การออกแบบชุดควบคุมดีซีมอเตอร์แบบไร้แปรงถ่านระบบดิจิตอลโดยใช้ดิจิตอลซิกเนลคอลโทรลเลอร์

Implement Digital Signal Controller(DSPIC30F2010)

for brushless dc motor controller

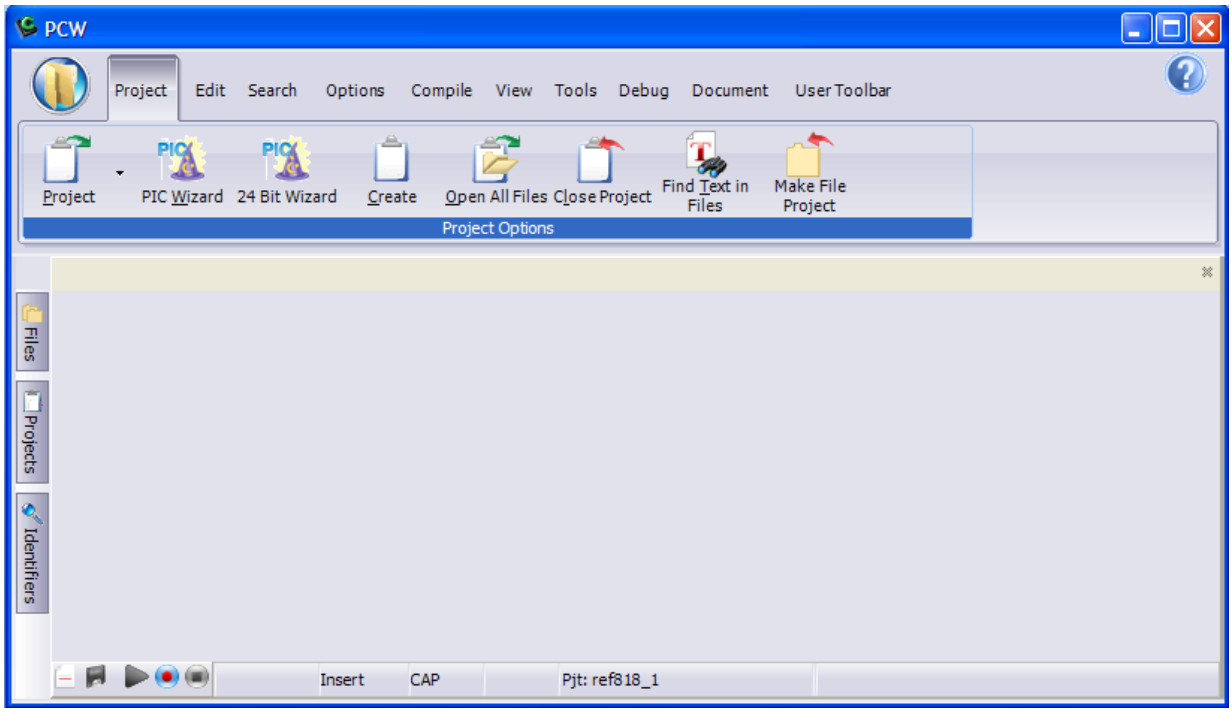
การพัฒนานวัตกรรมภายโลกของการแข่งขันอย่างเสรีถือเป็นโจทย์สำคัญที่นักประดิษฐ์ต้องให้ความสำคัญในการคิดค้นนวัตกรรมเพื่อผลิตเป็นสินค้าที่สามารถในการสร้างส่วนแบ่งในตลาดซึ่งมีการแข่งขันกันด้านราคา ความทันสมัย และการตอบสนองความต้องการของผู้ใช้ ซึ่งเทคโนโลยีที่จะทำให้การพัฒนาวัตกรรมให้ได้ตามเงื่อนไขดังกล่าวข้างต้นที่ดีที่สุดในยุกต์ปัจจุบัน คือ การนำเทคโนโลยีไมโครคอนโทรลเลอร์ หรือ ไมโครคอมพิวเตอร์ มาเป็นตัวกำกับการทำงานให้เป็นไปตามความต้องการของผู้ผลิต ซึ่งมีการให้คำจำกัดความกับสิ่งประดิษฐ์เหล่านี้ ว่า เทคโนโลยี “ระบบสมองกลฝังตัว “(Embedded System) ปัจจุบันวัตกรรมและสินค้าทางเทคโนโลยีแทบทุกชนิด ล้วนแล้วแต่ใช้ระบบดังกล่าวทั้งสิ้น การสร้างนวัตกรรมที่สามารถฝังแนวความคิดของมนุษย์ลงไปได้โดยนวัตกรรมดังกล่าวจะทำงานตามแนวความคิดที่มนุษย์ได้ฝังลงไปซึ่งนับว่าเป็นความเจริญก้าวหน้าทางวิทยาศาสตร์เป็นอย่างมาก ที่สำคัญเทคโนโลยี สมองกลฝังตัว มีความยืดหยุ่นทั้งการนำไปประยุกต์ใช้งาน และ การปรับปรุงแก้ไขระบบ ที่สำคัญเหนือสิ่งอื่นใดก็คือ คู่แข่งขันลอกเลียนแบบนวัตกรรมได้ยากยิ่ง

ในบทนี้จะเป็นการนำเทคโนโลยีที่ได้กล่าวมาข้างต้นเป็นสมองกลในการควบคุมการทำงานของมอเตอร์ดีซีแบบไร้แปรงถ่าน ซึ่งผู้เขียนจะขอเรียงลำดับการเรียนรู้และการใช้เครื่องมือเพื่อนำไปสู่การสร้างชุดควบคุมมอเตอร์ดีซีแบบไร้แปรงถ่านแบบสมองกลฝังตัว ออกเป็น ขั้นตอนดังนี้

1 การใช้โปรแกรม PIC C COMPILER , MPLAB IDE ,PROTEUS ในการพัฒนาระบบสมองกลโดยใช้ DSPIC 30F2010 เป็นตัวประเมินผลกลาง(Central Processing Unit CPU) และ พื้นฐานการพัฒนาโปรแกรม

2 การออกแบบวงจรและการพัฒนาชุดคำสั่งในการควบคุมการทำงานของมอเตอร์ดีซีแบบไร้แปรงถ่าน

1.1 การใช้ PIC _ C COMPILER V 4.084



รูป 3.1 USER INTERFACE OF PIC C V4.08

PIC_C COMPILER V4.08 ได้เพิ่มความสามารถและสิ่งอำนวยความสะดวกให้กับผู้ใช้งานอย่างมาก โดยสามารถใช้งานได้กับ pic microcontroller ได้ตั้งแต่ PIC12 , PIC16 PIC18 , PIC24 ไปจนถึง DSPIC และในส่วนของ user interface ก็หน้าตาดีมีการแบ่งออกเป็นหมวด หมู่ ทำให้ง่ายต่อการใช้งานเป็นอย่างมาก ในบทนี้จะขอแนะนำผู้เริ่มต้นใช้ PIC_C COMPILER V4.08 พอสังเขบ พอที่จะค้นคว้าได้ด้วยตัวเอง จากห้องสมุด(help)ที่ติดมากับโปรแกรมอย่างละเอียดดีมาก ๆ

จากรูปที่ 1.1 แสดงถึงการแบ่งกลุ่มเมนูเป็นกลุ่มดังนี้

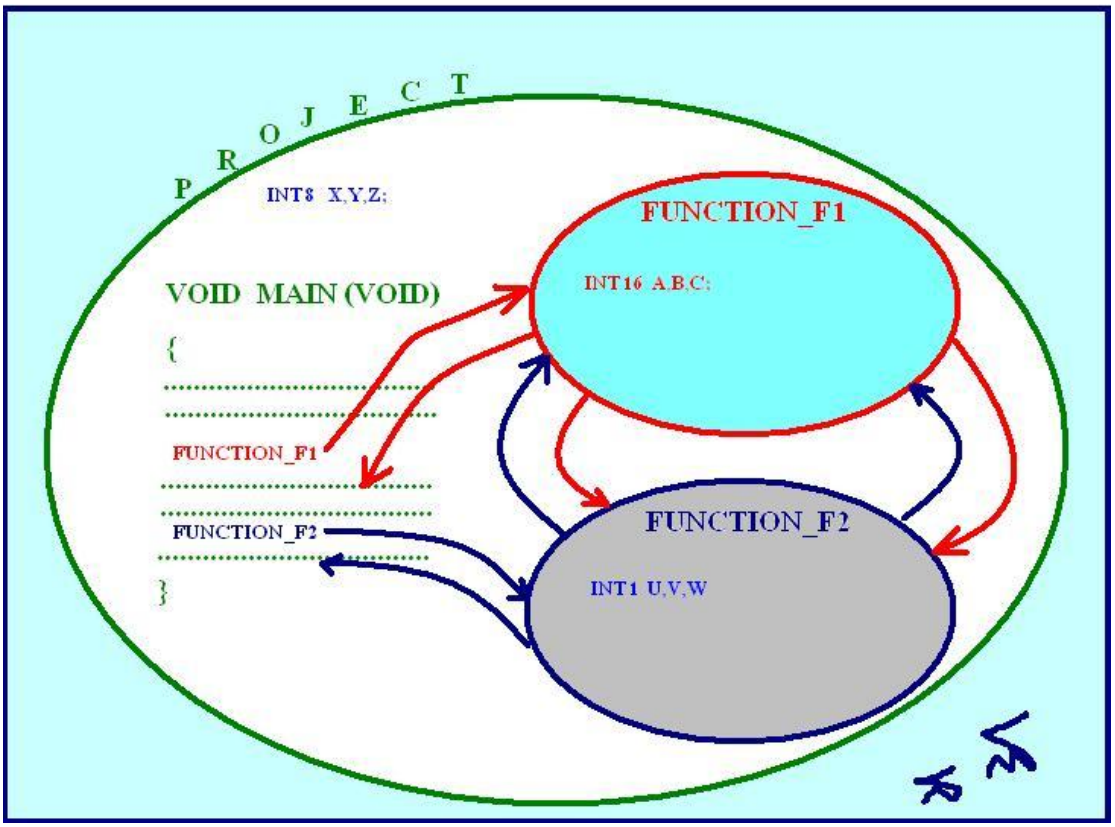
1) TAB project เป็นเมนูเริ่มต้นในการสร้างงาน หน้าที่หลักของเมนู project คือนำ source file (*.c) ซึ่งในแต่ละโปรเจ็กอาจจะประกอบด้วยหลาย source file แต่ต้องมี 1 source file ที่มีฟังชั่น main() อยู่ใน project เนื่องจากงานที่สร้างขึ้นทั้งหมดจะถูกบัญชาการด้วยฟังชั่นmain()นั่นเอง
2) TAB Edit ใช้ในการแก้ไขปรับปรุงการเขียนโปรแกรมเป็นหลัก
3) TAB Search ใช้อำนวยความสะดวกในการค้นหาและ แทนที่คำใน project
4) ใช้ในการกำหนดคุณลักษณะต่าง ๆ ของโปรเจ็ก เช่น Editor Tollbar และ Project เป็นต้น

5) Tab Compile ใช้ในการตรวจสอบไวยากรณ์และแปรงโปรเจ็กเป็น HEX File

6) Tab view ใช้ในการเยี่ยมชมองค์ประกอบต่างต่าง ๆ ของโปรเจ็ก เช่น datasheet cpu , CPU Register การใช้ Preprocesser ต่าง ๆ เป็นต้น

ในส่วน Tab อื่น ๆ จะนำมากล่าวถึงในโอกาศต่อไป

โครงสร้าง pic_c compiler



รูป 3.2 structure pic c compiler

จากรูปที่ 1.2 พอจะอธิบายได้ว่าใน project ใด ๆ จะมีฟังชั่น main() เป็นฟังชั่นหลักในการบริหา ร จัดการระบบงานใน project ทั้งหมด นั่นคือว่า sub function ต่าง ๆ ที่อยู่ใน project จะถูกเรียกใช้งานโดย function main() และระหว่าง sub function ก็สามารถเรียกใช้ซึ่งกันและกันได้ ในส่วนของตัวแปรถ้าประกาศไว้นอกฟังชั่นให้ถือว่าเป็นตัวแปรแบบ global (ยึดพื้นที่หน่วยความจำอย่างถาวร) ส่วนตัวแปรที่ประกาศในฟังชั่นจะเป็นตัวแปรแบบ private (ใช้พื้นที่หน่วยความจำเมื่อมีการเรียกใช้ฟังชั่นเท่านั้น) การพัฒนาโปรแกรมบน microcontroller การบริหารจัดการเรื่องของการใช้ memory (RAM)ถือเป็นสิ่งที่ต้องให้ความสำคัญมาก ๆ เนื่องจาก memory ใน microcontroller มีอยู่อย่างจำกัด

Pre-Processor

Pre-processor ใช้สำหรับเตรียมความพร้อมให้กับ compiler ให้ทำความรู้จักกับองค์ประกอบต่าง ๆ ของโปรเจ็กอาธิเช่น cpu (#include<30F2010.h>) การกำหนดฟังชั่นการทำงานให้กับ cpu(#Fuse NOWDT, NOPUT, PROTECT)และ กำหนด clock speed ให้กับ cpu(#use delay(clock=20000000)) เป็นต้น

Data Definition

การพัฒนาโปรแกรมในเรื่องใด ๆ ก็ตามการประกาศตัวแปรปีนสิ่งที่หลีกเลี่ยงได้ยาก เนื่องจากงานด้านพัฒนาโปรแกรมเป็นการนำอินพุทข้อมูลที่มีการเปลี่ยนแปรงตามสภาวะต่างมาประเมินผลเพื่อให้ได้ค่าเอาท์พุทตามที่เราต้องการนั่นเอง

int1	1 bit number	0 to 1	N/A
int8	8 bit number	0 to 255	-128 to 127
int16	16 bit number	0 to 65535	-32768 to 32767
int32	32 bit number	0 to 4294967295	-2147483648 to 2147483647
float32	32 bit float	-1.5×10^{45} to 3.4×10^{38}	

Constance Data

A =123;	ฐานสิบ
B = O123;	ฐานแปด
C = 0X0F;	ฐานสิหก

D = 0B00110011; ฐานสอง

Function Definition

ระเบียบว่าด้วยการสร้าง sub function มีดังนี้

- ประกาศฟังชั่น prototype ให้ compiler ทราบทุกครั้งก่อนการเรียกใช้ฟังชั่น
- กำหนดรูปแบบการรับและส่งค่าของฟังชั่นให้ถูกต้อง

ตัวอย่างโปรแกรมที่ประกอบด้วยฟังชั่นย่อยแบบต่าง ๆ

```
#include <30F2010.h>
#device adc=8
#Fuses  HS
#Fuses  BROWNOUT
#Fuses  NOMCLR
#Fuses  PUT4
#Fuses  NOWDT
#use delay(clock=10000000,restart_wdt)
#use rs232(UART1,baud=19200,parity=N,bits=8,)
void f1(void);
void f2(int8 data);
int16 f3(int8 a,int8 b);

void main()
{
   int16 x;

     while(true)
   {
     f1();
     f2(10);
     f3(10,20);
   }
}
void f1(void)
{
  output_b(0xff);
```

Function prototype

Call Function

```
}
void f2(int8 data)
{
 output_b(data);
}
int16 f3(int8 a,int8 b)
{
  return a+b;
}
```

Operators Table

+	Addition Operator
+=	Addition assignment operator, x+=y, is the same as x=x+y
&=	Bitwise and assignment operator, x&=y, is the same as x=x&y
&	Address operator
&	Bitwise and operator
^=	Bitwise exclusive or assignment operator, x^=y, is the same as x=x^y
^	Bitwise exclusive or operator
l=	Bitwise inclusive or assignment operator, xl=y, is the same as x=xly
l	Bitwise inclusive or operator
?:	Conditional Expression operator
- -	Decrement
/=	Division assignment operator, x/=y, is the same as x=x/y
/	Division operator
==	Equality
>	Greater than operator
>=	Greater than or equal to operator
++	Increment
*	Indirection operator
!=	Inequality
<<=	Left shift assignment operator, x<<=y, is the same as x=x<<y
<	Less than operator
<<	Left Shift operator

<=	Less than or equal to operator
&&	Logical AND operator
!	Logical negation operator
ll	Logical OR operator
%=	Modules assignment operator x%=y, is the same as x=x%y
%	Modules operator
*=	Multiplication assignment operator, x*=y, is the same as x=x*y
*	Multiplication operator
~	One's complement operator
>>=	Right shift assignment, x>>=y, is the same as x=x>>y
>>	Right shift operator
->	Structure Pointer operation
-=	Subtraction assignment operator
-	Subtraction operator
sizeof	Determines size in bytes of operand

Statements Table

if (expr) stmt;
[else stmt;]
	if (x==25)
	x=1;
	else
	x=x+1;

while (expr) stmt;	while (get_rtcc()!=0) putc('n');

do stmt while (expr);	do {
	putc(c=getc());
	} while (c!=0);

for (expr1;expr2;expr3) stmt;	for (i=1;i<=10;++i) printf("%u\r\n",i);

switch (expr) {
case cexpr: stmt; //one or more
case [default:stmt]
	switch (cmd) {
	case 0: printf("cmd 0");
	break;

... }	case 1: printf(“cmd 1”); break; default: printf(“bad cmd”); break; }
return [expr];	return (5);
goto label;	goto loop;
label: stmt;	loop: I++;
break;	break;
continue;	continue;
expr;	i=1;
;	;
{[stmt]} Zero or more	

BUILT-IN-FUNCTIONS

Built in function นับได้ว่าเป็นสิ่งที่ช่วยอำนวยความสะดวกกับผู้ใช้โปรแกรม pic c compiler เป็นอย่างมากโดยที่ผู้ใช้แค่ทำความเข้าใจกับการส่งผ่านค่าตัวแปรและ การ return ค่าของฟังชั่น เท่านั้น

ตัวอย่างที่ 1

Syntax: bit_set(*var*, *bit*)

Parameters: *var* may be a 8,16 or 32 bit variable (any lvalue)

bit is a number 0-31 representing a bit number, 0 is the least significant bit.

Returns: Undefined

Function: Sets the specified bit (0-7, 0-15 or 0-31) in the given variable. The least significant bit is 0. This function is the similar to: var |= (1<<bit);

Availability:	All devices
Requires:	Nothing

Examples:	int x;
	x=5;
	bit_set(x,3);
	// x is now 13

ตัวอย่างที่ 2

Syntax:	value = read_adc ([*mode*])
Parameters:	*mode* is an optional parameter. If used the values may be: ADC_START_AND_READ (continually takes readings, this is the default) ADC_START_ONLY (starts the conversion and returns) ADC_READ_ONLY (reads last conversion result)
Returns:	Either a 8 or 16 bit int depending on #DEVICE ADC= directive.
Function:	This function will read the digital value from the analog to digital converter. Calls to setup_adc(), setup_adc_ports() and set_adc_channel() should be made sometime before this function is called. The range of the return value depends on number of bits in the chips A/D converter and the setting in the #DEVICE

ADC= directive as follows:

#DEVICE	8 bit	10 bit	11 bit	12 bit	16 bit
ADC=8	00-FF	00-FF	00-FF	00-FF	00-FF
ADC=10	x	0-3FF	x	0-3FF	x
ADC=11	x	x	0-7FF	x	x
ADC=16	0-FF00	0-FFC0	0-FFEO	0-FFF0	0-FFFF

Note: x is not defined

Availability: This function is only available on devices with A/D hardware.

Requires: Pin constants are defined in the devices .h file.

Examples:

```
setup_adc(  ADC_CLOCK_INTERNAL  );
setup_adc_ports( ALL_ANALOG );
set_adc_channel(1);
while ( input(PIN_B0) ) {
   delay_ms( 5000 );
   value = read_adc();
   printf("A/D value = %2x\n\r", value);
}
 read_adc(ADC_START_ONLY);
sleep();
value=read_adc(ADC_READ_ONLY);
```

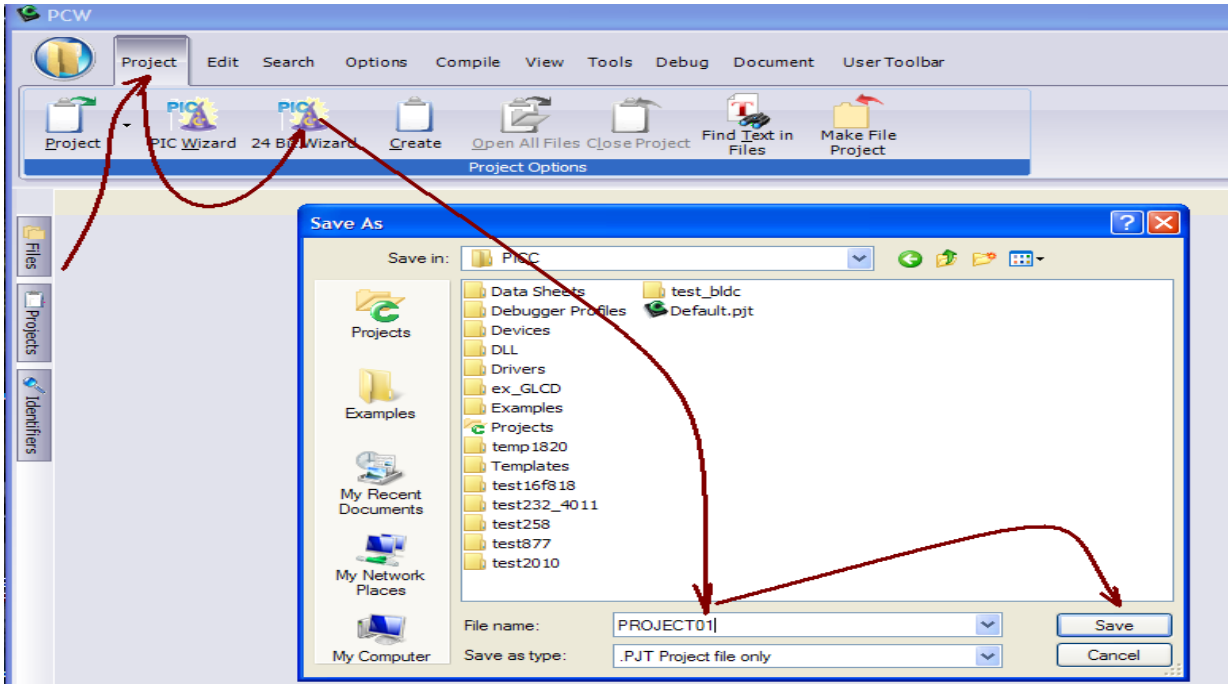
การสร้าง Project กับ pic_c compiler มีทางเลือกได้ 2 ทางดังนี้คือ

1) ใช้ Project Wizard
2) ใช้ Manual create Project

Project Wizard นับได้ว่ามีประโยชน์ต่อผู้เริ่มต้นใช้โปรแกรม เป็นอย่างมาก เนื่องจาก pic microcontroller มี register ค่อนข้างมากทำให้ยากต่อการ config ในการนำไปใช้งานในแต่ละอย่าง และ project wizard จะช่วยนำทางมือใหม่สู่เป้าหมายได้

ขั้นตอนการใช้ **Project Wizrad**

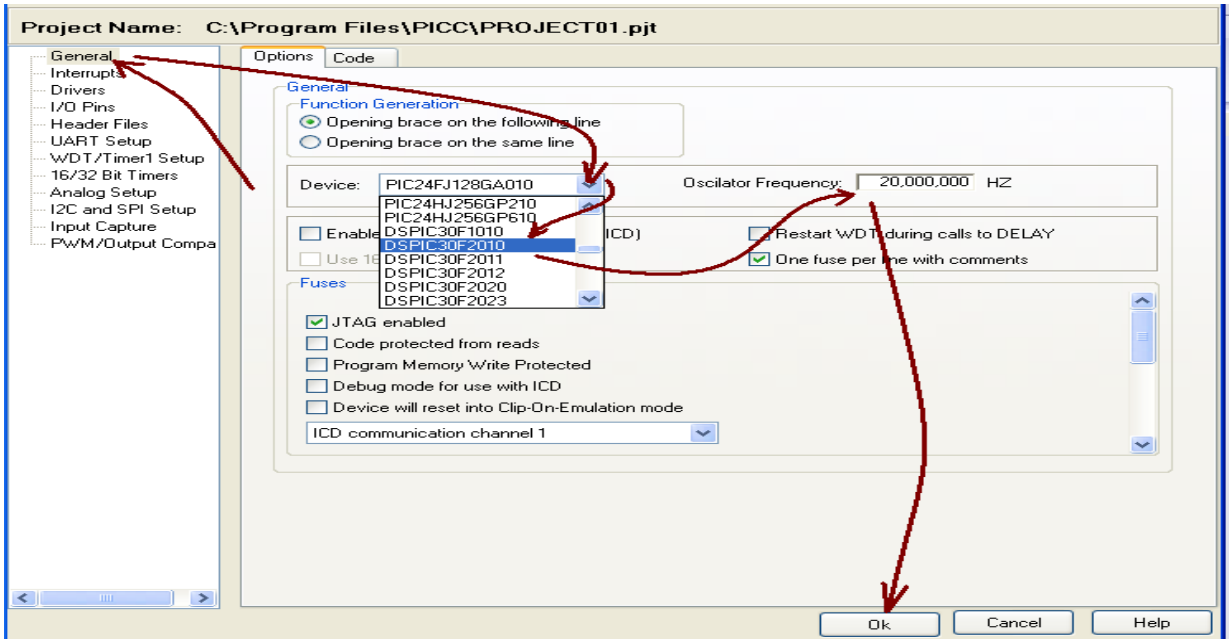
Step 1



รูปที่ 3.3 เริ่ม project wizard

- เลือก Tab Project → Pic Wizard สำหรับ(8 bit databus) or 24 bit Wizard(16 bit data bus) → project name → save

Step 2



รูปที่ 3.4 register config

Pic c compiler จะแบ่งกลุ่มส่วนที่ผู้ใช้ต้องกำหนดค่าเริ่มต้นให้กับ compilerไว้ในlist box ทางด้านซ้ายมือ เช่น ส่ วนของ general,Interrupt,Driver,I/O Pin เป็นต้น ซึ่งการกำหนดค่าในแต่ละส่วนนั้นจำเป็นต้องทำความเข้าใจโครงสร้างและฟังชั่นการทำงานอย่างละเอียด หลังจากที่เราเข้าไปกำหนดส่วนต่าง ๆ เรียบร้อยแล้ว click ok โปรแกรมจะสร้าง source code ส่วนหนึ่งมาใส่ไว้ในฟังชั่นหลักเ พื่อกำหนดการทำงานเริ่มต้นให้กับ CPU เพื่อพร้อมที่จะใช้งานตามความต้องการต่อไป

ตัวอย่าง code ได้จากการ wizard

```
//  #include "C:\Program Files\PICC\PROJECT01.h"
```

