UNIVERSITY OF MICROCHIP
UofM
MASTERs 2007

11029 CCR

Using the CCS C Compiler for Rapid Development of Microcontroller Applications

Class Objective

When you finish this class you will:

- Be able to use your C knowledge and write applications using CCS C
- Know the ways the CCS C Compiler speeds up development time
- Develop new application and coding ideas

Agenda

- **Overview**
  - Design Goals
  - Compiler Products
- **Compiler Methodology**
  - Differences to ‘Standard’ C
  - Memory Allocation and Data Types
  - Built-In Functions Overview
  - Inline Assembly

Agenda (Continued)

- **Programming Details**
  - Required Setup
  - Interrupts
- **Built-In Functions**
  - GPIO
  - Delays
  - Serial (I^2C™, SPI, Asynchronous)

Agenda (Continued)

- **Hands-On Lab**
  - EX1: 0 to blinky LED in 60 seconds
  - EX2: The CCS IDE Debugger
  - EX3: Stopwatch using interrupts
  - EX4: A/D with Fixed Point Decimal
  - EX5: Write to program memory

Design Goals

- Ease of use - compile and go!
- Quick development time (with built-in functions and extensive examples)
- Easy PIC® MCU to PIC MCU migration
- Optimization comparable vs. assembly

CCS Command Line Compiler

- **Product Line:**
  - PCB – Baseline PIC® MCUs
  - PCM – Mid-Range PIC MCUs
  - PCH – PIC18 Family PIC MCUs
  - PCD – PIC24/dsPIC30/dsPIC33 Family
- **Integrates into MPLAB® IDE**
- **Linker**
  - Will not create relocatable objects
- **Linux and Windows versions**

CCS Windows IDE

- **Product Line:**
  - PCW – Baseline + Mid-Range
  - PCWH – All 8-bit PIC® MCUs
  - PCWHD – All PIC MCUs
- **Advanced Windows IDE**
  - C Aware Editor
  - Project Management
  - Debugging (using a CCS ICD)

CCS Windows IDE Features

- Integrated RTOS
- Linker and Re-locatable objects
- Document Generator
- Statistics
- Flow Chart Editor

CCS Support Policy

- **E-Mail / Phone Support**
  - Duration is Unlimited!
- **Updates (bug fixes, new devices)**
  - Requires Maintenance Contract
  - Maintenance can be extended at any period
- **Large Customer Base**
- **Rapid Releases**
  - Average once per week

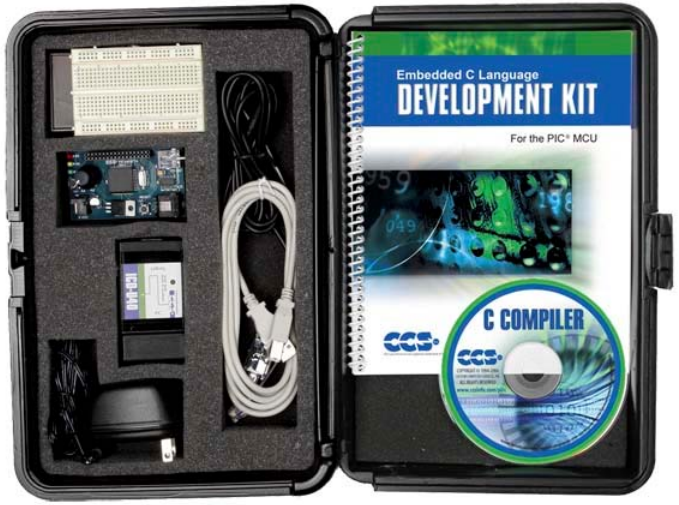
CCS ICD

- **Comparable to a Microchip ICD2**
- **Two varieties**
  - ICD-S = Serial
  - ICD-U = USB

- **Control Software for ICP**
  - Windows and Linux
- **The CCS PCW IDE for debugging**
- **ICD cable is compatible with Microchip ICD2**

CCS Development Kits

- **A combined package that gives a user:**
  - C Compiler
  - Prototyping Board
  - ICD
  - Tutorial Book and Examples
  - Relevant Parts and Cables



Compiler Methodology

Differences to 'Standard' C

- **Integer size for 8-bit PIC® MCUs:**
  - int is 8-bit
  - long is 16-bit
  - #type can be used to change default size
    - **`#TYPE SHORT=8, INT=16, LONG=32`**
  - int32 is provided for a 32-bit data type
- **Data is unsigned by default**
- **Float is PIC14000 MCHP (not IEEE)**

Differences to 'Standard' C

- **Case in-sensitive**
  - #case will make it case sensitive
- **Variables not initialized to zero**
  - #zero_ram inits all variables to 0
  - static variables are init to 0
- **const places variable into ROM**
- **#device ANSI**

ANSI Non-Compliance

- **fopen(), fclose(), etc**
- **printf() formatting string must be known at compile time**
- **Recursion not allowed**
- **No <time.h>**
  - Rest of Standard C library is provided

RAM Allocation

- **RAM is allocated in this order:**
  - Globals
  - Call tree analyzed
  - First come, first serve
- **Call tree determines which RAM can be reused**
- **User can force RAM placement**
- **Structure/Array cannot be larger than bank**

Data Types

- **Standard**
  - short int – 1-bit / boolean
  - char – unsigned 8-bit
  - int – unsigned 8-bit
  - long – unsigned 16-bit
  - float – MCHP PIC14000 32-bit float format
- **Non-Standard**
  - int1 – 1-bit / boolean
  - int8 – unsigned 8-bit
  - int16 – unsigned 16-bit
  - int32 – unsigned 32-bit
  - __address__ - 16 or 32-bits, depends on architecture
  - _fixed – Fixed point decimal

Constant Data

- **Two qualifiers: ‘const’ and ‘rom’**
- **const qualifier has two definable modes:**
  - #device CONST=ROM ***DEFAULT***
  - #device CONST=READ_ONLY
- **rom qualifier places variable in ROM**
- **Examples**
  - `const int lookup[16]={0...15};`
  - `const char string[]=“Hello”;`
  - `rom char *cptr = “Hello”;`

Fixed Point Decimal

- **Represent decimal numbers with integers, instead of floats**
  - Faster, Smaller
  - 100% precision
- **Example Declaration:**

```
[type] _fixed(y) [declarator]
int16 _fixed(2) money;
```

- **Supported by printf()**

Fixed Point Examples

- `int16 _fixed(2) money;`
  - Range: 0.00 to 655.35
- `money = 20.50;`
- `money += 5;  //adds 5.00`
- `money += value;`
- `printf("%w", money);`

Reference Parameters

- **Pass address, not value.**
  - More efficient for large structures
  - Values can be changed in function
- **Declare reference parameters with &**

```
void Inc(int &i) {
   i++;
}
```

- **Many times compiler will inline**

Default Parameters

- **Default value passed to function if no value specified**

```
int1 Get(char *c, int time=200);
```

  - Prototype

```
Get(&c);
```

  - time will be 200

```
Get(&c, 500);
```

  - time will be 500

Function Overloading

- **Multiple functions, same name**
- **Different parameters**
- **Example:**

  Three different functions called Inc
  - int Inc(int *i);
  - long Inc(long *l);
  - float Inc(float *f);

University of Microchip UofM MASTERs 2007

In-Line Assembly

- **#asm**
  - Starts an assembly block
- **#endasm**
  - Ends an assembly block
- **_return_**
  - Assign return value
  - May be corrupted by any C code after #endasm
- **Supports all opcodes**
- **C variables can be accessed**
- **Automatic banking**
  - #asm ASIS - disables auto-banking

```
int find_parity (int data)
{
   int count;
 #asm
   movlw   0x8
   movwf   count
   movlw   0
  loop:
   xorwf   data,w
   rrf     data,f
   decfsz  count,f
   goto    loop
   movlw   1
   addwf   count,f
   movwf   _return_
 #endasm
}
```

Built-In Functions and PIC® MCU Migration

Built-In Functions

- **CCS provides functions to control PIC® MCU peripherals**
  - Serial
  - A/D Conversion
  - Timers
- **Internal to compiler, not linked**
- **Simplify PIC MCU to PIC MCU migration**
- **Follow any errata workarounds**

Built-in Functions, Drivers and Examples

- **Drivers provided for complex features or external peripherals**
  - EEPROMs, LCDs, USB, FAT, etc
- **Many example programs provided to showcase built-in functions/drivers**
- **Always kept up to date with latest peripherals**
- **Before you start a project, examine the libraries and example programs CCS provides you to reduce your development time!**

