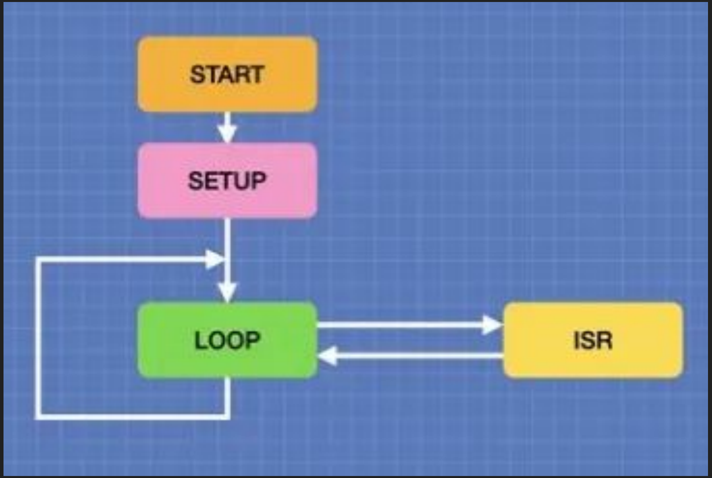
```
unsigned int toggleTime = 500;  // Amount of milliseconds to wait to toggle
unsigned int lastToggled = 0;   // The last time (in milliseconds) when we toggled

void setup()
{
  // Set the LED pin as an output
  pinMode(13, OUTPUT);
}

void loop()
{
  // Get the current time in milliseconds
  unsigned int currentTime =  millis();

  // If the current time is greater than the
  // last time we toggled + the amount of milliseconds
  // we need to wait to toggle, then toggle the pin
  // and record the current amount of milliseconds
  if(currentTime >= lastToggled + toggleTime)
  {
    digitalWrite(13, !digitalRead(13));
    lastToggled = currentTime;
  }
}
```

Alternatives to Polling (cont.)

- Use interrupts
  - An interrupt pauses your program’s execution, executes the code inside the interrupt service routine (ISR), then returns back to your previous program
  - This is most commonly used with external devices for time sensitive execution
  - If an external device sends a rising/falling edge, this can be used to trigger an interrupt