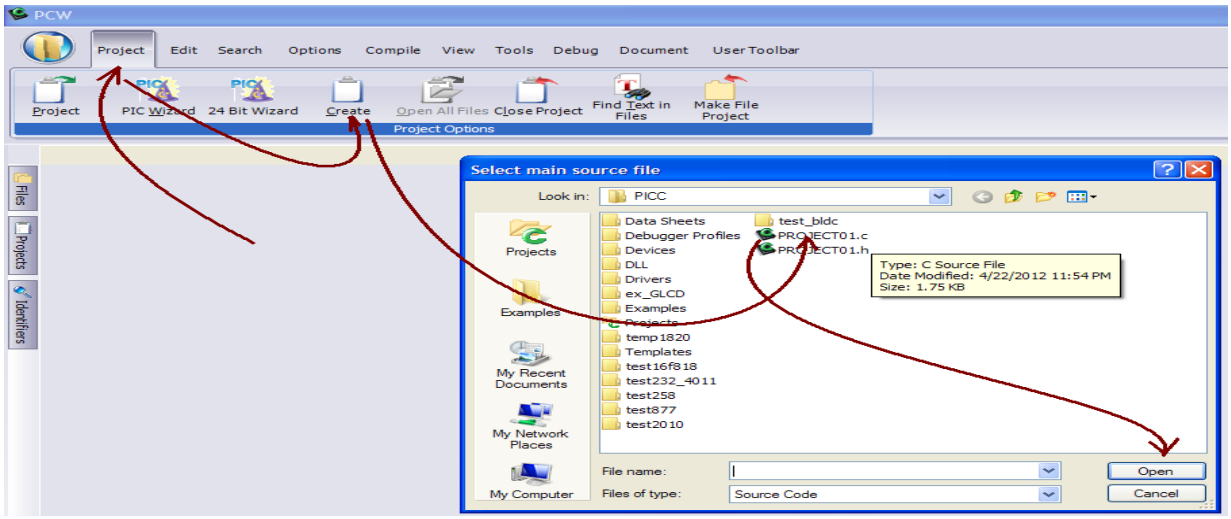
```
#include <30F2010.h>
#FUSES NOWDT                    //No Watch Dog Timer
#FUSES HS                       //High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)
#FUSES PR                       //Primary Oscillator
#FUSES NOCKSFSM                 //Clock Switching is disabled, fail Safe clock monitor is disabled
#FUSES WPSB16                   //Watch Dog Timer PreScalar B 1:16
#FUSES WPSA512                  //Watch Dog Timer PreScalar A 1:512
#FUSES PUT64                    //Power On Reset Timer value 64ms
#FUSES NOBROWNOUT               //No brownout reset
#FUSES BORV47                   //Brownout reset at 4.7V
#FUSES LPOL_HIGH                //Low-Side Transistors Polarity is Active-High (PWM 0,2,4 and 6)
  //PWM module low side output pins have active high output polar
#FUSES HPOL_HIGH                //High-Side Transistors Polarity is Active-High (PWM 1,3,5 and 7)
  //PWM module high side output pins have active high output polarity
#FUSES NOPWMPIN                 //PWM outputs drive active state upon Reset
#FUSES MCLR                     //Master Clear pin enabled
#FUSES NOPROTECT                //Code not protected from reading
#FUSES NOCOE                    //Device will reset into operational mode
#FUSES ICS0                     //ICD communication channel 0
#FUSES RESERVED                 //Used to set the reserved FUSE bits
#use delay(clock=10000000)
#use rs232(UART1,baud=9600,parity=N,bits=8)
```

```
void main()
{
  setup_spi(SPI_SS_DISABLED);
  setup_wdt(WDT_ON);
  setup_timer1(TMR_DISABLED|TMR_DIV_BY_1)
  // TODO: USER CODE!!
}
```

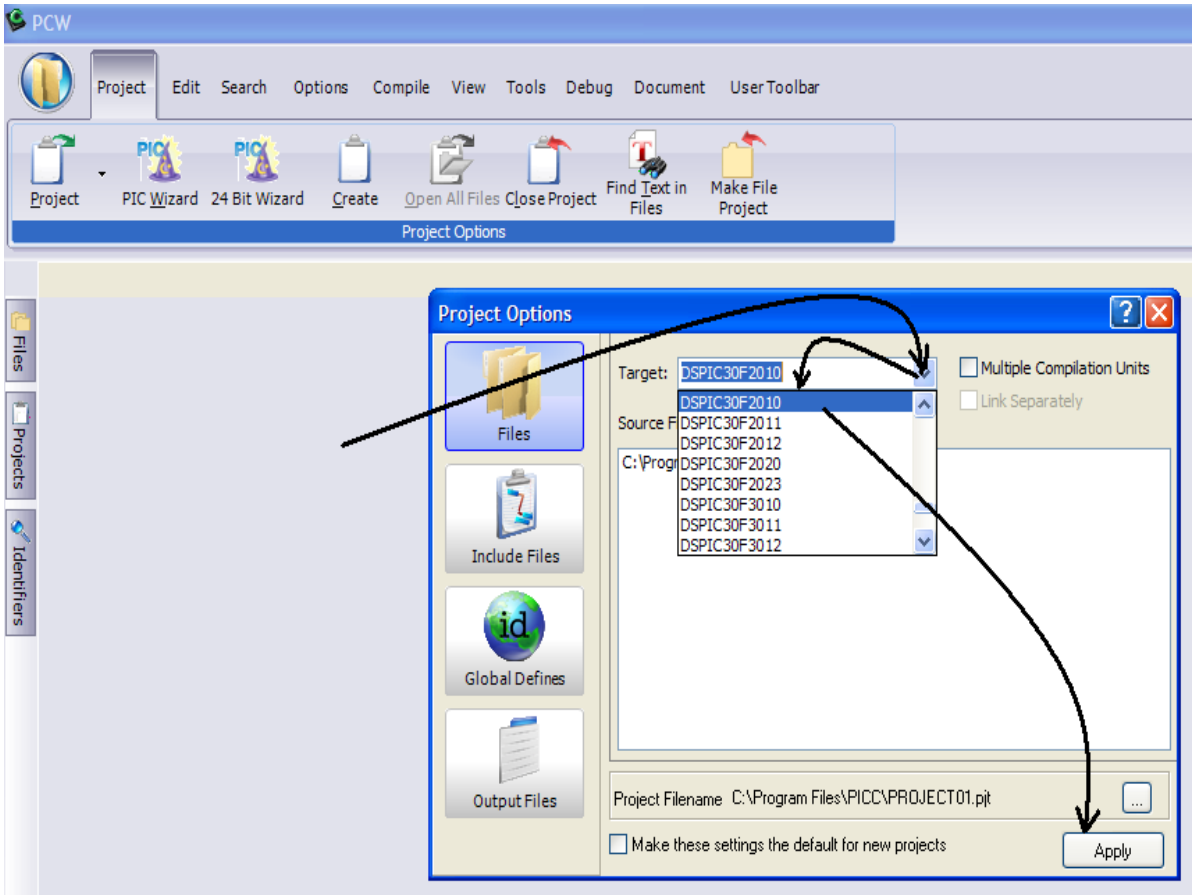
Main() function

Manual Create Project

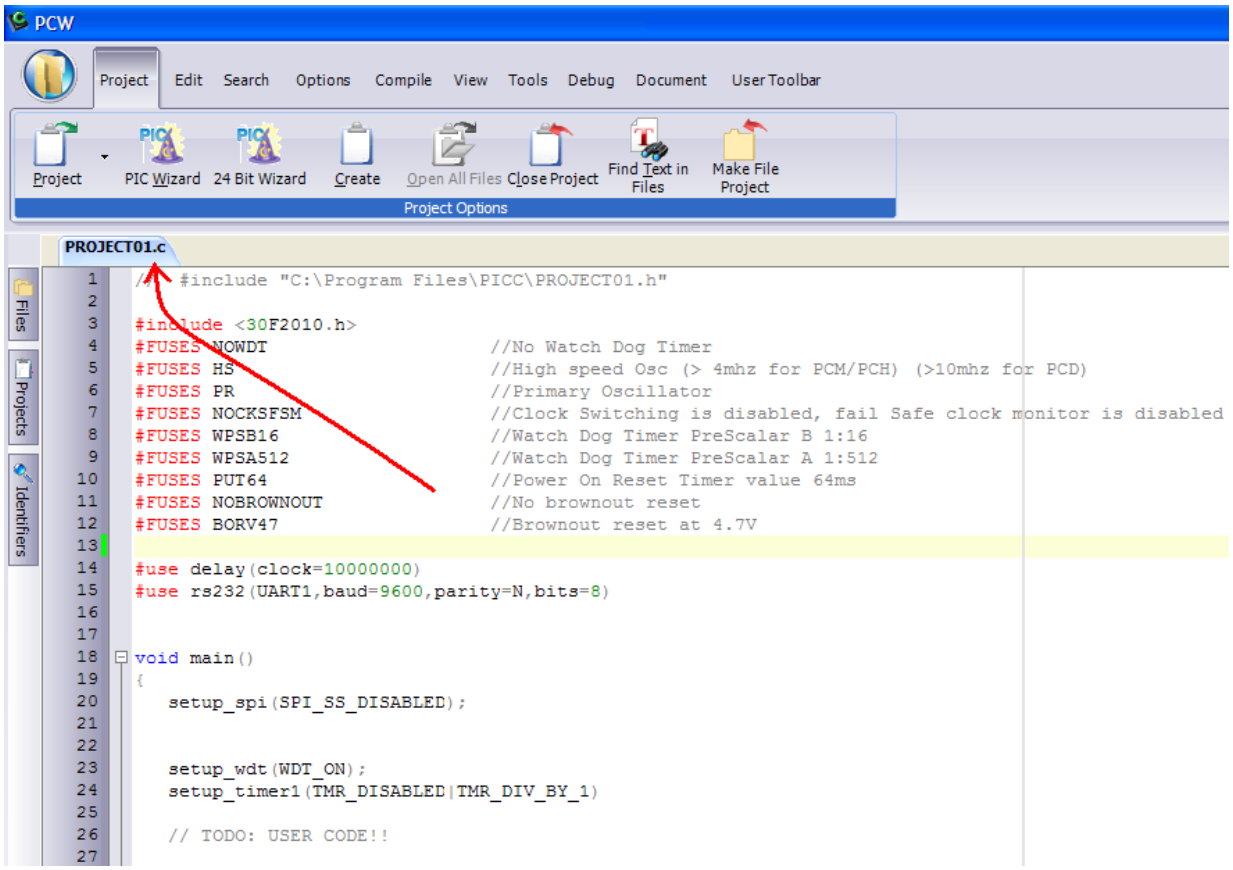
เป็นวิธีการนำ source file ที่ถูกสร้างขึ้นมาแล้ว มาประกอบรวมเป็น project อย่างไรก็ตามจะต้องมี function main() อยู่ในsource file ด้วยเนื่องจากกฎเกณฑ์ของภาษาซีที่ project ใด ๆ ก็ตามจะต้องใช้ function main()ในการบริหารจัดการกับ project ทั้งหมด ขั้นตอนการ manual create project แสดงด้วยรูปภาพเป็นลำดับดังนี้



รูปที่ 3.5 step 1 เส้นทางสู่ การเลือก source file



รูป 3.6 เลือกDevice



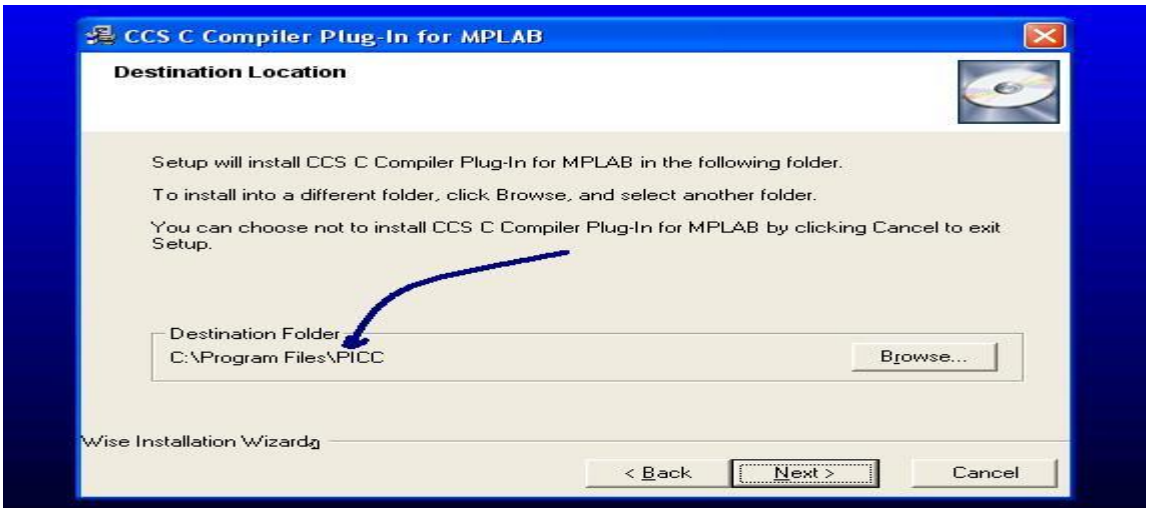
รูป 3.7 การ manual create project เสร็จสมบูรณ์

3.2 การใช้ PIC C V4.08 ร่วมกับ MPLAB IDEในการพัฒนาและตรวจสอบโปรแกรม

PIC C COMPILER สามารถทำงานร่วมกับ MPLAB IDE เพื่ออำนวยความสะดวกในการพัฒนาและตรวจสอบความถูกต้องของชุดคำสั่ง หรือ กระบวนการทำงานในโปรแกรม ว่าเป็นไปตามแผนงานที่ออกแบบไว้หรือไม่ ซึ่งขั้นตอนการ **setup** ให้ PIC C COMPILER สามารถทำงานร่วมกับ MPLAB IDE มีขั้นตอนดังนี้