Easy PIC® MCU Migration

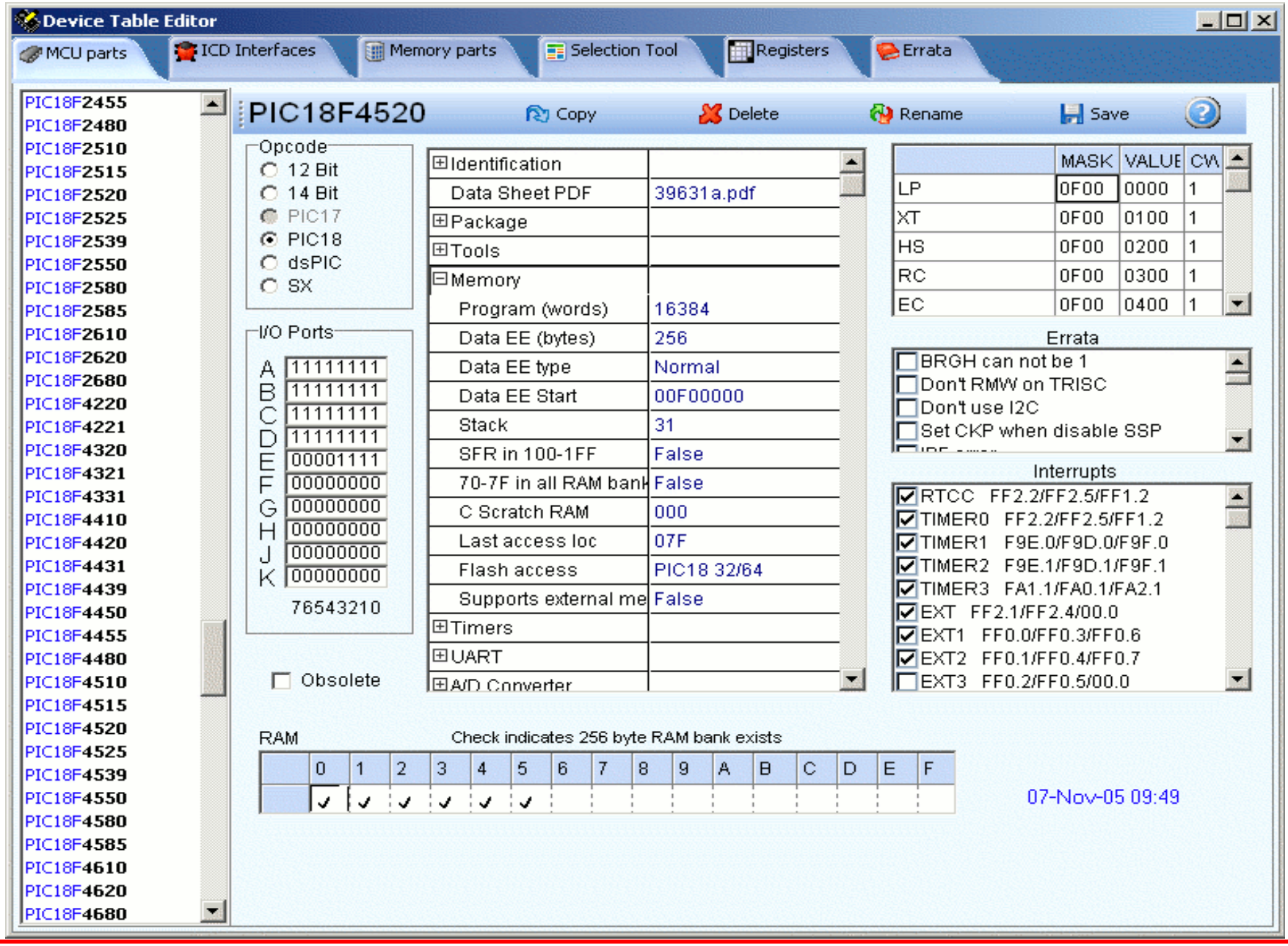
```
#if defined(__PIC18__)
  #include <18F4520.h>
#elif defined(__PIC14__)
  #include <16F877A.h>
#endif

#fuses HS, NOWDT, NOLVP
#use delay(clock=20000000)

Void main(void) {
   while(TRUE) {
      output_toggle(PIN_C0);
      delay_ms(500);
   }
}
```

getenv()

- **Assists in making portable code**
- **Returns chip information**
  - Program Memory Size
  - Bulk Erase Size
  - Configuration Bits Set
- **Options can be conditionally compiled:**

```
#if getenv("ADC_CHANNELS") > 0
  setup_adc_ports(NO_ANALOGS);
#endif
```

Device Database and Editor



Programming Details

Required Setup

Pre-processor

- **Design Goal: All in front of you**
  - Everything is in the source code
  - No linker scripts
  - Small applications easy to implement
- **Pre-processor commands change compiler-behaviour:**
  - Methodology
  - ROM/RAM placement
  - Library configuration

#device

- **Loads personality for the target PIC® MCU**
  - Example: #device PIC18F4520
- **Can be used to alter low-level specifics**
  - #device ICD=TRUE
- **PIC MCU header files have #device**
- **If creating multiple compilation units (linking), must be defined in each file.**
  - Put it in a common include file

#fuses

- **Define the configuration bits**
  - #fuses HS, NOWDT, NOLVP
- **To get list of valid fuses:**
  - In IDE, use View Valid Fuses window
  - Header file for the device has list
- **To get a description of each fuse:**
  - In IDE, use View Valid Fuses window
  - fuses.txt

#use delay

- **Configure the built-in libraries for the clock speed for your target**
  - #use delay(clock=value)
- **Value is system clock speed, after PLL**
- **Only needed if using the following:**
  - Serial
  - Delay
- **For multiple clock speed applications, you can define this more than once**

Programming Details

Interrupts

Interrupt Service Routine

- **CCS C provides an ISR**
  - Saves status and scratch registers
  - Checks enable/flags, goes to user ISR function for that flag
  - Clears interrupt flag (if needed)
  - Restores status/scratch registers

Interrupt Service Routine API

- **enable_interrupts(INT_*)**
  - enable_interrupts(GLOBAL)
  - enable_interrupts(INT_TIMER0)
- **disable_interrupts(INT_*)**
- **clear_interrupt(INT_*)**
- **interrput_active(INT_*)**

Interrupt Service Routine API

- **#int_***
  - The following function will be processed for that interrupt

```
#int_timer0
void isr_timer0(void)
{
  //HANDLE TIMER0 OVERFLOW
}
```

Interrupt Service Routine Special Identifiers

- **#int_global**
  - Function overrides CCS ISR
  - User must save and restore all registers altered by their ISR (W, STATUS, etc.)
- **#int_default**
  - Traps unknown interrupt

Interrupt Service Routine Options

- **The following keywords may follow #int_xxxx :**
  - high – (PIC18) Interrupt is high priority
  - fast – (PIC18) Same as high, but it won't save/restore registers
  - noclear – ISR will not clear interrupt flags
- **#device HIGH_INTS=TRUE**
  - Must be set to use high/fast interrupt

Programming Details

Built-In Functions

Built-In Functions
General Purpose Input / Output

- **output_high(PIN_XX)**
- **output_low(PIN_XX)**
  - Sets the pin to desired level
  - PIN_XX (ex PIN_C0, PIN_B5, etc) are defined in the device header file
- **output_toggle(PIN_XX)**
  - Toggle high/low state of the pin
- **bool=input(PIN_XX)**
  - Read value of pin

Built-In Functions
General Purpose Input / Output

- **output_X(value)**
  - Sets the port X (A to J) to the desired value
- **byte=input_X()**
  - Read value of port X
- **set_tris_X(value)**
- **val = get_tris_X()**
  - Get/Set the tristate setting

Built-In Functions
General Purpose Input / Output

- **#use standard_io(X)**
  - Output functions set TRIS to output
  - Input functions set TRIS to input
  - This is the default operation
- **#use fast_io(X)**
  - Compiler does not alter TRIS
- **#use fixed_io(port_outputs=pins)**
  - #use fixed_io(d_outputs=PIN_D7)

Built-In Functions
Delays

- **delay_cycles(x)**
- **delay_us(x)**
- **delay_ms(x)**
  - Uses a series of loops and NOPs
  - Timing based on #use delay()
  - Multiple #use delay() allowed

```
#use delay(clock=125000)

//1 second delay @ 125kHz
void Delay1s125khz(void) {
  delay_ms(1000);
}

#use delay(clock=8000000)

//1 second delay @ 8MHz
Void Delay1s8Mhz(void) {
  delay_ms(1000);
}
```

UART

- **Powerful UART library built into compiler**
- **#use rs232(baud=xxxx, xmit=PIN_XX, rcv=PIN_XX, stream=yyyyy)**
  - TX/RX pins can be any pins
  - Many more options exist
    - **Enable bit, parity, collision detection, open-collector mode, etc.**
  - Timing is based on #use delay()

UART (Continued)

- **Standard C I/O, last #use rs232()**
  - printf("string", …)
  - char=getc()
  - putc(char)
- **Standard C I/O, streams:**
  - fprintf(stream, "string", …)
  - char=fgetc(stream)
  - putc(char, stream)
- **bool=kbhit(stream)**
- **setup_uart(newBaud, stream)**
  - Dynamically change the UART

Dual UART Example

```
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7,
           stream=HW_UART)

#use rs232(baud=9600, xmit=PIN_D0, rcv=PIN_B0,
           stream=SW_UART, disable_ints)

fprintf(HW_UART, “HELLO HARDWARE UART\r\n”);

fprintf(SW_UART, “HELLO SOFTWARE UART\r\n”);

if (kbhit(HW_UART) {
   c=fgetc(HW_UART);
   fputc(c,SW_UART);
}
if (kbhit(SW_UART) {
   c=fgetc(SW_UART);
   fputc(c,HW_UART);
}
```

printf() Redirection

- printf() can be redirected to a user-defined function
- Example:

```
void LCDPut(char c);
printf(LCDPut, "%U", val);
```

I²C™ Library

- **Multiple I²C channels on any I/O pins**
- **#use i2c(master, sda=pin_XX, scl=pin_XX, address=YY, stream=id)**
  - The SDA and SCL pins can be any pins
  - Slave Mode is HW MSSP only
  - Address Is only needed in slave mode

I²C™ Library (Continued)

- **i2c_start()**
  - Can also be used to send a re-start signal
- **char=i2c_read(ack)**
  - Ack is an optional parameter
- **i2c_write(char)**
- **bool=i2c_available()**
  - Can only be used with a hardware MSSP
- **i2c_stop()**

SPI

- **Configurable SPI library**
- **#use spi(parameters)**
  - HW or SW pins
  - Multiple streams
  - Clock rate and clock mode configurable
- **in = spi_xfer(STREAM_SPI, out)**

A/D Converter

- **#device ADC=X**
  - Assign the ADC result size
- **setup_adc(mode)**
  - Configure and the ADC; mode varies for each type
- **setup_adc_ports(config)**
  - Configure the pins for analog or digital mode

A/D Converter (Continued)

- **set_adc_channel(channel)**
  - Set the channel for subsequent reads
- **val = read_adc(mode)**
  - ADC_START_ONLY – Start a conversion
  - ADC_READ_ONLY – Read last conversion
  - ADC_START_AND_READ – Full cycle
- **adc_done()**
  - Returns TRUE if conversion is done

