

```

/*****
/*                               IODrive2007DAC.C                               */
/*-----*/
/*   Task           : Data Acquisition Lib for IO Drive 2000 Card   */
/*-----*/
/*   Author          : Flaxer Eli                                     */
/*   Developed on    : 01/08/2004                                     */
/*   Last update     : 30/06/2007                                     */
*****/

```

```

/*****
    Digital Parameters:
        Data = (0..255)

    Analog Parameters:
        Output Chan = (0, 1)
        Input Chan  = (0..7)

    PWM Parameters:
        Duty Cycle  = (0..255)
        Output Chan = (0, 1)
        Bit Number  = (1..4)

    CLK Parameters:
        Count Range = (0..2^16)
        Input Chan  = (1, 2)

    Timer Parameters:
        Count Range = (0..2^16)
*****/

```

```
typedef unsigned char byte;
```

```
#include "EppSppDriver.h"
```

```

#define READ_TIME_OUT    1000
#define TIME_OUT_ERROR   -100.0
#define MAX_VOLT         9.995           // 10*2047/2048
#define MIN_VOLT         -10.000        // 10*2048/2048
#define MAX_VOLT_IN      10.000

```

```

#define ADCL              0x00
#define ADCH              0x01
#define ADCS              0x02
#define ADCINX            0x03
#define DAC0L             0x04
#define DAC0H             0x05
#define DAC1L             0x06
#define DAC1H             0x07
#define DIGITAL           0x08
#define PWM0              0x09
#define PWM1              0x0A
#define CONTROL           0x0B
#define CODE              0x0C
#define CNTC              0x0D
#define LATCH_L           0x0E
#define LATCH_H           0x0F

```

```
static unsigned char ControlReg;
static char AdcInxReg = 0;
```

```

/*****
void setParallelPortMode(int M)
{
    setPortMode(M);
}
/*****
char * getParallelPortMode()
{
    return(getPortMode());
}

```

```

}
/*****
int setParallelPortAddr(unsigned addr)
{
    setBaseAddr(addr);
    return (addr);
}
/*****
unsigned getParallelPortAddr()
{
    return(getBaseAddr());
}
/*****
void initCard()
{
    resetCard();
}
/*****
void DigitalOut(byte data)
{
    setAddr(DIGITAL);
    writeData(data);
}
/*****
byte DigitalIn()
{
    setAddr(DIGITAL);
    return(readData());
}
/*****
void AnalogOut(byte Chan, double Volt)
{
    int Temp;
    byte Lo, Hi;

    byte outChan = (Chan == 0) ? DAC0L : DAC1L;

    if (Volt > MAX_VOLT)
        Volt = MAX_VOLT;
    else if (Volt < MIN_VOLT)
        Volt = MIN_VOLT;

    Temp = (int)(Volt / 10.0 * 2048);
    Lo = Temp;
    Hi = ((Temp >> 8) ^ 0x08) & 0x0F;
    setAddr(outChan);
    writeData(Lo);
    setAddr(outChan+1);
    writeData(Hi);
}
/*****
void SetAnalogInx(char Inx)
{
    AdcInxReg = Inx & 0x01;
    setAddr(ADCINX);
    writeData(AdcInxReg);
}
/*****
double AnalogIn(byte Chan)
{
#define BIP 1 // 1 - Bipolar 0 - Unipolar
#define RNG 1 // 1 - 10V 0 - 5V
#define ACQ 0 // 0 - Internal Acquisition 1 - External
    Acquisition
#define PD10 1 // Normal Operation & Internal Clock

    short temp; // must be short for << operation
    byte lo, hi; // must be byte unless the the sign of lo will
        expanded
    int time = READ_TIME_OUT;
    setAddr(ADCS);
    writeData( (PD10 << 6) | (ACQ << 5) | (RNG << 4) | (BIP << 3) | (Chan &

```

```

    0x07) );

    //while (readData() & 0x01) // Check the Max197 Int on bit 0 or 1
    while (readData() & (1 << AdcInxReg)) // Check the Max197 Int on bit 0 or
    1
        if (time-- == 0) return (TIME_OUT_ERROR);

    setAddr(ADCL);
    lo = readData();
    setAddr(ADCH);
    hi = readData();

    temp = (short)((hi<<8) | lo);
    temp <= 4; temp >= 4; // store the sign bit

    return( (double)(temp * MAX_VOLT_IN / 2048.0) );
}
/*****
void PwmWriteDutyCycle(char Chan, unsigned char Duty)
{
    unsigned char outChan = (Chan == 0) ? PWM0 : PWM1;
    setAddr(outChan);
    writeData(Duty);
}
/*****
void PWM0Enable(char Out1, char Out2)
{
    unsigned char TempReg;

    Out1 = Out1 ? 1 : 0;
    Out2 = Out2 ? 1 : 0;
    TempReg = (Out2 << 1) | Out1;

    ControlReg &= 0xFC;
    ControlReg |= TempReg;
    setAddr(CONTROL);
    writeData(ControlReg);
}
/*****
void PWM1Enable(char Out3, char Out4)
{
    unsigned char TempReg;

    Out3 = Out3 ? 1 : 0;
    Out4 = Out4 ? 1 : 0;
    TempReg = (Out4 << 3) | (Out3 << 2);

    ControlReg &= 0xF3;
    ControlReg |= TempReg;
    setAddr(CONTROL);
    writeData(ControlReg);
}
/*****
void Clk1Enable(char OnOff)
{
    int Bbit = OnOff ? 1 : 0;
    ControlReg &= 0xEF;
    ControlReg |= (Bbit << 4);
    setAddr(CONTROL);
    writeData(ControlReg);
}
/*****
void Clk2Enable(char OnOff)
{
    int Bbit = OnOff ? 1 : 0;
    ControlReg &= 0xDF;
    ControlReg |= (Bbit << 5);
    setAddr(CONTROL);
    writeData(ControlReg);
}
/*****
void BuzzerEnable(char OnOff)

```

```

    {
        int Bbit = OnOff ? 1 : 0;
        ControlReg &= 0x7F;
        ControlReg |= (Bbit << 7);
        setAddr(CONTROL);
        writeData(ControlReg);
    }
}
/*****/
void RstCNT(char Rst1, char Rst2)
{
    unsigned char TempReg;
    Rst1 = Rst1 ? 1 : 0;
    Rst2 = Rst2 ? 1 : 0;
    TempReg = (Rst2 << 1) | Rst1;

    setAddr(CNTC);
    writeData(TempReg);
}
/*****/
void LatchCNT(char Clk)
{
    unsigned char TempReg;
    if (Clk < 1 || Clk > 2)
        return;

    TempReg = (1 << (Clk+1));
    setAddr(CNTC);
    writeData(TempReg);
}
/*****/
unsigned short ReadCNT(char Clk)
{
    unsigned char lo, hi;
    unsigned short Temp;

    if (Clk < 1 || Clk > 2)
        return 0;

    LatchCNT(Clk);

    setAddr(LATCH L);
    lo = readData();
    setAddr(LATCH H);
    hi = readData();

    Temp = (unsigned short)((hi<<8) | lo);

    return(Temp);
}
/*****/
int IsConnected()
{
    setAddr(CODE);
    return(readData() == 0x55); // Return 0x55 if detect the card
}
/*****/

```