ECE 372A Spring 2015 - Lecture 4

Garrett Vanhoy

September 3, 2015

Outline

I/O Ports

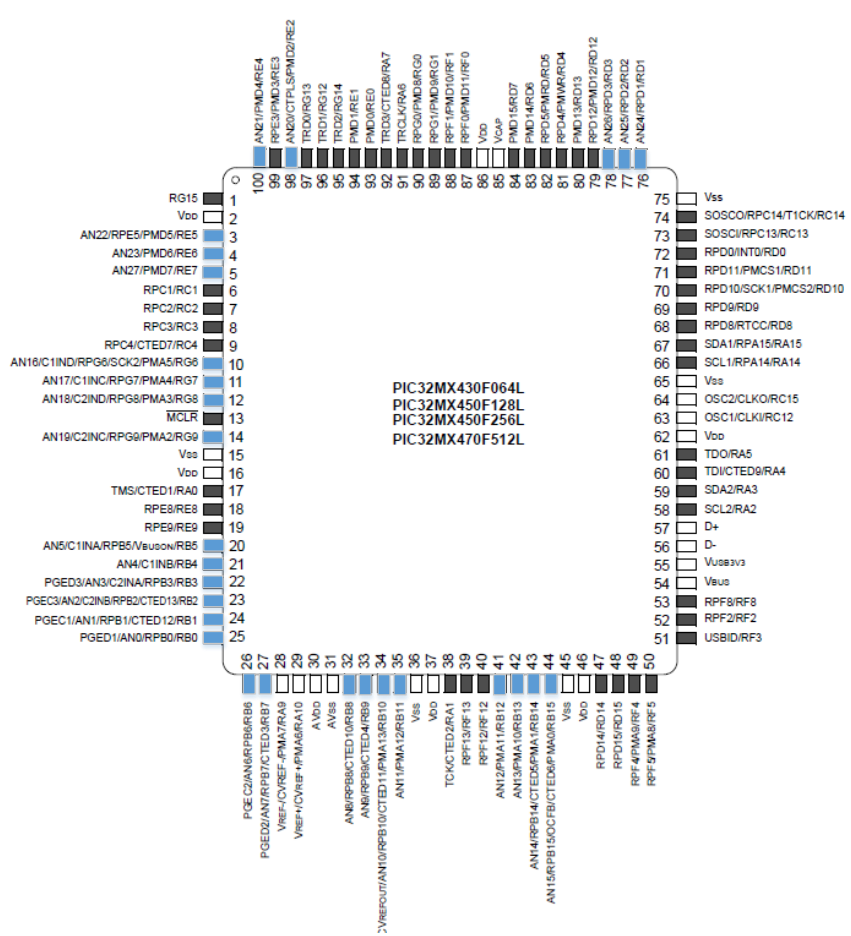
Reference Material

Section 12 in the PIC32 Family Reference Manual
Section 12 in the PIC32MX Data Sheet

Digital Input on Analog Pins

ANSELx

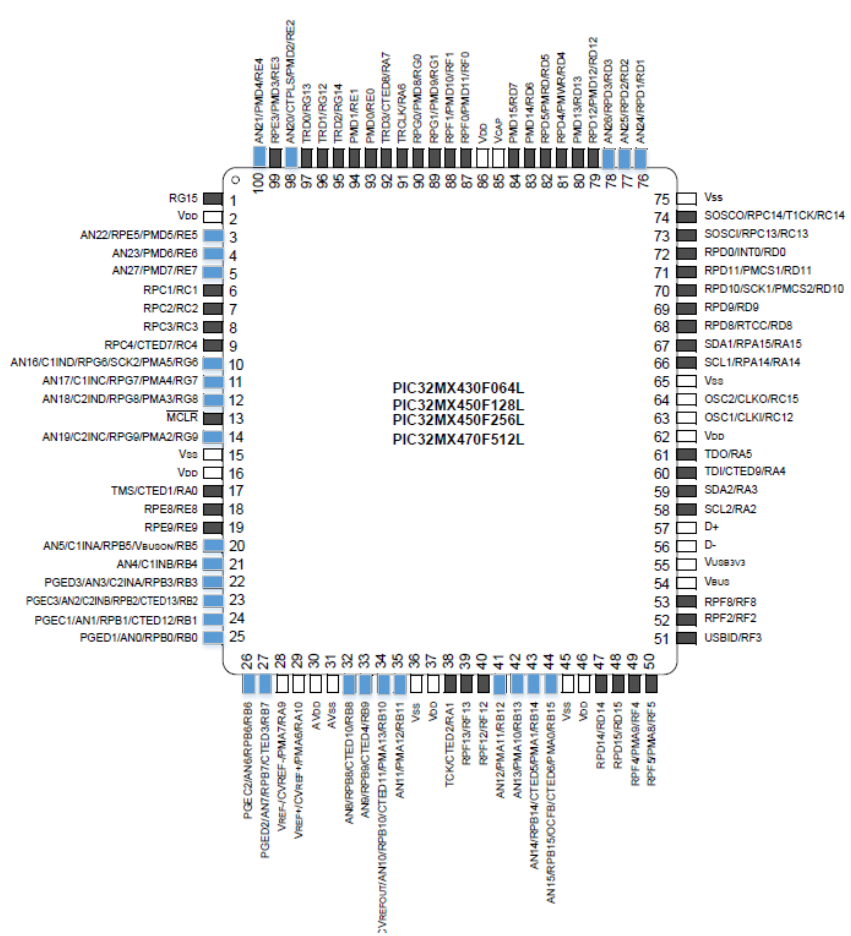
- All "ANx" pins can take analog inputs. They are in analog mode by DEFAULT.