A/D Converter Example

```
//INIT
#device ADC=10
setup_adc(ADC_CLOCK_INTERNAL );
setup_adc_ports(AN0);

//APPLICATION
int16 result;
set_adc_channel(0);
read_adc(ADC_START_ONLY);
   /* do something while A/D is busy */
while (!adc_done());
result=read_adc(ADC_READ_ONLY);
```

Internal Data EEPROM

- Access internal data EEPROM
- v = read_eeprom(address)
- write_eeprom(address, v)

Read / Write Program Memory

- **Treat program memory like data EEPROM.**
- **v=read_program_eeprom(a)**
- **write_program_eeprom(a, v)**
  - Size of v depends on architecture.
- **read_program_memory(a, ptr, num)**
- **write_program_memory(a, ptr, num)**
- **erase_program_memory(a)**

What's Next

Hands-On Lab Agenda

- **After Break, Hands-On Lab**
  - IDE and Project Wizard
  - Integrated Debugger
  - Interrupt Service Routine
  - A/D Converter
  - Write/Read Program Memory

‘Advanced’ Class

- **Attend the ‘advanced’ class for more**
  - New features since last year
  - ROM / RAM Placement
  - addressmod
  - RTOS

Q & A

UNIVERSITY OF MICROCHIP
UOFM
MASTERS 2007

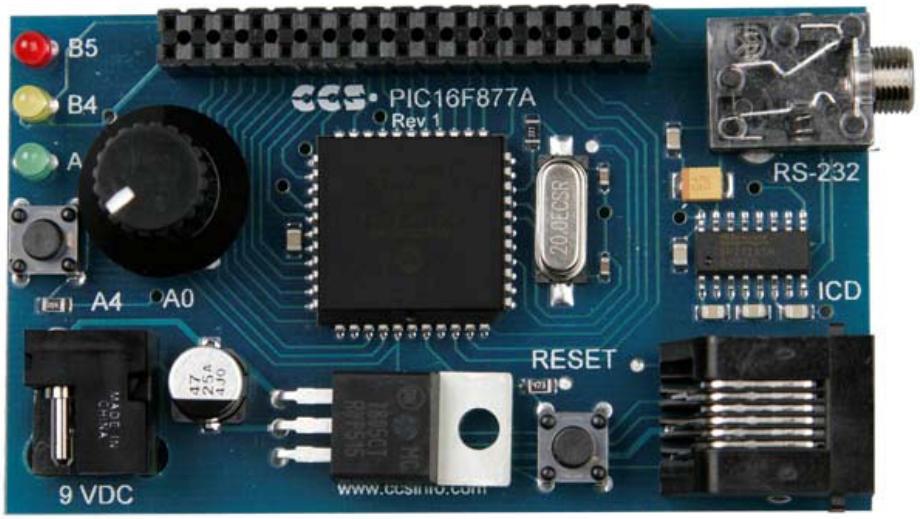
Hands-On Lab

Hands-On Lab

- EX1: 0 to blinky LED in 60 seconds
- EX2: The CCS IDE Debugger
- EX3: Stopwatch using interrupts
- EX4: A/D with Fixed Point Decimal
- EX5: Write to program memory

Prototyping Hardware

- **CCS 16F877A**
  - ICD Connection
  - Pot on AN0
  - Button on RA4
  - LEDs on RA5, RB4 and RB5
  - RS232 driver and mini-connector
  - Header to all GPIO



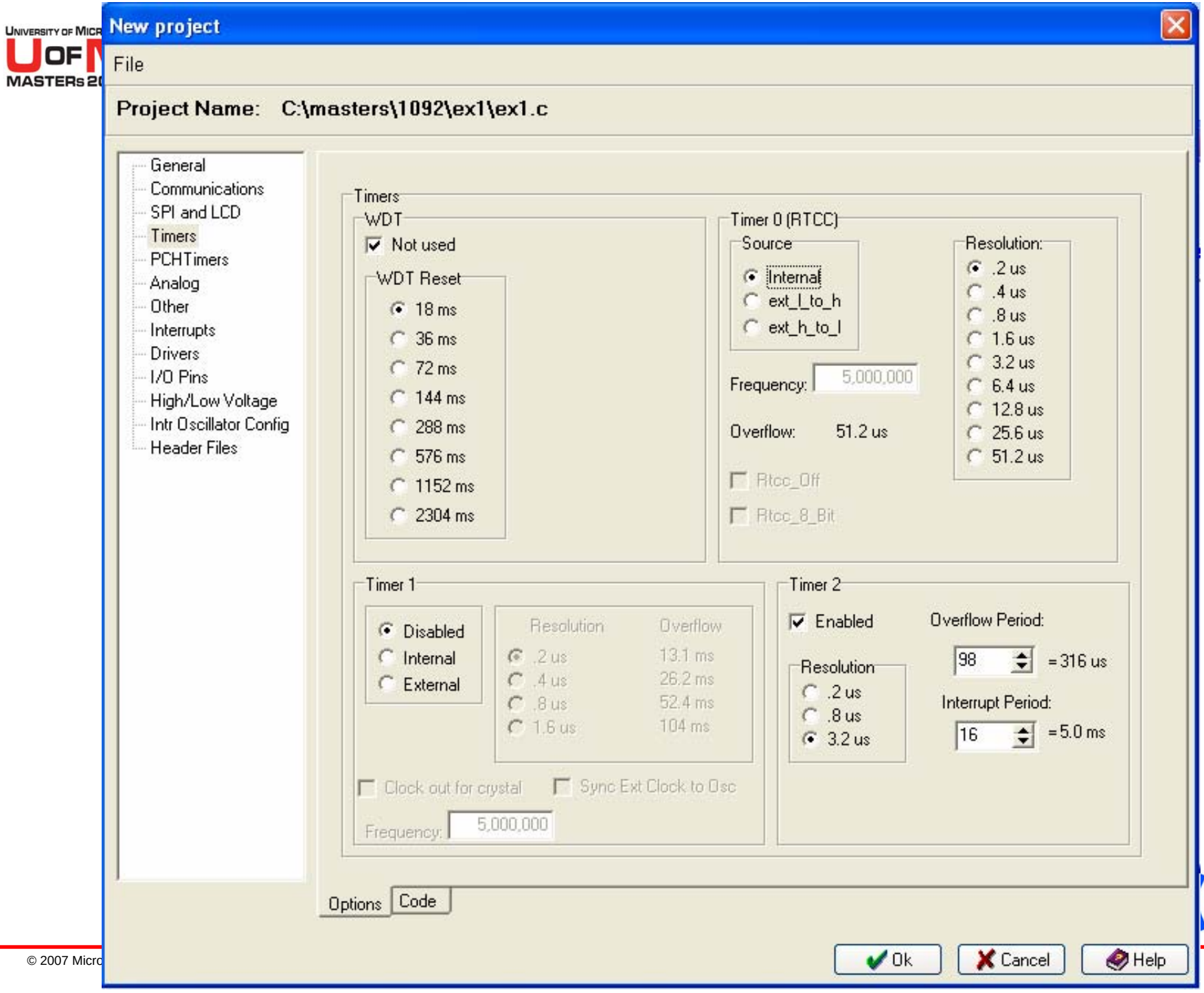
UNIVERSITY OF MICROCHIP
UofM
MASTERs 2007

Hands-On Lab

Lab 1: Create a new project with the Project Wizard

Lab 1: Project Wizard
Project Wizard Overview

- Creates an initial framework
- Assists in the initial configuration of internal peripherals
- Configuration screen for all internal peripherals
- Windows IDE Only

