Digital Storage Oscilloscope

Appendix

Thomas Grocutt

April 2000

Contents

8) Appendix

8.1) DSO Control Software

8.1.1) Main.c

```
/* DSO Control Software */
/* Version 1.2          */
/*                      */
/* By Thomas Grocutt    */


/* local define statments */
#define FILE_MAIN


/* local include statments */
#include "main.h"
#include "gui.h"
#include "error.h"
#include "dso_io.h"
#include "p_port_io.h"
#include "file_io.h"
#include "plot_data.h"
#include <stdio.h>
#include <stdlib.h>


/* Global veraibles            */
HINSTANCE instance_handler;
struct prefs_struct prefs;
```

```
/* Initial function, this is the win32 entry point    */
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow )
    {
    /* store hInstance globaly                         */
    instance_handler=hInstance;
    /* set the port address                            */
    p_port_set_address( DEFAULT_P_PORT, 1 );

    /* setup the def scope settings                    */
    clear_dso_data();
    scope_data.scope_mode=0;
    scope_data.pre_trigger=0;
    scope_data.clock_div=0;
    scope_data.trig_sel=0;
    scope_data.trig_value=0;
    scope_data.trig_falling=FALSE;
    /* setup def plot settings                         */
    prefs.zoom_pan_mode=TRUE;
    prefs.x_axis_time=FALSE;
    prefs.show_pre_trig=TRUE;
    prefs.show_trig_level=TRUE;

    /* show the gui */
    show_gui( nCmdShow );
    quit(0);
    }


/* quit program with clean up */
void quit( int ret_code)
    {
    PostQuitMessage( 0 );
    fcloseall();
    exit( ret_code );
    }
```

8.1.2) Main.h

```
/* check if we have included this header files before */
#ifndef FILE_MAIN_HEADER
    /* define FILE_MAIN_HEADER so we dont include this file again */
    #define FILE_MAIN_HEADER

    /* include statments */
    #include <windows.h>

    /* prefs struct define */
    struct prefs_struct
        {
        BOOL zoom_pan_mode;
        BOOL x_axis_time;
        BOOL show_pre_trig;
        BOOL show_trig_level;
        };

    /* Check whether if this file is being used for main.c or another file */
    #ifdef FILE_MAIN
        /* File used for main.c */
        #define FILE_MAIN_EXTERN
    #else
        /* File used for other source file */
        #define FILE_MAIN_EXTERN extern

        /* extern referance to global verables */
        extern HINSTANCE instance_handler;
        extern struct prefs_struct prefs;
    #endif

    /* external and internal function declirations */
    FILE_MAIN_EXTERN void quit( int ret_code);
#endif
```

8.1.3) GUI.c

```
/* local define statments   */
#define FILE_GUI


/* Include Statments        */
#include "gui.h"
#include "main.h"
#include "error.h"
#include "file_io.h"
#include "p_port_io.h"
#include "dso_io.h"
#include "define.h"
#include "plot_data.h"
#include "numeric.h"
#include "resource.h"
#include <stdio.h>


/* setup adn show gui                                */
int show_gui( int nCmdShow )
    {
    MSG msg;
    HACCEL accel_table_handle;

    /* setup and show the window with error check    */
    if( setup_window( nCmdShow ) == NULL )
        return( 1 );

    /* load the keybourd translation table           */
    accel_table_handle=LoadAccelerators( instance_handler, (LPCTSTR)IDK_GUI_KEYS );
```

```
    /* get new messages and process them            */
    while( GetMessage( &msg, NULL, 0, 0 ) )
        {
        if( !TranslateAccelerator( msg.hwnd, accel_table_handle, &msg) )
            DispatchMessage(&msg);
        }

    return(0);
    }


/* setup and show the main window           */
static HWND setup_window( int nCmdShow )
    {
    HWND window_handle;
    WNDCLASSEX window_class;

    /* setup the window settings            */
    window_class.cbSize            = sizeof( WNDCLASSEX );
    window_class.style             = CS_HREDRAW | CS_VREDRAW;
    window_class.lpfnWndProc    = ( WNDPROC ) main_window_message_handler;
    window_class.cbClsExtra         = 0;
    window_class.cbWndExtra         = 0;
    window_class.hInstance         = instance_handler;
    window_class.hIcon             = LoadIcon( instance_handler, (LPCTSTR) IDI_DSO_ICON );
    window_class.hCursor         = LoadCursor( NULL, IDC_ARROW );
    window_class.hbrBackground     = NULL;
    window_class.lpszMenuName     = ( LPCSTR ) IDM_GUI_MENU;
    window_class.lpszClassName     = GUI_CLASS_NAME;
    window_class.hIconSm         = LoadIcon( instance_handler, (LPCTSTR) IDI_DSO_ICON_SMALL );
    /* register the window class            */
    RegisterClassEx( &window_class );
```

```
    /* create the window, with error check  */
    window_handle=CreateWindow
        (
        GUI_CLASS_NAME,
        APP_NAME,
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,
        0,
        CW_USEDEFAULT,
        0,
        NULL,
        NULL,
        instance_handler,
        NULL
        );
    if( !window_handle )
        return( NULL );

    /* show the window and go home          */
    ShowWindow( window_handle, nCmdShow );
    UpdateWindow( window_handle );
    return( window_handle );
    }


