```
/******************************************************************************
            Project: JHU Resolver Electronics
------------------------------------------------------------------------------
** File Name:       "Resolver_main.c"
**
** Description:     Main control loop for Resolver Electronics control software.
**
**
**  Author:         Joe Orndorff - Johns Hopkins University
**
** Creation Date:   May 17, 2004
** Revisions:
*
******************************************************************************/
/*
   The following files are used and compiled with the Keil uVision2 Version 2.12 IDE:

89C51RC2.h


These are UVision Files used by the compiler:
STARTUP-RC2.A51

******************************************************************************/

#include "Resolver.h"
#include "globals.h"
//#include "string.h"

/******************************/
    void main(void)
/******************************/
{

//unsigned char keyed;
while(1)
    {
    initialize_Resolver();              // system initialization cmd
    R2Dname = EE_read_byte(EER2DADD);   // read R2D address char and store in global
    LED_toggle(200);
    Read_R2D(); //take initial reading
    Read_R2D(); //take initial reading
    user_mode();
    }//end while
}

/****************************************/
  float Read_R2D(void)
/****************************************/
{
float angle;
unsigned int Reading;
xdata float position;
xdata float scalefact;
xdata float origin;
xdata unsigned char direction;
xdata unsigned char polarity;

xdata float amp;
xdata float phase;
xdata float freq;
xdata unsigned char sin_cor_en;


origin = Read_Float_From_EE(EE_ORIGIN_ADD);
scalefact = Read_Float_From_EE(EE_SCALEFACT_ADD);
direction = EE_read_byte(EE_DIR_ADD);
polarity = EE_read_byte(EE_BIPOL_ADD);


amp = Read_Float_From_EE(EE_SIN_AMP_ADD);
phase = Read_Float_From_EE(EE_SIN_PHASE_ADD);
freq = Read_Float_From_EE(EE_SIN_FREQ_ADD);
sin_cor_en = EE_read_byte(EE_EN_SINE_CORR_ADD);

Reading = Read_R2D_Raw();

angle =  (360 * (float)Reading/65536);
```

```
//RS485_Driver(ON);//turn on RS485 Driver
//printf("byte1 = %Bu, byte2 = %Bu Combined bytes = %d, angle = \n",R2D_Reading.byte_1, R2D_Reading.byte_0, Readi
ng,angle);
//RS485_Driver(OFF);//turn on RS485 Driver

if(sin_cor_en == TRUE) //add sine ftn correction factor
    {
    angle = angle + (amp * sin((angle/freq) + phase));
    }

if(direction == 1)
    {
    angle = 360 - angle; //reverse direction
    }
if(polarity == 1)
    {
    angle = angle -180; //now reads from -180 to +180
    }
position = (angle * scalefact) + origin;

return(position);
}

/*****************************************/
  unsigned int Read_R2D_Raw(void)
/*****************************************/
{
struct word16 R2D_Reading;
unsigned int Reading;
//unsigned int old_reading;
//unsigned char hits;
//  hits = 0;
    Reading = 0;
while(Reading ==0)
    {
    BYTESEL = 0; // set to read LSByte
    nINHIBIT = 0; //stop conversion
    delay(2); //need 600nS for data to set up
    nENABLE = 0; // enable the output latches
    R2D_Reading.byte_1 = P0; //LSByte
    BYTESEL = 1; // read MSByte
    R2D_Reading.byte_0 = P0; //MSByte
    nINHIBIT = 1; // allow conversion to proceed
    nENABLE = 1; // disable the outputs
    Reading = (R2D_Reading.byte_0 * 256) + R2D_Reading.byte_1; //combine bytes
    if(Reading == 0)
        delay(20); // allow time for new reading
    }
return(Reading);
}

/*****************************************/
   void LED_toggle(unsigned int wait_time)
/*****************************************/
{
/* function toggles test led on and off by
   amount specified in wait_time */

LED_TEST = ON;
delay (wait_time);
LED_TEST = OFF;
}



/*****************************************************************/
 // void msg(unsigned int msgidx)
/*****************************************************************/
//{
/*
  this function just displays one line of text
  to the LCD or serial port
*/

//printf("%s",general_mnu[msgidx]);
//}
```

```
/*******************************
 * Utilities.c                  *
 * Rev 6-18-2004                 *
 * Johns Hopkins University      *
 *******************************/

#include "Resolver.h"
#include "eeprom_map.h"
#include "globals.h"

/****************************************/
  void delay (unsigned int milliseconds)
/****************************************/
/*
  this function assumes an oscillator
  frequency of 11.0592Mhz
  to calculate TH0 and TL0 register
  values:
  assume a delay of 1ms
  TH0/TL0 = 65536 - ((delay x OSC frequency)/12)
  convert to hex and divide the bytes between
  the TH0 and TL0 registers

 THIS DELAY FUNCTION CAN NOT RUN IN AN 'ISR' -Interrupt Service Routine
*/

{
unsigned int i;

TMOD |= 0x01;                    // configure timer 0 for 16 bit
    for(i=0;i<milliseconds;i++) // wait #of milliseconds
        {
            TH0 = 0xFC;         // load the 16bit timer High Byte
            TL0 = 0x66;         // load the 16bit timer low Byte
            TR0=1;              // start timing
            while (!TF0);       // Wait for Overflow to be set
            TF0=0;              // reset overflow
            TR0=0;              // stop timer
        }
}


/****************************************/
 void get_serial_string(char string[50])
/****************************************/
{
/*
reads input from terminal until a CR is encountered. Only allows
alpha-numerica, white space, +/- chars to be entered.
