Getting Started with the dsPIC30F 16-bit DSC

Hands-on Workshop

dsPIC® Hands-on Workshop

- Goals
  - Gain knowledge of dsPIC® DSC architecture
  - Use MPLAB® IDE development environment
  - Learn how to develop code on a dsPIC DSC
    - MPLAB C30 compiler
    - Use Peripherals and C Peripheral Libraries
    - Use DSP FilterLab® software
  - Use MPLAB ICD 2 In-Circuit Debugger
  - Debug on dsPICDEM™ 1.1 Development Board

Corporate Overview

- Leading semiconductor manufacturer:
  - of high-performance, field-programmable, 8-bit & 16-bit RISC microcontrollers
  - of analog & interface products
  - of related memory products
  - for high-volume embedded control applications
- $651M in product sales in FY03
- More than 3,000 employees
- Headquartered near Phoenix in Chandler, AZ

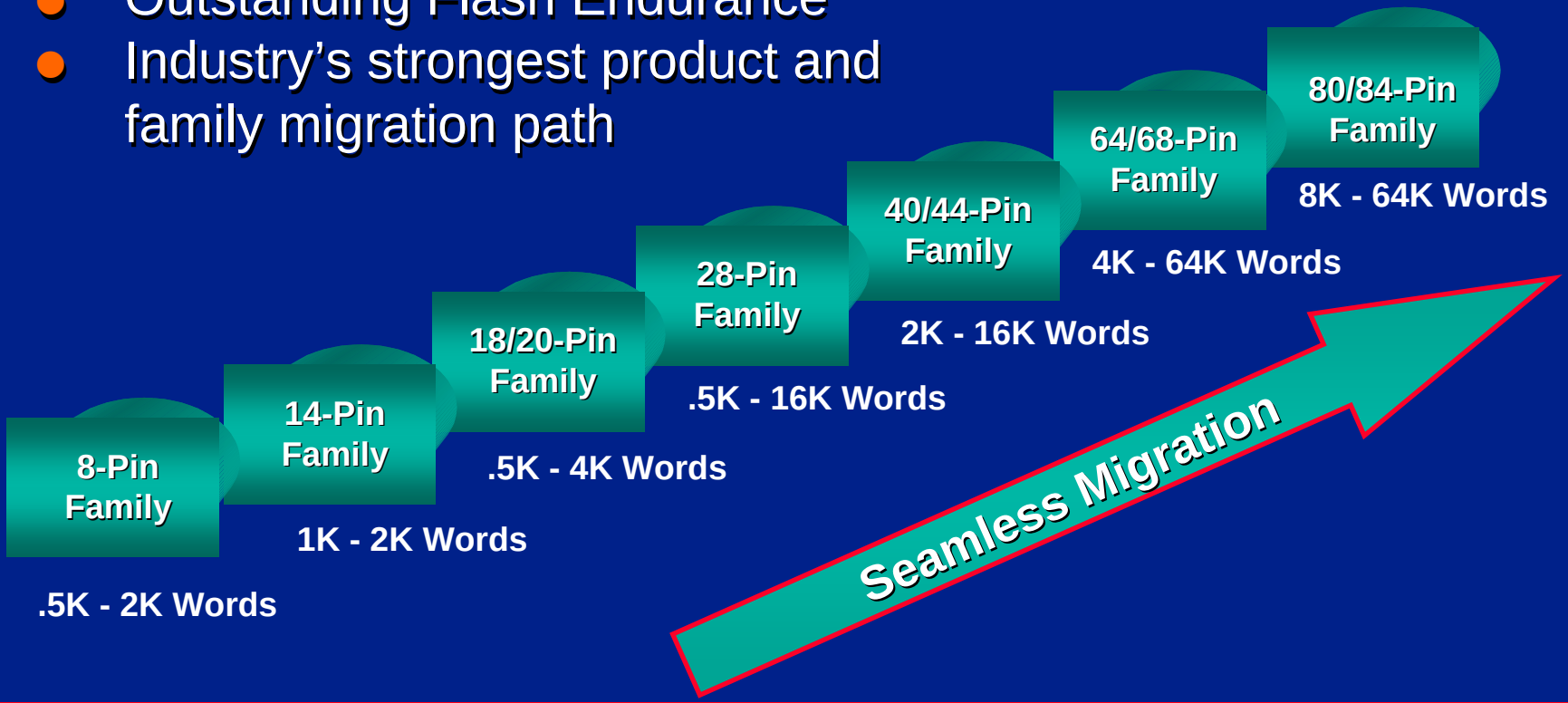
“The Silicon Desert”

PICmicro® MCU Product Migration Path

212 Products

- Flash, OTP and ROM
- Superior Analog functionality
- Outstanding Flash Endurance
- Industry's strongest product and family migration path

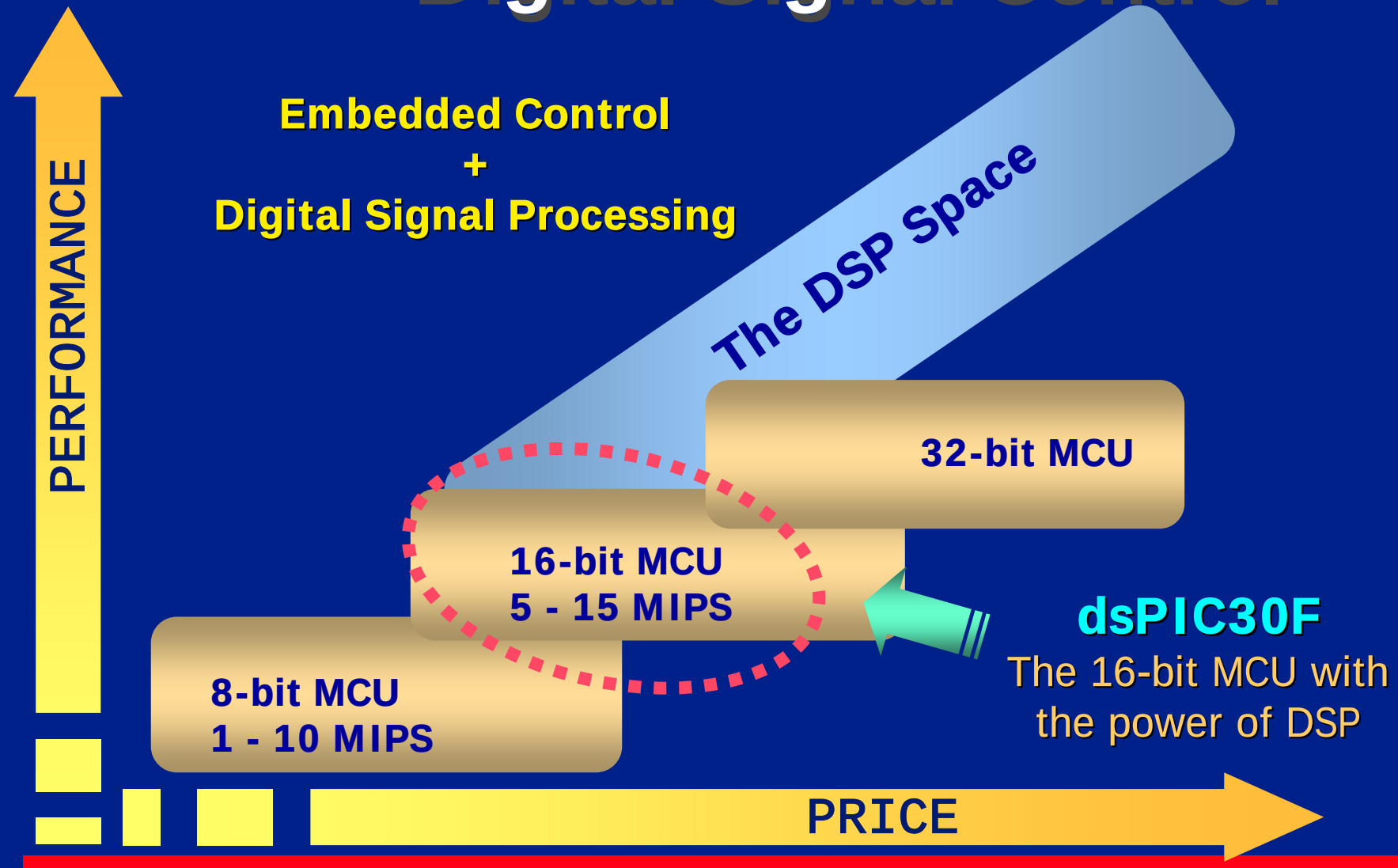
#1 in 8-bit Microcontroller Shipments



80+ Flash PICmicro® MCUs!

- **Best in Industry Program Memory Endurance**
  - 100,000 Erase/Write Cycles Program Memory
  - 1,000,000 Erase/Write Cycles Data EEPROM
- **Separate data EEPROM space**
  - MCU can continue executing code while writing to data EEPROM
  - Store calibration values and serial numbers
- **In-Circuit Debugging Support**
  - Low cost development tools
  - Develop with the production microcontroller

What is DSC ?
Digital Signal Control



dsPIC30F

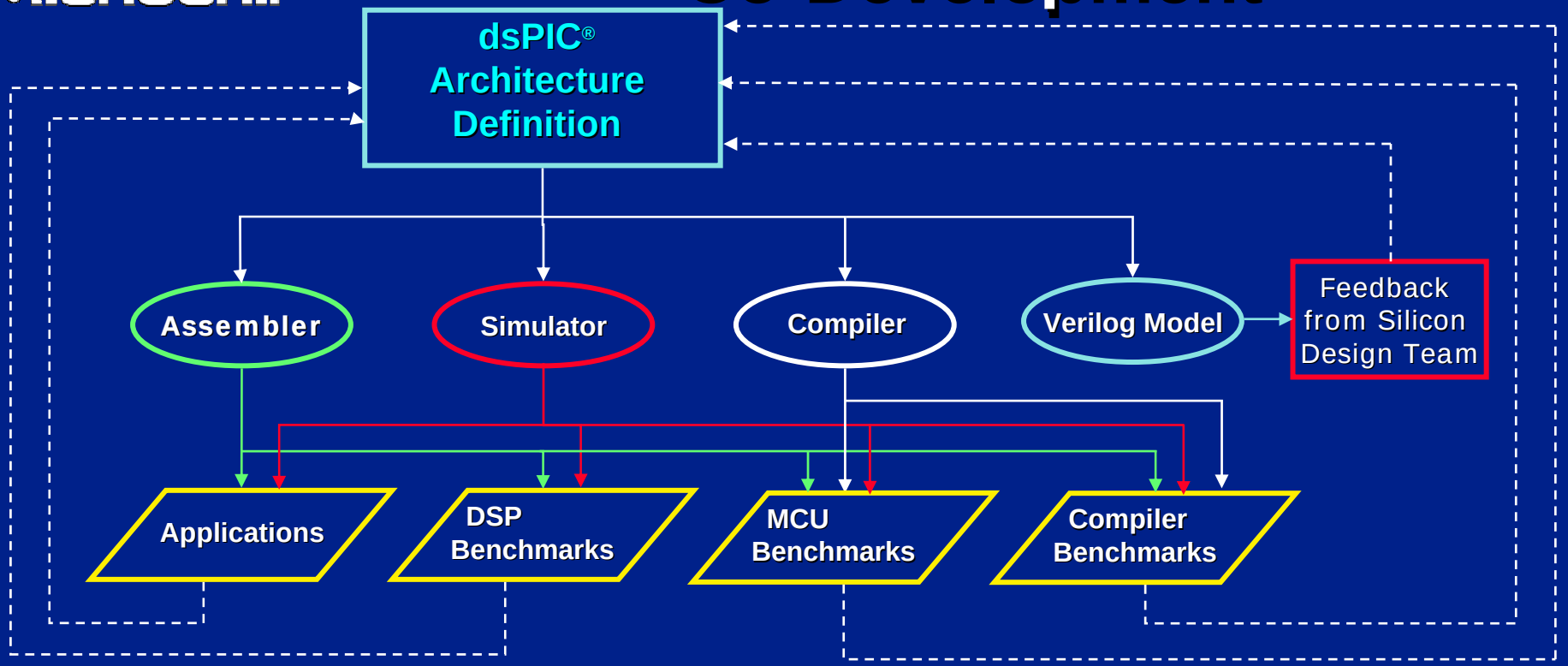
- Looks Like a MCU
  - Easy to use; Built-in Peripherals
- Performs Like a DSP
  - Powerful DSP engine
- Priced Like a MCU

The 16-bit MCU with the power of DSP

dsPIC® Family: Architected from Scratch

- Seamlessly integrates a DSP and an MCU
- MCU look and feel, easy to use
- Competitive DSP performance
- Optimized for C compiler
- Fast, deterministic, flexible interrupts
- Excellent RTOS support

Co-Development



Co-development of Architecture, Compiler, Applications & Algorithms

- In-house compiler team
- 3rd party compiler vendor
- Speech/modem consultants
- Motor control consultants
- RTOS vendors
- Encryption consultants

C Compiler Efficiency

- dsPIC30F Processor includes features to enhance C code efficiency

New instruction types

+

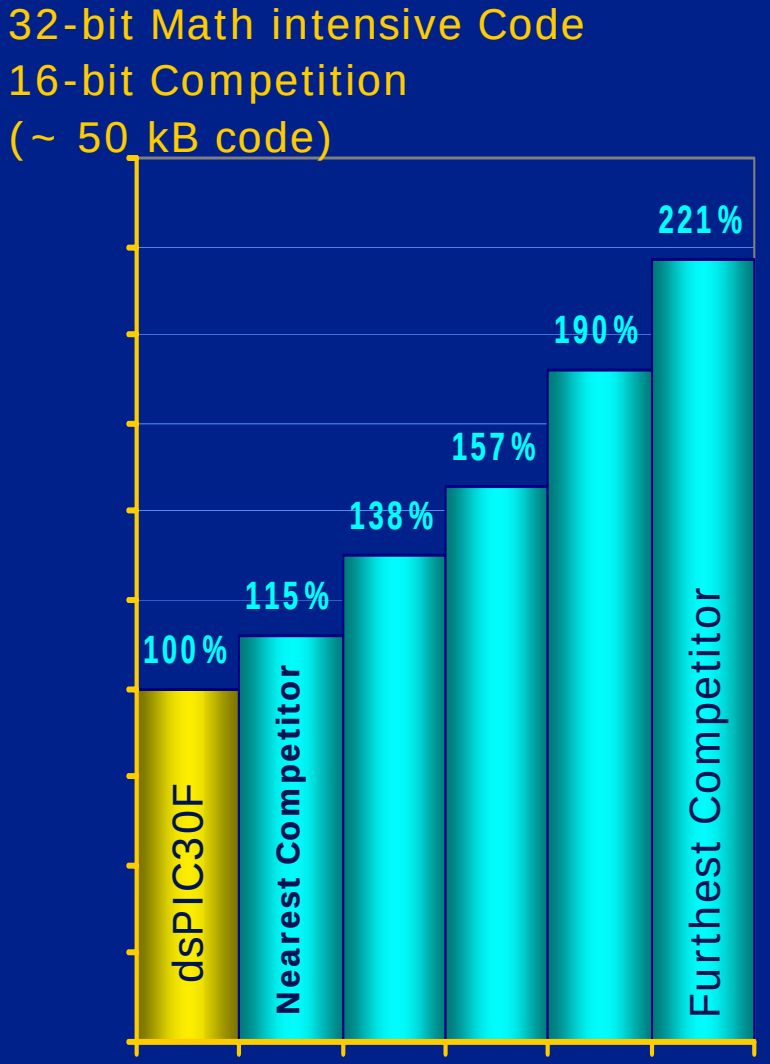
More flexible addressing

+

Software stack

=

Compact C code



dsPIC30F Families

- Power Conversion and Motor Control Family
- Sensor Processor Family
- General Purpose Controller Family

dsPIC30F: 16-bit MCU with the power of DSP

dsPIC30F Products

Power Conversion & Motor Control Family

Product dsPIC(R) DSC	Pins	Flash KB	SRAM Bytes	EE Bytes	Timer 16-bit	Input Cap	Output Comp/ Std PWM	Motor Cntrl PWM	A/D 10-bit 500 KSPS	Quad Enc	UART	SPI (TM)	I2C (TM)	CAN
dsPIC30F2010	28	12	512	1024	3	4	2	6	6 ch	Yes	1	1	1	
dsPIC30F3010	28	24	1024	1024	5	4	2	6	6 ch	Yes	1	1	1	
dsPIC30F4012	28	48	2048	1024	5	4	2	6	6 ch	Yes	1	1	1	1
dsPIC30F3011	40	24	1024	1024	5	4	4	6	9 ch	Yes	2	1	1	
dsPIC30F4011	40	48	2048	1024	5	4	4	6	9 ch	Yes	2	1	1	1
dsPIC30F5015	64	66	2048	1024	5	4	4	8	16 ch	Yes	1	2	1	1
dsPIC30F6010	80	144	8192	4096	5	8	8	8	16 ch	Yes	2	2	1	2

- Brushless DC Motor Control
- AC Induction Motor Control
- Switch Reluctance Motor Control
- UPS, Inverters and Power Supplies

- Appliances
- Power Tools
- Automotive
- Industrial

dsPIC30F Products

General Purpose Controller Family

Product dsPIC(R) DSC	Pins	Flash KB	SRAM Bytes	EE Bytes	Timer 16-bit	Input Capture	Output Compare Std PWM	A/D 12-bit 100 KSPS	UART	SPI (TM)	I2C (TM)	CAN	Codec Interface
dsPIC30F3014	40	24	2048	1024	3	2	2	13 ch	2	1	1		
dsPIC30F4013	40	48	2048	1024	5	4	4	13 ch	2	1	1	1	AC97, I2S
dsPIC30F5011	64	66	4096	1024	5	8	8	16 ch	2	2	1	2	AC97, I2S
dsPIC30F6011	64	132	6144	2048	5	8	8	16 ch	2	2	1	2	
dsPIC30F6012	64	144	8192	4096	5	8	8	16 ch	2	2	1	2	AC97, I2S
dsPIC30F5013	80	66	4096	1024	5	8	8	16 ch	2	2	1	2	AC97, I2S
dsPIC30F6013	80	132	6144	2048	5	8	8	16 ch	2	2	1	2	
dsPIC30F6014	80	144	8192	4096	5	8	8	16 ch	2	2	1	2	AC97, I2S

dsPIC30F Products

Sensor Processor Family

Product dsPIC(R) DSC	Pins	Flash KB	SRAM Bytes	EE Bytes	Timer 16-bit	Input Capture	Output Compare Std PWM	A/D	UART	SPI (TM)	I2C (TM)	CAN
dsPIC30F2011	18	12	1024		3	2	2	12-bit, 8 ch	1		1	
dsPIC30F3012	18	24	2048	1024	3	2	2	12-bit, 8 ch	1		1	
dsPIC30F2012	28	12	1024		3	2	2	12-bit, 10 ch	1		1	
dsPIC30F3013	28	24	2048	1024	3	2	2	12-bit, 10 ch	2		1	

- Glass Break Detect
- Gas Sensor
- Torque Sensor
- Tire Pressure Sensor
- Steering Angle Sensor
- Rain Sensor
- Low power, Smart Sensor
- Airbag Sensor Processor
- Pressure Sensors
- Vibration Measurement

Hands-on Labs

- Learn to use dsPIC30F tools in MPLAB® IDE
  - Create a project in MPLAB IDE
  - Compile code with MPLAB C30 Compiler
  - Program dsPIC30F6014 with MPLAB ICD 2
  - Debug code on dsPICDEM™ 1.1 Development Board
  - Use dsPIC® peripherals - ADC and UART
  - Use dsPIC DSP functions - digital filter

Hands-on Labs

- Learn to use important features of the dsPIC® Digital Signal Controller
  - I/O Ports
  - Timers
  - Interrupts
  - UART
  - A/D Converter
  - DSP Features

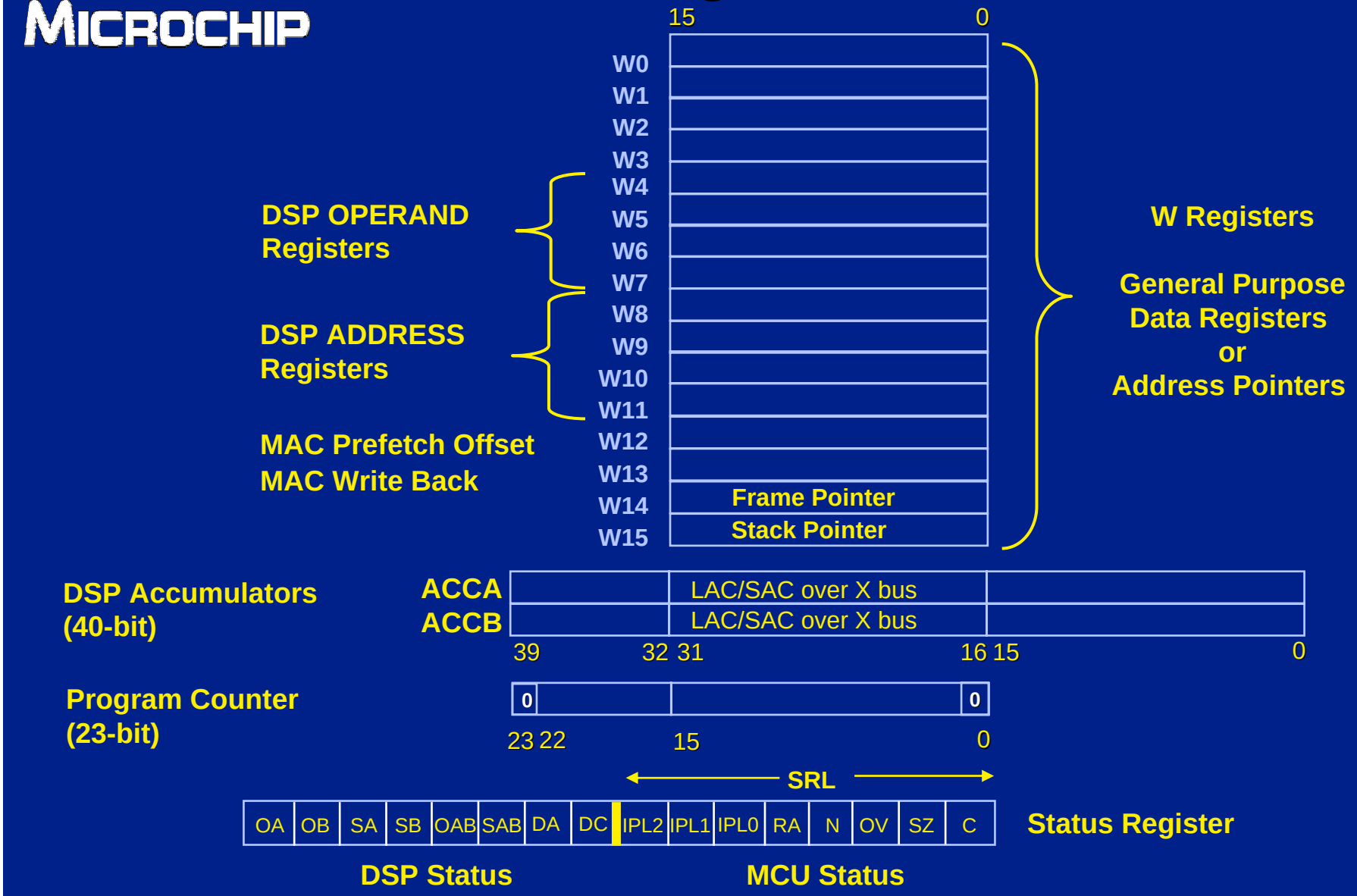
dsPIC® DSC Architecture

- Main Features
  - Single Core Integrating an MCU & a DSP
  - Modified Harvard Architecture
  - Data is 16-bit wide
  - Instruction is 24-bit Wide
  - Linear Program Memory up to 12 MB
  - Linear Data (RAM) up to 64 kB
  - True DSP Capability
  - Many Integrated Peripherals

dsPIC® DSC Architecture

- Main Features (continued)
  - 16 x 16-bit Working Register Array
  - Software Stack
  - Fast, Deterministic Interrupt Response
  - Three Operand Instructions: C = A + B
  - Extensive Addressing Modes

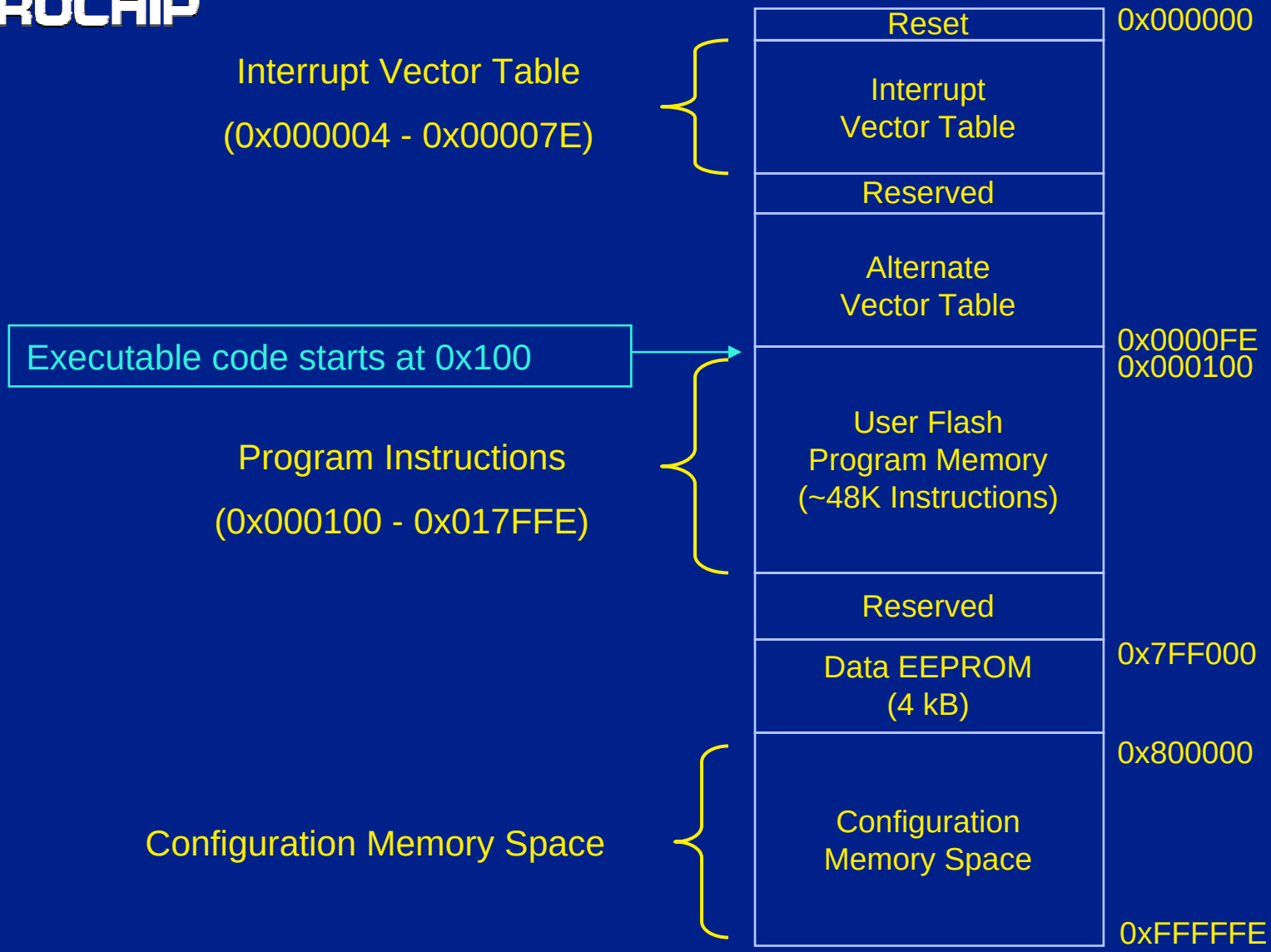
Programmers Model



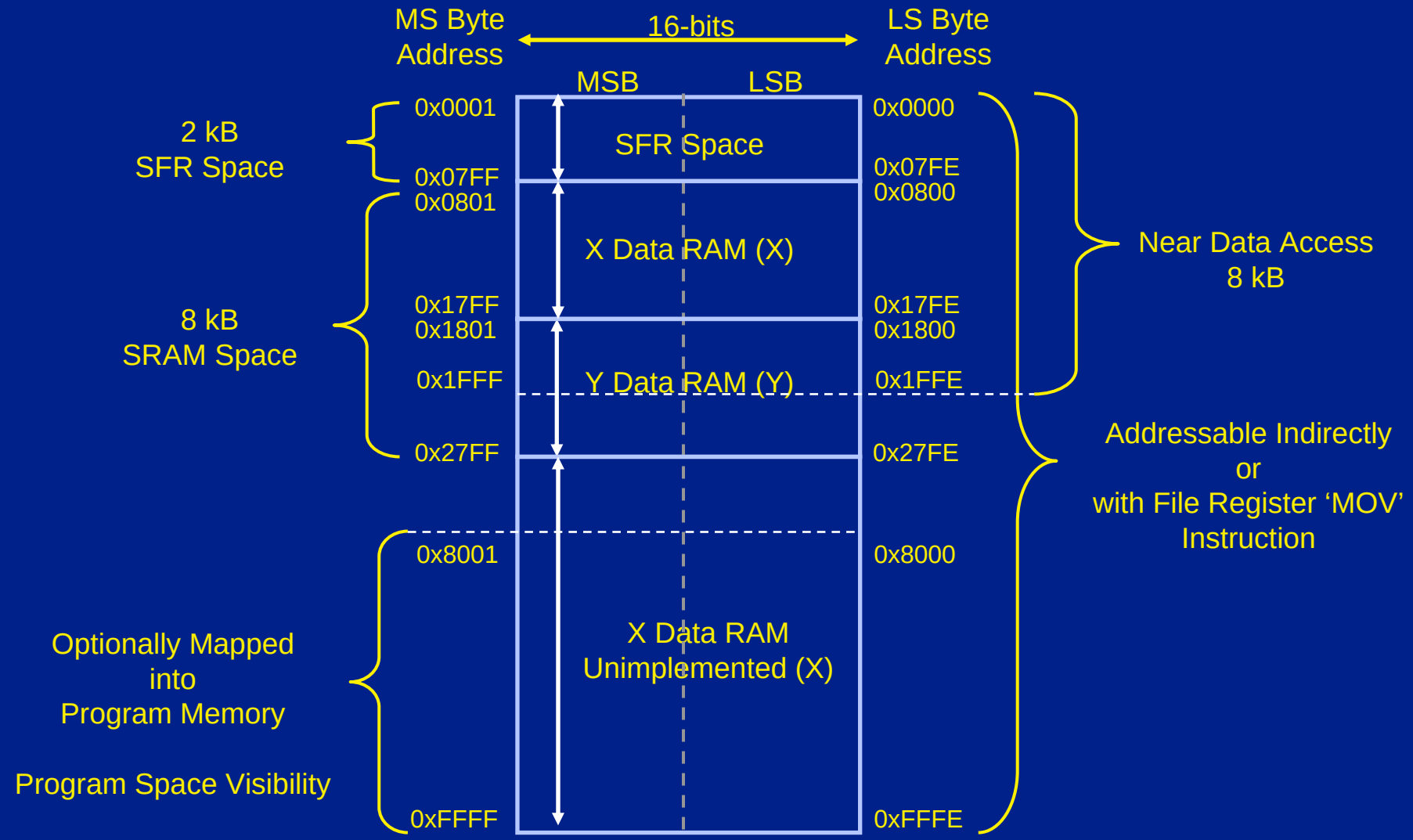
Programmers Model

- 16 x 16-bit Working Registers
  - More Flexibility for
    - DSP and MCU Algorithms
    - General Purpose Pointers
    - Stack Manipulation
  - Stack and Frame Pointers
    - W15 is Stack Pointer
    - W14 is Frame Pointer

Program Memory - dsPIC30F6014



Data Memory - dsPIC30F6014



dsPIC DSC Instruction Set

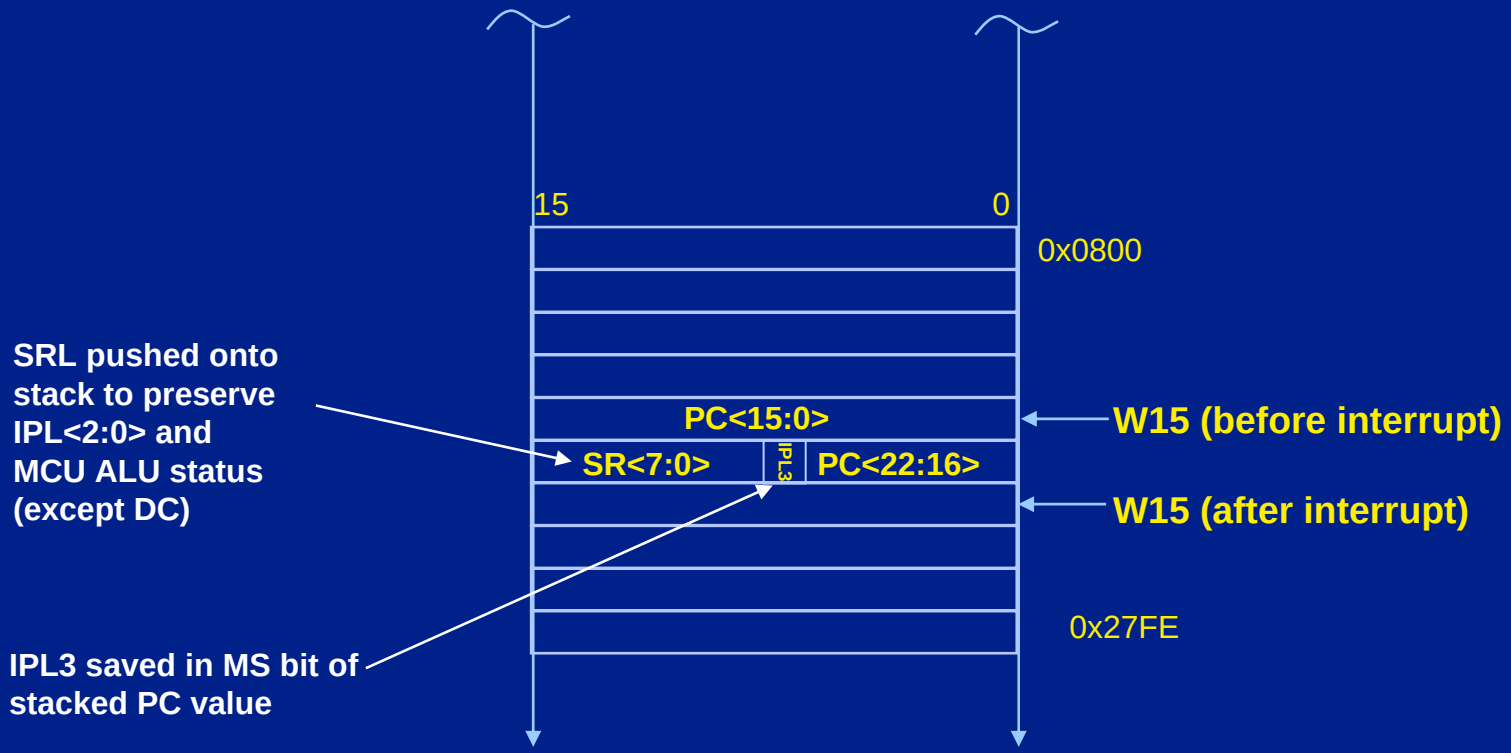
- Instruction Set Features
  - 84 instructions
  - Most are one cycle
  - Most are one word (24 bit)
  - Three operand instructions A = B op C
    - For compiler efficiency
  - DSP instructions
  - DO instruction
  - REPEAT instruction

Addressing Modes

- Generic Addressing Modes
  - Register — ADD W0,W1,W2
  - Memory Direct — CLR CORCON
  - Register Indirect — DEC [++W0],W1
  - with Signed Offset — MOV [W0 + #0xC3],W1
  - Literal (Immediate) — SUB #0x1D7,W0
  - Register indexed — MOV [W0 + W1],W2
- Special addressing methods for DSP
  - Modulo (Circular Buffer)
  - Bit Reversed (for FFT)

Interrupts

- Stack Operation Prior to ISR Entry



Now…

A Few Details About

Development Tools

Editor, Projects

Simulator

MPLAB® C30 C Compiler

Editor

- Text editor, integrated with MPLAB® IDE
- Context sensitive color coding
  - For assembler and C compiler
- Many editor features
  - Cut, paste, copy, find/replace, match brace, goto line/label/function, bookmarks, indent/outdent block, comment/uncomment block, format upper/lower case, auto indent, word wrap, etc

