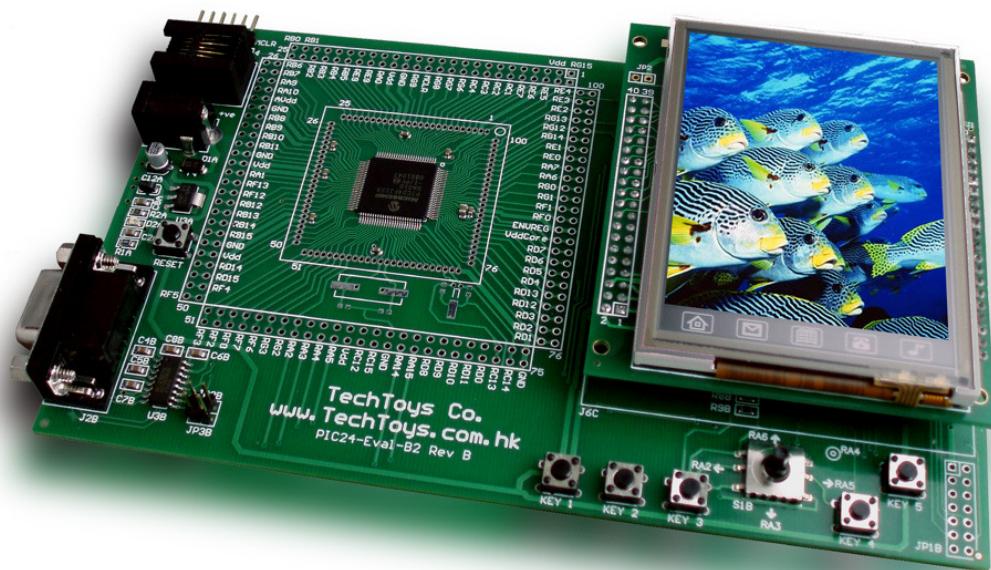


# **Development board for PIC24FJ128GA010 with 262k TFT color LCD module**



Picture shown with optional 3.2" TFT LCD with touch panel

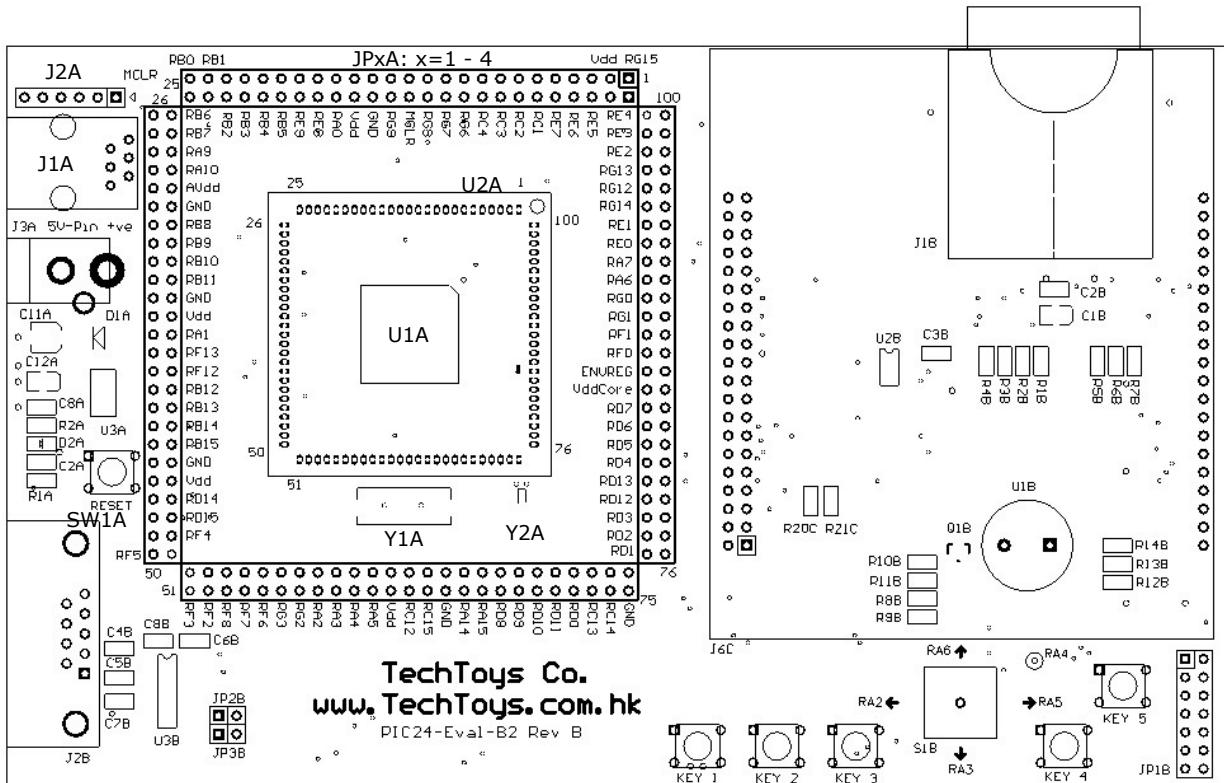
## **INTRODUCTION**

Development board for PIC24FJ128GA010 provides a low cost platform to evaluate the high pin-count 16-bit microcontrollers from Microchip. A rich set of on-chip peripherals such as Fast RC Oscillator, Real Time Clock & Calendar, flexible Output Compare, & Parallel Master Port module (just to name a few of them) differentiates PIC24-series from the previous PIC16 and PIC18 microcontroller series.

