CMPE-013/L

Introduction to “C” Programming

Max Dunne
Hardware Peripherals

- Digital pins
- Timers
- ADC
Hardware Peripherals

- Communications (UART) \( I^2C \), SPI
- Pin change notification (crude)
- DMA Direct Memory Access
- Output compare
- Input capture
- Digital pins
- Timers
- ADC
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

- Voltage
  - High: 1 3.3
  - Low: 0 0
- Direction
  - Input: 1 1
  - Output: 0 0
- Polling interface
Hardware Peripherals

Digital pins

Dedicated Port Module

RD TRISx

WR TRISx

WR LATx

RD LATx

RD PORTx

Synchronization

I/O Cell

I/O pin
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

Gabriel Hugh Elkaim – Winter 2015
Hardware Peripherals

Digital pins

Dedicated Port Module

I/O Cell

I/O pin

RD TRISx

WR TRISx

WR LATx

RD LATx

RD PORTx

Synchronization

ODCx

TRISx

LATx

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

Digital pins

[Detailed diagram of digital pins and synchronization circuits]
Pic32

- Port

- \( \text{PORTESET} = 00100 \)
- \( \text{PORTECIR} = 00100 \)
- \( \text{PORTENV} = 00100 \)
Hardware Peripherals

Timers

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

Timer SFRs

- TMRx – Timer counter
  - `uint16`
  - Ticks every instruction clock cycle (20MHz)
- PRx – Timer x prescalar
  - Limit for when to trigger the timer interrupt.
  - Valid values are `[1, UINT16_MAX]`
  - 0 is a special value, disables peripheral.
Hardware Peripherals

Timers

- To modify timer interrupt period, set PRx register.
- To set a period of the timer interrupt:
  - $20\text{MHz} / \text{PRx} = \text{periodicity}$
- PRx of 20000 $\rightarrow$ 1kHz interrupts

\[1\text{ms} \quad 2\text{minutes} \quad 16\text{-bit} \quad 32\text{-bit}\]
Hardware Peripherals

Timers
Hardware Peripherals

Timers

![Graph showing hardware peripherals with TMRx and values range 0 to 65535]
Hardware Peripherals

Timers

PRx

TMRx

event
Hardware Peripherals

Timers

PRx -> TMRx -> event -> CPU
Hardware Peripherals

Timers

PRx → TMRx → event

CPU

Interrupt()
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

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

\[ 0 - 1023 \]

\[ \frac{3.3V}{1024} = \]
Hardware Peripherals

ADC

- The input signal is continuously sampled
- Every 8\textsuperscript{th} 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

Binary search

SAR ADC

Guess and check
# Hardware Peripherals

**ADC**

<table>
<thead>
<tr>
<th>ADC1BUF0</th>
<th>ADC1BUF1</th>
<th>ADC1BUF2</th>
<th>ADC1BUF3</th>
<th>ADC1BUF4</th>
<th>ADC1BUF5</th>
<th>ADC1BUF6</th>
<th>ADC1BUF7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>146</td>
<td>288</td>
<td>420</td>
<td>563</td>
<td>691</td>
<td>829</td>
<td>987</td>
</tr>
</tbody>
</table>
Hardware Peripherals

ADC

<table>
<thead>
<tr>
<th>ADC1BUF0</th>
<th>ADC1BUF1</th>
<th>ADC1BUF2</th>
<th>ADC1BUF3</th>
<th>ADC1BUF4</th>
<th>ADC1BUF5</th>
<th>ADC1BUF6</th>
<th>ADC1BUF7</th>
</tr>
</thead>
<tbody>
<tr>
<td>950</td>
<td>600</td>
<td>100</td>
<td>65</td>
<td>81</td>
<td>93</td>
<td>107</td>
<td>122</td>
</tr>
</tbody>
</table>
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
Event-driven Programming

The event loop

```c
{
    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
    }
}
```
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);
}
```