Editor

Title bar with file changed indicator

Gutter display

Vertical split-pane control

Horizontal split-pane control

Editor

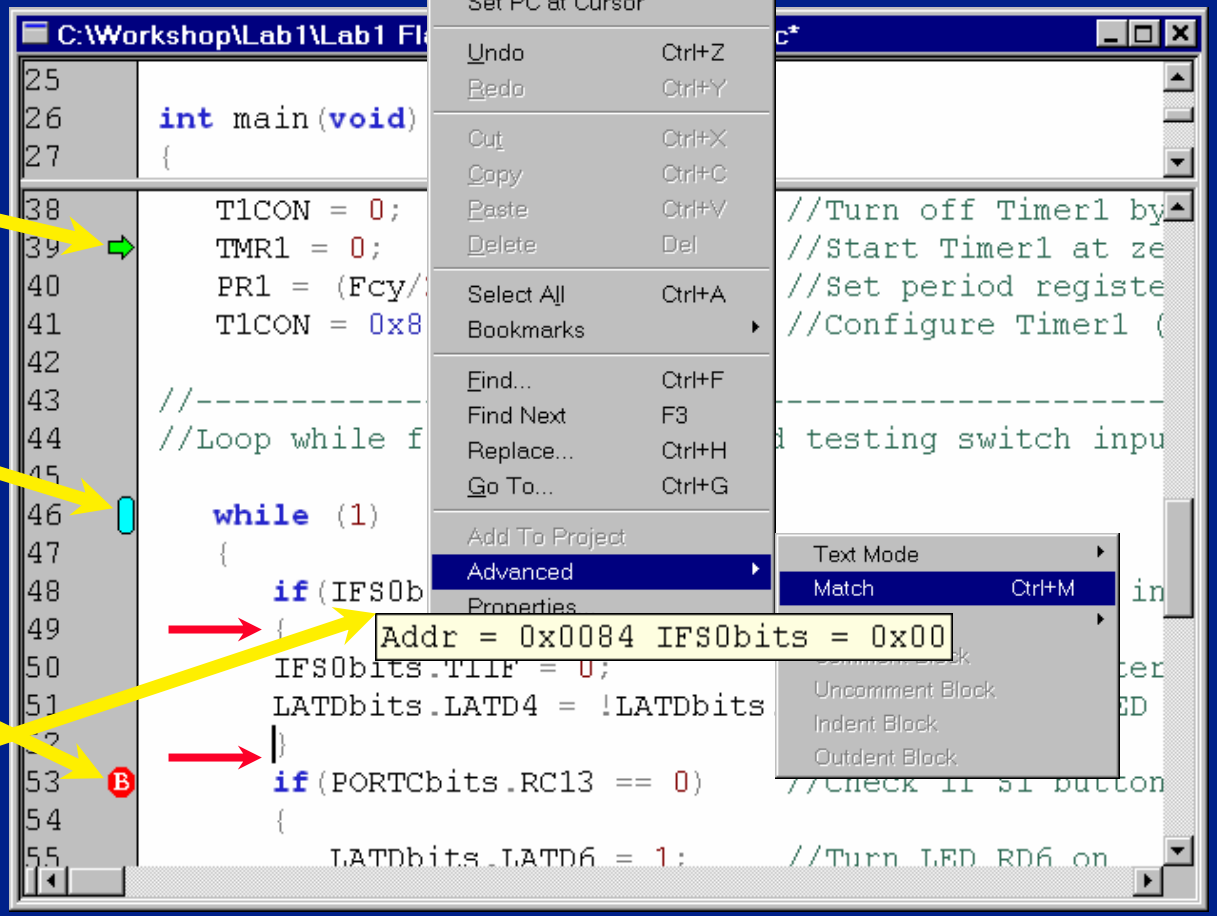
Right click menu

Execution point

Bookmark

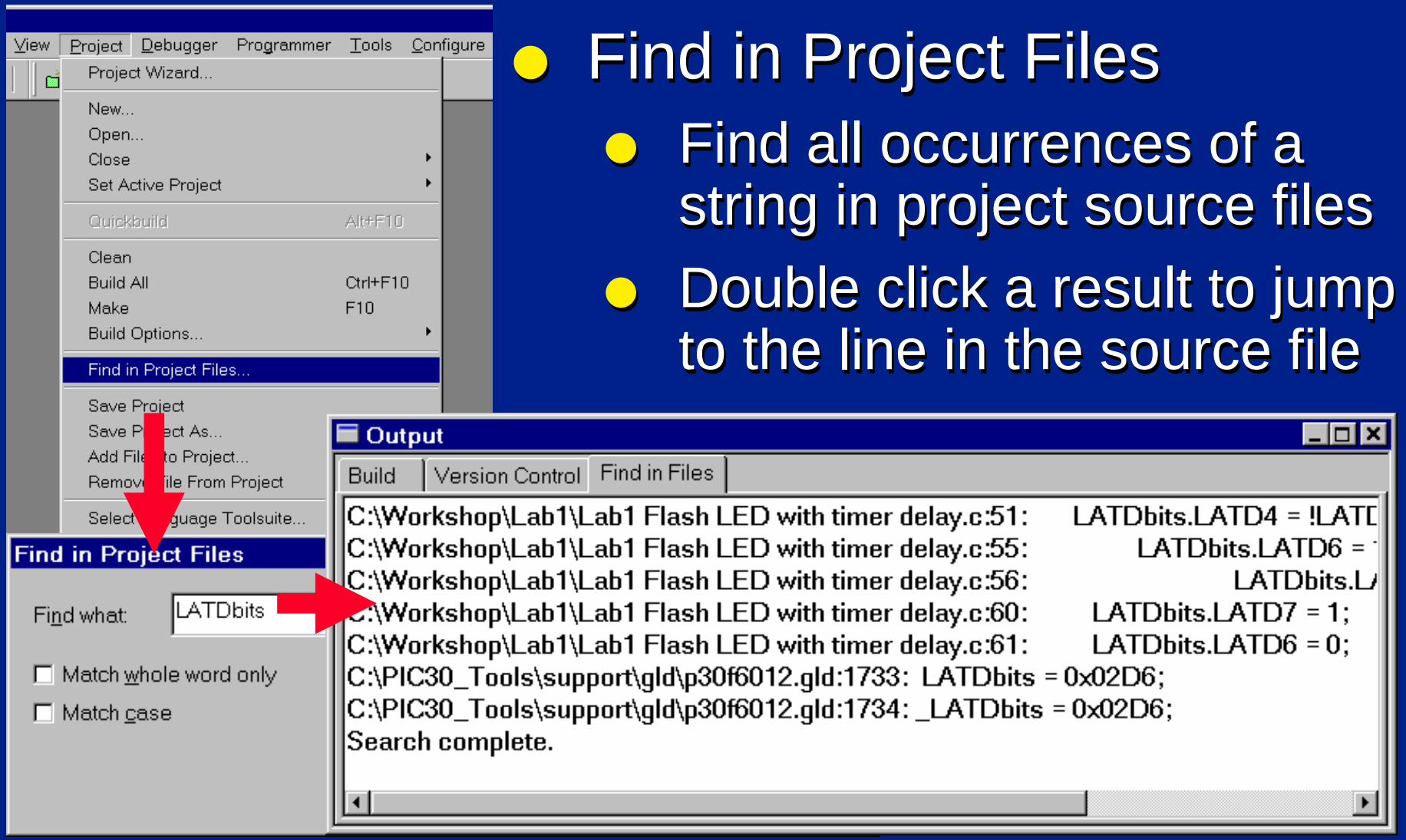
Breakpoint

Mouse over



Projects

- Find in Project Files
  - Find all occurrences of a string in project source files
  - Double click a result to jump to the line in the source file



Simulator

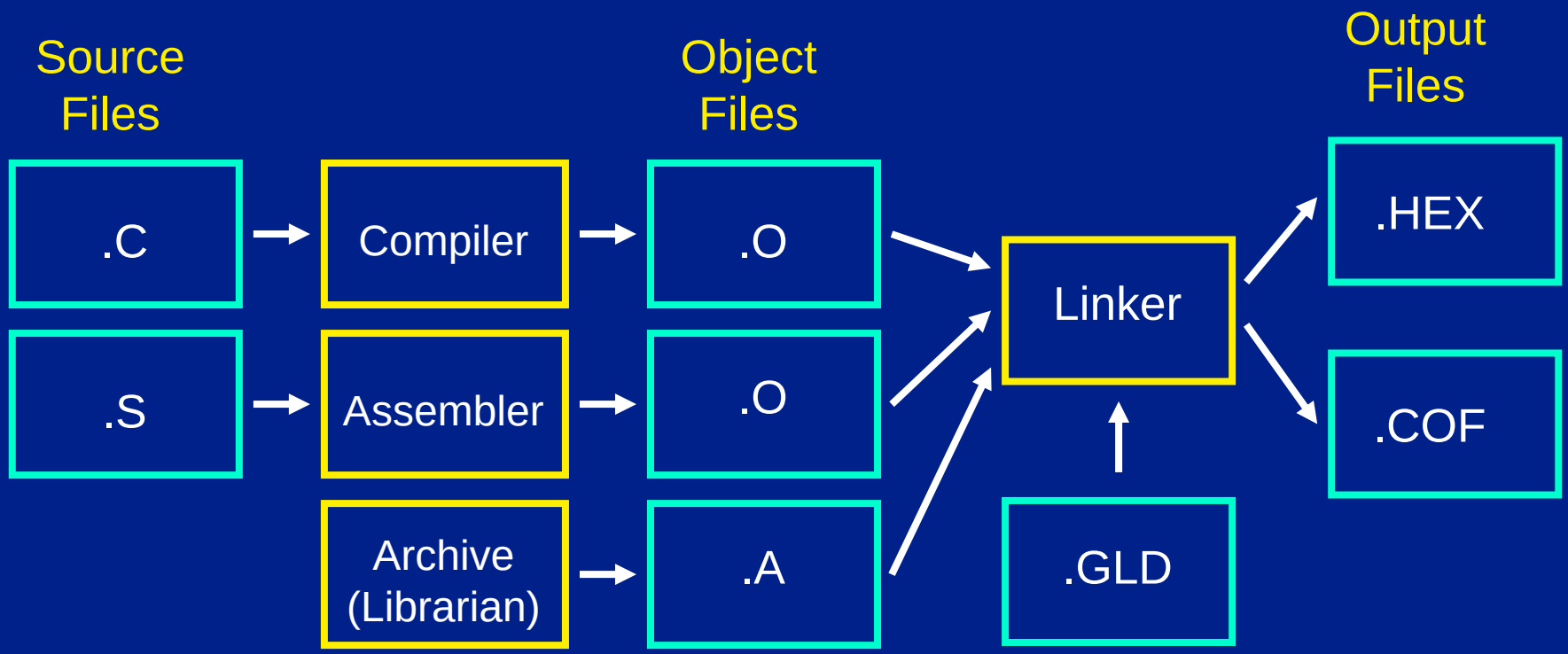
- Discrete event simulator
  - Operates on instruction cycle boundary
- Multiple breakpoints
- Stopwatch for time measurement
- Trace memory to see past execution
- View and modify variables and registers
- External stimulus

MPLAB® C30 C Compiler

- GNU based
- Comes with assembler, linker and librarian
- ANSI C compliant with standard libraries
- Optimized for dsPIC® architectural features
  - 16-bit native data type
  - Uses 3-operand instructions
  - Complex addressing modes
  - Efficient multi-bit shift operations
  - Efficient signed/unsigned comparisons

MPLAB® C30 C Compiler

Compiler, Assembler, Linker and Librarian



MPLAB® C30 C Compiler

- Libraries
  - Math - sin(), log(), sqrt(), etc.
  - Peripheral Driver - ADC, PWM, UART, etc.
  - DSP Algorithm - IIR filter, Convolution, etc.
  - TCP/IP and Soft Modem
  - Speech recognition
  - Noise and Echo Cancellation
  - Encryption
  - RTOS compatible - CMX, OSEK

Lab 1

Learn to Use dsPIC30F Tools in MPLAB® Integrated Development Environment

Lab 1

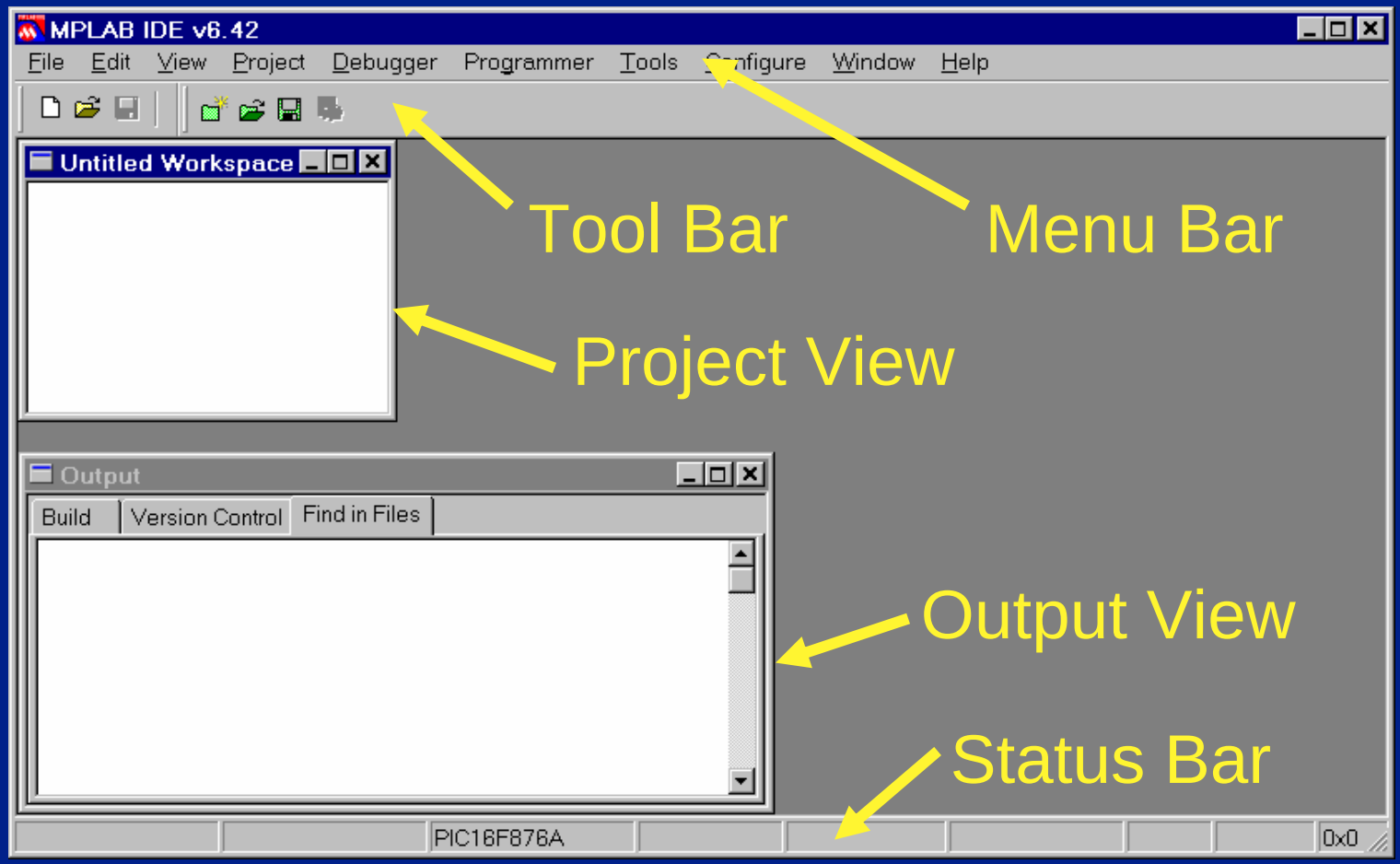
- dsPIC30F Tools we will use in the lab:
  - MPLAB® IDE v6.XX
  - MPLAB C30 C Compiler
  - MPLAB LINK30 linker
  - MPLAB ICD 2 In-Circuit Debugger
- Literature that will help you:
  - Quick Reference Card

Lab 1

- Learn to use dsPIC30F tools in MPLAB® IDE
- Four steps
  - Step 1 - Create a project in MPLAB IDE
  - Step 2 - Compile code with MPLAB C30 C Compiler
  - Step 3 - Program dsPIC30f6014 using MPLAB ICD 2
  - Step 4 - Run/Debug code on dsPICDEM™ 1.1 Demonstration Board

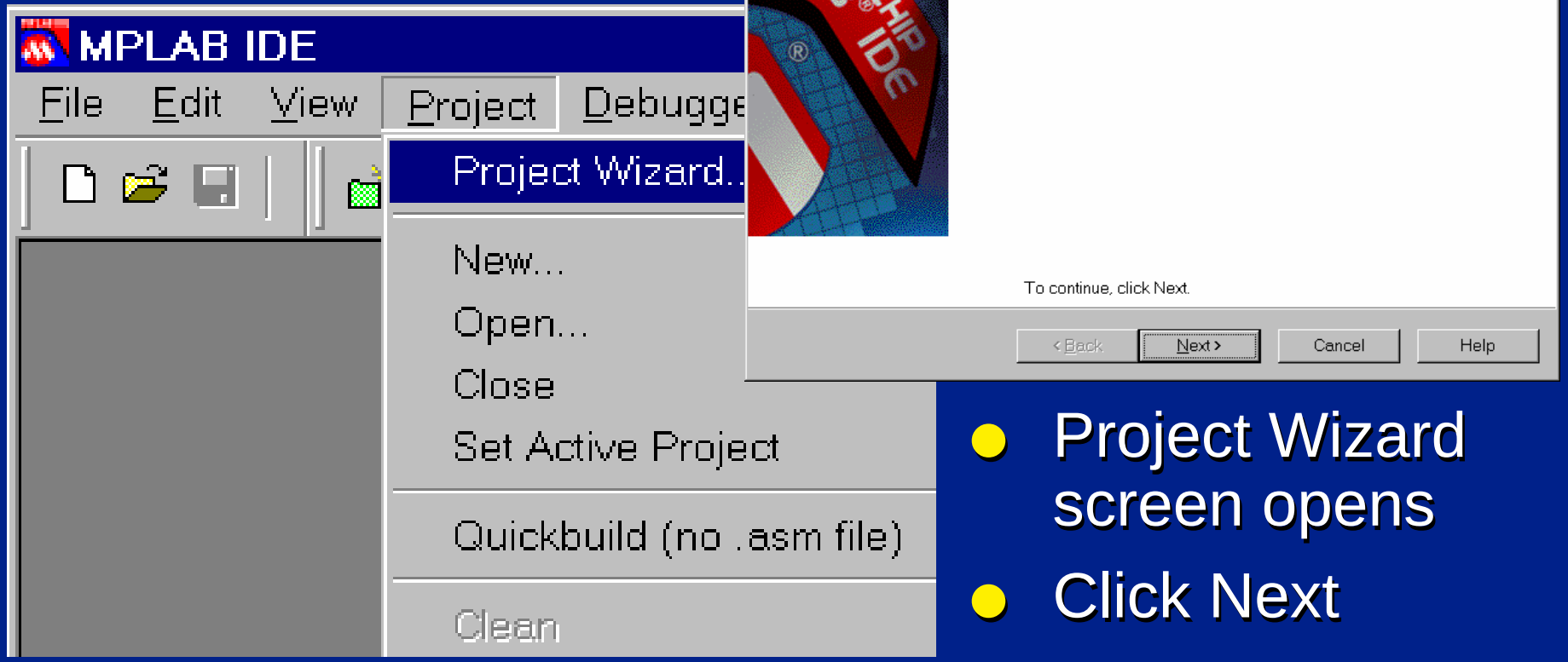
Lab 1

- Start MPLAB® IDE



Lab 1

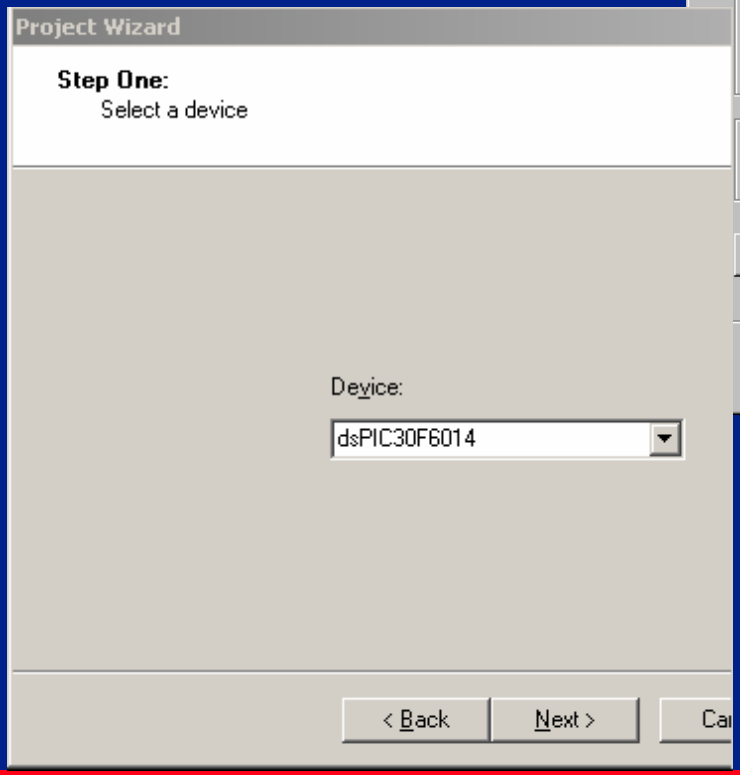
- Click Project Menu
- Select Project Wizard