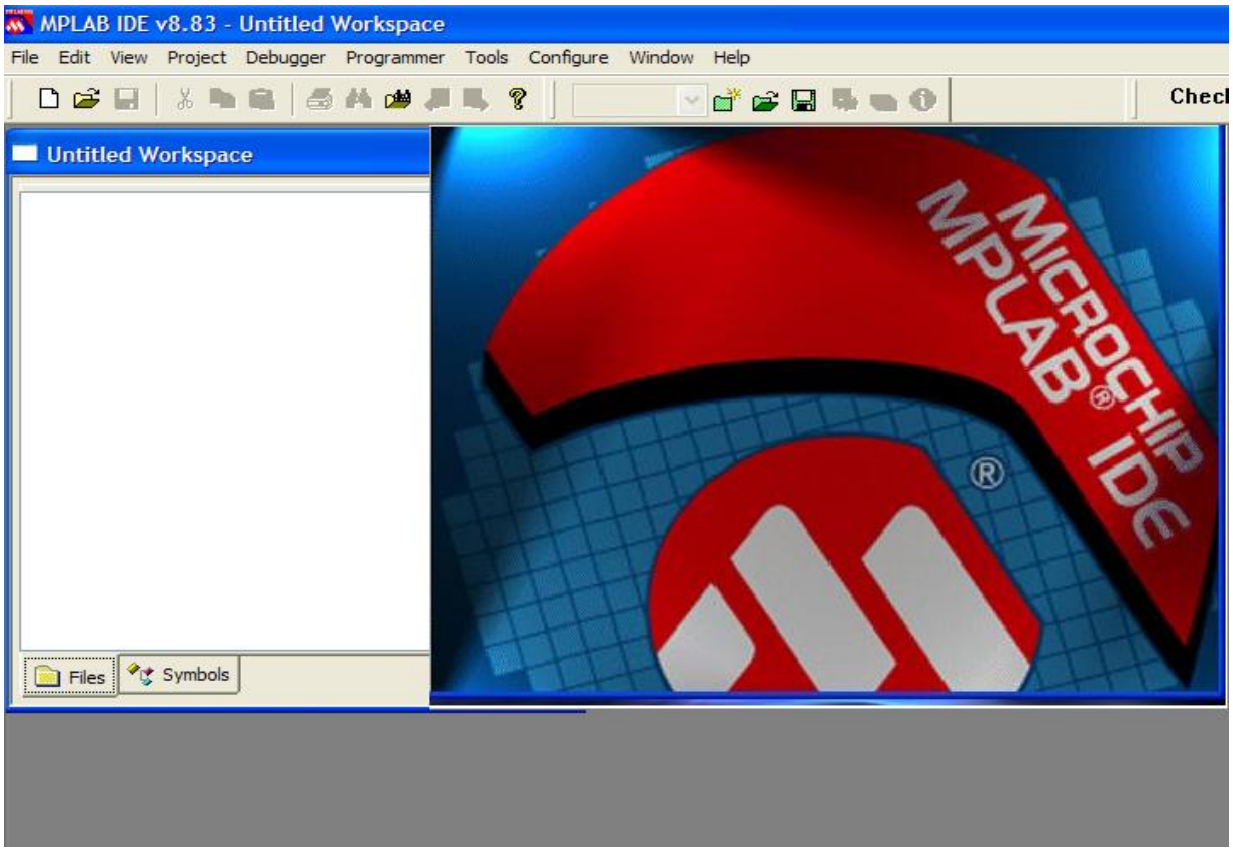
3.2.1 setup_mplab_plugin

รูป 3.8 setup MPLAB plugin

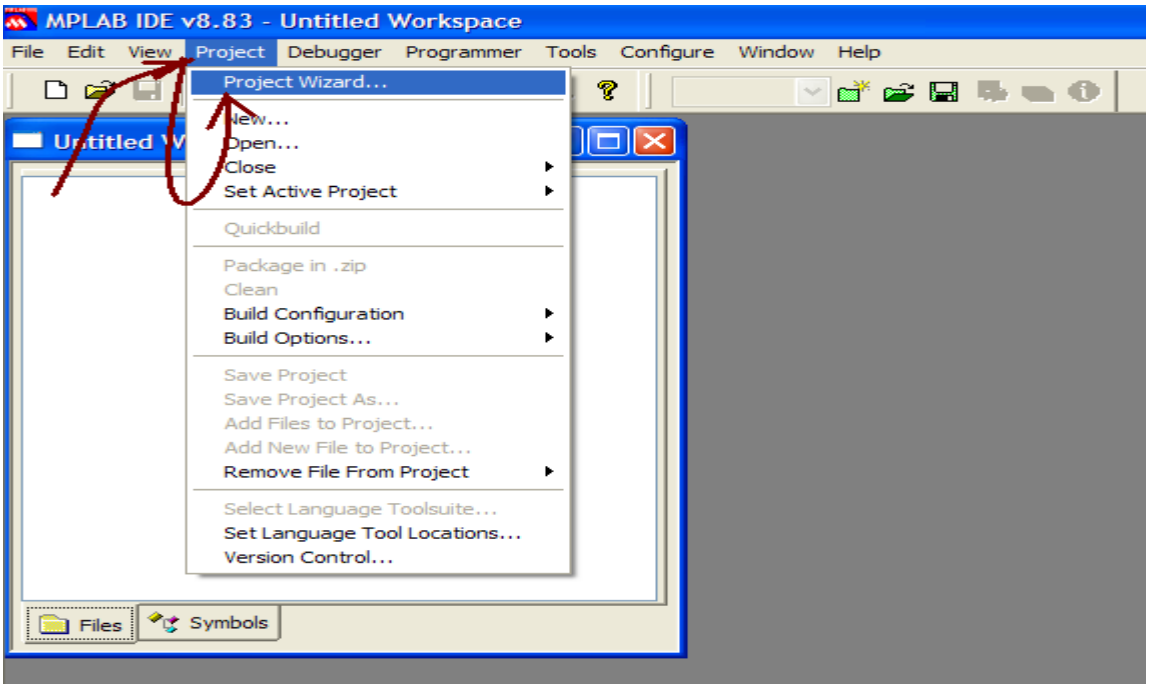


รูป 3.9 Location install

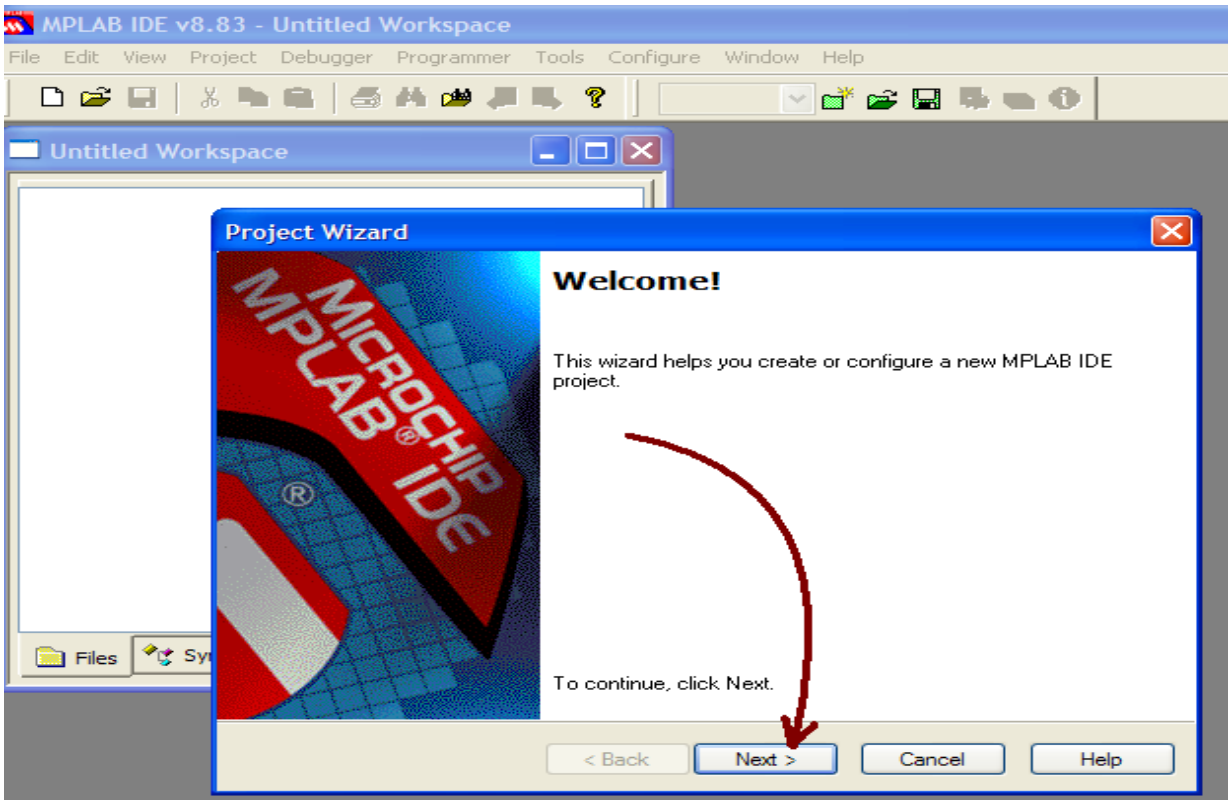
3.2.2 การสร้าง PROJECT ใน MPLAB IDE



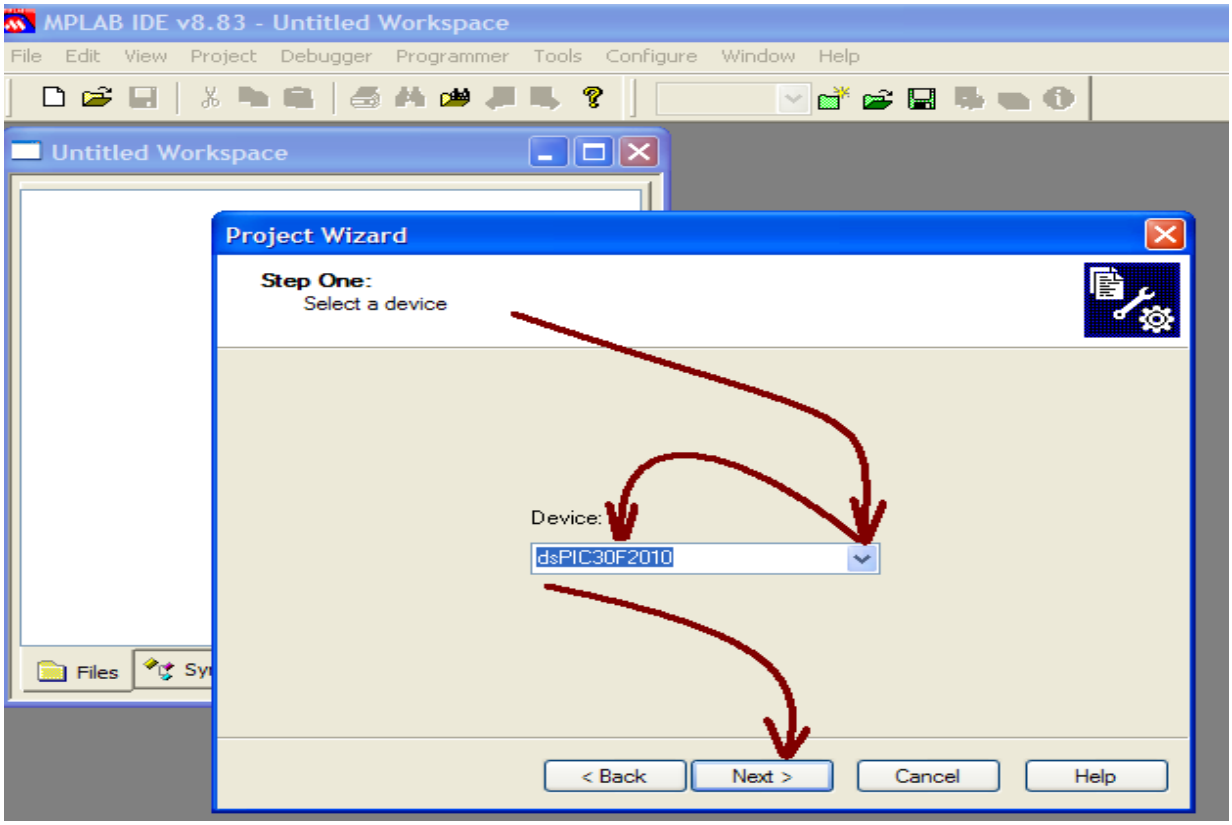
รูป 3.10



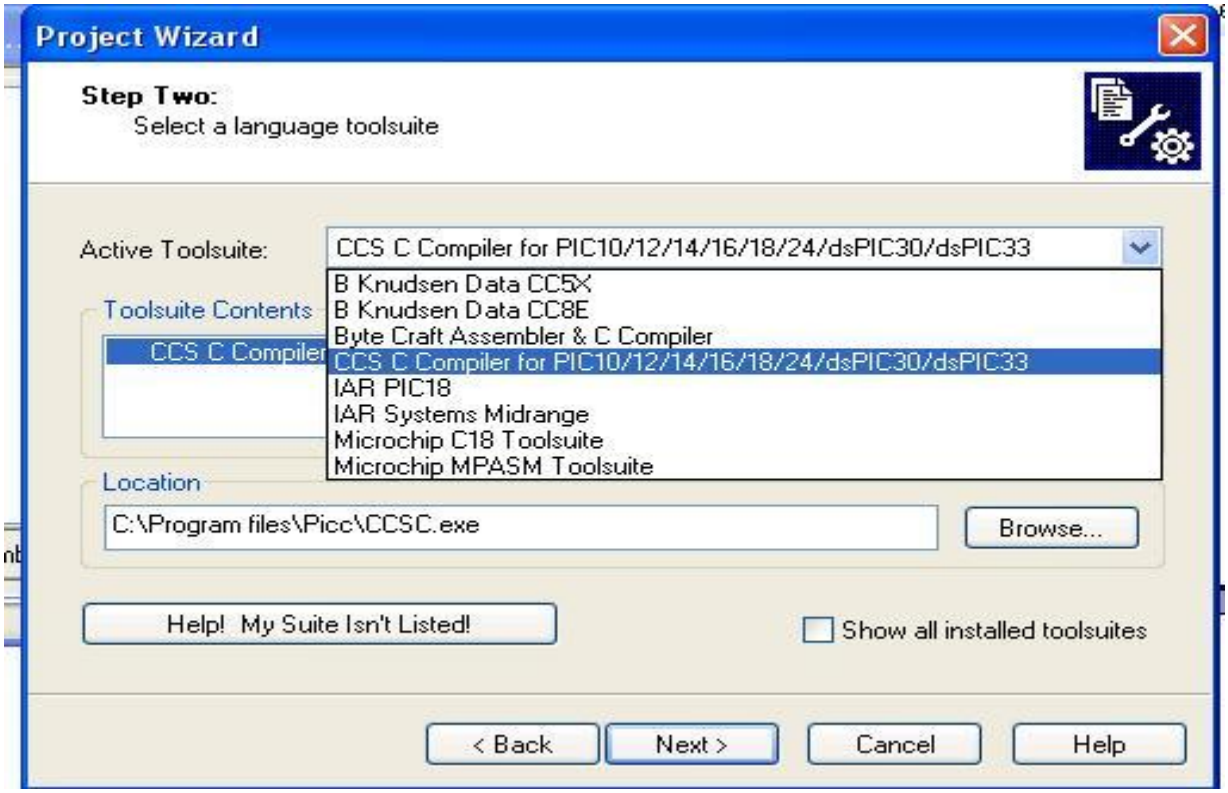
รูป 3.11 เข้าถึงเมนู PROJECT WIZARD



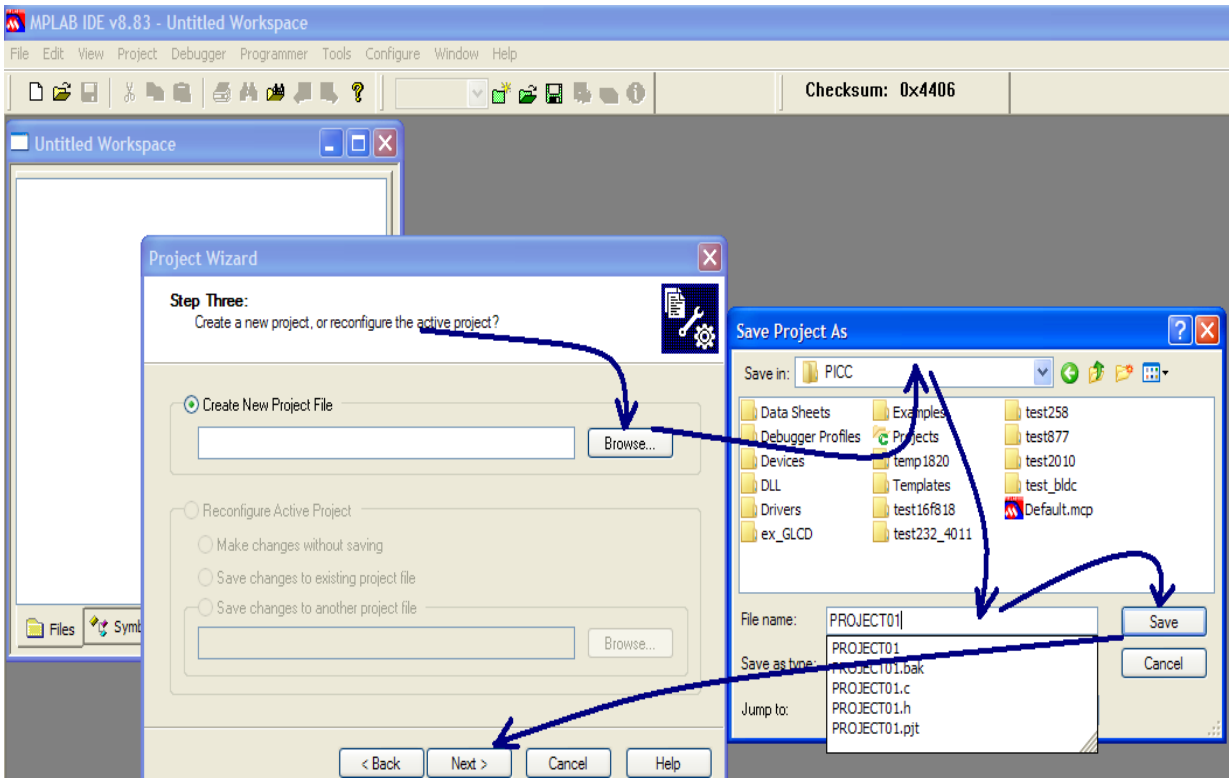
รูป **3.12**



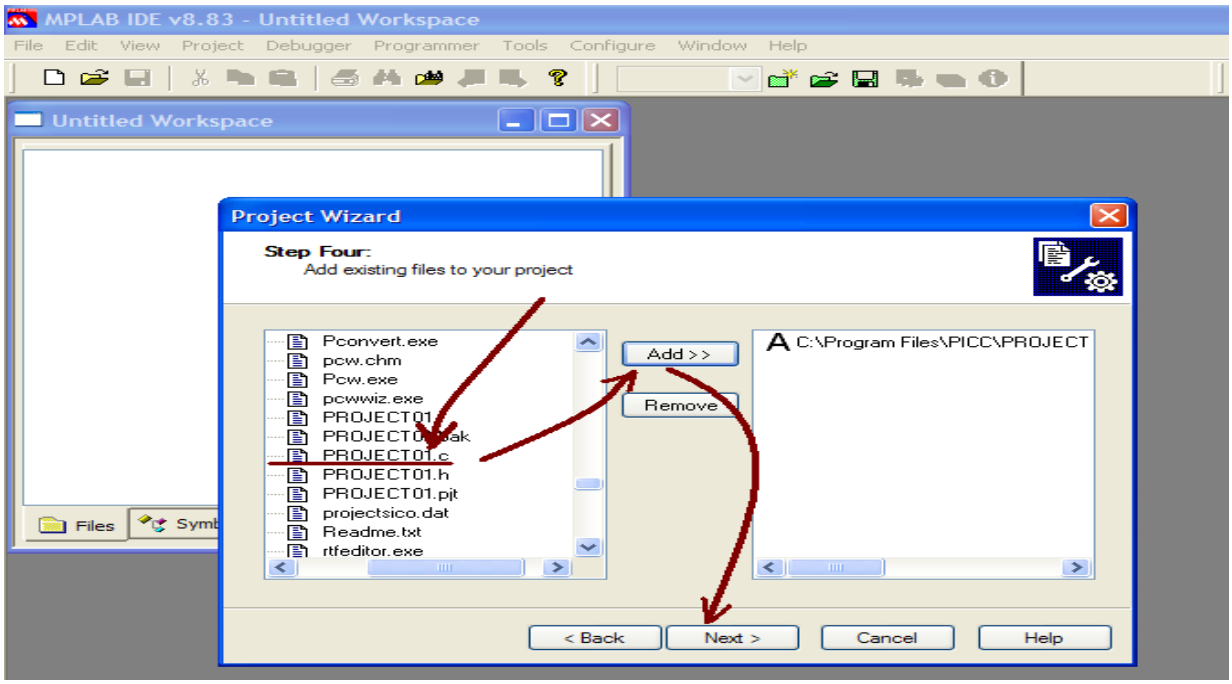
รูป **3.13 เลือกตัวประเมินผล(CPU)**



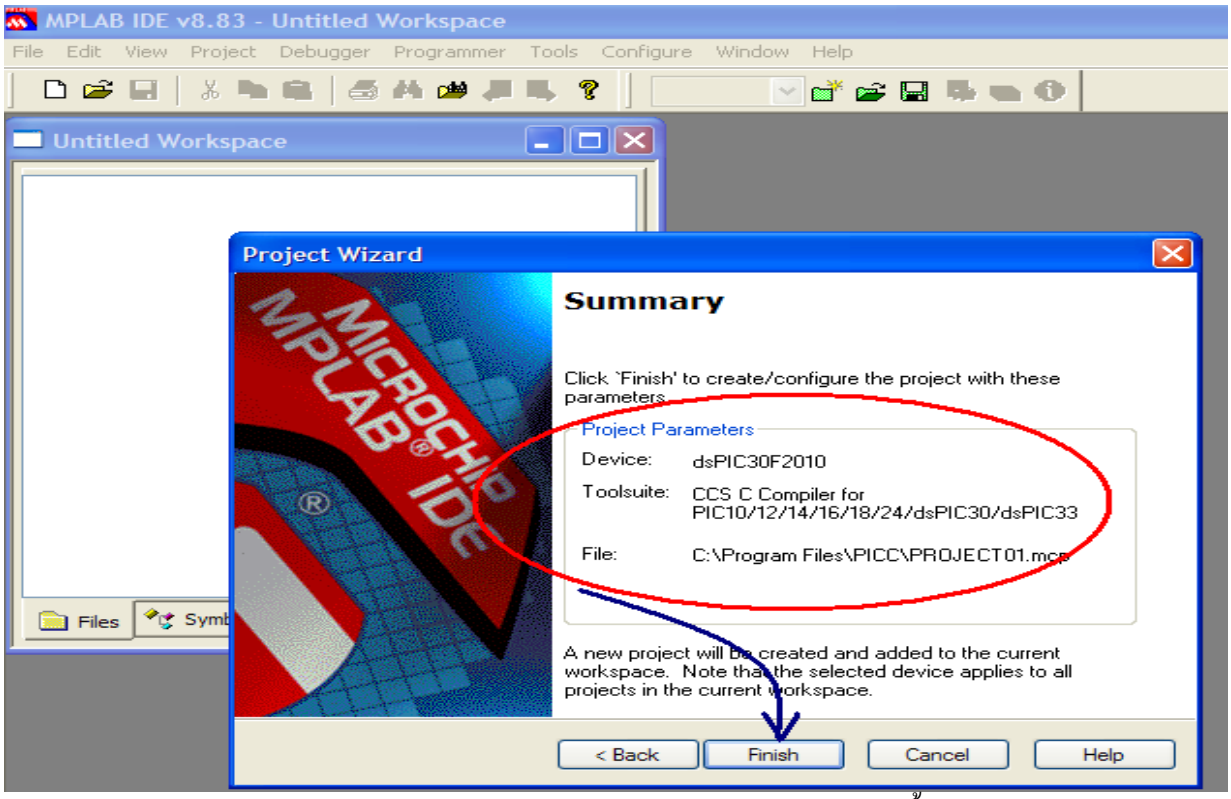
รูป **3.14** Select language toolsuite



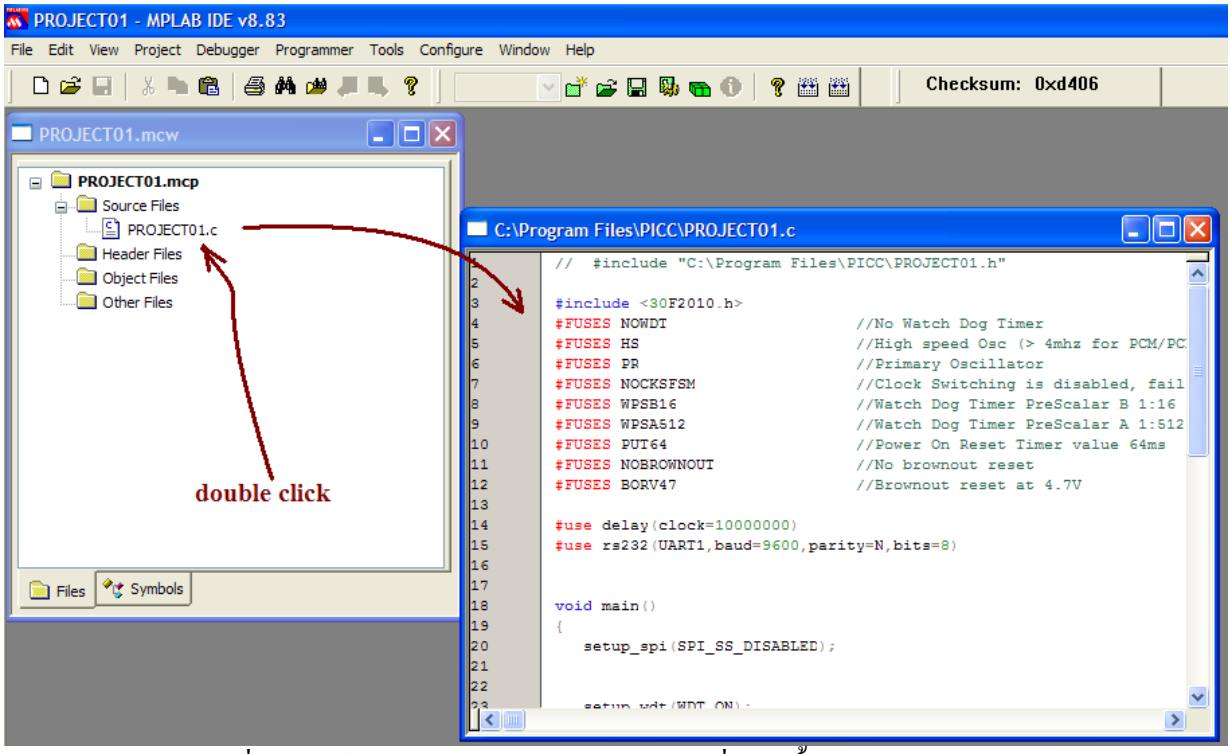
รูป **3.15** เลือก **floder**ในการจัดเก็บ และ ตั้งชื่อ **project**



รูป **3.16** add file to project

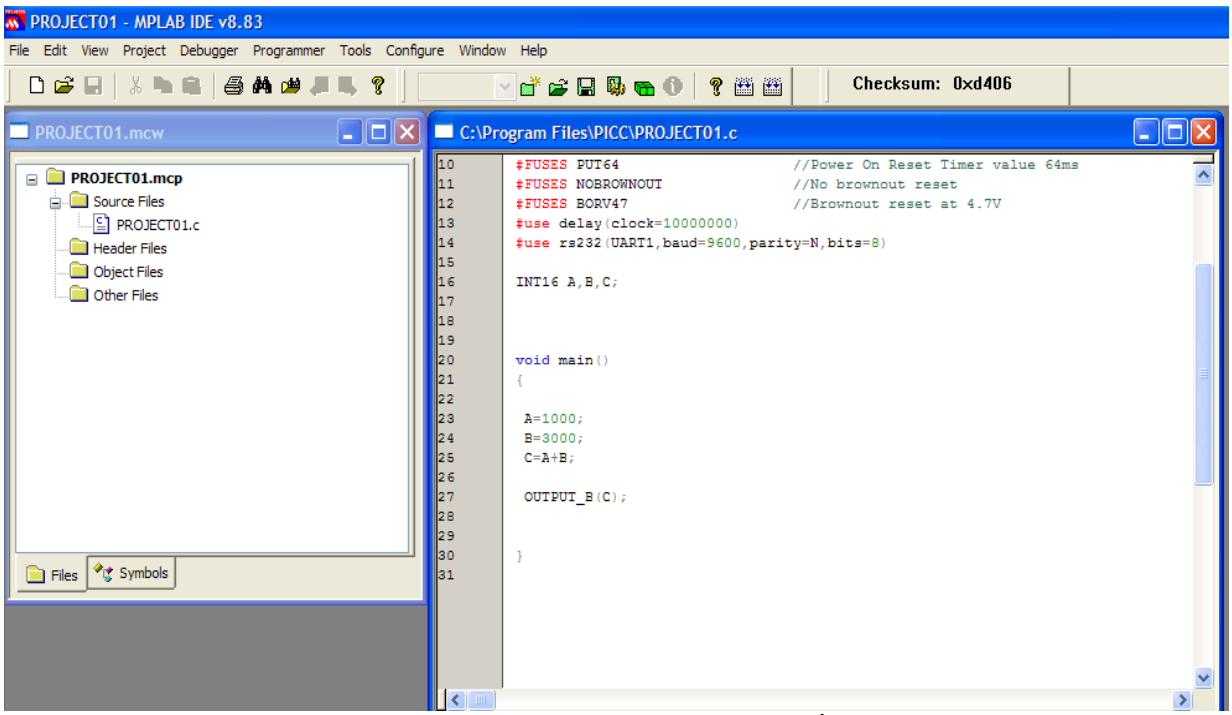


รูป **3.17** ดูความถูกต้องในองค์ประกอบของ **project** และ **click Finish** จบขั้นตอนการสร้าง **project**



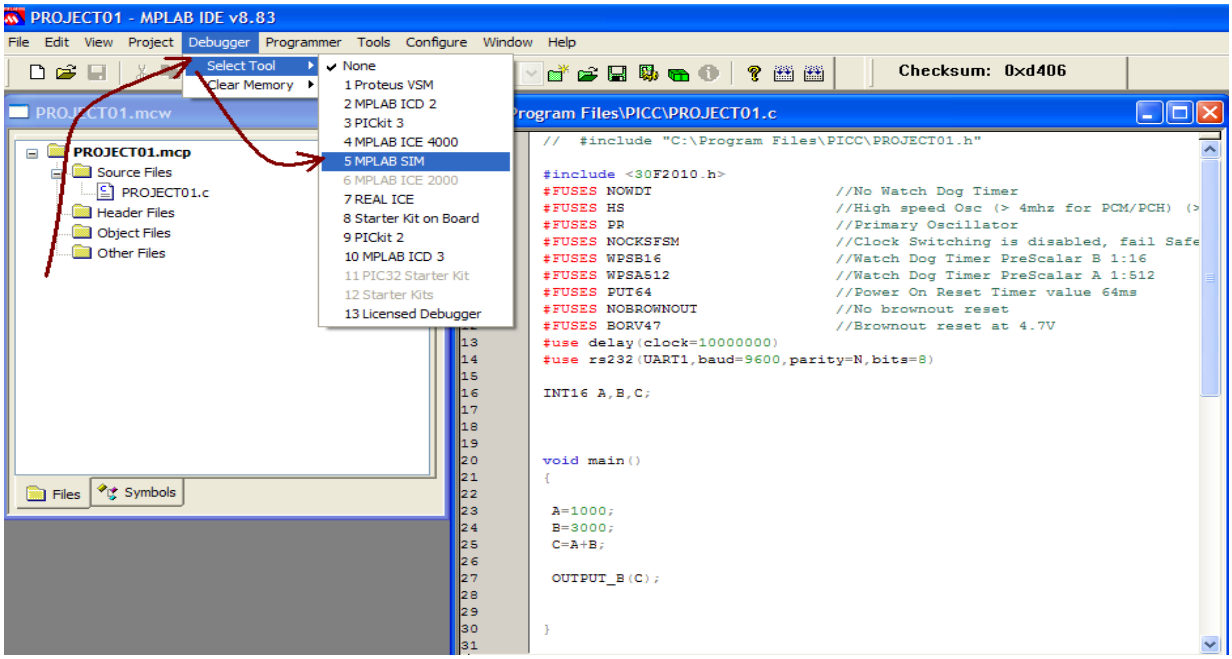
รูป **3.18** **Project** ที่สมบูรณ์มี**source file PROJECT01.C** ที่สร้างขึ้นจากโปรแกรม **PIC C COMPILER**

ตัวอย่างการใช้ **MPLAB IDE** ตรวจสอบค่าตัวแปรและค่าใน **REGISTER**

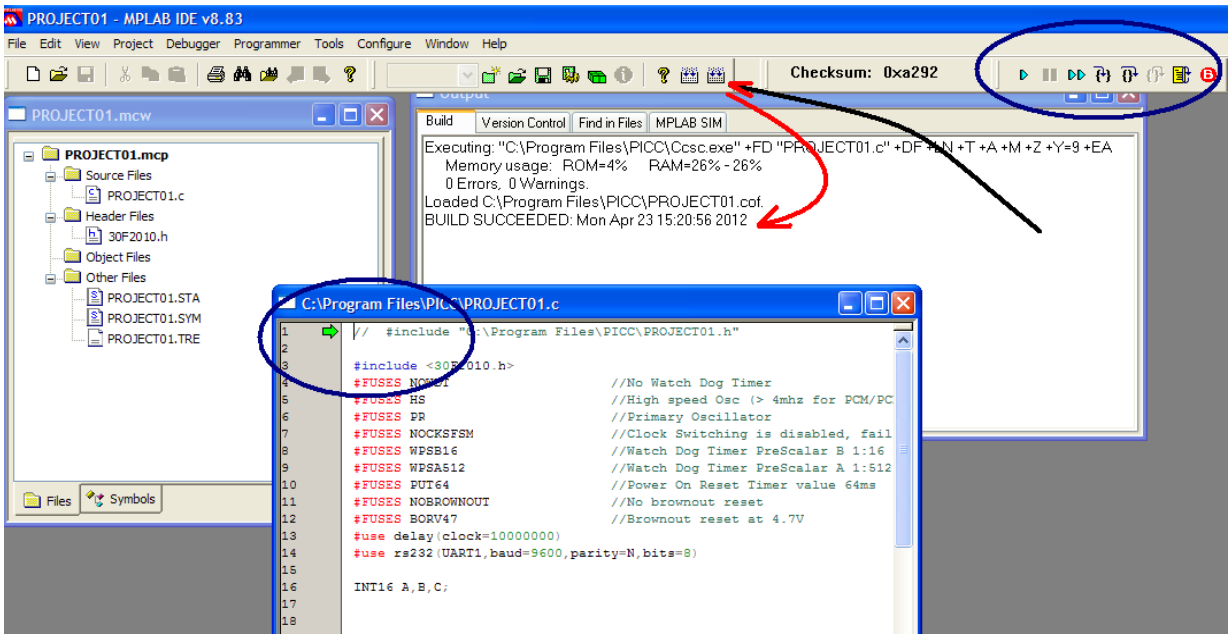


รูป **3.19** ตัวแปร **A B C** และ **REGISTER PORT B** ที่ต้องการตรวจสอบ

จากรูป(**3.19**) เราต้องการตรวจสอบค่าในตัวแปร **A**หลังจากโปรแกรมปฏิบัติตามคำสั่งในบรรทัดที่ **23** ตรวจสอบค่าในตัวแปร **B** หลังจากโปรแกรมปฏิบัติตามคำสั่งในบรรทัดที่ **24** ตรวจสอบค่าในตัวแปร **C** หลังจากโปรแกรมปฏิบัติตามคำสั่งในบรรทัดที่ **25** และ ตรวจสอบค่าใน **REGISTER PORTB** หลังจากโปรแกรมปฏิบัติตามคำสั่งในบรรทัดที่ **27** ซึ่งขั้นตอนในการตรวจสอบแสดงเป็นลำดับตามรูป

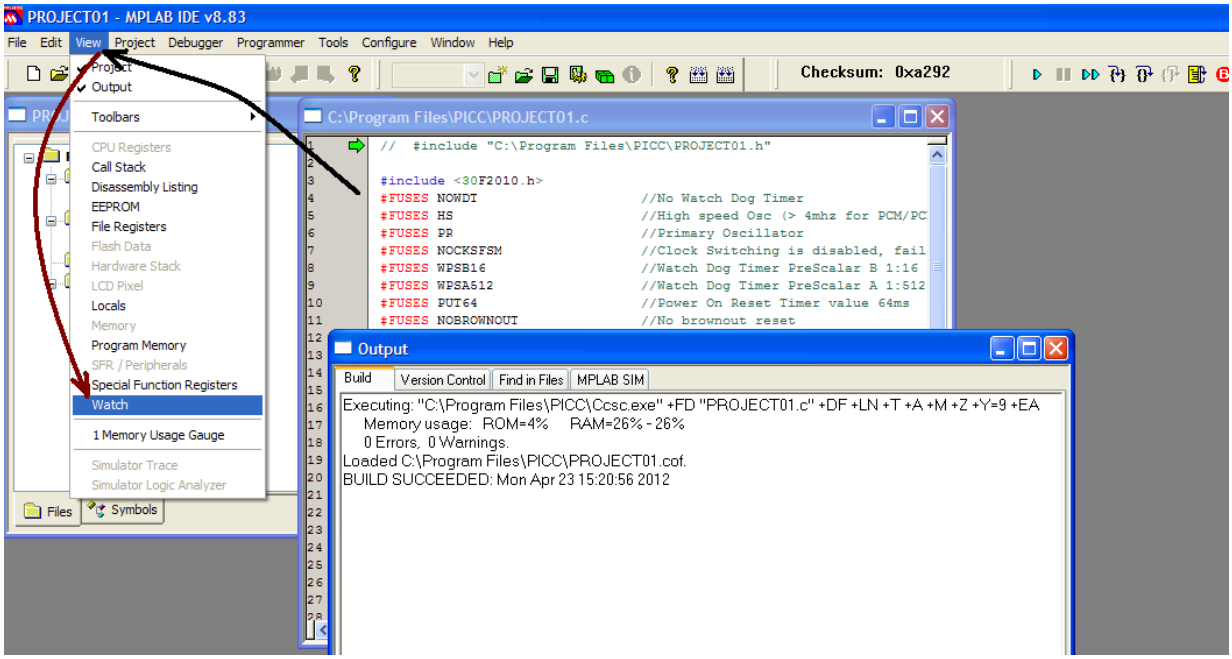


รูป **3.20** เลือก **debug tool**

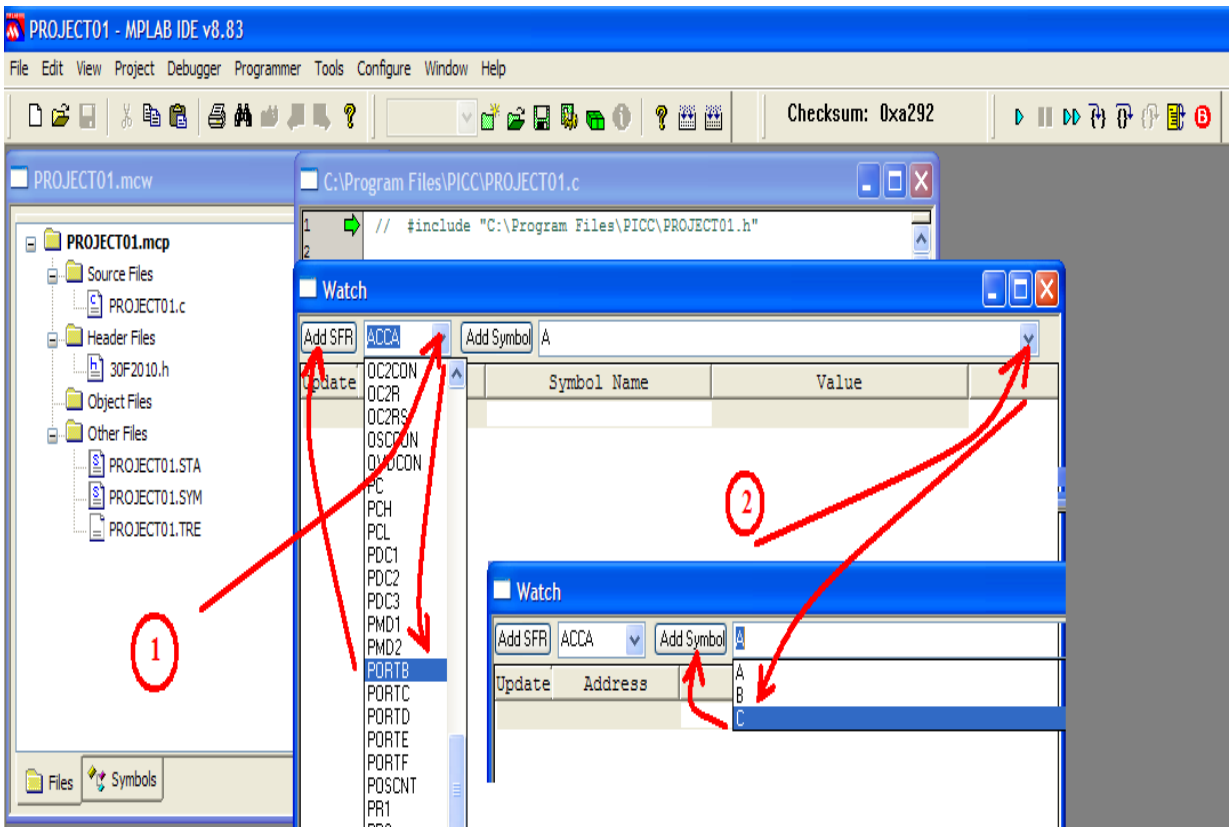


รูป **3.21** ใช้ toolbar menu make project

จากรูปให้สังเกตุความถูกต้องของไวยากรณ์และชุดคำสั่งหลังจากการใช้ **toolbar Make Project** และสังเกตว่าจะมี **toolbar** ควบคุม **step** ในการตรวจสอบ และมีลูกศรบอกบรรทัดของการตรวจสอบแสดงไว้ในวงกลม

