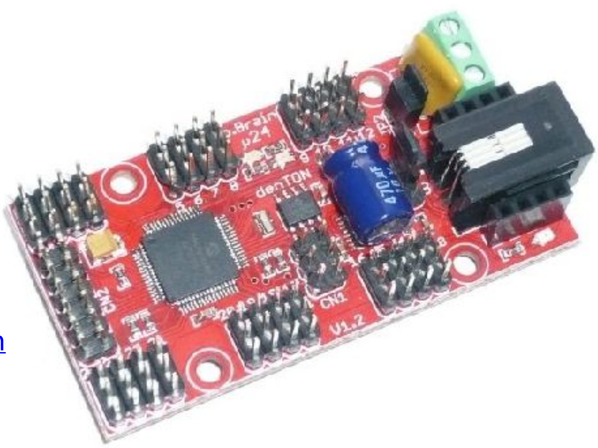
micromagic systems

Animatronic & Puppet control systems for Film & Television

p.Brain-µ24 V1.2 Robot Controller - User Guide V1.1

Preliminary (Updated 26/07/09)

Contents



This controller is designed for people who have an understanding of PIC microconrollers, associated peripherals and general PWM techniques, It is not suitable for the beginner or novice.

A Pre-Programmed version is available for controlling hexapod robots (p.Brain-µHexEngine) , or as a simple serial servo controller p.Brain-µSSC.

P.Brain-µ24 (v1.1) User Guide

Description

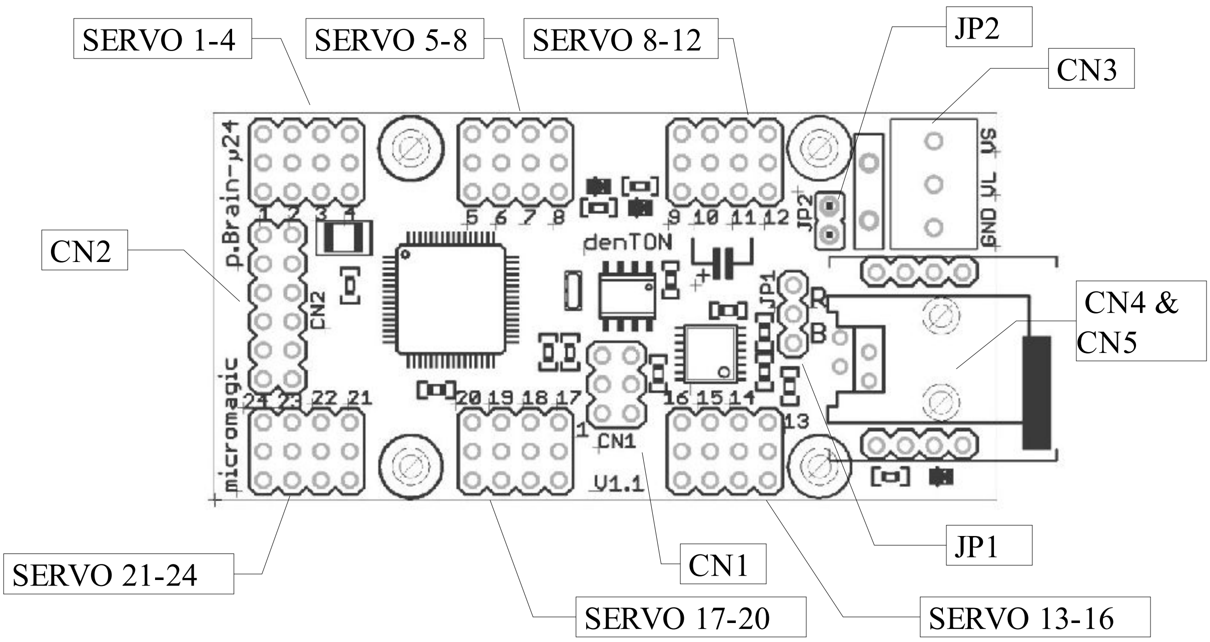
The p.Brain-µ24 has be designed for use in advanced hobby robotics, where multiple R/C type servos need to be controlled with a standard PWM signal, such as hexapod robots. At the core of the p.Brain-µ24 is a microchip dsPIC33F 16bit micro controller with 128Kb programme flash, 8Kb RAM and 32Kb external EEPROM. This micro controller can be programmed using microchips ICD2 programmer and MPLAB IDE. Although the micro controller can be programmed in assembler, I suggest using a C compiler such as Hi-techs (www.htsoft.com) All programming examples are written for the Hi-tech compiler.

The p.Brain-µ24 has 24 PWM channels for R/C servo control, along with a variety of other peripherals such as UART's, I2C, ADC and Digital I/O. Unlike the p.Brain-ds24, the p.brain-µ24 does not require a motherboard for operation, all 24 PWM signals are routed to servo connector pins with power supplied from a terminal block.

Key Features

- Compact size (approx 68 x 35 mm)
- On Board 3.3V regulator
- dsPIC33FJ128GP206 16bit, 40Mips Processor
- 128Kb Programme Flash, 8Kb RAM, 64Kb External EEPROM
- External 8Mhz Ceramic Resonator or Internal 7.37Mhz R/C with PLL to 40MIPS
- UART1, Inverted TTL
- UART2, RS232, or Bluetooth via optional ESD200 (Jumper selectable)
- I2C (Internally connected to 8Kbyte EEPROM)
- 8 x Digital IO, 6 with pull-up, or 8 x 12 bit Analogue capture (ADC)
- 2 x On Board LED's
- 24 PWM servo outputs
- ESD200 Bluetooth socket, with blue connection LED
- RJ11 4/4 RS-232 socket
- Power terminal for Logic power , Servo power and Ground
- Jumper selectable Logic power = Servo power
- Resettable Servo Power Fuse

p.Brain LAYOUT



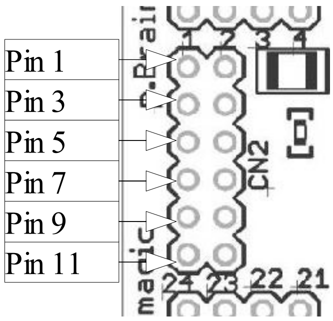
Connector CN1 (6pin 0.1" (2.54mm) pitch)

PIN	NAME	DESCRIPTION
1	SDA	I2C SDA
2	CTRL_IN	UART1 TTL Receive
3	SCL	I2C SCL
4	CTRL_OUT	UART1 TTL Transmit
5	VCC OUT	Regulated 3.3V output (100ma MAX)
6	GND	Power Ground

Connector CN2 (12pin 0.1" (2.54mm) pitch)

In Circuit Serial Programming pins are available on this connector.

PIN	NAME	DESCRIPTION
1	GND	Power Ground
2	DIG0/ANA0	Digital I/O 0 or Analogue Capture 0
3	GND	Power Ground
4	DIG1/ANA1	Digital I/O 1 or Analogue Capture 1
5	VCC OUT	Regulated 3.3V output (100ma MAX)
6	DIG2/ANA2	Digital I/O 2 or Analogue Capture 2
7	MCLR	ISSP MCLR & Reset
8	DIG3/ANA3	Digital I/O 3 or Analogue Capture 3
9	PGC	ISSP Clock, Digital I/O 6 or Analogue Capture 6 (No Pull Up)
10	DIG4/ANA4	Digital I/O 4 or Analogue Capture 4
11	PGD	ISSP Data, Digital I/O 7 or Analogue Capture 7 (No Pull Up)
12	DIG5/ANA5	Digital I/O 5 or Analogue Capture 5



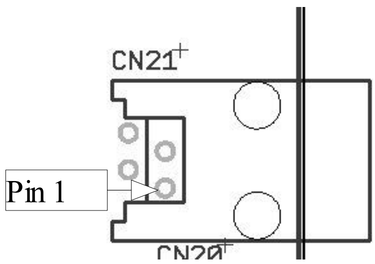
Connector CN3 (3 way screw terminal)

PIN	NAME	DESCRIPTION
1	GND	Servo & Logic Power Ground
2	VL	Logic Power Supply, 4.5 to 12 V DC
3	VS	Servo Power Supply, level depends on servos used.

Note: Servo power is fused via a polyswitch. This fuse will trip at approx 8.5Amps, and will reset when the short circuit condition has been removed. The fuse is rated at 16V max.