However, prototyping with PIC24 series especially for 100-pin microcontrollers PIC24FJ128GA010 / PIC24HJ128GA010 could be difficult as they are offered in 100/TQFP package only. A tiny lead pitch of merely 0.50mm of these devices makes it almost impossible for hand-solder. Our development board for PIC24FJ128GA010 offers a handy way to evaluate this 100-pin microcontroller with the following features

1. Onboard AMS1117-3.3V linear regulator for a clean and regulated 3.3V
2. ICD2 compatible socket
3. PICKit2 compatible header
4. Double row 2.54mm pin headers for 0.100" (2.54) cables or DSO/Logic Analyzer probes
5. 2.54mm pin headers compatible with two color LCD modules : 3.2" QVGA TFT LCD with touch panel and 2.4" QVGA TFT LCD modules
6. SD Card socket
7. UART interface via SIPEX SP3232
8. 5-way navigator joystick and 5 tact switches wired as a matrix keypad
9. SPI eeprom for storage of critical board information such as the calibration data for touch panel
10. Buzzer for user alert like touch panel audio beep

## ANNOTATION : BOARD LAYOUT



	Function	Designator
1	Microcontroller – PIC24FJ128GA010 in TQFP package.	U1A
2	External pads for the microcontroller in 1.27mm pitch. These pads are directly compatible with Plug-In Module (PIM) for PIC24xx128GA010 released by Microchip	U2A
3	AMS1117-3.3V linear voltage regulator supplies a stable 3.3V as Vdd for the whole board	U3A
4	2.1mm DC power supply jack. Pin positive and the shell is ground. A supply voltage of 5V recommended (max 7.5V).	J3A
5	ICSP connector directly compatible with Microchip PICKit2 programmer. Triangular mark indicates the MCLR pin.	J2A
6	ICSP connector directly compatible with Microchip ICD2 debugger and programmer.	J1A
7	RESET switch connected to MCLR pin of the microcontroller.	SW1A
8	Top solder pads for the Primary Oscillator. Only a land pattern has been provided for freedom of crystal choice. Internal Fast RC Oscillator has been used for examples in this manual therefore this land pattern has been left empty.	Y1A
9	Top solder pads for the Secondary Oscillator, typically a 32.768kHz would be used there for Real Time Clock applications.	Y2A
10	Double row 2.54mm PCB headers for strip wires, and/or DSO/Logic Analyzer probes	JP1A – JP4A
11	SD Card Socket wired to the SPI module of the microcontroller	J1B

12	Microchip 25AA080 SPI eeprom for data storage	U2B
13	Buzzer	U1B
14	SIPEX SP3232, for UART connection with PC COMPORT	U3B
15	5-way navigator joystick and tact switches	S1B SW1B- SW5B
16	2*7 2.54mm headers for extension purpose	JP1B
17	Custom-made module socket for 3.2" TFT or 2.4" TFT LCD modules.	J6C

## SOFTWARE : MPLAB C30 COMPILER

The MPLAB®C C30 compiler is a full-featured ANSI compliant C compiler for the Microchip 16-bit devices. MPLAB C30 v3.xx Student Edition is free and thus this is used for examples in this manual. Microchip is generous to offer this full-featured compiler for the first 60 days, and after 60 days only its optimization levels would be limited.

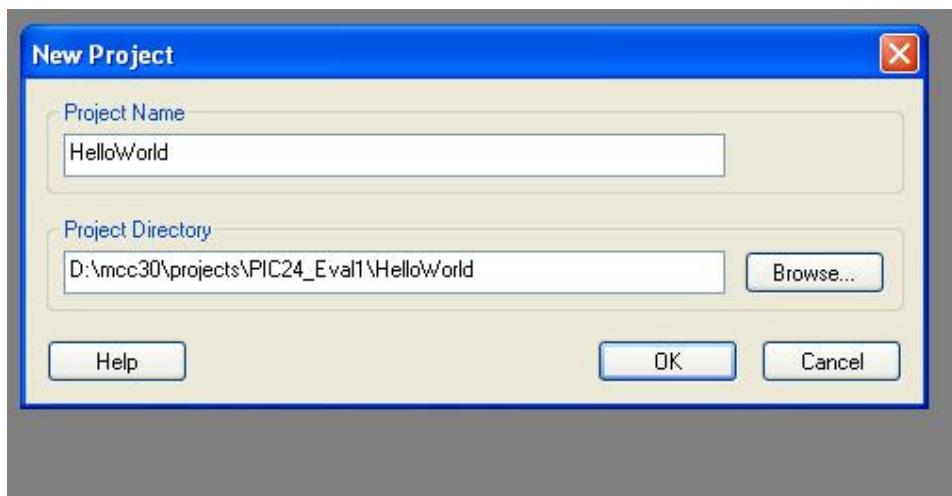
Click the link below to download and see a full description of the compiler.

