CMPE-013/L

Introduction to “C” Programming

Max Dunne
Hardware Peripherals

- Digital pins
- Timers
- ADC
Hardware Peripherals

- Communications \textit{UART}, \textit{SPI}, \textit{I^2C}
- Pin change notification
- DMA \textit{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

\[ \text{MIPS4K} \]
\[ n \leq 16 \]
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 → 2.7 - 3.3
  - Low: 0 → 0 - 0.5

- Direction
  - Input
  - Output

- Polling interface
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
A

Ports

---
Hardware Peripherals

Digital pins

Dedicated Port Module

RD TRISx

WR TRISx

WR LATx
WR PORTx
RD LATx
RD PORTx

Synchronization

I/O Cell

I/O pin
Hardware Peripherals

Digital pins

Dedicated Port Module

RD TRISx

WR TRISx

WR LATx

RD PORTx

I/O Cell

I/O pin

Synchronization

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
Hardware Peripherals

Timer SFRs

- **TMRx** – Timer counter
  - `uint16`
  - Ticks every instruction clock cycle (20MHz)

- **PRx** – Timer x prescaler
  - Limit for when to trigger the timer interrupt.
  - Valid values are `[1, 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:
  - $20\text{MHz} / \text{PRx} = \text{periodicity}$
- PRx of 20000 -> 1kHz interrupts

\[
\frac{20\text{MHz}}{1, 2, 4, 8, \ldots, 256}
\]
Hardware Peripherals

Timers
Hardware Peripherals

Timers

65535

TMRx
Hardware Peripherals

Timers

PRx

TMRx

event
Hardware Peripherals

Timers

PRx → event → TMRx → CPU
Hardware Peripherals

Timers

PRx → 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

\[
\frac{3.3V}{1024} = 0.00322V
\]

0 - 1023
0 - 3.3V
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

[Diagram of an ADC with labels and connections]
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

- **RTOS**
  - 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
    }
}
```
```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 (buttonEvent) {
            // 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);
}
CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
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
Bit manipulation

Bit masks

Example

// Abort the current CAN message transmission
ClCTRL1 = ClCTRL1 | 0x1000;

ClCTRL1 |= 0x1000;
Bit manipulation

Example

// Disable CAN message timestamping
C1CTRL1 = C1CTRL1 & 0xFFF7;
Bit manipulation

Bit masks

Example

// Disable CAN message timestamping
ClCTRL1 &= ~(1 << 3);

BIT_0
BIT_1
Bit manipulation

Bit masks

- A constant that indicates which bits are relevant for a given variable
- One bits indicate significant bits
- Zero bits indicate ignore bits
Bit manipulation

Bit masks

```
#define CxCTRL1_MASK_CANCAP (1 << 3)

// Disable CAN message timestamping
C1CTRL1 &= ~CxCTRL1_MASK_CANCAP;
```

Example
Bit manipulation

Bit masking

• Setting a bit
  – ORing with 1
    \[ C1CTRL1 \ |= \ CxCTRL1\_MASK\_CANCAP; \]

• Clearing a bit
  – ANDing with 0
    \[ C1CTRL1 \ &= \ \sim CxCTRL1\_MASK\_CANCAP; \]

• Toggling a bit
  – XORing with 1
    \[ C1CTRL1 \ ^= \ CxCTRL1\_MASK\_CANCAP; \]
Bit manipulation

Bit masking

- Setting a bit can OR multiple masks together

```
enum {
    BUTTON_EVENT_1UP = 0x01,
    BUTTON_EVENT_2UP = 0x04
};

{  
    uint8_t event = BUTTON_EVENT_1UP | BUTTON_EVENT_2UP;
}
```
Bit manipulation

Bit masking

- Getting a bit
  - ANDing with 1

Example

```c
#define CxCTRL1_MASK_CANCAP (1 << 3)

// If CAN message timestamping is enabled
if ((CxCTRL1 & CxCTRL1_MASK_CANCAP) == CxCTRL1_MASK_CANCAP) {
    ...
}
```
Bit manipulation

Bit masking

• Getting a bit
  – ANDing with 1

Example

```
#define CxCTRL1_MASK_CANCAP (1 << 3)

// If CAN message timestamping is enabled
if (C1CTRL1 & CxCTRL1_MASK_CANCAP) {
    ...
}
```
Bit manipulation

Bit masking

Example

```c
// Retrieve the operating mode of the CAN hardware
int opmode = (C1CTRL1 & 0xBE0) >> 5;
```
**Bit Fields**

**Definition**

Bit Fields are (unsigned) int members of structures that occupy a specified number of adjacent bits from one to sizeof(int). They may be used as an ordinary int variable in arithmetic and logical operations.