Connector CN4 (RJ11 4/4)

PIN	NAME	DESCRIPTION
1	TX_232	RS-232 Data out
2	RX_232	RS-232 Data in
3	GND	Ground
4	N/C	Not Connected



An RS-232 RJ11 to DB9 cable is available separately (p.Brain-RJ232) or you can make your own serial lead using the following wiring:

RJ11 to PC DB9 (Female)

RJ11 Pin	Name	DB9 Pin
1	p.Brain-µ24 TX	3
2	p.Brain-µ24 RX	2
3	GND	5

Connector CN5 (2 x 4 way 0.1" pitch sockets for ESD200)

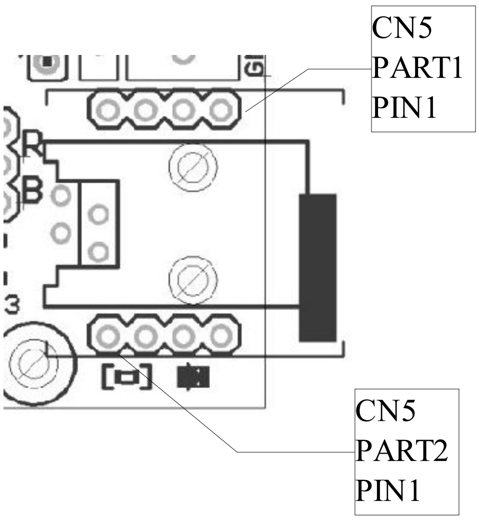
CN5 comprises of two connectors for fitting the ESD200 bluetotth module.

CN5 Part1

PIN	NAME	DESCRIPTION
1	GND	Power Ground
2	VCC	3.3V Power
3	Connected	Connection Status
4	RESET	Reset Config Defaults

CN5 Part2

PIN	NAME	DESCRIPTION
1	GND	Power Ground
2	N/C	Not Connected
3	RX	Data out
4	TX	Data in



Jumper JP1 (3pin 0.1" (2.54mm) pitch)

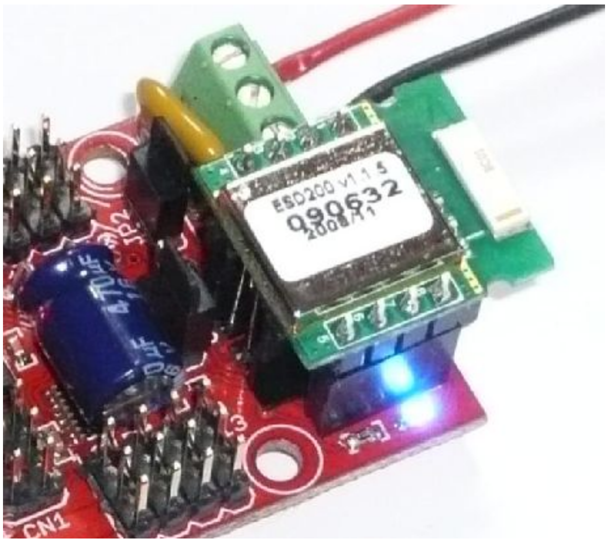
This jumper selects between RS232 or ESD200 bluetooth input to UART2 RX. The PCB is indicated with a 'B' and 'R' position for the jumper: B = bluetooth, R = RS232. The output transmit of UART2 is always routed to both RS-232 and the ESD200 socket. This means that if you are using the p.Brain-µ24 configured in bluetooth mode JP1=B, you could plug in an RJ-232 adaptor lead to a serial port to monitor the data being sent to the ESD200, or visa versa.

Jumper JP2 (2pin 0.1" (2.54mm) pitch)

This jumper connects the fused side of the servo power supply VS to the logic power supply VL. This allows the processor to run from the same power as the servos, however, in some cases power fluctuations on the servo power source may cause the processor to reset.

ESD200 Installation & Configuration

Ensure the power to the p.Brain-µ24 is switched off, and insert the ESD200 into the socket as shown in the image.



Before you configure the ESD200 device, you may need to change the desired baud rate, this defaults to 115200. If you require a different baud rate, you will need to use the RJ-232 adaptor and configure the HexEngine TBR setting before proceeding. *Note, the baud rate into the ESD200 does not have to match the baud rate on the host PC bluetooth adaptor.*

Although not essential, if you have an RJ-232 adaptor lead, it is advised to plug this in during ESD200 configuration so that the progress and any configuration errors can be monitored. Connect the RJ-232 lead from the RJ11 port to a free serial port on your host PC, and start up your terminal software.

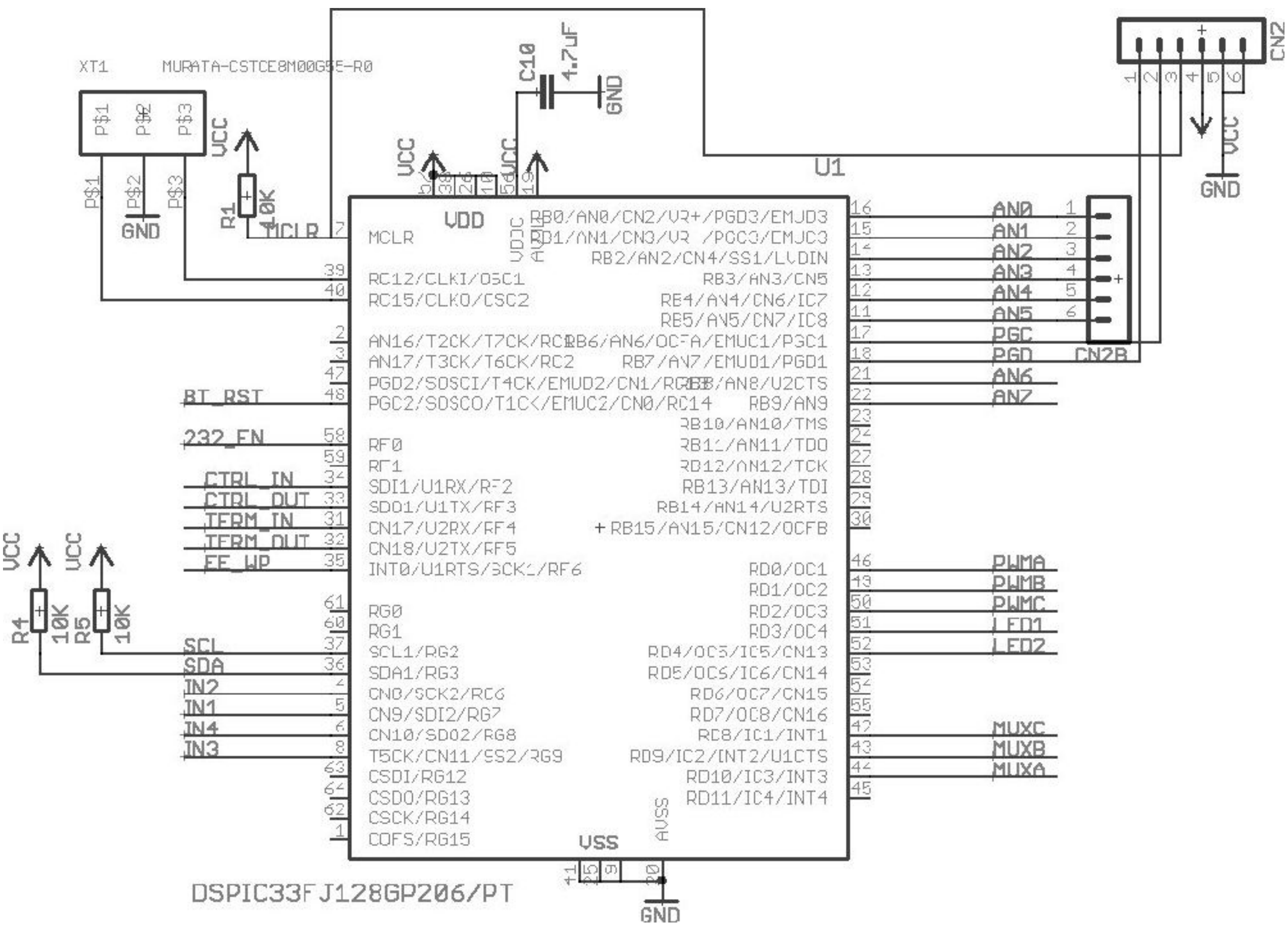
Place the JP1 jumper in the 'B' position, switch on power to the p.Brain-µ24, wait until the unit has booted up, (approx 2 seconds), then press and hold the small tactile switch located under the power terminal CN3 for at least 2 seconds, then release. All going well your ESD200 will be configured in approximately 5 seconds.