*/
char inkey = '\0';
xdata unsigned char exit = 0;
xdata int i = 0;

string[0] = '\0'; //initialize to empty string

while(!exit && i<=48)
    {
    inkey = _getkey();
    switch(inkey)
        {
        case 0x00a: // Line Feed (^J)
            if(nPARTY == 0) // in party mode
                {
                string[i] = inkey;
                string[i+1] = '\0';
                exit = 1;
                }
            break;
        case 0x00d: //Carriage Return (^M)
            if(nPARTY == 1) // in single mode
                {
                string[i] = inkey;
                string[i+1] = '\0';
                exit = 1;
                }
            break;
        case 0x003: // ^C just return the ctrl C character
            exit = 1;
```

```
            string[0] = 0x003;
            string[1] = '\0';
            break;
        default:
            if(isalnum(inkey) || isspace(inkey) || inkey == '.' || inkey == '+' || inkey == '-')
                {
                string[i] = inkey;
                string[i+1] = '\0';
                if(nPARTY == 0) //in party mode only echo if R2Dname is detected
                    {
                    if(string[0] == R2Dname) //R2Dname detected
                        {
                        RS485_Driver(ON);
                        putchar(inkey);
                        RS485_Driver(OFF);
                        } //end if
                    } //end if

                else // not in party mode
                    {
                    RS485_Driver(ON);
                    putchar(inkey);
                    RS485_Driver(OFF);
                    }//end else
                i++;
                }//end if
            break;
        } // end switch
    } //end while
} //end ftn
```

```
#include "Resolver.h"
#include "globals.h"
#define CMD_STR_SIZE 50

/*******************************/
    void user_mode(void)
/*******************************/
{
xdata unsigned char exit = 0;
xdata char InString[CMD_STR_SIZE] = ""; //string input from serial port
xdata unsigned char keyed;

if (nPARTY != 0) //if not in PARTY mode
    {
    while (keyed != 0x020) //loop until spacebar encountered
        {
        keyed = getchar(); //read char from PC
        switch(keyed)
            {
            case 'a': case 'b': case 'c': case 'd': case 'e': case 'f': case 'g':
            case 'h': case 'i': case 'j': case 'k': case 'l': case 'm': case 'n':
            case 'o': case 'p': case 'q': case 'r': case 's': case 't': case 'u':
            case 'v': case 'w': case 'x': case 'y': case 'z':
            case 'A': case 'B': case 'C': case 'D': case 'E': case 'F': case 'G':
            case 'H': case 'I': case 'J': case 'K': case 'L': case 'M': case 'N':
            case 'O': case 'P': case 'Q': case 'R': case 'S': case 'T': case 'U':
            case 'V': case 'W': case 'X': case 'Y': case 'Z':
                R2Dname = keyed; // if not spacebar, user is setting R2D board address
                break;
            case 0x020:
                RS485_Driver(ON);//turn on RS485 Driver
                printf("Resolver Readout Version: %s\n",VERSION);
                RS485_Driver(OFF);//turn off RS485 Driver
                break;
            default:
                break;
            } //end switch
        } // end while
    }// end if

while(!exit)
    {
    get_serial_string(InString);
    if(InString[0] == 0x003 || InString[0] == 0x01B) //Exit and reset on ^C or ESC
        {
        exit = 1;
        }
    if(nPARTY == 0) // in party mode
        {
        if(InString[0] == R2Dname) //is first character proper address
            {
            StripOffR2Dname(InString); // remove indexer name from string
            if(!exit) ExecuteCMD(InString); // only execute cmd if no errros
            }
        }
    if(nPARTY != 0) // in single mode
        {
        if(!exit) ExecuteCMD(InString); // only execute cmd if no errros
        }
    }
}

/****************************************/
 void StripOffR2Dname(char string[50])
/****************************************/
{

xdata unsigned char i = 0, j = 0;
unsigned char done = 0;

//remove first character (indexer name)
for(i=0;i<CMD_STR_SIZE;i++)
    {
    string[i] = string[i+1];
    }
//remove any white characters
while((!done) && (j<CMD_STR_SIZE))
    {
    if(isspace(string[0])) //chck for whitespace char
```

```
        {
        for(i=0;i<CMD_STR_SIZE;i++)
            {
            string[i] = string[i+1]; //rotate chars left
            }
        }
    else done=1;
    j++;
    }
} //end ftn


/****************************************/
 void ExecuteCMD(char InString[50])
/****************************************/
{
xdata float position; //Resolver angle to be sent to serial port
xdata float fparam;
xdata unsigned char iparam;
xdata unsigned int Raw;
fparam = 0.00;
iparam = 0;
position = 0;

    switch(InString[0])
        {
        case 'C':
        case 'c'://clear and restore
            Clear_And_Restore();
            break;
        case 'Z': // read position
        case 'z':
            position = Read_R2D();
            RS485_Driver(ON);//turn on RS485 Driver
            printf("\t%f",position);
            RS485_Driver(OFF);//turn on RS485 Driver
            break;
        case 'Q': // read position
        case 'q':
            Raw = Read_R2D_Raw();
            RS485_Driver(ON);//turn on RS485 Driver
            printf("\t%d",Raw);
            RS485_Driver(OFF);//turn on RS485 Driver
            break;
        case 'S': //store address
        case 's':
            EE_write_byte(EER2DADD, R2Dname); //store new R2D address in EEPROM
            break;
        case 'X': //examine settings
        case 'x':
            RS485_Driver(ON);//turn on RS485 Driver
            printf("\taddress = %c \n",EE_read_byte(EER2DADD)); //read R2D address from EEPROM
            fparam = Read_Float_From_EE(EE_ORIGIN_ADD);
            printf("\torigin = %f\n",fparam);
            fparam = Read_Float_From_EE(EE_SCALEFACT_ADD);
            printf("\tScaleFactor = %f\n",fparam);
            iparam = EE_read_byte(EE_DIR_ADD);
            printf("\tDirection = %Bu\n",iparam);
            iparam = EE_read_byte(EE_BIPOL_ADD);
            printf("\tBiPolar = %Bu\n",iparam);
            fparam = Read_Float_From_EE(EE_SIN_AMP_ADD);
            printf("\tSine Correction Amplitude = %f\n",fparam);
            fparam = Read_Float_From_EE(EE_SIN_PHASE_ADD);
            printf("\tSine Correction Phase = %f\n",fparam);
            fparam = Read_Float_From_EE(EE_SIN_FREQ_ADD);
            printf("\tSine Correction Frequency = %f\n",fparam);
            iparam = EE_read_byte(EE_EN_SINE_CORR_ADD);
            printf("\tUse Sine Correction (1 = yes) = %Bu\n",iparam);
            RS485_Driver(OFF);//turn on RS485 Driver
            break;
        case 0x00d: //carriage return
        case 0x003: //^C
        case 0x01B: //esc
            break;
        case 'O': // set origin
        case 'o':
            fparam = Strip_Parameter1(InString);
            Store_Float_to_EE(fparam,EE_ORIGIN_ADD);
            break;
```

```
        case 'F': // set scaling factor
        case 'f':
            fparam = Strip_Parameter1(InString);
            if(fparam == 0) fparam = 1; //zero not allowed
            Store_Float_to_EE(fparam,EE_SCALEFACT_ADD); //save to EE prom
            break;
        case 'd':
        case 'D': // set direction (0= plus, else = negative)
            iparam = (unsigned char)Strip_Parameter1(InString);
            if(iparam != 0) iparam = 1; //force to zero or 1
            EE_write_byte(EE_DIR_ADD, iparam);
            break;
        case 'p':
        case 'P': // set Polarity (0= single ended esle= BiPolar)
            iparam = (unsigned char)Strip_Parameter1(InString);
            if(iparam != 0) iparam = 1; //force to zero or 1
            EE_write_byte(EE_BIPOL_ADD, iparam);
            break;
        case 'A':// Set Amplitude of Sine Correction
        case 'a':
            fparam = Strip_Parameter1(InString);
            Store_Float_to_EE(fparam,EE_SIN_AMP_ADD); //save to EE prom
            break;
        case 'B':// Set Phase of Sine Correction
        case 'b':
            fparam = Strip_Parameter1(InString);
            Store_Float_to_EE(fparam,EE_SIN_PHASE_ADD); //save to EE prom
            break;
        case 'E': // Set Frequency of Sine Correction
        case 'e':
            fparam = Strip_Parameter1(InString);
            if(fparam == 0) fparam = 1; //zero not allowed
            Store_Float_to_EE(fparam,EE_SIN_FREQ_ADD); //save to EE prom
            break;
        case 'u':
        case 'U': // Use Sine Correcton (0= Not Used esle= Used)
            iparam = (unsigned char)Strip_Parameter1(InString);
            if(iparam != 0) iparam = 1; //force to zero or 1
            EE_write_byte(EE_EN_SINE_CORR_ADD, iparam);
            break;


        default:
            break;

        } // end switch

RS485_Driver(ON);//turn on RS485 Driver
printf("\n");
RS485_Driver(OFF);//turn off RS485 Driver
}

/************************************************/
 float Strip_Parameter1(char string[CMD_STR_SIZE])
/************************************************/
{
xdata char c;
xdata unsigned char i;
xdata unsigned char exit;
xdata unsigned char strParam [] = "0.00";
xdata float param1;

param1 = 0.00; //set default

StripOffR2Dname(string); // in this case we're stripping off the CMD letter and any spaces

exit = 0;
i = 0;
c = " ";

while(c != '\0' && !exit)
    {
    c = string[i]; //grab char from string
    if(!isdigit(c) && (c != '.') && (c != '+') && (c != '-')) //make sure it is a number or decimal
        {
        strParam[i] = '\0'; //add terminator
        exit = 1; //exit when do digits are left to read.