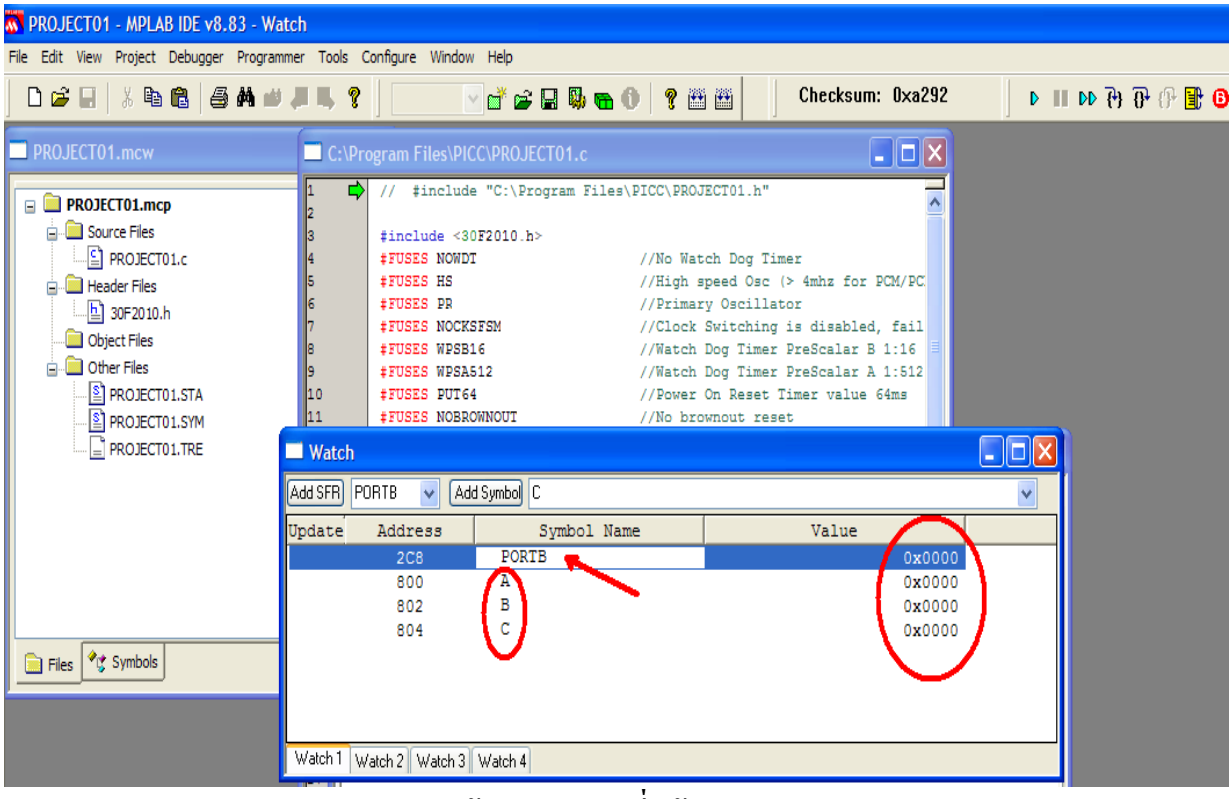


รูป **3.22** เปิดหน้าต่าง WATCH

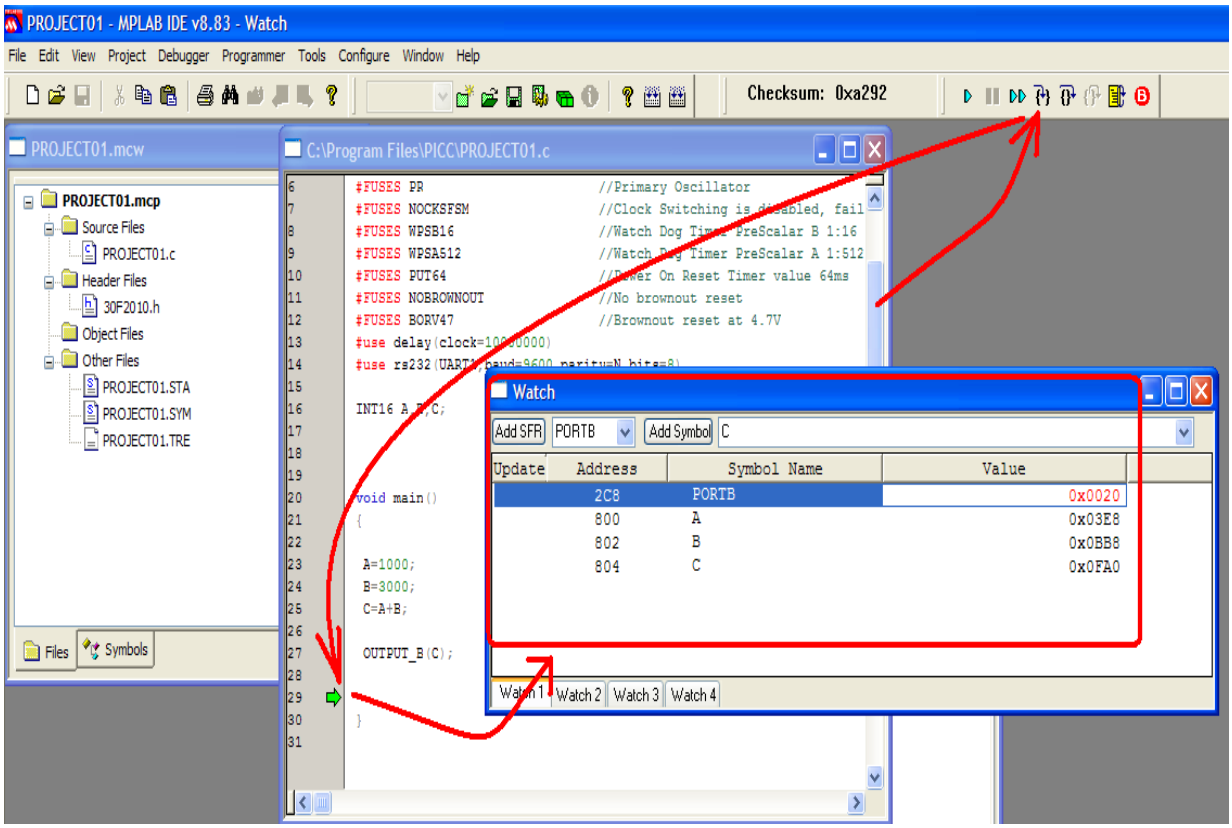


รูป **3.23** Add SFR และ Add Symbol

จากรูปเป็นการนำ **Register** และ ตัวแปรที่ต้องการตรวจสอบมาไว้ที่หน้าต่าง **Watch**



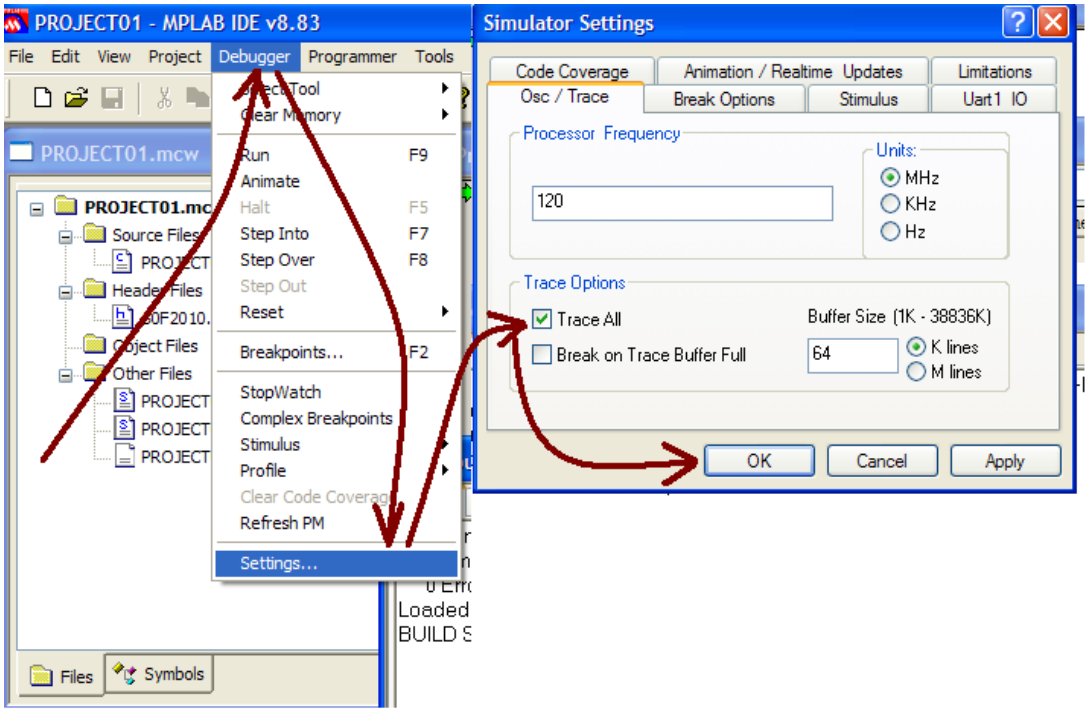
รูป **3.24** หน้าต่าง Watch ที่พร้อมทำการตรวจสอบ



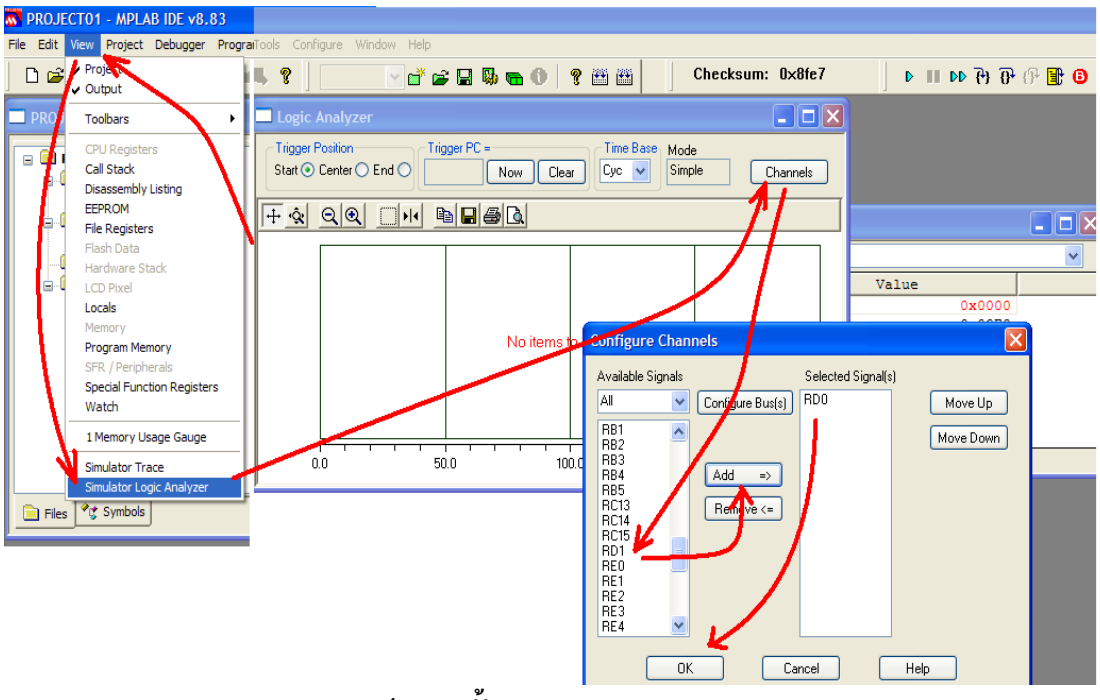
รูป **3.25** ตรวจสอบค่าใน Register และค่าตัวแปรหลังจากโปรแกรมปฏิบัติตามคำสั่งในแต่ละบรรทัด

ตัวอย่างการตรวจสอบโดยใช้ Logic Analyzer

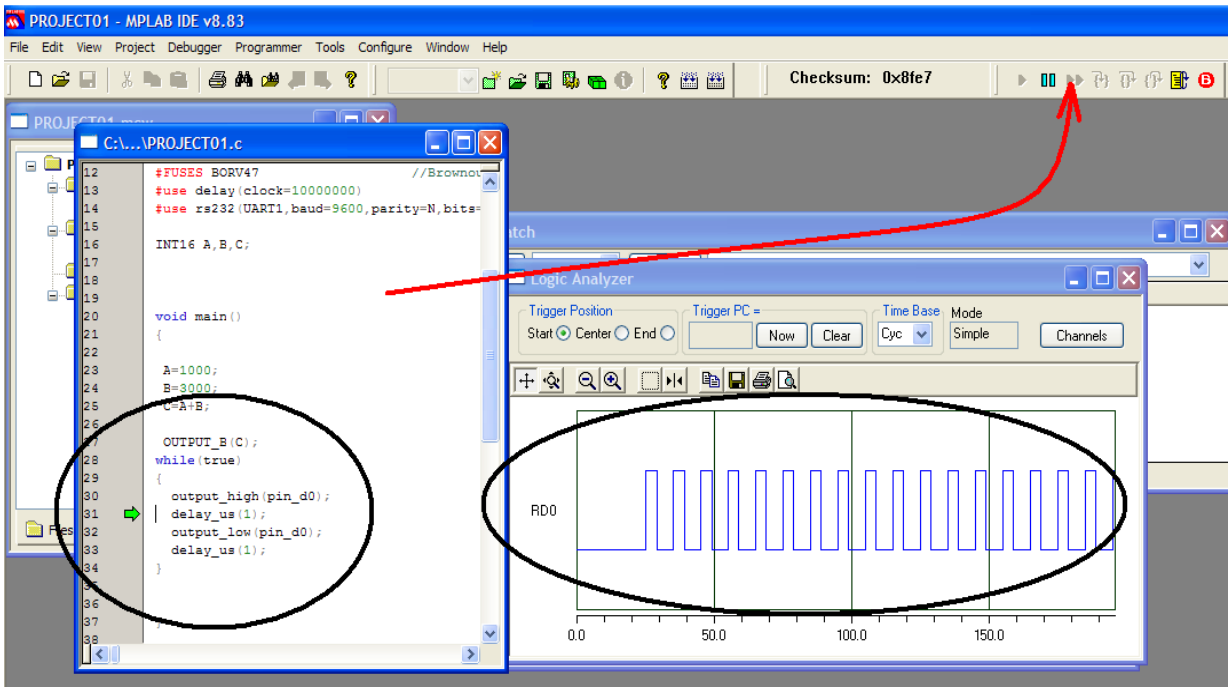
ในบางครั้งการวิเคราะความถูกต้องของระบบจำเป็นต้องดูสถานะทางลอจิกเทียบกับแกนเวลาหลาย ๆ สัญญานพร้อม ๆ กันไม่ว่าจะเป็น อินพุทหรือเอาท์พุท หรือ ความสัมพันธ์ระหว่างอิพุทกับเอาท์พุท **logic Analyzer** ถือว่าเป็นเครื่องมือที่ใช้ตรวจสอบงานในลักษณะดังกล่าวได้ดีมาก ซึ่งขั้นตอนการใช้ แสดงลับดับขั้นตอนได้ดังรูป



รูป 3.26 sitting เพื่อรองรับการใช้ LOGIC ANALYZER



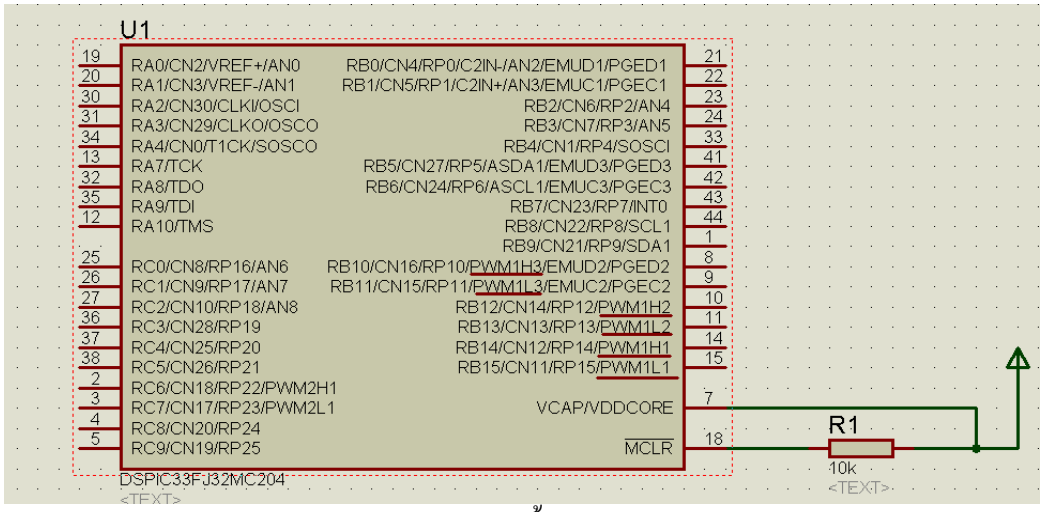
รูป 3.27 ขั้นตอนการ setup Logic Analyzer



รูป 3.28 **เลือกการตรวจสอบแบบ** Animation

3.2 การใช้ PIC C V4.08 ร่วมกับ PROGRAM PROTEUS V 7.4 SP3 ในการพัฒนาและตรวจสอบกระบวนการทำงาน

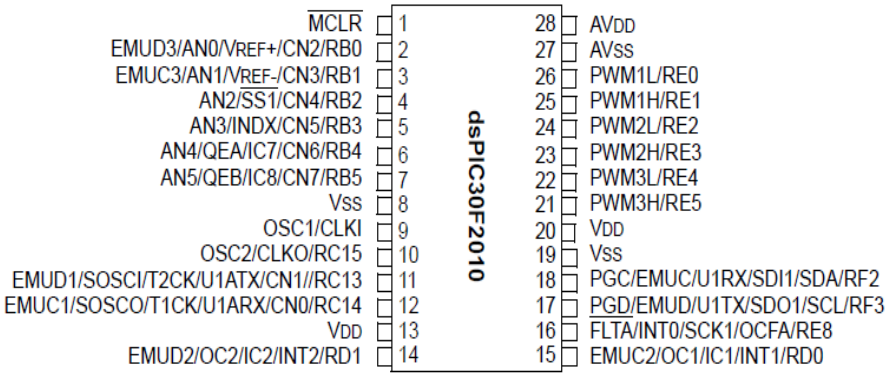
Program Proteus นับได้ว่าเป็นโปรแกรมที่ตอบสนองความต้องการของนักออกแบบระบบควบคุมแบบสมองกลฝังตัวได้เป็นอย่างดี ด้วยความมารถในการจำลองการทำงานของระบบควบคุมที่มีส่วนของหน่วยประเมินผลกลาง(**Central Processing Unit**) ที่ต้องอาศัยการพัฒนาชุดคำสั่งหรือโปรแกรมในการบริหารจัดการให้กับระบบ

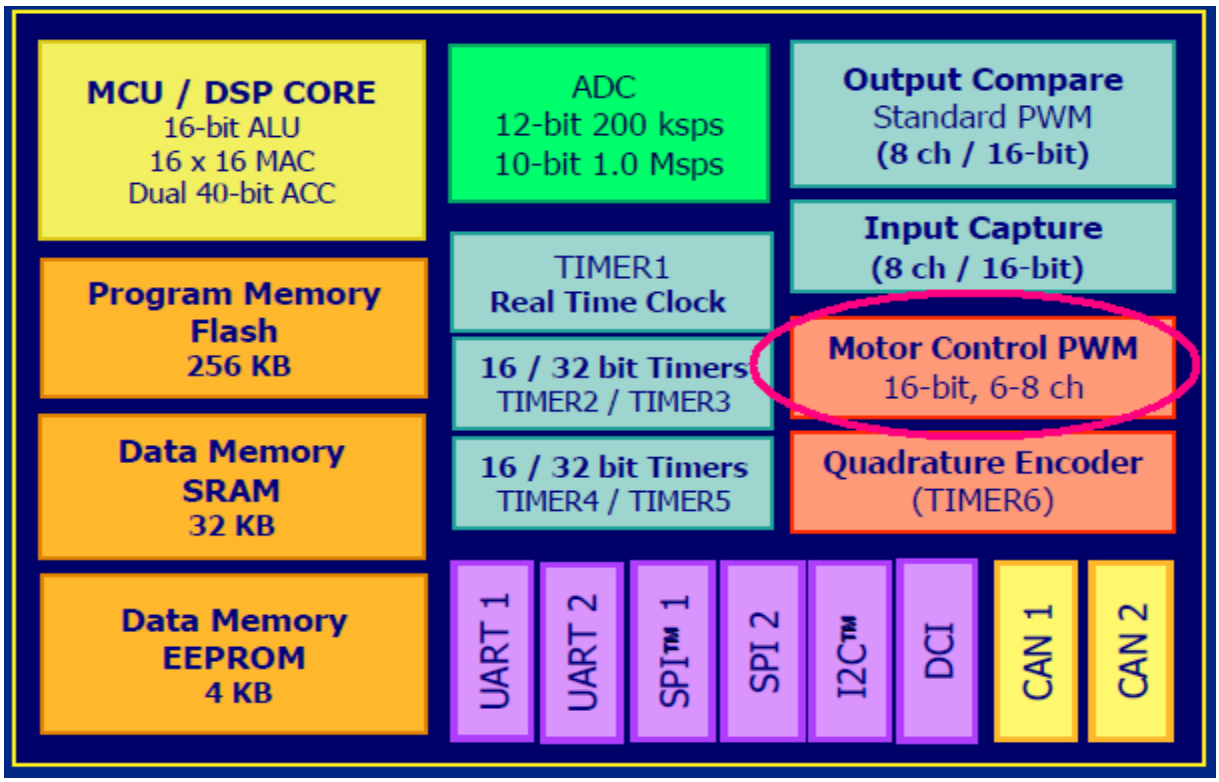


รูป 3.29 วงจรพื้นฐานในการจำลองการทำงานของ **DSPIC**

เนื่องจาก PROGRAM PROTEUS V 7.4 SP3 ไม่มี SIMULATOR MODEL ของ **DSPIC 30F2010** แต่อย่างไรก็ตามเราสามารถใช้ **DSPIC 33FJ32MC204** แทนได้ เนื่องจาก**CPU** ทั้งคโมดูล **POWER CONTROL PWM** เหมือน ๆ กัน ซึ่งในลำดับถัดจากนี้ไปเราจะมุ่งประเด็นการเรียนรู้ไปที่การใช้งานโมดูล **POWER CONTROL PWM** ในการสร้างชุดควบคุมมอเตอร์ดีซีแบบไร้แปรงถ่านต่อไป

3.2.1 ใช้โปรแกรม **Proteus** ในการเรียนรู้และการเข้าถึงสถาปัตยกรรมของไมโครคอนโทรลเลอร์





รูป **3.30** รูปลักษณ์ภายนอกและองค์ประกอบภายใน **DSPIC30F2010**

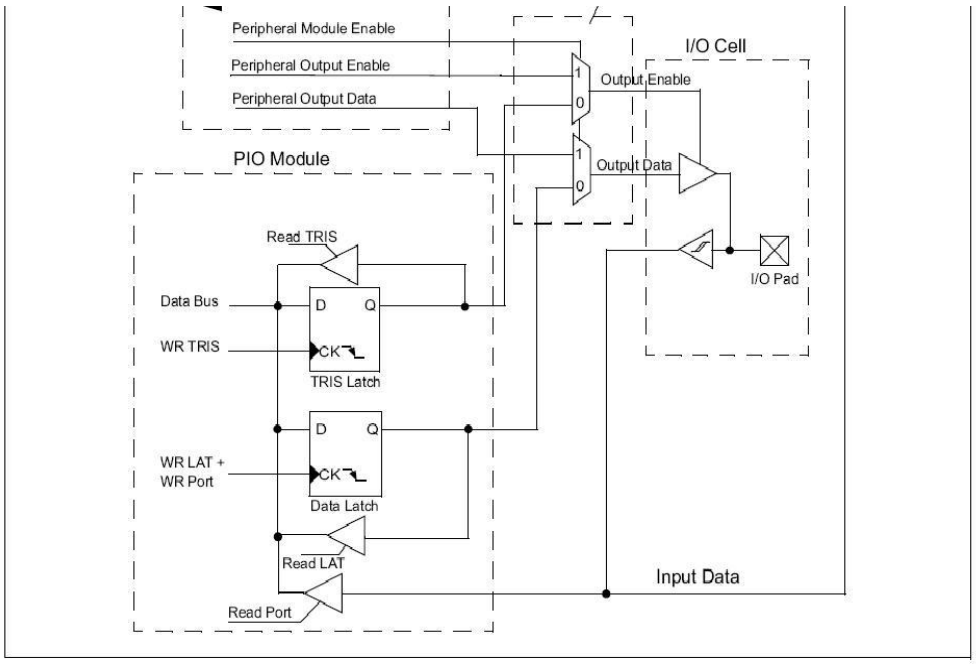
รูปร่างหน้าตาภายนอกและองค์ประกอบภายในของ **DSPIC30F2010**

จะเห็นได้ว่าองค์ประกอบภายในของ **DSPIC30F2010** ได้ออกแบบมาเพื่อรองรับกับระบบควบคุมโดยเฉพาะซึ่งองค์ประกอบดังกล่าวพอจะแบ่งออกเป็นกลุ่ม ๆ คือ

- **กลุ่ม**ที่ใช้ในการจัดเก็บลำดับคำสั่ง,หน่วยประเมินผลผลและการบริหารจัดการข้อมูลเช่น **MCU/DSP CORE,Program Memory,Data Memory** และ **Data Memory EEPROM** เป็นต้น
- **กลุ่ม**ที่ใช้จัดการกับเรื่องของจำนวนนับและเวลา เช่น **Timer1,Timer2,Timer3,Output Capture** และ **Input Capture** เป็นต้น
- **กลุ่ม**ที่ใช้สำหรับเปลี่ยน สัญญานอนาล็อกเป็นดิจิตอล คือ **ADC**
- **กลุ่ม**ที่ใช้ในการติดต่อสื่อสารกับอุปกรณ์ภายนอกเช่น **UART1,UART2,SPI1,SPI2,I2C,CAN1** และ **CAN2** เป็นต้น
- กลุ่มที่ใช้ในการออกควบคุมการทำงานของมอเตอร์ เช่น **Motor Control PWM** และ **Quadrature Encoder**

PIC C compiler และ **PROTEUS** นับได้ว่าเป็นเครื่องมือที่ทำให้นักออกแบบและพัฒนาระบบควบคุม ทำงานได้ง่ายขึ้นมากเนื่องจากมีความสะดวกพัฒนาโปรแกรมและตรวจสอบความถูกต้องของโปรแกรมได้สะดวกมาก ก่อนที่เราจะทำความเข้ากับการใช้โมดูล **Motor Control PWM** ให้เรามาทำความเข้าใจกับองค์ประกอบรอบข้างซึ่งจะเป็นส่วนที่ช่วยสนับสนุนให้ระบบของเราสมบูรณ์มากยิ่งขึ้น

ตัวอย่าง การใช้ built in function เพื่อการเข้าถึงข้อมูลใน register I/O PORT



รูป **3.31** โครงสร้างภายในของ **I/O PORT**

จากรูป (**3.31**) จะเห็นว่าการที่จะเข้าถึงข้อมูลในแต่ละ I/O PORT นั้น มี่ส่วนที่เข้ามาเกี่ยวข้องถึง 4 กลุ่มด้วยกันคือ

- Peripheral Module
- Output Multiplexers
- PIO Module
- I/O Cell

เพราะจะนั้นการที่จะทำความเข้าใจในการใช้คำสั่งในการติดต่อกับ i/o port คือการตรวจสอบค่าในแต่ละกลุ่มว่าอยู่ใน register ตัวไหนบ้างและแต่ละตัวมีความสัมพันธ์ในกลุ่มกันอย่างไร

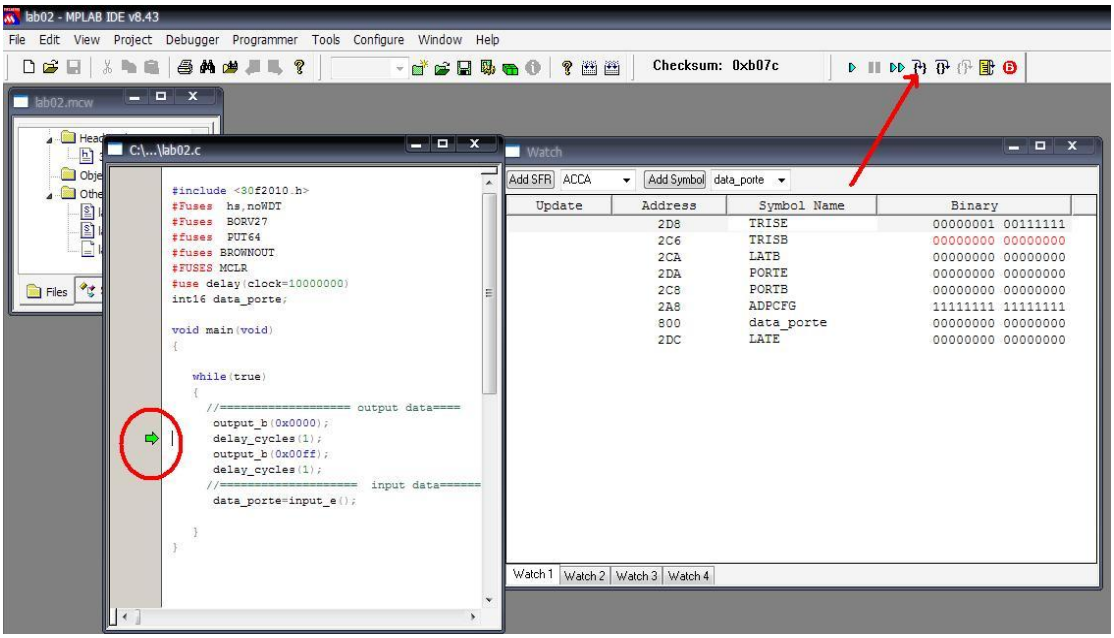
<u>ตัวอย่างโปรแกรม</u> การใช้คำสั่งในการติดต่อกับ i/o port