The bluetooth pass key or pin code is: 1138 *This is only true for version 1.2 and higher of the HexEngine, for versions below 1.2, please see the relative documentation.*

Once configuration is complete, your ESD200 should now be in discover mode, this means that you should be able to pair with your ESD200 using Windows bluetooth utility or similar. During the pairing procedure you will be asked for the pass key defined above. When pairing is complete, you should be able to connect you your ESD200 with the COM port assigned by your host operating system.

The blue LED next to the ESD200 is the connection LED. This should illuminate when a connection is made between the device and a host system. *Note I have noticed in some cases the LED does not light even when a connection is established, this seems to be determined by the host software initiating the connection!?*

Micro controller Schematic



This schematic displays the micro controller connections to the PWM block, EEPROM, LED's and various ports which are described in more detail in the following pages.

Analogue Capture / Digital IO x 8

There are eight p.Brain pins that can be configured as either analogue capture or digital IO:

PORT PIN NAME	p.Brain NAME	DESCRIPTION
RB0/AN0/CN2	DIG0/AN0	Digital IO 0, Ana Chan 0
RB1/AN1/CN3	DIG1/AN1	Digital IO 1, Ana Chan 1
RB2/AN2/CN4	DIG2/AN2	Digital IO 2, Ana Chan 2
RB3/AN3/CN5	DIG3/AN3	Digital IO 3, Ana Chan 3
RB4/AN4/CN6/IC7	DIG4/AN4	Digital IO 4, Ana Chan 4, Input Capture 0
RB5/AN5/CN7/IC8	DIG5/AN5	Digital IO 5, Ana Chan 5, Input Capture 1
RB6/AN6/PGC1	DIG6/AN6	Digital IO 6, Ana Chan 6
RB7/AN7/PGD1	DIG7/AN7	Digital IO 7, Ana Chan 7

DIG4/AN4 and DIG5/AN5 can be configured as input capture ports if needed. DIG0 thru DIG5 have a programmable weak pull-up and change notification interrupt capabilities. DIG6 and DIG7 have no internal pull up resistors and are also the ICSP pins.

For further information on analogue capture, input capture and change notification pins, please see microchips data sheet at www.microchip.com or the supplied sample code.

UART1 / Serial Port 1

Serial port 1 can only operate at Inverted TTL levels. In order to use the port the following micro controller pins are required:

PORT PIN NAME	p.Brain NAME	DESCRIPTION
SDI1/U1RX/**RF2**	CTRL_IN	TTL Data In
SDO1/U1TX/**RF3**	CTRL_OUT	TTL Data Out

To use serial port 1, RF3 needs to be configured as an output, and RF2 as an input. Also the associated UART registers require configuration.

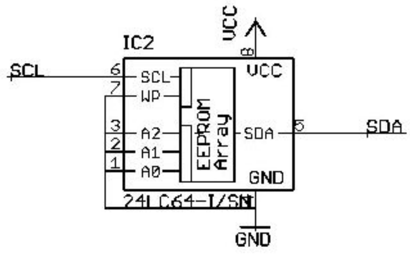
For further information on accessing ports or using the UART please refer to the data sheet at www.microchip.com or see the supplied code samples.

LED Indicators

PORT PIN NAME	SCHEMATIC NAME	DESCRIPTION
RD3	LED1	Red Led
RD4	LED2	Green Led

There are two LED's on the p.Brain module accessible by the user. These LED's have there anode's connected to the port pins defined above. To turn on an LED, enable the relative port bit as an output, and set the port pin high. For further information on accessing ports please see microchips data sheet at www.microchip.com or the supplied code samples.

EEPROM Schematic

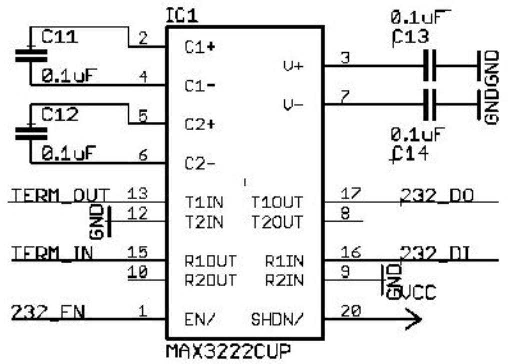


EEPROM Operation

PORT PIN NAME	EEPROM PIN	DESCRIPTION
RG2/SCL1	SCL	Serial Clock
RG3/SDA1	SDA	Serial Data

The on board EEPROM is a microchip 24LC64 device which can accessed using the pins in the above table. The write protect pin and address pins on the EEPROM have been tied to GND, which gives a base address of 0xA0. Communication with the device is achieved using the Synchronous Serial Port 1 in I2C master mode. RG2 & 3 must be configured as inputs, and the SSP port configured as I2C master mode. For further information on accessing ports or the 24LCxx series EEPROM's please see microchips data sheet at www.microchip.com or the supplied *code samples*.

RS232 Schematic



The RS232 transceiver is connected to UART2 on the PIC micro controller. In order for the RS232 receiver to be connected to UART2 RX, JP2 must be installed in the 'R' position. This is due to UART2 being shared between RS232 and/or ESD200 bluetooth module. See below for operation information.

UART2 / Serial Port 2

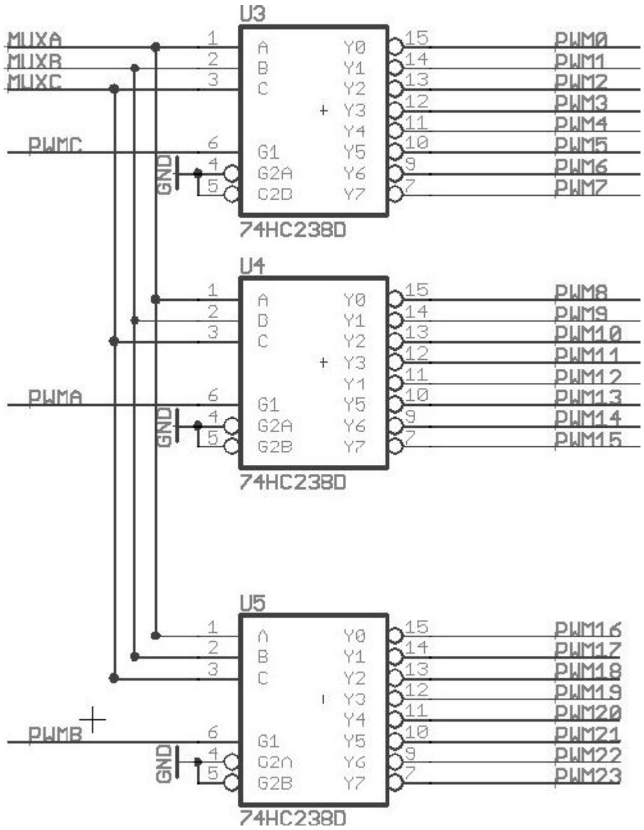
Serial port 2 can be configured as either RS232 or Bluetooth using the optional ESD200 module. UART2 TX pin is sent to both the RS232 transceiver and the ESD200 module socket. UART2 RX pin is switched using JP2 to be driven either from the RS232 transceiver or the ESD200 module. In order to use the port the following micro controller pins are required:

PORT PIN NAME	p.Brain NAME	DESCRIPTION
CN17/U2RX/**RF4**	TERM_IN	TTL Data In
CN18/U2TX/**RF5**	TERM_OUT	TTL Data Out
RF0	232_EN	RS232 Enable

To use serial port 2, RF0,4 need to be configured as outputs, and RF5 as an input. Also the associated UART registers require configuration.

For further information on accessing ports or using the UART please refer to the data sheet at [www.microchip.com]() or see the supplied [code samples]().

PWM Multiplexor Schematic



PWM Multiplexor Operation

In order to get 24 channels of PWM from the dsPIC, three 1 to 8 channel multiplexors are used. Each multiplexor has 3 address lines A, B and C and a gait line which is tied to one of the PWM outputs from the micro controller, PWM1, PWM2 and PWM3. There are two more gaits on the multiplexors which are tied to ground and so are not used. With this arrangement, three PWM channels are driven at a time, one on each multiplexor. Once the PWM cycle is complete for the current three channels, the address select lines are incremented and the next three channels can be driven. This is repeated eight times to give the full 24 channels.

The average R/C PWM signal is 1 to 2ms long, and repeats 50 times per second. With the multiplexor arrangement we need to output 8 PWM signals sequentially, so if we were to say the longest PWM time is 2ms then the maximum time for 8 channels would be 16ms which would give a maximum refresh rate of 62.5Hz. Given that there would be some time required to service interrupts and setup registers, a more realistic refresh rate would be 60Hz.