[http://www.microchip.com/stellent/idcplg?IdcService=SS\\_GET\\_PAGE&nodeId=1406&docName=en010065&part=SW006012](http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&docName=en010065&part=SW006012)

If you accept the default installation path, you will get a **bin** directory under the folder **C:\Program Files\Microchip\mcc30\**. This **bin** folder contains the execute file. It is just a matter of individual preference for the installation and my path to mcc30 is simply **D:\**; therefore the bin folder is located under **D:\mcc30\** and the project folder has been set to **D:\mcc30\ projects\PIC24\_Eval1\**.

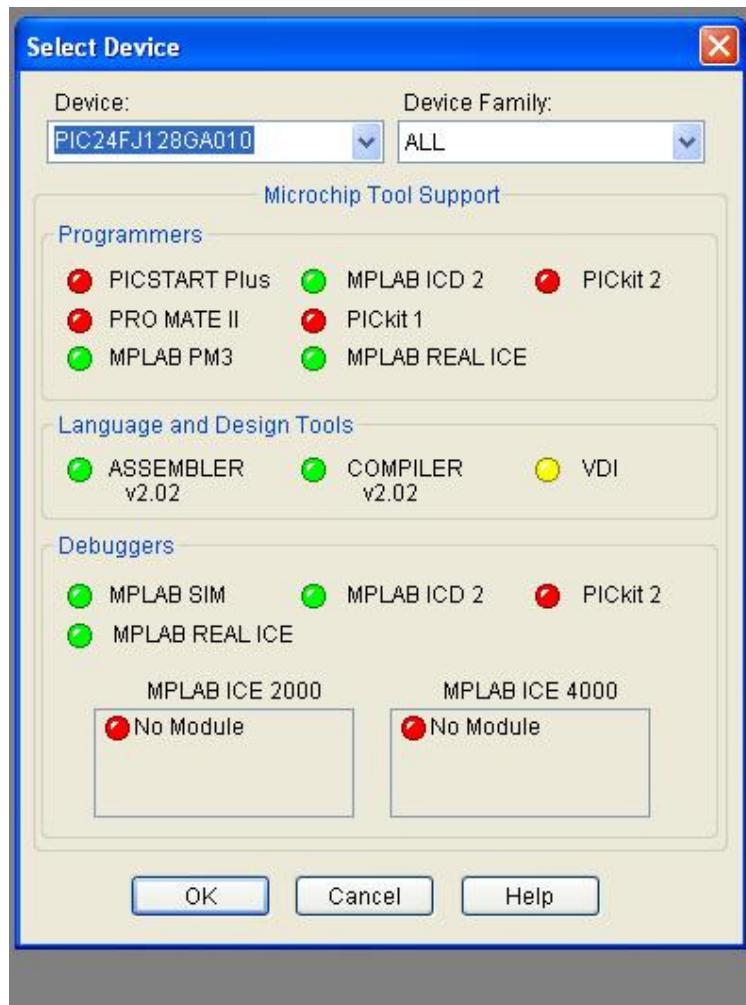
It is assumed that MPLAB version 7.60 or later has been installed in your development workstation. First create a new project under **Project→New...** in MPLAB. A New Project window will pop up with **Project Directory** and **Project Name** information that you need to fill in. Just create a directory of your convenience (my case being **D:\mcc30\projects\PIC24\_Eval1\HelloWorld**) and the first project being the **HelloWorld** as the Project Name. If you don't want to type it all over, please download the full source code under Doc 04 at the following html link.

[http://www.techtoys.com.hk/PIC\\_boards/PIC24-Eval-B2/PIC24-Eval-B2\\_RevB.htm](http://www.techtoys.com.hk/PIC_boards/PIC24-Eval-B2/PIC24-Eval-B2_RevB.htm)



From **Configure→Select**

**Device...** menu of the MPLAB, select PIC24FJ128GA010 as the microcontroller to use (or PIC24HJ128GA010 if you have ordered this part number). Click **OK**.



Click **Project→Select Language Toolsuite** to select Microchip C30 Toolsuite. Point locations of all MPLAB ASM30 Assembler (pic30-as.exe), C30 C Compiler (pic30-gcc.exe), LINK30 Object Linker (pic30-ld.exe), and LIB30 Archiver (pic30-ar.exe) to the right path of the compiler. In my case, it is D:\mcc30\bin\. Click **OK** to exit upon finish.