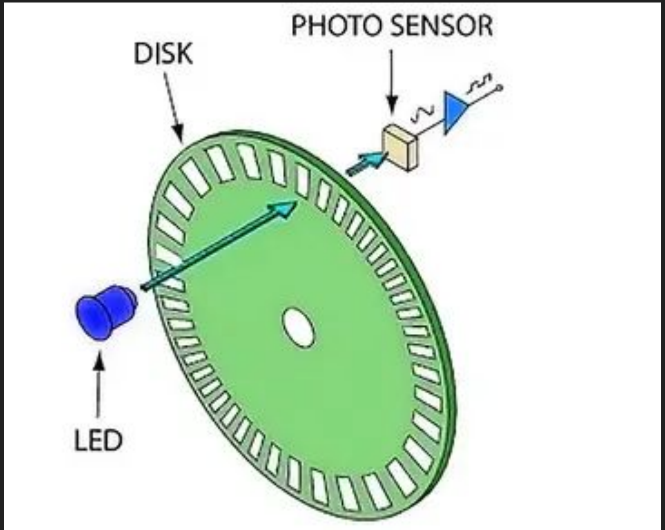
How Interrupts Work

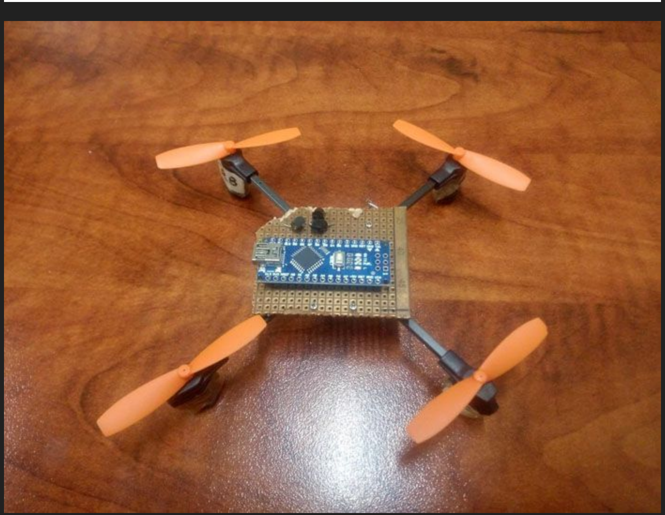




Interrupts Examples

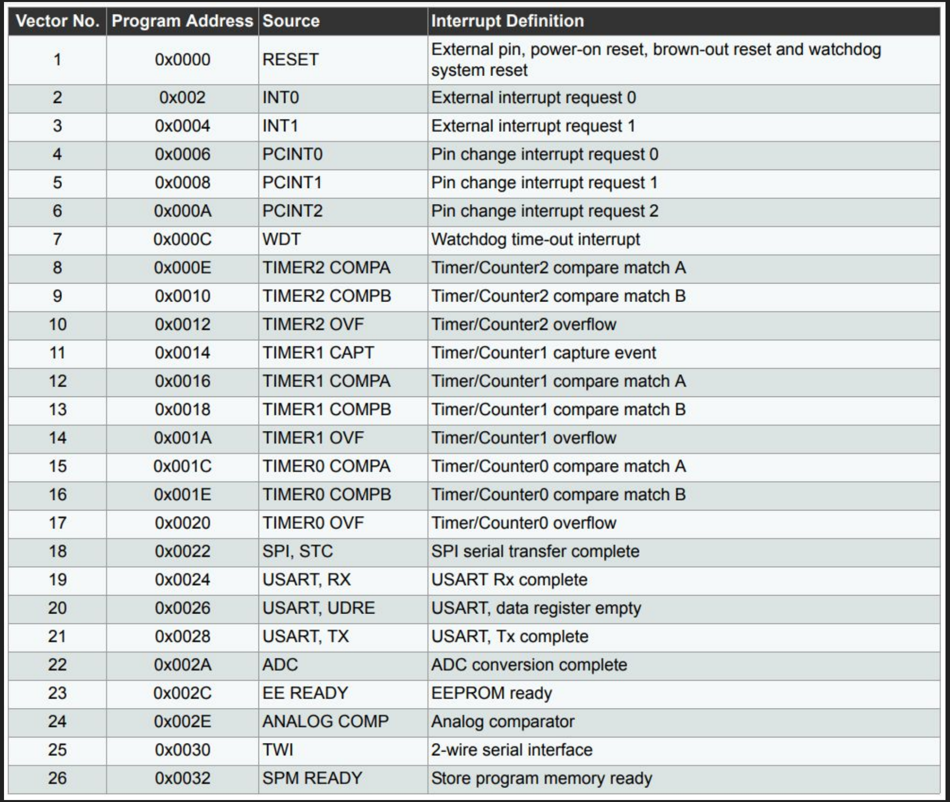
- Wheel Speed Encoder
    - Used in electric vehicle motor
    - LED sensed by photo sensor
    - Pulse looks like square wave that relates rotational velocity to PWM
- Inertial Measurement Unit (IMU)
    - Used in drones
    - Precise timing and calculating is required to keep a drone flying
    - Receive a pulse from the IMU when it is ready to transmit fresh data





Interrupt Vector Table

- Location in memory of all the addresses the interrupts exist at
- Lists all the possible interrupts on Arduino



Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete
23	0x002C	EE READY	EEPROM ready
24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready

Interrupt Types

- The Arduino offers 26 types of interrupts but we mostly care about 14:
    - Input Capture
        - (Timer1)
    - Overflow
        - (Timer0, Timer1, Timer2)
    - Compare Match (Output Compare Match)
        - (Timer0 A/B, Timer1 A/B, Timer2 A/B)
    - External
        - (INT0, INT1)
    - Reset
    - Watchdog

Interrupt Types (cont.)