- FPS = Servo Refresh Rate in Hz
- MAXPWM = Maximum PWM length in mili seconds
- CHANS = Number of sequential channels (remember, each sequential channel drives 3 PWM outputs)

FPS = 1 / (MAXPWM * CHANS) = 1 / (2m * 8) = 62.5Hz

This of course gives the maximum refresh rate for all 24 channels at 2ms PWM pulse width. With less channels, higher refresh rates can be achieved, however, 50 or 60hz is sufficient.

You will notice in the multiplexor schematic that PWM1 does not correspond to the first multiplexor, and the multiplexor outputs are not necessarily in ascending order in reference to servo output connectors. This is due to PCB routing constraints and may seem confusing at first, however, all PWM outputs can be re-mapped in software to the correct servo output pin. (see example code)

PORT PIN NAME	SCHEMATIC NAME	DESCRIPTION
RD0/OC1	PWM1	Output Compare Register PWM output 1
RD1/OC2	PWM2	Output Compare Register PWM output 2
RD2/OC3	PWM3	Output Compare Register PWM output 3
RD10/IC3/INT3	MUXA	Multiplexor Block Address A (LSB)
RD9/IC2/INT2/U1CTS	MUXB	Multiplexor Block Address B
RD8/IC1/INT1	MUXC	Multiplexor Block Address C (MSB)

In order to access the multiplexor block, RD0,1,2,8,9,10 must be configured as outputs in the port configuration registers. Also output compare registers 1 thru 3 must be configured for PWM output. For further information on accessing ports, and output compare registers please see microchips data sheet at www.microchip.com or the supplied code samples.

Code Examples

All code examples are written for the Hitec dsPICC compiler. All code examples are subject to the following license agreement.

Software License Agreement

Processor Initialisation

In the following code example you will see how to initialize the processor oscillator and configuration bits. The processor is configured to run at 32Mhz using the 8Mhz external resonator. The main programme then uses functions for port initialisation, serial communications, time delays and EEPROM access. These functions are defined in the code examples sections below.

```
//      **********************************************************************
//      DEFINITIONS
//      **********************************************************************

#define         PIC_CLK         32000000                //      CLOCK SPEED
#define         FCY             (PIC_CLK/2)             //      INSTRUCTION SPEED

                //      SOME USEFULL TYPE DEFINITIONS
typedef         BYTE    unsigned char
typedef         WORD    unsigned int


//      **********************************************************************
//      HEADER FILES
//      **********************************************************************

#include        <htc.h>
#include        <stdio.h>
#include        <stdlib.h>

//      **********************************************************************
//      CONST STRING DATA
//      **********************************************************************

const char build_title[] = {"Example"};
const char build_author[] = {"micromagic systems ltd"};
const char build_date[] = {__DATE__};
const char build_time[] = {__TIME__};
const char build_vers[] = {"1.0"};
```

```
//      ********************************************************************
//      CONFIGURATION BITS
//      ********************************************************************

__CONFIG( FOSCSEL, OSCPLL & IESODIS & TEMPEN );      // PLL ON
__CONFIG( FOSC, POSCXT & FCKSMDIS );                 // EXTERNAL XT OSCILATOR
__CONFIG( FWDT, WDTDIS );                    //      WATCHDOG TIMER DISABLED
__CONFIG( FPOR, PWRT128 );                   //      POWERUP TIMER = 128mS
__CONFIG( FGS, GCPU );                       //      NO CODE PROTECT


//      ********************************************************************
//      PSECT. NOT NECESSARY BUT CAN BE USEFULL FOR DEBUGGING
//      ********************************************************************

#pragma psect text=main

//      ********************************************************************
//      PORT DEFINITIONS
//      ********************************************************************

#define PORTB_SETUP     0b0000001111111111
#define PORTC_SETUP     0b0000000000000000
#define PORTD_SETUP     0b0000000000000000
#define PORTF_SETUP     0b0000000000010100
#define PORTG_SETUP     0b0000001111001100

#define PORT_CNPU1      0b0000111111111100
#define PORT_CNPU2      0b0000000000000000

#define RED_LED         RB14    // LED1
#define GRN_LED         RB15    // LED2

#define PWM1            RD0     // USED BY PWM BLOCK
#define PWM2            RD1
#define PWM3            RD2
#define PWM4            RD3

#define MUX_A0          RD10    // PWM BLOCK MULTIPLEXOR PINS
#define MUX_A1          RD9
#define MUX_A2          RD8

#define EE_WP           RD7     // EEPROM WRITE PROTECT
#define SCL             RG2     // I2C SERIAL CLOCK
#define SDA             RG3     // I2C SERIAL DATA

#define RS232_EN        RF0     // RS232 ENABLE

#define JP0             !RG8
#define JP1             !RG9
#define JP2             !RG6
#define JP3             !RG7


//      ********************************************************************
//      SERIAL PORT DEFINITIONS AND GLOBAL VARIABLES
//      ********************************************************************

//      THE FOLLOWING BAUD RATE VALUES ARE FOR CLOCK SPEEDS OF 32 MHZ
//      PLEASE CONSULT THE dsPIC33F DATA SHEET FOR BAUD RATE CALCULATIONS
//      OR USE THE Pic Baud Calcualator TOOL FROM MICROMAGIC SYSTEMS.
//      NOTE: DO NOT USE BRGH=1 SETTING ON THIS PROCESSOR, IT DOES NOT WORK
//      ON EARLY SILICON REVISIONS

#define         B9600           103     //      0.16% ERROR
#define         B19200          51      //      0.16% ERROR
#define         B38400          25      //      0.16% ERROR
#define         B57600          16      //      2.12% ERROR
```

```
#define        B115200        8       //      -3.55% ERROR

//      MOST PC BASED UARTS WILL ACCEPT BAUD RATE ERRORS OF +/- 3.55%, HOWEVER
//      IF YOUR HOST SYSTEM ALSO HAS A BAUD RATE ERROR, YOU MAY EXPERIENCE  PROBLEMS.

#define        TX2_BUFFER_SIZE       32      // MUST BE 2^n EG. 2,4,8,16,32,64...
#define        RX2_BUFFER_SIZE       32      // MUST BE 2^n EG. 2,4,8,16,32,64...

volatile       BYTE    gTx2Buffer[ TX2_BUFFER_SIZE ];
volatile       WORD    gTx2Cntr, gTx2Ptr;
volatile       bit     gTx2BufferFull;

volatile       BYTE    gRx2Buffer2[ RX2_BUFFER_SIZE ];
volatile       WORD    gRx2Cntr2, gRx2Ptr;
volatile       bit     gRx2BufferFull;

//      *********************************************************************
//      PWM DEFINITIONS & GLOBAL VARIABLES
//      *********************************************************************

#define OC_TIMER_PRE_SCALER    8       // OUTPUT COMPARE TIMER PRE SCALER, USED BY TIMER 3.
#define PWM_MUL         (double)(FCY/OC_TIMER_PRE_SCALER/1000000)    // CLOCK CYCLES PER uSECOND
#define PWM_RANGE       (WORD)(PWM_MUL * 1000)                       // PWM RANGE IN uS
#define PWM_MID         (WORD)(PWM_MUL * 1500)                       // PWM CNETRE IN uS
#define PWM_MIN         (WORD)(PWM_MID - (PWM_RANGE/2) )             // PWM MIN
#define PWM_MAX         (WORD)(PWM_MID + (PWM_RANGE/2) )             // PWM MAX

#define SERVO_MAX       (int)(PWM_RANGE)                             // USEFULL DEFINITIONS FOR
#define SERVO_MID       (int)(PWM_RANGE/2)                           // SERVO TRAVEL RANGE
#define SERVO_MIN       0                                            //

BYTE           gMuxAddress;          // PWM MULTIPLEXOR ADDRESS, 0 to 5
BYTE           gServoEnable[24];     // SERVO ENABLE ARRAY, NON ZERO + ENABLE PWM OUTPUT
WORD           gServoPos[24];        // SERVO POSITION ARRAY.
                                     // CONSTANT SERVO REMAP ARAY
                                     // THIS ARRAY IS THE CONFIGURATION FOR THE p.Brain-SMB
                                     // MOTHERBOARD.
const BYTE     cServoRemap[24] = {0,1,2,3,8,9,10,11,16,17,18,19,23,20,21,22,15,12,13,14,7,4,5,6};


//      *********************************************************************
//      USEFULL DELAY MACROS
//      *********************************************************************

#define        dly1u          NOP();NOP();NOP();NOP();NOP();NOP();NOP();NOP()
#define        dly500n        NOP();NOP();NOP();NOP()


//      *********************************************************************
//      FUNCTION PROTOTYPES
//      *********************************************************************

void    Initialise( void );

void    putch( char pC );             // SERIAL ROUTINES
char    getch( void );
char    getche( void );
int     kbhit( void );
void    interrupt      tx2isr( void );
void    interrupt      rx2isr( void );

void    i2cWaitForIdle( void )        // EEPROM ROUTINES
void    i2cStart( void )
void    i2cRepStart( void )
void    i2cStop( void )
BYTE    i2cRead( BYTE pAck )
WORD    i2cWrite( BYTE pData )
void    my_eeprom_write_byte( WORD pAddr, BYTE pData )
BYTE    my_eeprom_read_byte( WORD pAddr )
```

```
void    my_eeprom_write_buffer( WORD  pAddr, BYTE *pData )

void    DelayUs( WORD pDelay )        // DELAY ROUTINES
void    DelayMs( WORD pDelay )




//      *********************************************************************
//      MAIN CODE
//      *********************************************************************

void    main()
{
BYTE    lT;
int     lI;

//      SETUP THE CLOCK PLL (PHASE LOCKED LOOP).
//      THRE ARE CERTAIN RESTRICTIONS TO THE CLOCK SPEED AT EACH STAGE
//      OF THE PLL:
//      PLLPRE : CLK MUST BE BETWEEN 0.8 > 8.0Mhz HERE
//      PLLFBD : ( MUL FACTOR ) CLK MUST BE 100 > 200 Mhz Here
//      PLLPOST : PLL OUTPUT CLK MUST BE 12.5 > 80 Mhz.

                        //      WITH OUR INPUT CLOCK OF 8Mhz
CLKDIV = 0;             //      PLLPRE = /2 = 4Mhz
PLLFBD = 0x001e;        //      PLL x 32 = 128Mhz
PLLPOST1 = 1;           //      PLLPOST = /4 = 32Mhz

While( !LOCK )          //      NOW WE SHOULD WAIT FOR THE PLL TO LOCK.
        continue;

Initialise();                   //      CONFIGURE OUR PORTS & HARDWARE

GRN_LED = 1;                    //      MAIN PROGRAMME BEGINS HERE

U2RXIE = 1;                     //      ENABLE UART2 RECEIVER INTERRUPTS
RS232_EN = 0;                   //      ENABLE RS232 TRANSCEIVER

                                //      DISPLAY MESSAGE
printf("Hello World!\r\n\r\nPress SPACE bar to continue.");

while( 1 )                      //      WAIT FOR SPACE BAR
        {
        if( kbhit() )           //      WAS KEYBOARD HIT?
                {
                lC = getche(); //      GET & CHECK KEY
                if( lC == ' ' )
                        break;
                }
        }



lI = 0;
while(1)                        //      LOOP
        {
        if( T1IF )
                {
                T1IF = 0;                       //      RESET T1 FLAG
                RED_LED ^= 1;                   //      FLASH LED

                //      FILL THE SERVO BUFFER WITH SOME SERVO DATA
                //      THIS ROUTINE JUST MOVES THE SERVOS BACK AND FORWARD
                //      THROUGH THEIR FULL RANGE.

                if( gServoBuffer[0] == SERVO_MAX )
                        lI = -1;
                else if( gServoBuffer[0] == SERVO_MIN )
                        lI = 1;
```

```
        for( lT = 0; lT < 24; lT++ )
                gServoBuffer[ lT ] += lI;

                                        //      COPY & RE-MAP SERVO DATA TO gServoPos BUFFER
        DistributeServoOutputs( &gServoBuffer );

        StartPWMFrame();                //      START PWM FRAME
        }
}
```