All we have to do now is to create a new document and type in a simple program as the HelloWorld project. Under **File**, click on **New**. A new document will be created with an empty template. Type in the following code and then save it as HelloWorld.c under the project folder.

```
#include <p24fj128ga010.h> (1)

_CONFIG1(JTAGEN_OFF & FWDTEN_OFF) (2)
_CONFIG2(FNOSC_FRCPLL & OSCIOFNC_OFF) (3)

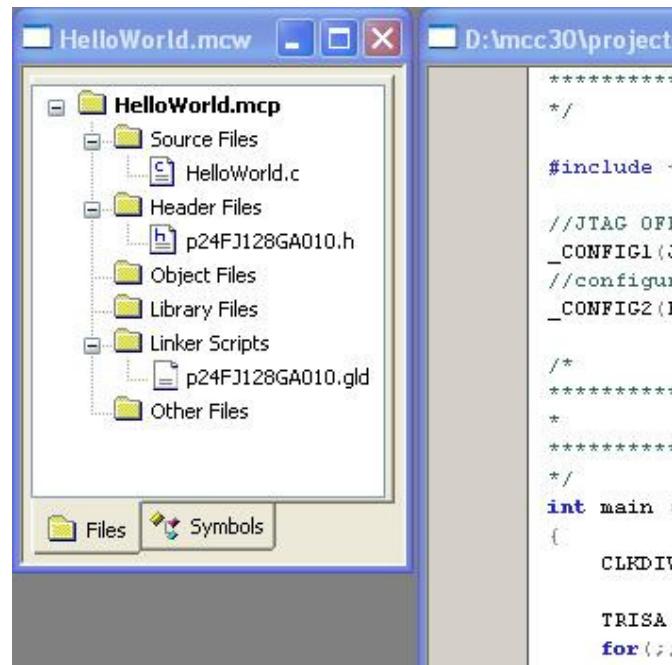
int main (void)
{
    CLKDIV = 0x0000; //FRC postscaler divided by 1 (8MHz from internal RC) (4)

    TRISA = 0; (5)

    for(;;){
        PORTA = 0xC5; (6)
        PORTA = 0x38; (7)
        PORTA = 0x47; (8)
    }

    return (0);
}
```

Right click on **Source Files** at the project workspace at the left Panel and add HelloWorld.c to the Source Files. Then we need to add the linker script p24FJ128GA010.gld (located under ..\mcc30\support\gld\). Optionally we may put p24FJ128GA010.h (under mcc30\support\h\ ) to Header Files section for reference purpose. That is all we need! Under **Project**, press **Build** to compile this simple program.



The **Output Window** shows that we have at least successfully compiled a program, even though this program will not do anything yet. Browse to the project folder you will see there is a HelloWorld.hex file created. This is the hex file we need to download to PIC24FJ128 to build an embedded system.

## PROGRAM SECTION

### - ICD2 operation

The first choice is to use an ICD2 if you have one because it allows single code stepping for program debug.

### - PICKit2 operation

PIC24-Eval-B1 board is compatible with PICKit2. Match the triangular mark on PICKit2 with the white triangular mark on board as shown below.

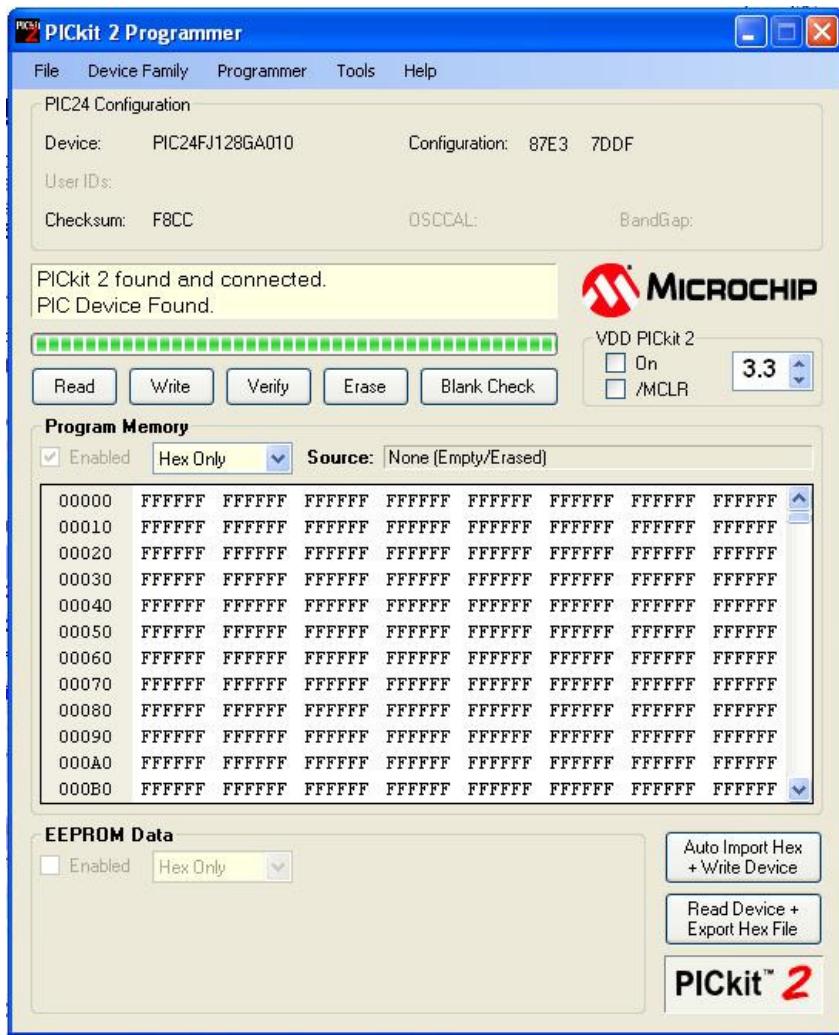


Download the latest version (it was V2.40 at the time of writing) from Microchip web site.