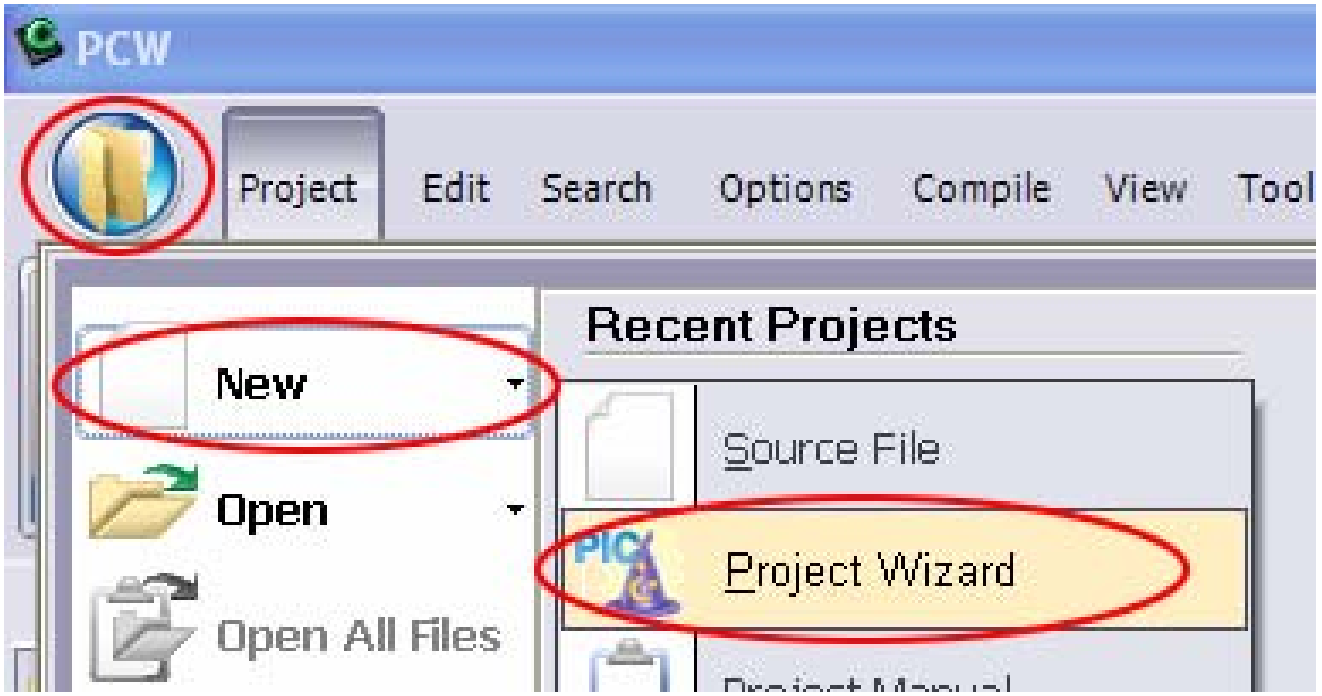

New project
File
Project Name: C:\masters\1092\ex1\ex1.c
General
Communications
SPI and LCD
Timers
PCHTimers
Analog
Other
Interrupts
Drivers
I/O Pins
High/Low Voltage
Intr Oscillator Config
Header Files
Timers
WDT
Not used
WDT Reset
18 ms
36 ms
72 ms
144 ms
288 ms
576 ms
1152 ms
2304 ms
Timer 0 (RTCC)
Source
Internal
ext_l_to_h
ext_h_to_l
Frequency: 5,000,000
Overflow: 51.2 us
Rtcc_Off
Rtcc_8_Bit
Resolution:
.2 us
.4 us
.8 us
1.6 us
3.2 us
6.4 us
12.8 us
25.6 us
51.2 us
Timer 1
Disabled
Internal
External
Resolution
Overflow
.2 us 13.1 ms
.4 us 26.2 ms
.8 us 52.4 ms
1.6 us 104 ms
Clock out for crystal
Sync Ext Clock to Osc
Frequency: 5,000,000
Timer 2
Enabled
Resolution
.2 us
.8 us
3.2 us
Overflow Period:
98
= 316 us
Interrupt Period:
16
= 5.0 ms
Options
Code
Ok
Cancel
Help

Lab 1: Project Wizard
Step 1 – Start Compiler IDE

- **Start the CCS Compiler IDE**
  - Double click on the icon labeled ‘CCS C’

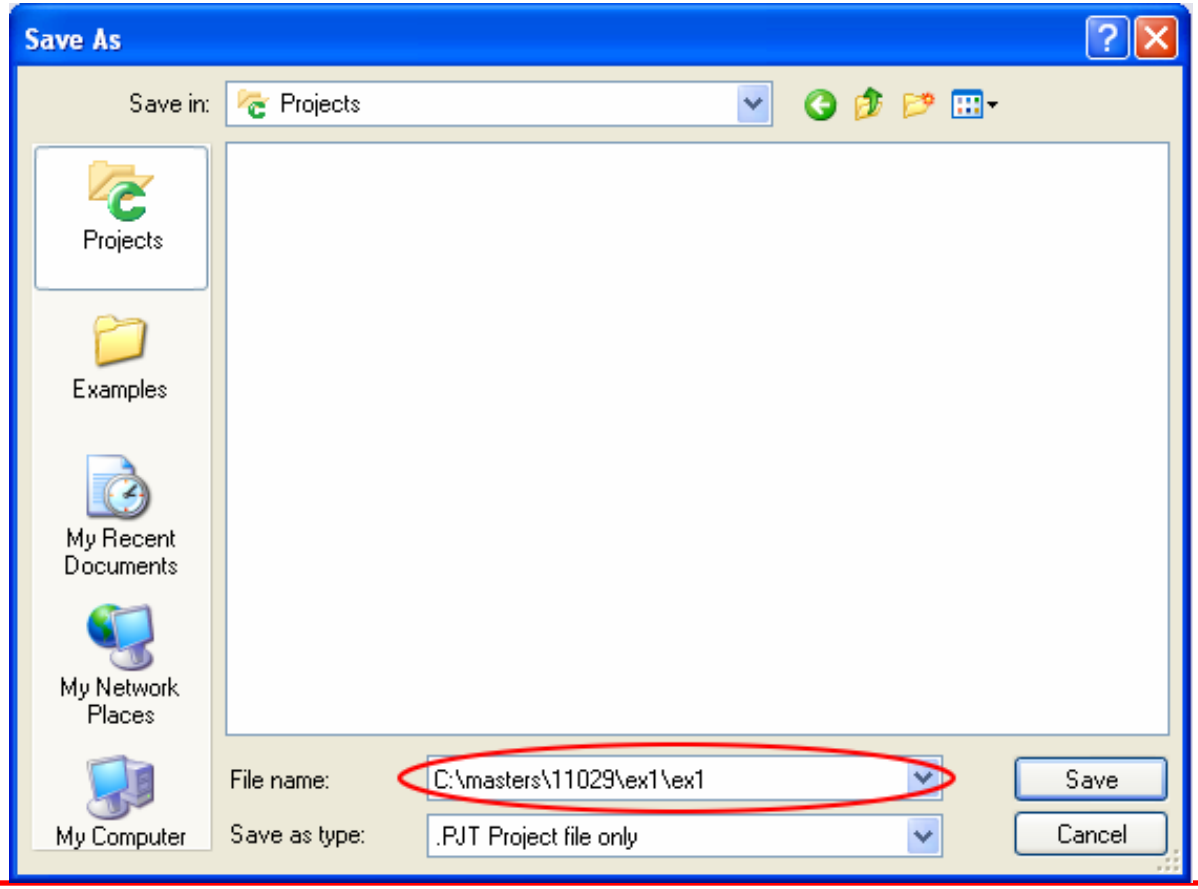
Lab 1: Project Wizard
Step 2- Start Project Wizard

- **Start the Project Wizard**
  - File -> New -> Project Wizard



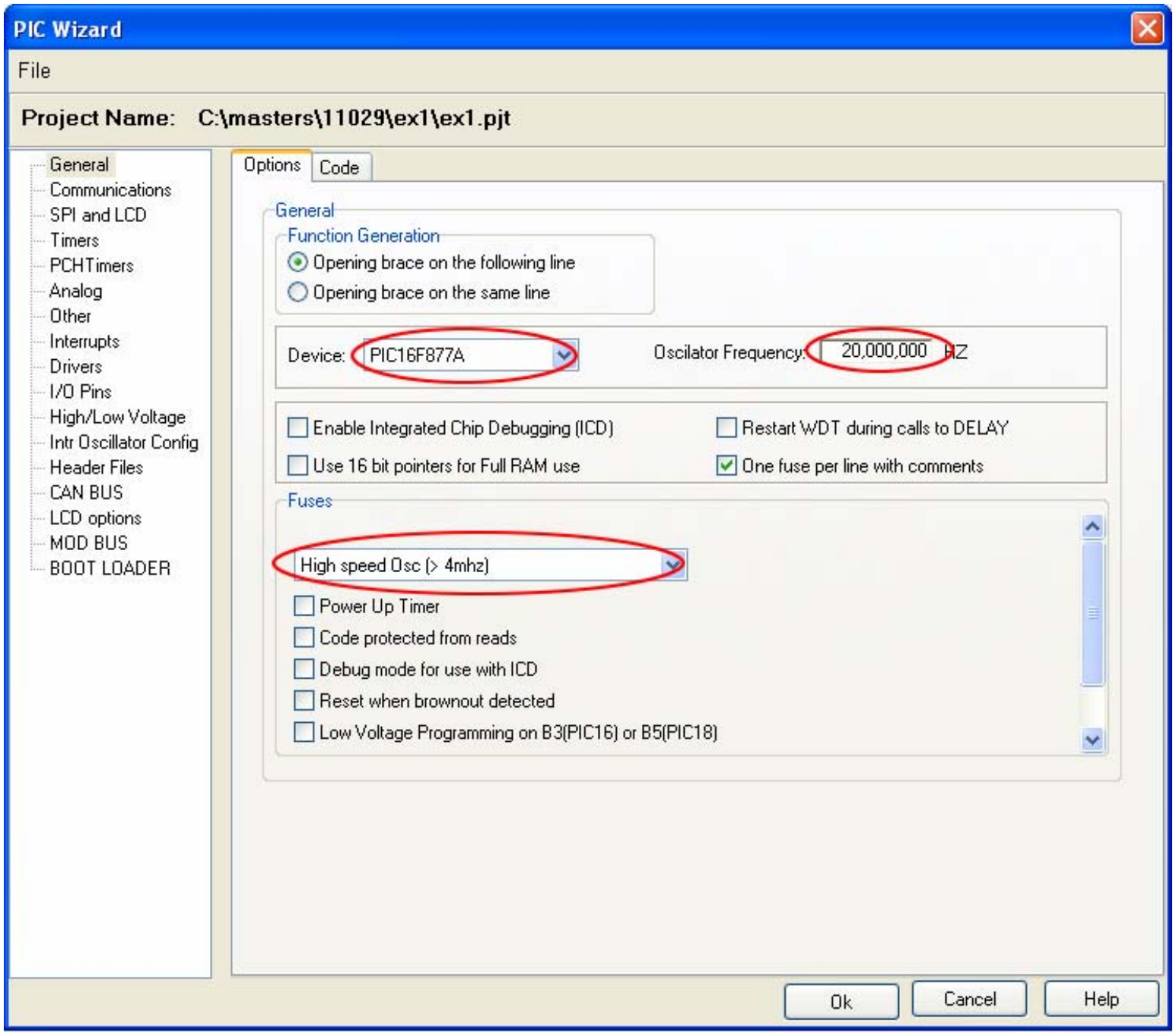
Lab 1: Project Wizard
Step 3 – File Name Dialog

- File name: C:\masters\11029\EX1\ex1.c
- Press Save



Lab 1: Project Wizard Step 4 – Configuration

- **You will now be in the Project Wizard Dialog**
- **In the General Section:**
  - Select 16F877A under Device
  - Set oscillator to 20000000 Hz
  - Select ‘High Speed Osc’ under fuse