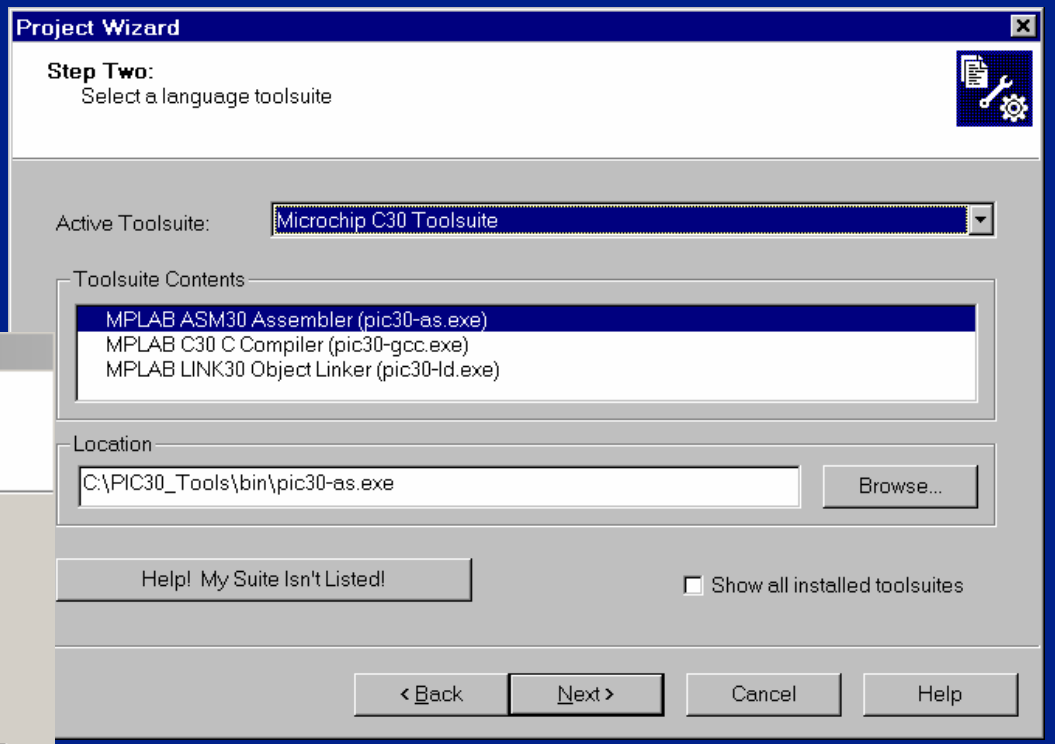


- Project Wizard screen opens
- Click Next

Lab 1

- Select dsPIC30F6014
- Click Next

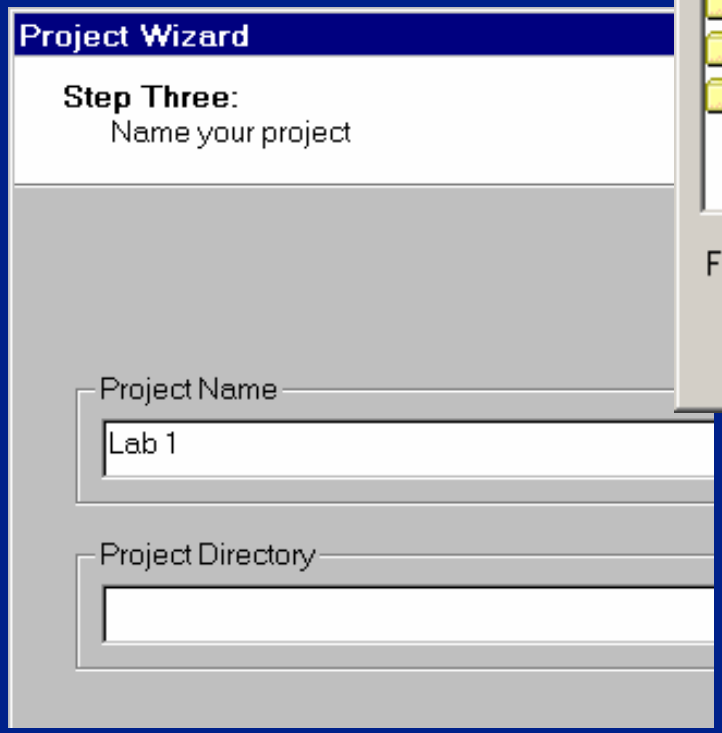


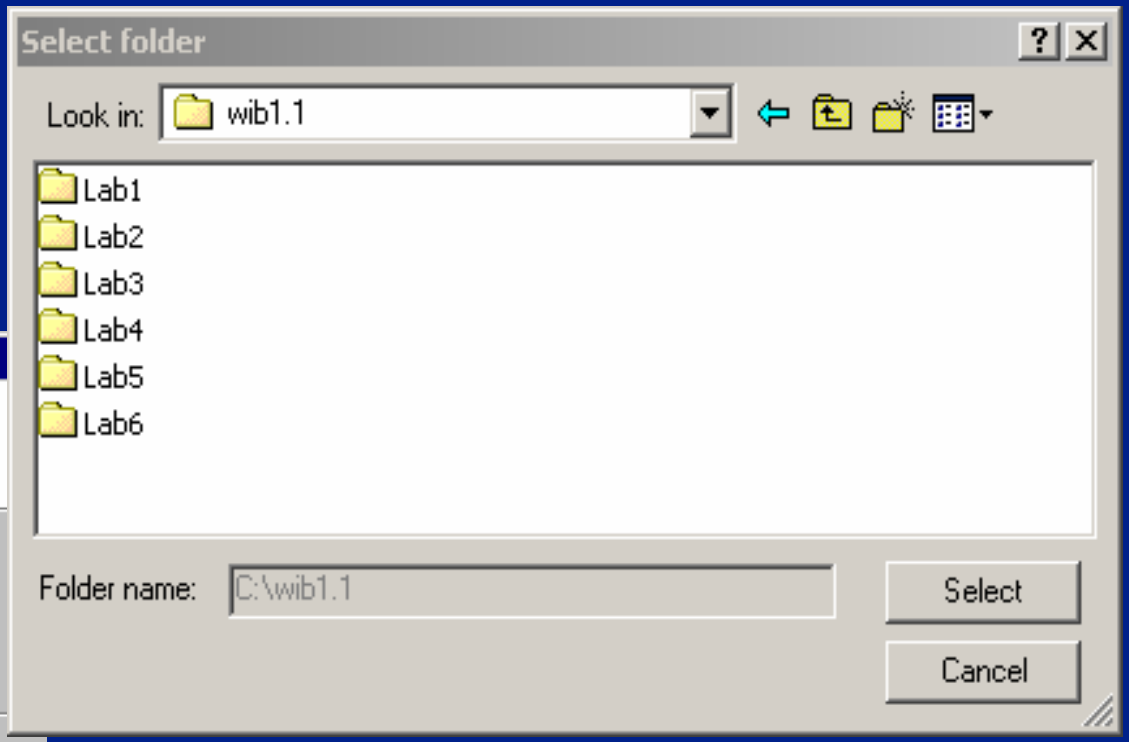


- Select Microchip C30 Toolsuite
- Click Next

Lab 1

- Type Lab 1 for Project Name
- Click Browse

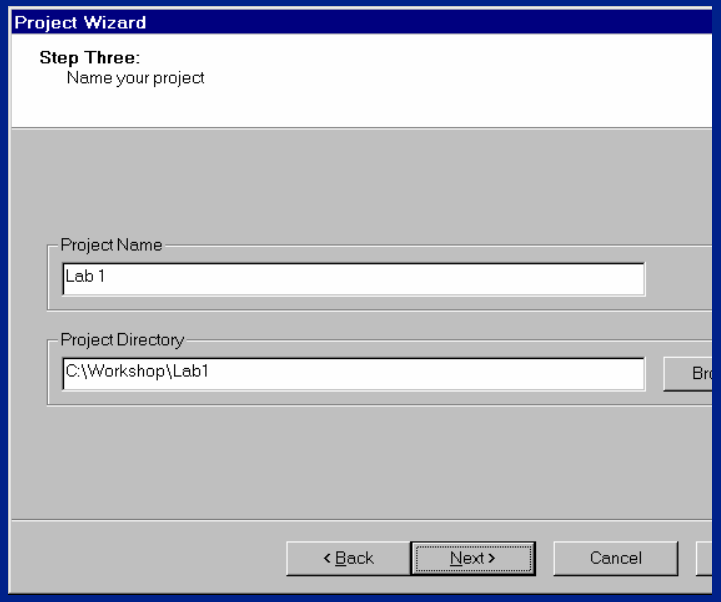


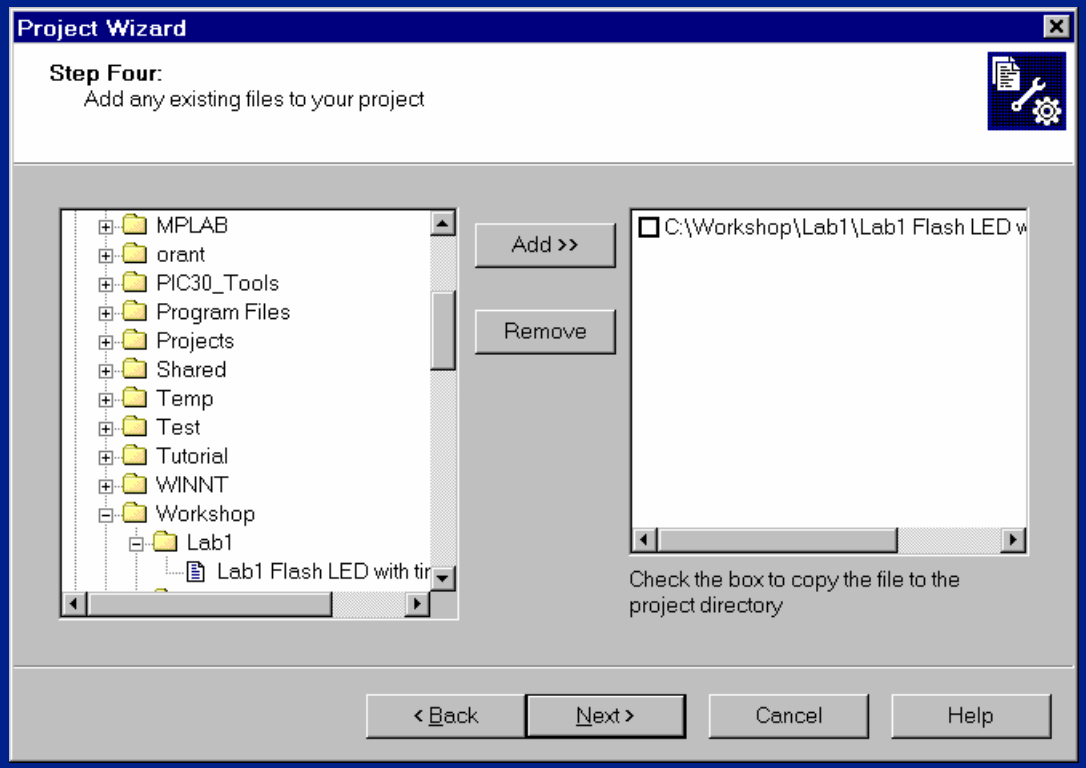


- Navigate to C:\WIB1.1\Lab1
- Click Select

Lab 1

- Click Next

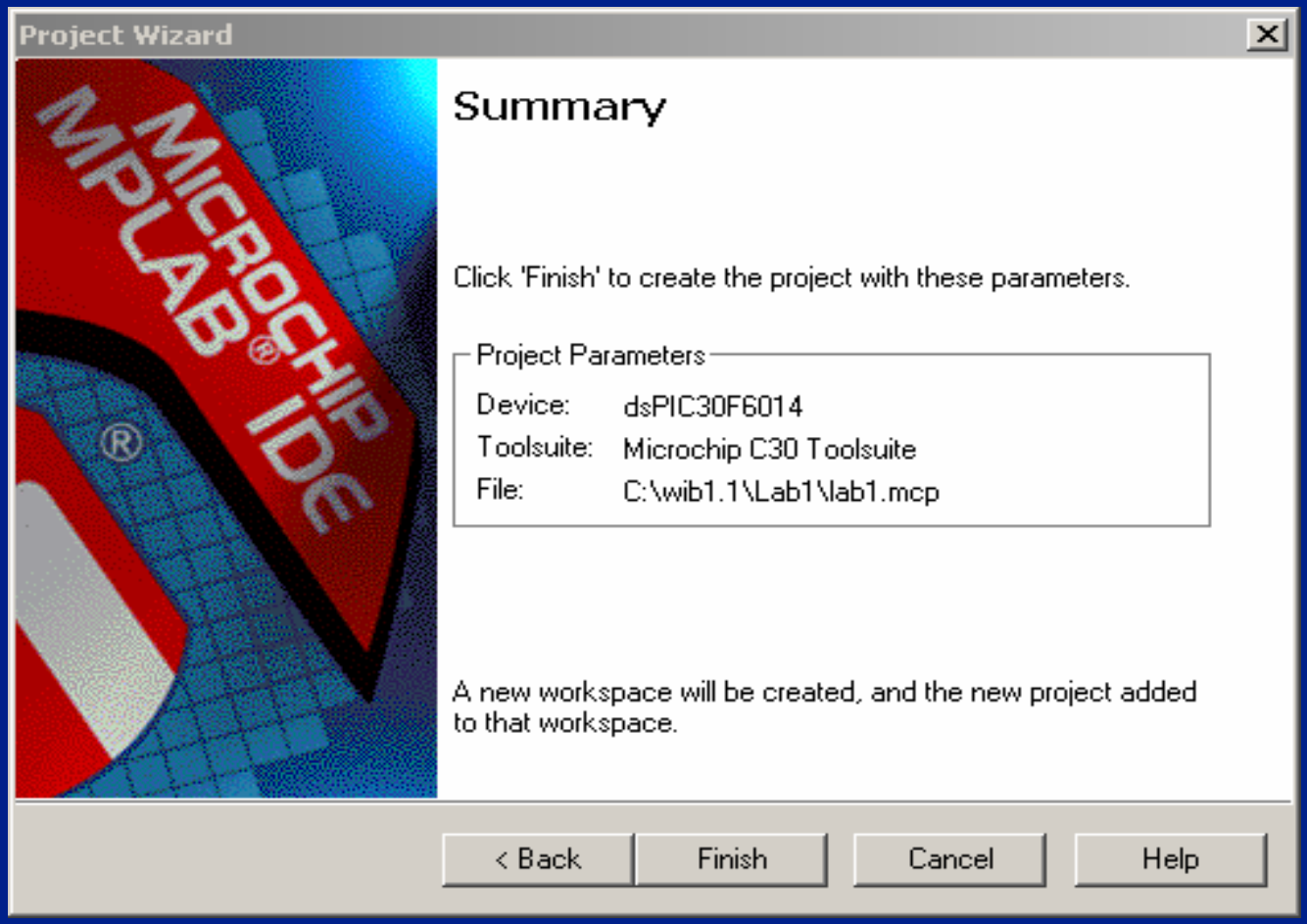




- Select Lab1 Flash LED with timer delay.c
- Click Add
- Click Next

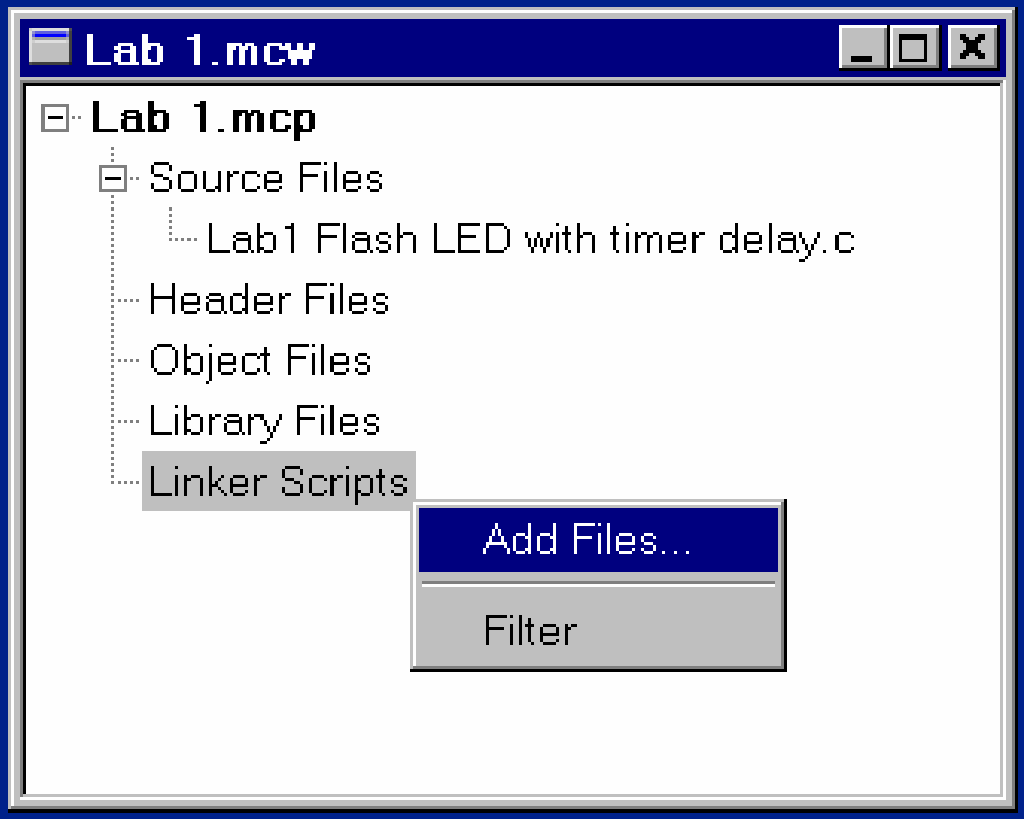
Lab 1

- Summary Screen Appears
- Click Finish



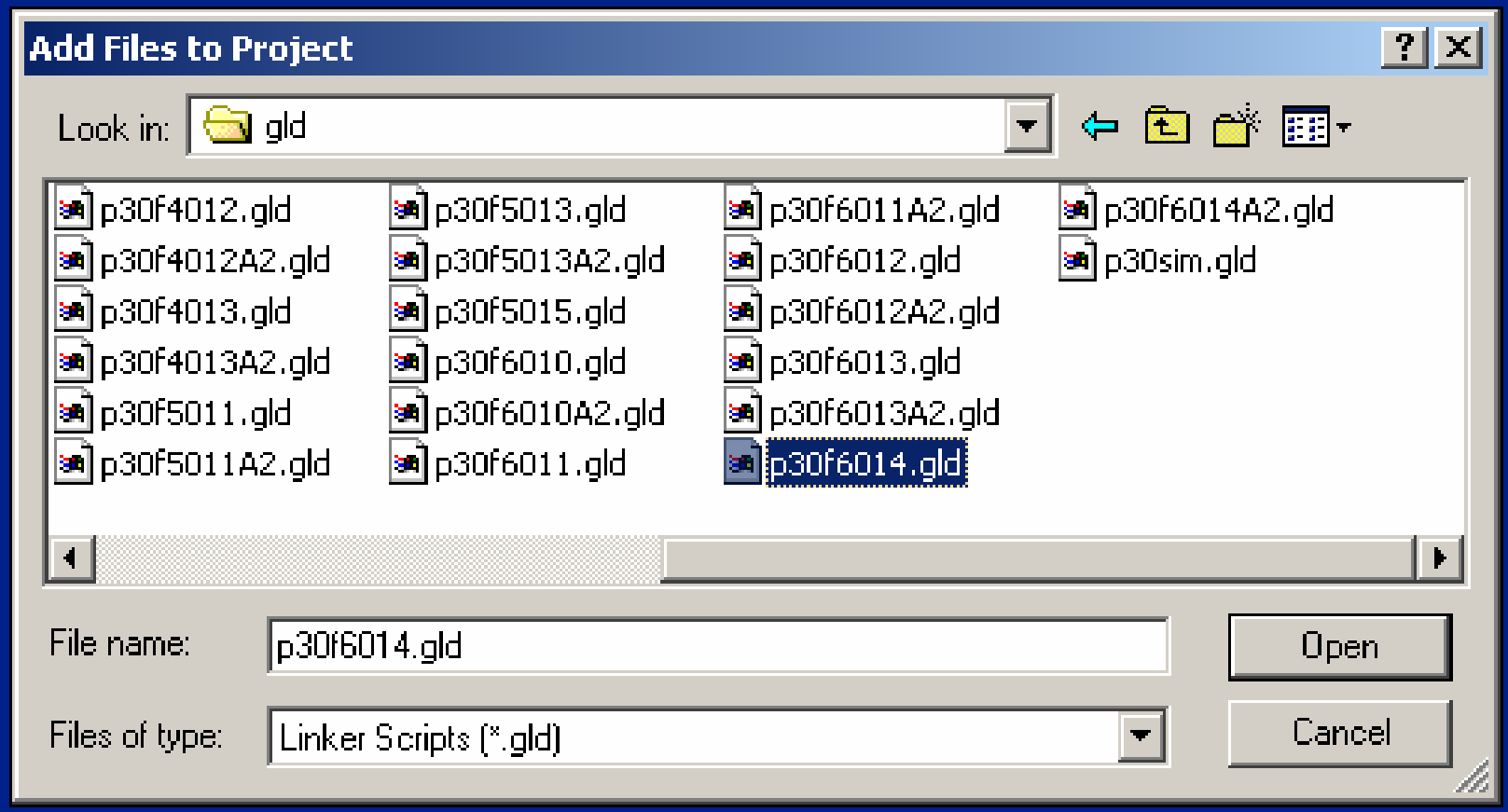
Lab 1

- Project Window Now Shows the Lab 1 Project



- Right Click Linker Scripts and Select Add Files

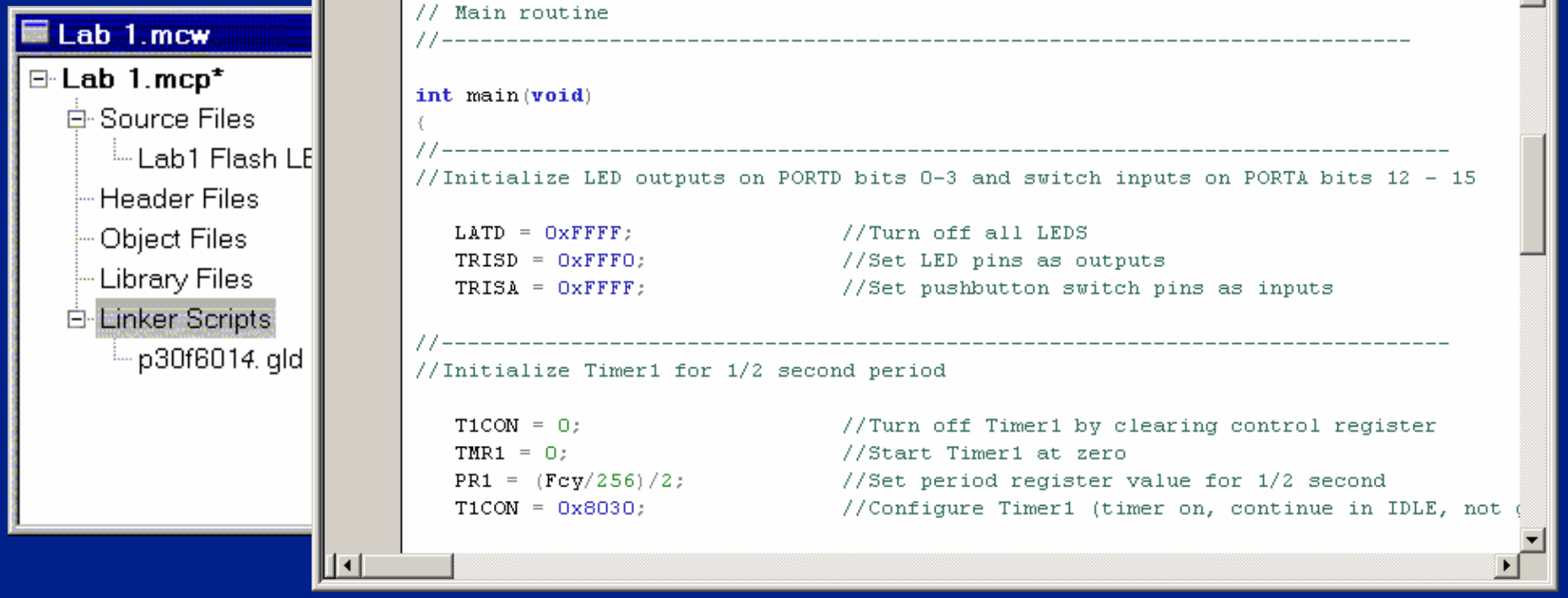
Lab 1



- Go to C:\PIC30_Tools\support\gld
- Select p30f6014.gld and Click Open

Lab 1

- Project Window Now Shows the Linker Script File



- Double Click a File to View and Edit the File

Step 2

Compile and Link
the Project

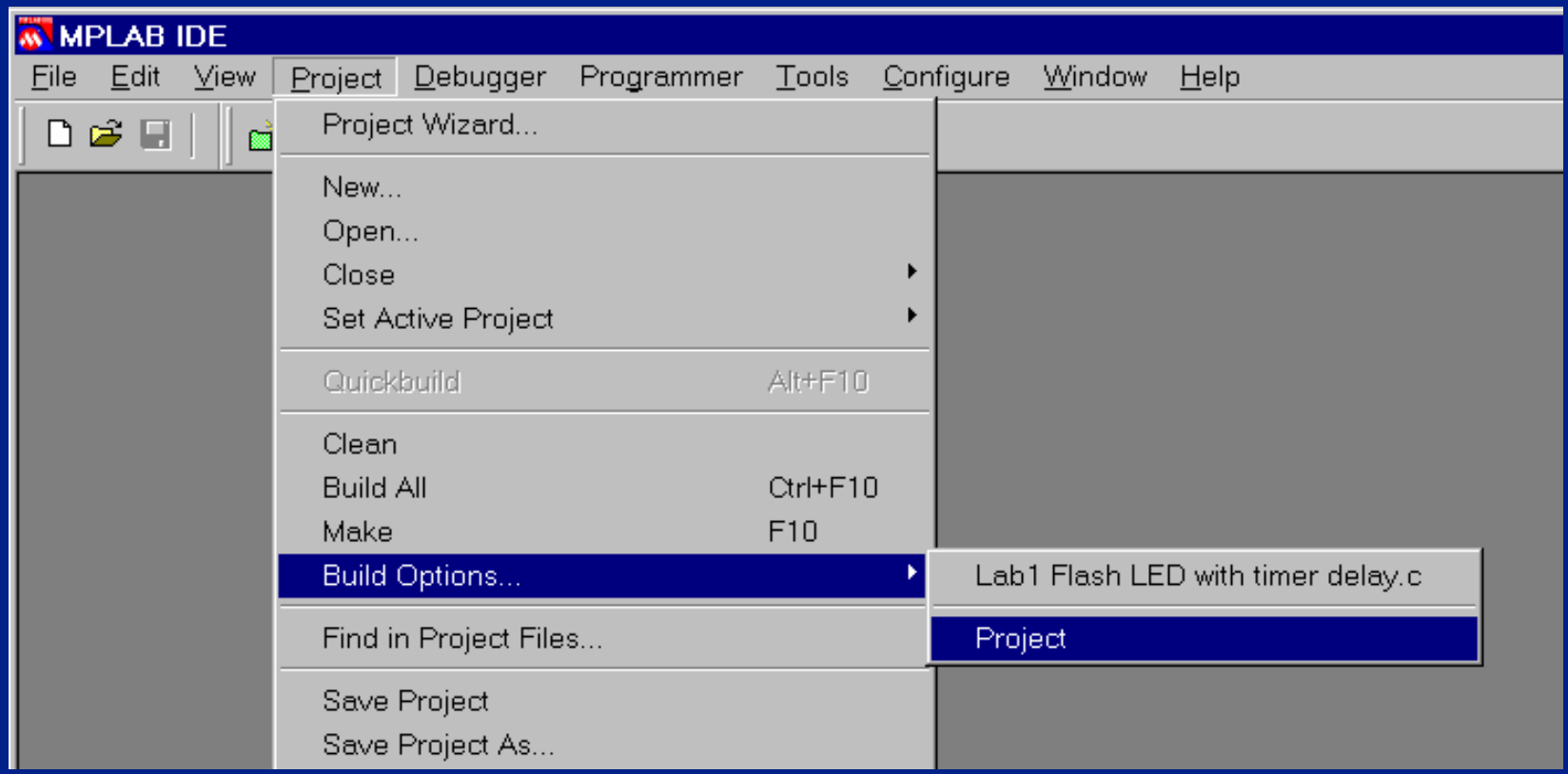
Lab 1

- Before We Compile and Link
  - Tell compiler where to find header file

    #include <p30f6014.h>
  - Tell linker to reserve memory for the MPLAB® ICD 2

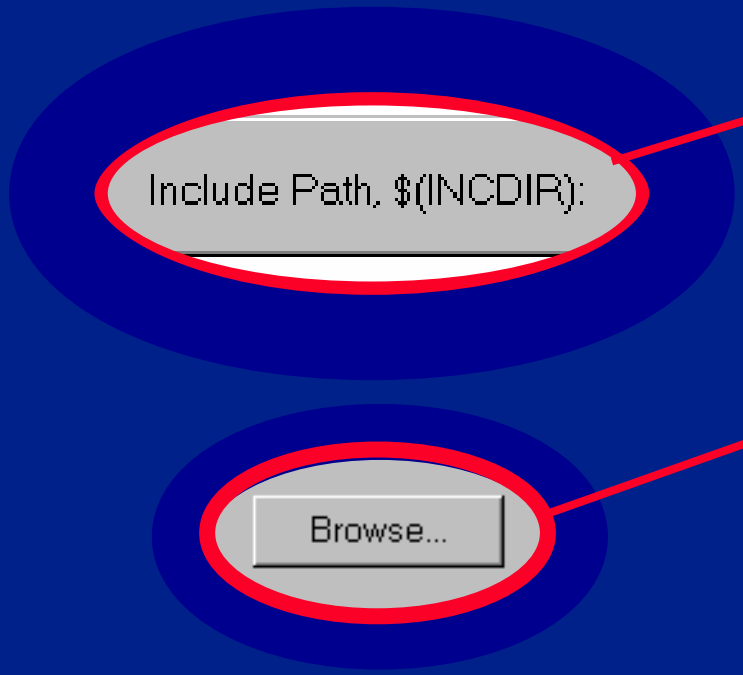
Lab 1

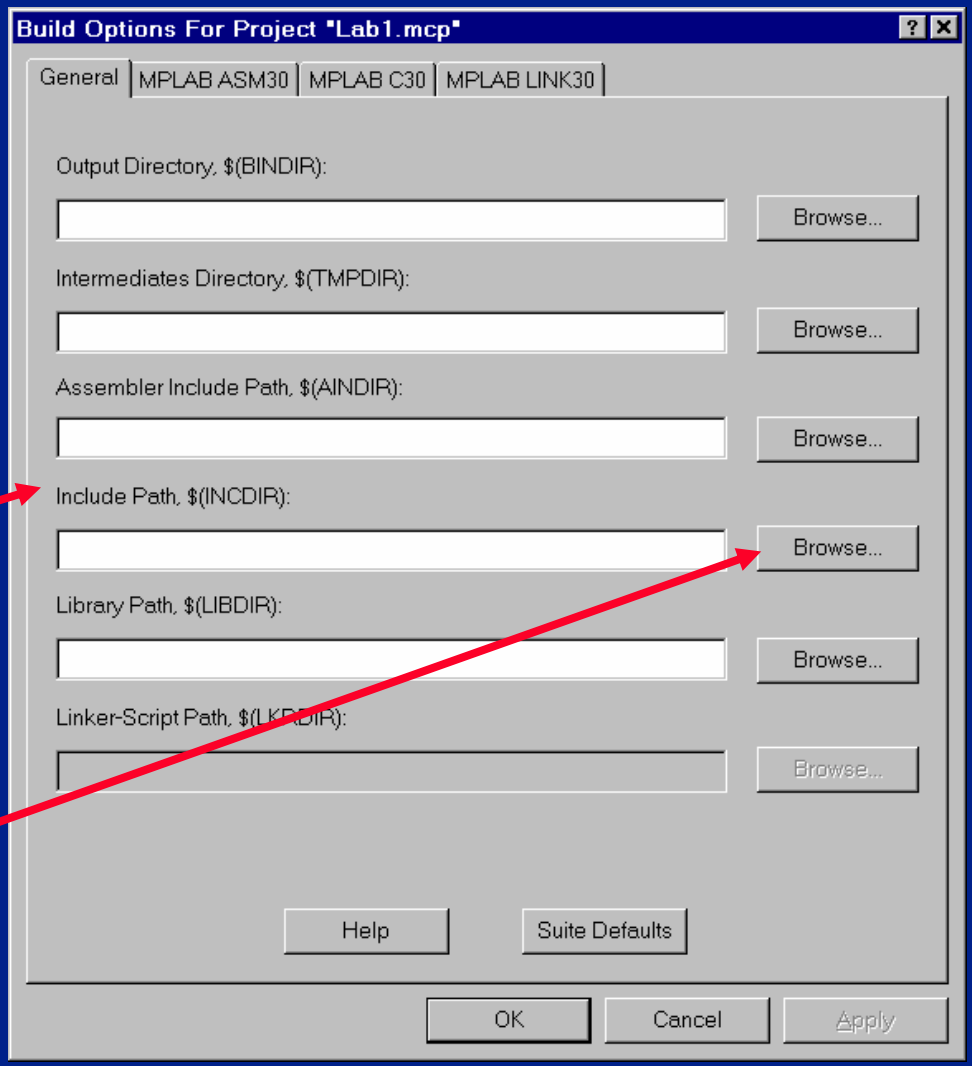
- Click Project Menu
- Select Build Options and Project



Lab 1

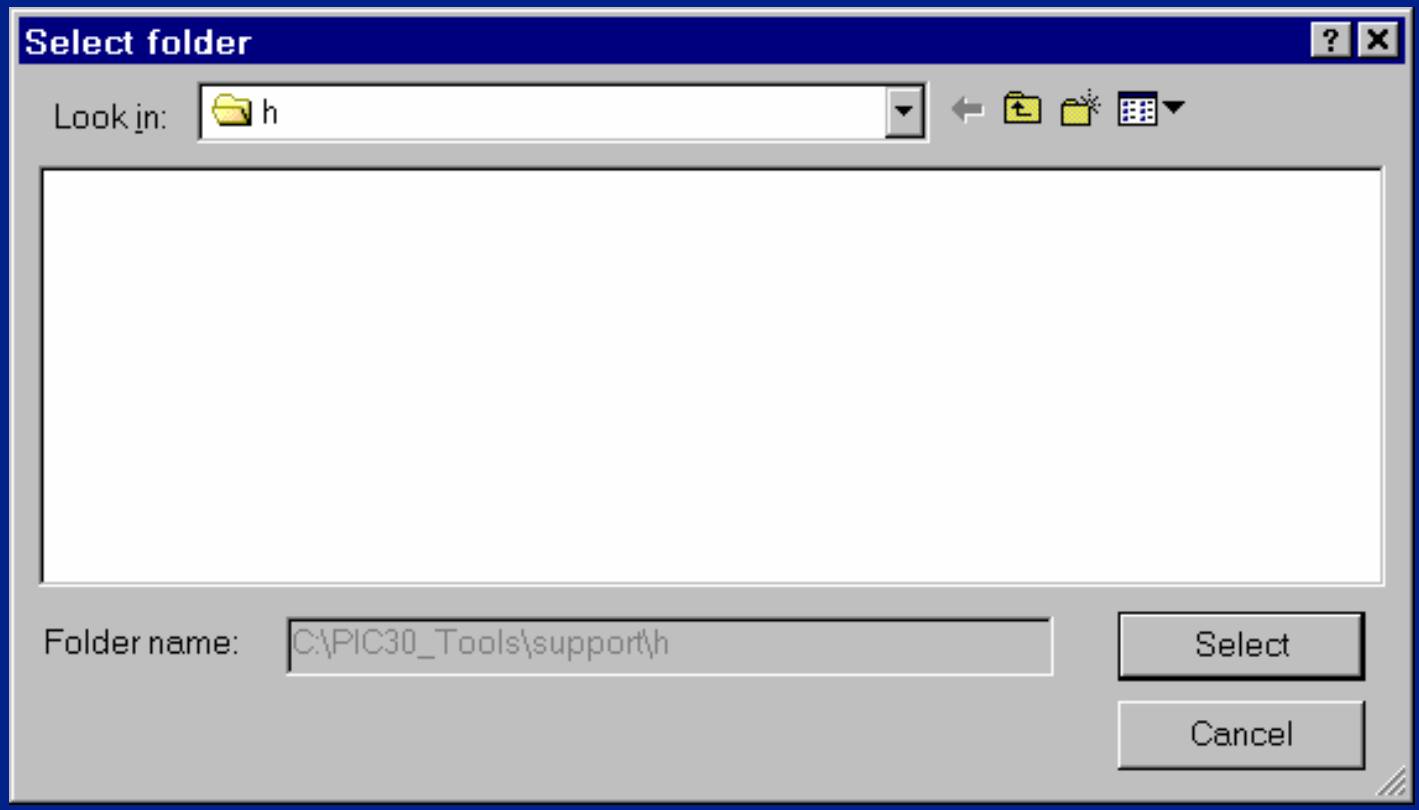
- Find Include Path
- Click Browse





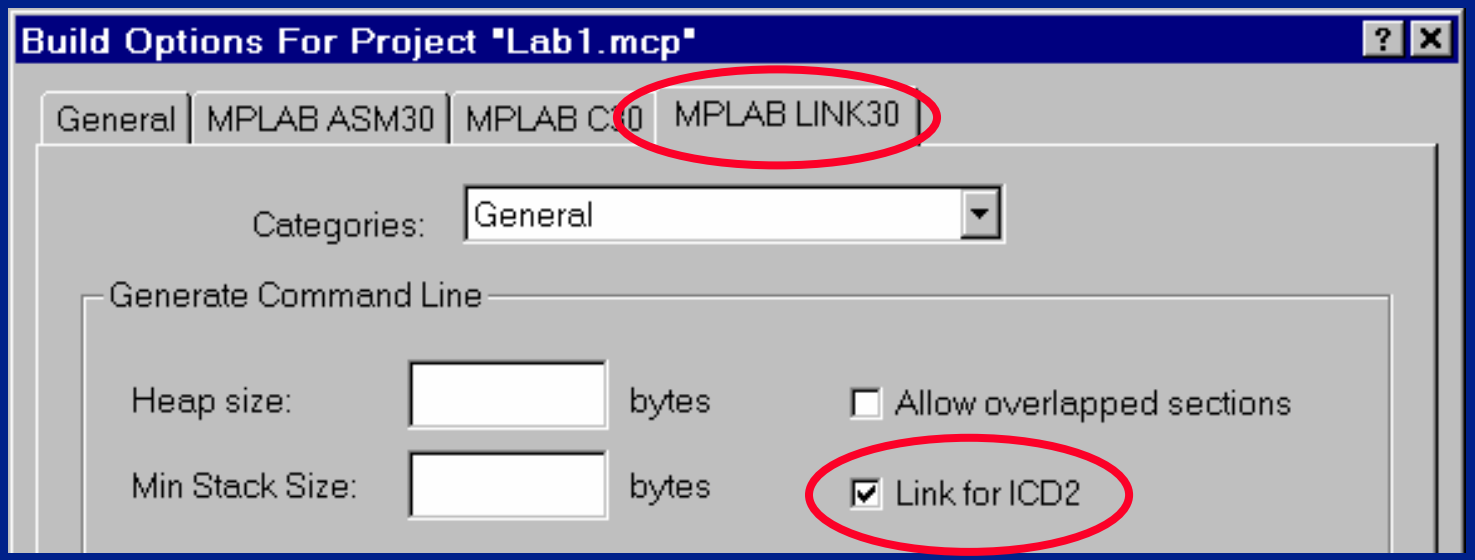
Lab 1

- Go to C:\PIC30_Tools\support\h
- Click Select



Lab 1

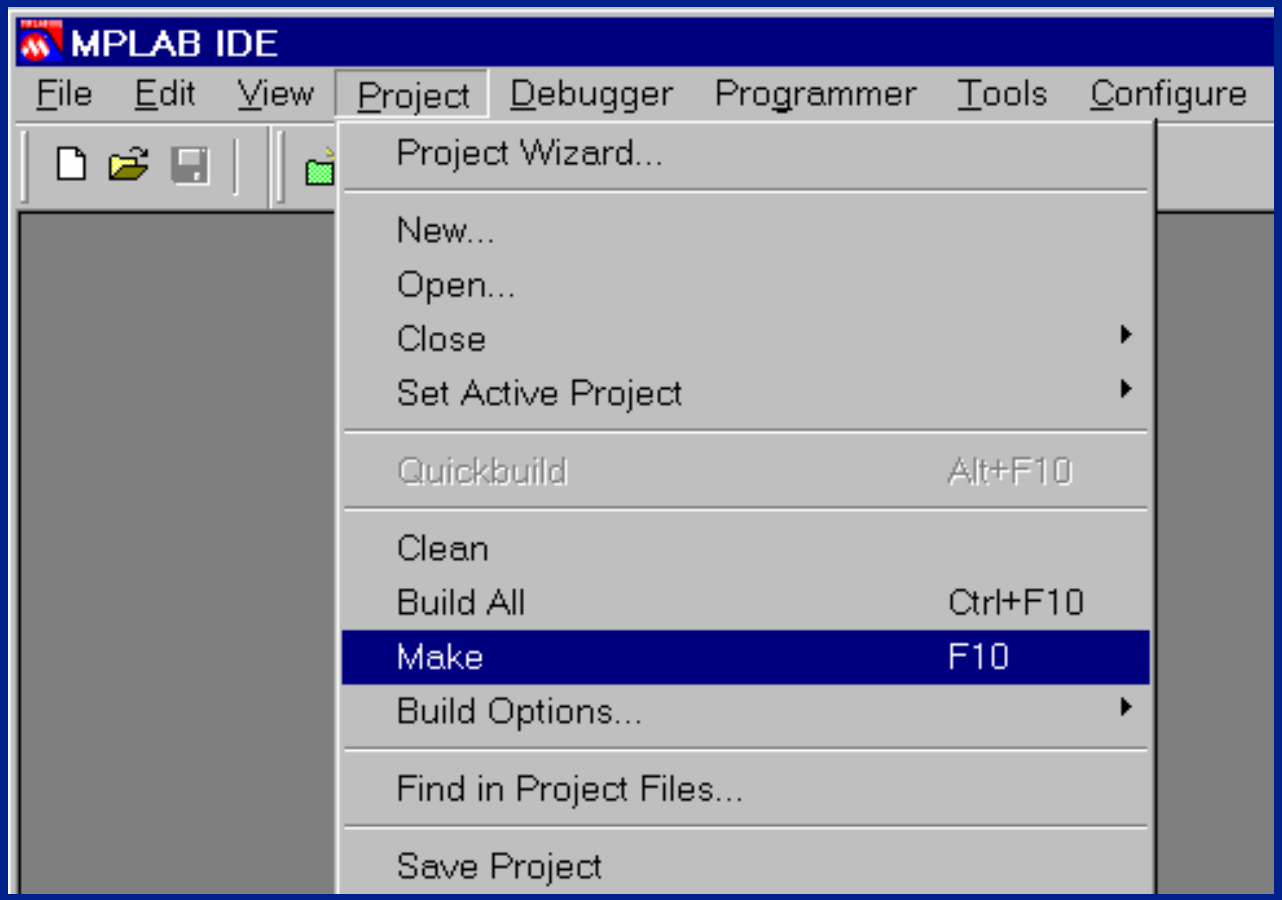
- Include Path Now Points to Header Files



- Click MPLAB® LINK30 Tab
- Select Link for ICD2 and Click OK

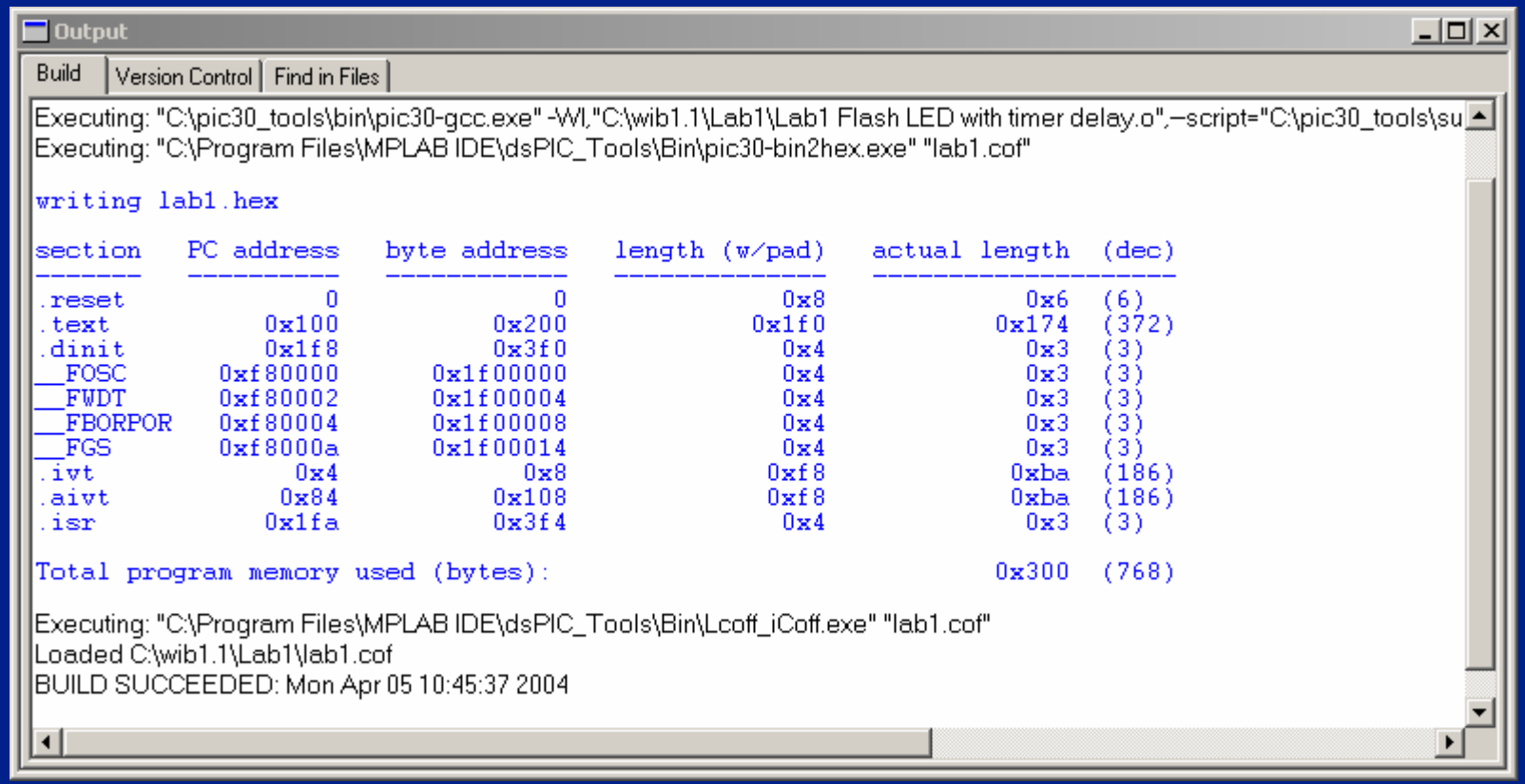
Lab 1

- Click Project Menu
- Select Make



Lab 1

- Build Results are Shown in Output Window

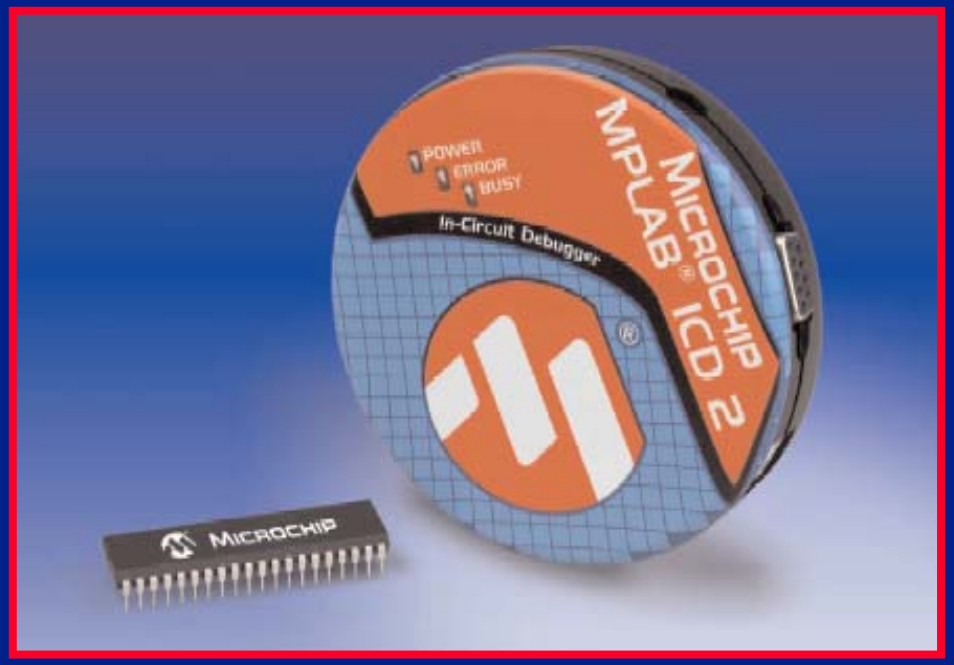


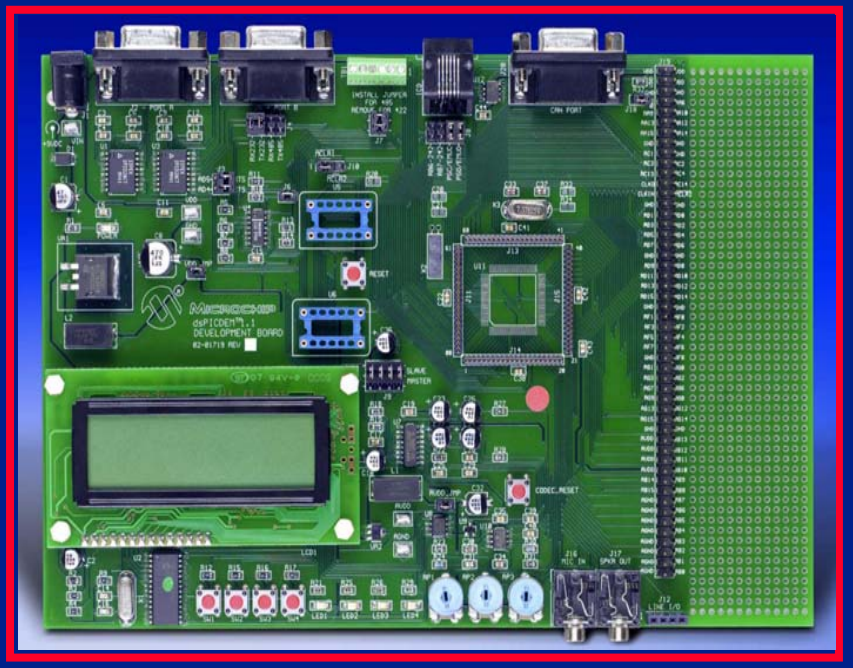
Step 3

Use the MPLAB® ICD 2
to Program and Run
the Code

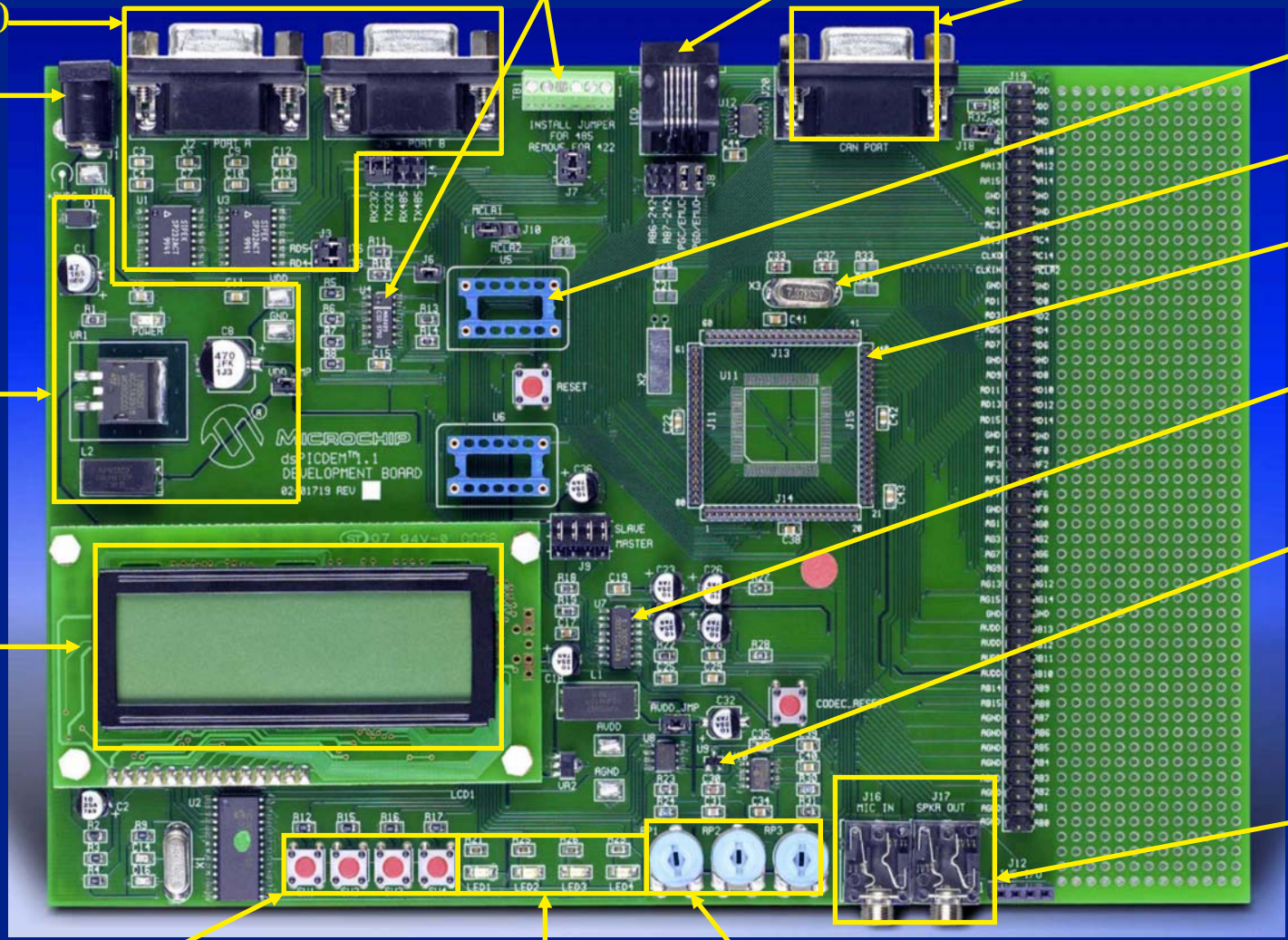
Lab 1

- Hardware
  - MPLAB® ICD 2
  - dsPICDEM™ 1.1 Development Board





dsPICDEM™ 1.1 Development Board

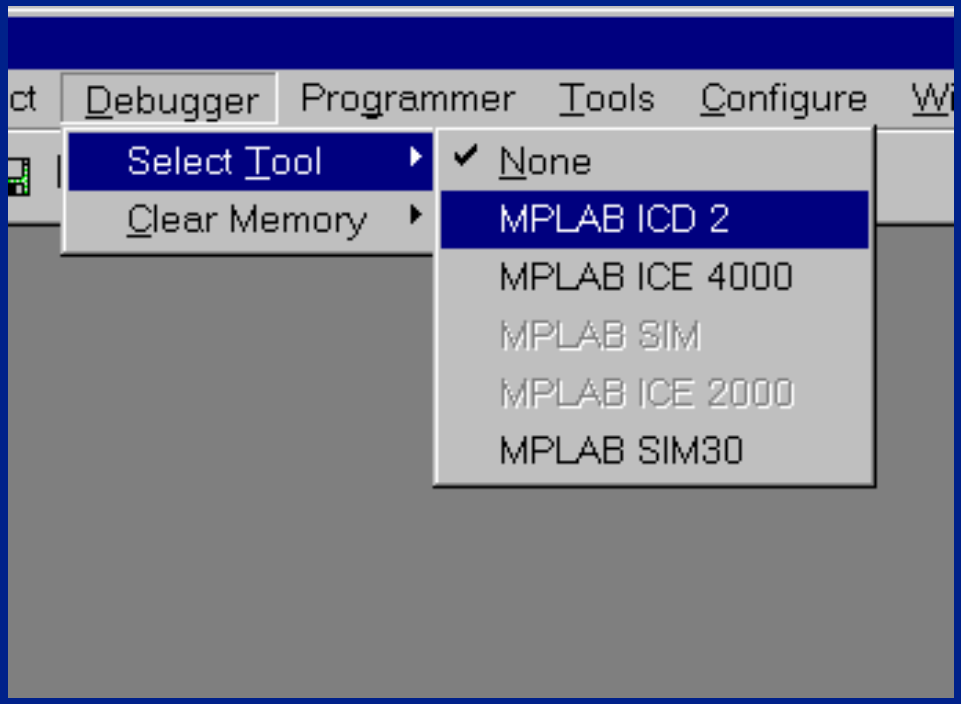


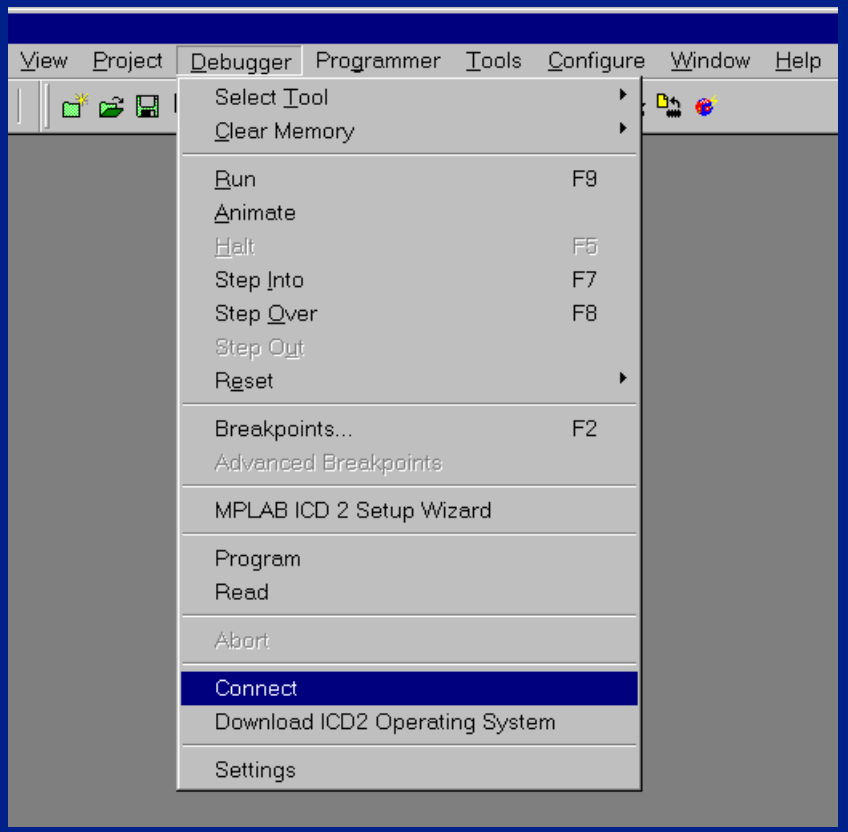
Lab 1

- Set up the Hardware
  - Connect power the to dsPICDEM™ 1.1 Development Board.
  - Connect USB cable from PC to MPLAB® ICD 2
  - Connect cable from MPLAB ICD 2 to dsPICDEM 1.1 Development Board