### Downloads

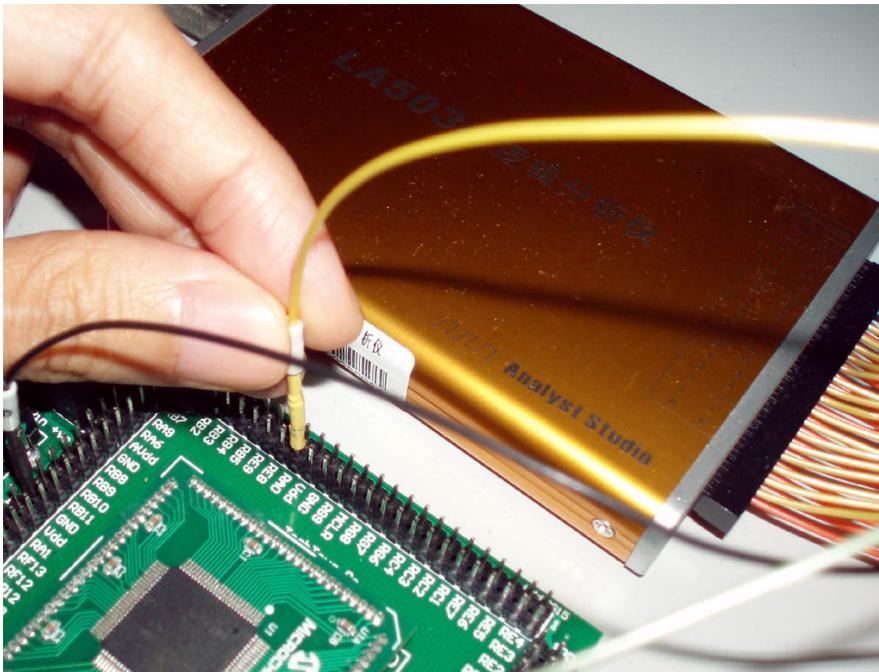
Software Firmware	Size
PICkit 2 V2.40 Install	1.99 MB
PICkit 2 V2.40 Install with .NET Framework	28.3MB
Readme for PICkit 2 V2.40	30 KB
PICkit 2 Firmware V2.10	22 KB
Device File Update V1.41.00 for PICkit 2 V2.40	17 KB
PK2CMD V1.00 PICkit 2 Command Line Interface	52 KB

Launch PICKit 2 v2.40 from the Desktop and select the correct microcontroller. After power up with a 5V-9V DC supply you will see the following screen. Under **File→Import Hex**, browse to the project folder and select HelloWorld.hex. Click on the **Write** button then you will see a successful message.



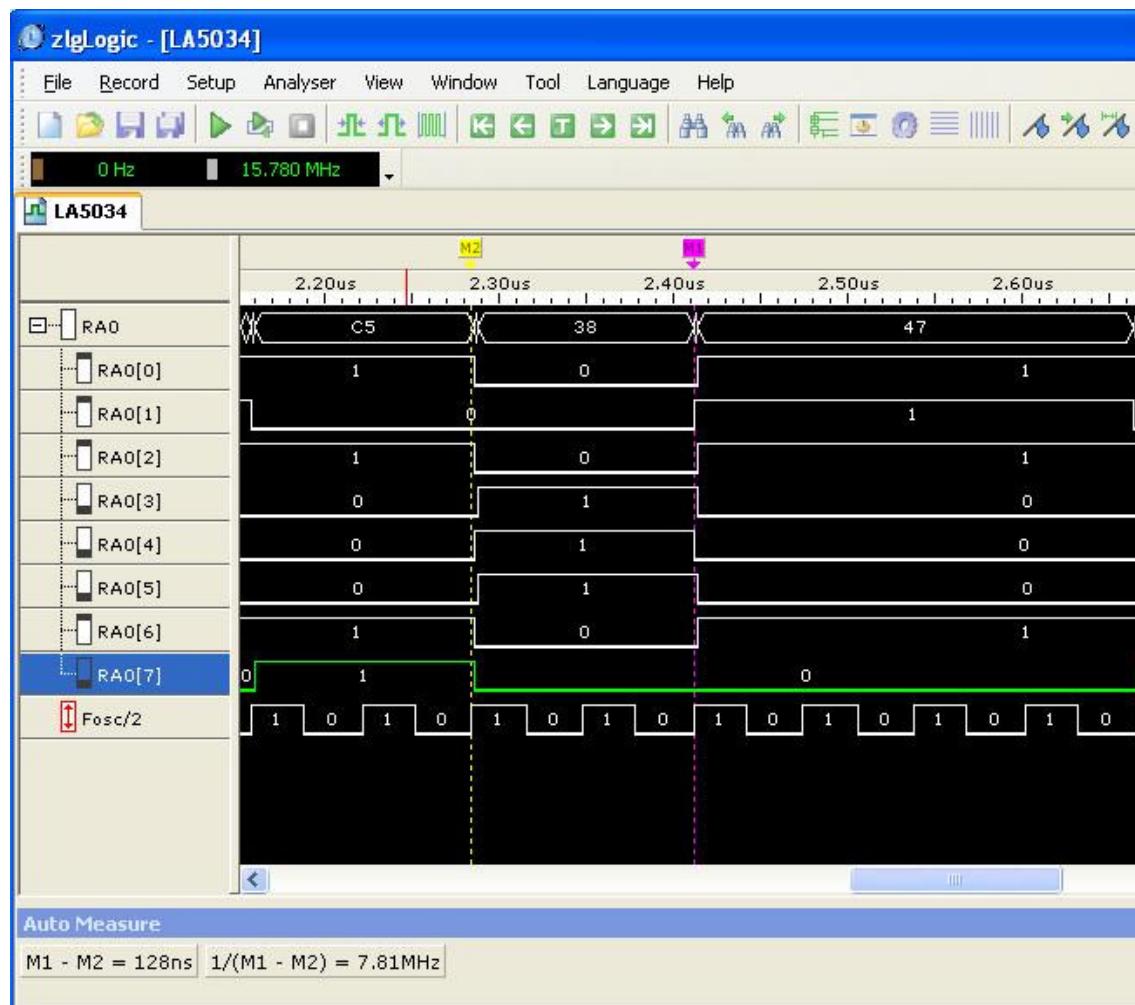
The problem is: how can we tell from such simple application if it is really working? There is no LED, no push button, nor serial port; not to say a color LCD. Here comes an interesting tool from my own collection. It is a low-cost PC-based Logic Analyzer. Picture in the next page shows my setup.