        }
    else
```

```
        {
        strParam[i] = c; //if it is a number add it to string
        }
    i++;
    }
param1 = atof(strParam);
return(param1);
}


/**********************************************/
 void Store_Float_to_EE(float value,int address)
/**********************************************/
{
/*
  Stores Origin in EEPROM
*/

unsigned char i;
xdata unsigned char *cptr;
xdata unsigned char record;



cptr =(unsigned char*)&value;
//address = EE_ORIGIN_ADD;
for(i=0;i<sizeof(float);i++)
    {
    record = *cptr++;
    EE_write_byte(address++ ,record);
    }
}

/**********************************************/
 void Clear_And_Restore(void)
/**********************************************/
{
/*
  Resets settings to default values. Does not effect
  the address setting
*/
float fparam;
unsigned char iparam;
// set origin
fparam = 0;
Store_Float_to_EE(fparam,EE_ORIGIN_ADD);

// set scaling factor
fparam = 1;
Store_Float_to_EE(fparam,EE_SCALEFACT_ADD); //save to EE prom

// set direction (0= plus, else = negative)
iparam = 0;
EE_write_byte(EE_DIR_ADD, iparam);

// set Polarity (0= single ended esle= BiPolar)
iparam = 0;
EE_write_byte(EE_BIPOL_ADD, iparam);

// Set Amplitude of Sine Correction
fparam = 0;
Store_Float_to_EE(fparam,EE_SIN_AMP_ADD); //save to EE prom
// Set Phase of Sine Correction
fparam = 0;
Store_Float_to_EE(fparam,EE_SIN_PHASE_ADD); //save to EE prom

// Set Frequency of Sine Correction
fparam = 1;
Store_Float_to_EE(fparam,EE_SIN_FREQ_ADD); //save to EE prom

// Enable/Disable Sine Correction (0= Disabled esle = Enabled)
iparam = 0;
EE_write_byte(EE_EN_SINE_CORR_ADD, iparam);
}

/****************************************/
 float Read_Float_From_EE(unsigned int address)
/****************************************/
{
```

```
unsigned char i;
xdata unsigned char *cptr;
xdata float value;
xdata unsigned char record;

cptr =(unsigned char*)&value;
for(i=0;i<sizeof(float);i++)
    {
    record = EE_read_byte(address +i);
    *cptr++ = record;
    }

return value;

}
```

```
/******************************************************************************
            Project: JHU Resolver Electronics
-----------------------------------------------------------------------------
** File Name:       "SPI.c"
**
** Description:     functions to control the SPI bus
**                  These functions assume a uP clock of 11.0592MHz
**                  Delays may have to be scaled for faster clock
**
**  Author:         Joe Orndorff
**
** Creation Date:   May 18, 2004
** Revisions:
******************************************************************************/

#include "Resolver.h"
/************************************/
    unsigned char SPI_read(void)
/************************************/
{
unsigned char i,input_data;

input_data = 0x00;      // initialize
for(i=0;i<8;i++)        // receive 8 bits from the I2C Bus
    {
    input_data <<= 1;   // Shift the byte by one bit
    SCLK = 1;           // Clock the data into the I2C Bus
    input_data |= SDO;  // Input the data from the I2C Bus
    _nop_();            // allow setup time
    SCLK = 0;           // set clock low
    }
return input_data;
}


/**************************************************/
    void SPI_write(unsigned char output_data)
/**************************************************/
{
unsigned char i;
for(i=0;i<8;i++)        // Send 8 bits to the I2C Bus
    {
    SDI = ((output_data & 0x80) ? 1 : 0);
    output_data  <<= 1;                     // Shift the byte by one bit
    SCLK = 1;                               // Clock the data into the I2C Bus
    _nop_();
    SCLK = 0;
    }
}
```

```
/*********************************************
 * File:  Resolver.h
 *
 * Joe Orndorff / Johns Hopkins University.
  *********************************************/
//#include "string.h"
#include "type_defs.h"
#include "89c51rd2.h"   // controller definitions
#include "eeprom_map.h"
#include <stdio.h>      // prototype declarations for I/O functions
#include <intrins.h>    // instrinsic library
#include <math.h>       // math library
#include <ctype.h> // for character type checking routines
#include <stdlib.h> // for type conversion functions 'atof()' etc
#include <string.h>

#define VERSION "VER 0.13 " //software revision number

/* AT25256 constants */
#define EEPROM_LAST_ADDR 32768
#define WREN 0x06
#define WRDI 0x04
#define RDSR 0x05
#define WRSR 0x01
#define READ 0x03
#define WRITE 0x02



/* logic */
#define ENABLED 0
#define DISABLED 1
#define ON 1
#define OFF 0
#define TRUE 1
#define FALSE 0


/* PORT 1 assignments */

sbit  SPARE4    =   P1^7;
sbit  SPARE3    =   P1^6;
sbit  nPARTY    =   P1^5;
sbit  DRIVER_EN =   P1^4;
sbit  SDI       =   P1^2; // serial data in
sbit  SDO       =   P1^3; // serial data out
sbit  SCLK      =   P1^1; // serial clock for IC2
sbit  LED_TEST  =   P1^0; // Tuns on test LED

/* PORT 2 assignments */
sbit  LOS       =   P2^7; // Loss of signal
sbit  DIRECTION =   P2^6; // Rotation Direction
sbit  BUSY      =   P2^5; // R to D is busy
sbit  nENABLE   =   P2^4; // Enable R to D readout
sbit  nCOMP     =   P2^3; // compliment data
sbit  DLOAD     =   P2^2; // preload R to D data registers
sbit  BYTESEL   =   P2^1; // select upper/lower byte
sbit  nINHIBIT  =   P2^0; // set R to D to

/* port 3 assignments */

sbit SPARE6     =   P3^5;
sbit nSEL256    =   P3^4;


/* hardware */
unsigned int Read_R2D_Raw(void);
void RS485_Driver(unsigned char);
void ExecuteCMD(char*);
void StripOffR2Dname(char*);
void get_serial_string(char*);
void single_mode(void);
void user_mode(void);
void initialize_serial(void);