Port & Hardware Initialisation

The following example demonstrates the initialization of the processor ports and hardware configuration registers. This function is called “Initialise()”

```
void    Initialise( void )
{
//      ****************************************************************
//      PORT SETUP
//      ****************************************************************

PORTB = 0;
PORTC = 0;
PORTD = 0;
PORTF = 0;
PORTG = 0;

TRISB = PORTB_SETUP;   //      CONFIGURE PORT FOR I/O
TRISC = PORTC_SETUP;
TRISD = PORTD_SETUP;
TRISF = PORTF_SETUP;
TRISG = PORTG_SETUP;

CNEN1 = 0;                    //      CHANGE NOTIFICATION ENABLE REGISTER
CNEN2 = 0;
CNPU1 = PORT_CNPU1;    //      CHANGE NOTIFICATION PULL-UPS ENABLE REGISTER
CNPU2 = PORT_CNPU2;

ODCD = 0;                     //      ALL PORTS ARE NOT OPEN DRAIN
ODCF = 0;
ODCG = 0;


//      ****************************************************************
//      INTERRUPT SETUP
//      ****************************************************************

INTCON1 = 0;    // CLEAR  INTERRUPT CONTROL REGISTERS
INTCON2 = 0;

IFS0 = 0;              // CLEAR ALL INTERRUPT REQUEST FLAG REGISTERS
IFS1 = 0;
IFS2 = 0;
IFS3 = 0;
IFS4 = 0;

IEC0 = 0;              //      CLEAR ALL INTERRUPT ENABLE CONTROL REGISTERS
IEC1 = 0;
IEC2 = 0;
IEC3 = 0;
IEC4 = 0;


IPC0 = 0;              //      CLEAR ALL INTERRUPT PRIORITY REGISTERS
IPC1 = 0;
IPC2 = 0;
IPC3 = 0;
IPC4 = 0;
IPC5 = 0;
IPC6 = 0;
IPC7 = 0;
IPC8 = 0;
IPC9 = 0;
IPC10 = 0;
IPC11 = 0;
IPC12 = 0;
IPC13 = 0;
IPC14 = 0;
```

```
IPC15 = 0;
IPC16 = 0;
IPC17 = 0;


//      ***************************************************************
//      TIMER MODULES SETUP
//      ***************************************************************


//      TIMER 1 CONFIGURED AS MAIN LOOP TICK TIMER

T1CON = 0;              //
T1CKPS0 = 1;            //      PRESCALER 1:8
PR1 = TIMERVAL;
T1ON = 1;               //      T1 ON


//      T3 USED BY OC/PWM GENERATION.
//      USED AS INTERRUPT WHEN ALL FOUR OC/PWM OUTPTUS ARE DISABLED
//      SO AS TO CREATE NEXT PWM FRAME.

T3CON = 0;
T3CKPS0 = 1;            //      PRESCALER 1:8
PR3 = 0xffff;
T3IP0 = 1;              //      T3 INTERRUPT PRIORITY 3
T3IP1 = 1;
T3IP2 = 0;
T3ON = 1;               //      T3 ON

T2CON = 0;              //      UNUSED TIMERS
T4CON = 0;
T5CON = 0;
T6CON = 0;
T7CON = 0;
T8CON = 0;
T9CON = 0;

//      ***************************************************************
//      INPUT CAPTURE SETUP
//      ***************************************************************

IC1CON = 0;             //      ALL INPUT CAPTURES OFF
IC2CON = 0;
IC3CON = 0;
IC4CON = 0;
IC5CON = 0;
IC6CON = 0;
IC7CON = 0;
IC8CON = 0;

//      ***************************************************************
//      OUTPUT COMPARE SETUP
//      ***************************************************************

OC1CON = 0;             //      OUTPUT COMPARE 1
OC1_TSEL = 1;           //      USE TIMER3
OC1IP0 = 0;             //      INTERRUPT PRIORITY 2
OC1IP1 = 1;
OC1IP2 = 0;

OC2CON = 0;             //      OUTPUT COMPARE 2
OC2_TSEL = 1;           //      USE TIMER3
OC2IP0 = 0;             //      INTERRUPT PRIORITY 2
OC2IP1 = 1;
OC2IP2 = 0;

OC3CON = 0;             //      OUTPUT COMPARE 3
OC3_TSEL = 1;           //      USE TIMER3
OC3IP0 = 0;             //      INTERRUPT PRIORITY 2
```

```
OC3IP1 = 1;
OC3IP2 = 0;

OC4CON = 0;           //      OUTPUT COMPARE 4
OC5CON = 0;           //      UNUSED OUTPUT COMARE
OC6CON = 0;
OC7CON = 0;
OC8CON = 0;

//      ****************************************************************
//      SPI MODULE SETUP
//      ****************************************************************

SPI1STAT = 0;  //      UNUSED
SPI2STAT = 0;

//      ****************************************************************
//      I2C MODULE SETUP
//      ****************************************************************

I2C1CON = 0;          //      I2C1 CONFIGURATION
I2C1STAT = 0;
I2C1_EN = 1;          //      ENABLE
                      //      CALCULATE BIT RATE
I2C1BRG = ((FCY/400000) - (FCY/1111111)) - 1;
I2C1_BCL = 0;         //      CLEAR BUS COLLISION FLAG
I2C1MSK = 0;          //      CLEAR MASK
I2C1ADD = 0;          //      CLEAR ADDRESS

//      ****************************************************************
//      DATA CONVERTER INTERFACE MODULE SETUP
//      ****************************************************************

DCICON1 = 0;   //      UNUSED



//      ****************************************************************
//      ADC
//      ****************************************************************

AD1CON1 = 0;   //      UNUSED
AD1CON2 = 0;
AD1CON3 = 0;
AD1CON4 = 0;
AD1CHS123 = 0;
AD1CHS0 = 0;

AD1PCFGL = 0xffff;    //      ADC PORT PIN CONFIG: 0 = ADC, 1 = DIG.
AD1PCFGH = 0xffff;    //      ADC PORT PIN CONFIG: 0 = ADC, 1 = DIG.



//      ****************************************************************
//      UART1 SETUP
//      ****************************************************************


U1MODE = 0;           //      UART1 CONFIGURATION
U1_SPEN = 1;          //      ENABLE
U1STA = 0;            //      CLEAR STATUS REGISTER
U1_TXEN = 1;          //      ENABLE TX
U1_WAKE = 1;          //      UART WILL WAKE PROCESSOR FROM IDLE MODE
U1BRG = B38400;       //      SET BAUD RATE

U1TXIP0 = 1;          //      TX INTERRUPT PRIORITY 1
U1TXIP1 = 0;
U1TXIP2 = 0;
U1_TXISEL0 = 1;       //      0 = INT UPON TRANSFER TO SHIFT REG.
```

```
                        //      1 = INT UPON FIFO EMPTIED.
U1RXIP0 = 0;            //      RX PRIORITY 6
U1RXIP1 = 1;
U1RXIP2 = 1;

U1_RCISEL0 = 0;         //      INT UPON BYTE RECEIVED
U1_RCISEL1 = 0;         //




//      ****************************************************************
//      UART 2 SETUP
//      ****************************************************************

U2MODE = 0;             //      UART2 CONFIGURATION
U2_SPEN = 1;            //      ENABLE
U2STA = 0;              //      CLEAR STATUS REGISTER
U2_TXEN = 1;            //      TX ENABLE
U2_WAKE = 1;            //      UART WILL WAKE PROCESSOR FROM IDLE MODE
U2BRG = B38400;         //      SET BAUD RATE

U2TXIP0 = 1;            //      TX INTERRUPT PRIORITY 1
U2TXIP1 = 0;
U2TXIP2 = 0;
U2_TXISEL0 = 1;         // 0 = INT UPON TRANSFER TO SHIFT REG.
                        // 1 = INT UPON FIFO EMPTIED.

U2RXIP0 = 0;            //      RX INTERRUPT PRIORITY 6
U2RXIP1 = 1;
U2RXIP2 = 1;

U2_RCISEL0 = 0;         //      INT UPON BYTE RECEIVED
U2_RCISEL1 = 0;

//      SWITCH OFF ALL UNUSED PERIPHERALS. 0 = ON 1 = OFF
//      T5MD T4MD T3MD T2MD T1MD QEIMD PWMMD DCIMD
//      I2C1MD U2MD U1MD SPI2MD SPI1MD C2MD C1MD AD1MD

PMD1 = 0b1000011100011110;

//      IC8MD IC7MD IC6MD IC5MD IC4MD IC3MD IC2MD IC1MD
//      OC8MD OC7MD OC6MD OC5MD OC4MD OC3MD OC2MD OC1MD

PMD2 = 0b1111111111110000;

//      T9MD T8MD T7MD T6MD — — — — — — — — — — I2C2MD AD2MD

PMD3 = 0b1111111111111111;

}
```