An extract from Wikipedia: A Logic Analyzer is an electronic instrument that displays signals in a digital circuit that are too fast to be observed and presents it to a user so that the user can more easily check correct operation of the digital system. They are typically used for capturing data in systems that have too many channels to be examined with an oscilloscope. Software running on the logic analyzer can convert the captured data into timing diagrams, protocol decodes, state machine traces, assembly language, or correlate assembly with source-level software. Without a Logic Analyzer, one may stick to a DSO for timing diagrams but the number of traces would be limited.



From Wikipedia data, Agilent and Tektronix make up over 95% of the industry's revenue for such device! However, offer from these two giants could be daunting to most hobbyists, students, and even individual engineers too. Mine one is only a made-in-China PC-based Logic Analyzer. This yellowish, aluminum extruded casing device make jobs done for most cases. The next page shows a screen capture of what I found out from the timing diagram.

At least it shows our first “helloworld” project is working as expected. The microcontroller is using the Fast Internal RC Oscillator as the clock source.



Here we go through the code line-by-line:

Line (1) includes the header file for PIC24FJ128GA010.

Line (2), (3), & (4) define the Configuration bit settings and clock postscaler of the microcontroller. The oscillator source that is used at a device power-on reset is selected using Configuration bit settings. The configuration word 2 is more important for us in this example. The default configuration for FNOSC2:FNOSC0 is 1:1:1 (Fast RC Oscillator with Postscaler). The postscaler value is configured in register CLKDIV, with default value RCDIV2:RCDIV0 = 0:1:1. This gives a clock speed of 1MHz (divided by 8). Therefore, if we didn't configure the FNOSC values, the clock source of the mcu would be given by FRC oscillator at 8MHz with a postscaler divided by 8, giving a Fosc value of 1MHz with cpu peripheral clock ratio set to 1:1. If we want to set a faster cpu speed without using an external crystal, we need to use the PLL (x4). This can be done by using the configuration macro \_CONFIG2(FNOSC\_FRCPLL) provided by mcc30 compiler. The result is a cpu clock speed of 4MHz (8MHz\*4 divided by the post scaler 8). The full speed of the mcu can be achieved by setting RCDIV2:RCDIV0 to 000 (divide by 1). Thus Fosc would

be 32MHz. There is an easy way to monitor the cpu clock speed by setting RC15 pin to OSCO (Fosc/2). This can be done by using the \_CONFIG2 configuration macro with an argument OSCIOFNC\_OFF. If RC15 digital IO function is required, we just use the complementary argument OSCIOFNC\_ON to get it back.

From a frequency window near the top of the user interface, a value of 15.780MHz is displayed! This is exactly the cpu clock, being Fosc/2 with Fosc set to 32MHz by internal Fast RC Oscillator!

Line (5) sets PORTA to an output

Line (6) assigns the lower byte of PORTA to 0xC5. This is displayed by the timing diagram with annotation C5 on the waveforms. Then, hex values of 0x38, and then 0x47 displayed as we step through the code.