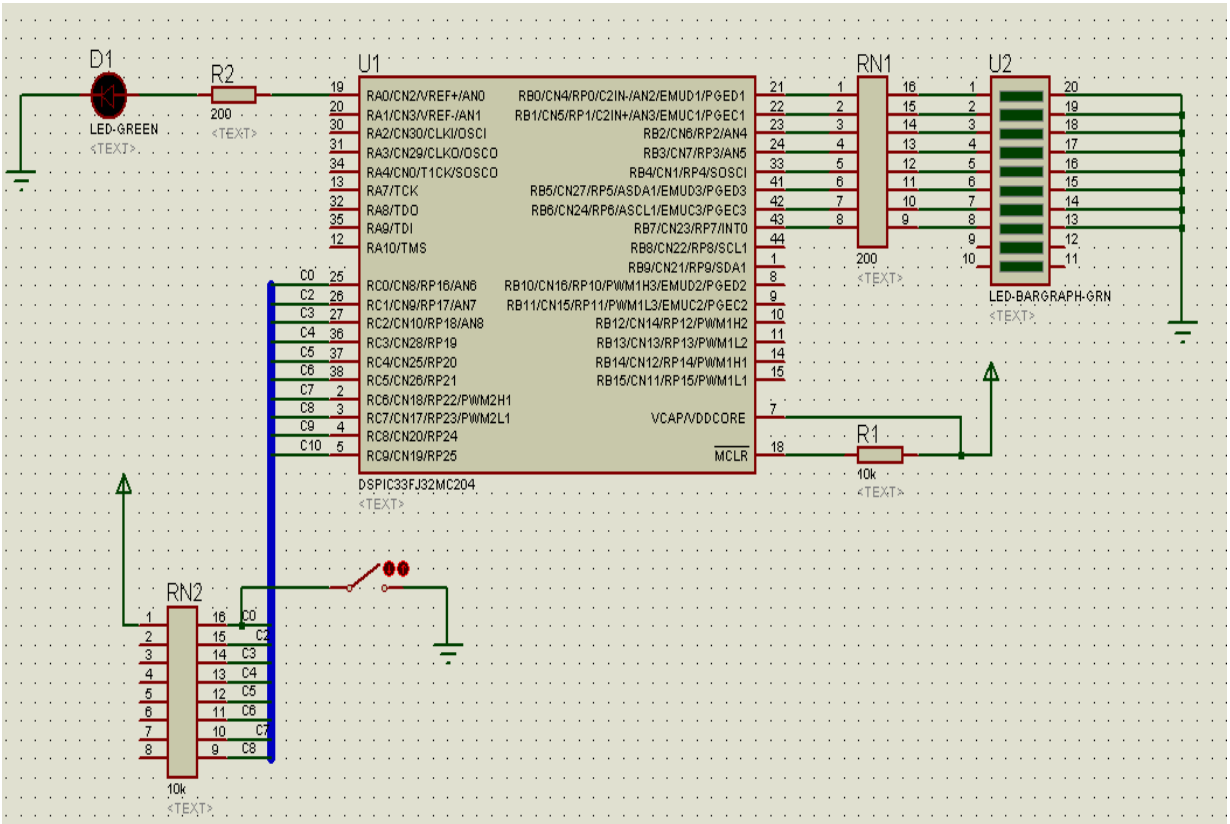
```
#include <30f2010.h>
#Fuses  hs,noWDT
#Fuses  BORV27
#fuses  PUT64
#fuses BROWNOUT
#FUSES MCLR
#use delay(clock=10000000)
int16 data_portc;

void main(void)
{
  while(true)
  {
    output_b(0x0000);
    delay_ms(1);
    output_b(0x00ff);
    delay_ms(1);

    data_portc=input_c();

  }
}
```

OUTPUT DATA

INPUT DATA

จากโปรแกรมดังกล่าวสามารถเรียนรู้กระบวนการจัดการในกลุ่ม **Register** ที่ทำหน้าที่รับหรือส่งข้อมูลออกภายนอกโดยใช้ **MPLAB IDE** ดูการเปลี่ยนแปลงในกลุ่ม **Register** ดังกล่าว และใช้โปรแกรม **Proteus** จำลองวงจรเสมือนจริง





รูป 3.32 การตรอบสอบค่าใน REGISTER และตรวจสอบโปรแกรมด้วยวงจร

ตัวอย่าง โปรแกรม การเข้าถึงข้อมูลใน **Register** โดยใช้ **Preprocessor #Locate**

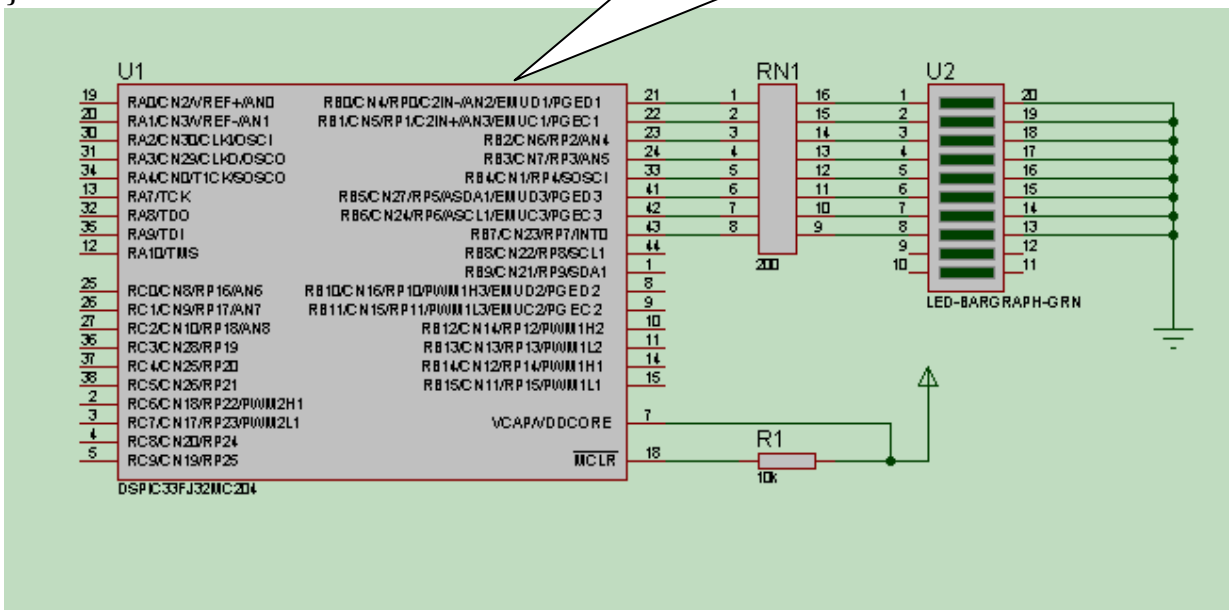
ในบางครั้งบางกรณีการใช้ชุดคำสั่งสำเร็จรูปทำให้เราขาดความยืดหยุ่นในการทำงาน หรือ ไม่สามารถผ่านความละเอียดในข้อกำหนดของงานนั้นได้ จำเป็นต้องมีการประยุคใช้ preprocessor ประกอบกับรูปแบบการเข้าถึงข้อมูลตามข้อกำหนดของ compiler มีไว้ให้ ต่อไปนี้จะเป็นวิธีการหาตำแหน่ง address register และ การใช้ #locate เพื่อระบุตำแหน่ง address ของ register

```
#include <30f2010.h>
#Fuses  hs,noWDT
#use delay(clock=10000000)

int16 trisb;              // trisb  access data 16 bit
#locate trisb=0x2c6  //  2c6  is address of  register  trisb
int16 latb;
#locate latb = 0x2ca
int16 portb;
#locate portb=0x2c8

void main(void)
{
  trisb=0x0000;
  while(true)
  {
    latb=0xffff;
    delay_ms(100);
    latb=0x0000;
    delay_ms(100);
    }
}
```

Direct accessing register by #locate

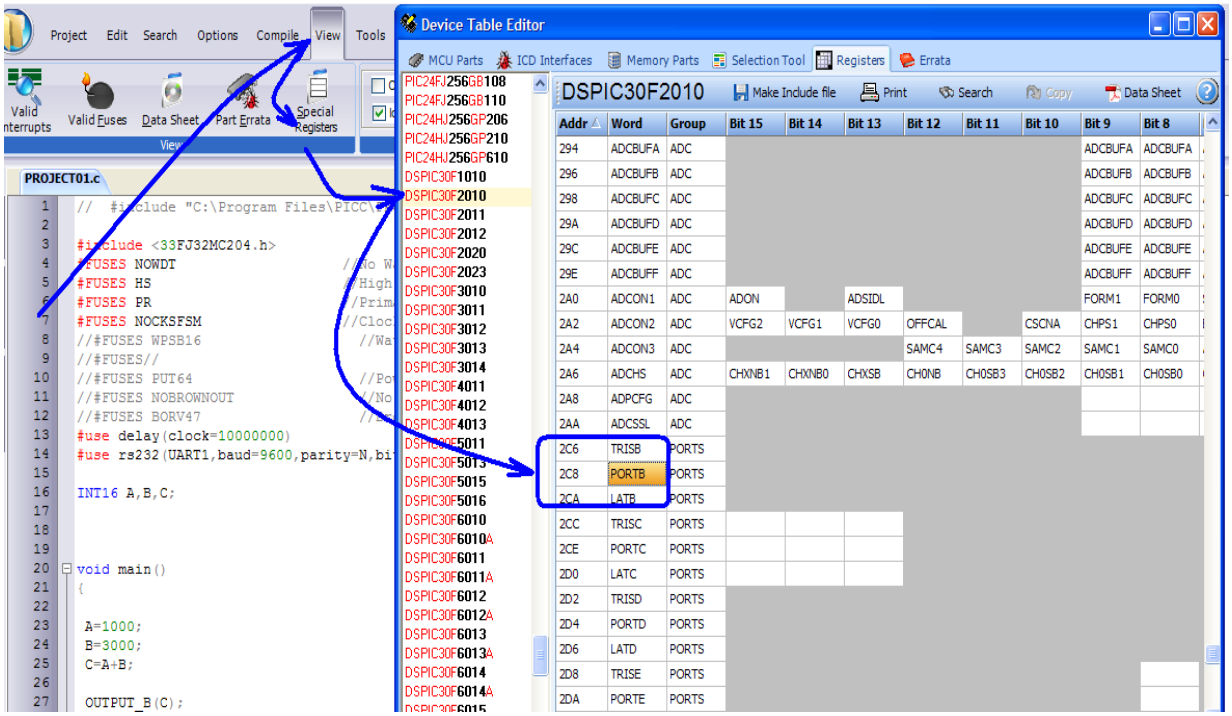
DSPIC33FJ32MC20

```
int16 trisb;              // trisb  access data 16
bit
#locate trisb=0x2c8  //  2c6  is address of
register  trisb
int16 latb;
#locate latb = 0x2cc
int16 portb;
#locate portb=0x2ca
```



รูป **3.33** วงจรจำลองการทำงาน

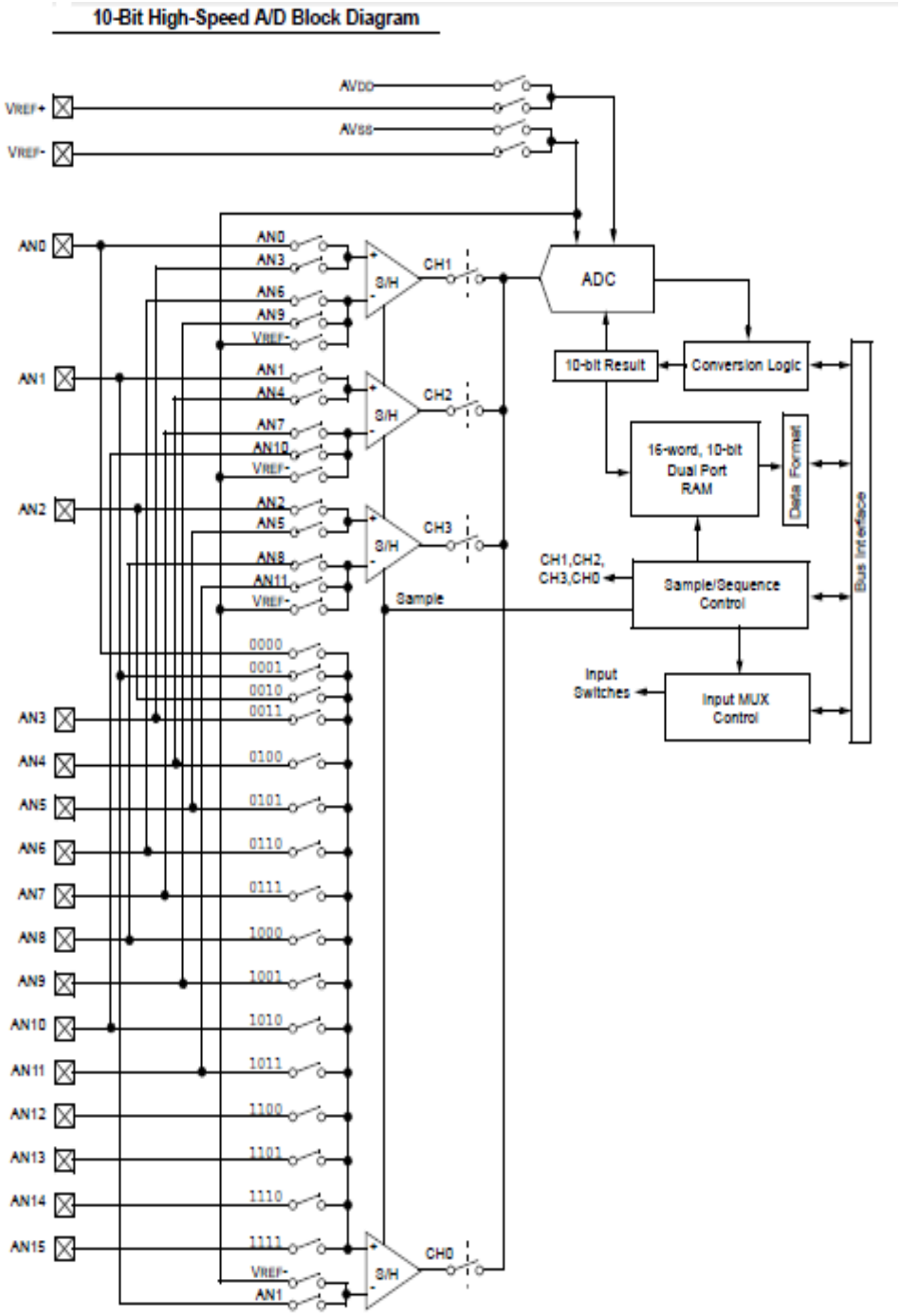
ชึ่งการค้นหาตำตำของ แต่ละ Register ใน PIC C Compiler ทำได้โดยใช้เมนู View ->Special Rrgister ดังรูป



รูป **3.34** การหาร Address Special Function Register(SFR)

การใช้งานโมดูล ANALOG TO DIGITAL CONVERTER

ANALOG TO DIGITAL CONVERTER MODULE เป็นอีกหนึ่งโมดูลที่ติดมากับไมโครคอนโทรลเลอร์เกือบทุกเบอร์อาจจะเนื่องมาจากระบบควบคุมโดยส่วนใหญ่แล้วต้องมีส่วนที่ต้องประเมินผลสัญญาอินพุทที่เข้ามาในรูปแบบของสัญญาณอนาล็อก เช่น การเปลี่ยนแปลงของอุณหภูมิ การเปลี่ยนแปลงของ แรงดัน และ กระแสไฟฟ้าเป็นต้น รูปข้างล่างแสดงให้เห็น BLOCK DIAGRAM ของโมดูล ANALOG TO DIGITAL CONVERTER



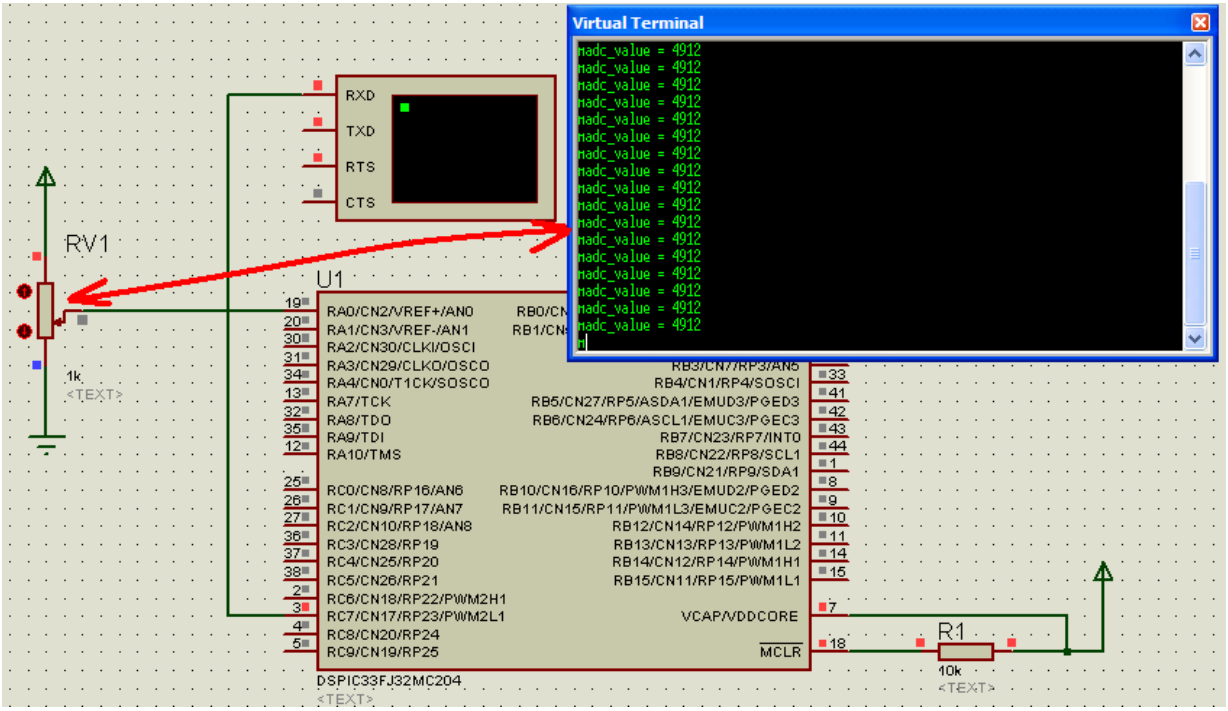
รูป **3.35** BLOCK DIAGRAM ANALOG TO DIGITAL CONVERTER

จะเห็นว่า DIGITAL SIGNAL CONTROL **สามารถรับสัญญาณอนาล็อกได้หลายช่อง** PIC C COMPILER **มีคำสั่งสำเร็จรูปในการเลือกรับสัญาณอนาล็อกในแต่ละช่อง ซึ่งทำให้การพัฒนาโปรแกรมที่เกี่ยวข้องกับการประเมินผลสัญญาณอนาล็อกทำได้ง่ายขึ้นมาก**

ตัวอย่าง **โปรแกรมการใช้งาน** ANALOG TO DIGITAL MODULE

```
#include <30f2010.h>
#device adc=10
#Fuses  hs,noWDT
#Fuses  BORV27
#fuses  PUT64
#fuses BROWNOUT
#FUSES MCLR
#use delay(clock=10000000)
#use rs232(UART1,baud=9600,parity=N,bits=8,XMIT=PIN_C7)

Int16  adc_value;

void main(void)
{
  SETUP_ADC_PORTS(sAN0|VSS_VDD );
  SETUP_ADC(ADC_CLOCK_INTERNAL);
  while(true)
  {
    set_adc_channel( 0 );
    duty = read_adc();
    printf("adc_value = %lu\r\m",adc_value);
  }
}
```

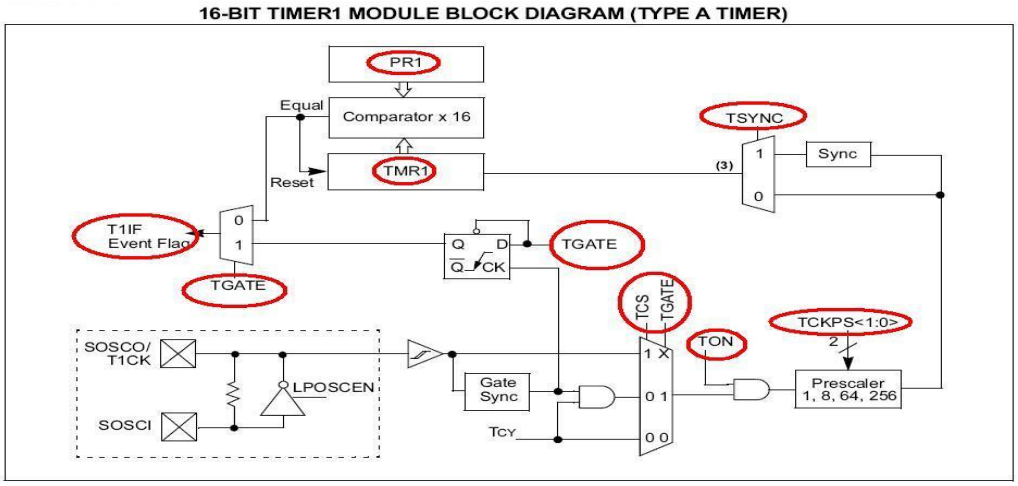
Function setup เพื่อใช้งานโมดูลรับสัญญาณ analog channel 0



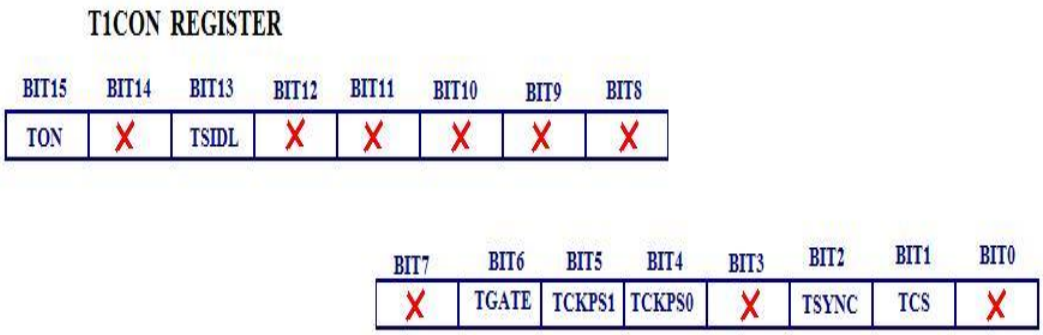
รูป **3.36** วงจรจำลองการทำงานการรับค่าสัญญาณอนาล็อก

การใช้โมดูล Timer1

Module timer ถือเป็นโมดูลที่ติดมากับ microcontroller เกือบทุกเบอร์หรือ ทุก ๆ ตระกูลก็ว่าได้เนื่องจากว่ามันเป็นโมดูลที่มีความจำเป็นกับการทำงานในหลาย ๆ ด้าน ที่ต้องเกี่ยวข้องกับเวลา และ การนับเพราะโดย ธรรมชาติ ของงานแล้วมักจะเกี่ยวข้องกับแกนเวลา และจำนวนนับเสมอ เพราะจะนั้นการทำความเข้าใจกับการใช้งาน โมดูลนี้ถือได้ว่าเป็นสิ่งจำเป็นอย่างยิ่ง



รูป **3.37** Block diagram logic control Timer1



รูป **3.38** ตำแหน่งประจำบิตของ register T1CON

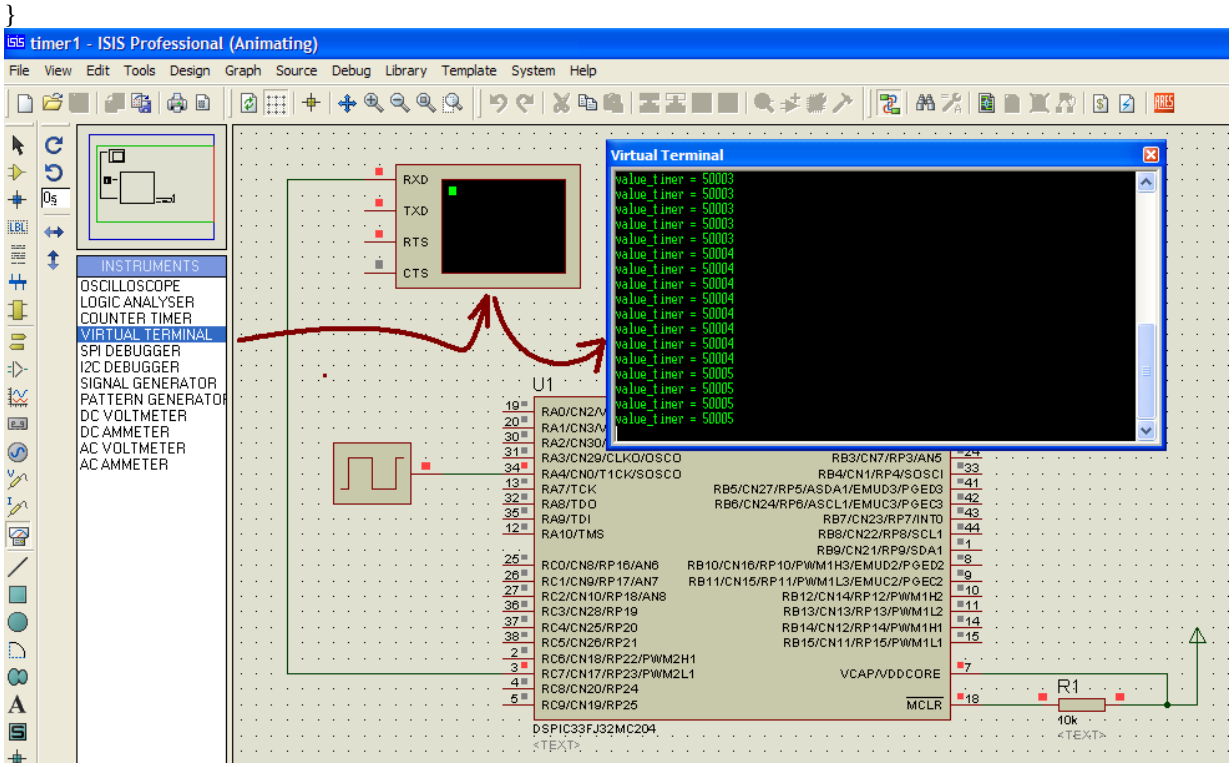
จากรูป จะได้ถึงองค์ประกอบการกำหนดการทำงานให้กับ timer 1 พอสังเขบดังนี้

1 bit เลือก input clock TCS(T1CON.1) TGATE(T1CON.6)
2 bit หารความถี่ (prescale input clock) TCKPS1(T1CON.5) TCKPS0(T1CON.4)
3 bit synchronous mode input clock TSYNC (T1CON.2)
4 bit T1IF เป็น flag จะ set ในกรณี timer1 over flow หรือ ค่าใน timer1 = ค่าใน PR1
5 register TMR1 เป็น counter 16 bit
6 register PR1 เป็น buffer 16 bit ใช้ในกรณีต้องการเปรียบเทียบค่าใน Timer 1