RS232 Communications

In this example interrupt driven communications will be configured for the RS232 serial port on UART2. The following are high level serial port functions:

```
//*************************************************************************
// PUTCH USED BY PRINTF COMMANDS TO TERMINAL PORT
//*************************************************************************

void    putch( char pC )
{
while( gTx2Cntr == TX2_BUFFER_SIZE )  //      WAIT FOR SPACE IN BUFFER
        {
        gTX2BufferFull = 1;
        U2TXIE = 1;
        }

U2TXIE = 0;                           //      DISABLE TX INTERRUPTS
NOP();
gTx2Buffer[ gTx2Ptr ] = pC;           //      STORE CHAR IN FIFO BUFFER
gTx2Ptr++;                            //      INC BUFFER POINTER
gTx2Ptr &= TX2_BUFFER_SIZE-1;         //      AND MASK.
gTx2Cntr++;                           //      INC BUFFER COUNT
U2TXIE = 1;                           //      ENABLE TX INTERRUPTS
}

//*************************************************************************
// GETCHE GET BYTE FROM TERMINAL SERIAL PORT
//*************************************************************************

char    getch( void )
{
char    lC;
                                      //      WAIT FOR A BYTE IN THE BUFFER
while( !gRx2Cntr )
        continue;

U2RXIE = 0;                           //      DISABLE RX INTERRUPT
NOP();
                                      //      GET BYTE FROM FIFO BUFFER
lC = gRx2Buffer[ ( gRx2Ptr - gRx2Cntr ) & RX2_BUFFER_SIZE-1 ];
gRx2BufferFull = 0;                   //      CLEAR BUFFER FULL FLAG
gRx2Cntr--;                           //      DEC BUFFER COUNTER
U2RXIE = 1;                           //      ENABLE RX INTERRUPTS

return lC;                            //      RETURN CHAR
}

//*************************************************************************
// GETCHE GET BYTE FROM TERMINAL SERIAL PORT AND ECHO BACK
//*************************************************************************

char    getche( void )
{
char    lC;

lC = getch();
putch( lC );

return lC;
}

//*************************************************************************
// KBHIT FUNCTION, EG. ARE THERE ANY BYTES IN RX FIFO
//*************************************************************************

int     kbhit( void )
{
return  gRx2Cntr;       //      RETURN THE FIFO BUFFER COUNTER
```

```
}
```

Now for the interrupt service routines:

```
//*************************************************************************
// UART2 TX ISR
//*************************************************************************

void	interrupt	tx2isr( void ) @ U2TX_VCTR
{
if( U2TXIF )				//	IS THE TXIF FLAG SET? IT SHOULD BE!
	{
	while( !U2_TXBF && tx_cntr2 )	//	SPACE IN THE UART FIFO? & DATA TO TRANSMIT?
		{
					//	LOAD THE UART BUFFER WITH NEW DATA FROM FIFO BUFFER
		U2TXREG = gTx2Buffer[ ( gTx2Ptr - gTx2Cntr ) & TX2_BUFFER_SIZE-1 ];
		gTx2BufferFull = 0;	//	CLEAR BUFFER FULL FLAG
		gTx2Cntr--;		//	DEC BUFFER COUNTER

		if( !gTx2Cntr)		//	IF THERE ARE NO MORE BYTES IN THE FIFO BUFFER
			{
			U2TXIE = 0;	//	DISABLE TX INTERRUPTS
			}
		}

	U2TXIF = 0;			//	CLEAR THE TX INTERRUPT FLAG
	}

}

//*************************************************************************
// UART2 RX ISR
//*************************************************************************

void	interrupt	rx2isr( void ) @ U2RX_VCTR
{
BYTE	lC;

if( U2RXIF )				//	IS THE RX INTERUPT FLAG STE? IT SHOULD BE!
	{
	while( U2_URXDA )		//	WHILE THRE IS DATA IN THE UART FIFO
		{
		if(U2_FERR)		//	WAS THE DATA CORRUPT?
			{
			lC = U2RXREG;	//	YES, SO ABSORB DATA
			}
		else
			{
			lC = U2RXREG;	//	NO, SO STORE DATA IN FIFO BUFFER
				if( gRx2Cntr == RX2_BUFFER_SIZE )
				{
				gRx2BufferFull = 1;
				}
			else
				{
				gRx2Buffer[ gRx2Ptr ] = lC;
				gRx2Ptr++;
				gRx2Ptr &= RX2_BUFFER_SIZE-1;
				gRx2Cntr++;
				}
			}
		}
	U2RXIF = 0;			//	CLEAR RX INTERRUPT FLAG
	}
}
```