/* main window message handler                                          */
static LRESULT CALLBACK main_window_message_handler( HWND window_handle, UINT message, WPARAM wParam, LPARAM
lParam )
    {
    char file_name[ FILE_NAME_LEN ];
    OPENFILENAME file_name_info_struct;
    PAINTSTRUCT paint_struct;

    /* what sort of message have we been sent                       */
    switch (message)
        {
```

```
/* we have been send a command                              */
case WM_COMMAND:
    /* so what command has we been sent                     */
    switch( LOWORD( wParam ) )
        {
        /* file menu commands                               */
        /* open the data file specified by user             */
        case IDM_FILE_OPEN:
            /* init the file struct with basic info         */
            file_name[0] = '\0';
            memset( &file_name_info_struct, 0, sizeof( file_name_info_struct ) );
            file_name_info_struct.lStructSize = sizeof( file_name_info_struct );
            file_name_info_struct.hwndOwner   = window_handle;
            file_name_info_struct.lpstrFile   = file_name;
            file_name_info_struct.lpstrDefExt = "dso";
            file_name_info_struct.nMaxFile    = FILE_NAME_LEN;
            file_name_info_struct.lpstrFilter = "DSO Data Files (*.dso)\0*.dso\0\0";
            file_name_info_struct.Flags       = OFN_FILEMUSTEXIST + OFN_HIDEREADONLY;
            /* display the requesta                         */
            if( GetOpenFileName( &file_name_info_struct ) != 0 )
                {
                /* load data, refresh, with max range       */
                load_dso_structure( file_name );
                goto_max_range();
                InvalidateRect( window_handle, NULL, TRUE );
                }
            break;
        /* save the data file specified by user             */
        case IDM_FILE_SAVE_AS:
            /* init the file struct with basic info         */
            file_name[0] = '\0';
            memset( &file_name_info_struct, 0, sizeof( file_name_info_struct ) );
            file_name_info_struct.lStructSize = sizeof( file_name_info_struct );
            file_name_info_struct.hwndOwner   = window_handle;
            file_name_info_struct.lpstrFile   = file_name;
            file_name_info_struct.lpstrDefExt = "dso";
```

```
    file_name_info_struct.nMaxFile    = FILE_NAME_LEN;
    file_name_info_struct.lpstrFilter = "DSO Data Files (*.dso)\0*.dso\0\0";
    file_name_info_struct.Flags       = OFN_FILEMUSTEXIST + OFN_HIDEREADONLY;
    /* display the requesta                          */
    if( GetSaveFileName( &file_name_info_struct ) != 0 )
        /* save the data to the file specified     */
        save_dso_structure( file_name );
    break;
/* export the data to csv file specified           */
case IDM_FILE_EXPORT:
    /* init the file struct with basic info        */
    file_name[0] = '\0';
    memset( &file_name_info_struct, 0, sizeof( file_name_info_struct ) );
    file_name_info_struct.lStructSize = sizeof( file_name_info_struct );
    file_name_info_struct.hwndOwner   = window_handle;
    file_name_info_struct.lpstrFile   = file_name;
    file_name_info_struct.lpstrDefExt = "csv";
    file_name_info_struct.nMaxFile    = FILE_NAME_LEN;
    file_name_info_struct.lpstrFilter = "CSV File (*.csv)\0*.csv\0\0";
    file_name_info_struct.Flags       = OFN_FILEMUSTEXIST + OFN_HIDEREADONLY;
    /* display the requesta                          */
    if( GetSaveFileName( &file_name_info_struct ) != 0 )
        /* save the data to the file specified     */
        export_csv( file_name );
    break;
/* trigger data capture                            */
case IDM_DATA_CAPTURE:
    DialogBox( instance_handler, (LPCTSTR)IDD_DSO_CAPTURE, window_handle,
                                 (DLGPROC)data_capture_message_handler );
    break;
/* trigger port select                             */
case IDM_SET_PORT:
    DialogBox( instance_handler, (LPCTSTR)IDD_SET_PORT, window_handle,
                                 (DLGPROC)set_port_message_handler );
    break;
/* show the prefs dialog box                       */
```

```
case IDM_PREFS:
    DialogBox( instance_handler, (LPCTSTR)IDD_PREFS, window_handle,
                                 (DLGPROC)prefs_message_handler );
    break;
/* exit program                                     */
case IDM_EXIT:
    PostQuitMessage(0);
    break;

/* view menu commands                               */
/* change the view range, show dialog box           */
case IDM_VIEW_RANGE:
    DialogBox( instance_handler, (LPCTSTR)IDD_VIEW_RANGE, window_handle,
                                 (DLGPROC)view_range_message_handler );
    break;
/* view changes, pan & zoom                         */
case IDM_ZOOM_IN:
    change_view( window_handle, 0 );
    break;
case IDM_ZOOM_OUT:
    change_view( window_handle, 1 );
    break;
case IDM_PAN_RIGHT:
    change_view( window_handle, 2 );
    break;
case IDM_PAN_LEFT:
    change_view( window_handle, 3 );
    break;
/* single channel offset correct                    */