Digital Input on Analog Pins

ANSELx

- All "ANx" pins can take analog inputs. They are in analog mode by DEFAULT.
- To use them as **digital inputs**, you must configure the ANSELx register properly.



Input Current

When Using Many Devices

- You will have many devices connected to the microcontroller.
- Motors, LEDs, LCD Display, Phototransistors, etc...

Absolute Maximum Ratings
(See Note 1)

Ambient temperature under bias	-40°C to +105°C
Storage temperature	-65°C to +150°C
Voltage on V_{DD} with respect to V_{SS}	-0.3V to +4.0V
Voltage on any pin that is not 5V tolerant, with respect to V_{SS} **(Note 3)**	-0.3V to (V_{DD} + 0.3V)
Voltage on any 5V tolerant pin with respect to V_{SS} when $V_{DD} \geq 2.3V$ **(Note 3)**	-0.3V to +6.0V
Voltage on any 5V tolerant pin with respect to V_{SS} when $V_{DD} < 2.3V$ **(Note 3)**	-0.3V to +3.6V
Voltage on D+ or D- pin with respect to V_{USB3V3}	-0.3V to (V_{USB3V3} + 0.3V)
Voltage on V_{BUS} with respect to V_{SS}	-0.3V to +5.5V
Maximum current out of V_{SS} pin(s)	200 mA
Maximum current into V_{DD} pin(s) **(Note 2)**	200 mA
Maximum output current sourced/sunk by any 4x I/O pin	15 mA
Maximum output current sourced/sunk by any 8x I/O pin	25 mA
Maximum current sunk by all ports	150 mA
Maximum current sourced by all ports **(Note 2)**	150 mA

Input Current

When Using Many Devices

- You will have many devices connected to the microcontroller.
- Motors, LEDs, LCD Display, Phototransistors, etc...
- Going over this limit will damage the device.

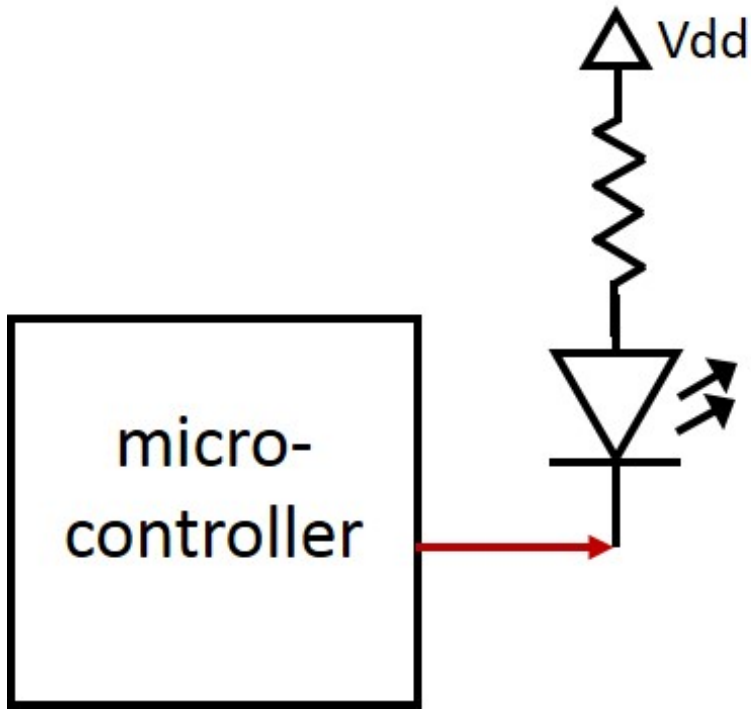
Absolute Maximum Ratings
(See Note 1)