Lab 1

- Click Debugger Menu
- Choose Select Tool and MPLAB® ICD 2

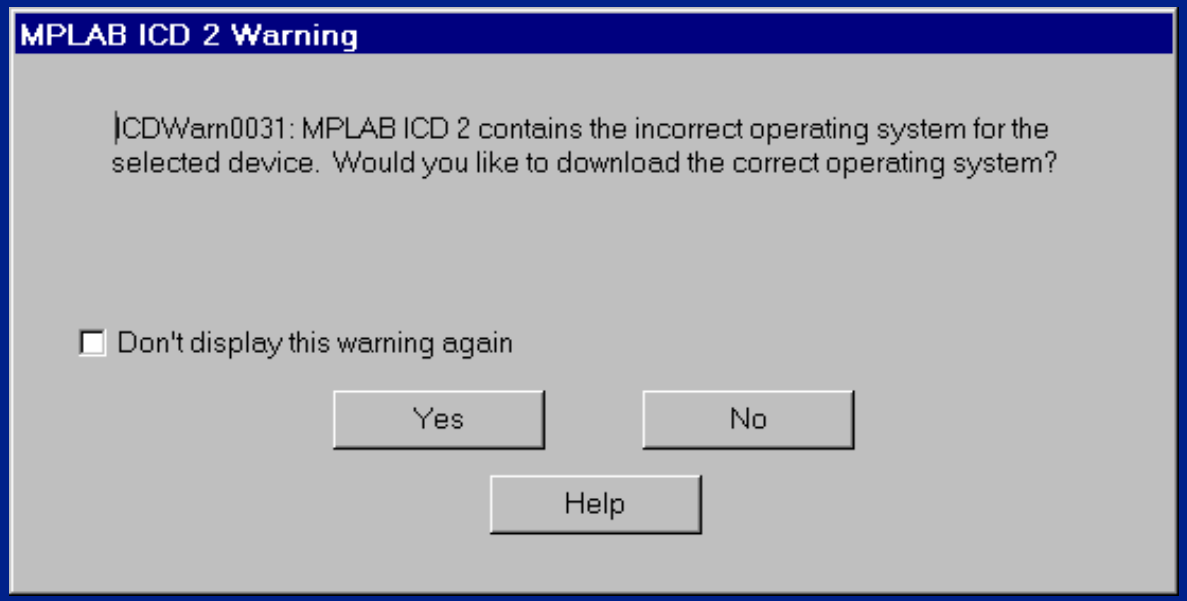




- Click Debugger menu
- Choose Connect

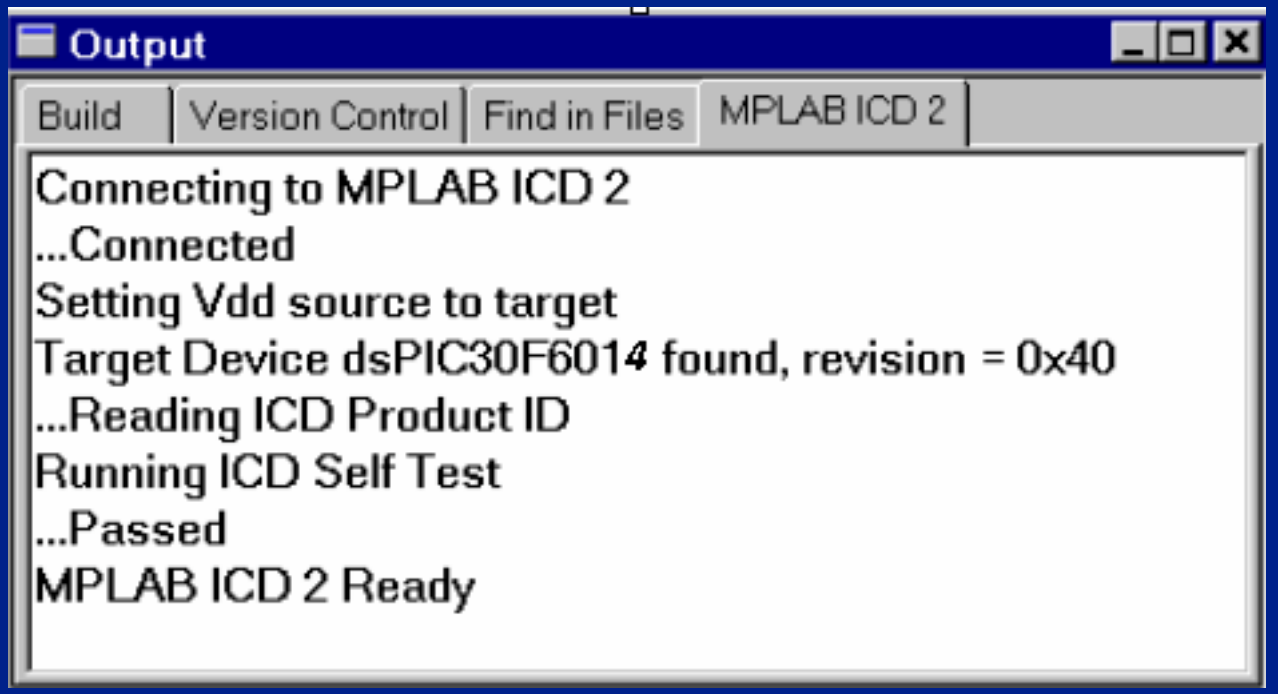
Lab 1

- Operating System Download
  - MPLAB® ICD 2 may ask to download a new operating system
  - Click Yes



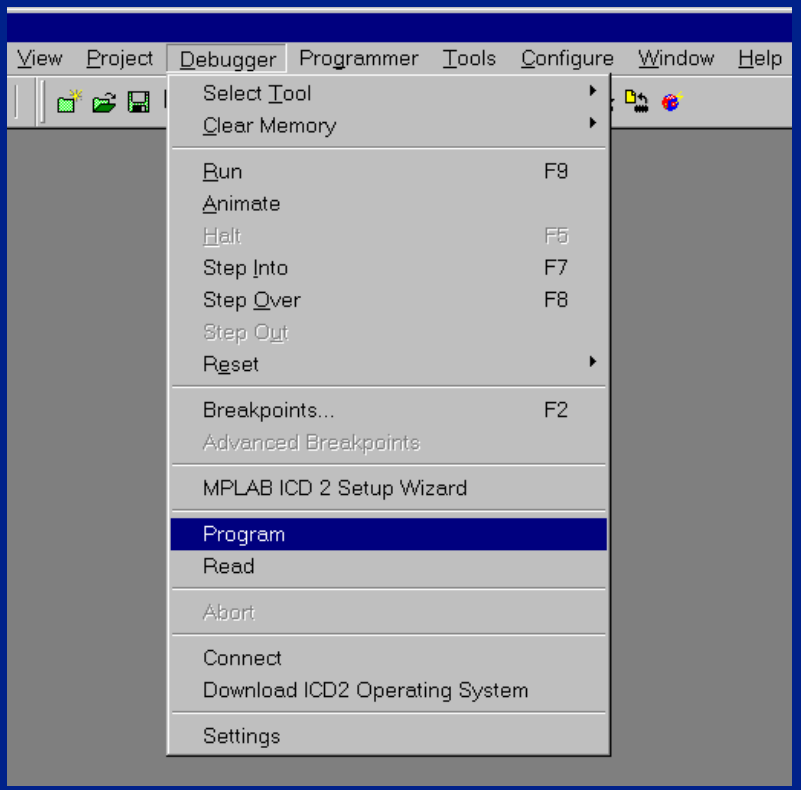
Lab 1

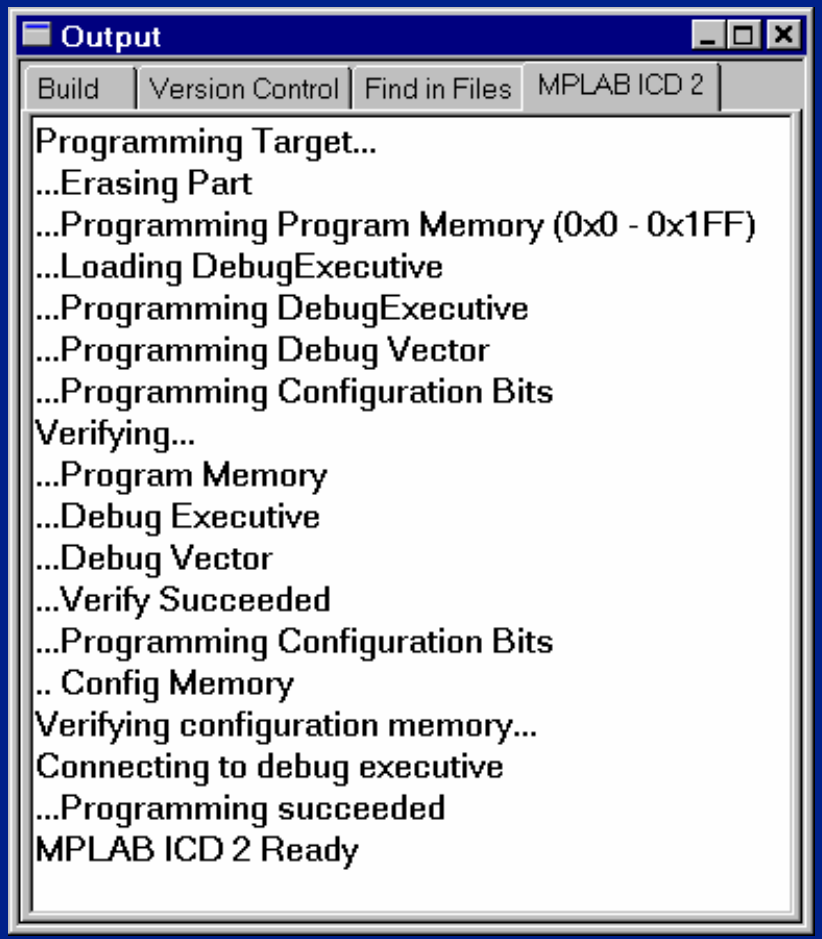
- Output Window
  - MPLAB® ICD 2 connected to the target device
  - Identified it as a dsPIC30F6014



Lab 1

- Click Debugger Menu
- Choose Program

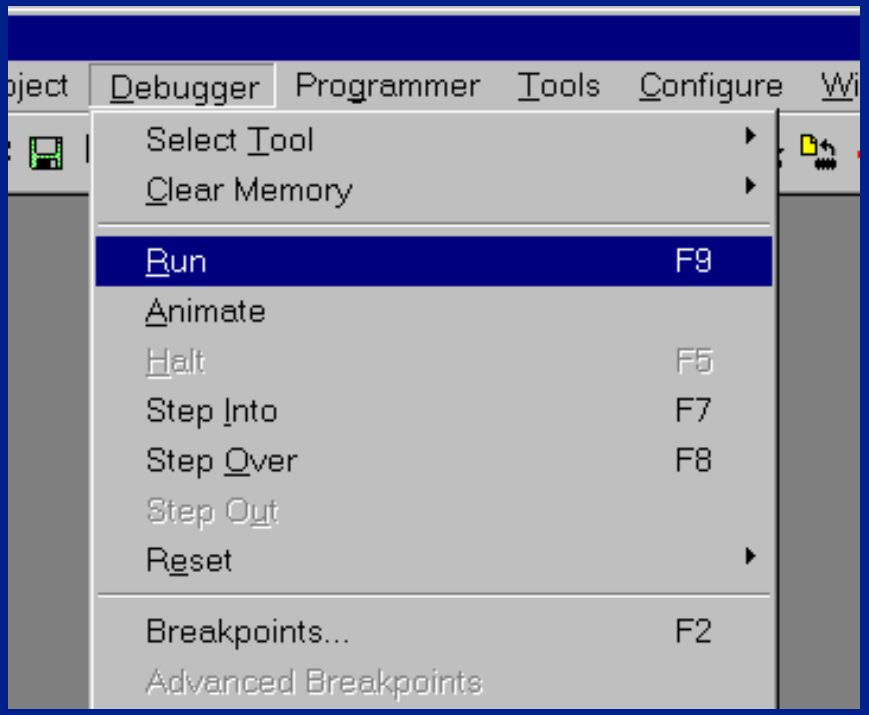


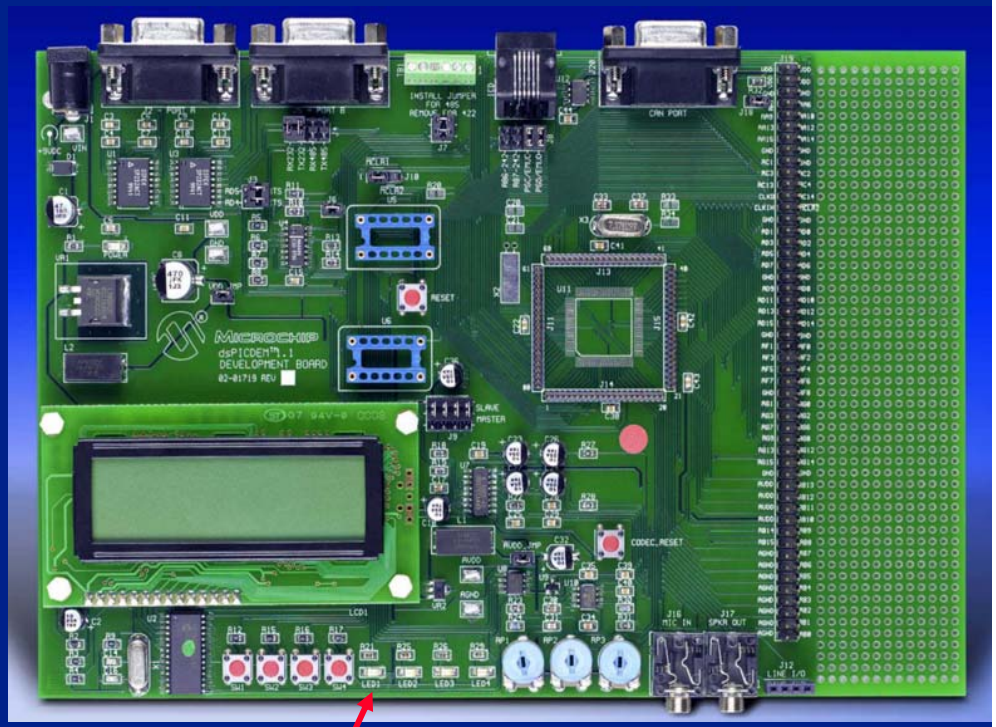


- Output window

Lab 1

- Click Debugger Menu
- Choose Run





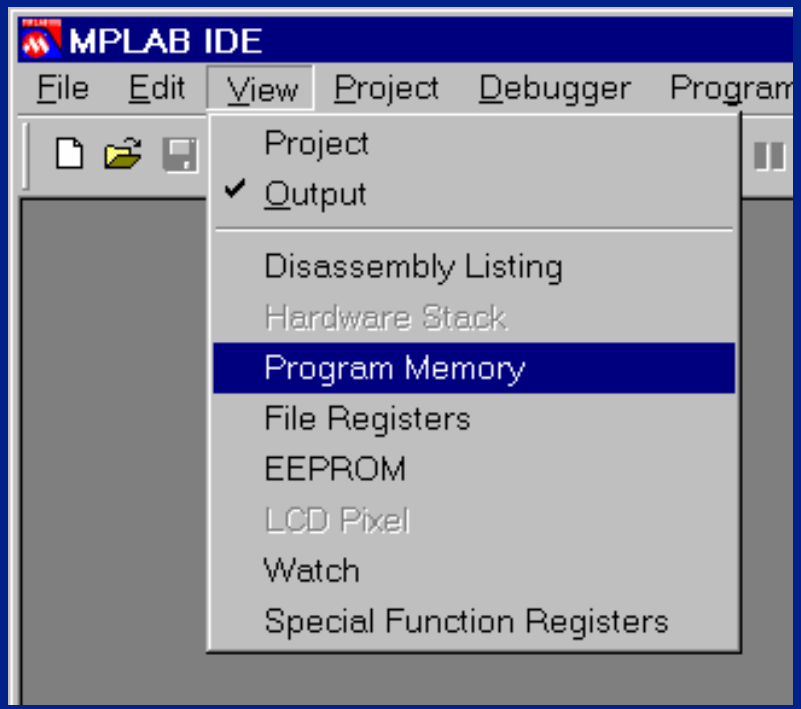
- LED1 flashes at a 1 Hz rate.

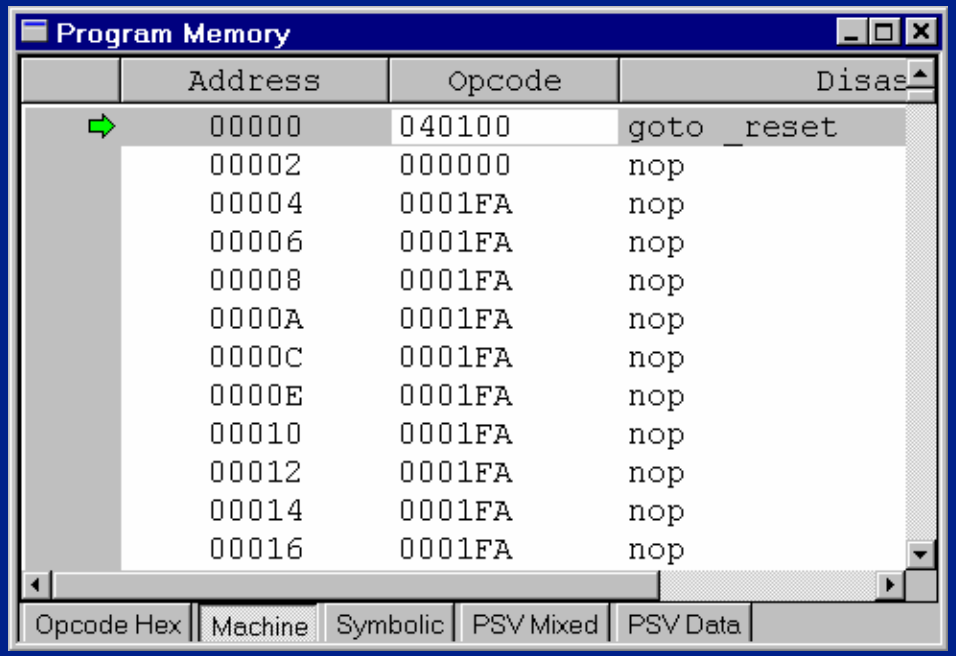
Step 4

Use the MPLAB® ICD 2 debugging features

Lab 1

- Click View Menu
- Choose Program Memory

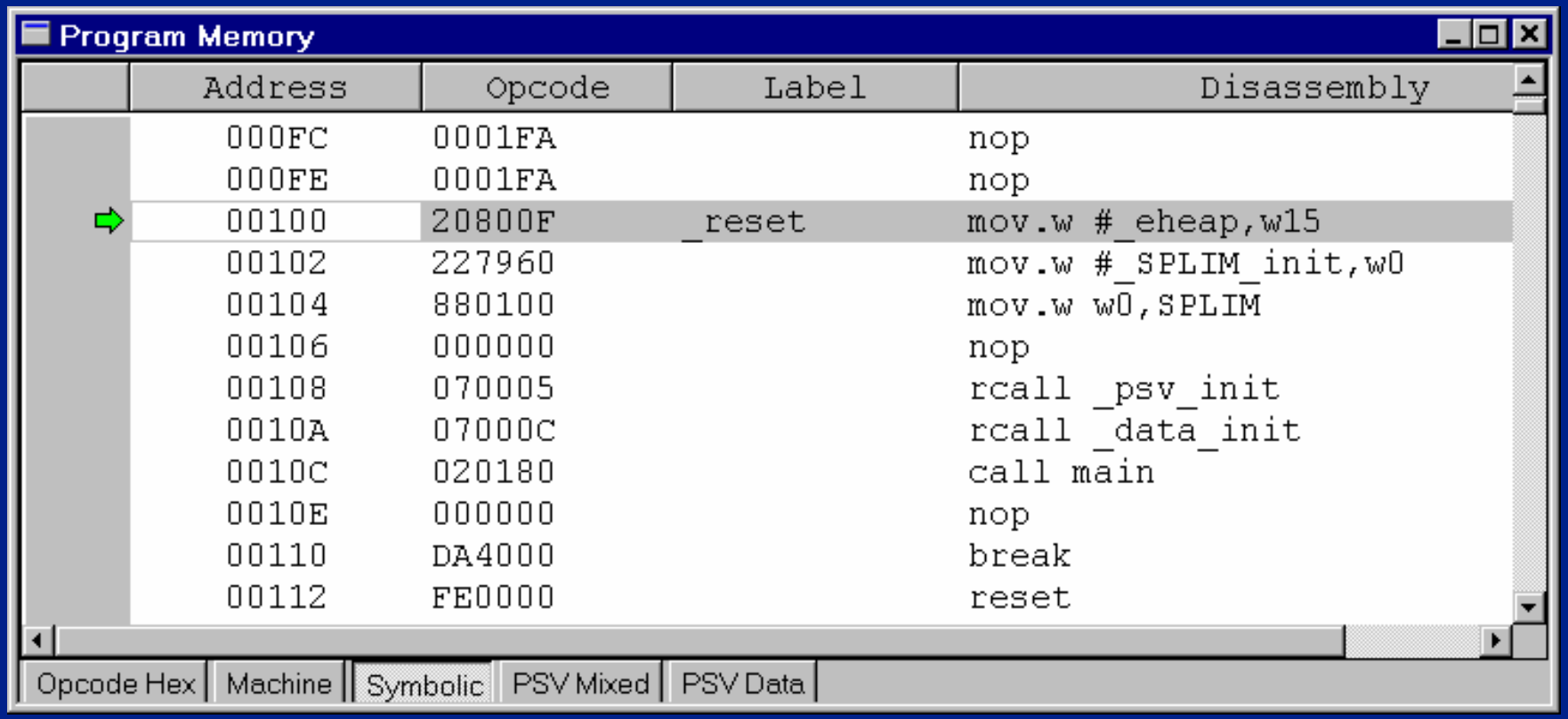




- Press F5 to Halt
- Press F6 to Reset
- Green Arrow Shows Code at Reset Vector 0x00000

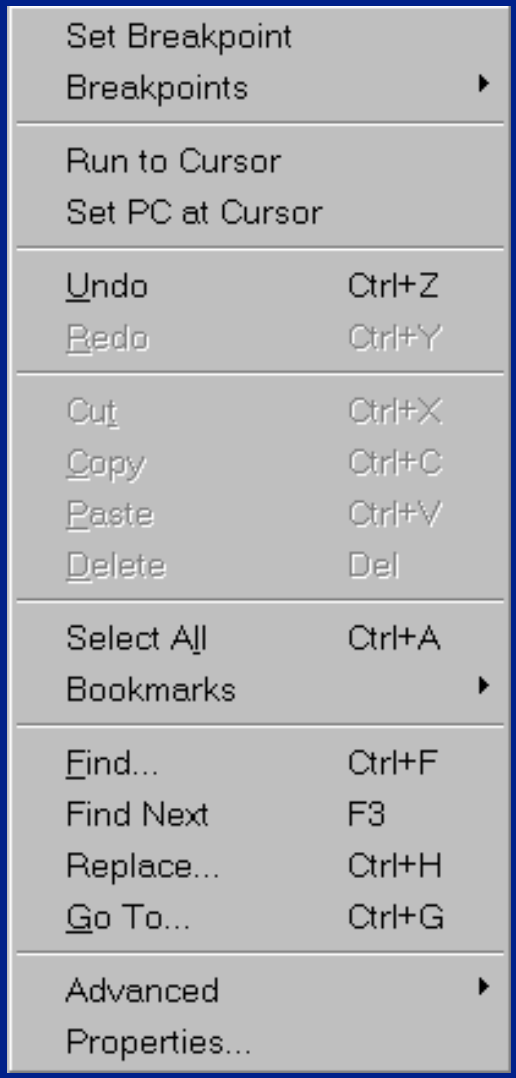
Lab 1

- Press F7 to Single Step
- Green Arrow Points to Code at _reset



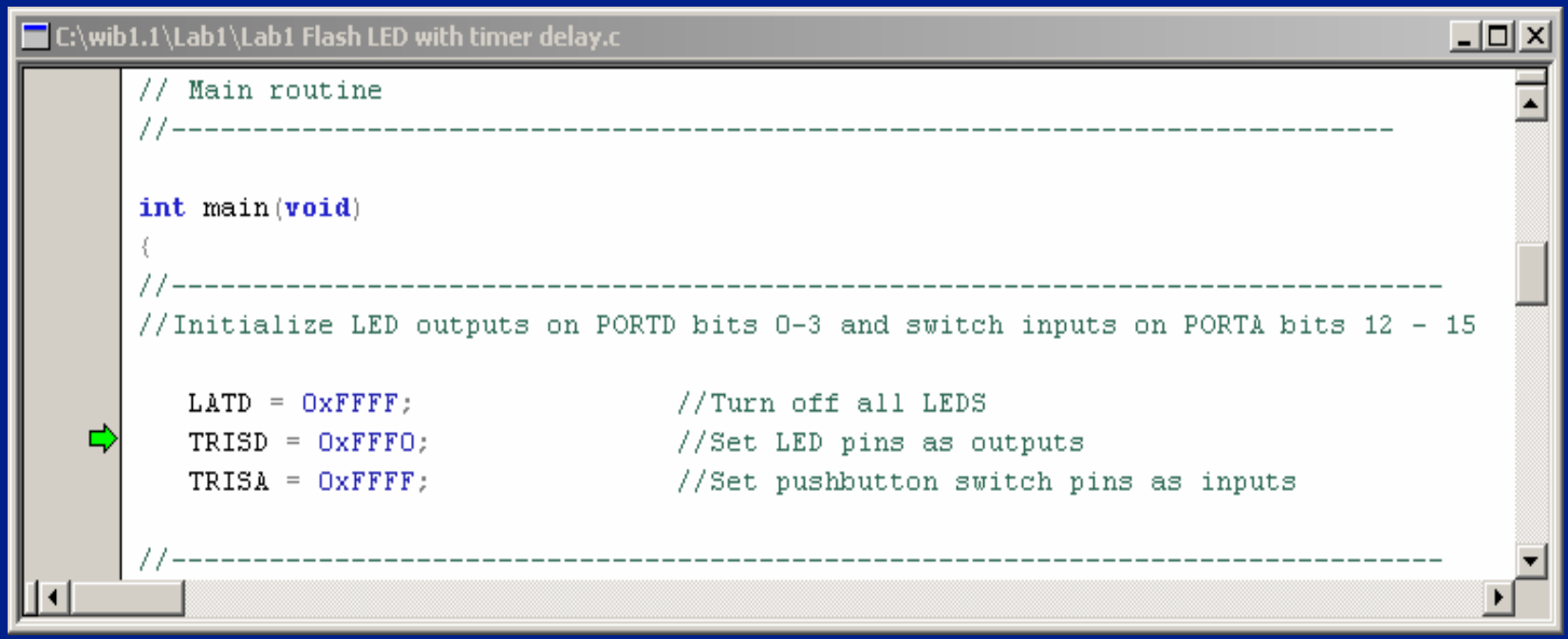
Lab 1

- MPLAB® ICD 2 Function Keys
  - F5 - Halt
  - F6 - Reset
  - F7 - Single Step
  - F9 - Run
- Right Click
  - Set Breakpoint
  - Run to cursor



Lab 1

- Right Click LATD = 0xFFFF; and Choose Run to Cursor
- Green Arrow Points to Next Line, TRISD = 0xFFF0;



- Run to Cursor Executes Line Under Cursor

Lab 1

- Find Line if(IFS0bits.T1IF == 1)
- Right Click and Choose Set Breakpoint

C:\wib1.1\Lab1\Lab1 Flash LED with timer delay.c

```
//Loop while flashing LED RD4 and testing switch inputs

   while (1)
   {
      if(IFS0bits.T1IF == 1)      //Check if Timer1 interrupt flag is set
      {
      IFS0bits.T1IF = 0;          //Clear Timer1 interrupt flag
      LATDbits.LATD0 = !LATDbits.LATD0; //Toggle LED1 RD0
      }
      if(PORTAbits.RA12 == 0)     //Check if SW1 button is pressed
      {
         LATDbits.LATD2 = 0;      //Turn LED3 RD2 on
         LATDbits.LATD3 = 1;      //Turn LED4 RD3 off
```


Lab 1

- Press F9 to Run
- Green Arrow Points to Breakpoint Line

C:\wib1.1\Lab1\Lab1 Flash LED with timer delay.c

```
//Loop while flashing LED RD4 and testing switch inputs

   while (1)
   {
      if(IFS0bits.T1IF == 1)      //Check if Timer1 interrupt flag is set
      {
      IFS0bits.T1IF = 0;          //Clear Timer1 interrupt flag
      LATDbits.LATD0 = !LATDbits.LATD0; //Toggle LED1 RD0
      }
      if(PORTAbits.RA12 == 0)     //Check if SW1 button is pressed
      {
         LATDbits.LATD2 = 0;      //Turn LED3 RD2 on
         LATDbits.LATD3 = 1;      //Turn LED4 RD3 off
```


Lab 1

- Press F7 to Single Step
- Green Arrow Points to if(PORTA bits.RA12 == 0)

C:\wib1.1\Lab1\Lab1 Flash LED with timer delay.c

```
//Loop while flashing LED RD4 and testing switch inputs

   while (1)
   {
      if(IFS0bits.T1IF == 1)      //Check if Timer1 interrupt flag is set
      {
      IFS0bits.T1IF = 0;          //Clear Timer1 interrupt flag
      LATDbits.LATD0 = !LATDbits.LATD0; //Toggle LED1 RD0
      }
      if(PORTAbits.RA12 == 0)     //Check if SW1 button is pressed
      {
         LATDbits.LATD2 = 0;      //Turn LED3 RD2 on
         LATDbits.LATD3 = 1;      //Turn LED4 RD3 off
```

Output

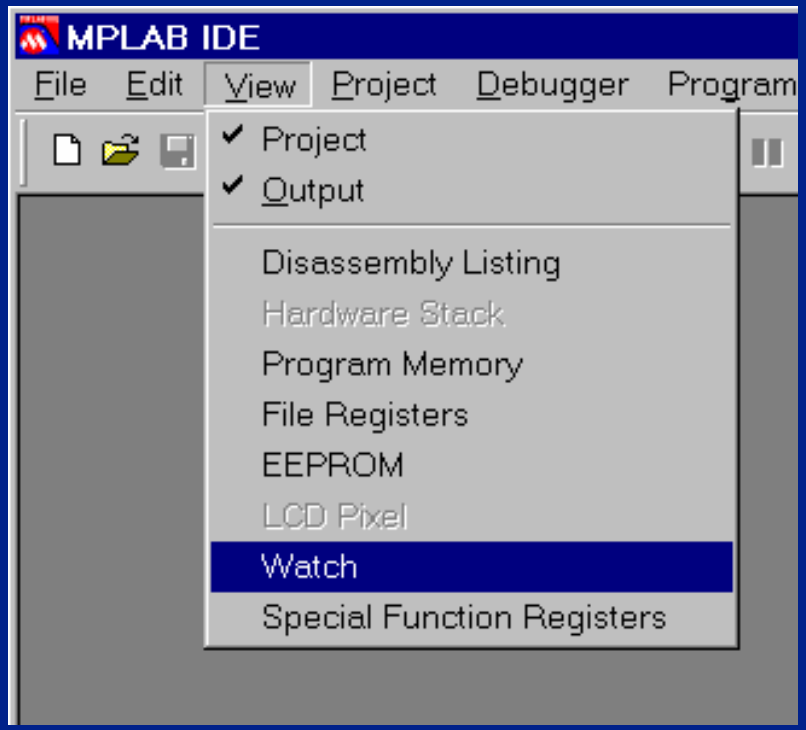
Build | Version Control | Find

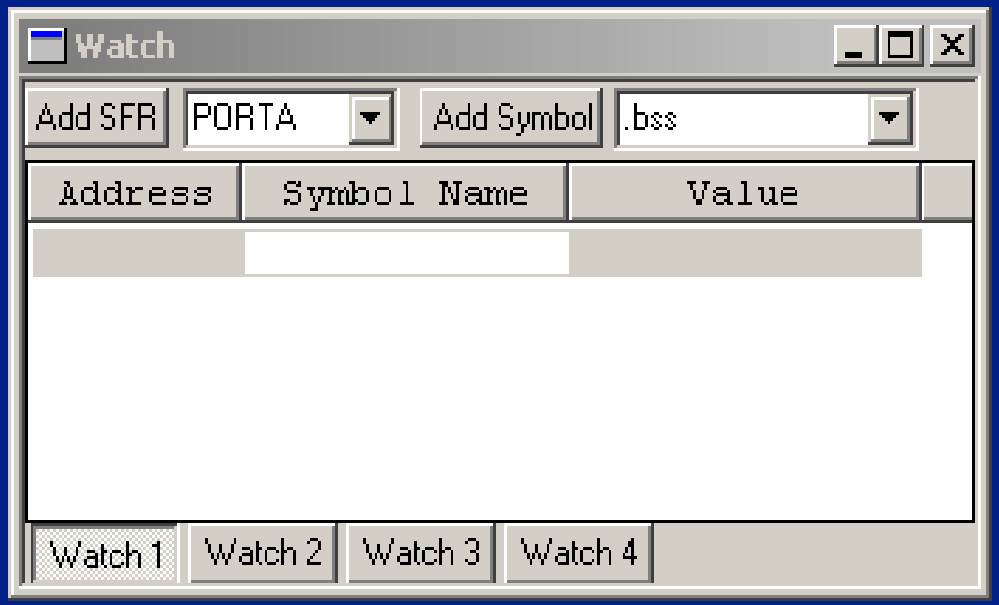
```
MPLAB ICD 2 Ready
Setting Breakpoint
Running Target
MPLAB ICD 2 Ready
Stepping Target
MPLAB ICD 2 Ready
Stepping Target
MPLAB ICD 2 Ready
Stepping Target
MPLAB ICD 2 Ready
Stepping Target
MPLAB ICD 2 Ready
Stepping Target
MPLAB ICD 2 Ready
Stepping Target
MPLAB ICD 2 Ready
```

- Note Several Steps in Output Window

Lab 1

- Click View Menu
- Choose Watch

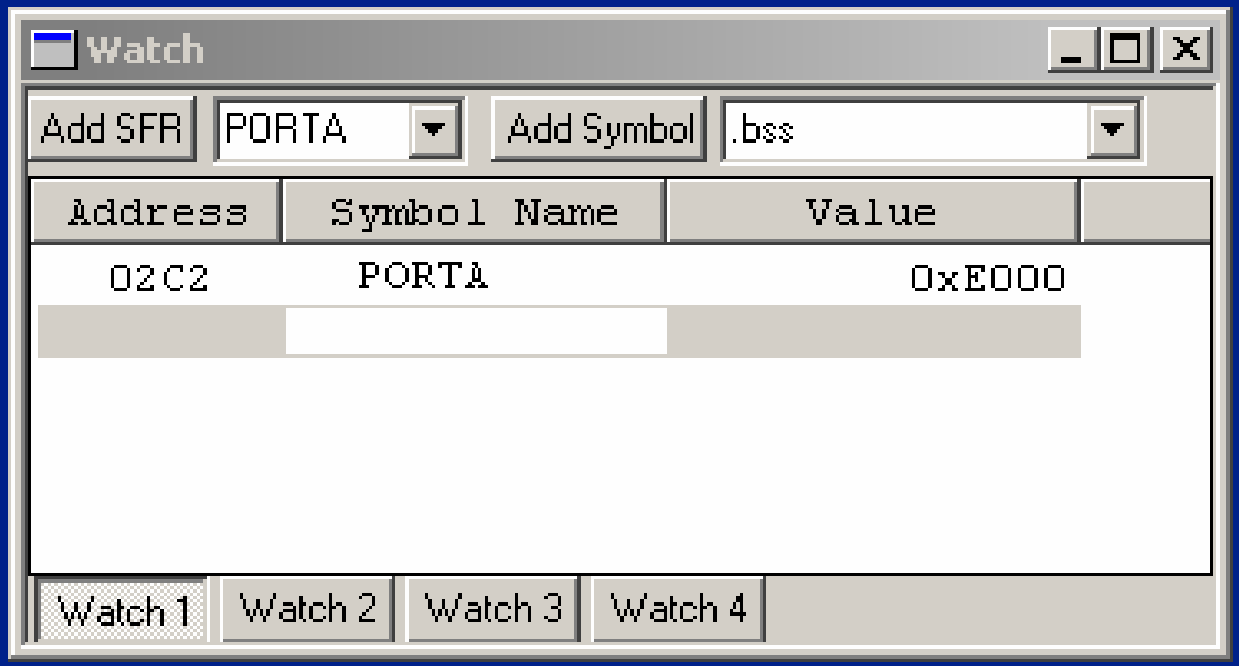




- Select PORTA from the Add SFR Field
- Click Add SFR

Lab 1

- Hold Down Switch SW1 or SW2 and Press F9
- PORTA Value Changes Depending on SW1 and SW2 Being Pressed