case IDM_OFFSET_CORRECT:
    if( offset_correct() == TRUE )
        error( 15, "main_window_message_handler", "" );
    else
        InvalidateRect( window_handle, NULL, TRUE );
    break;
/* user requested refresh so invalidate window      */
```

```
case IDM_REFRESH:
    InvalidateRect( window_handle, NULL, TRUE );
    break;

/* debug menu items                                 */
/* change the view mode                             */
case IDM_VIEW_MODE:
    DialogBox( instance_handler, (LPCTSTR)IDD_VIEW_MODE, window_handle,
                                 (DLGPROC)view_mode_message_handler );
    break;
/* copys a to b                                     */
case IDM_A_TO_B:
    copy_a_b();
    InvalidateRect( window_handle, NULL, TRUE );
    break;
/* copies channel b to a                            */
case IDM_B_TO_A:
    copy_b_a();
    InvalidateRect( window_handle, NULL, TRUE );
    break;
/* swaps the two channels                           */
case IDM_SWAP_AB:
    swap_a_b();
    InvalidateRect( window_handle, NULL, TRUE );
    break;
/* re read fifo a and update display                */
case IDM_REREAD_A:
    get_fifo_data( scope_data.data.dual_channel.a, TRUE,  FALSE );
    InvalidateRect( window_handle, NULL, TRUE );
    break;
/* re read fifo b and update display                */
case IDM_REREAD_B:
    get_fifo_data( scope_data.data.dual_channel.b, FALSE, FALSE );
    InvalidateRect( window_handle, NULL, TRUE );
    break;
case IDM_HW_IO:
```

```
            DialogBox( instance_handler, (LPCTSTR)IDD_HARDWARE_IO, window_handle,
                                         (DLGPROC)hardware_io_message_handler );
            break;

        /* help menu                                          */
        /* show the about message                             */
        case IDM_ABOUT:
            DialogBox( instance_handler, (LPCTSTR)IDD_ABOUT, window_handle,
                                         (DLGPROC)about_message_handler );
            break;

        /* misc commands                                      */
        case IDD_DISABLE:
            EnableWindow( window_handle, FALSE );
            break;
        /* unknown message so let windows deal with it        */
        default:
            return DefWindowProc( window_handle, message, wParam, lParam);
        }
    break;

/* re-paint request                                           */
case WM_PAINT:
    /* prep paint and get  coordinates of window              */
    BeginPaint( window_handle, &paint_struct );
    /* plot the stored data to the window                     */
    plot_scope_data( window_handle, paint_struct );
    /* end the painting                                       */
    EndPaint( window_handle, &paint_struct );
    /* enable or disable offset correct menu item             */
    if( scope_data.scope_mode == 1 ) EnableMenuItem( GetMenu( window_handle ), IDM_OFFSET_CORRECT,
                                     MF_ENABLED );
    else                             EnableMenuItem( GetMenu( window_handle ), IDM_OFFSET_CORRECT,
                                     MF_GRAYED  );
    break;
```

```
        /* setup the min window size                                */
        case WM_GETMINMAXINFO:
            ((LPMINMAXINFO) lParam)->ptMinTrackSize.x=MIN_WINDOW_X;
            ((LPMINMAXINFO) lParam)->ptMinTrackSize.y=MIN_WINDOW_Y;
            break;

        /* quits the program                                        */
        case WM_DESTROY:
            PostQuitMessage(0);
            break;

        /* unknown message so let windows deal with it              */
        default:
            return DefWindowProc( window_handle, message, wParam, lParam);
        }

    return 0;
    }


/* message handler for view range dialog box        */
static LRESULT CALLBACK view_range_message_handler( HWND window_handle, UINT message, WPARAM wParam,
                                                    LPARAM lParam )
    {
    long i, j;

    /* what sort of message have we been sent       */
    switch( message )
        {
        /* init dialog box with current values      */
        case WM_INITDIALOG:
            /* get the current range                */
            get_range( &i, &j );
            /* display range in edit box            */
            SetDlgItemInt( window_handle, IDD_VIEW_RANGE_START, i, TRUE );
            SetDlgItemInt( window_handle, IDD_VIEW_RANGE_STOP,  j, TRUE );
```

```
        return( 1 );
        break;

    case WM_COMMAND:
        /* what command have we been sent       */
        switch( LOWORD(wParam) )
            {
            case IDOK:
                /* get the new range values     */
                i=GetDlgItemInt( window_handle, IDD_VIEW_RANGE_START, NULL, TRUE );
                j=GetDlgItemInt( window_handle, IDD_VIEW_RANGE_STOP,  NULL, TRUE );
                /* update range, check valid    */
                if( set_range( i, j ) == TRUE )
                    {
                    /* valid so refresh display */
                    PostMessage( GetParent( window_handle ), WM_COMMAND, IDM_REFRESH, 0 );