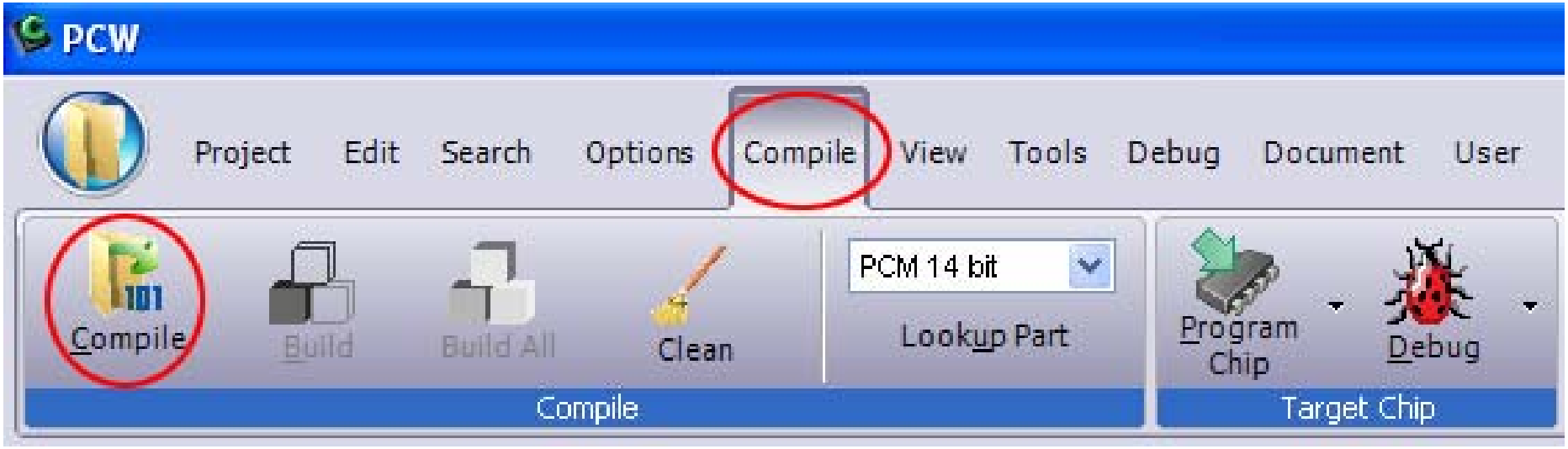
Lab 1: Project Wizard
Step 5 – Enter Code

- **Press the OK button in the Project Wizard**
- **Inside main(), find the line marked:**
  - //TODO: USER CODE
- **Replace this line with the following code:**

```
while(TRUE)
{
    delay_ms(500);
    output_toggle(PIN_B5);
}
```

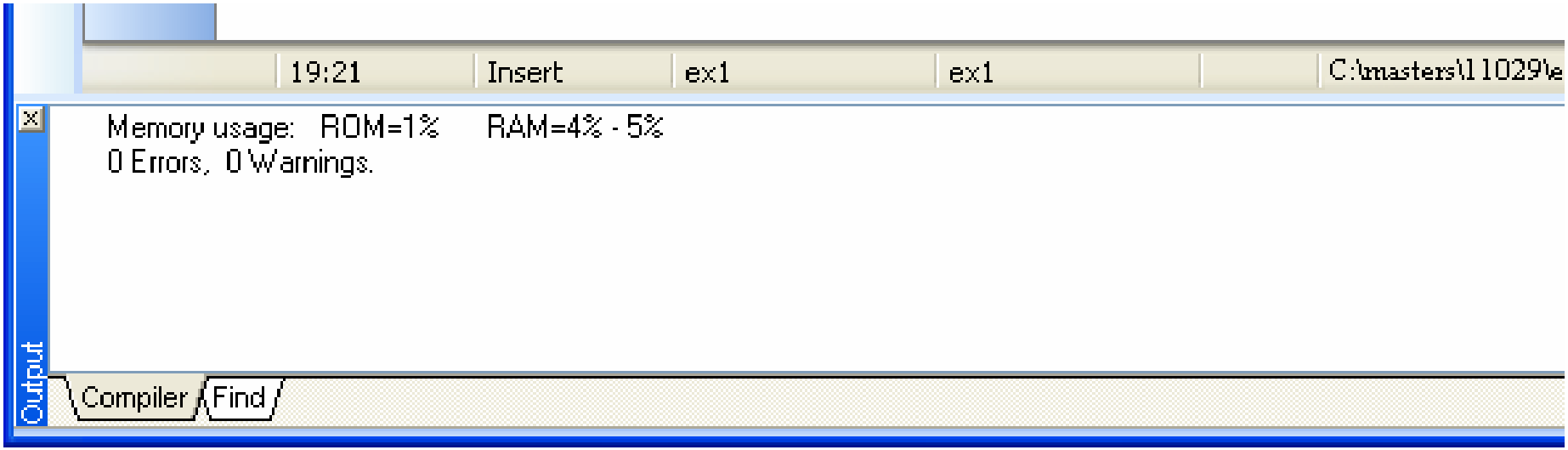
Lab 1: Project Wizard
Step 6 – Compile Code

- **Compile Code**
  - Compile -> Compile



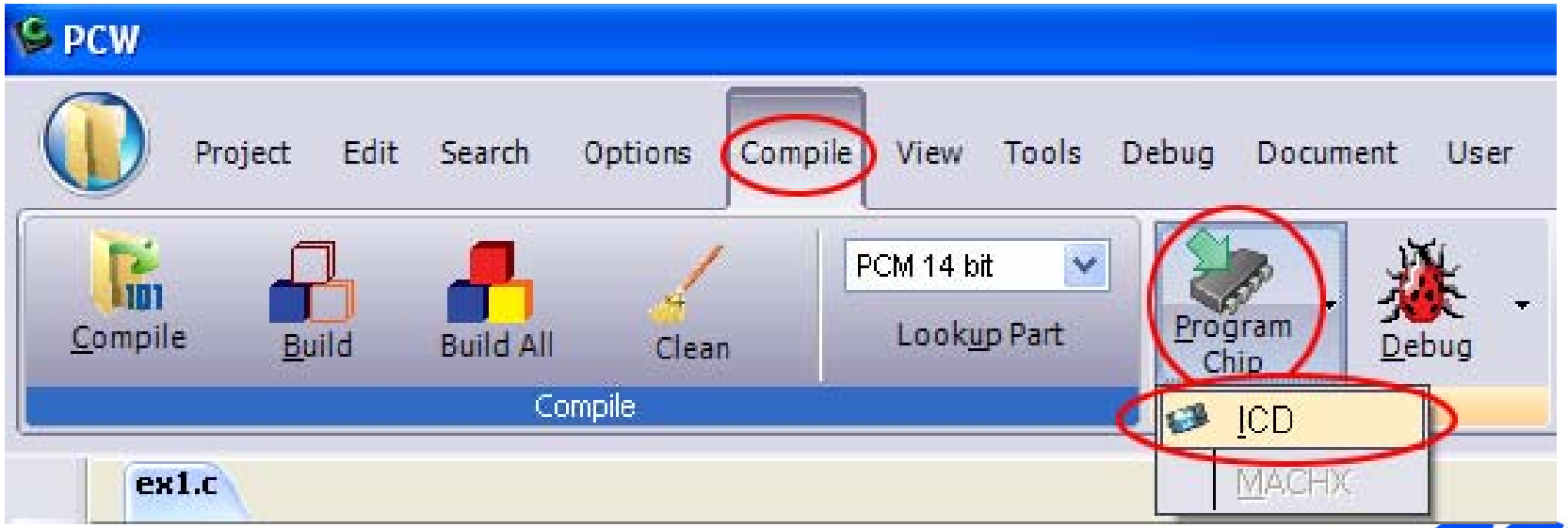
Lab 1: Project Wizard
Step 7 – Compile Results

- **Inspect the output window for compile results**
  - You should get 0 Errors.



Lab 1: Project Wizard Step 8 – Load HEX

- **Load HEX file with CCS ICD**
  - Click on the ‘Program Chip’ icon on the ‘Compile’ ribbon, select ICD



Lab 1 Hints

- **Follow handout**
- **Verify:**
  - Correct PIC® MCU Selected
  - Correct FUSEs selected
  - Correct #use delay() specified
  - Fix any compile errors
- **LED will blink at 1Hz rate**
- **C:\masters\11029\done\ex1.c**