EEPROM Read & Write

These examples demonstrate how to read and write to the on-board EEPROM. First we define the low level I2C access functions:

```
//*************************************************************************************
// WAIT FOR I2C IDLE STATE
//*************************************************************************************

void    i2cWaitForIdle( void )
{

while( (I2C1CON & 0x001f) | I2C1_TRSTAT )
        continue;
}

//*************************************************************************************
// START I2C
//*************************************************************************************

void i2cStart()
{

i2cWaitForIdle();

I2C1_SEN = 1;
}

//*************************************************************************************
// I2C REPEAT START
//*************************************************************************************

void i2cRepStart()
{

i2cWaitForIdle();

I2C1_RSEN = 1;
}

//*************************************************************************************
// I2C STOP
//*************************************************************************************

void i2cStop()
{

i2cWaitForIdle();

I2C1_PEN = 1;
}

//*************************************************************************************
// I2C READ BYTE
//*************************************************************************************

BYTE i2cRead( BYTE pAck )
{
BYTE lData;

i2cWaitForIdle();

I2C1_RCEN=1;

i2cWaitForIdle();
```

```
lData = I2C1RCV;

i2cWaitForIdle();

if ( pAck )
       I2C1_ACKDT = 0;
else
       I2C1_ACKDT = 1;

I2C1_ACKEN = 1;              // SEND ACKNOWLEDGE SEQUENCE

return( lData );
}

//*****************************************************************************************
// I2C WRITE DATA
//*****************************************************************************************

WORD i2cWrite( BYTE pData )
{
i2cWaitForIdle();

I2C1TRN = pData;

return ( !I2C1_ACKSTAT  ); // RETURNS 1 IF TX IS ACKNOWLEDGED
}
```

High level EEPROM functions used for storing BYTE data etc:

```
// ************************************************************************************
// WRITE EEPROM BYTE
// ************************************************************************************

void   my_eeprom_write_byte( WORD   pAddr, BYTE pData )
{
EE_WP = 0;                   //     DISABLE EEPROM WRITE PROTECT

i2cStart();                  //     START I2C
i2cWrite( 0xa0 );            //     WRITE EEPROM ADDRESS
i2cWrite( pAddr >> 8 );      //     WRITE EEPROM DATA ADDRESS
i2cWrite( pAddr & 0xff );
i2cWrite( pData);            //     WRITE EEPROM DATA
i2cStop();                   //     START THE DATA ERASE WRITE CYCLE.

DelayUs(6000);               //     WAIT FOR WRITE TO END. 6000uS

EE_WP = 1;                   //     ENABLE EEPROM WRITE PROTECT
}

// ************************************************************************************
// READ EEPROM DATA
// ************************************************************************************

BYTE   my_eeprom_read_byte( WORD pAddr )
{
BYTE pData;

i2cStart();                  //     START I2C
i2cWrite( 0xa0 );            //     WRITE EEPROM ADDRESS
i2cWrite( (addr >> 8) );     //     WRITE DATA ADDRESS
i2cWrite(addr & 0xff);
i2cRepStart();               //     REPEAT START CONDITION
i2cWrite(0xa1);              //     WRITE READ COMMAND
pData = i2cRead(0);          //     READ DATA
i2cStop();                   //     STOP I2C

return( pData );
```

```
}

// ****************************************************************************************
// WRITE EEPROM 32 BYTE BUFFER
// ****************************************************************************************
// MUST NOT CROSS PAGE BOUNDARY OF 32, EG ONLY 5 LOWER ADDRESS BITS ARE INTERNALY
// INCREMENTED.

void    my_eeprom_write_buffer( WORD  pAddr, BYTE *pData )
{
BYTE    pT;

EE_WP = 0;                      //      DISABLE EEPROM WRITE PROTECT

i2cStart();                     //      START I2C
i2cWrite( 0xa0 );               //      WRITE EEPROM COMMAND
i2cWrite( pAddr >> 8 );         //      WRITE DATA ADDRESS START
i2cWrite( pAddr & 0xff );
for( pT = 0; pT < 32; pT++ )    //      WRITE 32 BYTES OF DATA
        i2cWrite(*pData++);
i2cStop();                      //      START THE DATA ERASE WRITE CYCLE.

DelayUs(6000);                  //      WAIT FOR WRITE TO END. 6000uS

EE_WP = 1;                      //      ENABLE EEPROM WRITE PROTECT
}
```

Delay Routines

These functions are used for inserting milisecond or microsecond delays.

```
// ***********************************************************************************
// DELAY MICRO SECONDS. NEEDS ADJUSTING FOR DIFFERENT PROCESSOR SPEEDS
// ***********************************************************************************

void    DelayUs( WORD pDelay )
{
while(--pDelay > 0)
        {
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        }
}

// ***********************************************************************************
// DELAY MILLI SECONDS.
// ***********************************************************************************

void    DelayMs( WORD pDelay )
{
WORD lI;

while (pDelay--)
        {
        CLRWDT();
        lI = 4;
        while(lI--)
                {
                DelayUs( 250 );        // ADJUST FOR ERROR
                }
        }
}
```

PWM Multiplexor

This example demonstrates how to configure the interrupt service routines for the PWM multiplexor block. For the purpose of this example, each PWM frame will be started in the main programme loop, once the frame is started, the output compare ISR's will continiue until all 24 PWM channels are complete. There are four multiplexors, each with six outputs giving 24 channels of PWM, using multiplexing techniques, PWM signals are created in banks of four, one ofr each of the four Output Compare registers. See the schematic of the PWM block for further details.

Variables used by the PWM multiplexor:

PWM Definitions

Output compare interrupt service routines:

```
// *******************************************************
// OC1 ISR
// *******************************************************

void    interrupt      oc1_isr( void ) @ OC1_VCTR
{
OC1IE = 0;             // DISABLE OCx INTERRUPTS
OC1IF = 0;             // CLEAR INTERRUPT FLAG
NextPWMBank(); // CHECK NEXT PWM BANK
}


// *******************************************************
// OC2 ISR
// *******************************************************

void     interrupt     oc2_isr( void ) @ OC2_VCTR
{
OC2IE = 0;
OC2IF = 0;
NextPWMBank();
}


// *******************************************************
// OC3 ISR
// *******************************************************

void     interrupt     oc3_isr( void ) @ OC3_VCTR
{
OC3IE = 0;
OC3IF = 0;
NextPWMBank();
}

// *******************************************************
// IF ALL 4 PWM OUTPUTS IN THE CURRENT BANK ARE DISABLED
// THIS TIMER IS USED TO FORCE AN INTERRUPT TO START
// THE NEXT PWM BANK.
// *******************************************************

Void    interrupt      tmr3_isr( void ) @ T3_VCTR
{
T3IF = 0;              // CLEAR INTERRUPT FLAG
T3IE = 0;              // DISABLE INTERRUPT
NextPWMBank(); // CHECK NEXT PWM BANK
}
```

```
// ******************************************************
// USED TO START THE PWM FRAME, SETS MUX ADDRESS TO 0
// ******************************************************

void    StartPWMFrame( void )
{

T3ON = 0;                                   //      SWITCH TIMER OFF
TMR3 = 0;

if( OC1IE || OC2IE || OC3IE ) //      CHECK FOR FRAME RATE ERROR
        {
        // IF ANY OF THE OCIE BITS IS SET WHEN THIS FUNCTION
        // IS CALLED, SOMETHING HAS EITHER GONE WRONG, OR THE
        // PWM REFRESH RATE IS TOO HIGH. STARTING A NEW FRAME
        // BEFORE THE OLD FRAME HAS FINISHED WILL CAUSE SERVO
        // GLITCHES AND OTHER ISSUES!

        // AT THIS POINT IT IS DOWN TO THE USER TO DECIDE WHAT TO DO!
        // SET ERROR FLAG, OR RETURN ETC..
        // PWM_error_flag = 1;
        // return;
        }

OC1IE = 0;              // DISABLE ALL OC INTERRPUTS, INCASE WE IGNORE THE ABOVE ERROR CONDITION
OC2IE = 0;
OC3IE = 0;

gMmuxAddress = 0;       // SET MUX ADDRESS TO 0

MUX_A2 = 0;NOP();       // IF A2 AND A1 ARE CHANGED WITHOUT AN OP BETWWEN THEM,
MUX_A0 = 0;NOP();       // THE SECOND WILL NOT TAKE EFFECT. OLD PIC BUG ABOUT
MUX_A1 = 0;NOP();       // NOT CHANGING PORT PINS WITHIN CONSECUTIVE INSTRUCTIONS!

SetipOCRegisters();     // CONFIGURE OC REGISTERS FOR NEXT PWM BANK

T3ON = 1;               // START TIMER 3

}



// ***********************************************************************************
// THIS FUNCTION CONFIGURESS THE OC REGISTERS WITH THE NEXT BANK OF PWM VALUES
// EACH PWM START TIME IS OFFSET BY A SMALL AMOUNT DEFINED BELOW. THIS IS TO REDUCE
// SIGNAL NOISE ON THE P.BRAIN, BUT IS NOT ENTIRELY NECESSARY.
// ***********************************************************************************

#define PWM1_START_COUNT       2
#define PWM2_START_COUNT       4
#define PWM3_START_COUNT       6

void    SetipOCRegisters( void )
{

//      SETUP T3 TO INTERRUPT IN 50uS IN THE CASE THAT NONE OF THE CURRENT PWM BANK SERVOS
//      ARE ENABLED. IF ANY OF THE FOLLOWING 4 ARE ENABLED, THE T3 INTS ARE DISABLED
//      AND PR3 REG SET TO MAX.

PR3 = (WORD)((double)((FCY/OC_TIMER_PRE_SCALER)/1000000.0) * (50.0));      // IN uSeconds

T3IF = 0;       // CLEAR T3 INTERRUPT FLAG
T3IE = 1;       // T3 INTERRUPTS ENABLED, BUT T3 IS NOT YET RUNNING.
```

```
//      IN ORDER TO UNDERSTAND WHY OC1 DOES SERVO CHANNELS 6 TO 11, PLEASE LOOK AT THE PWM
//      MULTIPLEXOR BLOCK IN THE SCHEMATICS SECTION OF THE p.Brain-ds24 USER GUIDE.

//      SETUP COMPARE REGISTERS START AND STOP TIMES.
//      OCR1 = CHANELS 8 -> 15

OC1R = PWM1_START_COUNT;                     // CONFIGURE START TIME, EG RISING EDGE
                                             // CONFIGURE STOP TIME, EG FALLING EDGE
OC1RS = PWM1_START_COUNT + PWM_MIN + gServoPos[ 8 + gMuxAddress ];

if( gServoEnable [ 8 + gMuxAddress ] )       // IF THE SERVO IS ENABLED
        {
        OC1_M2 = 1;                          // CONFIGURE THE OC MODE
        OC1IE = 1;                           // ENABLE OC INTERRUPTS
        T3IE = 0;                            // DISABLE T3 INTERRUPTS
        PR3 = 0xffff;                        // RESET PR3 TO MAXIMUM
        }

//      OCR2 = CHANELS 16 -> 23
OC2R = PWM2_START_COUNT;
OC2RS = PWM2_START_COUNT + PWM_MIN + gServoPos[ 16- gMuxAddress ];
if( gServoEnable [ 16 - gMuxAddress ] )
        {
        OC2_M2 = 1;
        OC2IE = 1;
        T3IE = 0;
        PR3 = 0xffff;
        }



//      OCR3 = CHANELS 0 -> 7
OC3R = PWM3_START_COUNT;
OC3RS = PWM3_START_COUNT + PWM_MIN + gServoPos[ gMuxAddress ];
if( gServoEnable [ gMuxAddress ] )
        {
        OC3_M2 = 1;
        OC3IE = 1;
        T3IE = 0;
        PR3 = 0xffff;
        }



gMuxAddress++;                               // INC THE MUC ADDRESS.

}

// ***********************************************************************************
// ONCE THE PWM FRAME HAS BEEN STARTED, THIS FUNCTION IS CALLED WITHIN EACH OC ISR
// AND WITHIN THE T3 ISR (IF ENABLED) TO DETERMINE IF THE CURRENT PWM BANK IS COMPLET
// EG, ALL FOUR PWM OUTPUTS OF THE BANK ARE FINISHED. IF ALL FOUR CHANNELS OF THE BANK
// ARE COMPLETE, IT THEN CHECKS IF THE FRAME IS COMPLETE.
// ***********************************************************************************

void    NextPWMBank( void )
{

if( OC1IE || OC2IE || OC3IE ) // IF WE HAVE NOT DONE ALL 4 OUTPUTS, RETURN.
        return;

T3ON = 0;                                    // SWITCH TIMER OFF
TMR3 = 0;                                    // CLEAR TIMER 3

//      HAVE WE COMPLETED ALL CHANNELS? IF SO, THE FRAME IS COMPLETE
//      SO DO NOT RESET OUTPUT COMPARE MOCULES.

if( gMuxAddress > 7 )
        return;

if( gMuxAddress & 0x0001 )    //      SETUP MUX ADDRESS BITS
```

```
       MUX_A0 = 1;
else
       MUX_A0 = 0;

if( gMuxAddress & 0x0002 )    //     SETUP MUX ADDRESS BITS
       MUX_A1 = 1;
else
       MUX_A1 = 0;

if( gMuxAddress & 0x0004 )    //     SETUP MUX ADDRESS BITS
       MUX_A2 = 1;
else
       MUX_A2 = 0;

SetipOCRegisters();           //     CONFIGURE OC REGISTERS AND START NEXT PWM BANK

T3ON = 1;                     //     START TIMER 3
}



// ***********************************************************************************
// WHEN DESIGNING THE p.Brain-u24 MODULE, IT IS NOT ALWAYS
// POSSIBLE TO ROUTE THE PWM OUTPUTS TO THE MOTHERBOARD AREA THAT IS DESIRED, THEREFORE
// IT IS NECESSARY TO RE-MAP THE SERVO OUTPUTS IN SOFTWARE.
// ***********************************************************************************


void   DistributeServoOutputs( WORD *pServoData )
{
WORD   lT;

for( lT = 0; lT < 24; lT++ )          // LOOP
                                      // PLACE SERVO DATA INTO RE-MAPPED LOCATION
       gServoPos[ cServoRemap[ lT ] ] = *pServoData++;

}
```

Legal

Please read fully before purchasing any merchandise from micromagic systems ltd.

PRODUCTS

In no event shall micromagic systems be liable for any claim for incidental, consequential damages, or any injuries sustained due to the use of or improper use of products and / or kits purchased out of or in connection thereof with the manufacture, sale, delivery or use of any product in this catalogue or web site. All micromagic systems products purchased should NOT be used for medical, life-saving, life-support, or any applications that could cause injury, dangerous / hazardous situations or consequential damages resulting from the use of the mechanical, hardware or software products sold or represented by micromagic systems.

All products sold by micromagic systems are for Self Learning Experiences, and for Safe Entertainment.

Prices and Specifications

Note
Product specifications, prices listed and availability of items in our web site and in our printed catalogue are subject to change without notice. All prices shown in our web site and in our printed catalogue are believed accurate at time of publication, but subject to change without any notice. We will always advise you of the new price increase and seek your approval before processing any orders you place.

General Product Terms and Conditions

Call micromagic systems before returning any items.

PRODUCT RETURNS POLICY
All sales are final.
New products are warranted for 30 days. Any return for repair or replacement must be pre-authorized by micromagic systems and under no circumstance will returns be accepted unless so authorized. Return Products under warranty must be pre-approved by MMS and sent via certified mail, prepaid and insured, for your protection. (Please note we cannot refund shipping fees). All electronic kit sales are final. Due to the fact that components of the kit may be damaged during assembly we do not accept returns or refunds on any electronic kits

If you receive damaged merchandise, you must contact micromagic systems within 2 days of receipt of your original order. Specify clearly the reason for your refusal. We will exchange returned merchandise for same new merchandise, or for the item sterling amount within 7 days once we receive the returned damaged items from you. Proof of mailing is advised, as we cannot be held responsible for loss of the returned merchandise in mail transit. All return postage is non-refundable. The merchandise, including packing and wrapping material, being returned should be in the same condition as when you received them. Please contact us via e-mail at **matt@micromagicsysstems.com**. Defective merchandise will be replaced (No cash will be refunded). We reserve the right to refuse to replace any merchandise, which our micromagic systems technicians determine to be damaged by the user, or through inappropriate use of that merchandise.

WARRANTY POLICY
We guarantee all products except electronic kits to be free of defects in workmanship and material for 30 days from the purchase, delivery date. We will repair or replace non-electronic kits (No cash will be refunded), at our option providing there is no evidence of customer misuse or alteration to that product item.

micromagic systems carries a limited 30 day warranty on most all items, some items carry an additional number of warranty days or special restrictions. If you want specific warranty information about a product contact micromagic systems to obtain that information.

We are not able to offer any refunds or accept returns for the following items and products: Electronic Kits.

CANCELLATION POLICY
Please be aware that if you cancel an order you may be responsible for restocking fees and / or shipping charges, including charges for return shipping. Cancelled orders are subject to a 25% or £10.00 minimum restocking fee. Orders cancelled within 24 hours of order placement will not be subject to restocking fees however this does not apply to orders with Express Shipping and Handling.