Ambient temperature under bias	-40°C to +105°C
Storage temperature	-65°C to +150°C
Voltage on V_{DD} with respect to V_{SS}	-0.3V to +4.0V
Voltage on any pin that is not 5V tolerant, with respect to V_{SS} **(Note 3)**	-0.3V to (V_{DD} + 0.3V)
Voltage on any 5V tolerant pin with respect to V_{SS} when $V_{DD} \geq 2.3V$ **(Note 3)**	-0.3V to +6.0V
Voltage on any 5V tolerant pin with respect to V_{SS} when $V_{DD} < 2.3V$ **(Note 3)**	-0.3V to +3.6V
Voltage on D+ or D- pin with respect to V_{USB3V3}	-0.3V to (V_{USB3V3} + 0.3V)
Voltage on V_{BUS} with respect to V_{SS}	-0.3V to +5.5V
Maximum current out of V_{SS} pin(s)	200 mA
Maximum current into V_{DD} pin(s) **(Note 2)**	200 mA
Maximum output current sourced/sunk by any 4x I/O pin	15 mA
Maximum output current sourced/sunk by any 8x I/O pin	25 mA
Maximum current sunk by all ports	150 mA
Maximum current sourced by all ports **(Note 2)**	150 mA

Input Current Example

Example

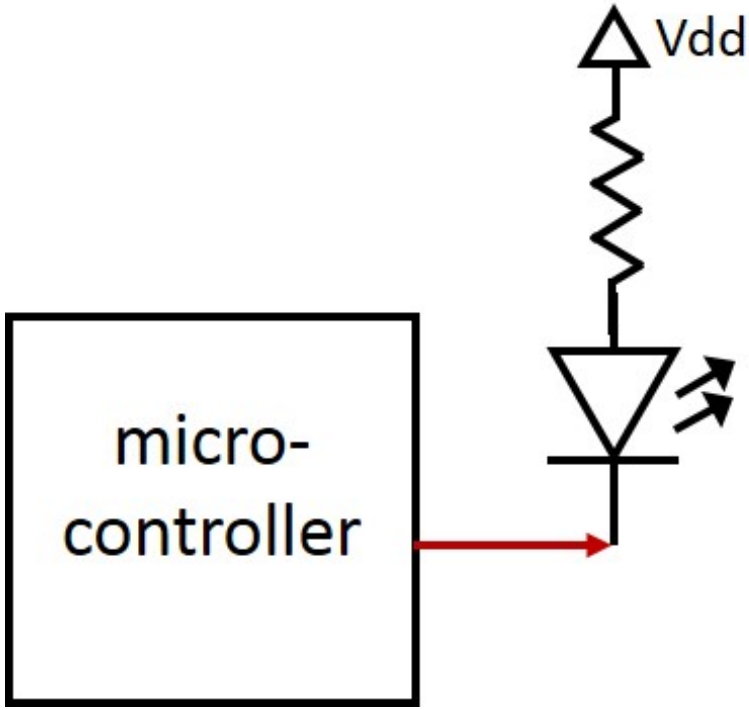
- An LED is connected to a 5V source in series with a resistor.
- What resistance should R be to limit the current? (The LED drops about 1.8 Volts)



Input Current Example

Example

- An LED is connected to a 5V source in series with a resistor.
- What resistance should R be to limit the current? (The LED drops about 1.8 Volts)
- $\frac{V_{dd}-1.8}{3.75mA} = 853$ ohms. (At the *very* least)



Outline

Purpose of Interrupts

Consider this scenario:

- Let's try to interface a keyboard with a microcontroller. What would the code look like?

Purpose of Interrupts

Consider this scenario:

- Let's try to interface a keyboard with a microcontroller. What would the code look like?
- We would use what is called "polling." We check every key as often as possible to see if something took place, then handle it.

Purpose of Interrupts

Consider this scenario:

- Let's try to interface a keyboard with a microcontroller. What would the code look like?
- We would use what is called "polling." We check every key as often as possible to see if something took place, then handle it.
- An interrupt allows us to stop execution of the main loop and handle the event.

The Execution of Interrupts

How an Interrupt Works:

- Interrupts are generated by *hardware* events.