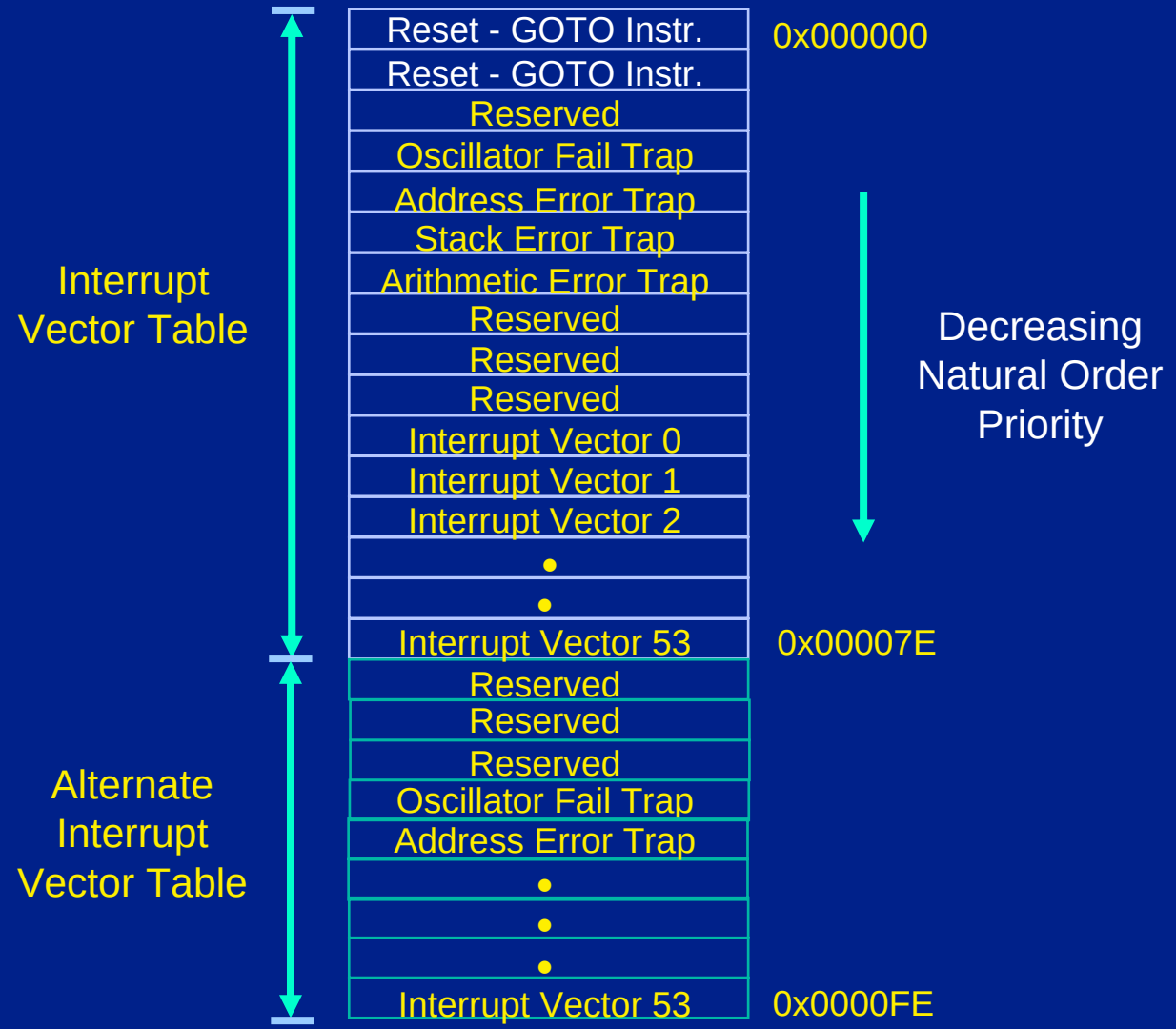


dsPIC® Device Interrupt Handling

Interrupt Controller

- Interrupts Overview
  - Interrupt Vector Table (IVT)
    - Vector location contains ISR address
  - Unique vector for each interrupt source
  - 8 non-maskable error trap vectors
  - 54 interrupt vectors
  - 7 user assigned priority levels for each source
  - Consistent 5-cycle latency for all instructions
  - 3-cycle return from interrupt

Interrupt Vector Table



Interrupt Controller

- Interrupt Priority Levels
  - CPU has 16 priority levels
  - Level 0 - 7 for normal peripheral interrupts
    - Can be assigned by user
  - Level 8 - 15 reserved non-maskable traps
    - Oscillator failure, stack error, etc.
    - Fixed priority cannot be disabled

Interrupt Controller

- CPU has some priority level at all times
  - Starts at zero after Reset
  - IPL<2:0> in Status Register - SR
  - IPL3 in Core Control Register - CORCON
- Higher priority interrupt can interrupt CPU
  - CPU then assumes new priority
  - Saves old priority on the stack

Interrupt Controller

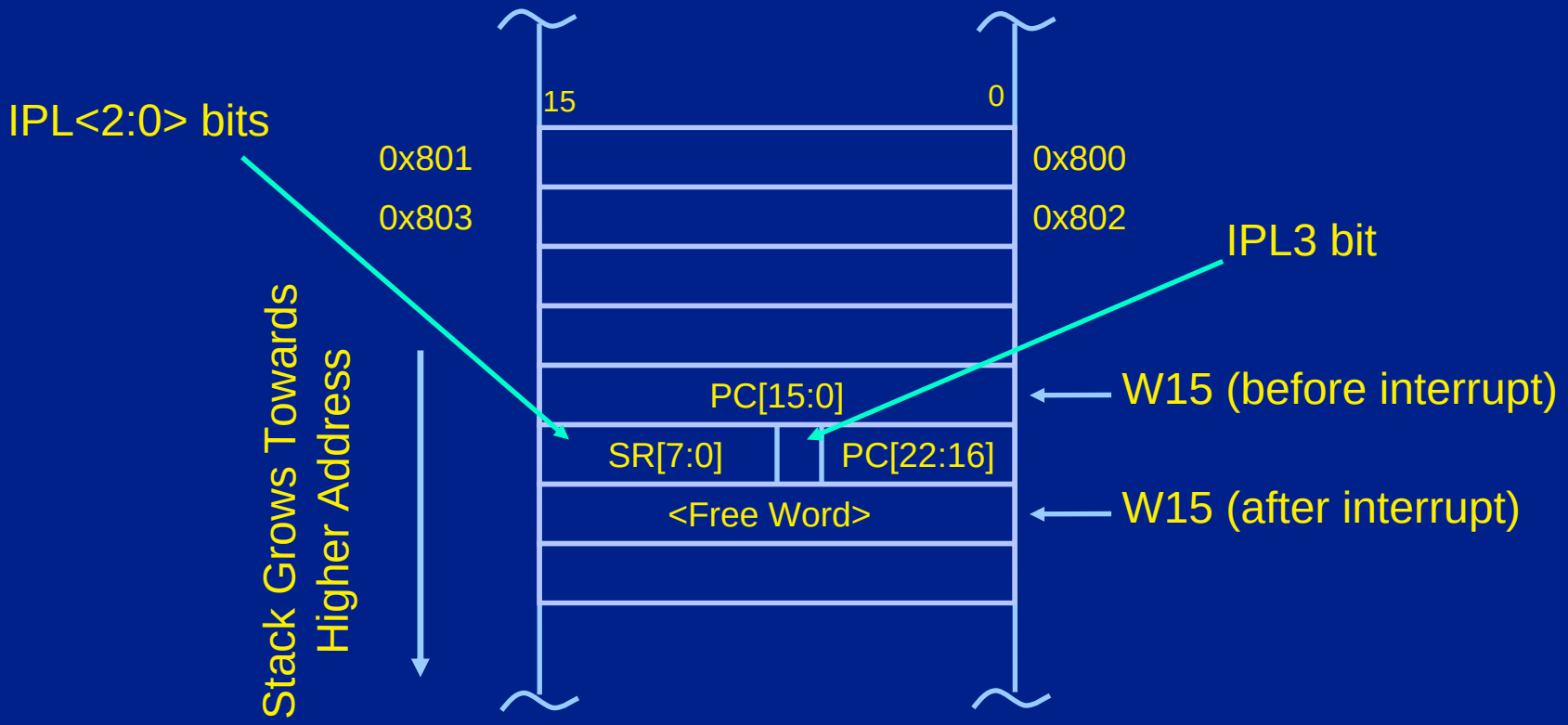
- Interrupt Priority Level set by IPCx registers
  - Interrupt sources can be levels 0 - 7
  - A level 0 source is effectively disabled
  - Default interrupt priority is 4
- Can change CPU priority by writing IPL<2:0>
  - IPL3 is read only - can't disable traps
  - IPL<2:0> = 111 disables all other interrupts
- DISI instruction disables level 1 - 6 interrupts

Interrupt Controller

- Interrupt Context Saving
  - Only SRL and PC are saved by hardware
    - Saved on stack
  - Compiler saves additional context data
  - User can specify other data to save

Interrupt Controller

Stack Operation



Interrupt Controller

- C30 Interrupt Attributes
  - __attribute__ allows interrupt attributes list

```
void __attribute__((interrupt)) _INT1Interrupt(void)

...((interrupt(save(Var1,Var2)))) _INT1Interrupt(void)
...((interrupt(save(Var1),irq(15)))) MyIRQ(void)
```

  - _ISR is defined in header files

```
void _ISR _INT1Interrupt(void)
```


Interrupt Controller

- Interrupt Nesting
  - Interrupts nested by default
  - Disable nesting by setting NSTDIS bit INTCON1<15>
  - CPU priority forced to level 7 during all interrupts when NSTDIS = 1

Lab 2

Using Interrupts

Lab 2

- Light LED3 and LED4 using INT1 and INT2 Interrupt
  - Learn about interrupts
- Use Lab 2 Workspace Provided
  - Add six lines as indicated by comments

Lab 2

- INT1 and INT2 Interrupt Registers
  - IFS1 - Interrupt Flag Status
    - INT1IF bit and INT2IF bit
  - IEC1 - Interrupt Enable Control
    - INT1IE bit and INT2IE bit
- Interrupt Vector Table
  - Use labels defined in Linker Script file
  - Listed in C30 Compiler User's Guide

Lab 2

- Linker Script file - Note label __INT1Interrupt

C:\PIC30_Tools\support\gld\p30f6012.gld

```
** Primary Interrupt Vector Table
*/
.ivt __IVT_BASE :
  {
    LONG(DEFINED(__ReservedTrap0)  ? ABSOLUTE(__ReservedTrap0)  :
         ABSOLUTE(__DefaultInterrupt));
    LONG(DEFINED(__OscillatorFail) ? ABSOLUTE(__OscillatorFail) :
         ABSOLUTE(__DefaultInterrupt));
```

```
    LONG(DEFINED(__MI2CInterrupt)  ? ABSOLUTE(__MI2CInterrupt)  :
         ABSOLUTE(__DefaultInterrupt));
    LONG(DEFINED(__CNInterrupt)    ? ABSOLUTE(__CNInterrupt)    :
         ABSOLUTE(__DefaultInterrupt));
    LONG(DEFINED(__INT1Interrupt)  ? ABSOLUTE(__INT1Interrupt)  :
         ABSOLUTE(__DefaultInterrupt));
```


Lab 2

- Operation of Lab 2
  - LED1 flashes in background in main()
  - Turn on/off LED3 when switch SW1 is pressed
  - Turn on/off LED4 when switch SW2 is pressed
- Tasks for Lab 2
  - Enable INTx interrupts
  - Create a INTx interrupt function
  - Clear interrupt flag to allow a new interrupt

Lab 2

- C:\WIB1.1\Lab2\Lab 2.mcw Workspace
- Code to be Added is Clearly Marked!

C:\wib1.1\Lab2\Lab2 Light LED with INTx interrupt.c

```
//-----------------------------------------------------------------------
// INT1 Interrupt Service Routine
//-----------------------------------------------------------------------

'Replace text with code here //Declare INT1 interrupt function
{
   'Replace text with code here      //Clear INT1 interrupt flag
   LATDbits.LATD2 = 0;      //Turn LED3 on
   LATDbits.LATD3 = 1;      //Turn LED4 off
}

//-----------------------------------------------------------------------
// INT2 Interrupt Service Routine
//-----------------------------------------------------------------------

'Replace text with code here //Declare INT2 interrupt function
{
   'Replace text with code here      //Clear INT2 interrupt flag
   LATDbits.LATD2 = 1;      //Turn LED3 off
   LATDbits.LATD3 = 0;      //Turn LED4 on
}
```


Lab 2

- Solution:

Line 46: IEC1bits.INT1IE = 1;

Line 47: IEC1bits.INT2IE = 1;

Line 67: void _ISR _INT1Interrupt(void)

Line 78: void _ISR _INT2Interrupt(void)

Line 69: IFS1bits.INT1IF = 0;

Line 80: IFS1bits.INT2IF = 0;

Lab 3

Using Interrupt Priorities

Lab 3

- Set Timer Interrupts to Different Priorities
- Change CPU Priority in Code
  - Learn about interrupt priorities and interaction
- Use Lab 3 Workspace Provided
  - Add five lines as indicated by comments

Lab 3

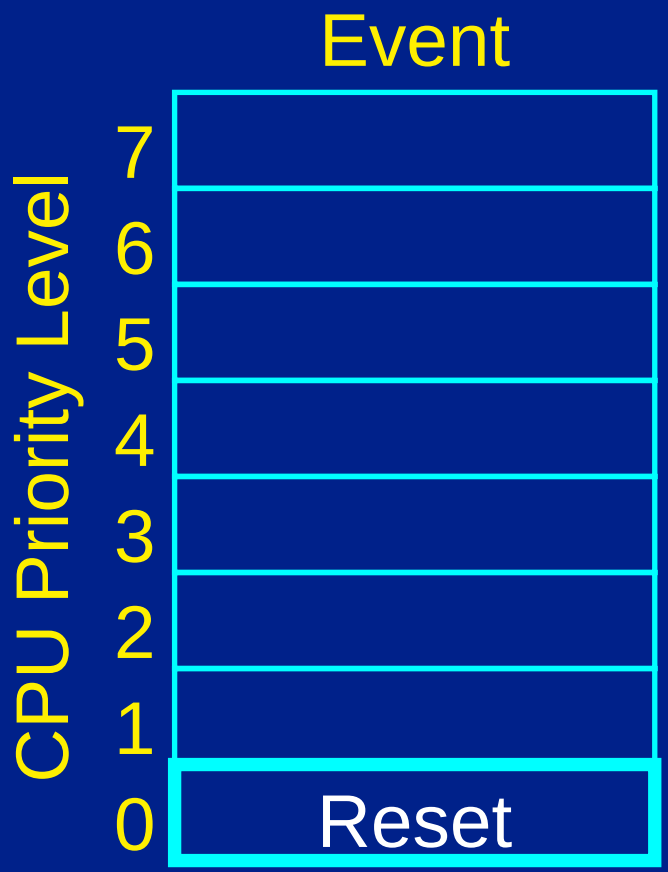
- Operation of Lab 3
  - Set CPU priority to level 4 (in code)
  - Timer1 level 3 interrupts every 1/10 second
    - Light LED1 for 1/10 second in ISR
  - Timer2 level 5 interrupts every one second
    - Light LED2 for one second in ISR
  - Notice that Timer1 interrupt to light LED1 never happens
    - Timer1 level 3 is less than CPU level 4

Lab 3

- Operation of Lab 3 Continued...
  - Detect when SW1 is pressed in main loop
    - Set CPU priority to level 6 and LED4 on
  - Detect when SW1 is released in main loop
    - Set CPU priority to level 2 and LED4 off
  - Notice that no timer interrupts occur while switch is pressed
  - Notice that both timer interrupts occur after switch is released

Lab 3

A Picture is Worth a Thousand Words!