                    EndDialog( window_handle, LOWORD(wParam) );
                    }
                else
                    /* not valid, tell user     */
                    error( 11, "view_range_message_handler", "" );
                break;

            case IDCANCEL:
                EndDialog( window_handle, LOWORD(wParam) );
                break;
            }
        break;
    }

return( 0 );
}
```

```
/* message handler for view range dialog box        */
static LRESULT CALLBACK view_mode_message_handler( HWND window_handle, UINT message, WPARAM wParam,
                                                   LPARAM lParam )
    {
    /* what sort of message have we been sent       */
    switch( message )
        {
        /* init dialog box with current values      */
        case WM_INITDIALOG:
            /* setup scope mode                     */
            switch( scope_data.scope_mode )
                {
                case 0:
                    CheckRadioButton( window_handle, IDD_DSO_MODE_0, IDD_DSO_MODE_3, IDD_DSO_MODE_0 );
                    break;
                case 1:
                    CheckRadioButton( window_handle, IDD_DSO_MODE_0, IDD_DSO_MODE_3, IDD_DSO_MODE_1 );
                    break;
                case 2:
                    CheckRadioButton( window_handle, IDD_DSO_MODE_0, IDD_DSO_MODE_3, IDD_DSO_MODE_2 );
                    break;
                case 3:
                    CheckRadioButton( window_handle, IDD_DSO_MODE_0, IDD_DSO_MODE_3, IDD_DSO_MODE_3 );
                    break;
                }
            return( 1 );
            break;

        case WM_COMMAND:
            /* what command have we been sent       */
            switch( LOWORD(wParam) )
                {
                case IDOK:
                    /* get scope mode               */
                    if( IsDlgButtonChecked( window_handle, IDD_DSO_MODE_0 ) == BST_CHECKED )
                        scope_data.scope_mode=0;
```

```
                    else if ( IsDlgButtonChecked( window_handle, IDD_DSO_MODE_1 ) == BST_CHECKED )
                        scope_data.scope_mode=1;
                    else if ( IsDlgButtonChecked( window_handle, IDD_DSO_MODE_2 ) == BST_CHECKED )
                        scope_data.scope_mode=2;
                    else
                        scope_data.scope_mode=3;
                    /* finished so refresh display  */
                    PostMessage( GetParent( window_handle ), WM_COMMAND, IDM_REFRESH, 0 );
                    EndDialog( window_handle, LOWORD(wParam) );
                    break;

                case IDCANCEL:
                    EndDialog( window_handle, LOWORD(wParam) );
                    break;
                }
            break;
        }

    return( 0 );
    }


/* message handler for view range dialog box        */
static LRESULT CALLBACK about_message_handler( HWND window_handle, UINT message, WPARAM wParam, LPARAM lParam )
    {
    /* what sort of message have we been sent       */
    switch( message )
        {
        /* init dialog box with current values      */
        case WM_INITDIALOG:
            /* set app name and email address       */
            SetDlgItemText( window_handle, IDD_APP_NAME,      APP_NAME            );
            SetDlgItemText( window_handle, IDD_APP_VERSION,   APP_VERSION         );
            SetDlgItemText( window_handle, IDD_EMAIL_ADDRESS, AUTHOR_MAIL_ADDRESS );
            return( 1 );
            break;
```

```
        case WM_COMMAND:
            /* what command have we been sent       */
            switch( LOWORD(wParam) )
                {
                case IDOK:
                case IDCANCEL:
                    EndDialog( window_handle, LOWORD(wParam) );
                    break;
                }
            break;
        }

    return( 0 );
    }


/* message handler for view range dialog box        */
static LRESULT CALLBACK set_port_message_handler( HWND window_handle, UINT message, WPARAM wParam,
                                                  LPARAM lParam )
    {
    /* what sort of message have we been sent       */
    switch( message )
        {
        /* init dialog box with current values      */
        case WM_INITDIALOG:
            /* put the data in the combo box        */
            SendDlgItemMessage( window_handle, IDD_PORT_ADDRESS, CB_ADDSTRING, 0, (LPARAM) "LPT1: (0x378)" );
            SendDlgItemMessage( window_handle, IDD_PORT_ADDRESS, CB_ADDSTRING, 0, (LPARAM) "LPT2: (0x278)" );
            SendDlgItemMessage( window_handle, IDD_PORT_ADDRESS, CB_ADDSTRING, 0, (LPARAM) "LPT3: (0x3BC)" );
            /* select the current port              */
            switch( p_port_get_address() )
                {
                case 0x378:
                    SendDlgItemMessage( window_handle, IDD_PORT_ADDRESS, CB_SETCURSEL, 0, 0 );
                    break;
```

```
        case 0x278:
            SendDlgItemMessage( window_handle, IDD_PORT_ADDRESS, CB_SETCURSEL, 1, 0 );
            break;
        case 0x3BC:
            SendDlgItemMessage( window_handle, IDD_PORT_ADDRESS, CB_SETCURSEL, 2, 0 );
            break;
        }
    return( 1 );
    break;

case WM_COMMAND:
    switch( LOWORD(wParam) )
        {
        case IDOK:
            /* find out what new port is    */
            switch( SendDlgItemMessage( window_handle, IDD_PORT_ADDRESS, CB_GETCURSEL, 0, 0 ) )
                {
                case 0:
                    p_port_set_address( 0x378, 1 );
                    break;
                case 1:
                    p_port_set_address( 0x278, 1 );
                    break;
                case 2:
                    p_port_set_address( 0x3BC, 1 );
                    break;
                }
            EndDialog( window_handle, LOWORD(wParam) );
            break;
        case IDCANCEL:
            EndDialog( window_handle, LOWORD(wParam) );
            break;
        }
    break;
}
```

```
    return( 0 );
    }


/* message handler data capture dialog box          */
static LRESULT CALLBACK data_capture_message_handler( HWND window_handle, UINT message, WPARAM wParam,
                                                      LPARAM lParam )
    {
    long l;
    int i, j;
    HANDLE rise_image_handle=NULL, fall_image_handle=NULL;
    char c, string_buf[50];

    /* what sort of message have we been sent       */
    switch( message )
        {
        /* init dialog box with current values      */
        case WM_INITDIALOG:
            /* setup the images for rise and fall   */
            rise_image_handle=LoadImage( instance_handler, (LPCTSTR) IDB_RISE, IMAGE_BITMAP, 0, 0,
                                         LR_MONOCHROME );
            fall_image_handle=LoadImage( instance_handler, (LPCTSTR) IDB_FALL, IMAGE_BITMAP, 0, 0,
                                         LR_MONOCHROME );
            SendDlgItemMessage( window_handle, IDD_TRIG_SLOPE_RISE, BM_SETIMAGE, IMAGE_BITMAP,
                                (LPARAM) rise_image_handle );
            SendDlgItemMessage( window_handle, IDD_TRIG_SLOPE_FALL, BM_SETIMAGE, IMAGE_BITMAP,
                                (LPARAM) fall_image_handle );

            /* setup pre trigger value              */
            SetDlgItemInt( window_handle, IDD_PRE_TRIG, scope_data.pre_trigger, FALSE );
            /* setup scope mode                     */
            switch( scope_data.scope_mode )
                {
                case 0: SendDlgItemMessage( window_handle, IDD_DSO_MODE_0, BM_CLICK, 0, 0 ); break;
                case 1:    SendDlgItemMessage( window_handle, IDD_DSO_MODE_1, BM_CLICK, 0, 0 ); break;
                case 2:    SendDlgItemMessage( window_handle, IDD_DSO_MODE_2, BM_CLICK, 0, 0 ); break;
```

```
    case 3: SendDlgItemMessage( window_handle, IDD_DSO_MODE_3, BM_CLICK, 0, 0 ); break;
    }
/* setup trig source                    */
switch( scope_data.trig_sel )
    {
    case 0: SendDlgItemMessage( window_handle, IDD_TRIG_SOURCE_0, BM_CLICK, 0, 0 );    break;
    case 1:    SendDlgItemMessage( window_handle, IDD_TRIG_SOURCE_1, BM_CLICK, 0, 0 );    break;
    case 2:    SendDlgItemMessage( window_handle, IDD_TRIG_SOURCE_2, BM_CLICK, 0, 0 );    break;
    case 3:    SendDlgItemMessage( window_handle, IDD_TRIG_SOURCE_3, BM_CLICK, 0, 0 );    break;
    }
/* setup scope sample frequency         */
/* fill combo box with dummy values     */
for( i=0 ; i<6 ; i++ )
    SendDlgItemMessage( window_handle, IDD_CLOCK_FREQ, CB_ADDSTRING, 0, (LPARAM) "DUMMY STRINF" );
SendDlgItemMessage( window_handle, IDD_CLOCK_FREQ, CB_SETCURSEL, scope_data.clock_div, 0 );
/* set the trig slope                   */
if( scope_data.trig_falling == FALSE )
    SendDlgItemMessage( window_handle, IDD_TRIG_SLOPE_RISE, BM_CLICK, 0, 0 );
else
    SendDlgItemMessage( window_handle, IDD_TRIG_SLOPE_FALL, BM_CLICK, 0, 0 );
/* setup the trig value                 */
if( scope_data.scope_mode == 2 || ( scope_data.scope_mode == 3 && scope_data.trig_sel == 1 ) )
    {
    /* num is hex                       */
    sprintf( string_buf, "0x%x", scope_data.trig_value );
    SetDlgItemText( window_handle, IDD_TRIG_VALUE, string_buf );
    }
else
    {
    /* num is dec                       */
    i=scope_data.trig_value - (FIFO_RANGE/2) + 1;
    SetDlgItemInt( window_handle, IDD_TRIG_VALUE, i, TRUE );
    }
return( 1 );
break;
```

```
/* we have been sent a command             */
case WM_COMMAND:
    /* which command were we sent          */
    switch( LOWORD(wParam) )
        {
        /* is this a notification message   */
        case IDD_DSO_MODE_0:
        case IDD_DSO_MODE_1:
        case IDD_DSO_MODE_2:
        case IDD_DSO_MODE_3:
        case IDD_TRIG_SOURCE_0:
        case IDD_TRIG_SOURCE_1:
        case IDD_TRIG_SOURCE_2:
        case IDD_TRIG_SOURCE_3:
            if( HIWORD(wParam) == BN_CLICKED )
                PostMessage( window_handle, WM_COMMAND, IDD_DSO_SETUP_VALIDATE, 0 );
            break;
        case IDD_PRE_TRIG:
        case IDD_TRIG_VALUE:
            if( HIWORD(wParam) == EN_KILLFOCUS  )
                PostMessage( window_handle, WM_COMMAND, IDD_DSO_SETUP_VALIDATE, 0 );
