This seminar is an introduction to the MPLAB Integrated Development Environment.

My name is Darrel Johansen and I'm a manager for the Development Systems Group here at Microchip Technology.
This program is intended to explain what the MPLAB IDE is used for and to walk you through a simple project.

We'll answer the question: What is MPLAB IDE? And demonstrate how this integrated development environment eases the design cycle.

We'll walk through a simple project and by the time you are finished, you should understand the basics of application development using MPLAB IDE.
MPLAB® IDE is a software program that runs on your PC to provide a development environment for your embedded system design.

What is MPLAB IDE?

MPLAB IDE is a software program that runs on your PC to provide a development environment for your embedded microcontroller design.

If you are an experienced embedded systems designer, that sentence probably made complete sense to you, and you may want to skip ahead to the first project demonstration to get familiar with MPLAB IDE. If not, we'll take just a moment to explain a little about embedded systems development and what MPLAB IDE is used for.
What Is An “Embedded System?”

- Design with a Microcontroller
  - CPU on chip
  - Additional circuits on chip

What is an “embedded system?”
It is typically a design making use of the power of a small microcontroller, like the Microchip PICmicro® microcontrollers and dsPIC® digital signal controllers. These chips have a microcontroller unit and some additional circuits on the same chip to make a small control module requiring few other external devices.
The PICmicro device has program memory for the firmware, or coded instructions, to run your program. It also has file register memory for storage of variables that your program will need for computation or temporary storage and for data going to and coming from the peripheral circuits.

It has a number of peripheral device circuits on the same chip.

Some peripheral devices are called I/O ports. These are pins on the microcontroller that can be set high or low to send signals, light LEDs, drive speakers—just about anything that can be sent through a wire. Often these are bi-directional and can be configured as inputs allowing the program to respond to an external switch, sensor, or to communicate with some other external device.

In order to design such a system, you need to decide what peripherals are needed for your application. Analog to Digital converters allow you to connect the microcontroller to sensors and receive changing voltage levels. Serial communication peripherals, allow you to stream communications over a few wires to another microcontroller, to a local network, or to the internet.

Timer peripherals accurately measure signals, clock data waveforms in and out of the device, and generate precise waveforms. One kind of timer automatically resets the microcontroller if it gets "hung" or lost because of a power glitch or hardware malfunction.

Certain peripherals detect if the external power is dipping below dangerous levels, so the micro can quickly save information before the power is completely removed.

The peripherals and the amount of memory your application needs to run your program largely determine which PICmicro to use. Other factors might include the power consumed by the microcontroller and its "form factor"—the size and characteristics of the physical package that must reside on your target design.
How does an embedded controller differ from a PC like my desktop PC?

The main difference is that the embedded controller is usually dedicated to one specific task or set of tasks. A PC is designed to run many different types of programs and to connect to many different external devices. An embedded controller has a single program and, as a result, can be made cheaply to include just enough computing power and hardware to perform that dedicated task. A PC has a generalized microcomputer at its heart, while an embedded system has a specialized microcontroller for its intelligence.
An example of an embedded system is a smoke detector. Its function is to evaluate signals from a sensor and sound an alarm if the signals indicate the presence of smoke. A small program in the smoke detector either runs in an infinite loop, sampling the signal from the smoke detector, or is almost dormant in a low-power “sleep” mode, being awakened by a signal from the sensor. The program then sounds the alarm. The program would have a few other functions, such as a user test function, and a low battery alert. While a PC with a sensor and audio output could be programmed to do the same function, it would not be a cost-effective solution. Embedded designs use inexpensive microcontrollers to put small amounts of intelligence into the everyday electronic devices in our environment, such as cameras, cellphones, appliances, automobiles, smartcards, and security systems.
The design cycle for developing an embedded controller application is:

1) Do the high level design – From the features and performance desired, decide which PICmicro or dsPIC device you need, then design the associated hardware circuitry.

2) Knowing which peripherals and pins control your hardware, write the software. Use either assembly language, which is directly translatable into machine code, or using a compiler that allows a more natural language for creating programs. With these Language Tools you can write and edit code that is more or less understandable, with constructs that help you organize your code.

3) Compile or assemble the software using a Language Tool to convert your code into machine code for the PICmicro device. This machine code will eventually becomes firmware, the code programmed into the microcontroller.

4) Test your code. Usually a complex program does not work exactly the way you might have imagined, and “bugs” need to be removed from your design to get it to act properly.

5) “Burn” your code into a microcontroller and verify that it executes correctly in your finished application.
To familiarize you with the basic components of MPLAB, let’s do a simple first project. We will write a short assembly language program to toggle an output pin on a PICmicro device which could then be programmed into a simple circuit to flash an LED.

This exercise will show you how to put together a project framework, how to build an application, and how to test it with the simulator to verify that it works as expected.