One interesting thing to watch out is the time required to step through one code line. Two cpu cycles has been spent for 0xC5, and two cpu cycles have been spent for 0x38 as well. This is contrary to our expectation that, simple value assignment to a physical port like

```
PORTA = 0xC5
PORTA = 0x38
```

should take a single instruction for each. At least this was true for our experience with PIC16 and PIC18 series. Why this time we have a more powerful microcontroller but the speed is halved?

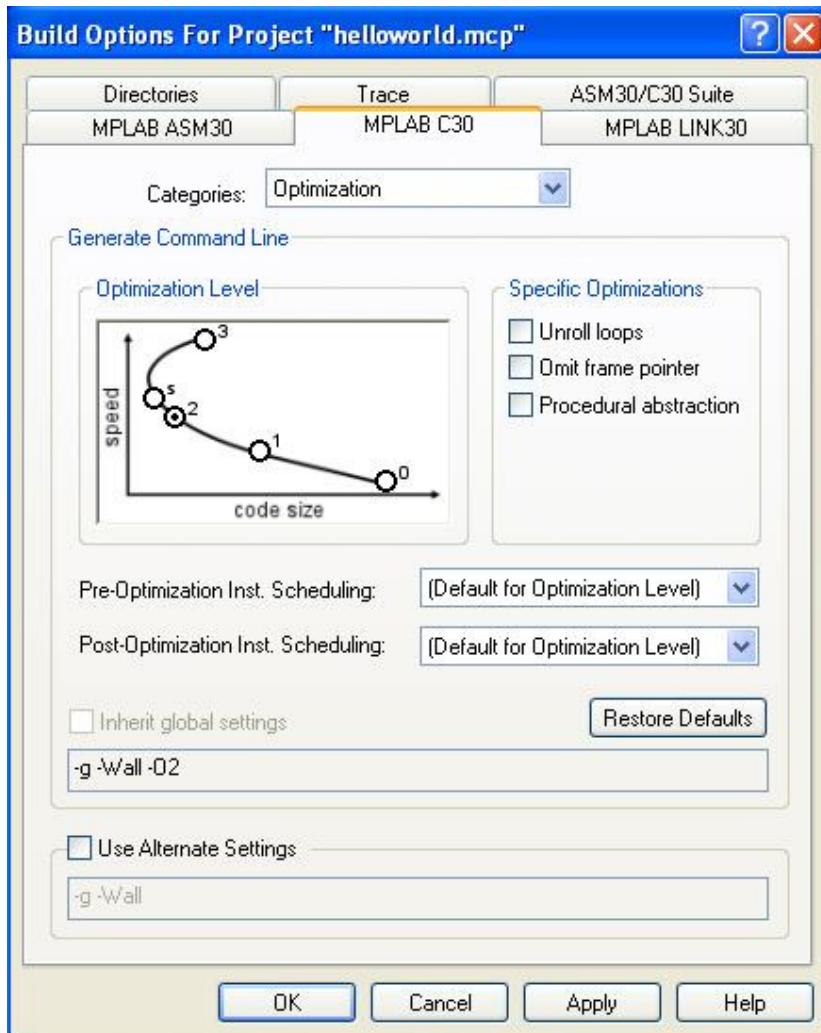
**It is the compiler option we need to take care!** Look closely at the program by disassembly listing shows the following. If we don't care about optimization and do a blind compile, the complier gave us two instructions each for all three assignments

```
PORTA = 0xC5;
PORTA = 0x38;
PORTA = 0x47;
```

As a result, there are two instruction cycles per line and hence the time it took for each code was 128ns (7.81MHz) instead of what we expected it to be a 16MHz device!

44:	*****	
45:	*/	
46:	<b>int main (void)</b>	
47:	{	
00280	FA0000	<b>lnd #0x0</b>
48:	<b>CLKDIV = 0x0000;</b>	
00282	EB0000	<b>clr.w 0x0000</b>
00284	883A20	<b>mov.w 0x0000,0x0744</b>
49:		
50:	<b>TRISA = 0;</b>	
00286	EB0000	<b>clr.w 0x0000</b>
00288	881600	<b>mov.w 0x0000,0x02c0</b>
51:	<b>for(;;){</b>	
52:	<b>PORTA = 0xC5;</b>	
0028A	200C50	<b>mov.w #0xc5,0x0000</b>
0028C	881610	<b>mov.w 0x0000,0x02c2</b>
53:	<b>PORTA = 0x38;</b>	
0028E	200380	<b>mov.w #0x38,0x0000</b>
00290	881610	<b>mov.w 0x0000,0x02c2</b>
54:	<b>PORTA = 0x47;</b>	
00292	200470	<b>mov.w #0x47,0x0000</b>
00294	881610	<b>mov.w 0x0000,0x02c2</b>
55:	}	
00296	37FFFF9	<b>bra 0x00028a</b>

Now go to the compiler option to see if there is any change upon a different compiler option is chosen.