            break;

        /* validate the new settings        */
        case IDD_DSO_SETUP_VALIDATE:
            /* see what mode we are in      */
            if(      IsDlgButtonChecked( window_handle, IDD_DSO_MODE_0 ) == BST_CHECKED )    c=0;
            else if( IsDlgButtonChecked( window_handle, IDD_DSO_MODE_1 ) == BST_CHECKED )    c=1;
            else if( IsDlgButtonChecked( window_handle, IDD_DSO_MODE_2 ) == BST_CHECKED )    c=2;
            else                                                                            c=3;

            /* validate the sample freq     */
            /* get the current value        */
            l=SendDlgItemMessage( window_handle, IDD_CLOCK_FREQ, CB_GETCURSEL, 0, 0 );
            /* check which frequency range  */
            if( c == 1 )    j=DSO_CLOCK_FREQ * 2;
```

```
else            j=DSO_CLOCK_FREQ;
/* enter freg range             */
SendDlgItemMessage( window_handle, IDD_CLOCK_FREQ, CB_RESETCONTENT , 0, 0 );
for( i=1 ; i<=32 ; i=i+i )
    {
    sprintf( string_buf, "%.1f MHz", j / (float) i );
    SendDlgItemMessage( window_handle, IDD_CLOCK_FREQ, CB_ADDSTRING,0,(LPARAM)string_buf );
    }
/* select old frequency         */
SendDlgItemMessage( window_handle, IDD_CLOCK_FREQ, CB_SETCURSEL, l, 0 );

/* validate the trig source     */
switch( c )
    {
    case 0:
        SetDlgItemText( window_handle, IDD_TRIG_SOURCE_1, "Analogue Channel A" );
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_SOURCE_2 ), TRUE );
        break;
    case 1:
        /* are we disable sel trig    */
        if( IsDlgButtonChecked( window_handle, IDD_TRIG_SOURCE_2 ) == BST_CHECKED )
            SendDlgItemMessage( window_handle, IDD_TRIG_SOURCE_1, BM_CLICK, 0, 0 );
        SetDlgItemText( window_handle, IDD_TRIG_SOURCE_1, "Analogue Channel A" );
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_SOURCE_2 ), FALSE );
        break;
    case 2:
        /* are we disable sel trig    */
        if( IsDlgButtonChecked( window_handle, IDD_TRIG_SOURCE_2 ) == BST_CHECKED )
            SendDlgItemMessage( window_handle, IDD_TRIG_SOURCE_1, BM_CLICK, 0, 0 );
        SetDlgItemText( window_handle, IDD_TRIG_SOURCE_1, "16 Bit Logic" );
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_SOURCE_2 ), FALSE );
        break;
    case 3:
        SetDlgItemText( window_handle, IDD_TRIG_SOURCE_1, "Logic Channel A" );
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_SOURCE_2 ), TRUE );
        break;
```

```
    }
/* get trig source                    */
if(      IsDlgButtonChecked( window_handle, IDD_TRIG_SOURCE_0 ) == BST_CHECKED )    i=0;
else if( IsDlgButtonChecked( window_handle, IDD_TRIG_SOURCE_1 ) == BST_CHECKED )    i=1;
else if( IsDlgButtonChecked( window_handle, IDD_TRIG_SOURCE_2 ) == BST_CHECKED )    i=2;
else                                                                                i=3;

/* validate trig value and slope      */
switch( i )
    {
    /* ext trig                       */
    case 0:
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_VALUE           ), FALSE );
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_VALUE_MESSAGE ), FALSE );
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_SLOPE_RISE    ), TRUE  );
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_SLOPE_FALL    ), TRUE  );
        break;
    /* no trig                        */
    case 3:
        SetDlgItemInt( window_handle, IDD_PRE_TRIG, 0, FALSE );
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_VALUE           ), FALSE );
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_VALUE_MESSAGE ), FALSE );
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_SLOPE_RISE    ), FALSE );
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_SLOPE_FALL    ), FALSE );
        break;
    /* channel b therefore anal       */
    case 2:
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_VALUE           ), TRUE );
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_VALUE_MESSAGE ), TRUE );
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_SLOPE_RISE    ), TRUE );
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_SLOPE_FALL    ), TRUE );
        SetDlgItemText( window_handle, IDD_TRIG_VALUE_MESSAGE, "Trigger Value (Dec)" );
        break;
    /* chan a therefore check mode    */
    case 1:
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_VALUE           ), TRUE );
```

```
        EnableWindow( GetDlgItem( window_handle, IDD_TRIG_VALUE_MESSAGE ), TRUE );
        if( c == 2 || c == 3 )
            {
            /* dig mode                */
            SetDlgItemText( window_handle, IDD_TRIG_VALUE_MESSAGE, "Trigger Value (Hex)" );
            EnableWindow( GetDlgItem( window_handle, IDD_TRIG_SLOPE_RISE    ), FALSE );
            EnableWindow( GetDlgItem( window_handle, IDD_TRIG_SLOPE_FALL    ), FALSE );
            }
        else
            {
            /* anal mode               */
            SetDlgItemText( window_handle, IDD_TRIG_VALUE_MESSAGE, "Trigger Value (Dec)" );
            EnableWindow( GetDlgItem( window_handle, IDD_TRIG_SLOPE_RISE    ), TRUE );