- Bit Fields:
  - Are ordinary members of a structure
  - Have a specified bit width
  - Provide bit access to a variable without masking operations
Bit Fields

- Bit Fields:
  - May only be integers (short, long, __, long long)
    - No larger than the base type
  - Unsigned by default, but may be signed
  - Non-portable across architectures/compilers!
    - Just like regular structs
Bit Fields

How to Create a Bit Field

Syntax

```
struct StructName {
    ((un)signed) int memberName1: bitWidth;
    ...
    ((un)signed) int memberNameN: bitWidth;
}
```

Example

```
struct ByteBits {
    unsigned int a: 1;
    long b: 1;
    short c: 2;
    unsigned d: 1;
    long long e: 3;
};
```
Bit Fields
How to Use a Bit Field

Example

typedef struct {
    unsigned int a: 1;
    long b: 1;
    short c: 2;
    unsigned d: 1;
    long long e: 3;
} ByteBits;

ByteBits x;

bitfield struct may be declared normally or as a typedef
Bit Fields
How to Use a Bit Field

Example

```c
struct ByteBits {
    unsigned a: 1;
    unsigned b: 1;
    unsigned c: 2;
    unsigned d: 1;
    unsigned e: 3;
} x;

int main(void)
{
    x.a = 1; // x.a may contain values from 0 to 1
    x.b = 0; // x.b may contain values from 0 to 1
    x.c = 0b10; // x.c may contain values from 0 to 3
    x.d = 0x0; // x.d may contain values from 0 to 1
    x.e = 7; // x.e may contain values from 0 to 7
}
```
Bit Fields

Microchip's SFRs

Example

```c
// SFR register declaration
extern volatile unsigned int C1CTRL1 __attribute__((__sfr__));

// SFR bitfield declaration
typedef struct {
    unsigned WIN : 1;
    unsigned    : 2;
    unsigned CANCAP : 1;
    unsigned    : 1;
    unsigned OPMODE : 3;
    unsigned REQQP : 3;
    unsigned CANCKS : 1;
    unsigned ABAT  : 1;
    unsigned CSIDL : 1;
} C1CTRL1BITS;
extern volatile C1CTRL1BITS C1CTRL1bits __attribute__((__sfr__));
```
Bit Fields
How to Use a Bit Field

Example

```c
int main(void)
{
    // Abort the current CAN message transmission
    ClCTRL1 |= 0x1000;
    ClCTRL1bits.ABAT = 1;

    // Disable CAN message timestamping
    ClCTRL1 &= 0xFFF7;
    ClCTRL1bits.CANCAP = 0;

    // If CAN message timestamping is enabled
    if (ClCTRL1 & 0x0008) {
        if (ClCTRL1bits.CANCAP) {
            ...
        }
    }
}
```
Bit Fields
Signed values

Example

typedef struct {
    signed int a: 3;
    short b: 2;
    signed short c: 2;
    long long d: 3;
} ByteBits;

ByteBits x;
Bit Fields

Signed values

Example

typedef struct {
    signed int    a: 3;
    short         b: 2;
    signed short  c: 1;
    long long     d: 3;
} ByteBits;

ByteBits x;
Bit Fields
Maximum bitness

Example

typedef struct {
    signed int   a: 3;
    short        b: 2;
    signed short c: 1;
    long long    d: 3;
} ByteBits;

ByteBits x;
Bit Fields
Maximum bitness

Example

typedef struct {
    signed short a: 3;
    short b: 2;
    signed short c: 1;
    short d: 3;
} ByteBits;

ByteBits x;
CMPE-013/L

RPN (stack)

Maxwell James Dunne
Stack Operation
struct Stack {
    float stackItems[STACK_SIZE];
    int currentItemIndex;
    uint8_t initialized;
};
Stack Functions

- void StackInit(struct Stack *stack)
  
  struct Stack *stack;
  StackInit(&stack);
  stack->initialized = TRUE;

- int StackPop(struct Stack *stack, float *value);
RPN

• Reverse Polish Notation
  – Calculator that uses a stack as a scratchpad
  – We will read strings from the keyboard and parse

• Simple Examples
  – 4 3 + = 4 + 3
  – 1 6 -3 - + =
RPN

- Reverse Polish Notation
  - Calculator that uses a stack as a scratchpad
  - We will read strings from the keyboard and parse

- Simple Examples
  - \( 4 \ 3 \ + = 4 + 3 \)
  - \( 1 \ 6 \ -3 \ - + = 6 - 3 \)
RPN

Error/End Conditions

• The RPN calculation is complete when the whole string has been processed.
• If only one item is left on the stack the string entered was valid.
• It can also fail in a few ways
  – Not one Item left on stack at end
  – Invalid RPN Token
  – Operator without at least 2 items on the stack
  – Stack is full (Specific to the class, not RPN in general)
RPN
Complex Example

- \((1 + 4) \times (6 - 4) \div 8\) as \("1 4 + 6 4 - * 8 / \)
Development Tips

\texttt{atof("3.0") \rightarrow 3.0}

\texttt{atof("0") \rightarrow 0.0}

\texttt{atof("+") \rightarrow 0.0}

\texttt{len("+")}
"Hello\0 the dog"

\( a = \text{strtok}(\text{"Hello\0\0 the dog"}, \text{"\0\0"}) \)

\( b = \text{strtok}(\text{"NULL,"}, \text{"\0\0"}) \)

" Hello\0\0 the dog!"

\( a! = \text{NULL} \)
Is the Friday lab section full?

1:40

Stock: a;

a: (II)