The Execution of Interrupts

How an Interrupt Works:

- Interrupts are generated by *hardware* events.
- When the hardware event occurs, a flag variable is set in an SFR.

The Execution of Interrupts

How an Interrupt Works:

- Interrupts are generated by *hardware* events.
- When the hardware event occurs, a flag variable is set in an SFR.
- Interrupt Service Routines (ISRs) are functions in code that are executed when this flag is raised.

Interrupt Considerations

How an Interrupt Works:

1. Flags are raised *regardless* of whether interrupts are enabled.

Interrupt Considerations

How an Interrupt Works:

1. Flags are raised *regardless* of whether interrupts are enabled.
2. If an interrupt is enabled, but no ISR is implemented, your program just hangs. (No error, no output, and no execution.)

Interrupt Considerations

How an Interrupt Works:

1. Flags are raised *regardless* of whether interrupts are enabled.
2. If an interrupt is enabled, but no ISR is implemented, your program just hangs. (No error, no output, and no execution.)
3. ISRs are meant to be quick. Do long-running operations in the main loop so other interrupts can be generated.

Interrupt Considerations

How an Interrupt Works:

1. Flags are raised *regardless* of whether interrupts are enabled.
2. If an interrupt is enabled, but no ISR is implemented, your program just hangs. (No error, no output, and no execution.)
3. ISRs are meant to be quick. Do long-running operations in the main loop so other interrupts can be generated.
4. NEVER call ISRs directly.

Interrupt Considerations

How an Interrupt Works:

1. Flags are raised *regardless* of whether interrupts are enabled.
2. If an interrupt is enabled, but no ISR is implemented, your program just hangs. (No error, no output, and no execution.)
3. ISRs are meant to be quick. Do long-running operations in the main loop so other interrupts can be generated.
4. NEVER call ISRs directly.
5. Put the flag down in the ISR. If you don't, it just keeps calling the ISR.

ISR Flow

The ISR Flow

- The main loop is executing.

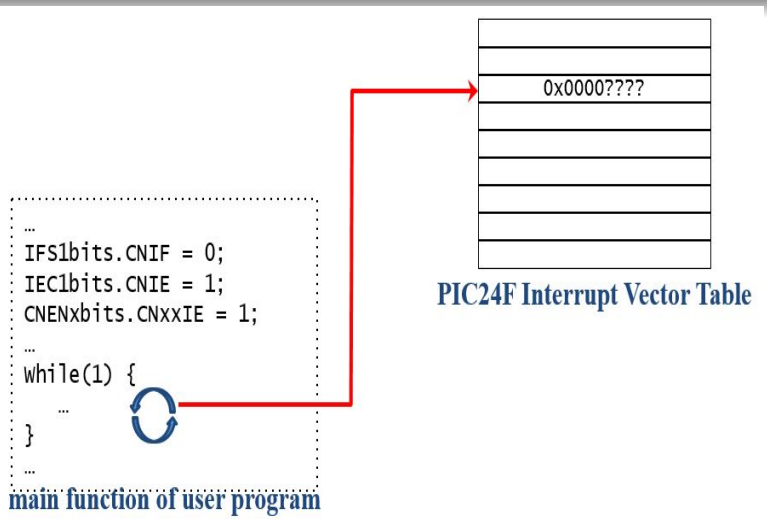
```
…
IFS1bits.CNIF = 0;
IEC1bits.CNIE = 1;
CNENxbits.CNxxIE = 1;
…
while(1) {
    …
}
…
```

main function of user program

ISR Flow

The ISR Flow

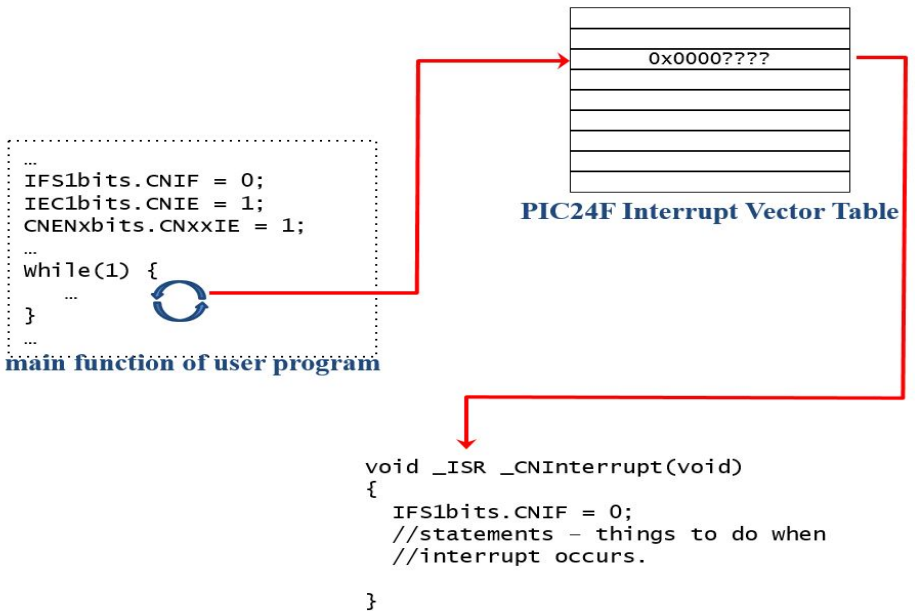
- An interrupt flag is set.
- *IF* the interrupt is enabled, the PIC24F looks to the Interrupt Vector Table for a function to call