- Input Capture
    - TIMER1_CAPT_vect
- Overflow
    - TIMER0_OVF_vect
    - TIMER1_OVF_vect
    - TIMER2_OVF_vect
- Compare Match (Output Compare Match)
    - TIMER0_COMPA_vect
    - TIMER0_COMPB_vect
    - TIMER1_COMPA_vect
    - TIMER1_COMPB_vect
    - TIMER2_COMPA_vect
    - TIMER2_COMPB_vect
- External
    - attachInterrupt()
    - detachInterrupt()

Enabling Interrupts

- Maskable Interrupt
  - An interrupt that you can enable/disable
- Non-maskable Interrupt
  - Cannot be disabled
  - Reset (the pushbutton on Arduino)
  - Watchdog Timer (automatically calls reset if processor is halted)
    - Requires WDTON bit = 0
- Globally Enable
  - Allows you to select if you want to allow any maskable interrupts to execute or not
  - Non-maskable interrupts cannot be disabled
- Locally Enable
  - Allows you to select which interrupts you want to enable specifically

Enabling Interrupts (cont.)

- Globally Enable/Disable
    - interrupts() or sei()
    - noInterrupts() or cli()
- Locally Enable
    - Use registers:
        - TIMSK0 - Bits 2, 1, 0
            - OCIE0B, OCIE0A, TOIE0
            - Output Compare B, Output Compare A, Overflow
        - TIMSK1 - Bits 5, 2, 1, 0
            - ICIE1, OCIE1B, OCIE1A, TOIE1
            - Input Capture, Output Compare B, Output Compare A, Overflow
        - TIMSK2 - Bits 2, 1, 0
            - OCIE2B, OCIE2A, TOIE2
            - Output Compare B, Output Compare A, Overflow

Write Your Interrupt

- After enabling the interrupt you must write your own ISR:

```
ISR(/* ISR name goes here */)
{
  // Write ISR code here
}
```

- ISR name should be from list of ISRs
  - Example: TIMER1_OVF_vect
- Code inside ISR should be short and quick to execute

External Interrupts

- Can only use INT0 and INT1
    - Arduino pins 2 and 3
- attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)
    - digitalPinToInterrupt(pin) selects INT0 or INT1
        - pin must equal 2 or 3
    - ISR selects the function you want to execute when interrupt occurs
    - mode selects LOW, CHANGE, RISING, FALLING
        - LOW interrupts on LOW
        - CHANGE interrupts both rising and falling edges
        - RISING/FALLING interrupts on rising/falling edge respectively

Disable External Interrupt

- Deactivates an interrupt that is on “pin”
- detachInterrupt(digitalPinToInterrupt(pin))
  - digitalPinToInterrupt(pin) selects INT0 or INT1
    - pin must equal 2 or 3

External Interrupts (cont.)

- External Interrupts are different from Input Capture Interrupts
- Input Capture capability captures external events and give them a time-stamp indicating time of occurrence
- External Interrupts do not provide time-stamping features

Attach Interrupt Sample Code

```
const byte ledPin = 13;
const byte interruptPin = 2;
// Volatile keyword needed for variables used inside of interrupt
volatile byte state = LOW;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  // Set the interrupt on pin 2 to trigger whenever a change (rising edge or falling edge) occcurs
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

void loop()
{
  digitalWrite(ledPin, state);
}

void blink()
{
  // Change the LED's state
  state = !state;
}
```

Interrupts are not perfect

- The interrupt execution response for all the enabled AVR interrupts is four clock cycles minimum
- ISRs require time to execute from start to finish
- Be aware that the Arduino takes time to:
  - Begin the ISR
  - Execute all code inside the ISR
  - Exit the ISR
  - Resume previous program
- This is not instantaneous!