CMPE-013/L

Introduction to “C” Programming

Max Dunne
Lately

Boats

$\frac{1}{3}$ of one file 3
Hardware Peripherals

Digital pins
Timers
ADC
Hardware Peripherals

- Communications
- Pin change notification
- DMA
- Output compare PWM
- Input capture
- Digital pins
- Timers
- ADC

UART SPI I²C

Direct Memory Access
Hardware Peripherals

Special function registers

- Peripherals are controlled by hardware registers
  - Referred to as Special Function Registers (SFRs)
- Memory-mapped unsigned 16-bit integers
- Accessible as global variables
  - Included from the `<xc.h>` header
Hardware Peripherals
Special function registers

- Declaration of Interrupt FlagS 0 register
- `volatile` qualifier indicates value can change outside of the code in this program
- `__attribute__` is a compiler directive to specify additional compiler parameters
  - `__sfr__` indicates that it's a memory-mapped SFR

```c
extern volatile unsigned int IFS0 __attribute__((__sfr__));
```
Hardware Peripherals

Digital pins 0 - 5V

- Voltage
  - High
  - Low

- Direction
  - Input
  - Output

- Polling interface

3.3 \times 15 =
3.3 \times 8 =
Car company

\[ N_{\text{cell}} = 100,000 \]

Error
Hardware Peripherals

Digital pins

Dedicated Port Module

RD TRISx

WR TRISx

WR LATx

WR PORTx

RD LATx

RD PORTx

I/O Cell

I/O pin

Synchronization

Gabriel Hugh Elkaim – Winter 2015

CMPE-013/L: “C” Programming
Hardware Peripherals

Digital pins

- **TRIS** – TRIState register. Sets pin direction.
  - Pin is an output when corresponding bit is 0, input when corresponding bit is 1.

- **LAT** – LATch register. Sets pin value/gets pin's desired value
  - Desired output value of the pin

- **PORT** – PORT register. Sets pin value/gets pin's actual value
  - Actual value of the pin
LAT x 0 1
Port reads 0
Hardware Peripherals

Digital pins

Dedicated Port Module

RD TRISx

WR TRISx

WR LATx
WR PORTx
RD LATx
RD PORTx

Synchronization

I/O pin

I/O Cell
Hardware Peripherals

Digital pins

Dedicated Port Module

RD TRISx

WR TRISx

WR LATx

RD LATx

RD PORTx

Synchronization

I/O Cell

I/O pin

CMPE-013/L: “C” Programming
Hardware Peripherals

Digital pins

Dedicated Port Module

I/O Cell

I/O pin

Synchronization

RD TRISx

WR TRISx

WR LATx

RD PORTx

RD TRISx

WR TRISx

WR LATx

RD PORTx

CMPE-013/L: “C” Programming
Hardware Peripherals

Timers

- Multiple 16-bit timers
  - 5 total
- Interrupt-based
  - ISR is called every X seconds
- Configurable periodicity
  - Range from 20MHz to 305Hz
Oscillator

FRC

Quartz Oscillator

PLL

8.0000000MHz  →  20MHz

laser trimmed resistors
Hardware Peripherals

Timer SFRs

• TMRx – Timer counter
  – uint16
  – Ticks every instruction clock cycle (20MHz) \( \div 2 \)

• PRx – Timer x prescalar
  – Limit for when to trigger the timer interrupt.
  – Valid values are \([1, \text{INT16}\_\text{MAX}]\)
  – 0 is a special value, disables peripheral.
\[
\frac{20 \text{ MHz}}{2} \Rightarrow \frac{2}{2} \Rightarrow \frac{2}{2}
\]
Hardware Peripherals

Timers

• To modify timer interrupt period, set PRx register.

• To set a period of the timer interrupt:
  – \( \frac{20\text{MHz}}{\text{PRx}} = \text{periodicity} \)
  – PRx of 20000 -> 1kHz interrupts
Hardware Peripherals

Timers
Hardware Peripherals

Timers

3.25 milliseconds

TMRx
Hardware Peripherals

Timers

\[ NR = 20 \times \frac{1}{20 \text{MHz}} = 50 \text{ns} \]

\[ \text{event} \]
Hardware Peripherals

Timers

PRx → TMRx → event → CPU
Hardware Peripherals

ADC

- Analog to Digital Converter
- Measures the voltage of a processor pin
- Used to read analog sensors
  - Temperature
  - Power
  - Battery levels
Hardware Peripherals

ADC SFRs

- **ADCxBUFFy**: Buffer for holding samples
  - x is the ADC
  - y is the sample [0, 7]
  - 16-bit unsigned value
    - Only lowest 10-bits matter
Hardware Peripherals

ADC

- The input signal is continuously sampled
- Every 8th sample triggers an interrupt
Hardware Peripherals

ADC

- Voltage range from $V_{\text{ref}^-}$ to $V_{\text{ref}^+}$
  - 0V to 3.3V
- Values are unsigned 10-bits, from [0, 1023]
- Units are in $V_{\text{ref}} / 1023 = 0.0032V$
Hardware Peripherals

ADC

---

guess and check

input match

---
Hardware Peripherals

ADC

ADC1BUF0  =  2
ADC1BUF1  =  146
ADC1BUF2  =  288
ADC1BUF3  =  420
ADC1BUF4  =  563
ADC1BUF5  =  691
ADC1BUF6  =  829
ADC1BUF7  =  987
Hardware Peripherals

ADC

event

ADC1BUF0 = 950
ADC1BUF1 = 600
ADC1BUF2 = 100
ADC1BUF3 = 65
ADC1BUF4 = 81
ADC1BUF5 = 93
ADC1BUF6 = 107
ADC1BUF7 = 122
Event-driven Programming

Events
Event loop
Event-driven Programming

- Real-time programming paradigm
- Build around the concept of events
- Events are then handled by specific event handlers
- Works well with systems with multiple inputs that need to be handled in a timely manner
  - Real-time system
- Integrates well with interrupts
Event-driven Programming

Events

- Any temporally-short sensor occurrence
- Usually the derivative of a signal
  - Button was pressed down
  - The mouse was clicked
  - This sensor value changed
  - This interrupt triggered
Event-driven Programming

The event loop

- A continual loop that checks for and processes events
- The core of an event-driven program

{
    while (1) {
        // Check for events

        // Process events
    }
}
Event-driven Programming

The event loop

```c
{
    while (1) {
        // Check for event 1
        // Check for event 2
        ...
        // Check for event n

        // Process event 1
        // Process event 2
        ...
        // Process event n
    }
}
```
Event-driven Programming

Event priorities

```c
{
  while (1) {
    // Check for event 3
    // Process event 3
    // Check for event 1
    // Process event 1
    // Check for event 2
    // Process event 2
  }
}
```
Event-driven Programming

Real-world example

```c
{   
    while (1) {
        if (buttonsEvent) {
            // Update fixed LED mask
        }
        if (adcEvent) {
            // Update OLED
        }
        if (timerEvent) {
            // Update bouncing LED mask
        }
        if (ledEvent) {
            // Update LEDs
        }
    }
}
```
Event-driven Programming

Real-world example

```c
static uint8_t buttonsEvent;

void main()
{
    while (1) {
        if (buttonsEvent) {
            // Event loop
        }
    }
}

void _ISR Timer1Int(void)
{
    buttonsEvent = ButtonsCheckEvents();
    IFS0 &= ~(1 << 3);
}
```