Hardware Peripherals

Digital pins
Timers
ADC
Hardware Peripherals

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

CAN, SPI, I2C, UART
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

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

Digital pins

- **Voltage**
  - High: 3.3V or 5V
  - Low: 0V or 0mA
- **Direction**
  - Input: Don’t care
  - Output
- **Polling interface**
Hardware Peripherals

Digital pins

Dedicated Port Module

I/O Cell

I/O pin

Synchronization
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
Hardware Peripherals

Digital pins
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

Q

D

Q

D

Q

CK

Q

CK
Hardware Peripherals

Digital pins

[Diagram of digital pins and peripheral connections]
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 \([1, \text{INT16\_MAX}]\)
  - 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:
  – $\frac{20 \text{MHz}}{\text{PRx}} = \text{periodicity}$

• PRx of 20000 -> 1kHz interrupts

\[
20,000,000 \rightarrow 1 \text{kHz} \\
> \text{uint16} \\
> \text{uint32}
\]

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

Timers
Hardware Peripherals

Timers

UINT16_MAX

65535

0

TMRx
Hardware Peripherals

Timers

PRx

TMRx

event
Hardware Peripherals

Timers

PRx → event → TMRx → CPU
Hardware Peripherals

Timers

void _ISR_T1_ISR(void)
{
  char *debug;
  main();
  if (debug != NULL)
    debug = NULL;
  ISR3;
  if (debug) debug = "oh no!"
}
#define 8 1000 Hz

void _ISR_T0_IRQHandler()
{
  printf("PR0: 


cpu

event

PRx

TMRx

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

\[ \text{Example: } 2.0V \rightarrow 550 \]
\[ 0.0433V \rightarrow (0, 1023) \]
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 $V_{\text{ref}} / 1023 = 0.0032V$

3.2mV
Hardware Peripherals

ADC

Sample and hold accumulator

SHA

3.3

0.0

VREFH

VREFL

SAR ADC

ADC1BUF0

ADC1BUF1

ADC1BUF2

ADC1BUF

ADC1BUF0

ADC1BUF1

ADC1BUF2

ADC1BUF

ADC1BUFF

ADC1BUF

ADC1BUFF

ADC1BUFF

ADC1BUF

ADC1BUF

ADC1BUFF

ADC1BUFF

ADC1BUF

ADC1BUF

ADC1BUFF

ADC1BUFF
Hardware Peripherals

ADC

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

ADC

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 a change (derivative)
  – 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
  }
}

// received new UART4 data
if (UART4 & 0x80) {
  // Code to process data
}
```
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
        if (...)
            // Process event 3

        // Check for event 1
        // Process event 1

        // Check for event 2
        // Process event 2
    }
}
```
Event-driven Programming

Real-world example

{ 
  while (1) {
    bool buttonsEvent;
    if (buttonsEvent) {
      // Update fixed LED mask
    }
    if (adcEvent) {
      // Update OLED
    }
    if (timerEvent) {
      // Update bouncing LED mask
    }
    if (ledEvent) {
      // Update LEDs
      ledEvent = false;
    }
  }
}
static uint8_t buttonsEvent;

void main()
{
    while (1) {
        if (buttonsEvent) {
            // Event loop: Button was pressed or released
        }
    }
}

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

Bit masking
Bit flags
Bit fields
Bit manipulation

Bit packing

• Data is commonly packed into larger unsigned integers on embedded systems

• Generally a tie in to hardware or when space is critical
  – Hardware
  – Storage
  – Binary formats
Bit manipulation

Bit packing

C1CTRL1 – dsPIC33EP256MC502