UNIVERSITY OF MICROCHIP
UOFM
MASTERS 2007

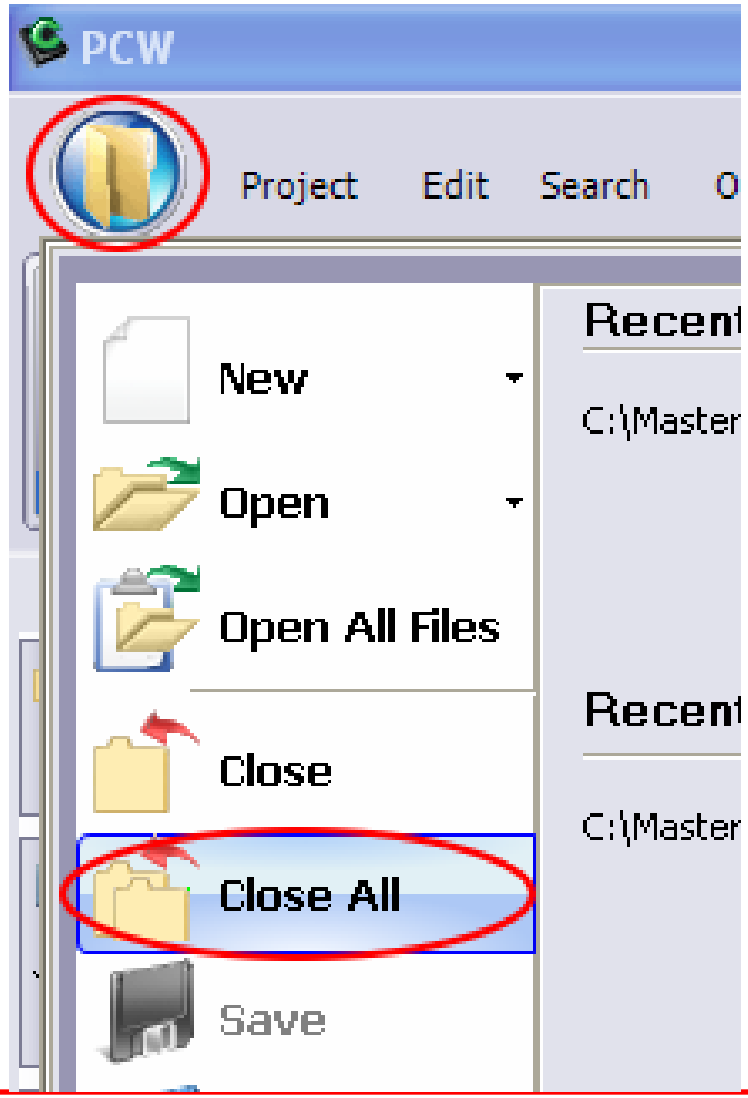
Hands-On Lab

Lab 2: Experiment with the Integrated Debugger

Lab 2: CCS IDE Debugger Overview

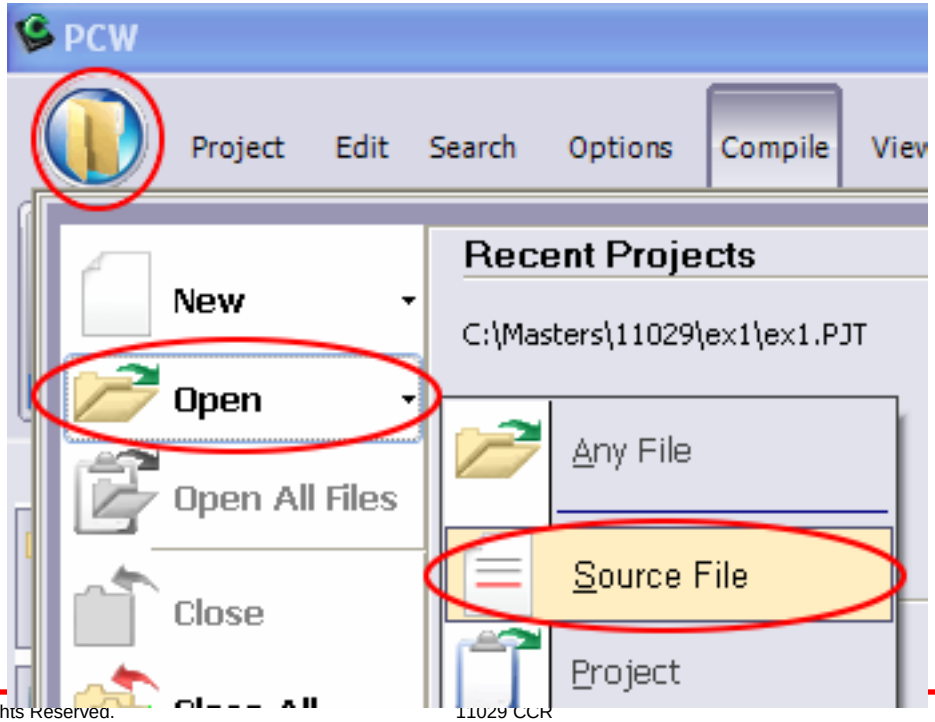
- Mouse-over tooltips with value
- Multiple-Breakpoints
- Watches
- ROM and RAM view
- Peripheral Status
- Stack Watch
- Logging
- 1Wire RS232 on RB3

Lab 2: CCS IDE Debugger
Step 1 – Close All Open Files



Lab 2: CCS IDE Debugger
Step 2 – Load Example Code

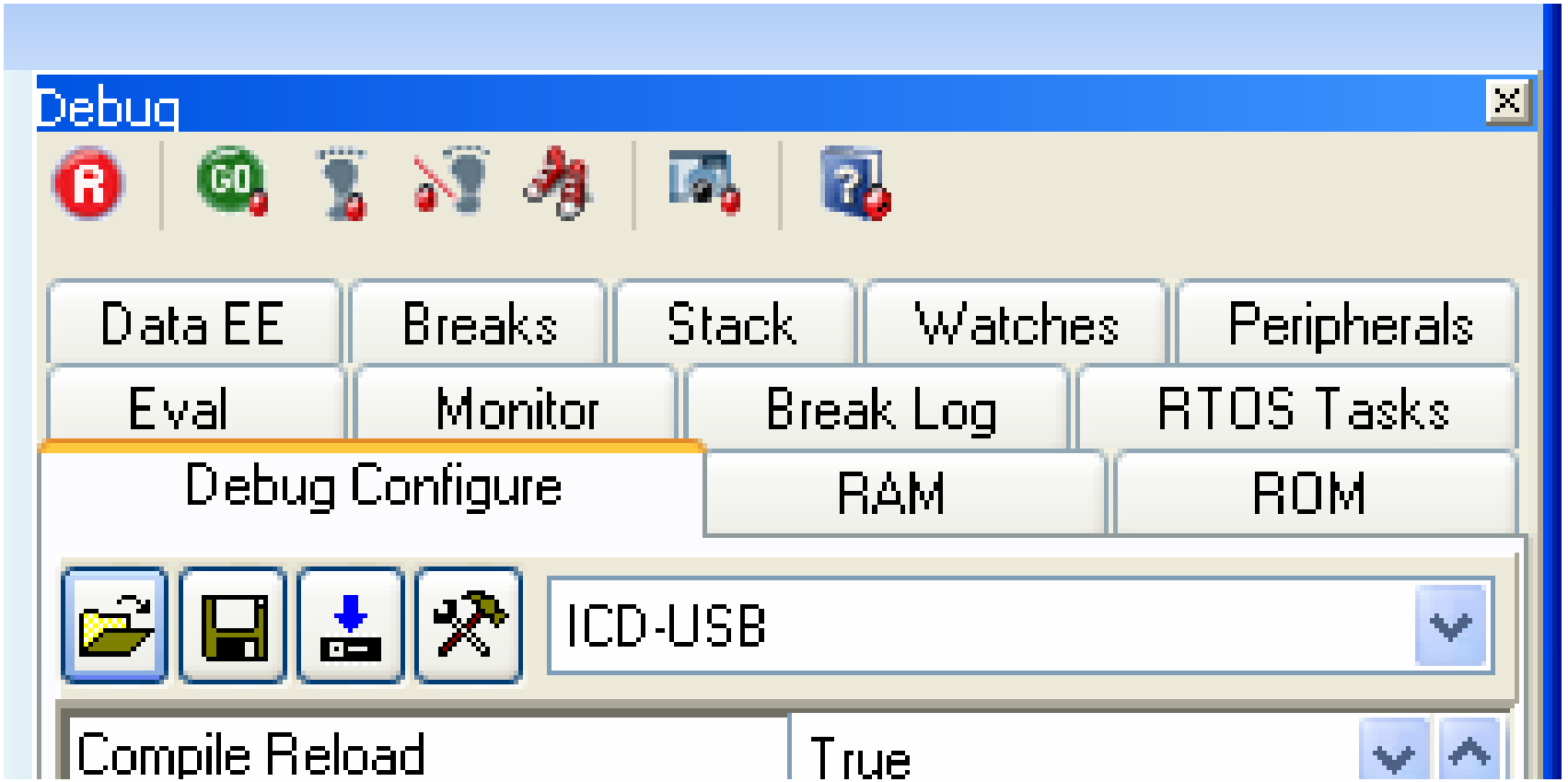
- **Load Example Code**
  - File->Open
  - C:\masters\11029\ex2\ex2.c



Lab 2: CCS IDE Debugger
Step 3 – Start Debugger

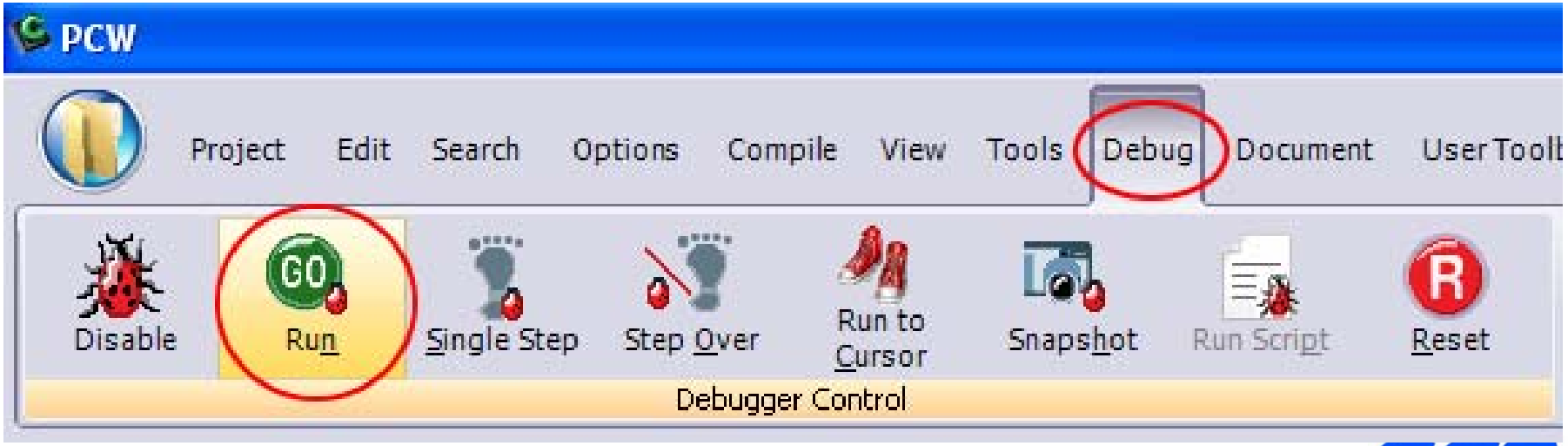
- **Start Debugger**
  - Debug -> Enable Debugger