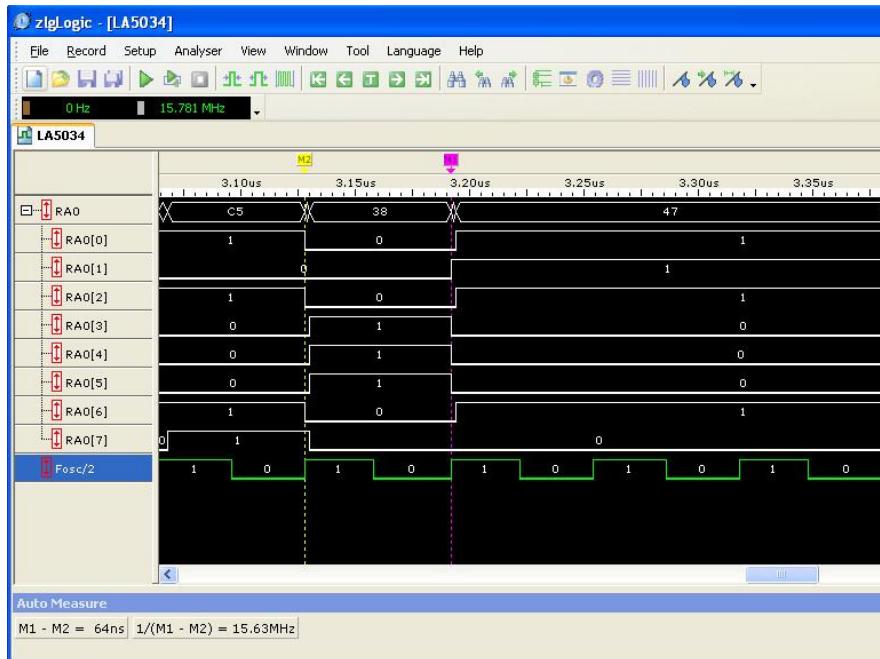


A recompile gives us the following disassembly listing. Display from the Logic analyzer showed a 16MHz device now! The time taken for a value of 0x38 at PORTA takes 64ns (15.63MHz) and so fro.

```

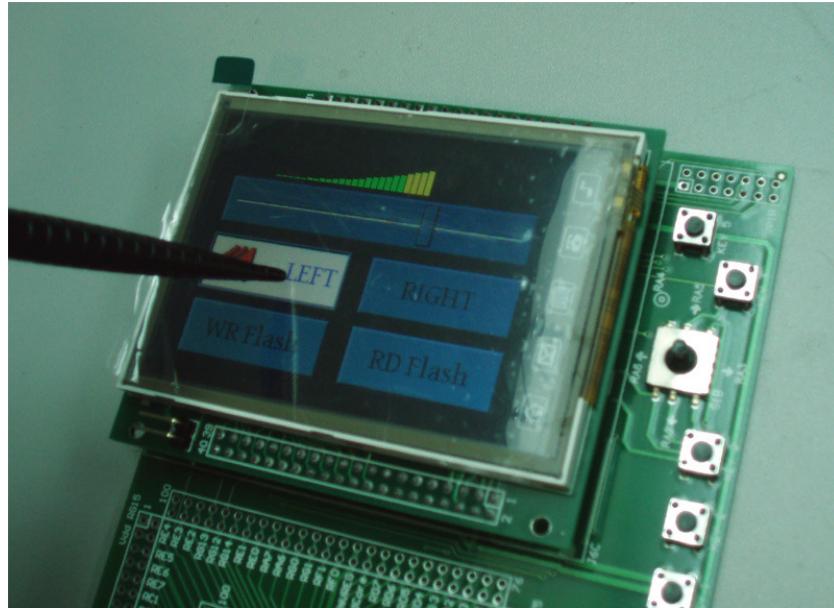
46:           int main (void)
47:           {
48:               CLKDIV = 0x0000;
00280  EB0000    clr.w 0x0000
00282  883A20    mov.w 0x0000,0x0744
49:
50:               TRISA = 0;
00284  881600    mov.w 0x0000,0x02c0
00286  200C52    mov.w #0xc5,0x0004
00288  200381    mov.w #0x38,0x0002
0028A  200470    mov.w #0x47,0x0000
51:               for(;;){
52:                   PORTA = 0xC5;
0028C  881612    mov.w 0x0004,0x02c2
53:                   PORTA = 0x38;
0028E  881611    mov.w 0x0002,0x02c2
54:                   PORTA = 0x47;
00290  881610    mov.w 0x0000,0x02c2
00292  37FFFC    bra 0x00028c

```



A simple “helloworld” project makes me know a bit better on this new PIC24 device. This debugging technique applies for other complex programs such as a low-level driver for color LCD displays with PMP module, which is one of the new features of PIC24 devices. Below please find a picture of my setup for 3.2” TFT color LCD module. The full source code has been released on our web site at the following html link under Doc 09.

[http://www.techtoys.com.hk/PIC\\_boards/PIC24-Eval-B2/PIC24-Eval-B2\\_RevB.htm](http://www.techtoys.com.hk/PIC_boards/PIC24-Eval-B2/PIC24-Eval-B2_RevB.htm)



Certainly there are more applications on 5-way navigator joystick, SD card for data storage, NAND Flash for picture storage, etc. All these examples will be illustrated in separate application notes in a near future.