void initialize_Resolver(void);     // system initialization cmd
void initialize_89C51RD2(void);
float Read_R2D(void);
void Store_Float_to_EE(float,int);
float Strip_Parameter1(char*);
```

```
float Read_Float_From_EE(unsigned int);
void Clear_And_Restore(void);

/* utility function prototypes */
void delay (unsigned int);
//void splash_screen(void);
void LED_toggle(unsigned int wait_time);

/* SPI bus functions */
unsigned char SPI_read(void);
void SPI_write(unsigned char);

/* EEPROM functions */
unsigned char EE_read_byte(unsigned int);
void EE_write_byte(unsigned int, unsigned char);
//void EE_write_page(unsigned int, unsigned char[],unsigned char);
unsigned char SPI_read(void);
void SPI_write(unsigned char output_data);
unsigned char EE_write_status(unsigned char);
unsigned char EE_read_status(void);
//void EE_enable_write(void);
//void EE_disable_write(void);
```

```
/******************************************************************************
            Project: JHU Resolver Electronis
------------------------------------------------------------------------------
** File Name:       "Hardware.c"
**
** Description:     General hardware interface routines that
**                  do not fit nicely anywhere else
**
**  Author:         Joe Orndorff
**
** Creation Date:   May 18, 2004
** Revisions:
******************************************************************************/

#include "Resolver.h"
#include "globals.h"

/****************************************/
  void initialize_Resolver(void)
/****************************************/
{
/* set up Auxiliary Register */
AUXR        = 0x0D;     // AO=1 (disable ALE),XRS0=1,XRS1=1,M0=0
error_reg   = 0x00;     // clear error register
EA = 0;                 // Disable all interrupts
initialize_89C51RD2();  // initialize micro
initialize_serial();    // initialize serial port
nENABLE     = ENABLED;  // Enable R to D
DLOAD       = 1;        // set R to D to output data
BYTESEL     = 1;        // Select LSB
nINHIBIT    = 1;        // Set inhibit High
nCOMP       = 1;        // Don't do n Comp
DRIVER_EN   = 0;        // Disable RS485 Driver
LED_TEST    = OFF;      //start with LED off

//EA    = 1;    //enable selected interrupts
//EX1 = 1;  //enable OFF switch interrupt
//EX0 = 1;  //enable LOWBAT interrupt
IT0 = 0;    //set int0 to level sensitivity
IT1 = 0;    // set int1 to level sensitivity



}

/*******************************************************************/
    void initialize_serial(void)
/*******************************************************************/
{
unsigned char i;

SCON    = 0x72; //8 bit UART

/*
 Set TI to enable TXSER UART function. Transmission Rate is 9600,N,8,1
*/

TMOD    = 0x20; // Use TIMER1, 8-bit auto-reload
TH1     = 0xFD; // Initialize Auto-reload to FD for 11.059MHz external clock
PCON    = 0x00; // when set we run at 19200 Baud, 9600 when clear
TCON    = 0X40; // Turn internal UART timer on

delay(20);      //allow 20ms for oscillator to start

/*Reset Serial Port*/
SDI = 1;                //SET SDI PIN
for(i=0;i<254;i++)
    {
    SCLK = 0x01;        //SET SCLK
    SCLK = 0x00;        //CLR SCLK
    }
SDI = 0;                //RESET SDI PIN
SCLK = 0x01;            //SET SCLK
SCLK = 0x00;            //CLR SCLK
}

/*******************************/
  void initialize_89C51RD2(void)
/*******************************/
```

```
{
/*
this function cofigures the micro to
initial power on conditions
Ouputs are set to 0, Inputs are set to 1
*/


/* initialize port 0*/
P0          =   0xFF;   //Use port 0 for data inputs

/* initialize port 1*/
SPARE4      =   0; // OUTPUT
SPARE3      =   0; // OUTPUT
nPARTY      =   1; // INPUT
DRIVER_EN   =   0; // OUTPUT
SDI         =   0; //
SDO         =   1; //
SCLK        =   0; // sclock output
LED_TEST    =   0; // output, therefore low

/* initialize port 2 */
LOS         =   1; // Loss of Signal input from R to D
DIRECTION   =   1; // Direction input from R to D
BUSY        =   1; // Busy input from R to D
nENABLE     =   0; // Enables R to D
nCOMP       =   0; // Compliment
DLOAD       =   0; // Data Load to R to D
BYTESEL     =   0; // Byte Select
nINHIBIT    =   0; // R to D conversion inhibit (active low)

/* initialize port 3*/
P3          =   0xFF;   //Use the alternative function of Port 3
nSEL256     =   1;      // disable 256
SPARE6      =   0;      // OUTPUT

/* set up Auxiliary Register */
AUXR        =   0x0D;   //AO=1 (disable ALE),XRS0=1,XRS1=1,M0=0
}

/****************************************/
  void RS485_Driver(unsigned char state)
/****************************************/
{
/*
  Turn LTC491 transmit drive ON/OFF
*/

if(state == ON)
    {
    LED_TEST = 1;
    DRIVER_EN = 1;
    }
else
    {
    delay(1); //allow time for chars to clear LTC491 before shutting off
    DRIVER_EN =0;
    LED_TEST = 0;
    }
}
```

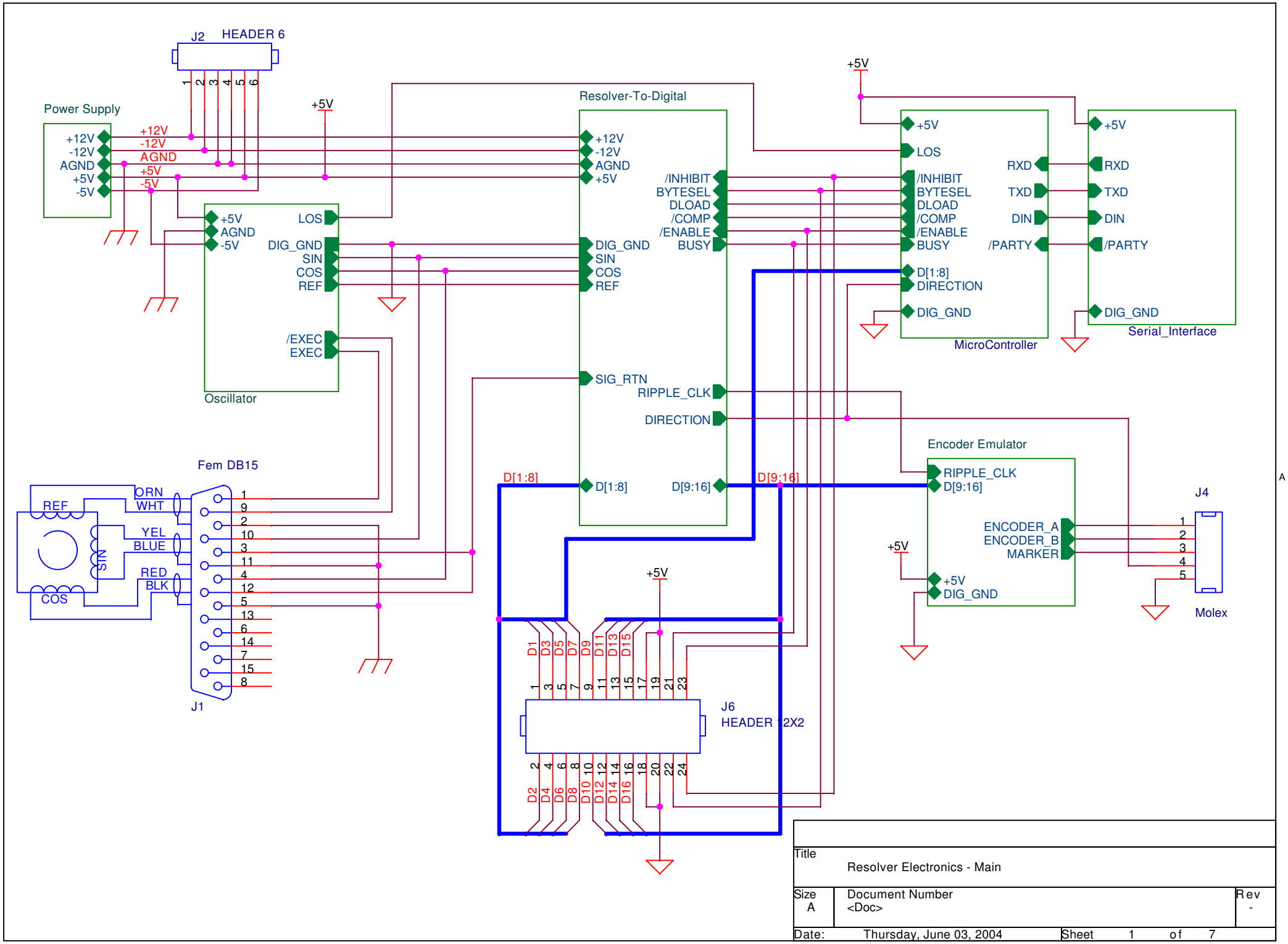
```
/*****************************************************************************
            Project: JHU Resolver Electronics
-----------------------------------------------------------------------------
** File Name:       "globals.c"
**
** Description:     Global vaiables used for the Resolver
**
**
**  Author:         Joe Orndorff
**
** Creation Date:   May 18, 2004
** Revisions:
*****************************************************************************/
/*
  global variables are referenced here
  so they can be easily defined in other
  source code modules. Globals that uses
  internal ram are defined here. Since global
  variables are such nasty buggers, I gave
  them there own file so that they would be
  easily identified.