Lab 2: CCS IDE Debugger
Debugger GUI Overview



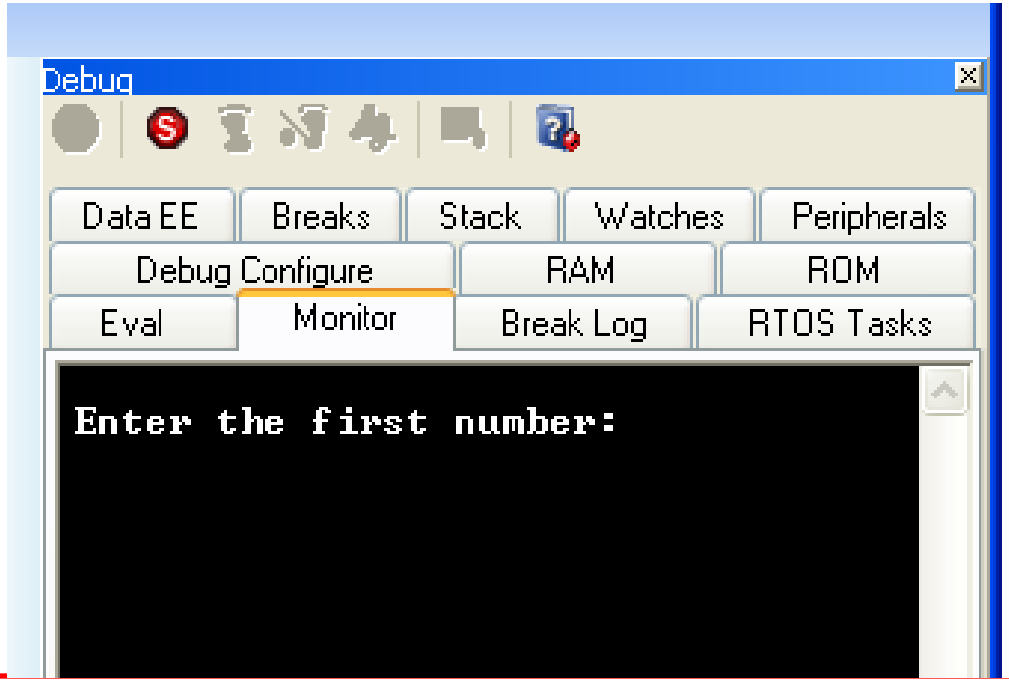
Lab 2: CCS IDE Debugger
Step 4 – Load Target Code

- **Compile Project**
  - Compile -> Compile
  - Project will be loaded
- **Press GO**



Lab 2: CCS IDE Debugger Step 5 – 1-Wire RS232

- **#use rs232(debugger)**
  - 1-Wire RS232 using RB3
  - 1-Wire Monitor in Debugger:

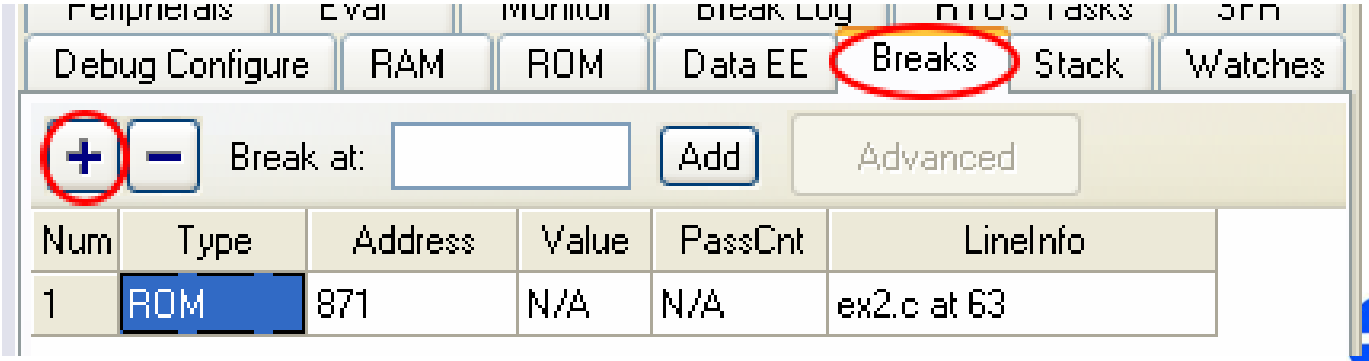


Lab 2: CCS IDE Debugger Step 6 – 1-Wire RS232

- Using the Monitor, experiment with the application
- Application is a simple calculator

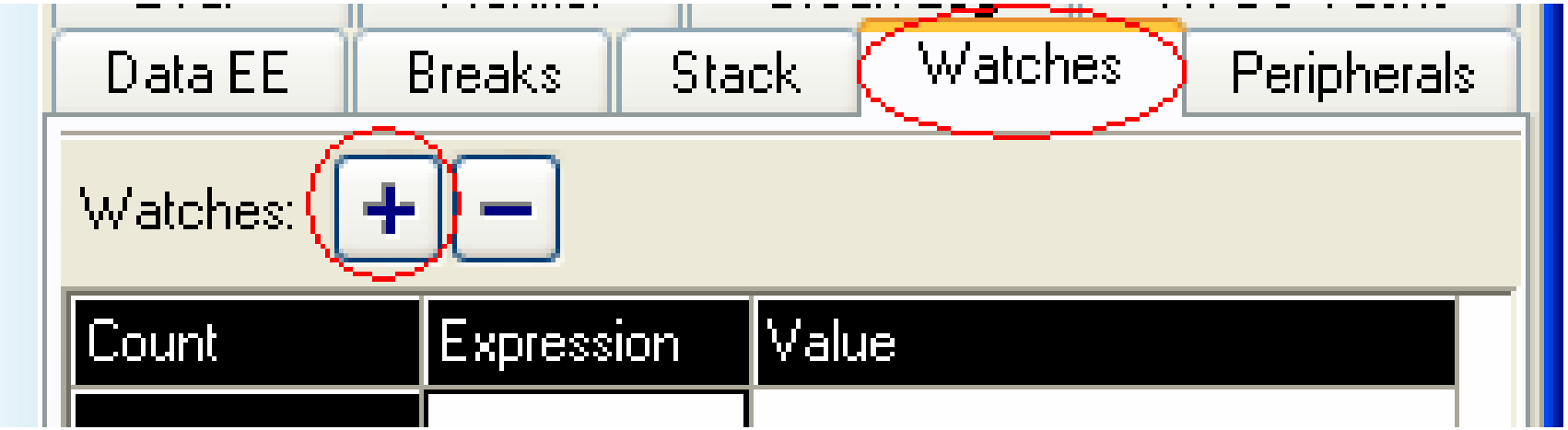
Lab 2: CCS IDE Debugger Step 7 – Breakpoints

- **Press STOP**
- **Add a Breakpoint to line 63**
  - printf("\r\nThe result is %lu", result)
  - Put cursor on Line 63, Press + on Break Tab



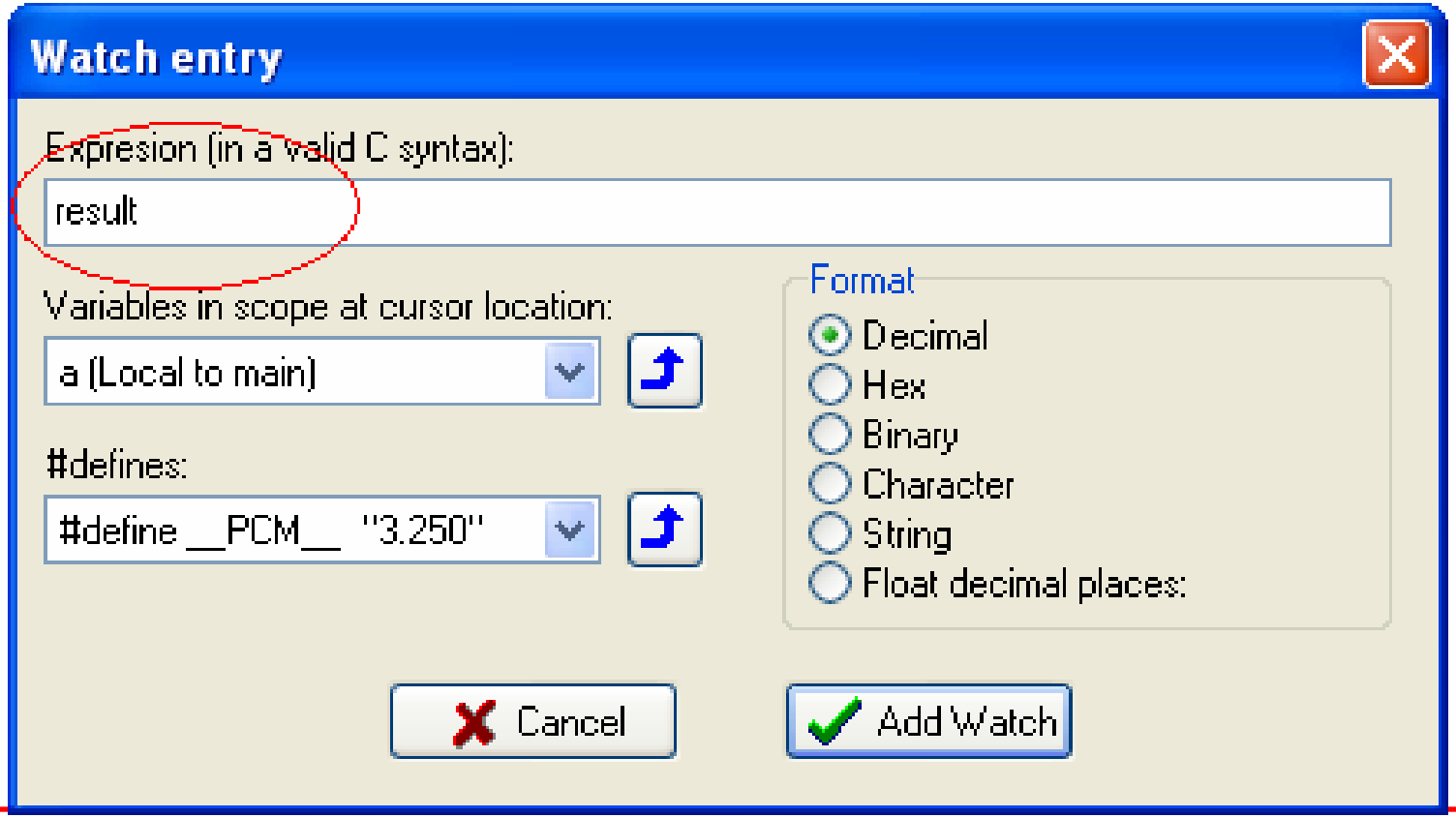
Lab 2: CCS IDE Debugger Step 8 – Watching