ISR Flow

The ISR Flow

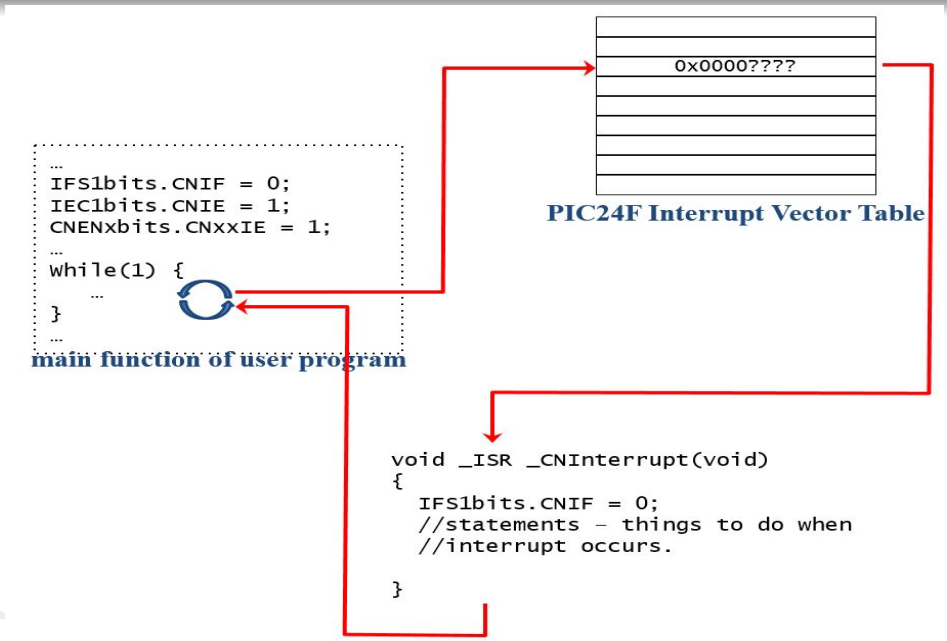
- *IF* an ISR is defined, it runs the ISR.



ISR Flow

The ISR Flow

- The flag is put down in the ISR
- Execution of the main loop resumes



ISR Flow

One Last note..

- Global variables are not accessible in the ISR.

ISR Flow

One Last note..

- Global variables are not accessible in the ISR.
- The keyword **volatile** must be used for them to be accessible in the ISR.

Outline

Using a CN Interrupt

Steps for using CN

1. Ensure that the CN pin is configured as a digital input by setting the associated bit in the TRISx register.
2. If the pin is analog, configure it as digital in ANSELx
3. Enable the overall CN functionality in CNCONx register
4. Enable interrupts for the selected CN pins by setting the appropriate bits in the CNENx registers.
5. Turn on the weak pull-up devices (if desired) for the selected CN pins by setting the appropriate bits in the CNPUx registers.
6. Clear the CNxIF interrupt flag.
7. Enable CN interrupts using the CNxIE control bit.
8. Set the interrupt priority level in IPCx registers

Notes on CN Interrupts

- **ALL** CN pins share the same ISR

Notes on CN Interrupts

- **ALL** CN pins share the same ISR
- In the CN ISR, you must find out which CN was issued.

Notes on CN Interrupts

- **ALL** CN pins share the same ISR
- In the CN ISR, you must find out which CN was issued.
- Also, two interrupts must be enabled for CN to work. (CNENx and IECx registers)

Demonstration

CN with LED Control

- Instead of polling, use CN interrupts
- Changing variables within interrupts
- Using multiple CN interrupts