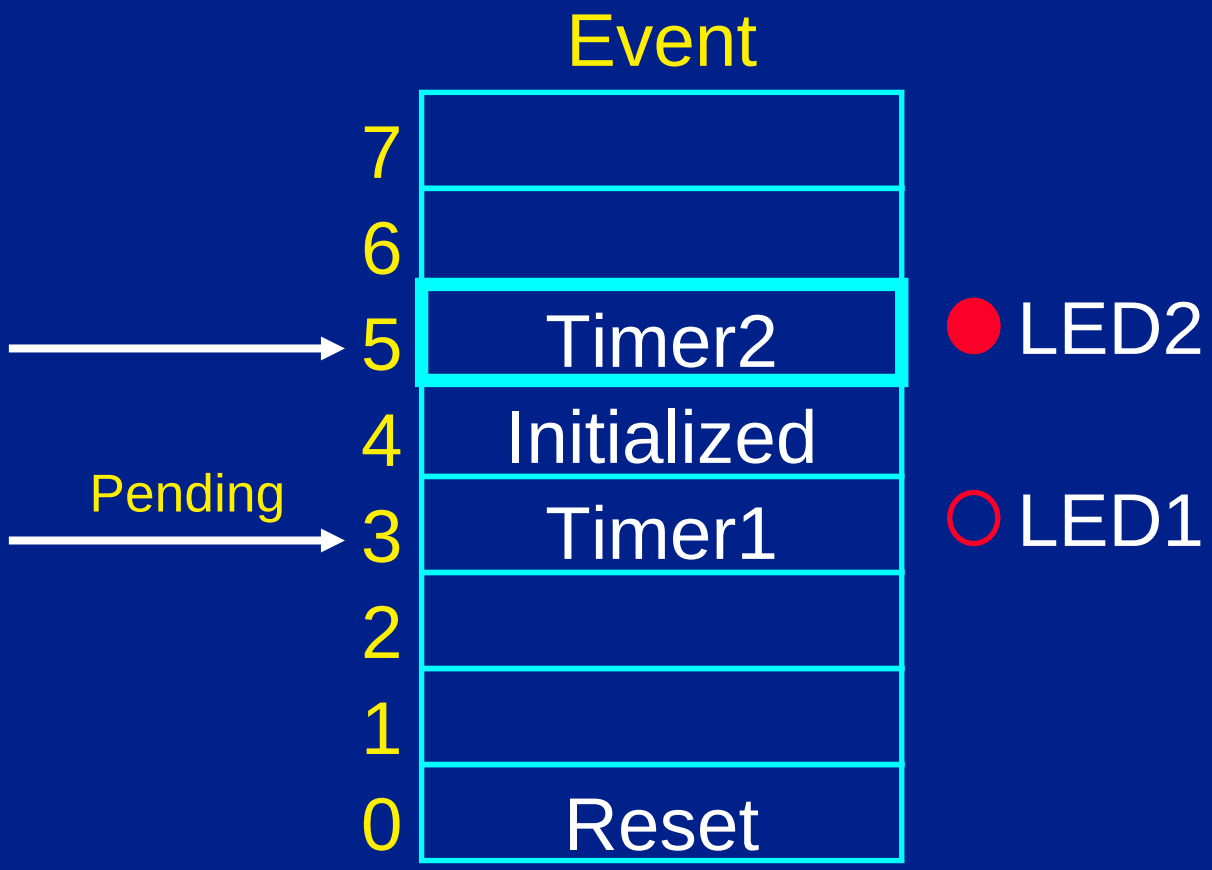


Lab 3

Initialize CPU, Timer1 and Timer2 Priority Level

Lab 3

Timer2 Interrupt Occurs

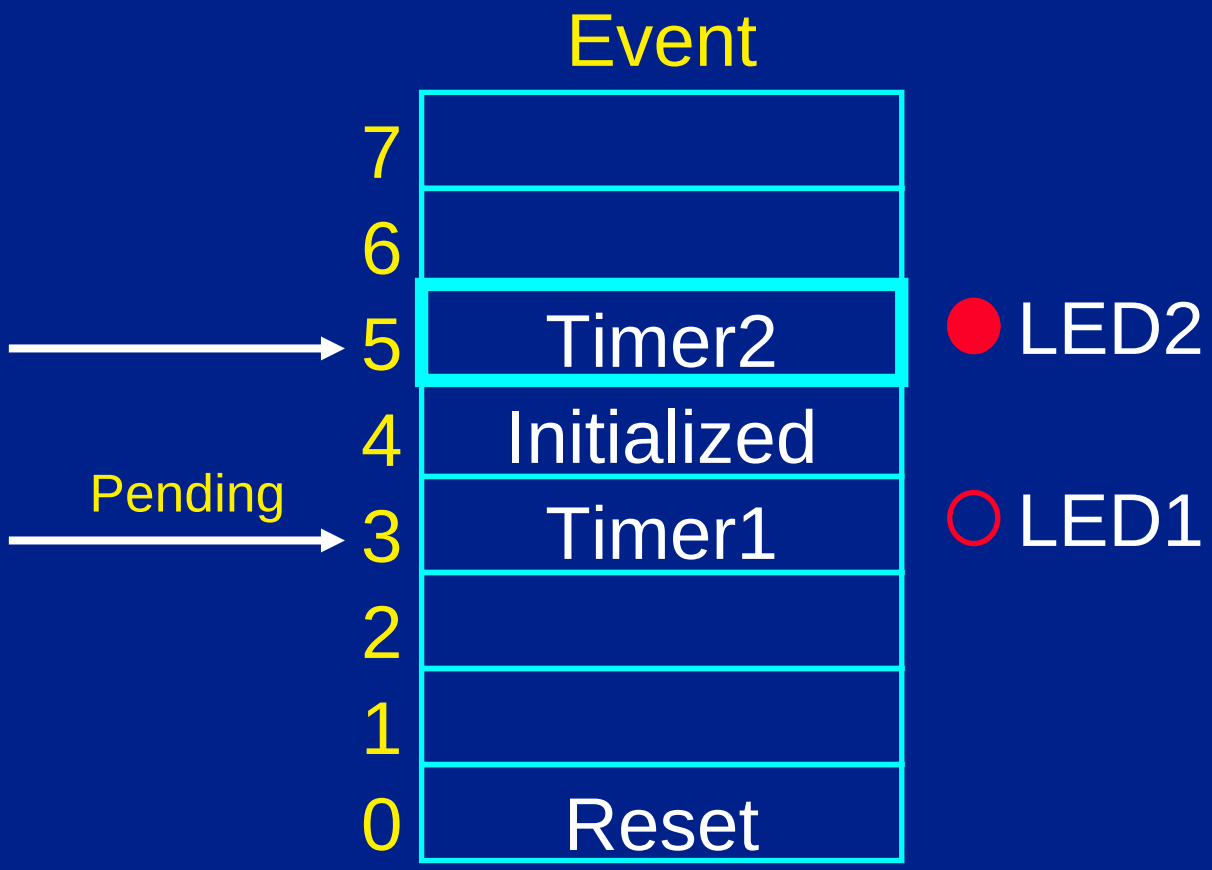


Lab 3

Timer2 Interrupt Returns

Lab 3

Timer2 Interrupt Occurs

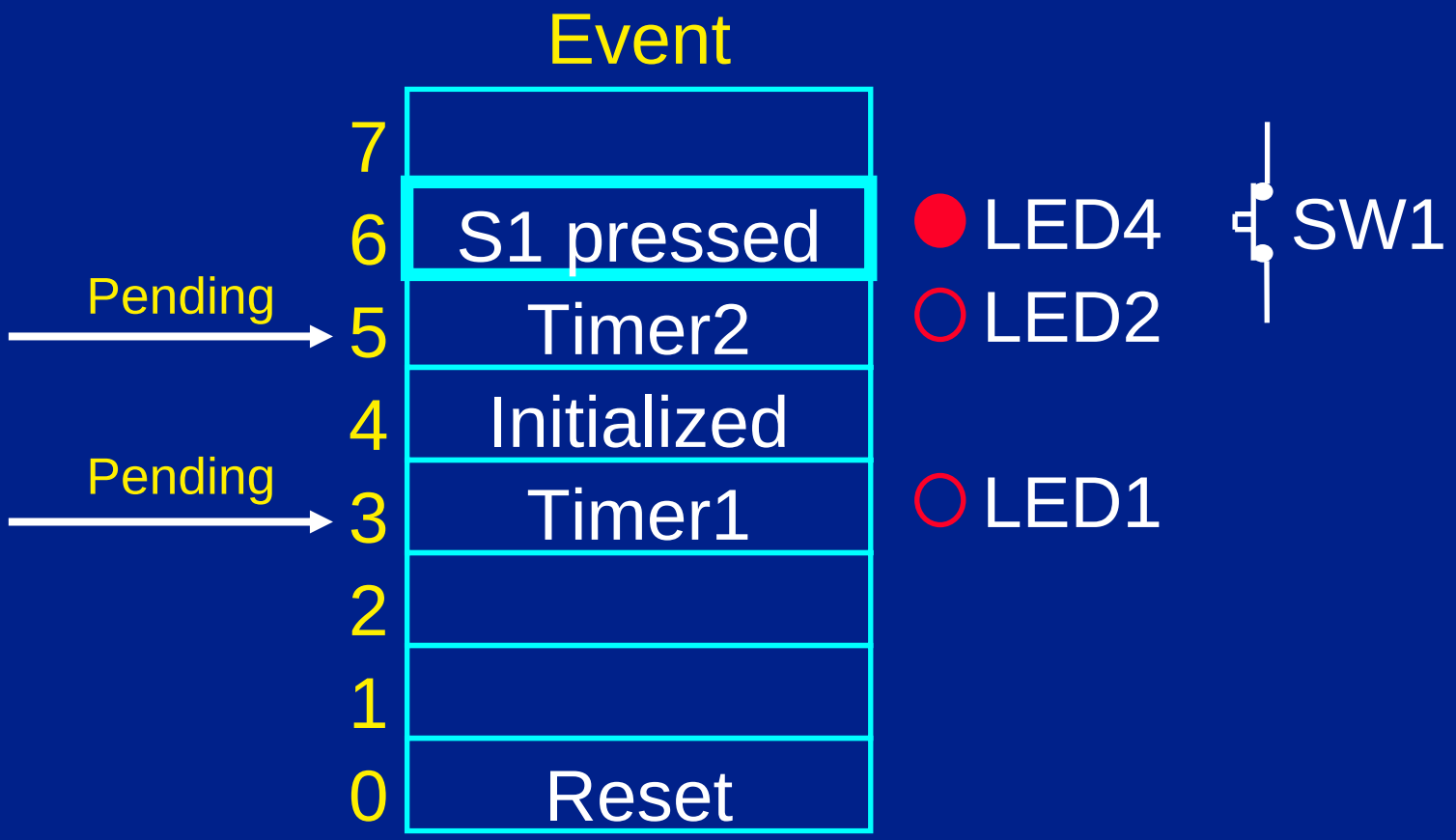


Lab 3

Timer2 Interrupt Returns

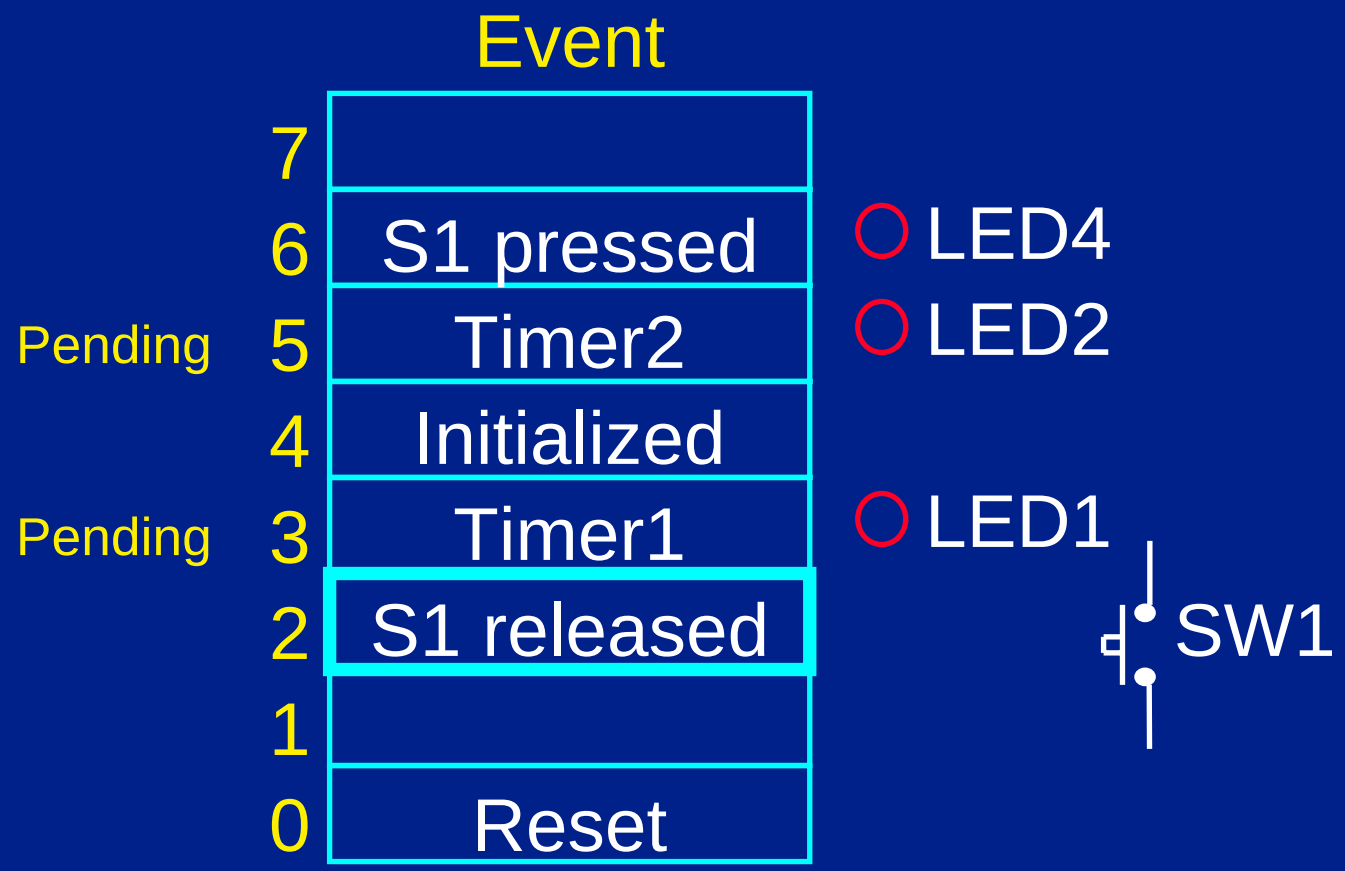
Lab 3

Code Changes CPU Priority When S1 Pressed



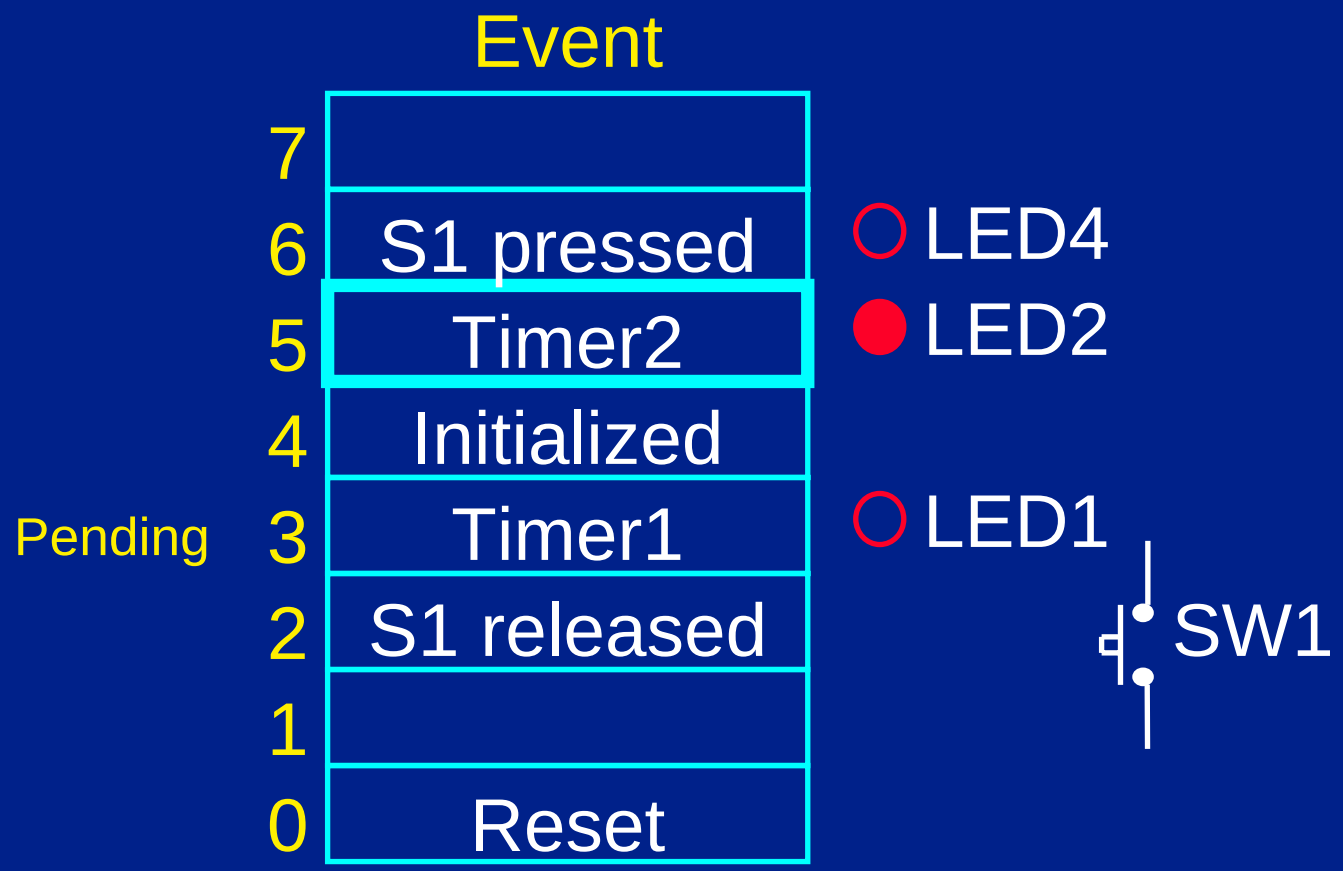
Lab 3

Code Changes CPU Priority When S1 Released



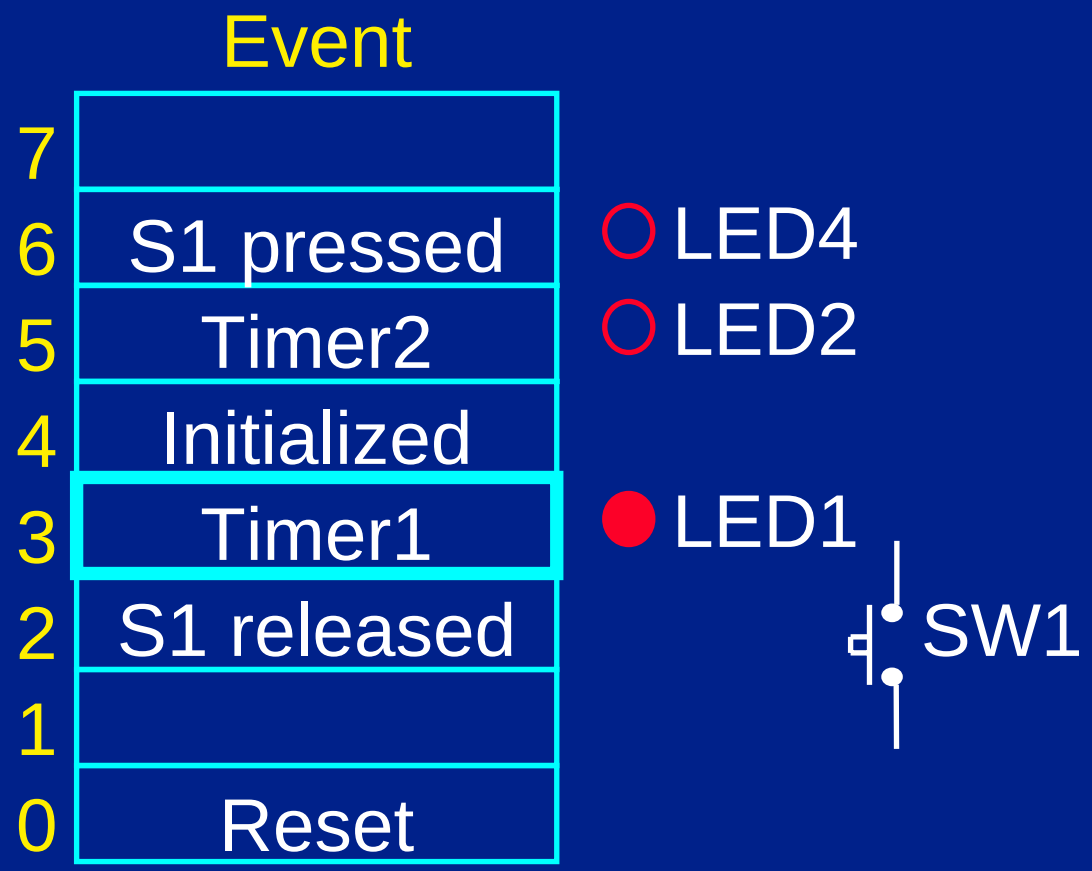
Lab 3

Pending Timer2 Interrupt is Handled



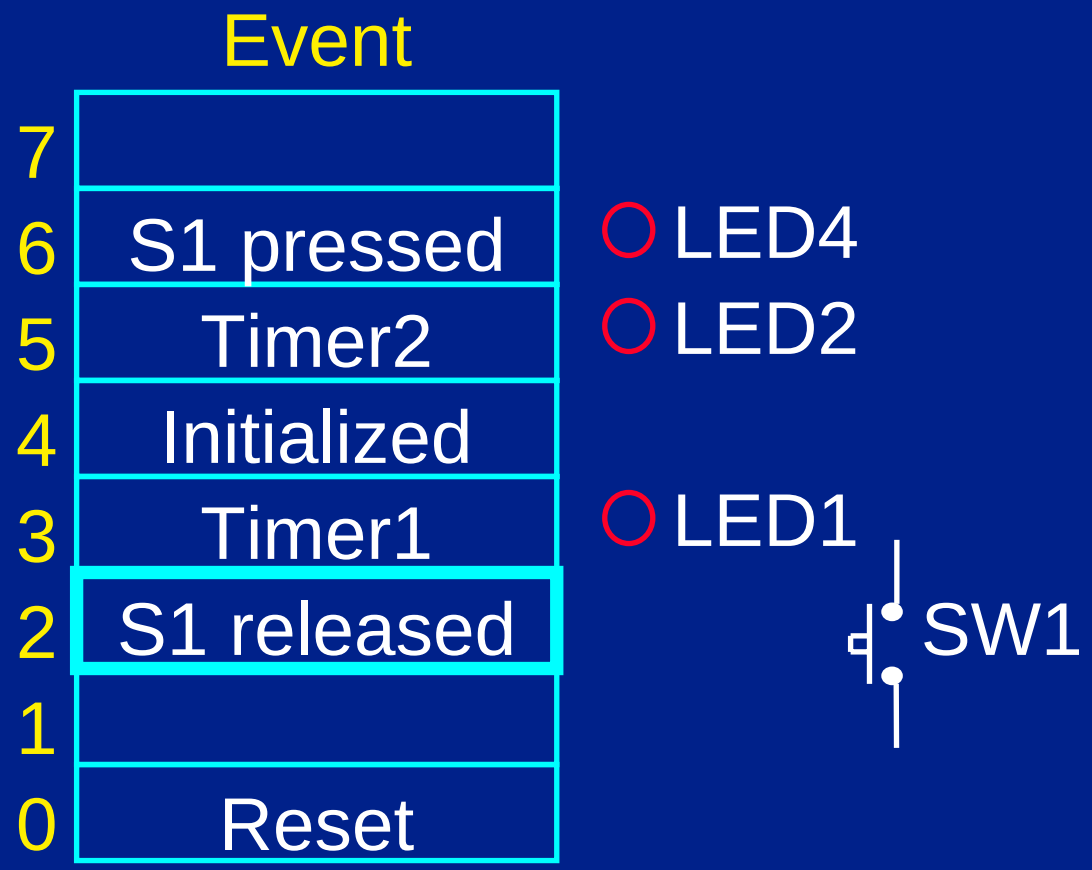
Lab 3

Pending Timer1 Interrupt is Handled



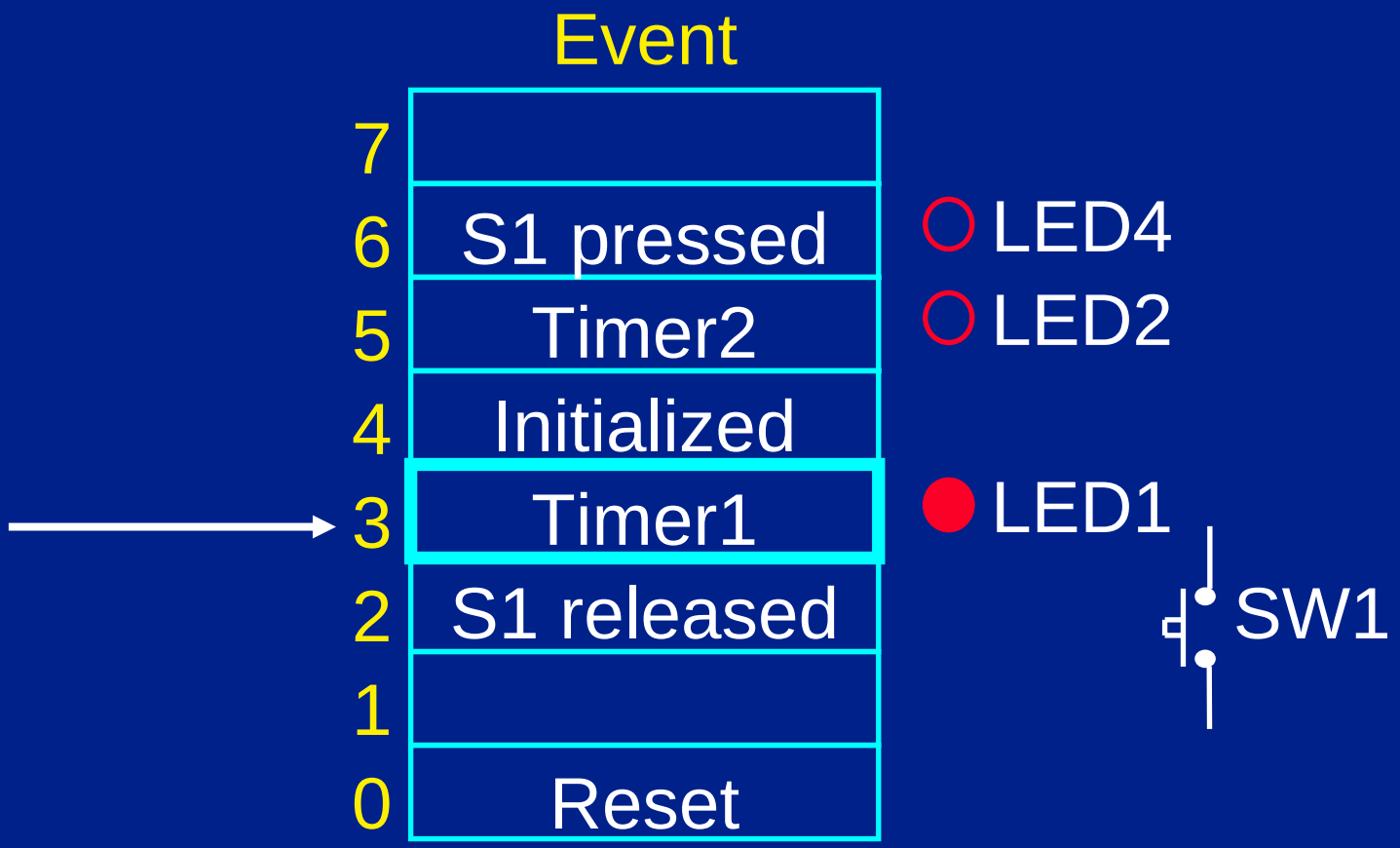
Lab 3

Return from Timer1 Interrupt



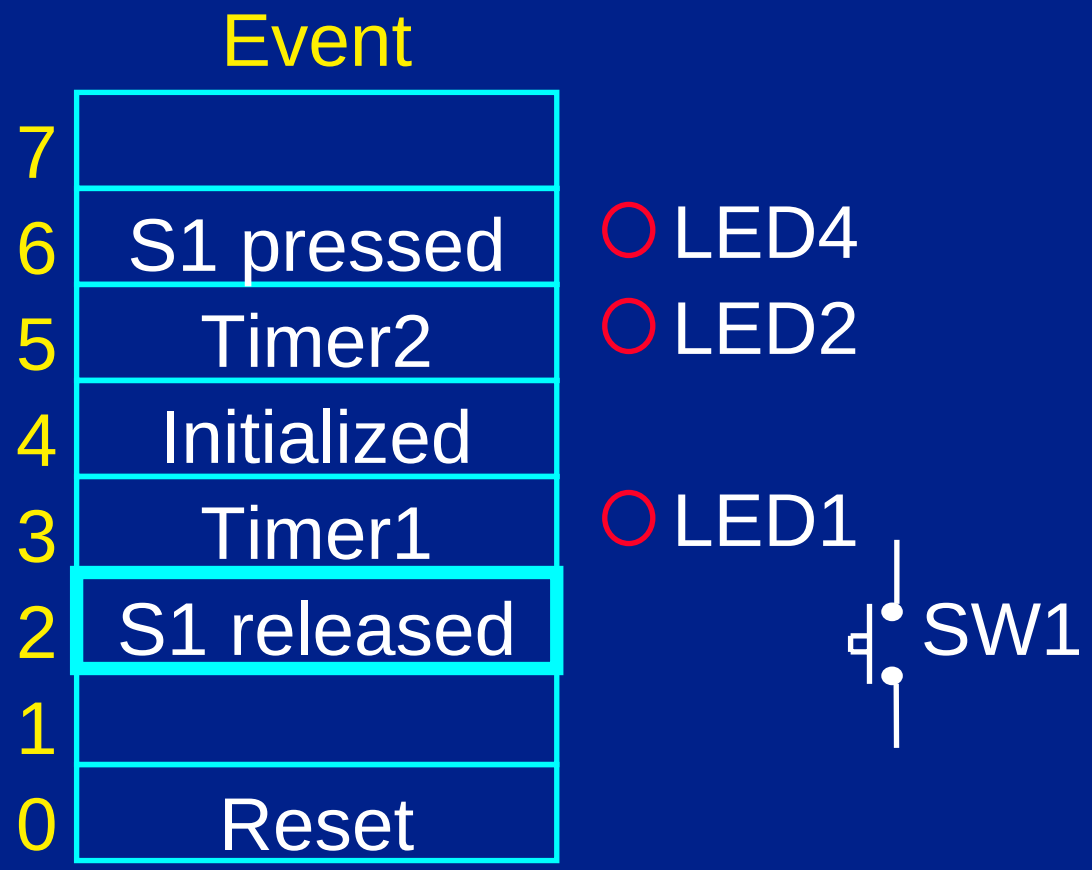
Lab 3

Timer1 Interrupt Occurs



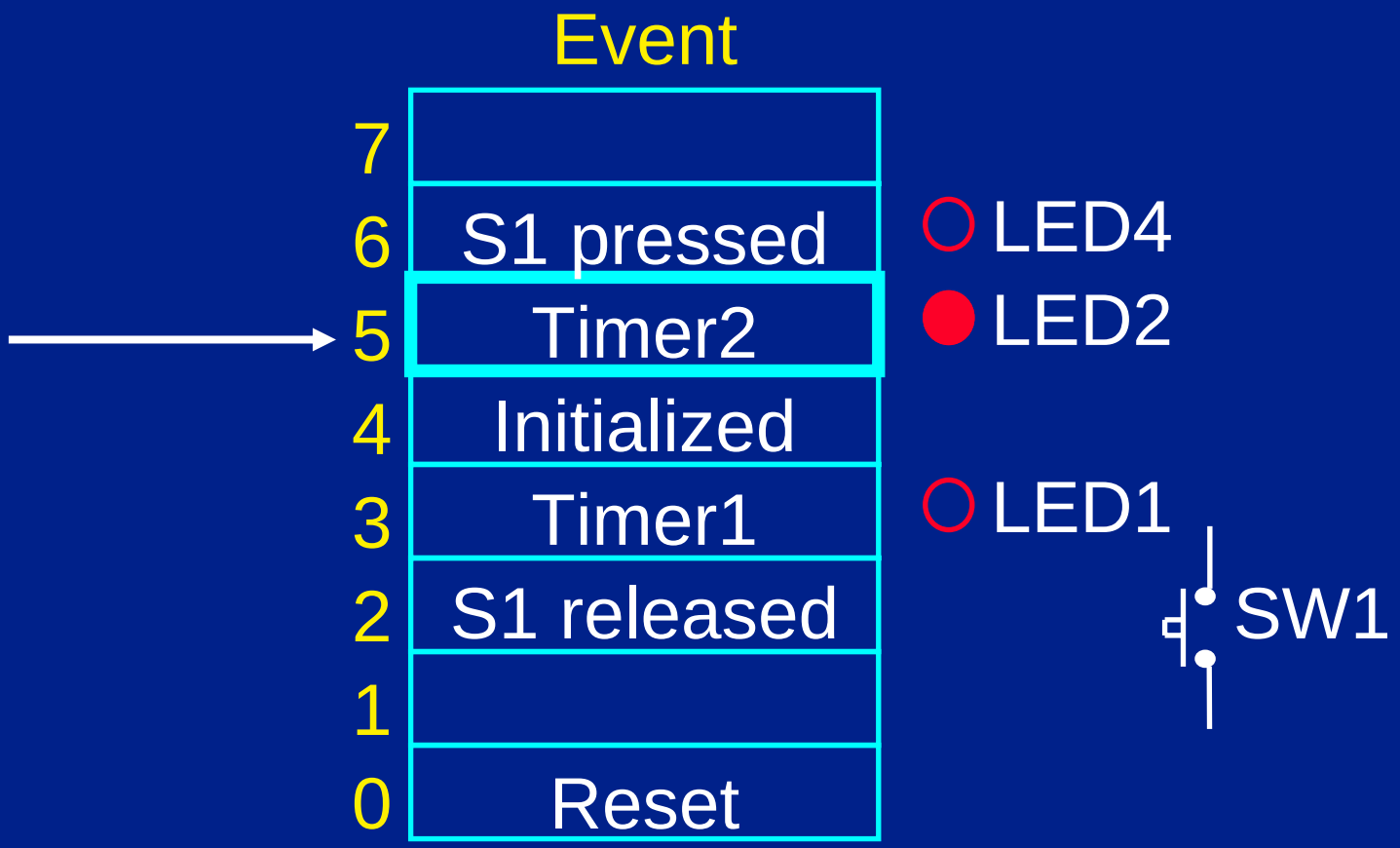
Lab 3

Return from Timer1 Interrupt



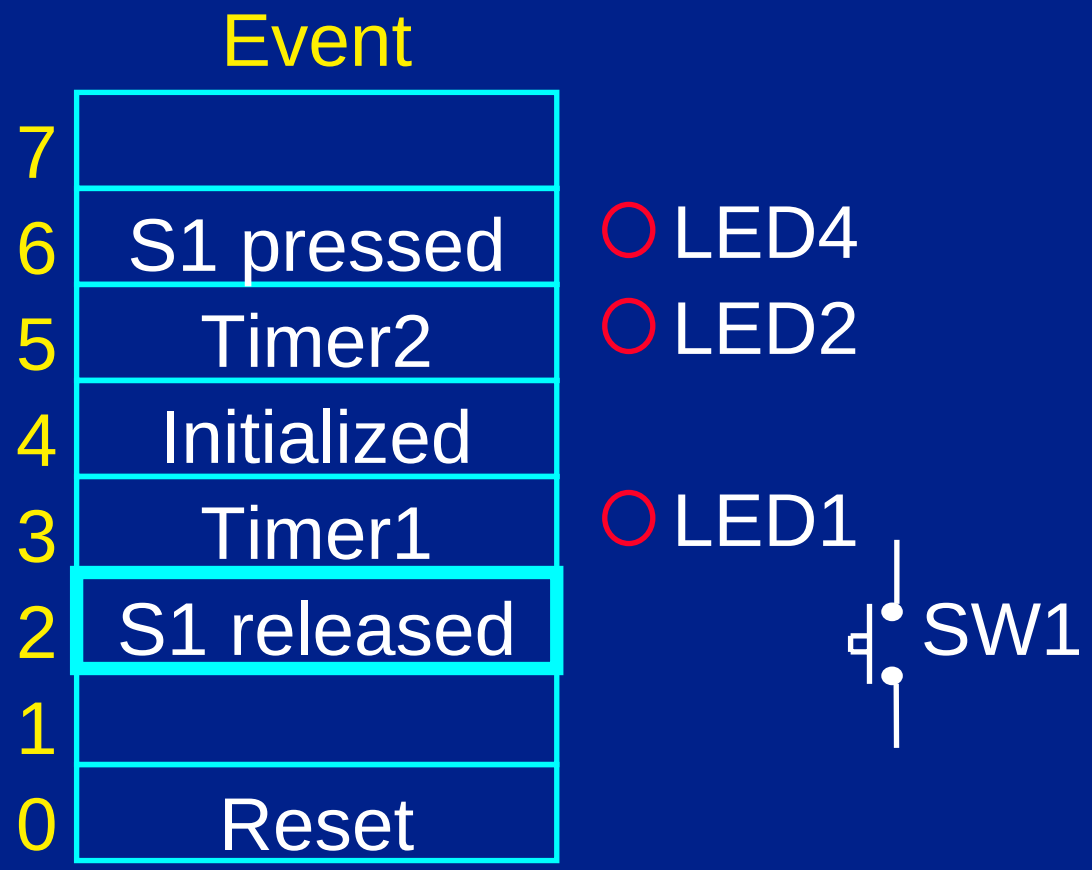
Lab 3

Timer2 Interrupt Occurs



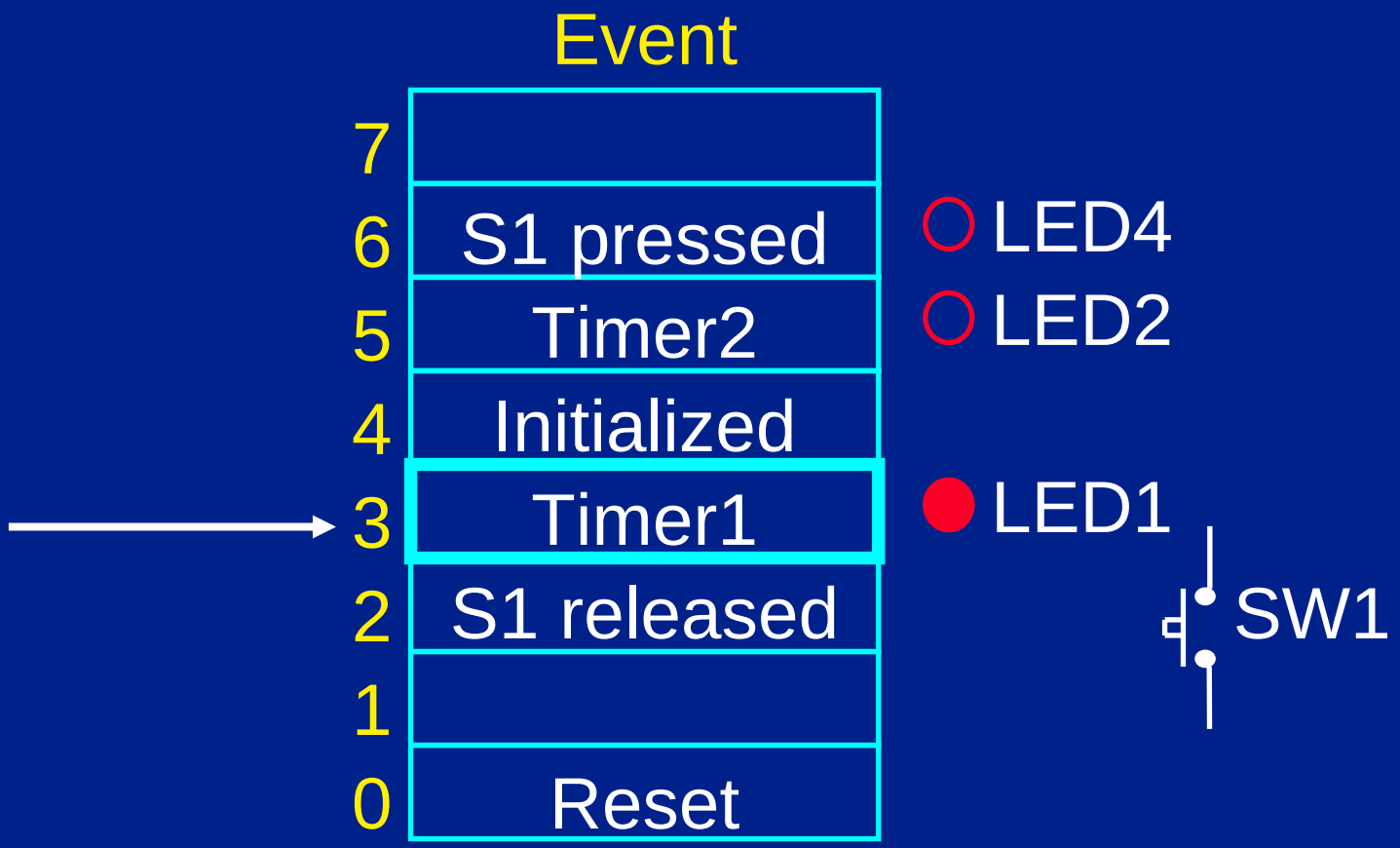
Lab 3

Return from Timer2 Interrupt



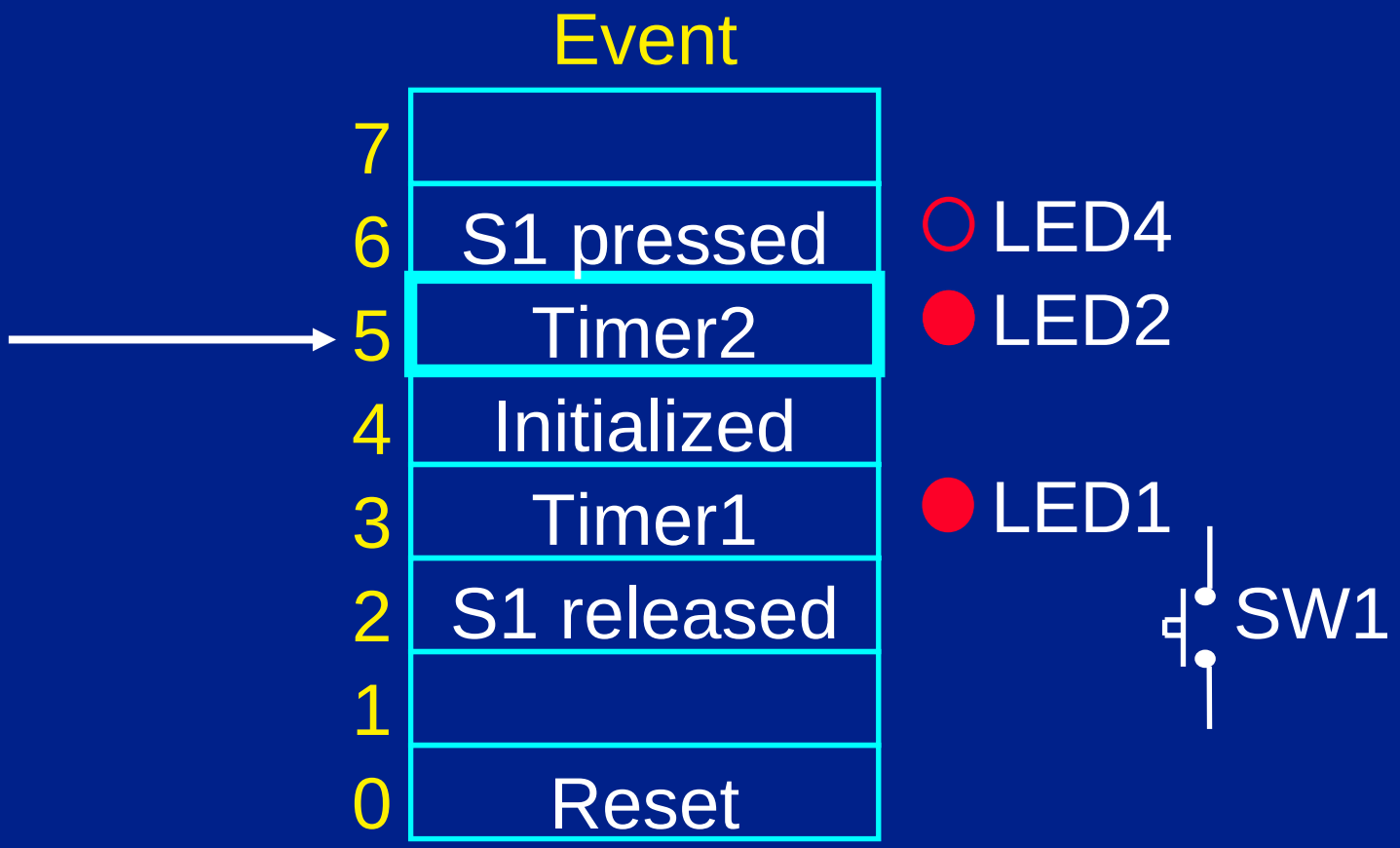
Lab 3

Timer1 Interrupt Occurs



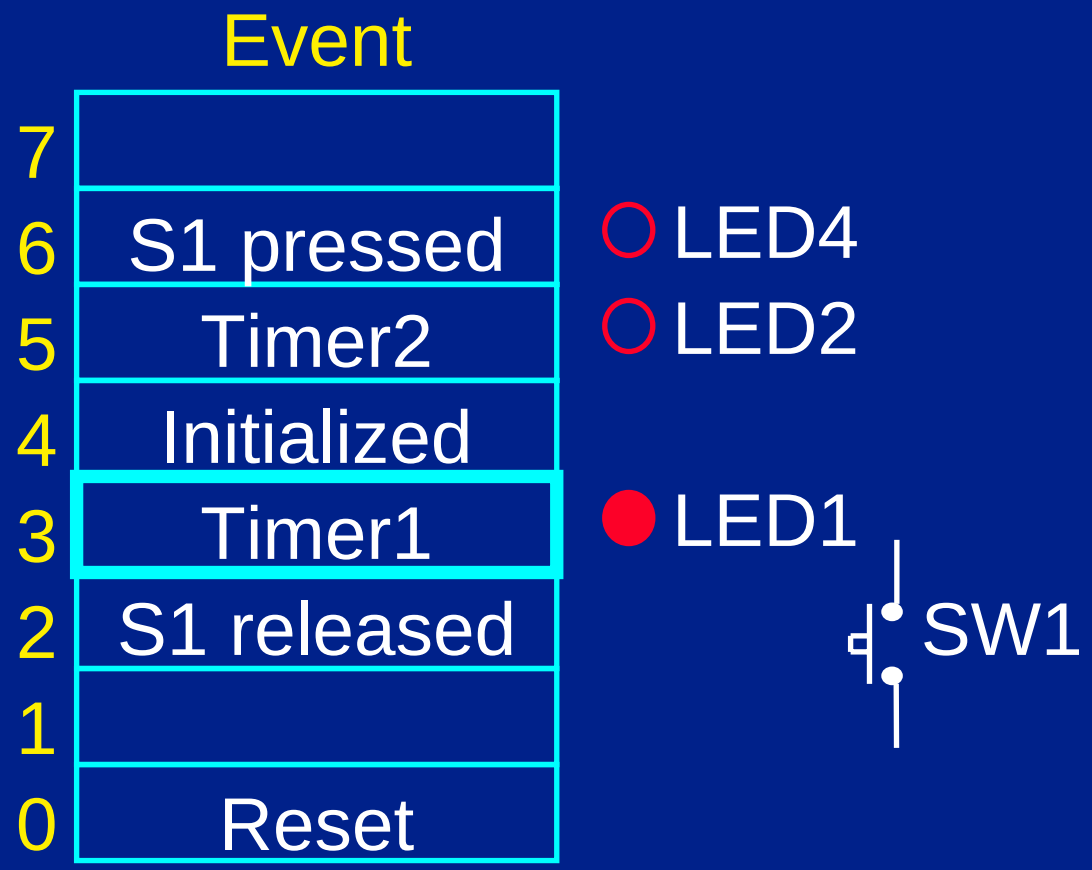
Lab 3

Timer2 Interrupt Occurs



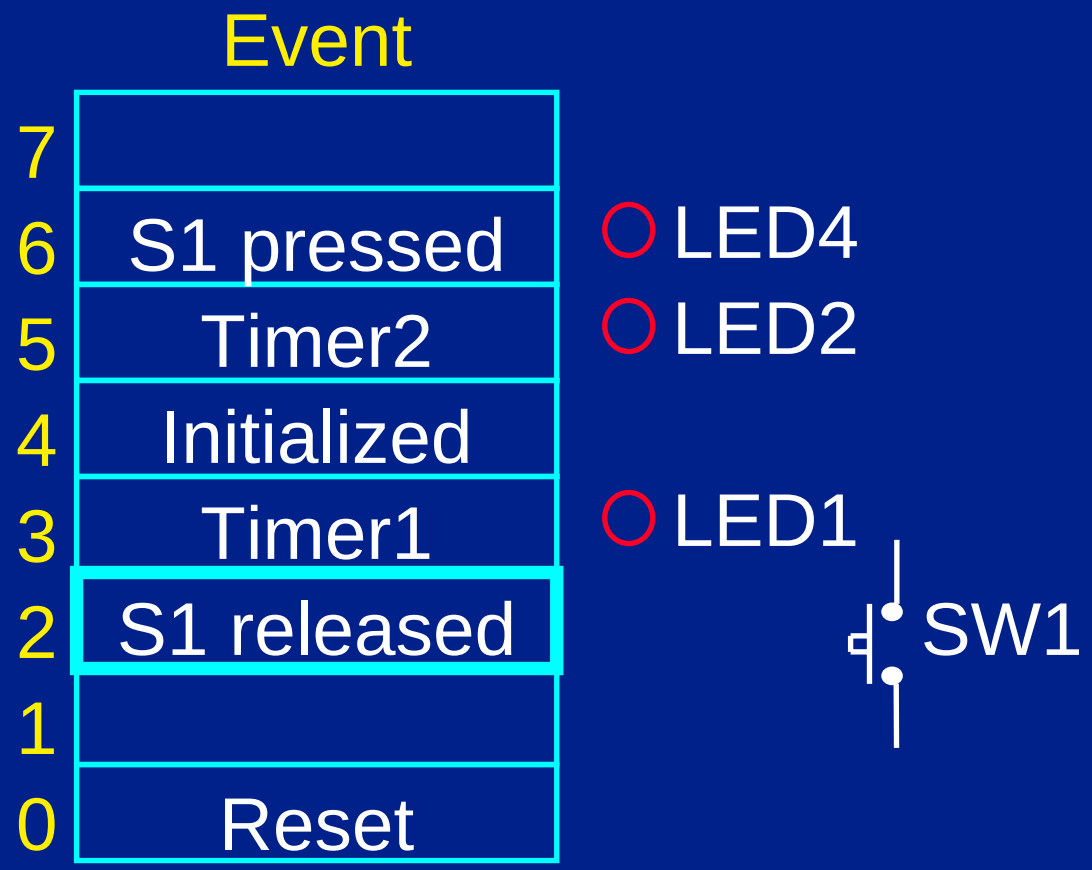
Lab 3

Return from Timer2 Interrupt



Lab 3

Return from Timer1 Interrupt



One click too many!

Lab 3

- Tasks for Lab 3
  - Set CPU priority to level 4
  - Set Timer1 interrupt priority to level 3
  - Set Timer2 interrupt priority to level 5
  - Wait until switch SW1 pressed
  - Wait in Timer1 ISR until next Timer1 interrupt

Lab 3

- C:\WIB1.1\Lab3\Lab 3.mcw Workspace
- Code to be Added is Clearly Marked!

C:\WIB\Lab3\Lab3 Flash LEDs with two timer interrupts.c

```
//------------------------------------------------------------------
//Set Timer 2 interrupt priority and enable interrupt

   "Replace this text with code" //Set Timer2 interrupt priority
   IFS0bits.T2IF = 0;            //Clear Timer2 interrupt flag
   IEC0bits.T2IE = 1;            //Enable Timer2 interrupt

//------------------------------------------------------------------
//Loop while testing switch S1 (active low) and change CPU prior

   "Replace this text with code" //Wait for S1 to be pressed

   while (1)
   {
      if(PORTCbits.RC13)         //If S1 is not pressed
      {
         LATDbits.LATD7 = 0;     //Turn LED RD7 off
```


Lab 3

- A Solution

Line 38: `SRbits.IPL = 4;`

Line 51: `IPC0bits.T1IP = 3;`

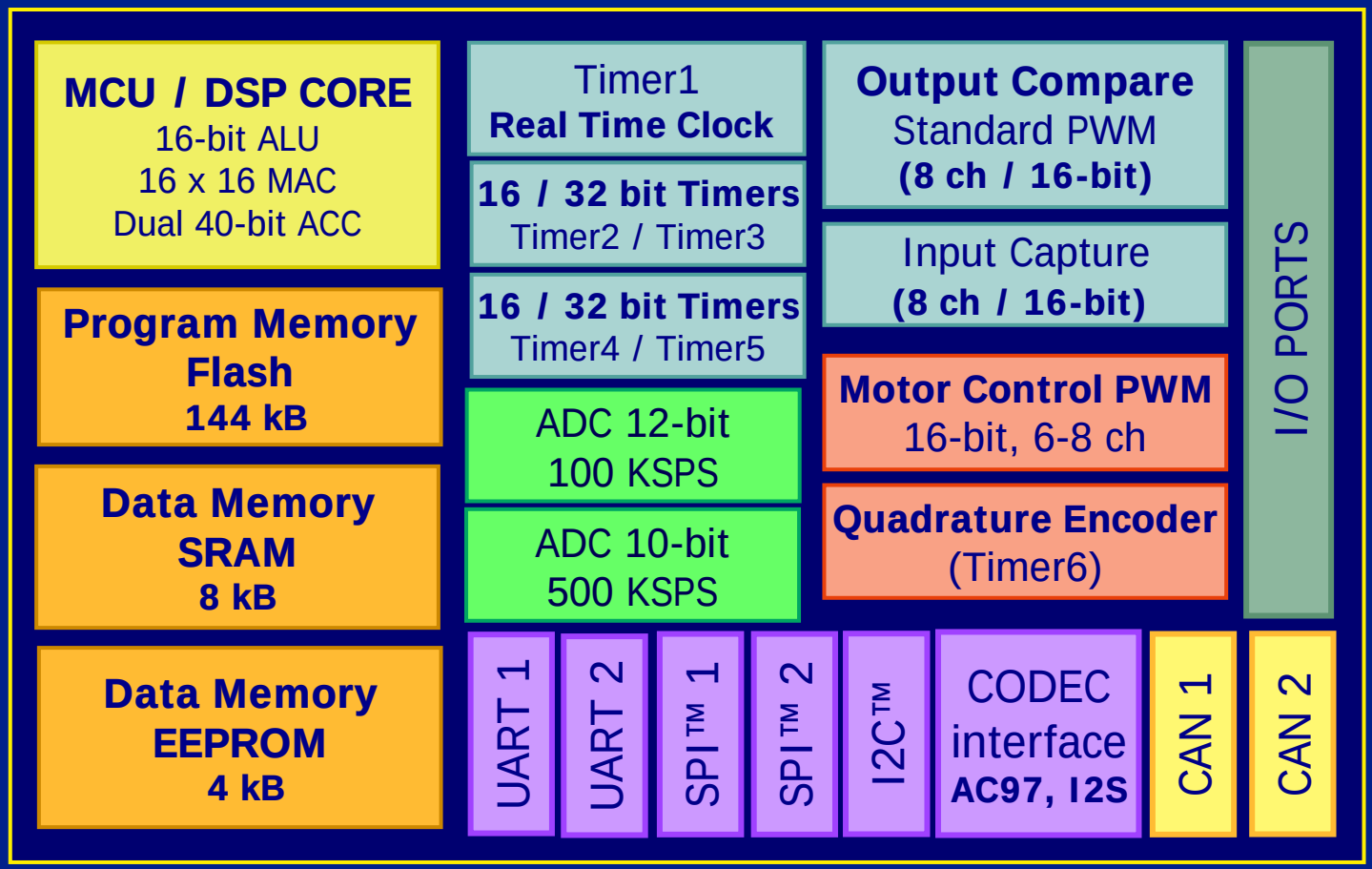
Line 66: `IPC1bits.T2IP = 5;`

Line 73: `while(!SW1);`

Line 98: `while(IFS0bits.T1IF == 0);`

Lab 4:
Peripheral Overview

dsPIC30F Peripherals



Packages

18 SDIP
20 SSOP

28 SDIP
28 SSOP

40 DIP
44 TQFP

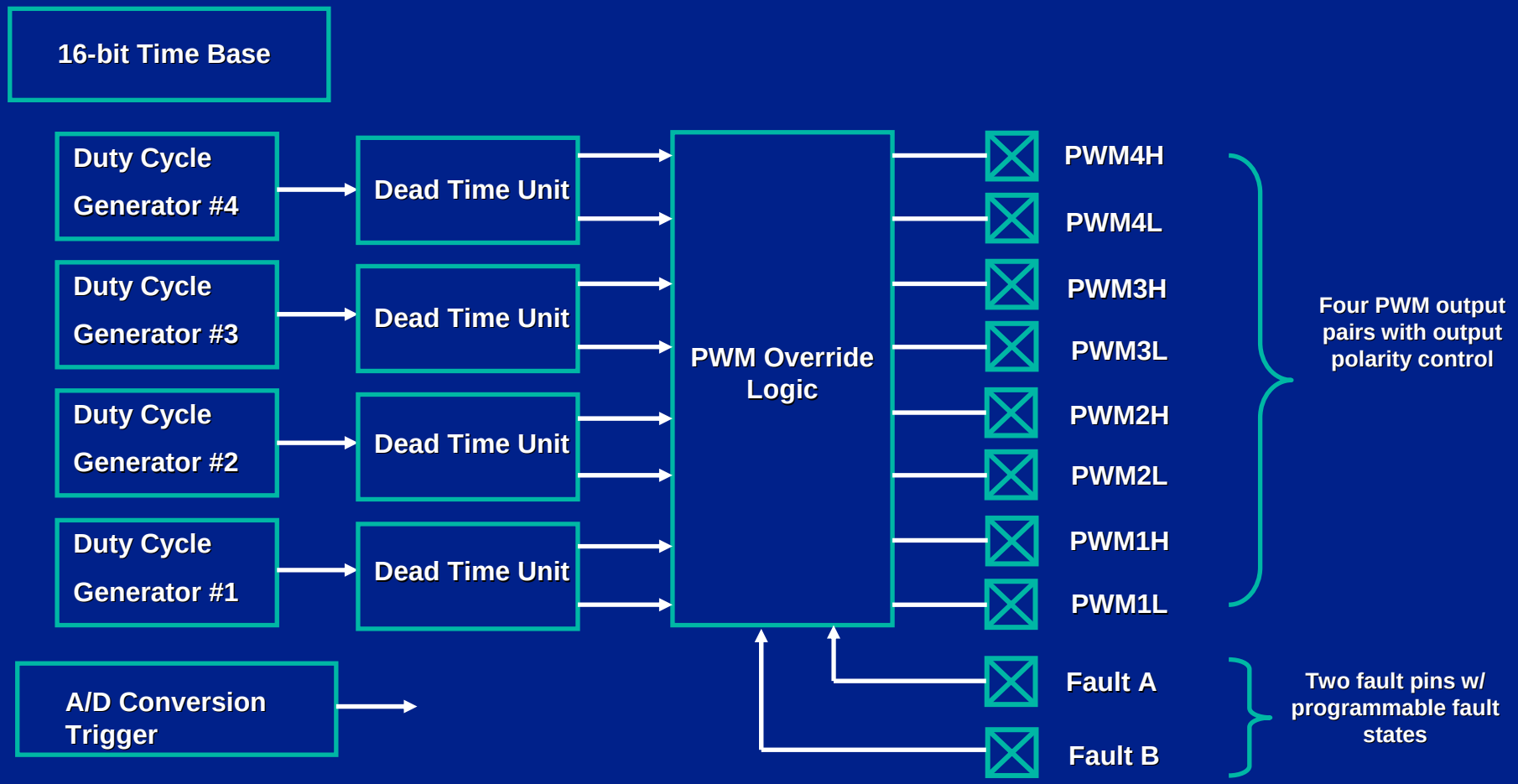
64 TQFP

80 TQFP

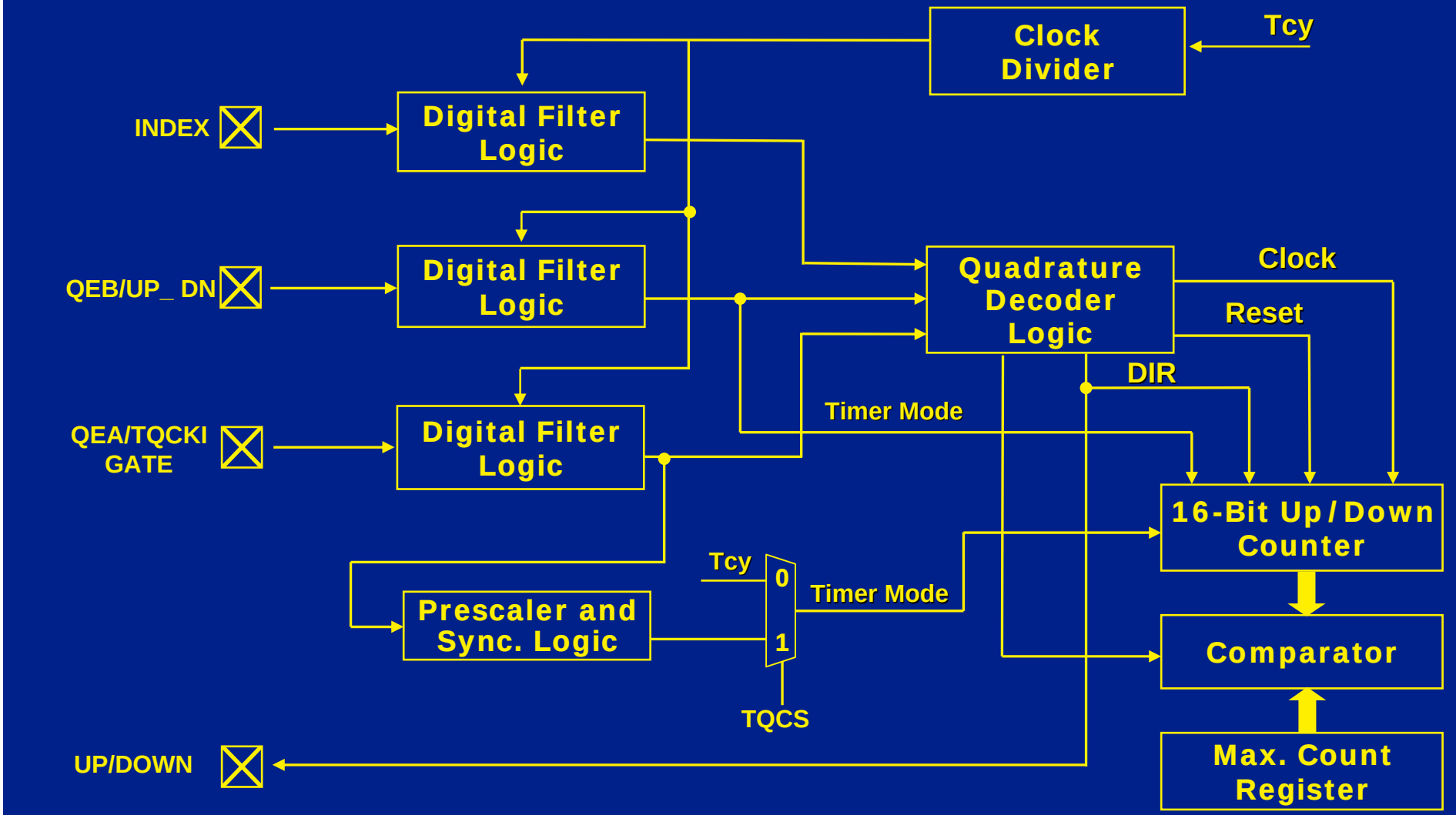
Timers/Counters Overview

- Five 16-bit General Purpose Timers / Counters
  - Similar functionality between all 5 timers
- Period Registers for Each
  - Interrupt generation on match
  - Reset on match
- Gated Timer Operation on Each
  - Interrupt-on-falling edge of gate
- Four of These Timers can Make a 32-bit Timer/Counter