ตัวอย่างโปรแกรมการใช้งาน TIMER1

```
#include <30f2010.h>
#Fuses  hs,noWDT
#Fuses  BORV27
#fuses  PUT64
#fuses BROWNOUT
#FUSES MCLR
#use delay(clock=10000000)
#use rs232(UART1,baud=9600,parity=N,bits=8,XMIT=PIN_C7)

INT16 value_timer1;
void main(void)
{
  SETUP_TIMER1(TMR_EXTERNAL |TMR_DIV_BY_1);
  set_timer1(50000); // define start timer1 value
 while(true)
  {
   value_timer1=get_timer1();
   printf("value_timer = %lu\r\n",value_timer1);
   DELAY_MS(100);
   }
}
```

Timer1 configuration



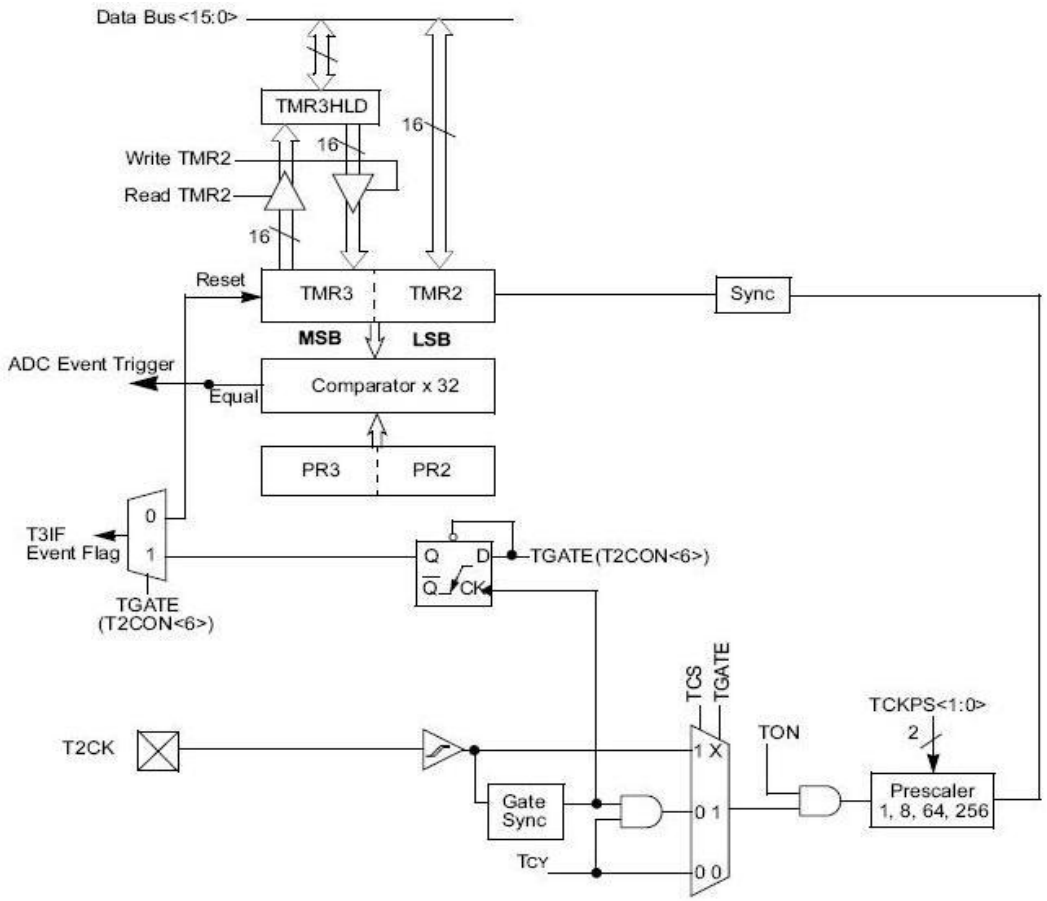
รูป **3.39** ใช้ VERTUAL TERMINAL แสดงผลคำสั่ง printf("value_timer = %lu\r\n",value_timer1)

จะเห็นได้ว่า PIC C COMPILER **ช่วยให้เราใช้งานโมดูล** Timer1 **ได้ง่ายมากเพียงใช้คำสั่งสำเร็จรูป** SETUP_TIMER1(TMR_EXTERNAL |TMR_DIV_BY_1) เพียงคำสั่งเดียวก็สามารถกำหนดให้ Timer1 รับ

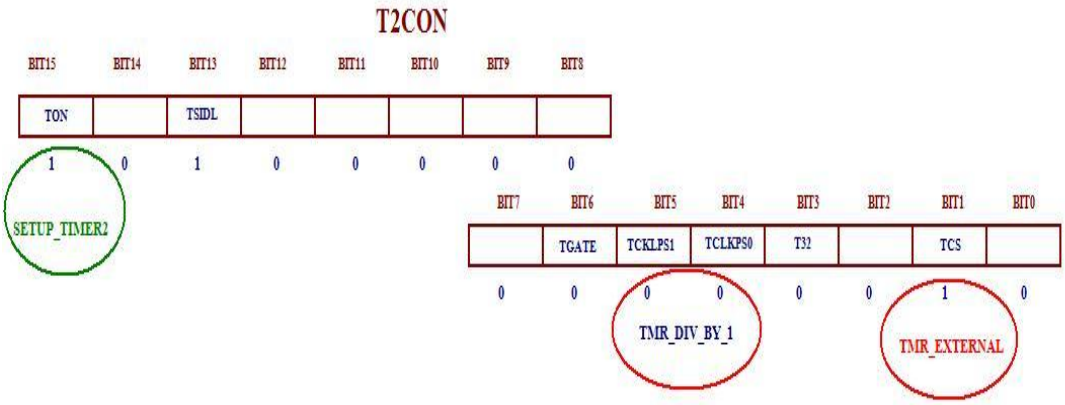
สัญญาณ CLOCK จากภายนอกได้แล้ว อย่างไรก็ตามถ้าจะศึกษารายละเอียดตาม Block diagram logic control Timer1 **จะต้องใช้** MPLAB IDE **เพื่อตรวจสอบค่าใน** REGISTER **ในการทำความเข้าใจกำการ** SETUP **ให้** TIMER1 **ทำงานตามที่เราต้องการ**

<u>การใช้งานโมดูล TIMER2/3ทำงานในโหมด 32 BIT</u>

ใน microcontroller ที่พัฒนาขึ้นมาเป็น digital signal controller จะมี timer/counter ขนาด 32 bit ซึ่งจะไม่มีใน pic12 pic16 และ pic18 ในส่วนของการกำหนดการทำงานนั้นก็ไม่แตกต่างไปจากเดิม คือสามารถกำหนดให้ทำงานใน mode counter(external clock) , mode timer(internal clock) และ mode comparator เหมือนเดิม



รูป **3.40** logic diagram timer 2/3 module



รูป **3.41** ตำแหน่ง bit ใน Register T2CON

เราสามารถกำหนดการทำงานให้กับ **TIMER23** ทำงานในโหมด **16 BIT** หรือ **32 BIT** ก็ได้โดยใช้ฟังชั่นสำเร็จรูปดังนี้

ฟังชั่นในการกำหนดให้ **TIMER23** ทำงานในโหมด **16 BIT**

```
SETUP_TIMER2(TMR_EXTERNAL |TMR_DIV_BY_1);
```

ฟังชั่นในการกำหนดให้ **TIMER23** ทำงานในโหมด **32 BIT**

```
SETUP_TIMER2(TMR_EXTERNAL |TMR_DIV_BY_8|TMR_32_BIT);
```

ตัวอย่างโปรแกรม การใช้ TIMER23 **ขนาด** 32 BIT

```
#include <30f2010.h>
#Fuses  hs,noWDT
#Fuses  BORV27
#fuses  PUT64
#fuses BROWNOUT
#FUSES MCLR
#use delay(clock=10000000)

void main(void)
{
 SETUP_TIMER2(TMR_EXTERNAL |TMR_DIV_BY_8|TMR_32_BIT);
   set_timer23(75530);

  while(true)
   {
    value_timer23=get_timer23();
    printf("value_timer = %lu\r\n",value_timer23);
    DELAY_MS(100);

   }
}
```

โปรแกรมบริการ INTERRUPT

การใช้บริการ interrupt ถือได้เป็นส่วนช่วยให้การบริหารจัดการในการทำงานคล่องตัวและมีประสิทธิมากขึ้น จะนั้นจะเห็นว่า ใน microcontroller ไม่ว่าจะเป็นตระกูลใด ๆ ก็จะมีการพัฒนาให้สามารถรองรับแหล่งกำเหนิด interrupt มากขึ้นเรื่อย

ในการพัฒนาโปรแกรมไม่ว่าเราะเลือกใช้ compiler ภาษาใด ๆ ก็ตาม จำเป็นต้องทำความเข้าใจในส่วนของข้อกำหนดการสร้างโปรแกรมบริการ interrupt เสมอ digital signal controller ได้สร้างส่วนของแหล่งกำเหนิดการ interrupt ไว้อย่างมากอทิเช่น

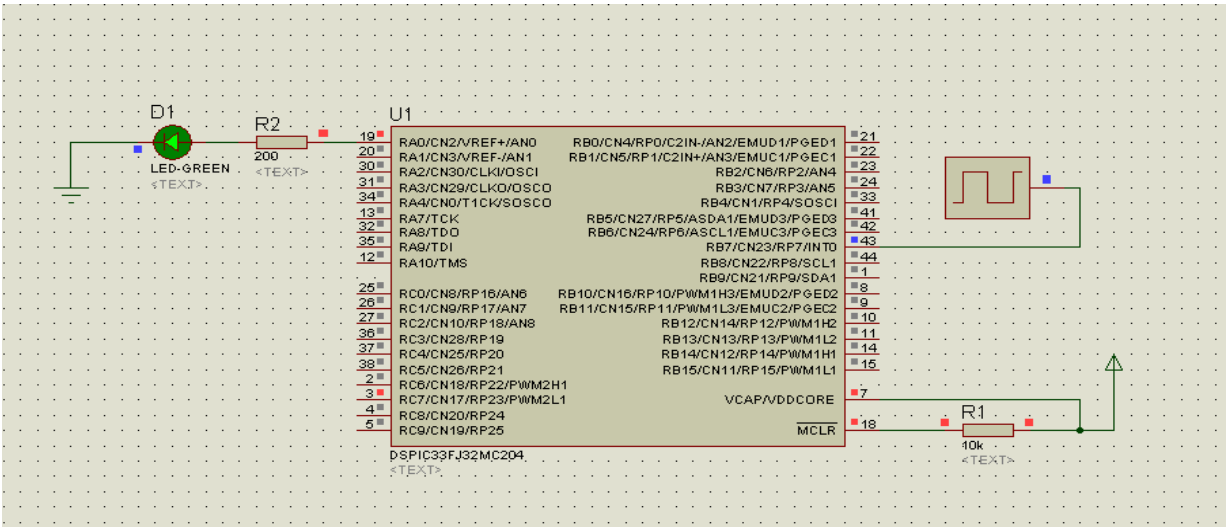
Interrupt จากภายนอก External Interrupt 0 , Input Capture 1, Input Capture 2, External Interrupt 1, External Interrupt 2

Interrupt จากภายใน Output Compare 1, Timer 1, Timer 2 , Timer 3 , UART1 Receiver , UART1Transmitter

ตัวอย่างการเขียนโปรแกรมบริการ INTERRUPT INT0

```
#include <30f2010.h>
#Fuses  hs,noWDT
#Fuses  BORV27
#fuses  PUT64
#fuses BROWNOUT
#FUSES MCLR
#use delay(clock=10000000)

#int_EXT0
void  EXT0_isr(void)
{
OUTPUT_TOGGLE(PIN_A0);
}

void main()
{
   setup_wdt(WDT_OFF);
  enABLE_INTERRUPTS(INT_EXT0);
  ENABLE_INTERRUPTS(INTR_GLOBAL);
  ext_int_edge(0, h_TO_l );

  WHILE(TRUE)
  {
  }

}
```

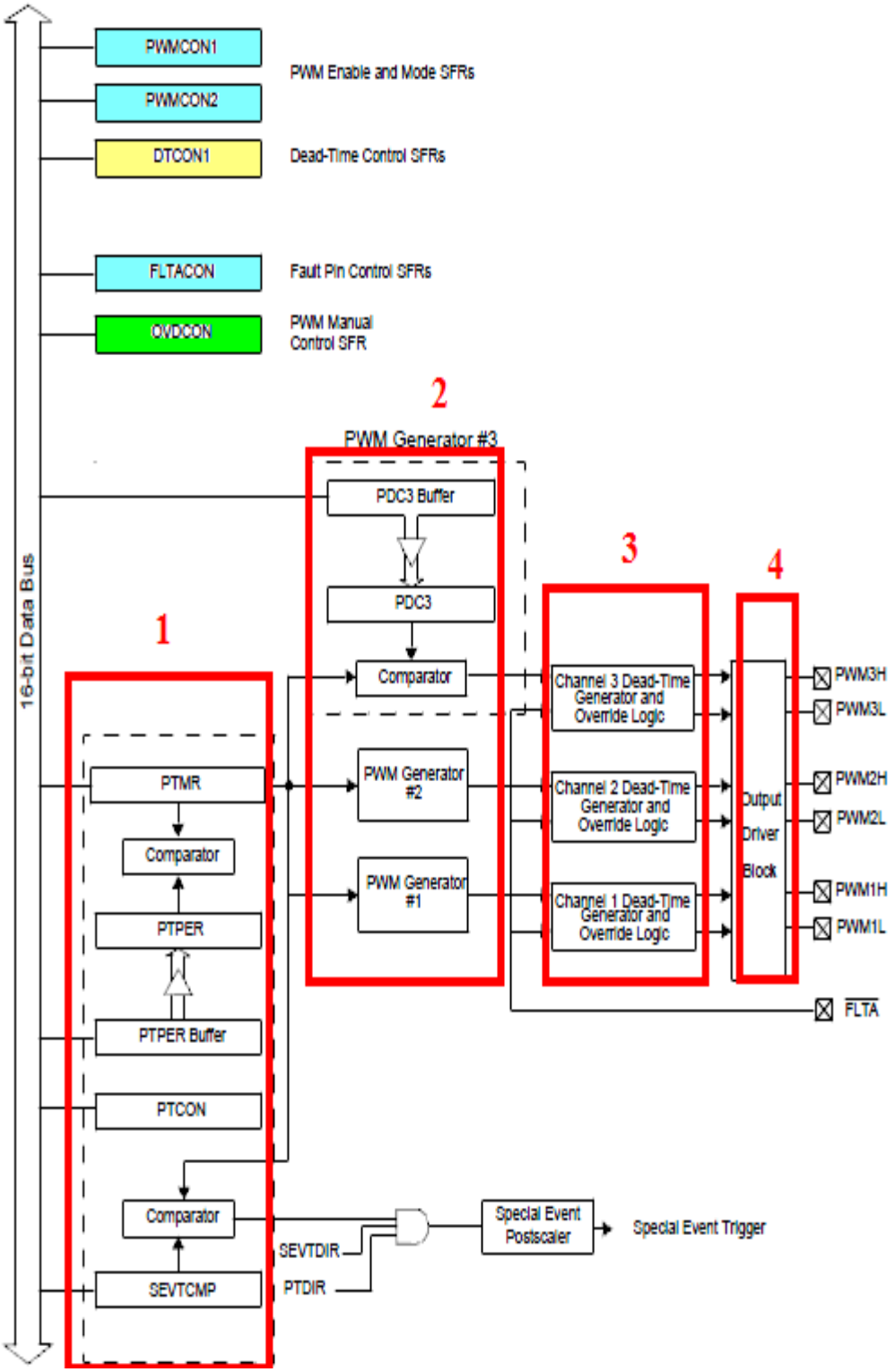
INTERRUPT SERVICE



รูป **3.42** วงจรจำลองการทำงานของโปรแกรมบริการ **EXTERNAL INTERRUPT INT0**

โมดูล MOTOR CONTROL PWM

MOTOR CONTROL PWM MODULE ถูกออกแบบมาเพื่อตอบสนองการของวิศวกรออกแบบสำหรับประยุกต์ในการสร้างสัญญานควบคุมระบบที่เกี่ยวข้องกับพลังงานเป็นหลัก อาธิเช่น สร้างสัญญาณควบคุม DC TO DC CONVERTER , DC TO AC CONVERTER และ สนับสนุนการสร้างสัญญานควบคุมชุดขับเคลื่อนมอเตอร์ **เช่น** AC INDUCTION MOTOR, Switched Reluctance (SR) Motor และ Brushless DC (BLDC) Motor เป็นต้น



รูป **3.43 Block Diagram of Motor Control PWM**

การที่เราจะใช้งานโมดูล MOTOR CONTROL PWM ได้อย่างมีประสิทธิภาพนั้นจำเป็นอย่างยิ่งที่จะต้องเข้าใจภาพรวมการทำงานของโมดูลให้มากที่สุด จากรูป() จะเห็นได้ว่ามีการแบ่งการควบคุมออกเป็น 4 ส่วนด้วยกันคือ

1 ส่วนของการกำหนดความถี่ของสัญญาณ PWM ประกอบด้วย REGISTER PTMR,PTPER,PTCON เป็นต้น

2 ส่วนของการกำหนด DUTY CYCLE คือ PDC1,PDC2 และ PDC3

3 ส่วนของการกำหนดค่า DEAD TIME และ DISABLE OR ENABLE สัญญาณ PWM

4 ส่วนของ BUFFER AND DRIVE สัญญาณ PWM

รายละเอียด REGISTER ที่ใช้ควบคุม โมดูล MOTOR CONTROL PWM

PWMCON1: PWM Control Register 1

Upper Byte:							
U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	PMOD4	PMOD3	PMOD2	PMOD1
bit 15							bit 8

Lower Byte:							
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
PEN4H	PEN3H	PEN2H	PEN1H	PEN4L	PEN3L	PEN2L	PEN1L
bit 7							bit 0

bit 15-12 **Unimplemented:** Read as '0'

bit 11-8 **PMOD4:PMOD1:** PWM I/O Pair Mode bits
1 = PWM I/O pin pair is in the independent output mode
0 = PWM I/O pin pair is in the complementary output mode

bit 7-4 **PEN4H-PEN1H:** PWMxH I/O Enable bits(1)
1 = PWMxH pin is enabled for PWM output
0 = PWMxH pin disabled. I/O pin becomes general purpose I/O

bit 3-0 **PEN4L-PEN1L:** PWMxL I/O Enable bits(1)
1 = PWMxL pin is enabled for PWM output
0 = PWMxL pin disabled. I/O pin becomes general purpose I/O

Note 1: Reset condition of the PENxH and PENxL bits depend on the value of the PWM/PIN device configuration bit in the FBORPOR Device Configuration Register.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Register 15-6: PWMCON2: PWM Control Register 2

Upper Byte:							
U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	SEVOPS<3:0>			
bit 15							bit 8

Lower Byte:							
U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
—	—	—	—	—	IUE	OSYNC	UDIS
bit 7							bit 0

bit 15-12 **Unimplemented:** Read as '0'

bit 11-8 **SEVOPS<3:0>:** PWM Special Event Trigger Output Postscale Select bits
1111 = 1:16 Postscale
•
•
0001 = 1:2 Postscale
0000 = 1:1 Postscale

bit 7-2 **Unimplemented:** Read as '0'

bit 2 **IUE:** Immediate Update Enable bit(1)
1 = Updates to the active PDC registers are immediate
0 = Updates to the active PDC registers are synchronized to the PWM time base

bit 1 **OSYNC:** Output Override Synchronization bit
1 = Output overrides via the OVDCON register are synchronized to the PWM time base
0 = Output overrides via the OVDCON register occur on next TCY boundary

bit 0 **UDIS:** PWM Update Disable bit
1 = Updates from duty cycle and period buffer registers are disabled

DTCON1: Dead Time Control Register 1

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DTBPS<1:0>		DTB<5:0>					
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DTAPS<1:0>		DTA<5:0>					
bit 7							bit 0

bit 15-14 **DTBPS<1:0>:** Dead Time Unit B Prescale Select bits
11 = Clock period for Dead Time Unit B is 8 TCY
10 = Clock period for Dead Time Unit B is 4 TCY
01 = Clock period for Dead Time Unit B is 2 TCY
00 = Clock period for Dead Time Unit B is TCY

bit 13-8 **DTB<5:0>:** Unsigned 6-bit Dead Time Value bits for Dead Time Unit B

bit 7-6 **DTAPS<1:0>:** Dead Time Unit A Prescale Select bits
11 = Clock period for Dead Time Unit A is 8 TCY
10 = Clock period for Dead Time Unit A is 4 TCY
01 = Clock period for Dead Time Unit A is 2 TCY
00 = Clock period for Dead Time Unit A is TCY

bit 5-0 **DTA<5:0>:** Unsigned 6-bit Dead Time Value bits for Dead Time Unit A

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

FLTACON: Fault A Control Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
FAOV4H	FAOV4L	FAOV3H	FAOV3L	FAOV2H	FAOV2L	FAOV1H	FAOV1L
bit 15							bit 8

Lower Byte:							
R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
FLTAM	—	—	—	FAEN4	FAEN3	FAEN2	FAEN1
bit 7							bit 0

bit 15-8 **FAOV4H-FAOV1L:** Fault Input A PWM Override Value bits
1 = The PWM output pin is driven ACTIVE on an external fault input event
0 = The PWM output pin is driven INACTIVE on an external fault input event

bit 7 **FLTAM:** Fault A Mode bit
1 = The Fault A input pin functions in the cycle-by-cycle mode
0 = The Fault A input pin latches all control pins to the programmed states in FLTACON<15:8>

bit 6-4 **Unimplemented:** Read as '0'

bit 3 **FAEN4:** Fault Input A Enable bit
1 = PWM4H/PWM4L pin pair is controlled by Fault Input A
0 = PWM4H/PWM4L pin pair is not controlled by Fault Input A

bit 2 **FAEN3:** Fault Input A Enable bit
1 = PWM3H/PWM3L pin pair is controlled by Fault Input A
0 = PWM3H/PWM3L pin pair is not controlled by Fault Input A

bit 1 **FAEN2:** Fault Input A Enable bit
1 = PWM2H/PWM2L pin pair is controlled by Fault Input A
0 = PWM2H/PWM2L pin pair is not controlled by Fault Input A

bit 0 **FAEN1:** Fault Input A Enable bit
1 = PWM1H/PWM1L pin pair is controlled by Fault Input A
0 = PWM1H/PWM1L pin pair is not controlled by Fault Input A

FLTBCON: Fault B Control Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
FBOV4H	FBOV4L	FBOV3H	FBOV3L	FBOV2H	FBOV2L	FBOV1H	FBOV1L
bit 15							bit 8

Lower Byte:							
R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
FLTBM	—	—	—	FBEN4	FBEN3	FBEN2	FBEN1
bit 7							bit 0

bit 15-8 **FBOV4H:FBOV1L:** Fault Input B PWM Override Value bits
1 = The PWM output pin is driven ACTIVE on an external fault input event
0 = The PWM output pin is driven INACTIVE on an external fault input event

bit 7 **FLTBM:** Fault B Mode bit
1 = The Fault B input pin functions in the cycle-by-cycle mode
0 = The Fault B input pin latches all control pins to the programmed states in FLTBCON<15:8>

bit 6-4 **Unimplemented:** Read as '0'

bit 3 **FAEN4:** Fault Input B Enable bit(1)
1 = PWM4H/PWM4L pin pair is controlled by Fault Input B
0 = PWM4H/PWM4L pin pair is not controlled by Fault Input B

bit 2 **FAEN3:** Fault Input B Enable bit(1)
1 = PWM3H/PWM3L pin pair is controlled by Fault Input B
0 = PWM3H/PWM3L pin pair is not controlled by Fault Input B

bit 1 **FAEN2:** Fault Input B Enable bit(1)
1 = PWM2H/PWM2L pin pair is controlled by Fault Input B
0 = PWM2H/PWM2L pin pair is not controlled by Fault Input B

bit 0 **FAEN1:** Fault Input B Enable bit(1)
1 = PWM1H/PWM1L pin pair is controlled by Fault Input B
0 = PWM1H/PWM1L pin pair is not controlled by Fault Input B

Note 1: Fault pin A has priority over Fault pin B, if enabled.

OVDCON: Override Control Register

Upper Byte:							
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
POVD4H	POVD4L	POVD3H	POVD3L	POVD2H	POVD2L	POVD1H	POVD1L
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
POUT4H	POUT4L	POUT3H	POUT3L	POUT2H	POUT2L	POUT1H	POUT1L
bit 7							bit 0

bit 15-8 **POVD4H-POVD1L:** PWM Output Override bits
1 = Output on PWMxx I/O pin is controlled by the PWM generator
0 = Output on PWMxx I/O pin is controlled by the value in the corresponding POUTxx bit

bit 7-0 **POUT4H-POUT1L:** PWM Manual Output bits
1 = PWMxx I/O pin is driven ACTIVE when the corresponding POVDxx bit is cleared
0 = PWMxx I/O pin is driven INACTIVE when the corresponding POVDxx bit is cleared

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

PWM Duty Cycle Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PWM Duty Cycle #1 bits 15-8							
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PWM Duty Cycle #1 bits 7-0							
bit 7							bit 0

PDC2: PWM Duty Cycle Register 2

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PWM Duty Cycle #2 bits 15-8							
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PWM Duty Cycle #2 bits 7-0							
bit 7							bit 0

Register 15-14: PDC3: PWM Duty Cycle Register 3

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PWM Duty Cycle #3 bits 15-8							
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PWM Duty Cycle #3 bits 7-0							
bit 7							bit 0

bit 15-0 **(PDC1 PD2 PDC3)** i<15:0>: PWM Duty Cycle Value

PTCON: PWM Time Base Control Register

Upper Byte:							
R/W-0	U-0	R/W-0	U-0	U-0	U-0	U-0	U-0
PTEN	—	PTSIDL	—	—	—	—	—
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PTOPS<3:0>				PTCKPS<1:0>		PTMOD<1:0>	
bit 7							bit 0

bit 15 **PTEN:** PWM Time Base Timer Enable bit
1 = PWM time base is ON
0 = PWM time base is OFF

bit 14 **Unimplemented:** Read as '0'

bit 13 **PTSIDL:** PWM Time Base Stop in Idle Mode bit
1 = PWM time base halts in CPU Idle mode
0 = PWM time base runs in CPU Idle mode

bit 12-8 **Unimplemented:** Read as '0'

bit 7-4 **PTOPS<3:0>:** PWM Time Base Output Postscale Select bits
1111 = 1:16 Postscale
•
•
0001 = 1:2 Postscale
0000 = 1:1 Postscale

bit 3-2 **PTCKPS<1:0>:** PWM Time Base Input Clock Prescale Select bits
11 = PWM time base input clock period is 64 TCY (1:64 prescale)
10 = PWM time base input clock period is 16 TCY (1:16 prescale)
01 = PWM time base input clock period is 4 TCY (1:4 prescale)
00 = PWM time base input clock period is TCY (1:1 prescale)

bit 1-0 **PTMOD<1:0>:** PWM Time Base Mode Select bits
11 = PWM time base operates in a continuous up/down mode with interrupts for double PWM updates
10 = PWM time base operates in a continuous up/down counting mode
01 = PWM time base operates in single event mode
00 = PWM time base operates in a free running mode

PTMR: PWM Time Base Register

Upper Byte:							
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PTDIR	PTMR <14:8>						
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PTMR <7:0>							
bit 7							bit 0

bit 15 **PTDIR:** PWM Time Base Count Direction Status bit (Read Only)
1 = PWM time base is counting down
0 = PWM time base is counting up

bit 14-0 **PTMR <14:0>:** PWM Timebase Register Count Value

Register 15-3: PTPER: PWM Time Base Period Register

Upper Byte:							
U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	PTPER <14:8>						
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PTPER <7:0>							
bit 7							bit 0

bit 15 **Unimplemented:** Read as '0'
bit 14-0 **PTPER<14:0>:** PWM Time Base Period Value bits

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Register 15-4: SEVTCMP: Special Event Compare Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SEVTDIR	SEVTCMP <14:8>						
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SEVTCMP <7:0>							
bit 7							bit 0

bit 15 **SEVTDIR:** Special Event Trigger Time Base Direction bit(1)
1 = A special event trigger will occur when the PWM time base is counting downwards.
0 = A special event trigger will occur when the PWM time base is counting upwards.

bit 14-0 **SEVTCMP <14:0>:** Special Event Compare Value bit(2)

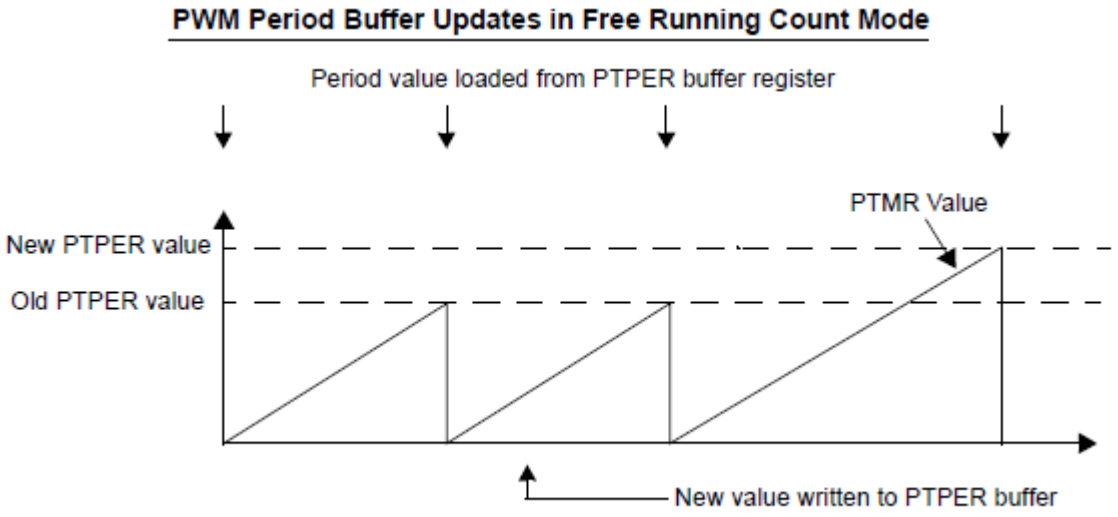
Note 1: SEVTDIR is compared with PTDIR (PTMR<15>) to generate the special event trigger.
2: SEVTCMP<14:0> is compared with PTMR<14:0> to generate the special event trigger.