All projects will have these basic steps:
Select the device. The capabilities of MPLAB IDE vary according to which device is selected. That is the first thing that must be done before anything else.
We will use the MPLAB Project Wizard to Create a Project.
In the Project Wizard we can select the language tools. For this presentation, the built in assembler and linker will be used. For other projects you might choose one of the Microchip compilers or other third party tools.
We’ll add two files for the project, a template file and a linker script. These exist in the MPLAB IDE folder and with these two files we can get started quickly.
We’ll then add some code to the template file to toggle an I/O bit.
Then we will build the project – causing our source files to be assembled and linked into machine code that can run on the selected PICmicro device.
And finally we will test the code with the simulator.
The first thing to do is to download and install MPLAB IDE on your PC. Go to www.microchip.com and go to the Development Tools section. Look for the latest production version of MPLAB software. The latest production version ends in a zero digit, like v6.40, v6.50. Interim versions end in non-zero digits like v6.42. For getting started, choose the latest production version. The interim versions are posted mainly for early support of new devices and some features may not be fully functional.

Install on your PC following the automatic installation instructions. There is some information about the hardware drivers that need to be followed if you are using MPLAB ICD 2, MPLAB ICE or one of the Microchip programmers, but if you do not have any of these, you can ignore this information for now. Remember, though, that if you later get one of these devices, you should install new software that came with the hardware and execute the driver installation as detailed.

Now you can start the MPLAB IDE program.

The first thing to do is to choose a PICmicro device. Almost any can do what we need for this example, but let’s choose a general purpose controller in the PIC18 family.

Choose Select Device from the Configure menu.
In the Device window, select the PIC18F452 from the list if it’s not already selected.

Note the “red” and “green lights.” These indicate which MPLAB components support this device. A green light indicates support. Sometimes you might see a yellow light. This would indicate minimal support for an upcoming part that might not yet be fully supported by the MPLAB component. Usage of components with a yellow light instead of a green light is often intended for early adopters of new parts who need quick support and understand that some operations or functions may not be available.
Next, we’ll create a project using the Project Wizard. A project is the way your files are organized to be compiled and assembled. We will use a single assembly file for this project. Choose the Project Wizard from the Project menu.
The Project Wizard will start and show this screen. Press Next> to continue.
The first step allows you to select the device, which we already did. If the Device shown is not the PIC18F452, change it on this dialog.
Step Two is used to set up the language tools suite. Select the Microchip MPASM™ Tools Suite and note that MPLAB IDE already knows the location of the assembler and the linker. You should not have to browse to set these up. If the location is not set, the default location for the assembler and linker are shown here.
Step three asks that you name the Project. Here we called it simply, “MyProject.” Then you need to establish a project directory. We choose “Browse” and made a new folder in our My Documents folder called MPLAB Project Folder.
Wizard – Select Template File

Step Four asks us to add files to the project. We do not have a source file yet, so we will use an MPLAB template file. The template files are simple files that can be used to start a project. They have the essential sections for any source file, and have information in them that will help you write and organize your code. These files are in the MPLAB IDE folder, which by default is in the Program Files folder on your PC. Scroll to the Program Files folder on Drive C:, open it, and scroll down to the MPLAB IDE folder.
Open the MPLAB IDE folder and scroll down to the Mchip_Tools directory
Open the MPLAB IDE folder and go to the Mchip_Tools directory and get a template from the Object folder in the Template folder.

The file we need is f452tmpo.asm. Use the Add>> button to add that to the list on the right side, then check the box to the left of its name to copy the file into your project directory. This leaves the original template file in the MPLAB subdirectory for use with another future project.
Next we need to add a linker script. These are in the folder named LKR in the MCHIP_Tools folder.

There is a file for each device. These files define the memory configuration and register names for the part.

These are located in the MPLAB IDE folder under Mchip_Tools named LKR. Use 18F452.LKR for this project. There is also a linker script named 18F452i.lkr, for use with this device when MPLAB ICD 2 is being used. That linker script reserves areas for memory for MPLAB ICD 2. Since we are using the simulator, we don’t need to use that linker script.
Wizard – Select Linker Script File

Template File: F452tempo.asm
Linker Script: 18f452.lkr

Now there should be two files in our project, the linker script 18F452.lkr and template file f452tempo.asm.
The last slide in the Project Wizard shows the information on this project. Click Finish to exit to the MPLAB IDE desktop.
The Project Window in MPLAB program should look like this.
From the Project menu, we can choose Build All to assemble and link the current files. They don’t have any of our code in them yet, but this just assures that the project is set up correctly.
The Output window shows the result of the build process. There should be no errors on any step.
Now we need to add our code to the source file. Open up the source file f452tmpo.asm by double clicking on its name in the project window or selecting it, then press the right mouse button to select “edit.”
The file has some comments at the beginning, and an area which you can use as a standard comment information header for the file. For now we'll leave this as it is, but if this were a real project, you could put information about your design here. Scroll down to the bottom of the file.
The code up to this point is for more advanced things such as setting up interrupts and configuration bits in a final application. We can ignore those details for now and just write the code. Our code will go into the file at the point after the symbol Main: is defined.
Looking at the data sheet for the PIC18F452, we see that I/O can be done on the set of pins called PORTB. Note that PORTA has pins labeled RA0/AN0, while PORTB has pins labeled RB0/INT0. This indicates that the pin can be used for two things. In the case of PORTA, the pin can be a general purpose I/O pin, RA0, or can be an Analog to Digital input, AN0.