            EnableWindow( GetDlgItem( window_handle, IDD_TRIG_SLOPE_FALL    ), TRUE );
            }
        break;
    }
/* now check trig values it's self    */
if( i == 1 && c >= 2 )
    {
    /* digital check                   */
    GetDlgItemText( window_handle, IDD_TRIG_VALUE, string_buf, 50 );
    sscanf(  string_buf, "%x",  &j );
    if( c == 2 ) j=j & 0xffff;
    else         j=j & 0xff;
    sprintf( string_buf, "0x%x", j );
    SetDlgItemText( window_handle, IDD_TRIG_VALUE, string_buf );
    }
if( c <= 1 || ( c == 3 && i == 2 ) )
    {
    /* analog check                    */
    j=GetDlgItemInt( window_handle, IDD_TRIG_VALUE, NULL, TRUE );
    if( j > ( FIFO_RANGE /  2 ) - 1 ) j=( FIFO_RANGE /  2 ) - 1;
    if( j < ( FIFO_RANGE / -2 ) + 2 ) j=( FIFO_RANGE / -2 ) + 2;
    SetDlgItemInt(   window_handle, IDD_TRIG_VALUE, j,    TRUE );
    }
```

```
    /* validate the pre_trig value        */
    if( i == 3 ) EnableWindow( GetDlgItem( window_handle, IDD_PRE_TRIG ), FALSE );
    else         EnableWindow( GetDlgItem( window_handle, IDD_PRE_TRIG ), TRUE  );
    l=GetDlgItemInt( window_handle, IDD_PRE_TRIG,  NULL, FALSE );
    if( l > MAX_PRE_TRIG ) l=MAX_PRE_TRIG;
    SetDlgItemInt( window_handle, IDD_PRE_TRIG, l, FALSE );
    break;

/* pressed ok buttom,  start dso    */
case IDOK:
    /* make sure data is valid      */
    PostMessage( window_handle, WM_COMMAND, IDD_DSO_SETUP_VALIDATE, 0 );
    /* get pre trig                 */
    scope_data.pre_trigger=GetDlgItemInt( window_handle, IDD_PRE_TRIG,  NULL, FALSE );
    /* get scope mode               */
    if( IsDlgButtonChecked(      window_handle, IDD_DSO_MODE_0 ) == BST_CHECKED )
        scope_data.scope_mode=0;
    else if( IsDlgButtonChecked( window_handle, IDD_DSO_MODE_1 ) == BST_CHECKED )
        scope_data.scope_mode=1;
    else if( IsDlgButtonChecked( window_handle, IDD_DSO_MODE_2 ) == BST_CHECKED )
        scope_data.scope_mode=2;
    else
        scope_data.scope_mode=3;
    /* get trig source              */
    if( IsDlgButtonChecked(      window_handle, IDD_TRIG_SOURCE_0 ) == BST_CHECKED )
        scope_data.trig_sel=0;
    else if( IsDlgButtonChecked( window_handle, IDD_TRIG_SOURCE_1 ) == BST_CHECKED )
        scope_data.trig_sel=1;
    else if( IsDlgButtonChecked( window_handle, IDD_TRIG_SOURCE_2 ) == BST_CHECKED )
        scope_data.trig_sel=2;
    else
        scope_data.trig_sel=3;
    /* get the dso clock div        */
    scope_data.clock_div=(char) SendDlgItemMessage( window_handle, IDD_CLOCK_FREQ,
                                                    CB_GETCURSEL, 0, 0 );
```

```
    /* get the trig slop            */
    if( IsDlgButtonChecked( window_handle, IDD_TRIG_SLOPE_RISE ) == BST_CHECKED )
        scope_data.trig_falling=FALSE;
    else
        scope_data.trig_falling=TRUE;
    /* get the trig value           */
    GetDlgItemText( window_handle, IDD_TRIG_VALUE_MESSAGE, string_buf, 50 );
    /* check if num is hex          */
    if( strcmp( string_buf, "Trigger Value (Hex)" ) == 0 )
        {
        /* num is in hex            */
        GetDlgItemText( window_handle, IDD_TRIG_VALUE, string_buf, 50 );
        sscanf(  string_buf, "%x",  &scope_data.trig_value );
        }
    else
        {
        /* num is dec               */
        scope_data.trig_value=GetDlgItemInt( window_handle, IDD_TRIG_VALUE, NULL, TRUE );
        scope_data.trig_value=scope_data.trig_value + (FIFO_RANGE/2) - 1;
        }
    /* show the capture dialog box  */
    CreateDialog( instance_handler, (LPCTSTR)IDD_PROGRESS, GetParent( window_handle ),
                  progress_message_handler );
    /* now close the current dialog */
    DeleteObject( rise_image_handle );
    DeleteObject( fall_image_handle );
    EndDialog( window_handle, LOWORD(wParam) );
    break;

/* user canceled so cleanup         */
case IDCANCEL:
    DeleteObject( rise_image_handle );
    DeleteObject( fall_image_handle );
    EndDialog( window_handle, LOWORD(wParam) );
    break;
}
```

```
            break;
        }

    return(0);
    }


/* progress indicator message handler               */
static LRESULT CALLBACK progress_message_handler( HWND window_handle, UINT message, WPARAM wParam, LPARAM
lParam )
    {
    char string_buf[40];
    HANDLE capture_thread_handle;
    DWORD  capture_thread_id;

    /* what sort of message have we been sent       */
    switch( message )
        {
        /* init dialog box with current values      */
        case WM_INITDIALOG:
            /* send disable message to main window  */
            PostMessage( GetParent( window_handle ), WM_COMMAND, IDD_DISABLE, 0 );
            /* start the capture                    */
            capture_thread_handle=CreateThread( NULL, 0, capture_thread_func, window_handle, 0,
                                                &capture_thread_id );
            if( capture_thread_handle == NULL )
                error( 12, "progress_message_handler", "capture_thread_func" );
            return( 1 );
            break;

        case WM_COMMAND:
            switch( LOWORD(wParam) )