Motor Control PWM Block Diagram



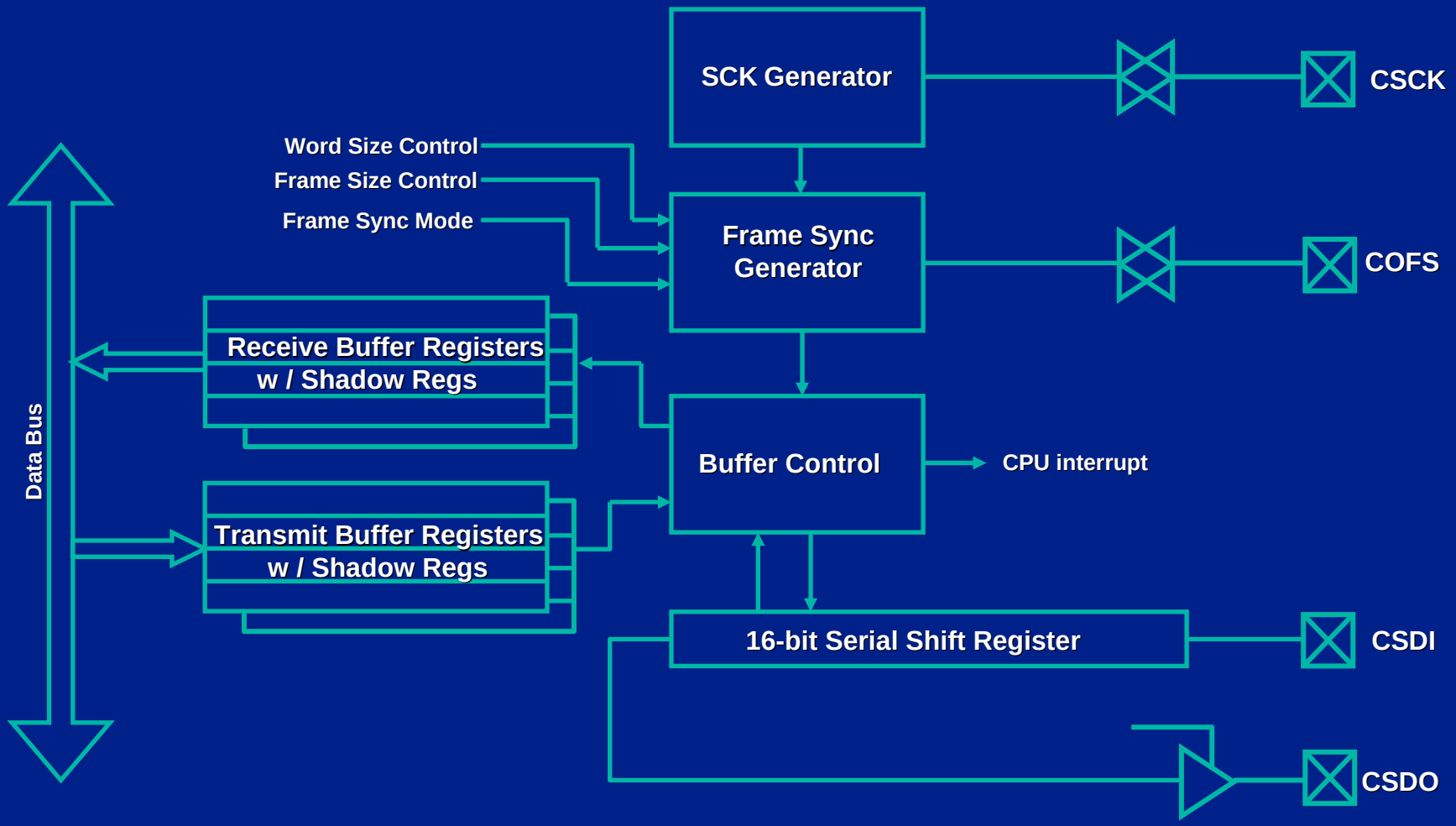
QEI Block Diagram



DCI Features

- Support for PCM Audio Codecs
  - DCI supports speech, modem and general audio applications
- Automatic Synchronous Serial Data Transfer
- Support for I2S and AC'97 Protocols
- TDM Features Support up to 16 Data Time Slots
- Module has up to 4 Word Buffer (16-bit words)
- Master or Slave Operation
- Separate Baud Generator for SCK Signal

DCI Block Diagram

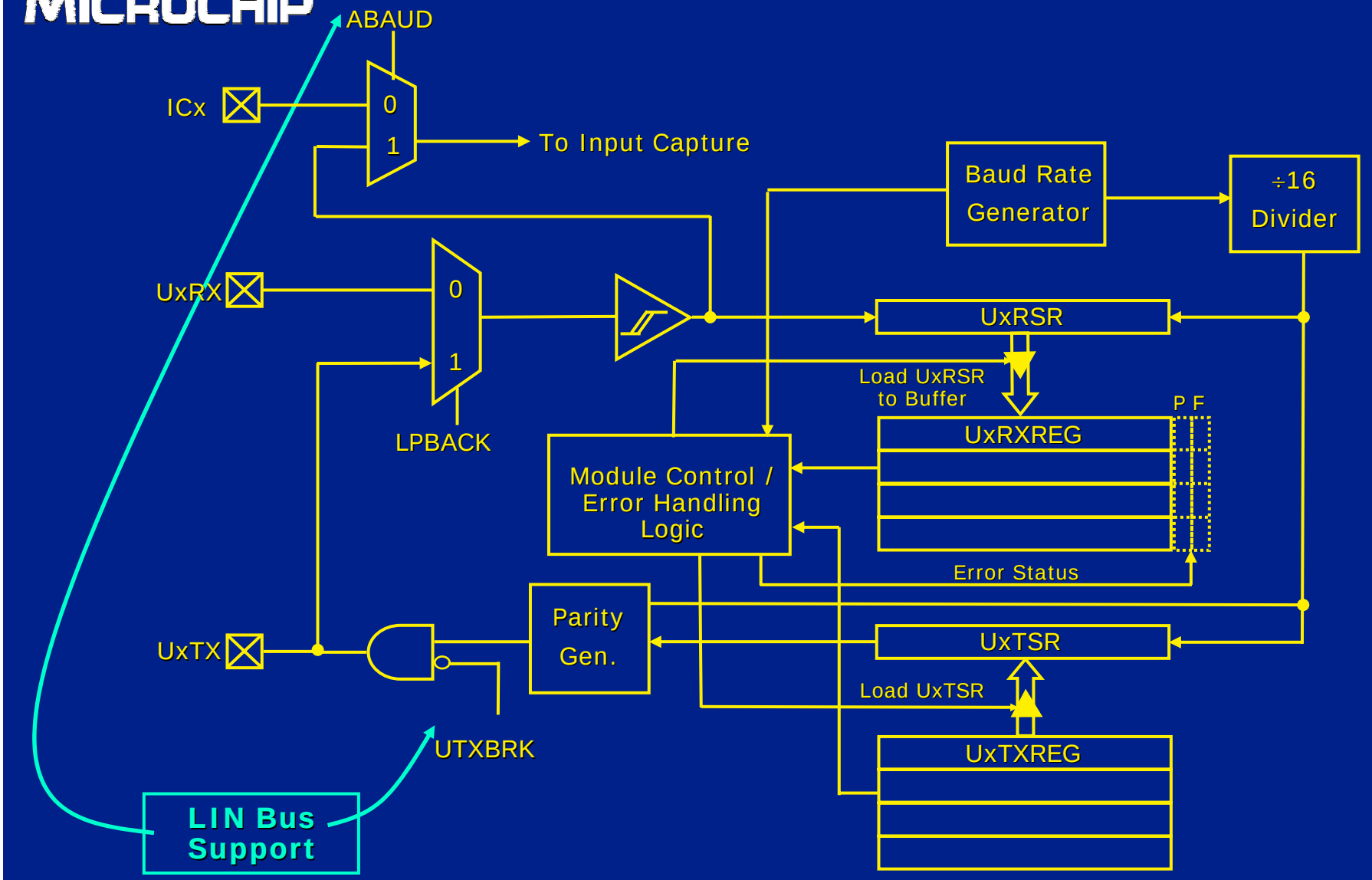


UART Module

UART - Overview

- Serial TX and RX of 8-bit or 9-bit Data
  - Full-duplex, asynchronous communication
  - Support for communication protocols such as RS-232, RS-422, RS-485 and LIN
  - 4-deep transmit and receive buffers
  - Transmit and receive interrupts
  - Error detection
  - Support for receiver addressing
- Additional Features
  - Loopback mode
  - Alternate TX/RX pins on some devices
  - Wake-up from SLEEP

UART



UART

- Baud Rate Generator
  - Each UART has a 16-bit baud rate generator
  - Baud Rate controlled by UxBRG register

Baud Rate = Fcy / (16 * (UxBRG + 1))

UxBRG = Fcy / (16 * Baud Rate) - 1

(Fcy = Instruction Cycle Frequency)

  - TX and RX use same baud rate

UART

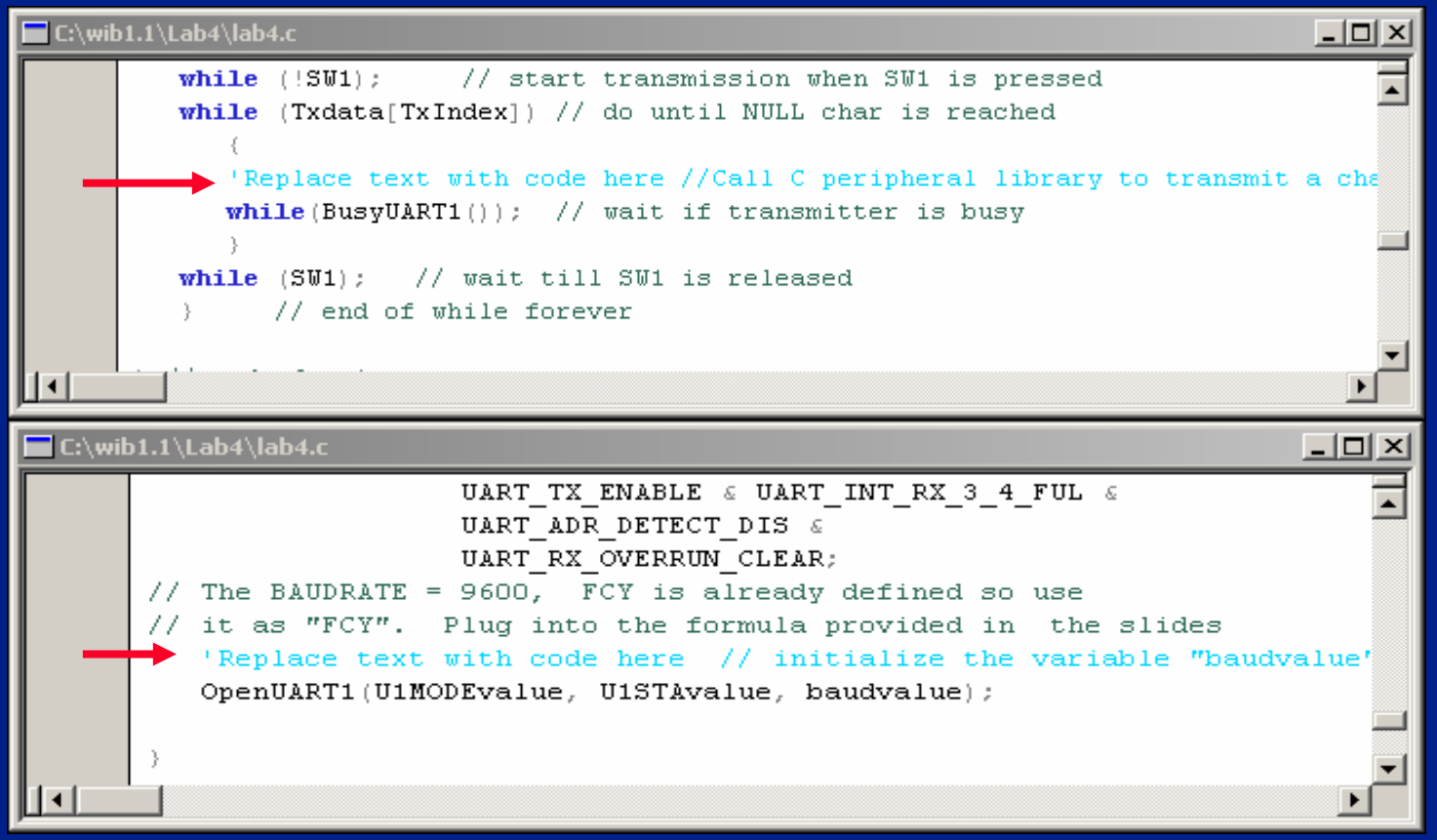
- Registers to Configure the UART
  - UxMODE, UxSTA, UxBRG
- C Library for the UART
  - OpenUARTx(int configmode, int configsta, int setbrg)
  - CloseUARTx()
  - unsigned int ReadUARTx()
  - WriteUARTx(int data)
- C Libraries also Support Classic C Type Routines:
  - getcUARTx(), putcUARTx(), putsUARTx() etc.
- Other Functions Available are:
  - BusyUARTx(); DataRdyUARTx();

Lab 4 Instructions

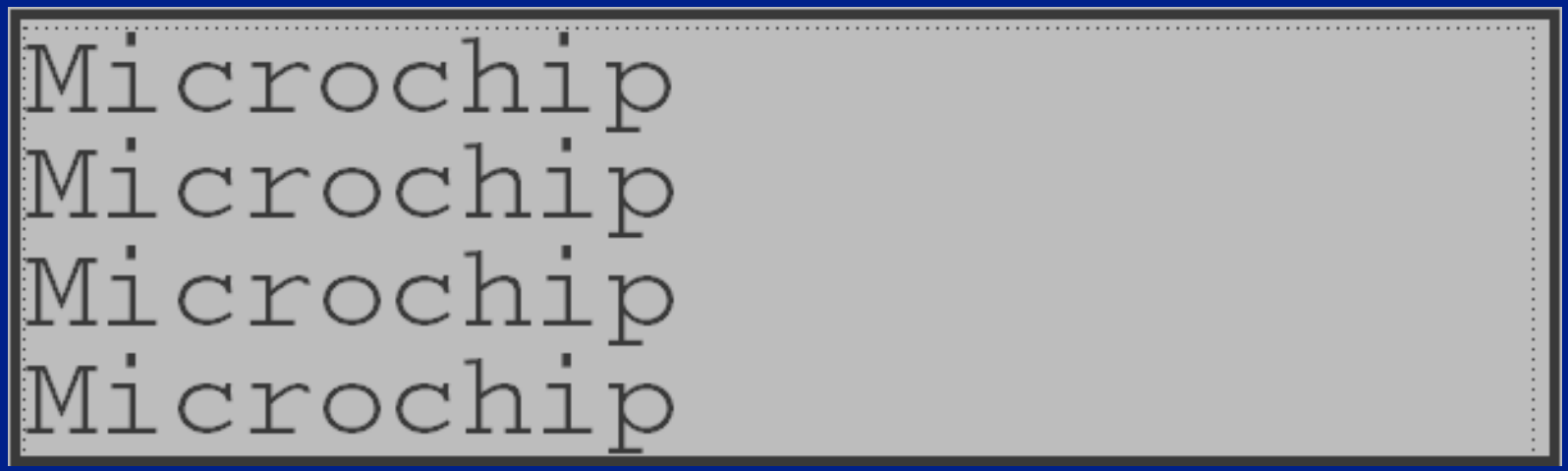
- Use Lab4.mcw Already Defined
  - libp30F6014.a and UART.h are already in the project
- Use C Libraries to Set UART1 Baud Rate to 9600
  - Use the formula U1BRG = Fcy / (16 x Baud Rate) - 1
  - Note that Fcy is already defined as ‘FCY’

  OpenUART1(U1MODEvalue, U1STAvalue, baudvalue);
- Transmit a Text Message Ending with a NULL
  - Use WriteUARTx(int data) - Note: int, not char
- Receive Buffer Receives Transmitted Data
  - Loopback mode is enabled
  - When SW1 is pressed LCD should display “Microchip”

Lab 4 Instructions



Lab 4 Result



Solution for Lab 4

- Solution

Line 54:

```
WriteUART1((int)Txdata[TxIndex++]);
```

Line 83:

```
baudvalue = FCY / 16/BAUDRATE - 1;
```


A/D Converters

10- or 12-bit

High Speed, Multi-channel

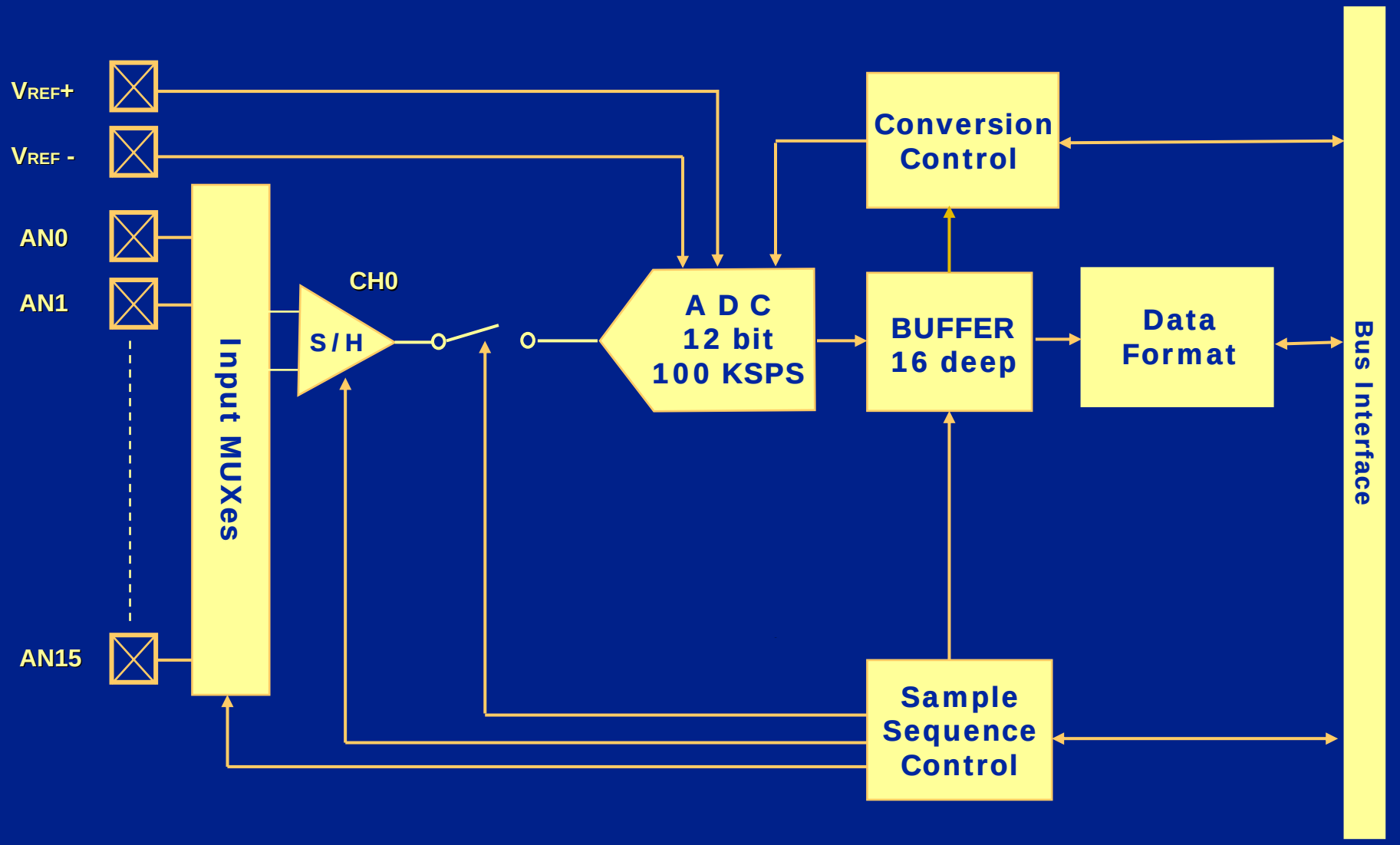
A/D Converter

- 10-bit A/D
  - 10-bit resolution with +/- 1-bit accuracy
  - 500K samples/second conversion rate
  - Up to 16 analog inputs, 4 S/H amplifiers
- 12-bit A/D
  - 12-bit resolution with +/- 1-bit accuracy
  - 100K samples/second conversion rate
  - Up to 16 analog inputs, single S/H amplifier

A/D Converter

- Common Features
  - External VREF+ and VREF-
  - Analog input range: (VREF-) to (VREF+)
  - Allows unipolar differential measurements
  - Programmable sampling sequence
    - 16 sample, dual-ported buffer
  - Multiple conversion trigger sources
  - Many conversion scheduling options

12-bit A/D Converter



12-bit A/D Converter

- Controlling Start of Sampling
  - ASAM = 1 - Automatically begin sampling after conversion
    - Maximizes the sampling time
  - ASAM = 0 - Does not automatically begin sampling
    - User must set SAMP bit to begin sampling
    - Clear SAMP bit to end sampling and start conversion
  - SAMP bit is in the ADCON1 register

12-bit A/D Converter

- Controlling Start of Conversion
- SSRC Bits Select Method of Conversion Start
  - Manual trigger
    - Clear SAMP bit to start conversion
  - Auto sample time
    - Internal T_{AD} count starts conversion
  - Motor Control PWM special event trigger
    - 10-bit A/D only - synchronize measurement
  - Timer3 period match trigger starts conversion
  - INT0 pin active edge starts conversion

12-bit A/D Converter

- When is the Result Available?

- SMPI Bits Specify Samples per Interrupt
- Interrupt on Completion of nth Conversion
  - ADIF bit in IFS0

ADCON2 Register

VCFG2	VCFG1	VCGF0	OFFCAL	-	CSCNA	-	-
bit15	14	13	12	11	10	9	bit8

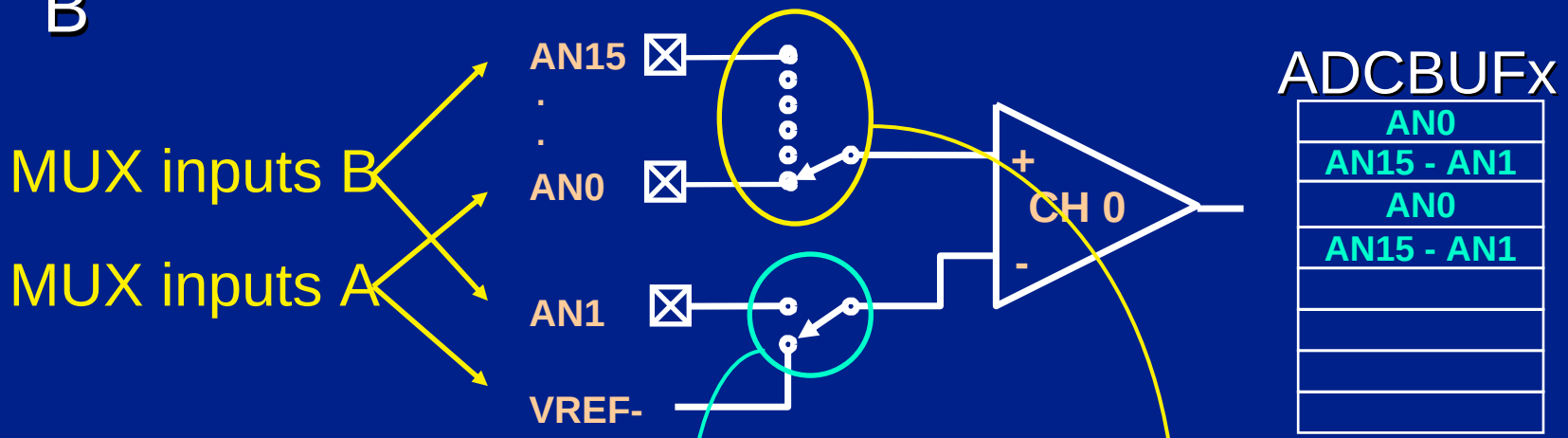
BUFS	-	SMPI3	SMPI2	SMPI1	SMPI0	BUFM	ALTS
bit7	6	5	4	3	2	1	bit0

12-bit A/D Converter

- Where is the ADC Result Stored?
  - 16-word result buffer - ADCBUF0,1,2…E,F
  - Buffer resets to ADCBUF0 after each interrupt
    - Overwrites ADCBUF0 after next conversion
    - Can software unload buffer fast enough?
  - Buffer Fill Mode - BUFM = 1
    - Buffer split into two 8-word buffers
    - BUFS shows buffer used by A/D Converter
    - BUFS = 1, ADC fills ADCBUF8 - ADCBUFF
    - Max 8 samples per interrupt

12-bit A/D Converter

- Alternate Input Sample Mode
- ALTS = 1 - Alternate Between MUX Inputs A and B



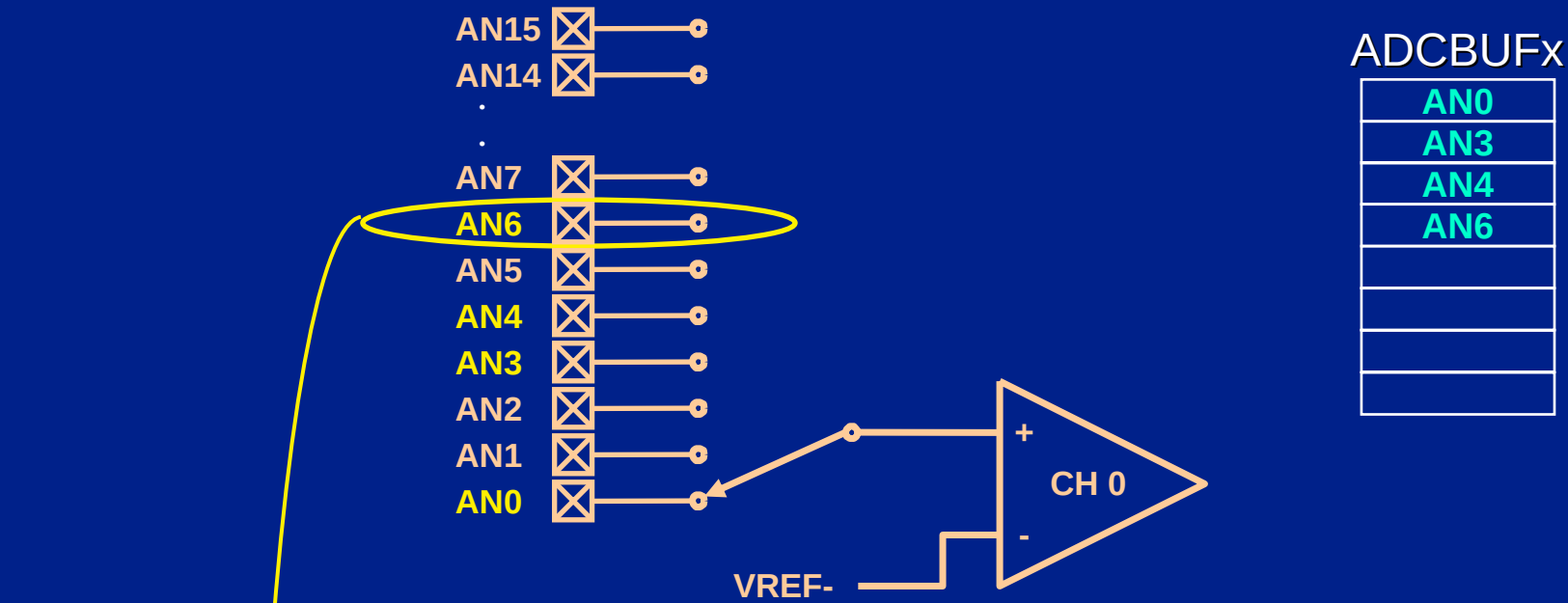
ADCHS Register

CHXNB<1:0>	CHXSB	CH0NB=0	CH0SB<3:0>=0000				
bit15	14	13	12	11	10	9	bit8

CHXNA<1:0>	CHXSA	CH0NA=1	CH0SA<3:0>=1111				
bit7	6	5	4	3	2	1	bit0

12-bit A/D Converter

- Channel Scanning on MUX A - CSCNA = 1



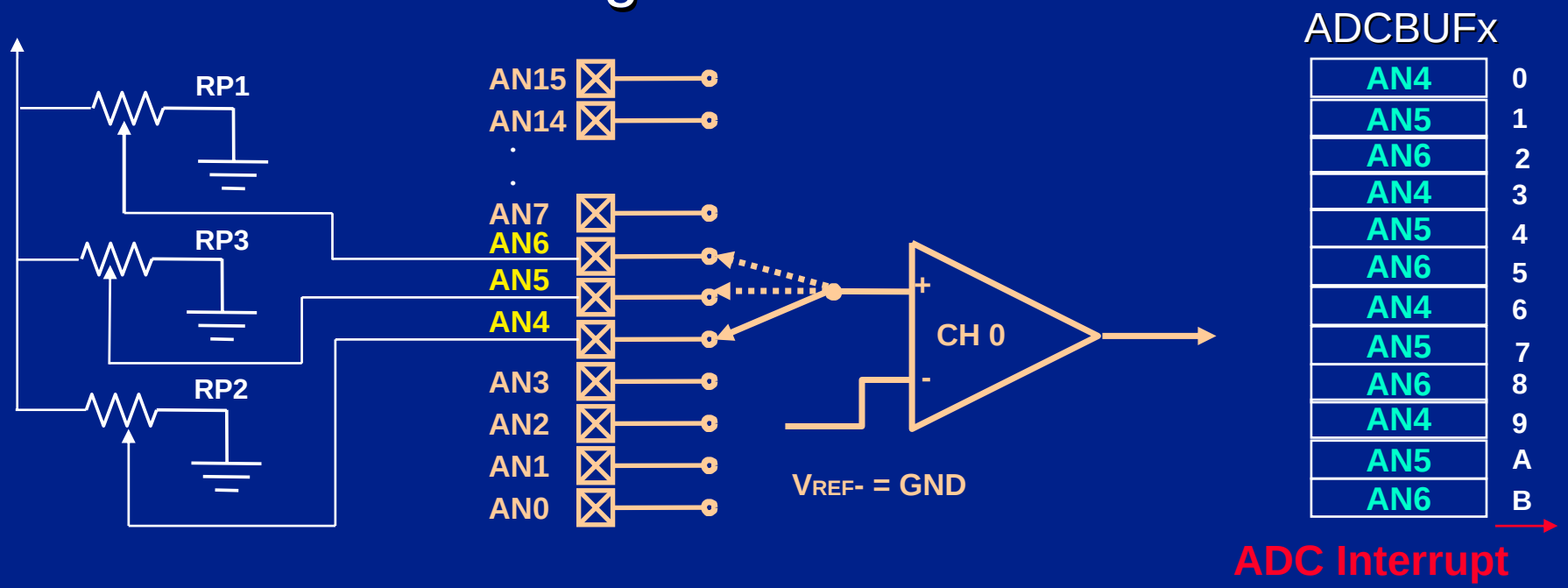
ADCSSL Register

CSSL15=0	CSSL14=0	CSSL13=0	CSSL12=0	CSSL11=0	CSSL10=0	CSSL9=0	CSSL8=0
bit15	14	13	12	11	10	9	bit8

CSSL7=0	CSSL6=1	CSSL5=0	CSSL4=1	CSSL3=1	CSSL2=0	CSSL1=0	CSSL0=1
bit7	6	5	4	3	2	1	bit0

Lab 5 Setup

- Channel Scanning on MUX A - CSCNA = 1



ADCSSL Register

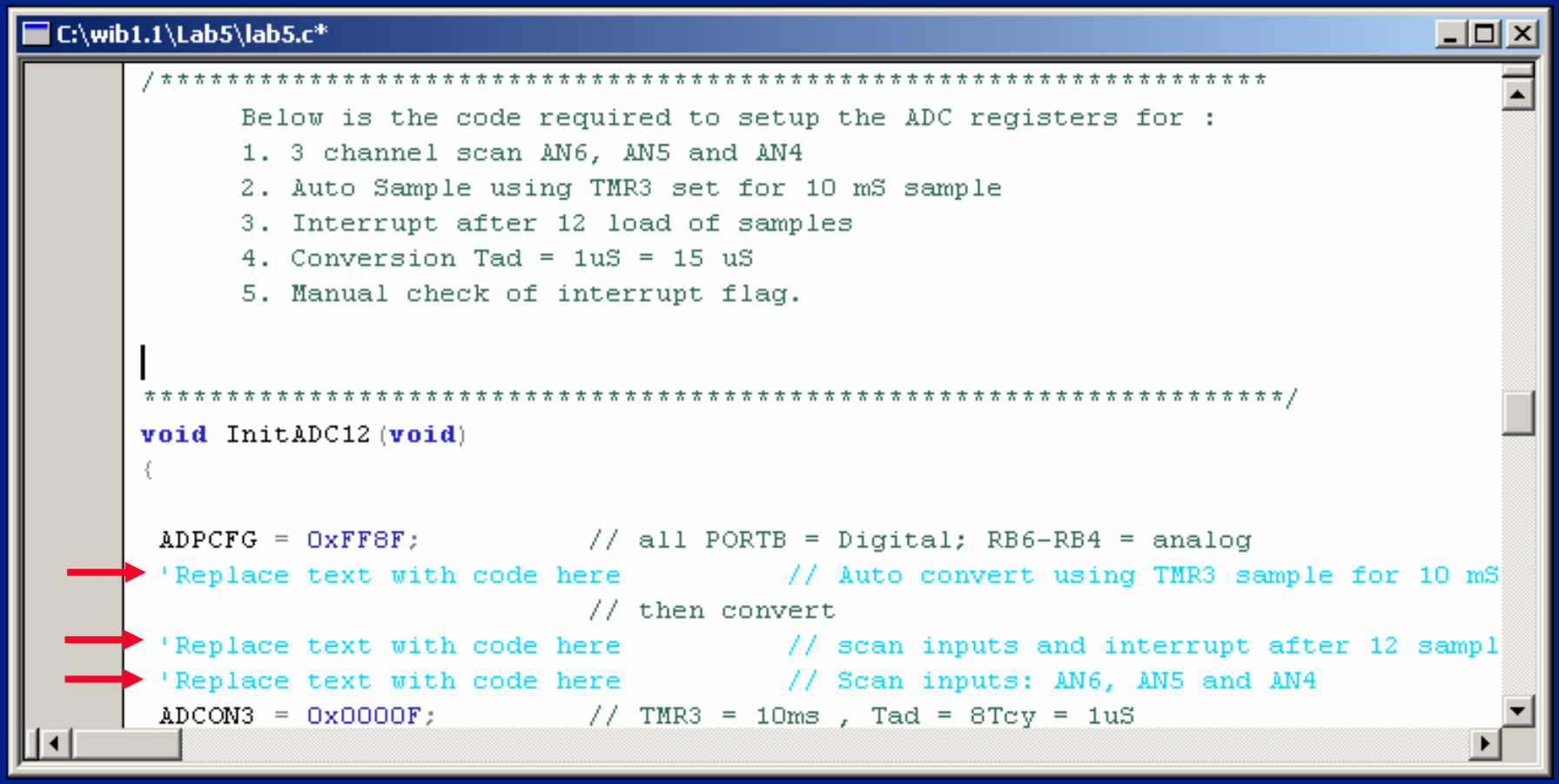
CSSL15=0	CSSL14=0	CSSL13=0	CSSL12=0	CSSL11=0	CSSL10=0	CSSL9=0	CSSL8=0
bit15	14	13	12	11	10	9	bit8

CSSL7=0	**CSSL6=1**	**CSSL5=1**	**CSSL4=1**	CSSL3=0	CSSL2=0	CSSL1=0	CSSL0=0
bit7	6	5	4	3	2	1	bit0

Instructions for Lab 5

- Use Lab5.mcw Workspace
- Setup Auto Sample Using 10mS TMR3 Trigger to Stop Sampling and Start Conversion
- Setup ADC to Scan AN4, AN5 and AN6 Connected to RP2, RP3 and RP1 Respectively
- Repeat Scan 4 Times by Setting Interrupts to Occur after 12 Sample/Convert Sequences
- Average the Pot Values and Display the Hex Value on the LCD Display.