PORTB can be used as RB0, simple input or output as denoted by the RB0 label, or can be used as an interrupt pin, INT0. We won’t be enabling interrupts in this example, and a look through the data sheet indicates that PORTB will operate normally with some simple configuration. PORTA, however, according to the datasheet defaults to use the analog to digital converter, so we would have to do more configuration to use PORTA for this example.
In the area beneath the label Main, we add these lines of code.

Comments are put in the code after semicolons. The assembler will ignore everything on a line after a semicolon. The code moves a value of zero into the PIC18F452 W register, then moves the contents of the W register to TRISB. TRISB is the data direction register for PORTB, determining whether each pin is used as an output or as an input. There are eight bits in TRISB and PORTB corresponding to 8 different pins on the PIC18F452.

There is no instruction to move a value directly into TRISB or PORTB, so we need to use the accumulator, called the W register in PICmicro MCUs, to load the data and move it to these registers.

Then we add a label, called Loop, which we will use later, and add lines of code to move a value of 1 into PORTB, then a 0.

At the end we put a goto Loop instruction, which causes the program to stay in this loop forever, alternately sending out 1s and 0s on pin RB0.

If we had used an alternating pattern of FF and 0 instead of 1 and 0, we would toggle all 8 pins on PORTB, not just the pin labeled RB0.

Now we can compile and test the code.
In order to test the code, we need a debug tool. MPLAB SIM is a debug tool. It is software simulator that can be used to test code on the PC.

Select MPLAB SIM from the Debugger>Select Tool pull down menu.
Reset Program

Next we select Reset from the Debugger pull down menu, and a green arrow shows us where our program will begin. This was part of the template file. The first instruction in memory jumps to the label called Main, where we put our code. This allows us to jump over the vector areas in memory.
Now we can press the Step-Into icon or choose Step-Into from the Debugger menu to single step to our code.
Continuing to press Step-into, we can step through the code.
Step Through Code

...step
Step Through Code

...step
Step Through Code

...step
Step Through Code

...step
..step and we go back to the start of the infinite loop.
Watch Window

In order to see if our code executes properly we need to see what’s happening on PORTB.

We will add PORTB to a watch window. After we press reset we end up back at the start of the program and the value on PORTB is 0.
After the instruction that sends a 1 to PORTB, the value of PORTB changes. Note that when single stepping, you are stopped at the instruction before it executes, so we didn’t see a change until we executed the MOVWF PORTB instruction in line 149.
As we step further, we see PORTB with a value of 0 at line 152. Continually following through the loop will show PORTB changing from 0 to 1 and back again.

We could program this code into a demo board with a a PIC18F452 on it. On that demo board, this pin on PORTB is connected to an LED. If we run it we’ll see the LED apparently lit all the time.

Why isn’t it blinking?

Because the loop is executing so fast, we cannot see the LED going on and off. We would need to put some delays in the code. In order to slow it down, we’d add a delay after sending a value of 1 out PORTB and also after sending a value of 0 out PORTB.
Summary: First Project

- Select Device
- Create Project
- Select Language Tools
- Put Files in Project
- Create Code
- Build Project
- Test Code with Simulator

We have done the major steps for creating, building and testing our simple project. We selected the device, the PIC18F452.

We used the Project wizard to create a project, and while using the wizard we …selected the MPLAB built-in MPASM™ and MPLINK™ language tools.

…added files for the project, a template file for the device we’ve selected and a linker script to build it properly.

We wrote some simple code to toggle an I/O bit.

Then we built the project.

And finally we tested the code with the simulator.
Summary: The Next Step

- We Covered the Basics of Building a Project
- Debug Tools:
  - Stop Watch
  - Stimulus
  - Advanced Breakpoints
- Programmers
- Hardware Debuggers

These are the essential steps for getting started with MPLAB IDE. Once you are familiar with these, you are ready to go on to explore the power of the MPLAB tool. All debugging features are covered in the MPLAB IDE documentation and on-line help.

Advanced debug tools are available in MPLAB IDE.

A Stop Watch helps you measure execution time of your code.

Stimulus can be used to apply signals to pins of the device while using the simulator.

Breakpoints can be set on complex conditions so you can run your code and halt at precise points in your program.

Programmers can be installed to burn the code into your design, and hardware debuggers, like MPLAB ICD2 and MPLAB ICE can be used to more fully test and analyze your final hardware.
If you haven’t already done it, now is the time to get started with MPLAB IDE. Simply go to our web site or contact your local Microchip Sales Office for the latest MPLAB CDROM.

For an overview of more of Microchip’s development tools, you might wish to view the web seminar titled “An Introduction to Microchip’s Development Tools.”

That is the end of our presentation. Thank you for your time.