*/

/* STATUS REGISTER IN BIT SPACE */
extern unsigned char bdata status_reg;

extern bit eeprom_busy; // high during eeprom write


/* ERROR REGISTER IN BIT SPACE */
extern unsigned char bdata error_reg;
extern bit error;



/* GLOBALS IN XRAM */
extern xdata char R2Dname;
```

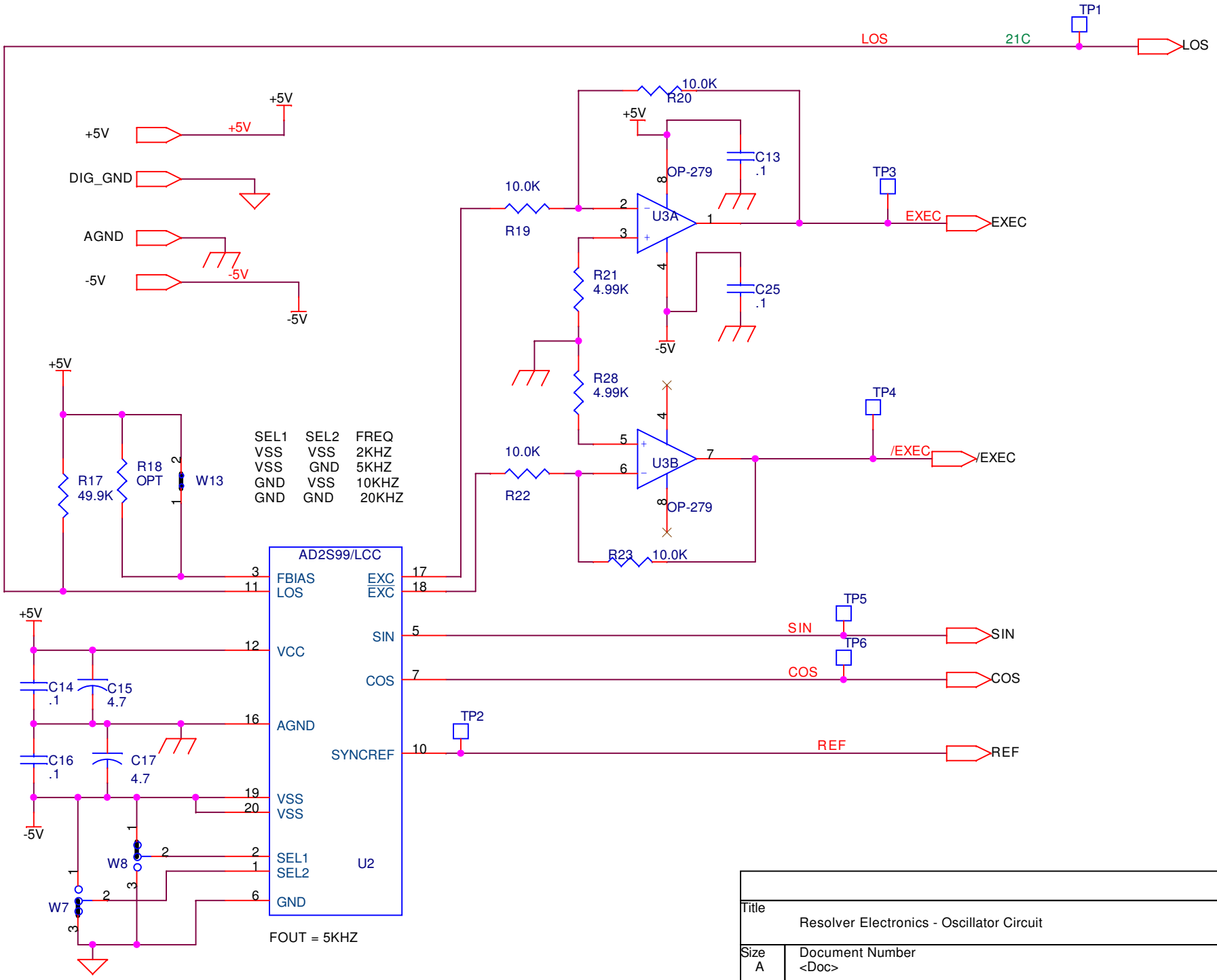
```
/******************************************************************************
            Project: JHU Resolver Electronics
------------------------------------------------------------------------------
** File Name:       "Global_vars.c"
**
** Description:     Keeping these nasty things in their own file
**
**
**  Author:         Joe Orndorff
**
** Creation Date:   May 18, 2004
** Revisions:
******************************************************************************/
#include "type_defs.h"
#include "eeprom_map.h"

/* STATUS REGISTER IN BIT SPACE */
unsigned char bdata     status_reg;     // general 8 bit status register
sbit underline      =   status_reg^0;   // will underline small font text
//sbit blank_done       =   status_reg^1;   // set if blank has been read
sbit eeprom_busy    =   status_reg^2;   // high during eeprom write

/* ERROR REGISTER IN BIT SPACE */
unsigned char bdata     error_reg;      // gerneral 8 bit error register
sbit error          =   error_reg^0;

/* GLOBALS IN DATA RAM */

/* GLOBALS IN XRAM */
xdata unsigned char R2Dname;
//unsigned char *xdata  temp_mnu[NUMTESTS + NUMUSRTESTS + 1]; // temporary storage to hold menu selections
//xdata unsigned char   usr_tst_name[NUMUSRTESTS][16]; //array to hold user testnames read from eeprom
//unsigned char xdata   pc;             //character column pointer
//xdata unsigned int        blank_done_reg; //bit register for all 12 gain/wavelentgh channels
//xdata unsigned int        blank[12];      //blank readings for each channel
```

```
/******************************************************************************
             Project: JHU Resolver Electronics
------------------------------------------------------------------------------
** File Name:  "EEPROM_map.c"
**
** Description:  Memory Map of EEPROM
**
**  Author:   Joe Orndorff
**
** Creation Date: May 18, 2004
** Revisions:
******************************************************************************/

#define CALTABLESIZE        16384 // size of calibration table (For Later Use)