ตารางแสดง ADDRESS **ของ** REGISTER **ที่เกี่ยวข้องกับ** MOTOR CONTROL PWM MODULE

REGISTER	ADDRESS
PTCON	0X01C0
PTMR	0X01C2
PTPER	0X01C4
SEVTCMP	0X01C6
PWMCON1	0X01C8
PWMCON2	0X01CA
DTCON	0X1CC
FLTA	0X1D0
FLTB	-
PDC1	0X01D6
PDC2	0X01D8
PDC3	0X01DA

การสร้างสัญญาณ PWM 6 **ช่องในโหมด** FREERUNER **และ** Continuous Up/Down Count mode

Free Running mode



รูป **3.44** แสดงความถี่ หรือ คาบเวลาของสัญญาณ PWM ที่ขึ้นอยู่กับค่าใน REGISTER PTER

การคำนวณคาบเวลาของสัญญาณ PWM ในโหมด Free Running mode

PWM Period Calculation for Free Running Count Mode (PTMOD = 10 or 11)

$$PTPER = \frac{F_{CY}}{F_{PWM} \cdot (PTMR\ Prescaler)} - 1$$

Example:

$F_{CY} = 20$ MHz
$F_{PWM} = 20{,}000$ Hz
PTMR Prescaler = 1:1

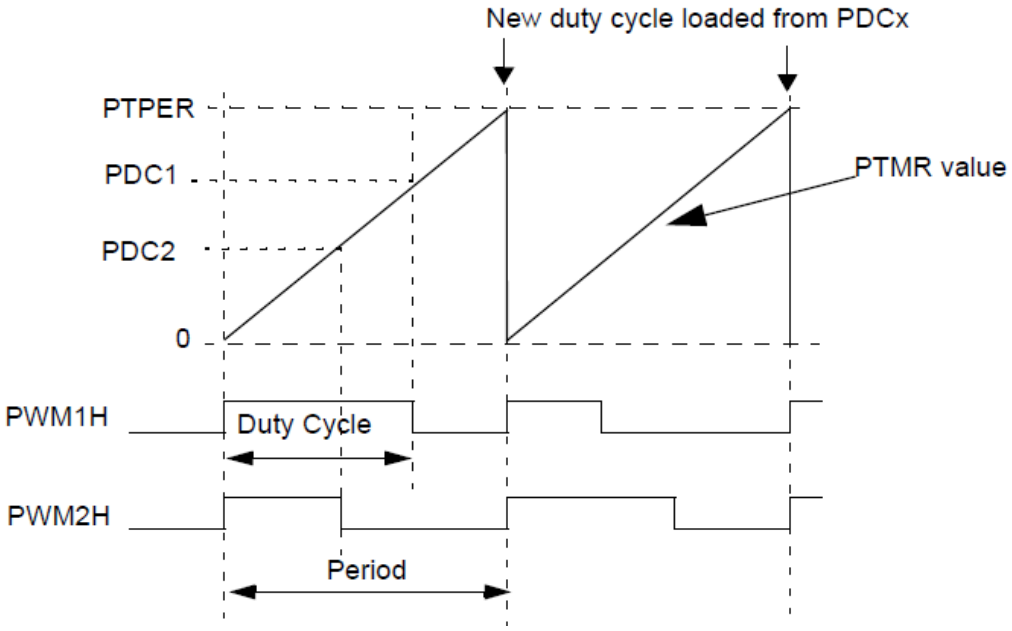
ความเร็วในการทำงานของ CPU (MACHINE CYCLE SPEED)

$$PTPER = \frac{20{,}000{,}000}{20{,}000 \cdot 1} - 1$$

$= 1000 - 1$
$= 999$

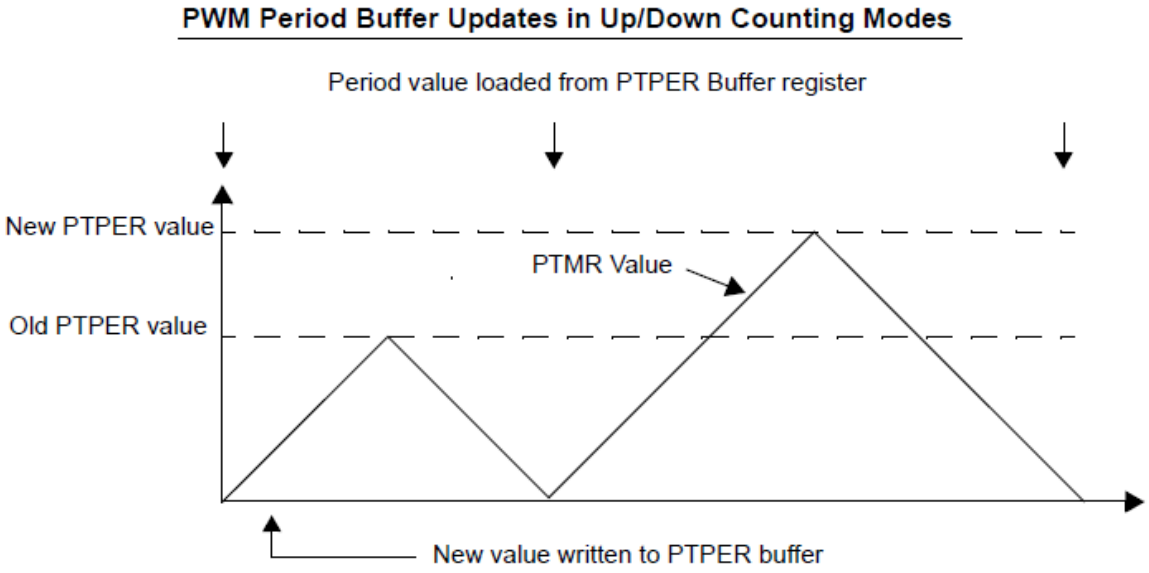
แสดงตัวอย่างการคำณวนค่าใน REGISTER PTPER

การเปลี่ยนแปลง DUTYCYCLE ของสัญญาณ PWM ในโหมด Free Running mode



รูป **3.45** แสดงความสัมพันธ์ของ REGISTER PTPER,PTMR,PDC ในการกำหนด DUTY CYCLE ของสัญญาณ PWM

Up/Down Counting Modes



รูป **3.46** แสดงความถี่ หรือ คาบเวลาของสัญญาณ PWM ที่ขึ้นอยู่กับค่าใน REGISTER PTER

PWM Period Calculation in Up/Down Counting Modes (PTMOD = 00 or 01)

$$PTPER = \frac{FCY}{FPWM \bullet (PTMR\ Prescaler) \bullet 2} - 1$$

Example:

FCY = 20 MHz

FPWM = 20,000 Hz

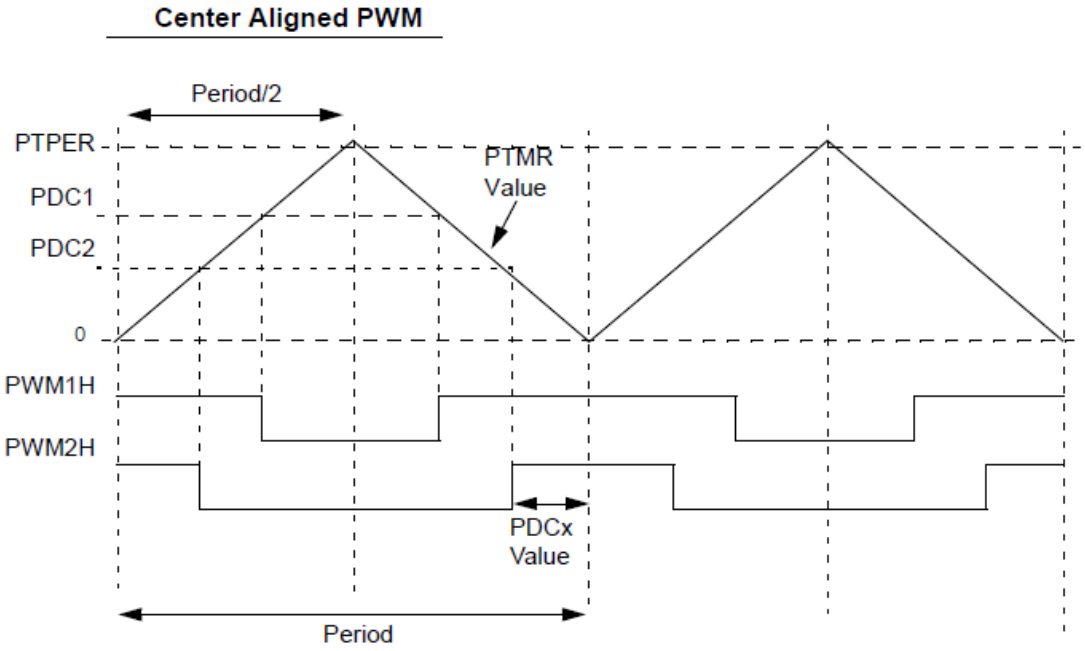
PTMR Prescaler = 1:1

$$PTPER = \frac{20{,}000{,}000}{20{,}000 \bullet 1 \bullet 2} - 1$$

$$= 500 - 1$$

$$= 499$$

แสดงตัวอย่างการคำณวนค่าใน REGISTER PTPER **Up/Down Counting Modes**



รูป **3.47** แสดงความสัมพันธ์ของ REGISTER PTPER,PTMR,PDC ในการกำหนด DUTY CYCLE ของสัญญาณ PWM

ตัวอย่างโปรแกรมการสร้างสัญญาณในโหมด FREE RUNNING และ UP/DOWN COUNTING

```
#include <30f2010.h>
#Fuses  hs,noWDT
#Fuses  BORV27
#fuses  PUT64
#fuses BROWNOUT
#FUSES MCLR
#use delay(clock=10000000)
int16 PTCON;
#locate PTCON = 0x1C0
//-----------------------------------
INT16 PTMR;
#LOCATE  PTMR = 0X1C2
INT16 PTPER;
#LOCATE PTPER = 0X1C4
INT16 SEVTCPP;
#LOCATE SEVTCPP = 0X1C6
INT16 PWMCON1;
#locate PWMCON1 = 0x1c8

INT16 PWMCON2;
#locate PWMCON2 = 0x1ca
INT16 DTCON1;
#LOCATE DTCON1 = 0X1CC
INT16 FLTACON;
```

```
#LOCATE FLTACON = 0X1D0
INT16 OVDCON;
#LOCATE OVDCON  =  0X1D4
INT16 PDC1;
#LOCATE PDC1 = 0X1D6
INT16 PDC2;
#LOCATE PDC2 = 0X1D8
INT16 PDC3;
#LOCATE PDC3 = 0X1DA

void main(void)
{
  ptper=0x007D;
  ptmr=0x1ff;
  PWMCON1=0X0FFF;
  PWMCON2=0X0002;
  PTCON=0X8000;
  OVDCON=0XFF00;
  PDC1=PDC2=PDC3=100;

  while(true)
  {

  }
}
```

จากโปรแกรมดังกล่าวเป็นชุดคำสั่งขั้นพื้นฐานในการสร้างสัญญาณ **PWM** สิ่งที่ต้องเรียนรู้จากโปรแกรมก็คือต้องเข้าใจความหมายของการกำหนดค่าลงใน **REGISTER** ที เกี่ยวข้องกับการสร้างสัญญาณ **PWM** ทั้งหมด เช่น **PWMCON1 = 0X0FFF**

PWMCON1: PWM Control Register 1

Upper Byte:

U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	PMOD4	PMOD3	PMOD2	PMOD1
0	0	0	0	1	1	1	1

BIT 15 ← **BIT 8**

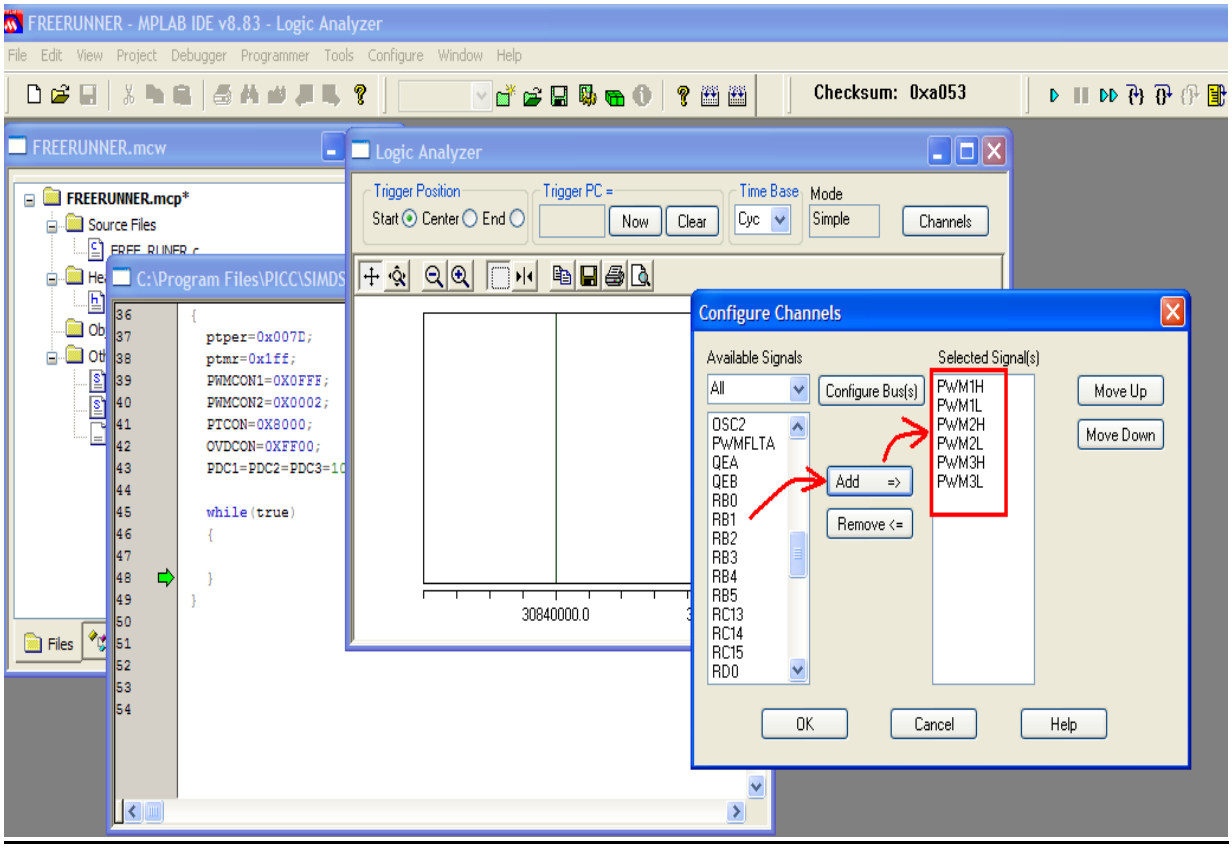
Lower Byte:

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
PEN4H	PEN3H	PEN2H	PEN1H	PEN4L	PEN3L	PEN2L	PEN1L
1	1	1	1	1	1	1	1

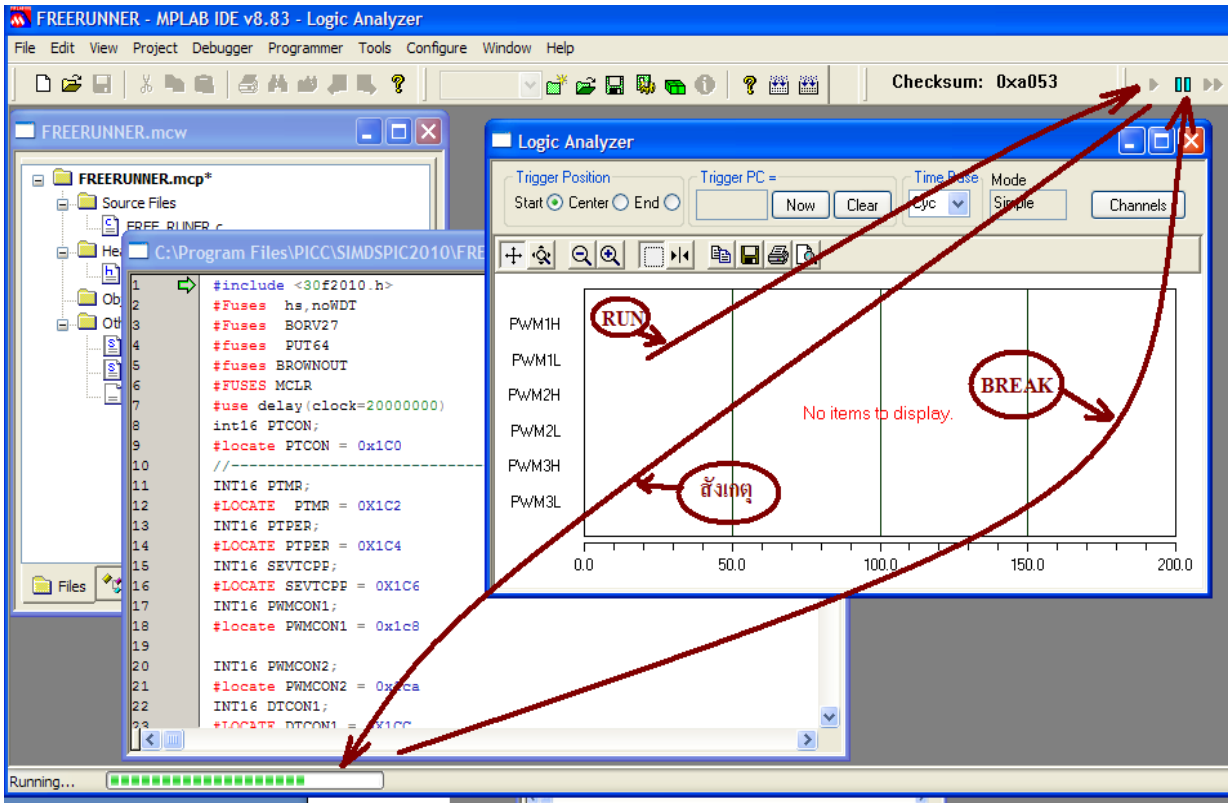
BIT 7 ← **BIT 0**

รูป **3.48** แสดงค่าแต่ละ **bit** ของ REGISTER PWMCON1

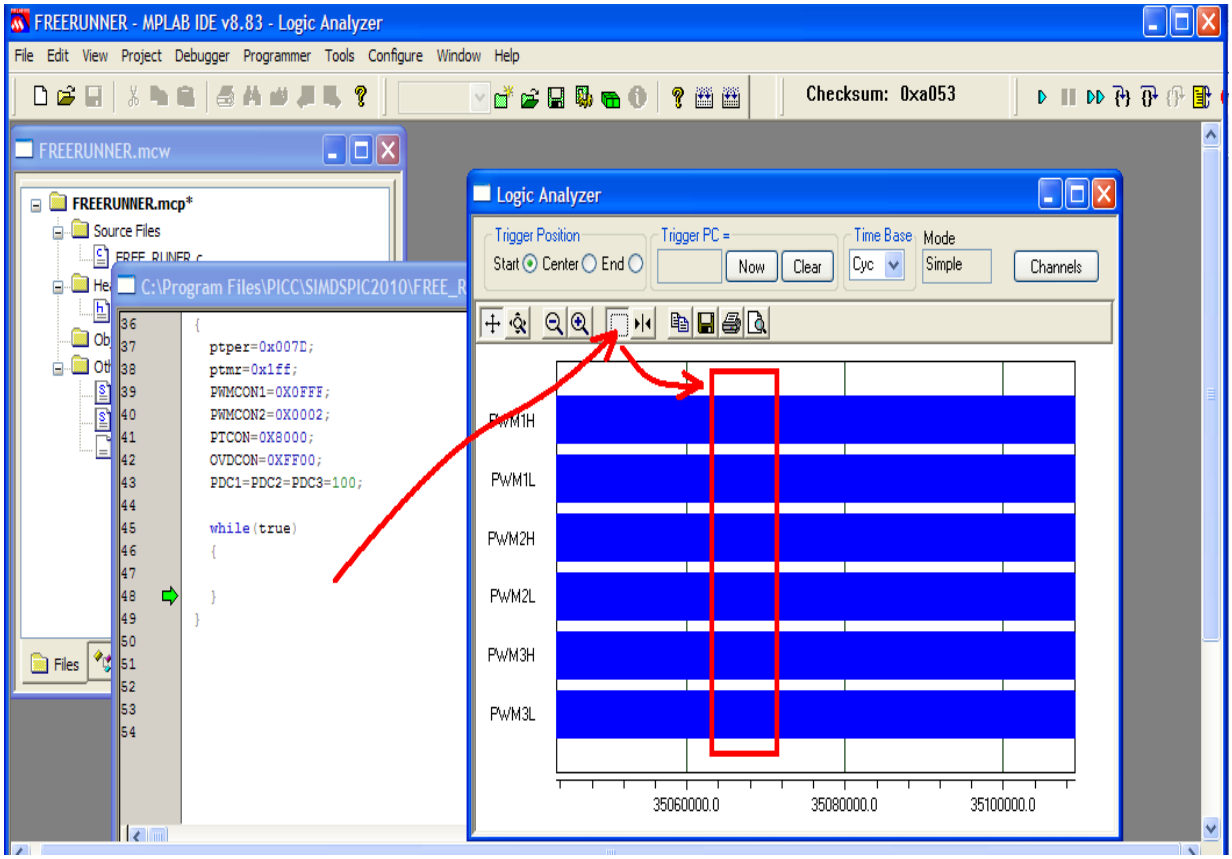
ใช้ LOGIC ANALYZER ในโปรแกรม MPLAB IDE ตรวจสอบและเรียนรู้ค่าใน REGISTER



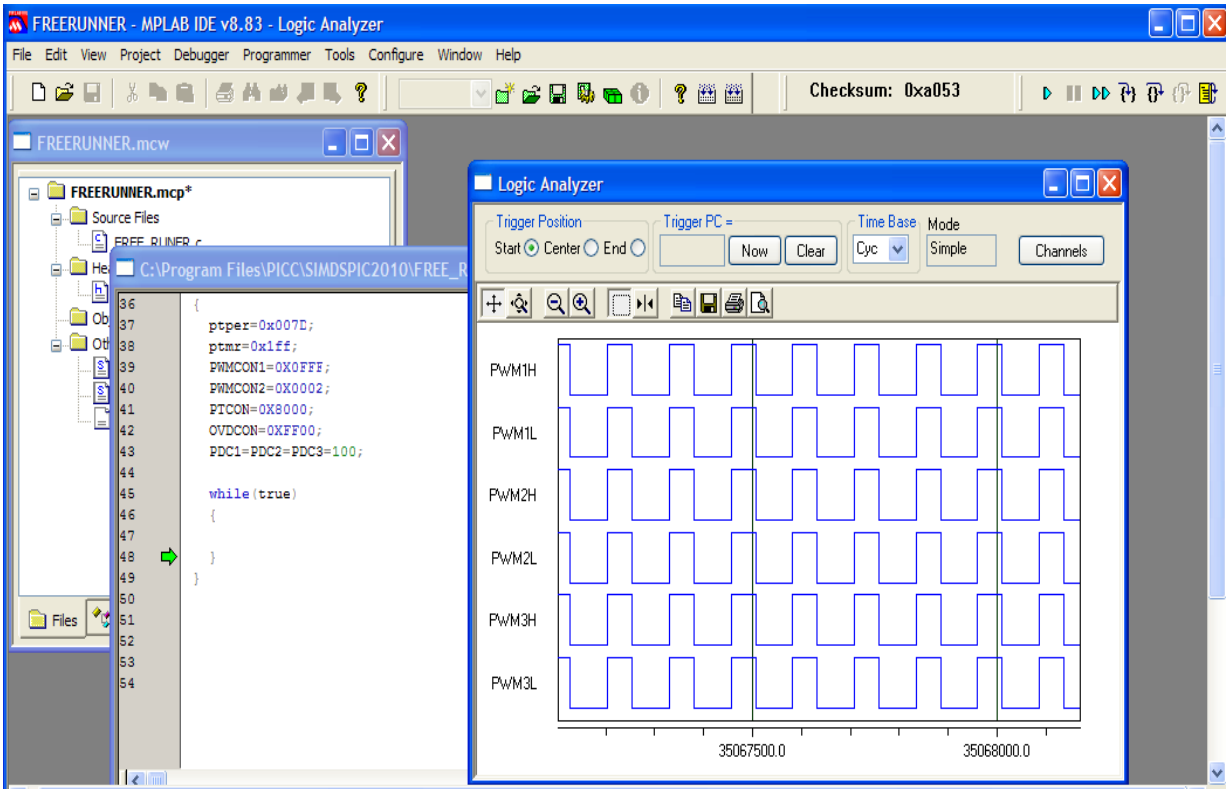
รูป **3.49** ADD PIN PWM1H,PWM1L,PWM2H,PWM2L,PWM3H,PWM3L