- **Open Watch Tab, Press +**



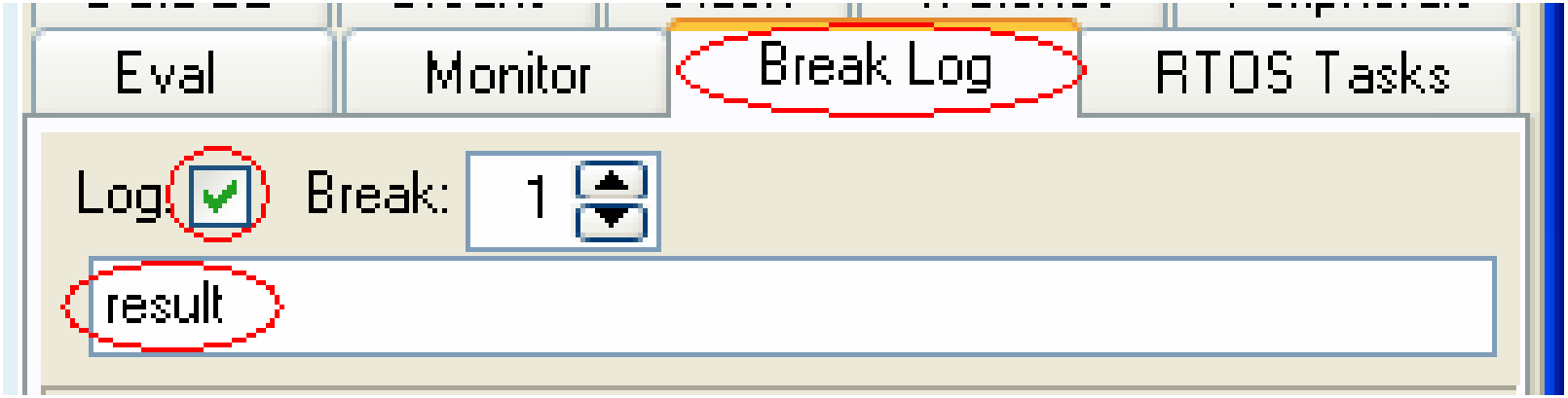
Lab 2: CCS IDE Debugger
Step 9 – Watch Dialog

- Enter ‘result’ into Expression Field
- Press Add Watch



Lab 2: CCS IDE Debugger
Step 10 – Break Logging

- **Logs an expression at breakpoint**
  - Breakpoint doesn't stop execution
- **Enter result into Break Log**



- **Press GO**

UNIVERSITY OF MICROCHIP
UOFM
MASTERS 2007

Hands-On Lab

Lab 3: Use a Timer Interrupt to create a simple stopwatch

Lab 3: Stop Watch Timer

- **A stop watch timer will be implemented**
  - A Timer 0 interrupt is used to time a duration
  - Press and hold the button to start the timer
  - Release the button to display the amount of time button was held (result will be displayed on the Debugger Monitor, or you could Break/Watch the result in the Debugger)
- **Open C:\masters\11029\EX3\ex3.c**
  - Complete the lines marked TODO

Lab 3 Hints

- **C:\masters\11029\EX3\ex3.c**
  - Completed example with no TODO
- **#int_timer0**
  - Following function is called on a Timer 0 overflow
- **setup_timer_0(RTCC_DIV_256)**
  - Configures Timer 0 to use a divide by 256 prescalar
- **enable_interrupts(int_xxx)**
  - INT_TIMER0 – Enable Timer 0 Interrupt
  - GLOBAL – Enables Global Interrupt
- **input(PIN_XX)**
  - Returns TRUE if PIN_XX is high, else returns FALSE
  - Button is RA4, or PIN_A4 in CCS
- **C:\masters\11029\done\EX3\ex3.c**

UNIVERSITY OF MICROCHIP
UOFM
MASTERS 2007

Hands-On Lab

Lab 4: Read A/D, use Fixed Point Decimal

Lab 4

- **Read A/D conversion**
- **Convert A/D result to 0.00-5.00 volts in fixed point decimal**
- **Light LEDs based upon voltage**
- **C:\masters\11029\ex4\ex4.c**
  - Complete the lines marked TODO

Lab 4 Hints

- **C:\masters\11029\ex4\ex4.c**
  - Complete the lines marked TODO
- **[type] fixed(y) [name]**
  - Math using fixed(y) is actually scaled by 10^y
- **ADC Troubles?**
  - Did you enable A/D with setup_adc()?
  - Did you set the channel?
  - Is A/D correct, but fixed() math wrong? Verify readings using debugger.
- **C:\masters\11029\done\ex4\ex4.c**

Hands-On Lab

Lab 5: Write/Read Program Memory

Lab 5

- **A block of program memory is reserved**
  - BASE_ADDRESS = 0x500
  - BLOCK_SIZE = 64
- **In debug monitor, content of block is displayed**
- **In debug monitor, you can change the content of block**
- **C:\masters\11029\ex5\ex5.c**
  - Complete lines marked TODO

Lab 5 Hints

- **C:\masters\11029\ex5\ex5.c**
  - Complete lines marked TODO
- **read_program_eeprom()**
- **write_program_eeprom()**
- **Don't forget to offset address by BASE_ADDRESS**
- **Don't write beyond BLOCK_SIZE**
- **C:\masters\11029\done\ex5\ex5.c**
- **Extra Credit: Use addressmod**

Appendix

Frequently Asked Questions

Why do I get out of ROM error, when there is ROM left?

- **A function must fit into one bank**
- **On 14-bit this is 2K**
- **Split large functions, and main(), into several smaller functions**
  - This is good programming practice any way!

How can I reduce code space?

- Use int1 or bit fields for flags
- Use fixed point decimal, not float
- Divide large functions
- Avoid ->, move structure to local
- Use access bank mode
  - #device *=8
  - read_bank(b,o), write_bank(b,o,v)

Why is my math clipped at 8-bits?

- **Examine the following code:**
  - val16=val8 * val16;
- **The optimizer will use the smallest data type for math, in this case int8**
- **Typecasting forces math into proper mode:**
  - val16=(int16) val8 * val16;

Programming Details

Miscellaneous

Pre-Processor Miscellaneous

- **__pcb__**
- **__pcm__**
- **__pch__**
- **__pcd__**
  - Returns the version of specified compiler
- **__date__**
- **__time__**
  - Time/Date project was compiled
- **#id CHECKSUM**
- **#id value**
  - Place this value (or checksum) into ID location

Appendix

Other Built-In Functions

Built-In Functions
Bit Manipulation

- **bool=bit_test(var, bit)**
  - Returns the value of the specified bit
- **bit_clear(var, bit)**
- **bit_set(var, bit)**
  - Set/clear the specified bit
- **The above bit_XXX() functions use the PIC® MCU's bit operation opcodes for maximum efficiency**

Built-In Functions
Byte Manipulation

- **int8=make8(variable, offset)**
  - Returns one byte from variable
- **int16=make16(i8MSB, i8LSB)**
- **int32=make32(iMSB..iLSB)**
  - Returns the combined value
- **swap(value)**
  - Swaps nibble, saves to value

Timers

- **setup_timer_X(mode)**
  - Mode contains configuration info, such as prescalar, postscalar, period, etc.
  - `setup_timer_1(T1_INTERNAL | T1_DIV_BY_4);`
- **set_timerX(new_count)**
- **current_count=get_timerX()**
  - Set/Get Timer count
  - Word Safe

Capture / Compare / PWM

- **setup_ccpX(mode)**
  - Mode contains configuration, examine header for full options.
  - **`setup_ccp1(CCP_PWM)`**
- **set_pwmX_duty(new_duty)**
  - % = new_duty / (4 * period)

Peripheral Control Parallel Slave Port

- **setup_psp(mode)**
  - Mode is PSP_ENABLED or PSP_DISABLED
- **full=psp_output_full()**
- **avail=psp_input_full()**
- **isOverflowed=psp_overflow()**
- **psp_data**
  - A variable mapped to the PSP I/O port

Built-In Functions Miscellaneous

- **ext_int_edge(which, mode)**
  - ext_int_edge(0, H_TO_L)
- **port_X_pullups(boolean)**
- **setup_oscillator(mode, finetune)**
  - finetune is optional
  - setup_oscillator(OSC_2MHZ)

For more CCS related slides, see the slides for the MASTERS class ‘11028 CCS’

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KeeLoq, KeeLoq logo, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rfPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AmpLab, FilterLab, Linear Active Thermistor, Migratable Memory, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Smart Serial, SmartTel, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.