//#define FLOATSIZE            4

#define EER2DADD            16384 // location of R2D board address
#define EE_DIR_ADD          16385 // Address Location holding direction parameter
#define EE_BIPOL_ADD        16386 // Address Locaiton of BiPolar selection variable
#define EE_ORIGIN_ADD       16387 // Address Location of Origin offset variable
#define EE_SCALEFACT_ADD    16391 // Address Location of Scaling Factor
#define EE_SIN_AMP_ADD      16395 // Address Location of Sine Correction Amplituede Variable
#define EE_SIN_PHASE_ADD    16399 // Address Location of Sine Correction Phase Variable
#define EE_SIN_FREQ_ADD     16403 // Address Location of Sine Correction Freq Variable
#define EE_EN_SINE_CORR_ADD 16407 // Address Location of variable that decides to apply sine correction
```

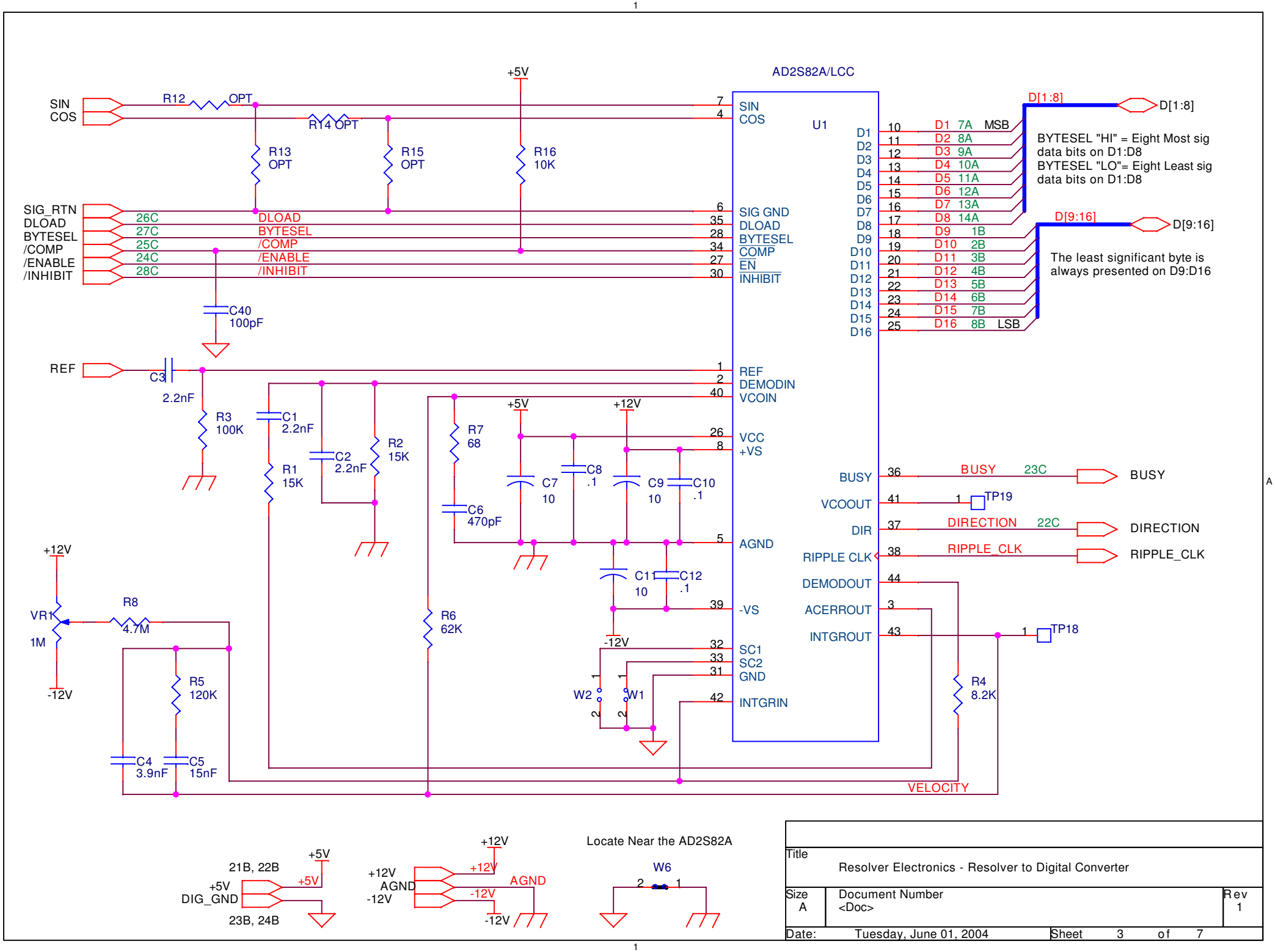
```
/*****************************************************************************
            Project: JHU Resolver Electronics
-----------------------------------------------------------------------------
** File Name:       "EEPROM.c"
**
** Description:     functions to control the AT25128/256 EEPROM
**
**
**
**  Author:         Joe Orndorff
**
** Creation Date:   May 18, 2004
** Revisions:
*****************************************************************************/
#define Resolver_GLOBALS
#include "Resolver.h"
#include "globals.h"

/*
  Note: chip select commands are part of these routines since
        it needs to be toggled for every command issued
        to the EEPROM.
*/
/***************************************************************************/
    unsigned char EE_write_status(unsigned char status)
/***************************************************************************/
{
nSEL256 = ENABLED;
SPI_write(WRSR);        // send write status register CMD
SPI_write(status);      // write status data
nSEL256 = DISABLED;
return status;
}

/***************************************************************************/
    unsigned char EE_read_status(void)
/***************************************************************************/
{
unsigned char status;
nSEL256 = ENABLED;
SPI_write(RDSR);     // send read status register CMD
status = SPI_read(); // read status data
nSEL256 = DISABLED;
return status;
}


/***************************************************************************/