รูป **3.50** Click Run สังเกต Progress bar ประมาณ 2 รอบแล้ว CLICK HALT

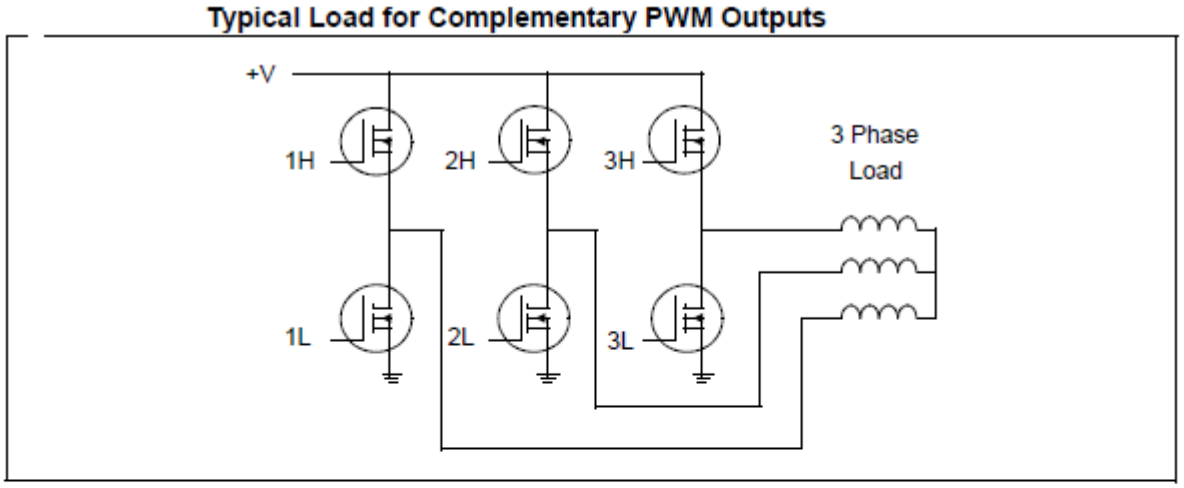


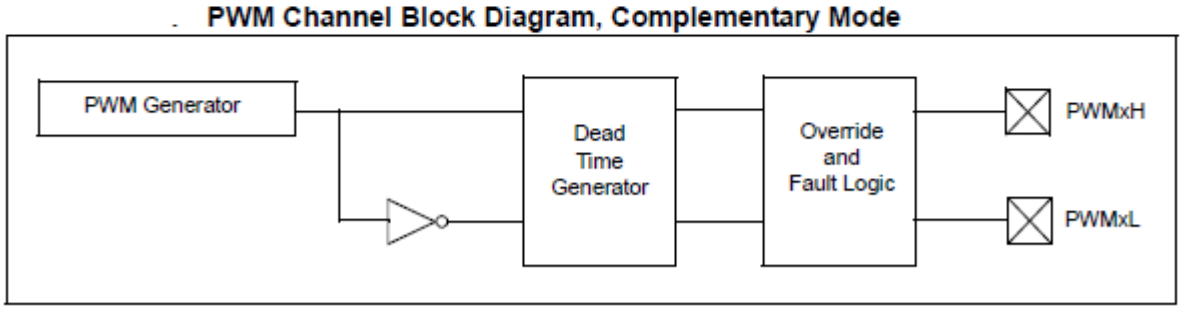
รูป **3.51** ขยาย TIME BASE



รูป **3.51** สัญญาณ PWM 6 CHANEL แกนเวลามีหน่วยเป็น MACHINE CYCLE

Complementary AND Independent output





คู่สัญญาณ PWMxH กับ PWMxL

BIT ที่ใช้ควบคุมการทำงานของคู่สัญญาณ PWM คือ PMOD1,PMOD2,PMOD3 ซึ่งจะคู่กับ PWM1,PWM2,PWM3 ตามลำดับ

ตัวอย่างการทำให้ คู่ของสัญญาณ PWM1 เป็น Complementary

PWMCON1: PWM Control Register 1

Upper Byte:

U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	PMOD4	PMOD3	PMOD2	PMOD1
0	0	0	0	1	1	1	0

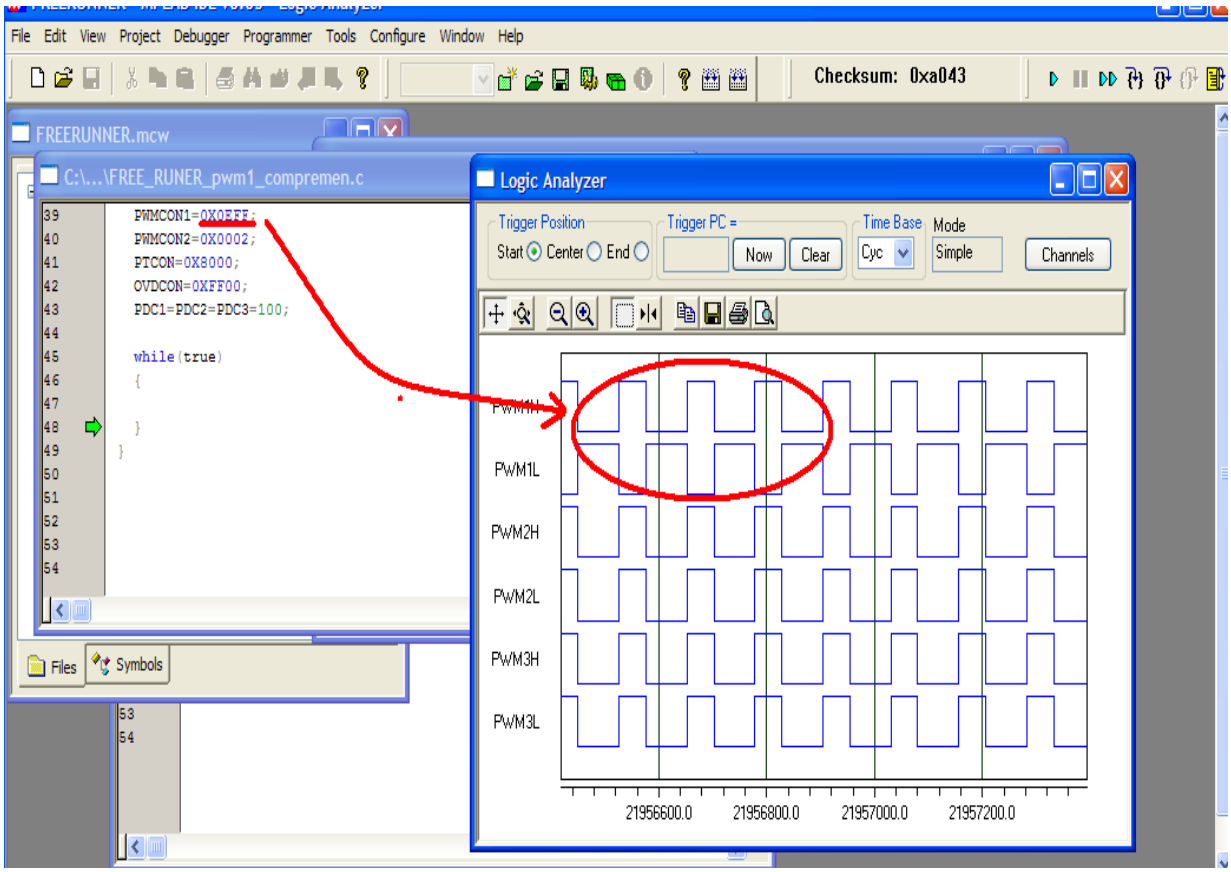
BIT 15 ← BIT 8

Lower Byte:

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
PEN4H	PEN3H	PEN2H	PEN1H	PEN4L	PEN3L	PEN2L	PEN1L
1	1	1	1	1	1	1	1

BIT 7 ← BIT 0

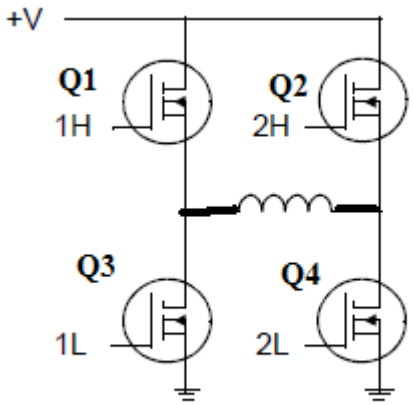
รูป **3.53** PMOD1 = 0 PWM1 ทำงานเป็น Complementary



รูป **3.54** คู่ PWM1 เป็น Complementary

<u>การกำหนดค่า DEAD TIME ให้กับคู่สัญญาณ PWM ที่เป็น Complementary</u>

คู่สัญญาณ PWM ที่เป็น Complementary_ จำเป็นที่จะต้องพิจารณาในเรื่องของค่า DEAD TIME เนื่องจากข้อจำกัดทางเวลา TURN OFF ของอุปกรณ์ ELECTRONIC SWITCHING ที่อยู่ในวงจรดังรูปข้างล่าง



รูป **3.55** วงจร SWITCHING ที่จำเป็นต้องมีค่า DEAD TIME

ตัวอย่างโปรแกรมในการกำหนดค่า DEAD TIME

```
void main(void)
{
 ptper=0x007D;
 ptmr=0x1ff;
 PWMCON1=0X00FF;
 PWMCON2=0X0002;
 PTCON=0X8000;
 OVDCON=0XFF00;
 PDC1=PDC2=PDC3=100;
 DTCON1=0X00FF;

 while(true)
 {

 }
}
```

กำหนดค่า DEAD TIME

Dead Time Calculation

$$DT = \frac{\text{Dead Time}}{\text{Prescale Value} \bullet T_{CY}}$$

DT (Dead Time) is the DTA<5:0> or DTB<5:0> register value.

การคำณวณค่า DEAD TIME

จากโปรแกรม DTCON1=0X00FF;

DTCON1: Dead Time Control Register 1

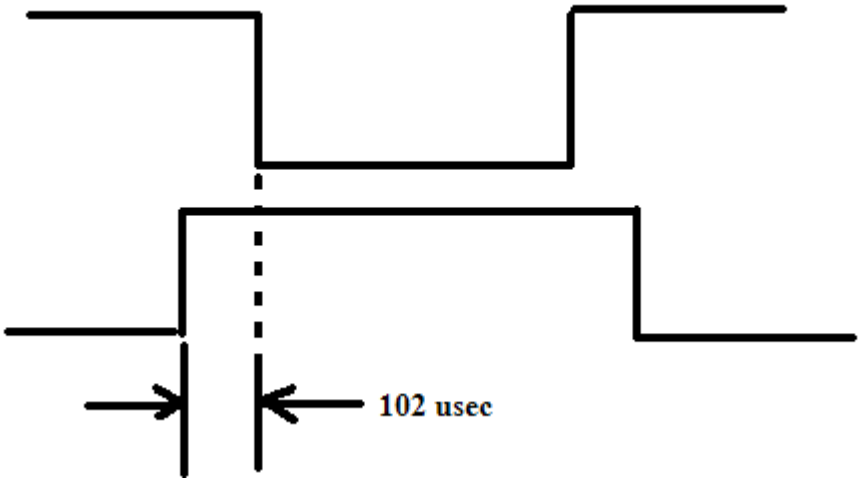
Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
0	0	0	0	0	0	0	0

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
1	1	1	1	1	1	1	1

รูป **3.56** ค่าใน REGISTER DTCON1

DT = 111111b = 64d ; Prescale = 8 ; TCY = 20000000/4 = 5000000

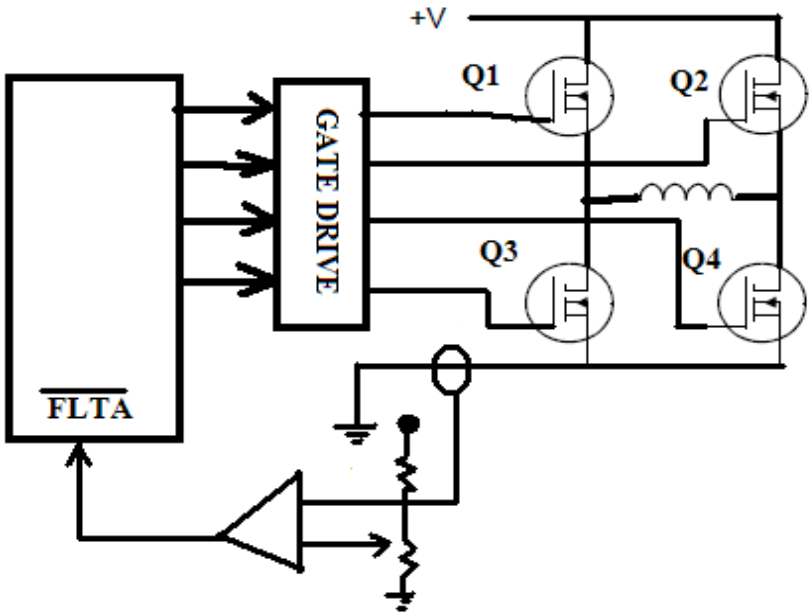
Dead Time = (1/5000000) x 8 x 64 = 102 usec



รูป **3.57** แสดงค่า DEAD Time

<u>การใช้ INPUT FAULT PINS ในการป้องกันความผิดปกติในภาคขยายกำลังไฟฟ้า</u>

INPUT FAULT PINS เป็นที่ใช้รองรับการพัฒนาโปรแกรมในการตรวจสอบความผิดปกติในส่วนของภาคขยายกำลัง เช่น กระแสเกินพิกัด หรือ เกิดการลัดวงจร เป็นต้น



รูป **3.58** การใช้งาน FAULT PIN

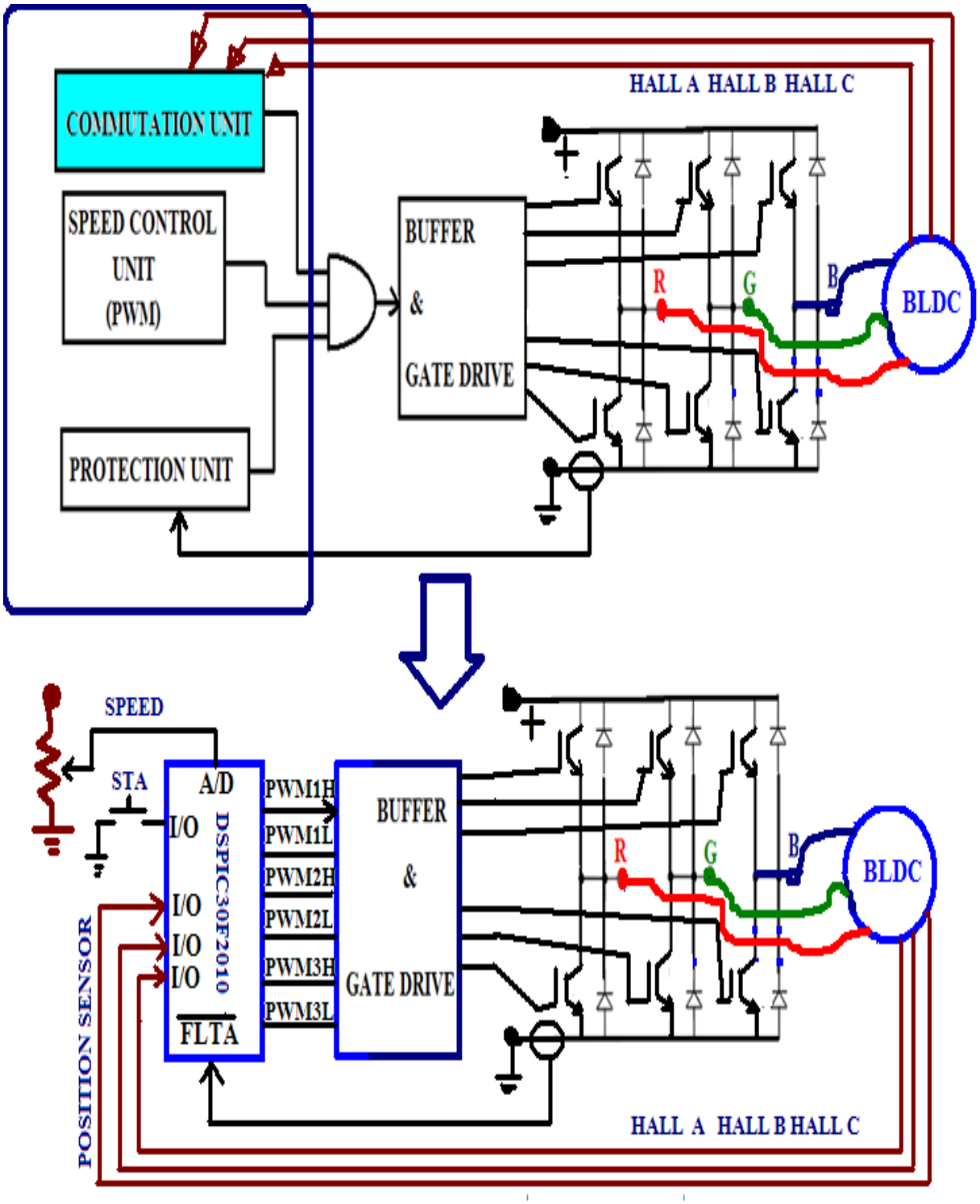
ตัวอย่างโปรแกรม การใช้งาน FAULT PIN

```
#include <30f2010.h>
#device adc=8
#Fuses XT_PLL16
#Fuses  BORV27
#fuses  PUT64
#fuses BROWNOUT
#FUSES MCLR
#use delay(clock=160000000,restart_wdt)
#use rs232(UART1,baud=19200,parity=N,bits=8,)

int16 PTCON;
#locate PTCON = 0x1C0
INT16 PTMR;
#LOCATE  PTMR = 0X1C2
INT16 PTPER;
#LOCATE PTPER = 0X1C4
INT16 SEVTCPP;
#LOCATE SEVTCPP = 0X1C6
INT16 PWMCON1;
#locate PWMCON1 = 0x1c8
INT16 PWMCON2;
#locate PWMCON2 = 0x1ca
INT16 DTCON1;
#LOCATE DTCON1 = 0X1CC
INT16 FLTACON;
#LOCATE FLTACON = 0X1D0
INT16 OVDCON;
#LOCATE OVDCON  =  0X1D4
INT16 PDC1;
#LOCATE PDC1 = 0X1D6
INT16 PDC2;
#LOCATE PDC2 = 0X1D8
INT16 PDC3;
#LOCATE PDC3 = 0X1DA
```

```
void main(void)
{
 ptper=0x0080;
 PWMCON2=0X0F00;
 PWMCON1=0X00FF;
 PWMCON2=0X0F00;
 PTCON=0X8000;
 OVDCON=0XFF00;
 PDC1=PDC2=PDC3=0;
 FLTACON=0x000F;

 while(true)
 {
  restart_wdt();
  }
}
```

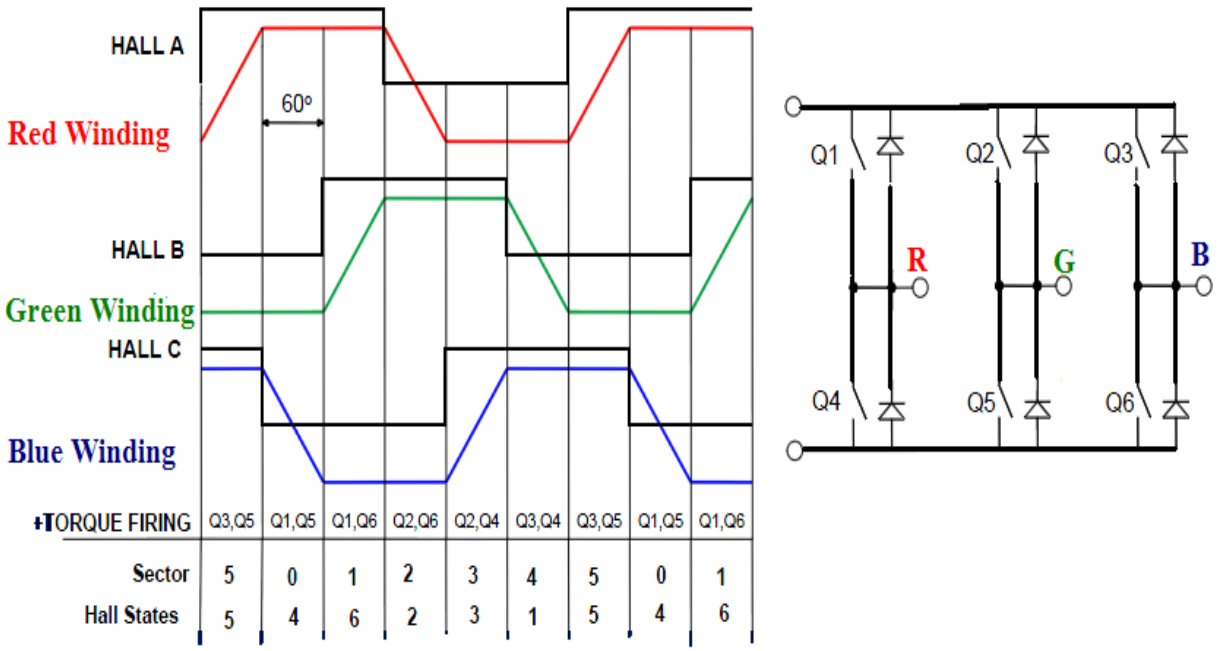
<u>การสร้างชุดควบคุม BRUSHLESS DC MOTOR แบบลูปเปิดด้วย DSPIC30F2010</u>

ดังที่ได้กล่าวมาแล้วในข้างต้นว่าระบบควบคุมแบบสมองกลฝังตัวนั้นมีข้อได้เปรียบกว่าระบบควบคุมแบบเดิม ๆ ที่มีส่วนของ HARDWARE เพียงอย่างเดียวมาก เนื่องจากมีความยืดหยุ่นสูง ทั้งในการตรวจสอบข้อบกพร่องของระบบ และ การพัฒนาปรับปรุงในวันข้างหน้า อีกทั้งสนับสนุนการติดต่อสื่อสารกับอุปกรณ์ภายนอก เช่น คอมพิวเตอร์ หรือ อุปกรณ์สื่อสาร อื่น ๆ อีกมาก เนื่องจากองค์ประกอบของ DSPIC30F2010 มีส่วนของโมดูลติดต่อสื่อสารกับอุปกรณ์อยู่หลายอย่าง เช่น UART1, UART2,SPI1,SPI1,CAN1 และ CAN2 เป็นต้น อีกทั้งการใช้ DSPIC2010 เป็นการยุบรวม BLOCK ต่าง ๆ ที่ใช้การควบคุมแบบลอจิกเกตุ ซึ่งจะเปรียบเทียบให้เห็นดังรูป



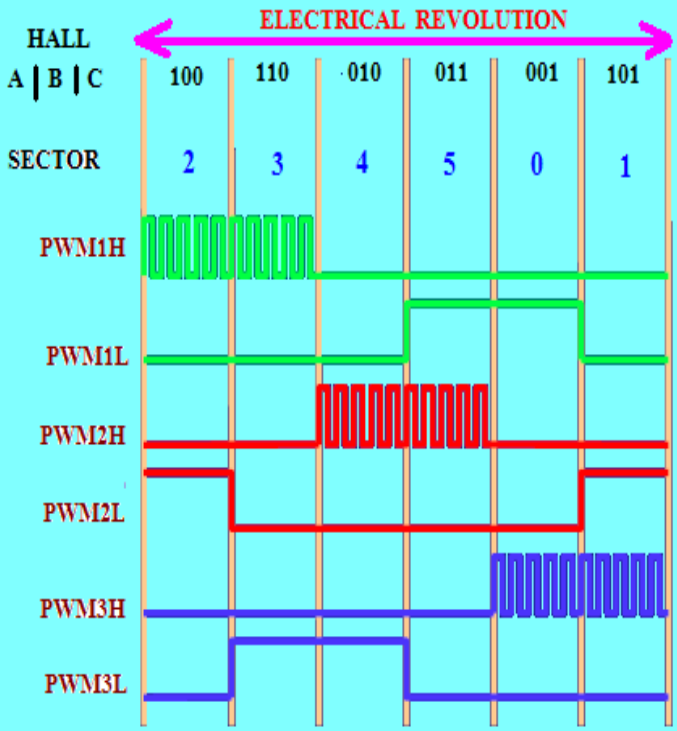
รูป **3.59** เปรียบ Block Diagram การสร้างชุดควบคุม Brushless Dc Motor ระบบดิจิตอลด้วนลอจิกเกตุกับระบบสมองกลฝังตัวโดยใช้ DSPIC30F2010 เป็นหน่วยประเมินผลกลาง

กระบวนการ COMMUTATION กับความสัมพันธ์ของค่าใน REGISTER OVDCON



รูป **3.60**

HALL A|B|C	OVDCON VALUE	
	POVD<15:8>	POUT< 7 : 0 >
000	00000000	00000000
001	00100000	00000001
010	00001000	00010000
011	00001000	00000001
100	00000010	00000100
101	00100000	00000100
110	00000010	00010000
111	00000000	00000000



รูป **3.61** SIXSTEP OVDCON COMMUTATION

โปรแกรมควบคุมการขับเคลื่อน BrushLess Dc Motor

```
//Hall effect PIN_B3-->Blue:PIN_b4-->Green:PIN_b5-->Yellow
#include <30F2010.h>
#device adc=8
#Fuses XT_PLL16
#Fuses  BROWNOUT
#Fuses  noMCLR
#Fuses  PUT4
#Fuses  NOWDT
#fuses WPSB10
#use delay(clock=160000000,restart_wdt)
#use rs232(UART1,baud=19200,parity=N,bits=8,)
//#FUSES HPOL_low //High.Side Transistors Polarity is Active.High (PWM 1,3,5 and 7)
//#FUSES LPOL_low //Low.Side Transistors Polarity is Active.Low (PWM 0,2,4 and 6)

int16 trisB;
#locate trisB=0x02C6
#bit    trisb3=trisb.3
#bit    trisb4=trisb.4
#bit    trisb5=trisb.5
struct sensor{
   INT NON  :3;
   int  data :3;
}hall_data;
#locate hall_data=0x02C8

int16 PTCON;
#locate PTCON = 0x1C0
INT16 PTMR;
#LOCATE  PTMR = 0X1C2
```

```
INT16 PTPER;
#LOCATE PTPER = 0X1C4
INT16 SEVTCMP;
#LOCATE SEVTCMP = 0X1C6
INT16 PWMCON1;
#locate PWMCON1 = 0x1c8
INT16 PWMCON2;
#locate PWMCON2 = 0x1ca
INT16 DTCON1;
#LOCATE DTCON1 = 0X1CC
int16 ifs2;
#locate ifs2=0x088
INT16 FLTACON;
#LOCATE FLTACON = 0X1D0
INT16 OVDCON;
#LOCATE OVDCON  =  0X1D4
INT16 PDC1;
#LOCATE PDC1 = 0X1D6
INT16 PDC2;
#LOCATE PDC2 = 0X1D8
INT16 PDC3;
#LOCATE PDC3 = 0X1DA
INT16   CONST TABLE_FW[]={0x0000,0x2001,0x0810,0x0801,0x0204,0x2004,0x0210};
INT16   CONST TABLE_RW[]={0x0000,0x0210,0x2004,0x0204,0x0801,0x0810,0x2001};
INT8 INDEX,TEMPINDEX;
int16 n;
int16 duty;


#int_TIMER1
void  TIMER1_isr(void)
{set_timer1(500);
set_adc_channel( 0 );
duty = read_adc();
pdc1= pdc2= pdc3=duty;
clear_interrupt(int_timer1);
}
```

```
void main(void)
{
 setup_wdt (WDT_OFF);
 trisb3=trisb4=trisb5=1;
 ptper=0x0080;
 PTCON=0X8000;
 SEVTCMP=ptper;
 PWMCON1=0X0FFF;
 PWMCON2=0X0F00;
 PDC1=PDC2=PDC3=0;  OVDCON=TABLE_FW[0];
 INDEX=0;
 SETUP_ADC_PORTS(sAN0|VSS_VDD );
 SETUP_ADC(ADC_CLOCK_INTERNAL);
 enable_interrupts(INTR_GLOBAL);
 ENable_interrupts(INT_TIMER1);
 setup_timer1(TMR_INTERNAL|TMR_DIV_BY_8); set_timer1(500);
  n=0;duty=200;
 WHILE(N<5)
 {
  OUTPUT_TOGGLE(PIN_c14);
  DELAY_MS(1000);
  N++;
 }
 OUTPUT_HIGH(PIN_c14);
 while(true)
{
 INDEX=hall_data.data;
   IF(INDEX!=TEMPINDEX)
 {
  OVDCON=TABLE_FW[INDEX];
 }
   TEMPINDEX=INDEX;
      n++;
    if(n>30000)
    {
    output_toggle(PIN_c14);    n=0;
    }
   }
}
```