Instructions for Lab 5



Lab 5 Result

```
RP1   =   801
RP2   =   812
RP3   =   7EB
```


Solution for Lab 5

- Solution

Line 121: ADCON1 = 0x0044;

Line 123: ADCON2 = 0x042C;

Line 124: ADCSSL = 0x0070;

Lab 6:
Digital Filtering Overview

Digital Filters

- What is Digital Filtering?
  - The digital processing of an input signal to produce a desired output signal with a certain frequency response or characteristic
- Advantages of Digital Filtering (Over Analog):
  - Less affected by noise
  - Lower power consumption
  - Programmable systems (easy to modify)
  - Minimal characteristic drift with age
  - Minimal characteristic drift with temperature

Digital Filter Types

- Digital Filters are Classified into Two Types
  - Finite Impulse Response (FIR)
  - Infinite Impulse Response (IIR)
- FIR and IIR Filters can be Designed to Provide Some Desired Response
  - Low Pass Filter (passes low frequencies)
  - High Pass Filter (passes high frequencies)
  - Band Pass Filter (passes a frequency band)
  - Band Stop Filter (stops a frequency band)

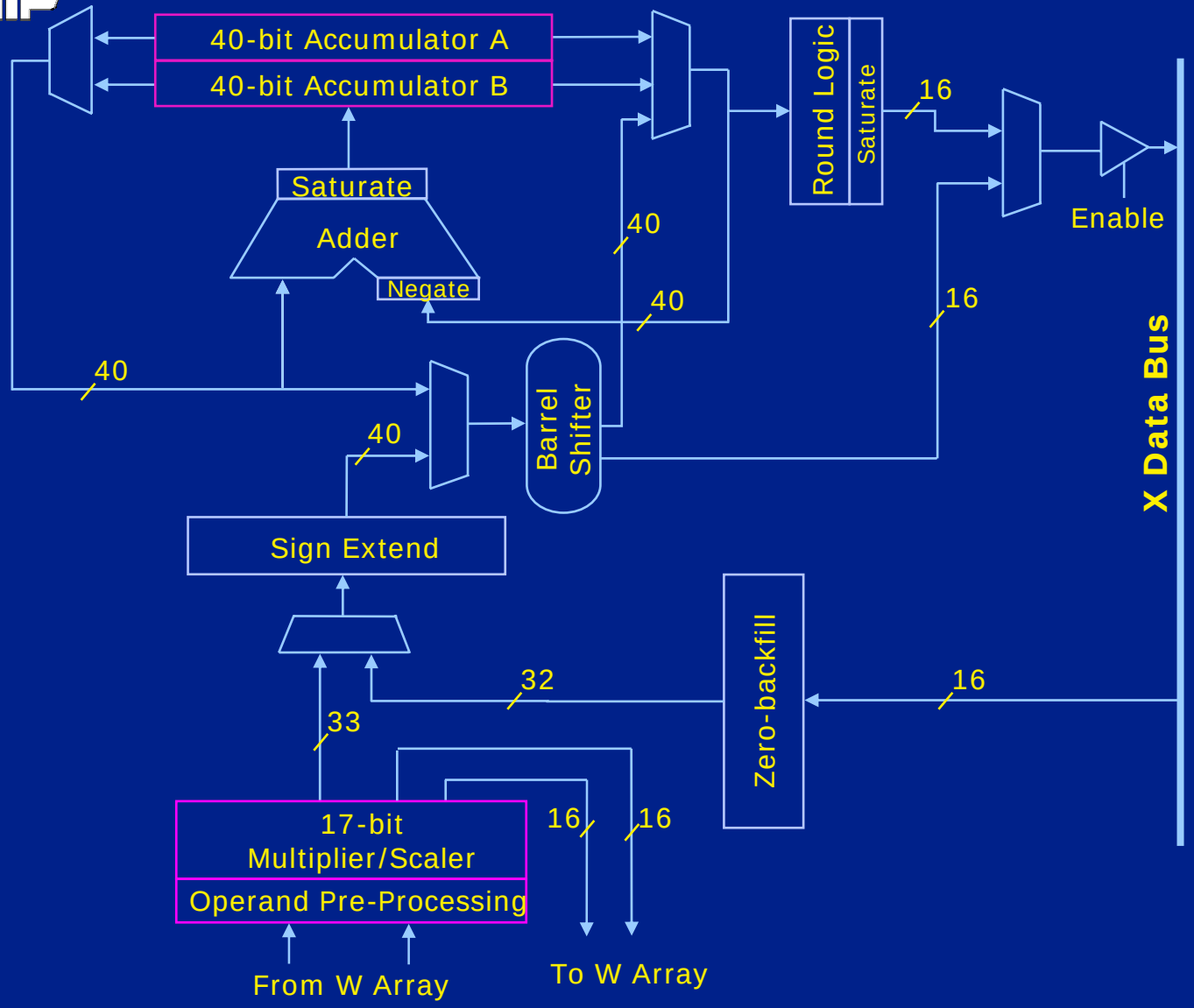
DSP Features

- All DSP Operations Execute in a Single Cycle
- Single Clock Cycle MAC Instruction:
  - Dual Address Generator Units (AGU) support parallel operand (data and coefficient) fetches
  - Accumulator write-back
- Fractional/Integer Multiplier for 16 x16-bit Operation
- Two 40-bit Accumulators
  - Supports algorithms using complex values (e.g. FFT's) or algorithms requiring multiple filter loops
  - Less need for data input scaling to prevent overflow
  - Separate overflow detect, flags and branches for each accumulator
  - Optional saturation mode

DSP Features

- 40 Stage Barrel Shifter (up to 16-bits Left or Right)
  - Shift accumulators, W registers or memory
- Accumulator Rounding During Store Operation
  - Conventional rounding
  - Convergent (unbiased) rounding
- Modulo Addressing (for Filters): Both AGU's
- Bit Reversed Addressing (for FFT's): One AGU
- Zero Overhead Instruction Loop Support
  - REPEAT: Repeat next instruction N times
  - DO: Repeat loop N times
  - Constant or variable loop count

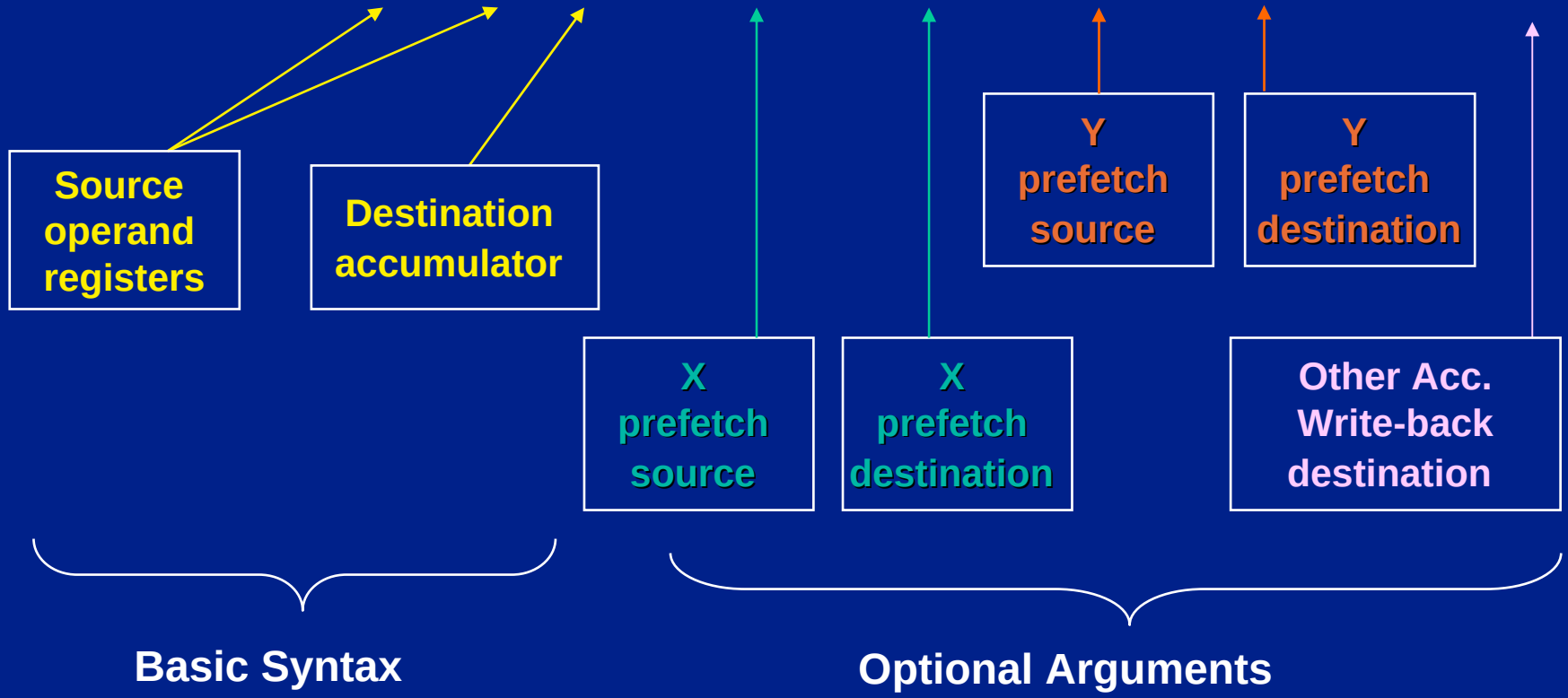
DSP Engine Block Diagram



Single Cycle MAC Instruction

FIR Filter Tap = 1 cycle

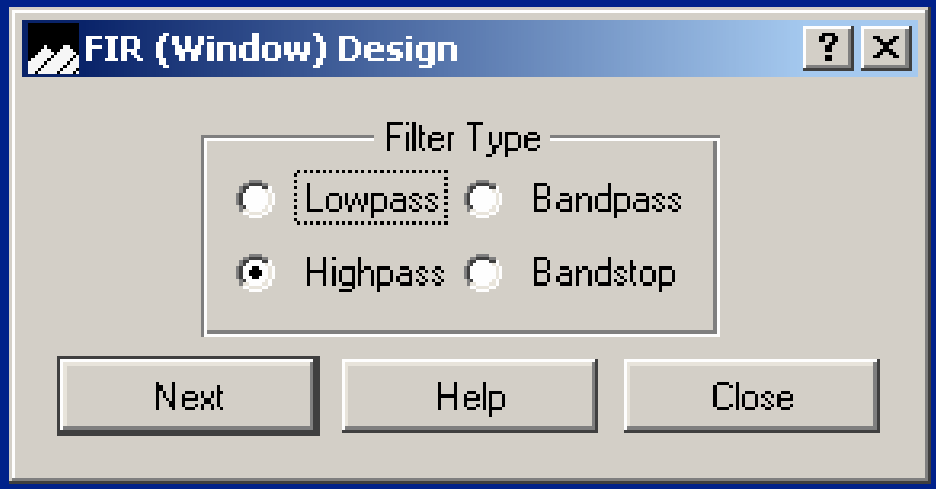
- Sample Instruction Syntax
  - **MAC W4*W5, A, [W8]+=2, W4, [W10]-=6, W5, W13**



Using dsPIC™ Digital Filter Design Simple as 1,2,3….

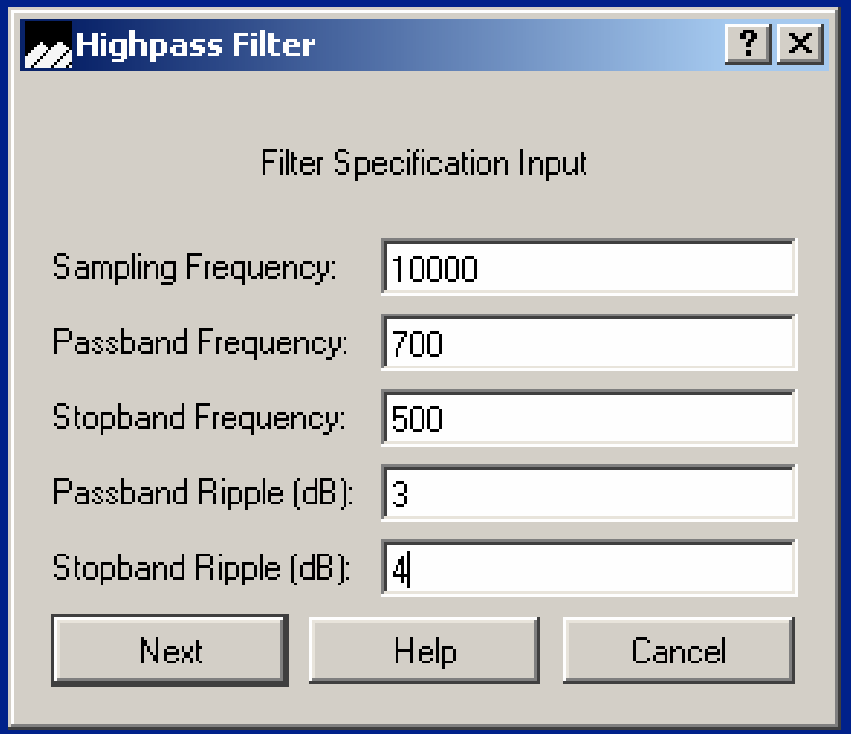
- Create a High Pass Filter with a Cutoff Frequency of 700 Hz

  Design -> FIR Window Design -> Highpass

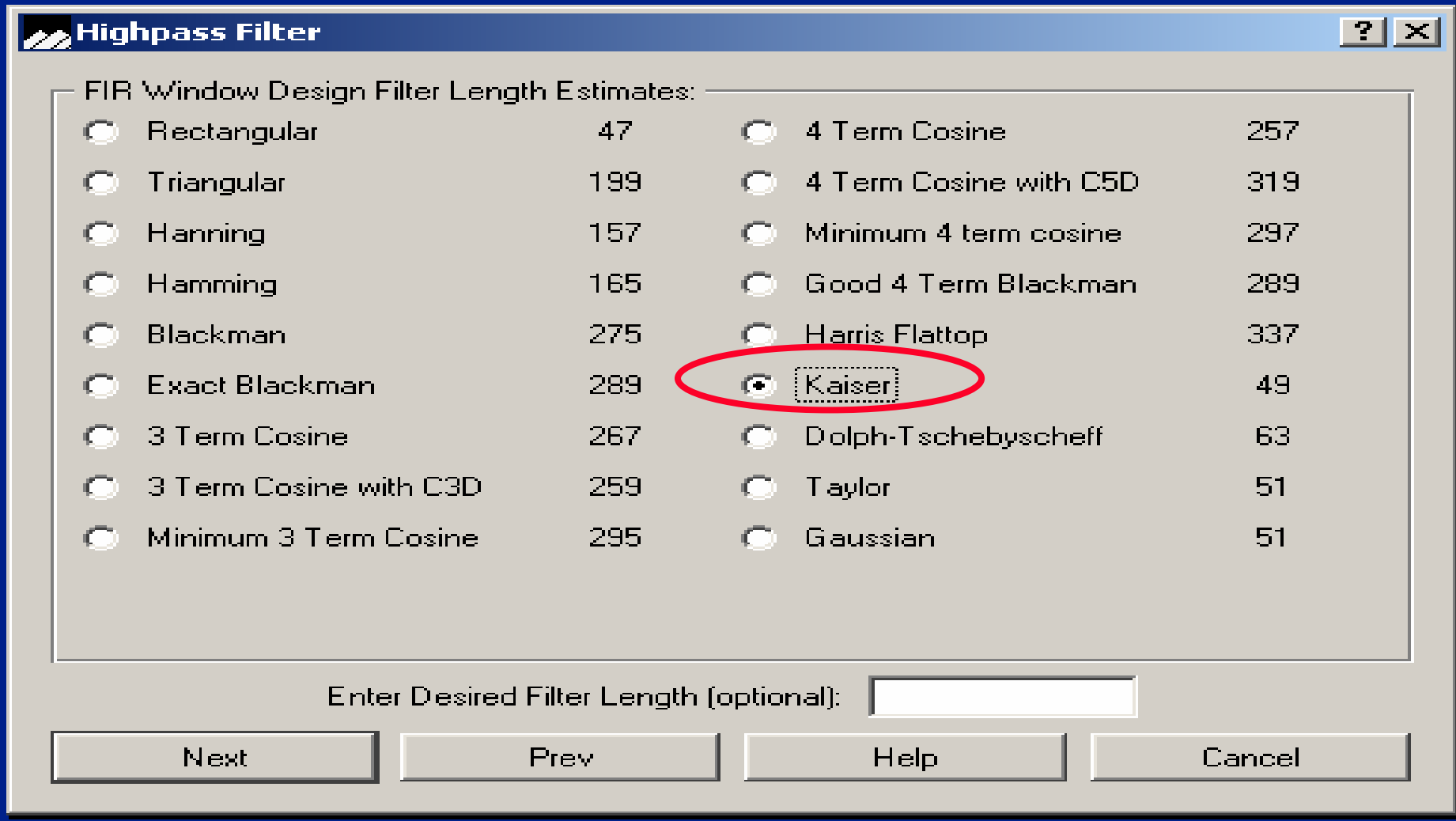


Specify the Filter Characteristics

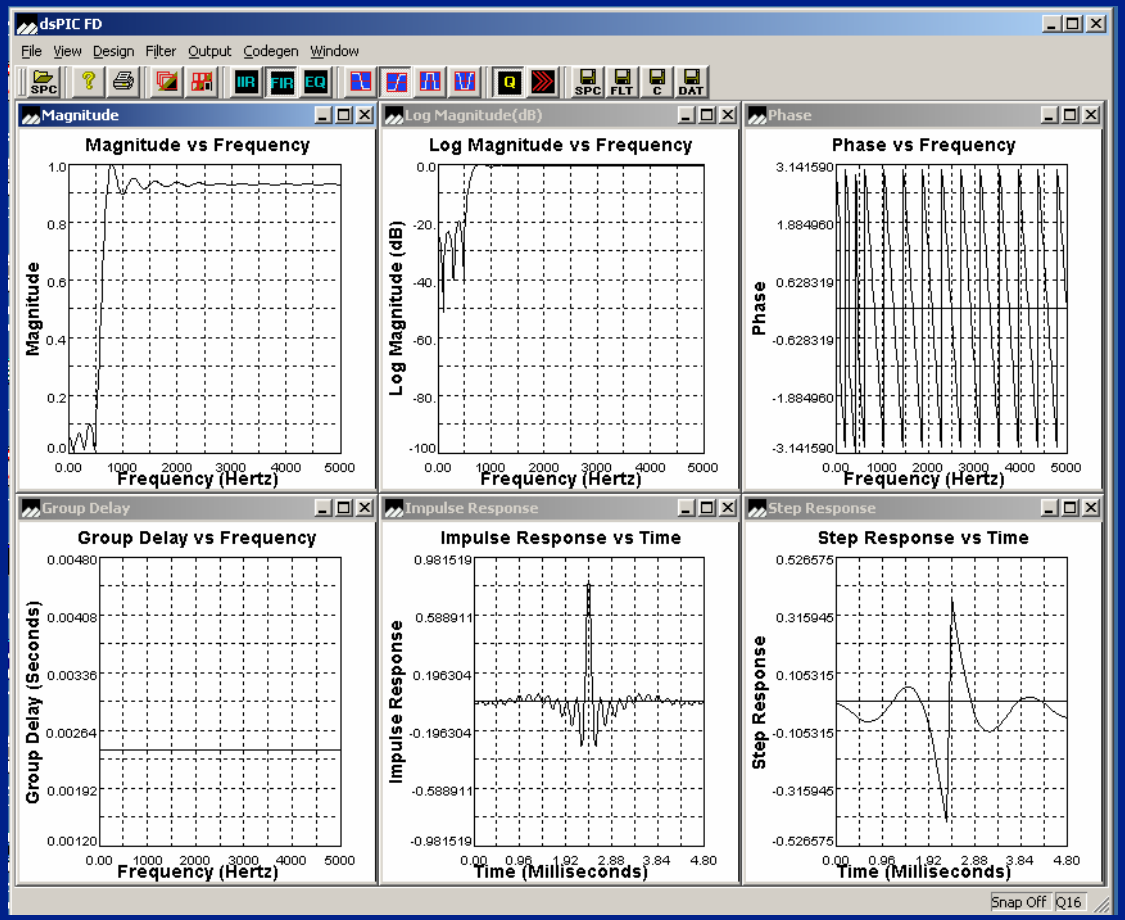
- Sample Frequency = 10,000 Hz
- Passband = 700 Hz
- Stopband = 500 Hz
- Passband Ripple = 3 dB
- Stopband Ripple = 4 dB



Choose the Windowing Function Applied to the Filter

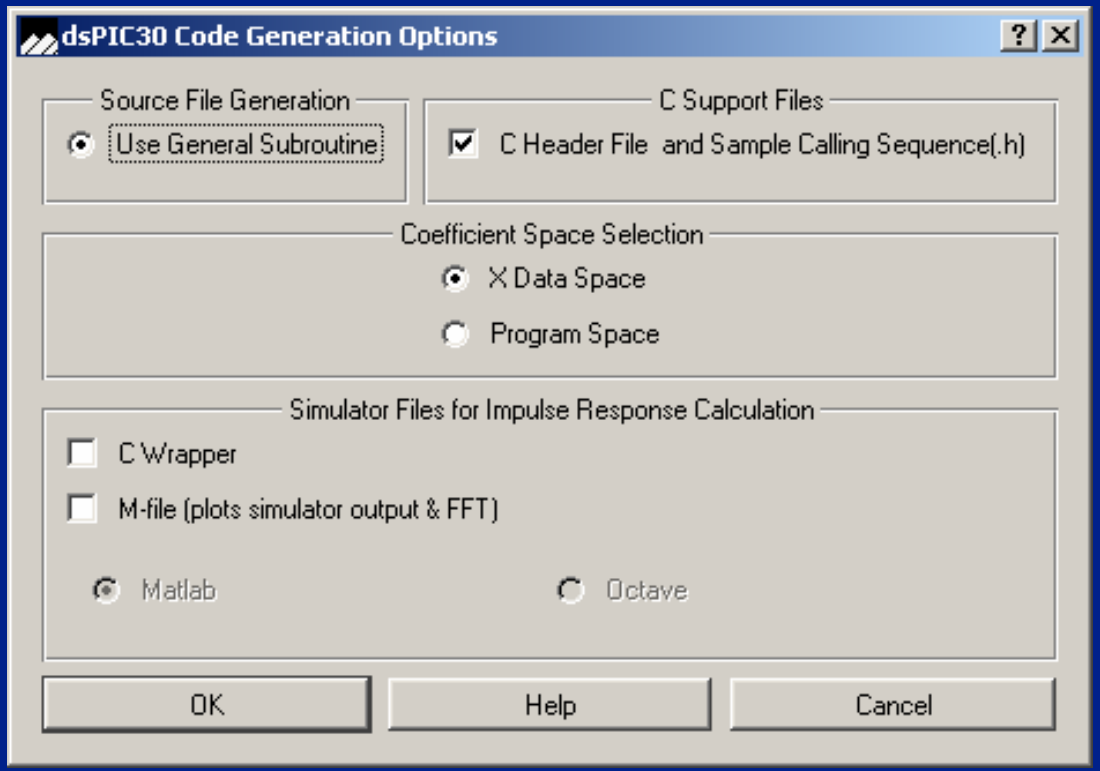


The Filter Characteristics are Displayed....



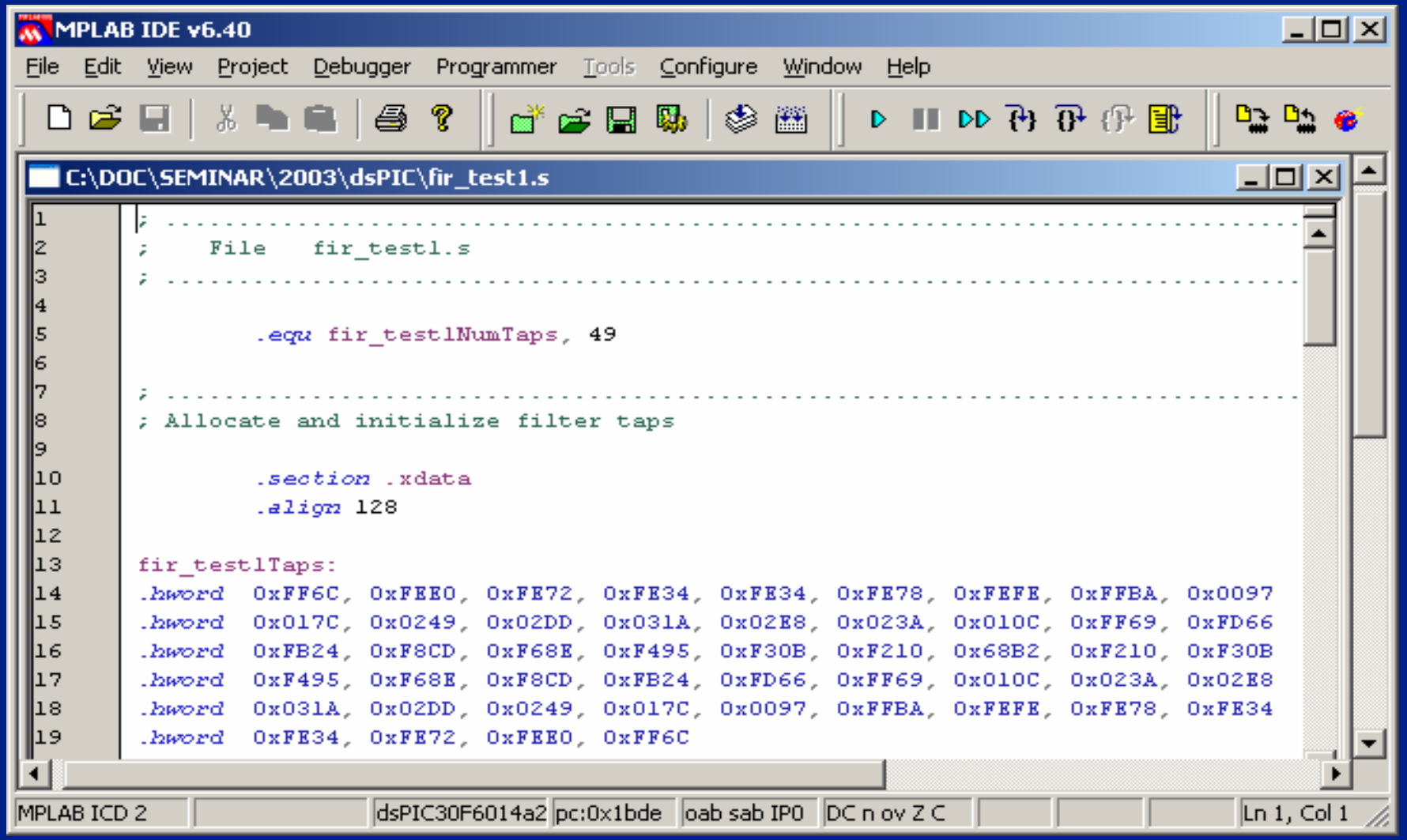
Generate the Filter Coefficients and Source Code

- Codegen -> Microchip -> dsPIC30



- Save the File as “fir_test1” in “C:\wib1.1\lab6\”

Filter Design Output File: Filter Taps Definition



DSP Performance Benchmark

Cycle Count Comparison (Normalized) Using BDTI Benchmarks™

BDTI Benchmark Results	Microchip dsPIC [1] 30Fxxx	TI [3] 'C24x/C24xx	Motorola [2] 56F83xx (56800E)	ADI [2] 2199x (219x)
Vector Dot Product	1.00	1.16	1.07	1.32
Real Block FIR	1.00	1.32	1.11	1.02
Two-Biquad IIR	1.00	2.12	1.29	1.47
Control	1.00	2.05	2.06	1.76

Notes: 1. Projected Results
2. Buyer's Guide to DSP Processors, 2001 Edition
3. Buyer's Guide to DSP Processors, 1999 Edition

Results (c) 1999-2004 Berkeley Design Technology, Inc.
Contact info@BDTI.com for info.

DSP Performance Benchmark

Execution Time Comparison (Normalized) Using BDTI Benchmarks™

BDTI Benchmark Results	dsPIC [1] 30Fxxx (30 MIPS)	TI [3] 'C2xx/C24xx (40 MIPS)	Motorola [2] 56F83xx (60 MIPS)	ADI [2] 2199x (219x) (160 MIPS)
Vector Dot Product	1.00	0.87	0.53	0.25
Real Block FIR	1.00	0.99	0.55	0.19
Two-Biquad IIR	1.00	1.59	0.65	0.28
Control	1.00	1.54	1.03	0.33

Notes: 1. Projected Results
2. Buyer's Guide to DSP Processors, 2001 Edition
3. Buyer's Guide to DSP Processors, 1999 Edition

Results (c) 1999-2004 Berkeley Design Technology, Inc.
Contact info@BDTI.com for info.

Additional Benchmarks

Benchmark	Cycle Count	Execution @ 30MIPs
FIR Filter Tap	1	33 ns
Biquad IIR (4 coeff) (single sample)	8	267 ns
PID (core only)	7	233 ns
PID (core, error calculation, limit check, wind-up control)	31	1.02 us
Division	18	600 ns

Interfacing to the DSP Library

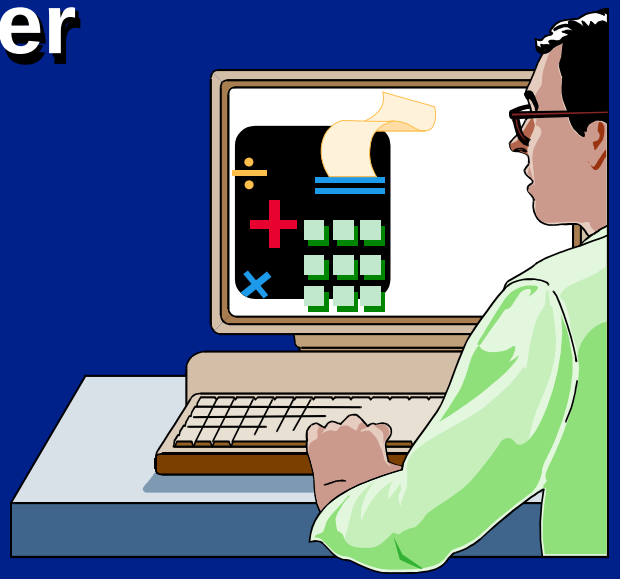
- DSP Library is Both “C” Callable and Assembly Callable
- “C” Interface Requires Just 2 files:
  - LIBDSP.A and DSP.H
  - Provided in C30Tools directory
- Library Typedefs

```
typedef int fractional;
```


Digital Filters and the DSP Library

- Library Filters Operate on Blocks of Data
- One Filter Initialization Call is Required
- FIR
  - `FIRDelayInit (&fir_test1Filter);`
  - `FIR (BLOCK_LENGTH, &filter_output_array[0], &array[0], &fir_test1Filter);`
- IIR (Transposed and Canonic Realizations)
  - `IIRTransposedInit (&fir_test1Filter);`
  - `IIRTransposed (BLOCK_LENGTH, &filter_output_array[0], &array[0], &fir_test1Filter);`
  - For Canonic, use `IIRCanonicInit()`, `IIRCanonic()`

Lab 6:
Digital Filtering and the dsPIC® Digital Signal Controller



Lab 6 Overview

- Objectives
  - Use filter coefficients created by dsPIC™ Filter Design tool
  - Learn how to integrate the designed filters into your project
  - Learn how to use the DSP Library

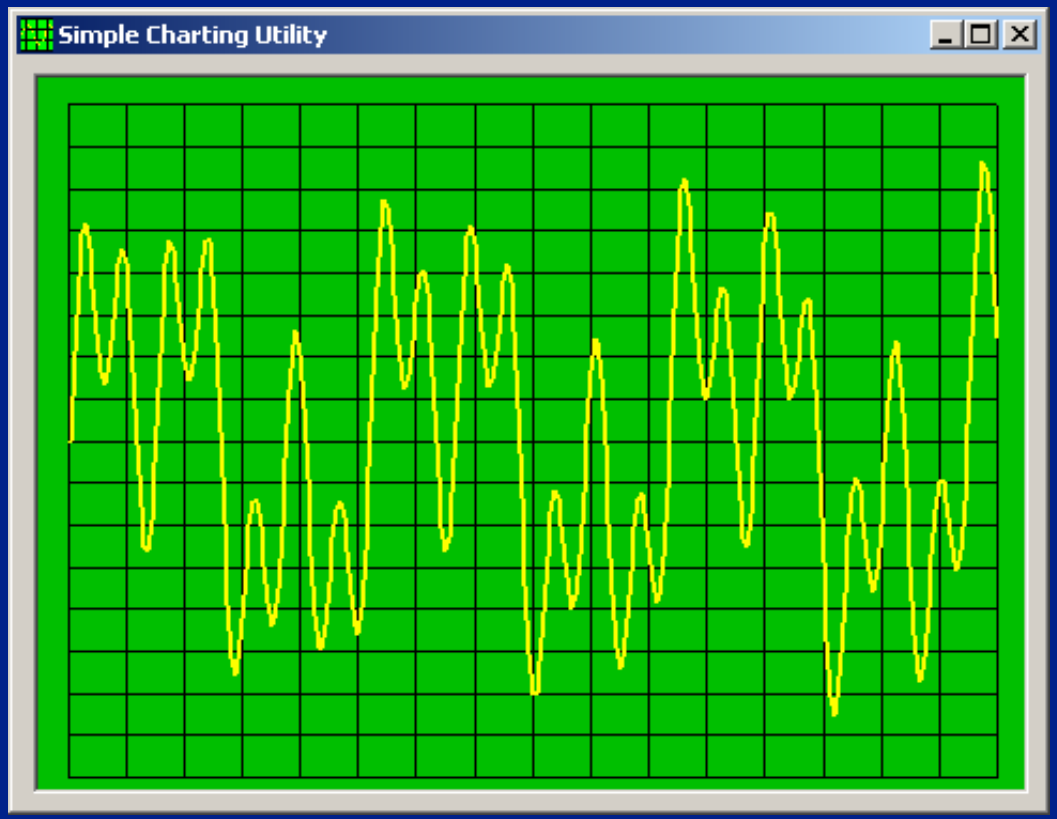
Lab 6 Overview

- Tasks
  - The input signal contains the sum of three sine waves at 847, 367 and 123 Hz.
  - The dsPIC™ Filter Design tool was used to design a high pass filter at 700 Hz to remove the 367 and 123 Hz noise components. The coefficients are stored in the file fir_test1.s
  - Call the “C” FIR library function which uses the filter coefficients to remove the identified noise components
  - Extra Credit:

    Benchmark the filter routines using TMR1

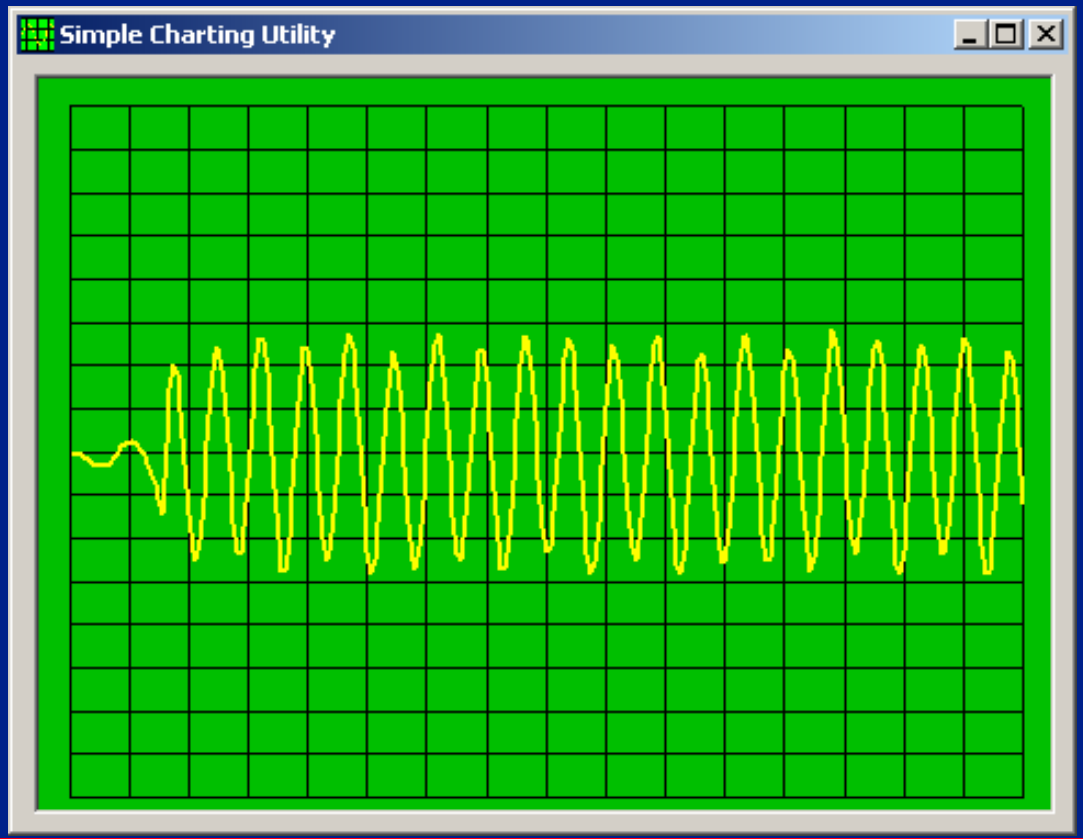
Input Signal

- Pressing the SW1 Button Displays the Filter Input on the LCD Display:



Output Signal

- Pressing the SW2 Button Displays the Filter Output on the LCD Display:



Conclusion

- The dsPIC30FXXXX Architecture and Peripheral Set is Ideal for DSP Applications
- Development Tools are Versatile, Inexpensive and Easy to Use When Developing Applications
- DSP Tools Reduce Mathematical Complexity With Easy Code Generation
- DSP Application Notes and Reference Designs:
  - Motor Control, Filters, Speech and Connectivity

Survey

- Please dsPIC30F Workshop in a Box Survey at:

http://techtrain.microchip.com/surveys/dspic/dspic_wsiab.aspx

- Those Who Complete the Survey Will be Entered in a Drawing for Several Valuable Prizes

Thanks for Attending!

Good Luck in Your Development