    void EE_write_byte(unsigned int address, unsigned char databyte)
/***************************************************************************/
{
unsigned char status;
unsigned char mask;
/*
  *** note commands are only excecuted after nSEL256 goes high
  *** ...so you have to DISABLE the AT25256 after every CMD
  *** The AT25256 is automatically returned to the write disable
  *** state at the completion of a WRITE cycle
*/

eeprom_busy = 1;            // set busy flag

nSEL256 = ENABLED;          // enable 256
SPI_write(WREN);            // enable write
nSEL256 = DISABLED;         // disable 256
nSEL256 = ENABLED;          // enable 256
SPI_write(WRITE);           // send write command
mask = address >> 8;        // shift right 8 bits
SPI_write(mask);            // send hi add byte
mask = address & 0x00FF;    // mask out hi add byte
SPI_write(mask);            // send lo add byte
SPI_write(databyte);        // send data byte
nSEL256 = DISABLED;         // disable 256
do //wait for write cycle to complete
    {
    status = EE_read_status();
    status &= 0x01;         // check for /RDY
    }while (status);        // wait for status to go low
```

```
eeprom_busy = 0;            // set busy flag
}

/************************************************************/
    unsigned char EE_read_byte(unsigned int address)
/************************************************************/
{
unsigned char mask;
unsigned char databyte;

nSEL256 = ENABLED;
SPI_write(READ);        // send read cmd
mask = address >> 8;    // shift right 8 bits
SPI_write(mask);        // send hi add byte
mask = address & 0x00FF;// mask out hi add byte
SPI_write(mask);        // send lo add byte
databyte = SPI_read();  // read data byte
nSEL256 = DISABLED;
return databyte;
}

}
```

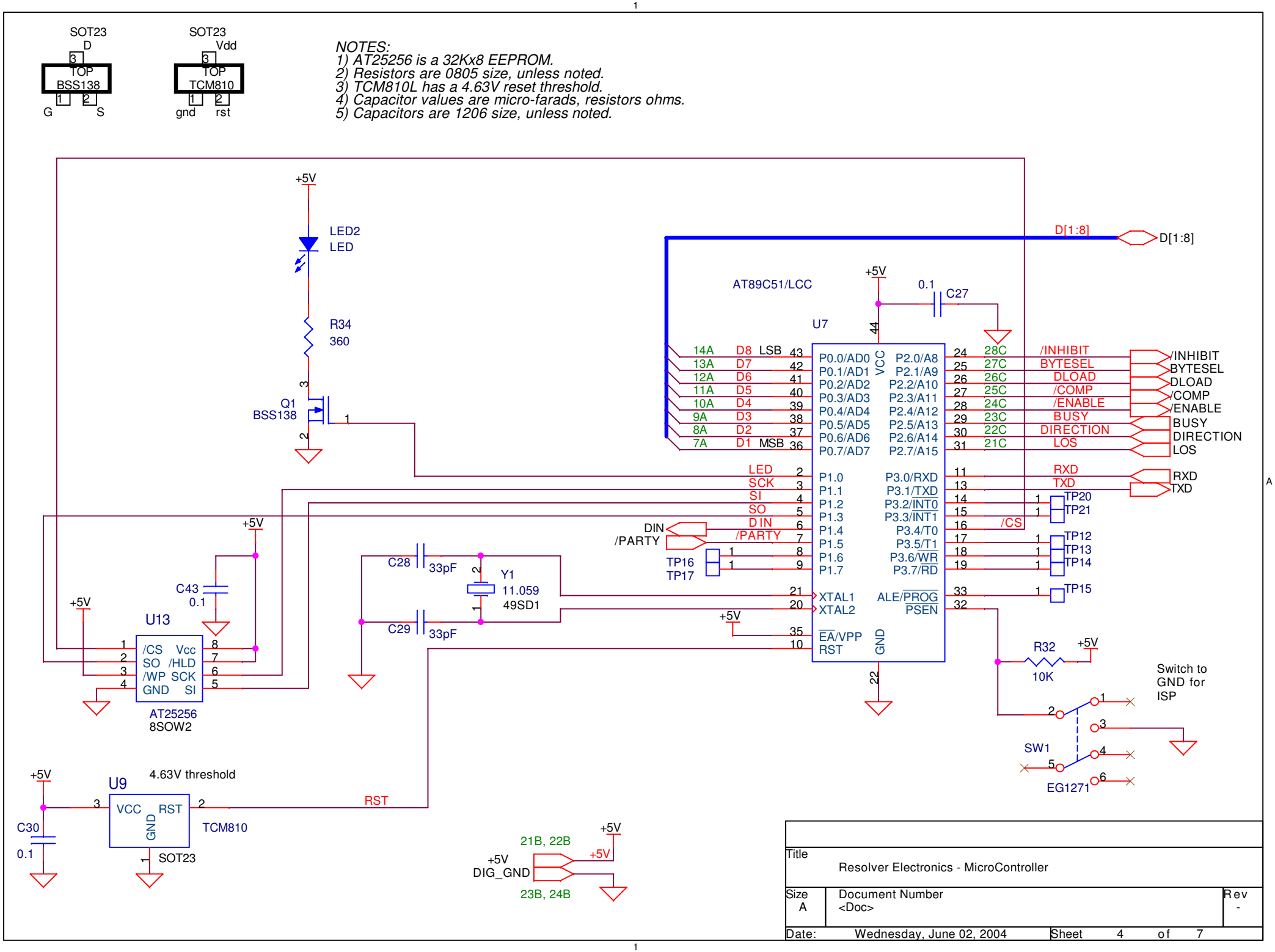

J2
HEADER 6
Power Supply
+12V
-12V
AGND
+5V
-5V
Resolver-To-Digital
/INHIBIT
BYTESEL
DLOAD
/COMP
/ENABLE
BUSY
DIG_GND
SIN
COS
REF
SIG_RTN
RIPPLE_CLK
DIRECTION
D[1:8]
D[9:16]
Oscillator
LOS
/EXEC
EXEC
MicroController
RXD
TXD
DIN
/PARTY
Serial_Interface
Encoder Emulator
ENCODER_A
ENCODER_B
MARKER
J4
Molex
Fem DB15
J1
ORN
WHT
YEL
BLUE
RED
BLK
REF
SIN
COS
J6
HEADER 12X2
Title
Resolver Electronics - Main
Size
A
Document Number
<Doc>
Rev
-
Date:
Thursday, June 03, 2004
Sheet
1
of
7

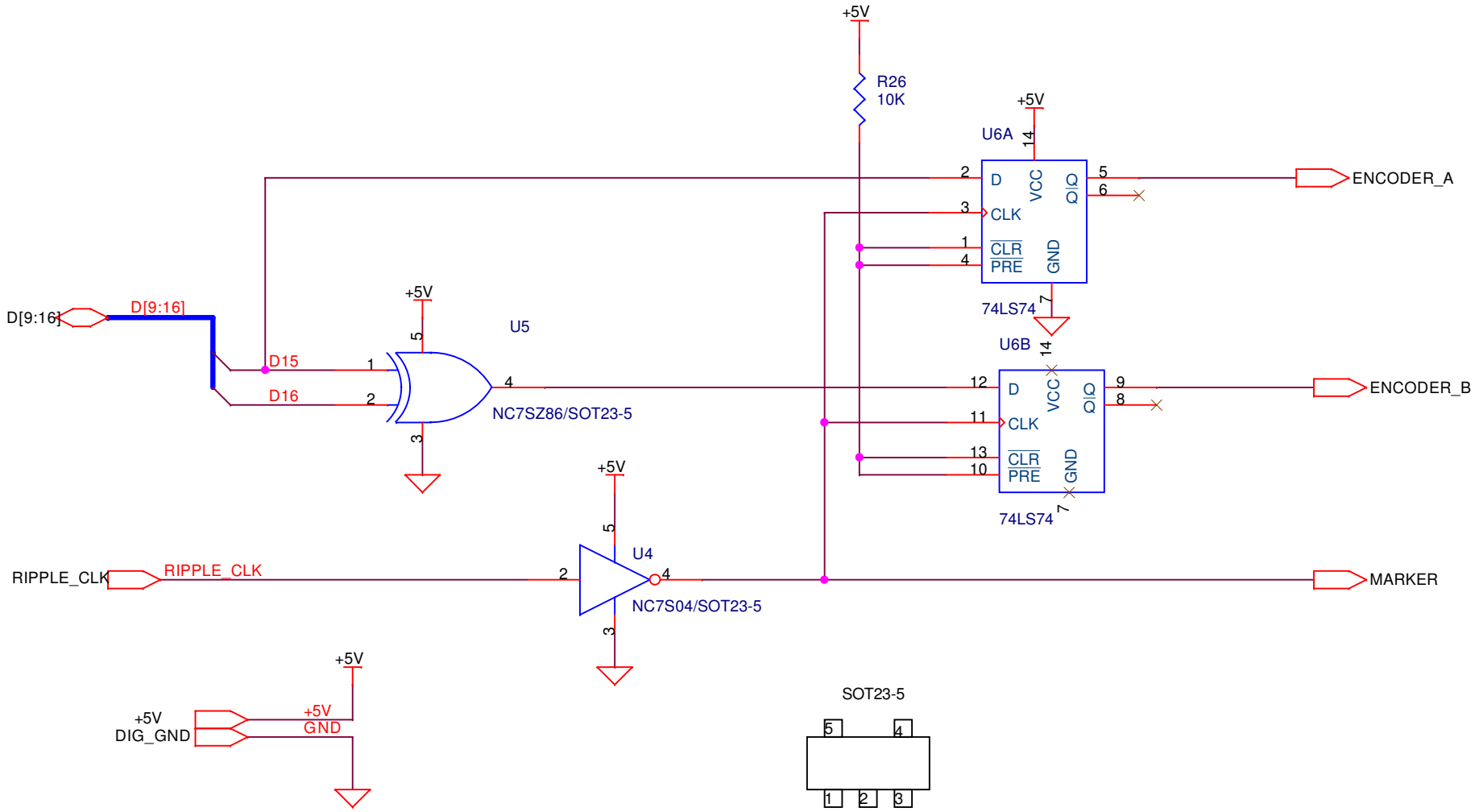


Title	Resolver Electronics - Oscillator Circuit	
Size A	Document Number <Doc>	Rev -
Date: Tuesday, June 01, 2004	Sheet 2 of 7	

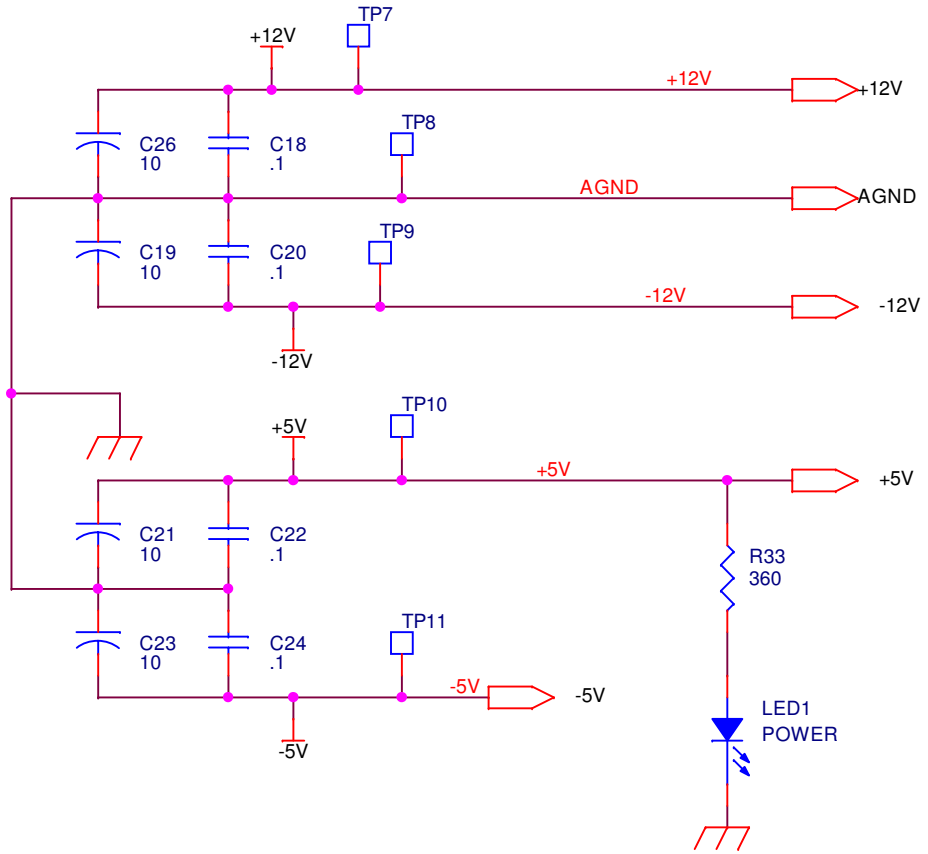


Title	Resolver Electronics - Resolver to Digital Converter	
Size A	Document Number <Doc>	Rev 1
Date: Tuesday, June 01, 2004	Sheet 3 of 7	

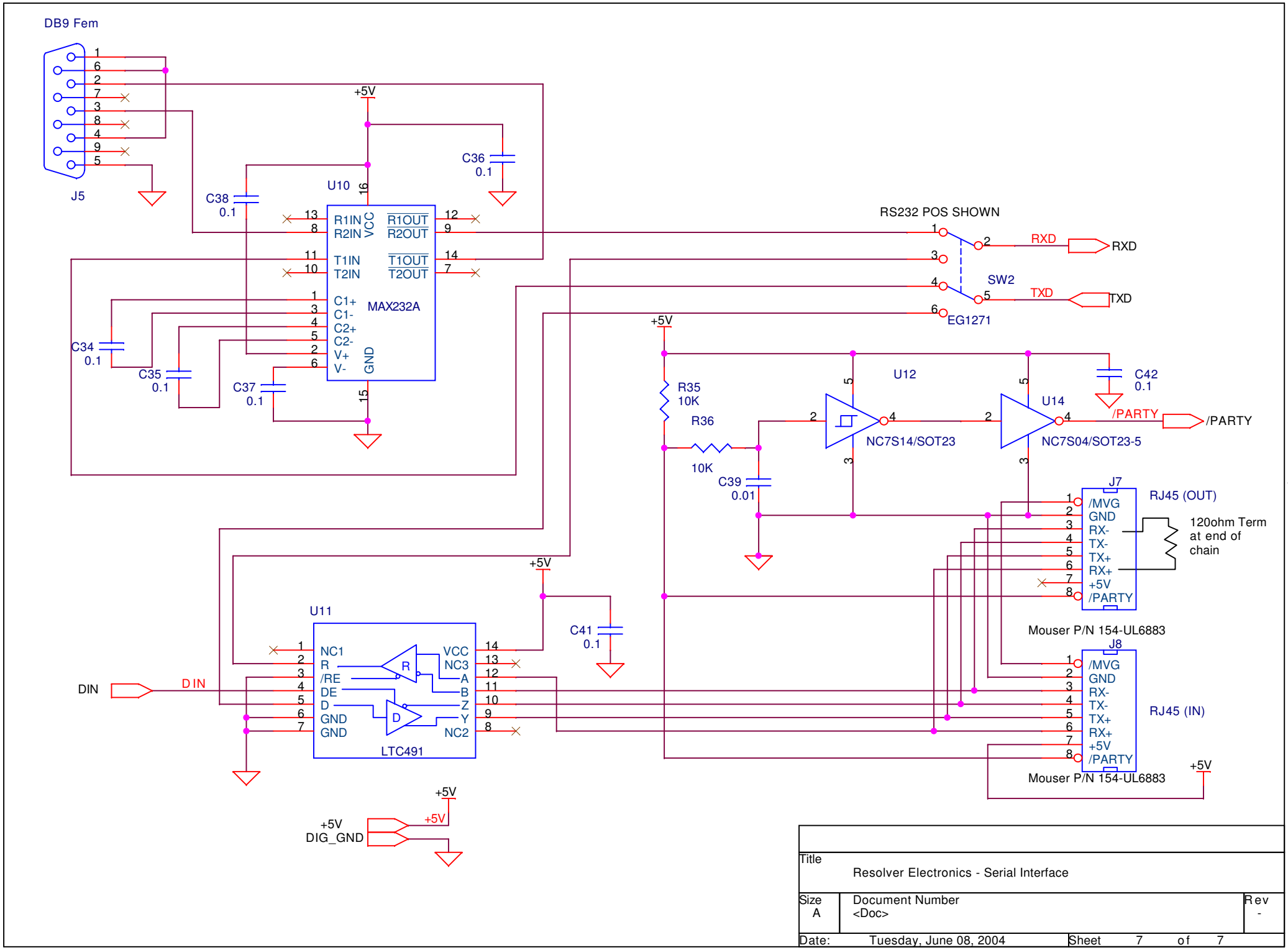

SOT23
D
TOP
BSS138
G
S
SOT23
Vdd
TOP
TCM810
gnd
rst
NOTES:
1) AT25256 is a 32Kx8 EEPROM.
2) Resistors are 0805 size, unless noted.
3) TCM810L has a 4.63V reset threshold.
4) Capacitor values are micro-farads, resistors ohms.
5) Capacitors are 1206 size, unless noted.
+5V
LED2
LED
R34
360
Q1
BSS138
D[1:8]
AT89C51/LCC
U7
C27
0.1
VCC
GND
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
P2.0/A8
P2.1/A9
P2.2/A10
P2.3/A11
P2.4/A12
P2.5/A13
P2.6/A14
P2.7/A15
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
XTAL1
XTAL2
ALE/PROG
PSEN
EA/VPP
RST
14A D8 LSB
13A D7
12A D6
11A D5
10A D4
9A D3
8A D2
7A D1 MSB
28C /INHIBIT
27C BYTESEL
26C DLOAD
25C /COMP
24C /ENABLE
23C BUSY
22C DIRECTION
21C LOS
LED
SCK
SI
SO
DIN
/PARTY
RXD
TXD
/CS
TP20
TP21
TP12
TP13
TP14
TP15
TP16
TP17
DIN
/PARTY
C28
33pF
C29
33pF
Y1
11.059
49SD1
C43
0.1
U13
/CS Vcc
SO /HLD
/WP SCK
GND SI
AT25256
8SOW2
R32
10K
Switch to
GND for
ISP
SW1
EG1271
U9
4.63V threshold
VCC RST
GND
TCM810
SOT23
C30
0.1
RST
21B, 22B
+5V
DIG_GND
23B, 24B
Title
Resolver Electronics - MicroController
Size
A
Document Number
<Doc>
Rev
-
Date: Wednesday, June 02, 2004
Sheet 4 of 7



Title	Resolver Electronics - Encoder Simulator	
Size A	Document Number <Doc>	Rev -
Date: Wednesday, June 02, 2004	Sheet 5 of 7	



Title	Resolver Electronics - Power Supply	
Size A	Document Number <Doc>	Rev -
Date: Tuesday, June 01, 2004	Sheet 6 of 7	


DB9 Fem
J5
+5V
C36
0.1
C38
0.1
U10
MAX232A
R1IN
R2IN
VCC
R1OUT
R2OUT
T1IN
T2IN
T1OUT
T2OUT
C1+
C1-
C2+
C2-
V+
V-
GND
C34
0.1
C35
0.1
C37
0.1
RS232 POS SHOWN
SW2
EG1271
RXD
TXD
R35
10K
R36
10K
C39
0.01
U12
NC7S14/SOT23
U14
NC7S04/SOT23-5
C42
0.1
/PARTY
J7
RJ45 (OUT)
/MVG
GND
RX-
TX-
TX+
RX+
+5V
/PARTY
120ohm Term
at end of
chain
Mouser P/N 154-UL6883
J8
RJ45 (IN)
Mouser P/N 154-UL6883
U11
NC1
R
/RE
DE
D
GND
GND
VCC
NC3
A
B
Z
Y
NC2
LTC491
C41
0.1
DIN
+5V
DIG_GND
Title
Resolver Electronics - Serial Interface
Size
A
Document Number
<Doc>
Rev
-
Date:
Tuesday, June 08, 2004
Sheet
7
of
7