                {
                /* set text to DSO download string  */
                case IDD_DSO_DOWNLOAD:
                    EnableWindow( GetDlgItem( window_handle, IDCANCEL ), FALSE );
```

```
                SetDlgItemText( window_handle, IDD_PROGRESS_TEXT, "Downloading Captured Data" );
                break;

            /* update the progress % on dlg box */
            case IDD_DSO_PROGRESS:
                sprintf( string_buf, "%d %%", lParam );
                SetDlgItemText( window_handle, IDD_PROGRESS_PERSENTAGE, string_buf );
                break;

            case IDOK:
                /* refresh the display and exit */
                goto_max_range();
                EnableWindow( GetParent( window_handle ), TRUE );
                PostMessage(  GetParent( window_handle ), WM_COMMAND, IDM_REFRESH, 0 );
                DestroyWindow( window_handle );
                break;

            case IDCANCEL:
                /* kill the capture thread      */
                cancel_capture();
                /* kill dialog box and exit     */
                clear_dso_data();
                EnableWindow( GetParent( window_handle ), TRUE );
                PostMessage(  GetParent( window_handle ), WM_COMMAND, IDM_REFRESH, 0 );
                DestroyWindow( window_handle );
                break;
            }
        break;
    }

return( 0 );
}
```

```
/* message handler for hardware io dialog box       */
static LRESULT CALLBACK hardware_io_message_handler( HWND window_handle, UINT message, WPARAM wParam,
                                                     LPARAM lParam )
    {
    char string_buf[20];
    unsigned char reg_num, reg_value;

    /* what sort of message have we been sent       */
    switch( message )
        {
        /* init dialog box with current reg values  */
        case WM_INITDIALOG:
            /* post refresh message, update display */
            PostMessage( window_handle, WM_COMMAND, IDD_REFRESH_DATA, 0 );
            return( 1 );
            break;

        /* we have been send a general command      */
        case WM_COMMAND:
            /* what command have we been sent       */
            switch( LOWORD(wParam) )
                {
                /* notify message, data edit box    */
                case IDD_DATA:
                    if( HIWORD(wParam) == EN_KILLFOCUS )
                        {
                        GetDlgItemText( window_handle, IDD_DATA, string_buf, 20 );
                        sscanf( string_buf, "%x", &reg_value );
                        p_port_put_data( reg_value );
                        PostMessage( window_handle, WM_COMMAND, IDD_REFRESH_DATA, 0 );
                        }
                    break;

                /* notify message, control edit box */
                case IDD_CONTROL:
                    if( HIWORD(wParam) == EN_KILLFOCUS )
```

```
        {
        GetDlgItemText( window_handle, IDD_CONTROL, string_buf, 20 );
        sscanf( string_buf, "%x", &reg_value );
        p_port_put_control( reg_value );
        PostMessage( window_handle, WM_COMMAND, IDD_REFRESH_DATA, 0 );
        }
    break;

/* update the displayed values      */
case IDD_REFRESH_DATA:
    /* update data values           */
    sprintf( string_buf, "0x%x", p_port_get_data() );
    SetDlgItemText( window_handle, IDD_DATA,    string_buf );
    /* update status values         */
    sprintf( string_buf, "0x%x", p_port_get_status() );
    SetDlgItemText( window_handle, IDD_STATUS,  string_buf );
    /* update control values        */
    sprintf( string_buf, "0x%x", p_port_get_control() );
    SetDlgItemText( window_handle, IDD_CONTROL, string_buf );
    break;

/* set altera reg to speficed value */
case IDD_SET_REG:
    /* get and update the reg num   */
    GetDlgItemText( window_handle, IDD_REG_NUM,   string_buf, 20 );
    sscanf(  string_buf, "%x",  &reg_num );
    sprintf( string_buf, "0x%x", reg_num );
    SetDlgItemText( window_handle, IDD_REG_NUM,   string_buf     );
    /* get and update reg value     */
    GetDlgItemText( window_handle, IDD_REG_VALUE, string_buf, 20 );
    sscanf(  string_buf, "%x",  &reg_value );
    sprintf( string_buf, "0x%x", reg_value );
    SetDlgItemText( window_handle, IDD_REG_VALUE, string_buf     );
    /* set register to the value    */
    set_reg( reg_num, reg_value );
    /* now do reg refresh           */
```

```
                PostMessage( window_handle, WM_COMMAND, IDD_REFRESH_DATA, 0 );
                break;

            /* set's the internal control reg   */

            case IDD_CTRL_SET_REG:
                GetDlgItemText( window_handle, IDD_INT_CTRL_REG,   string_buf, 20 );
                sscanf(  string_buf, "%x",  &reg_value );
                sprintf( string_buf, "0x%x", reg_value );
                SetDlgItemText( window_handle, IDD_INT_CTRL_REG,   string_buf     );
                /* set register to the value    */
                set_dso_control( reg_value );
                /* now do reg refresh           */
                PostMessage( window_handle, WM_COMMAND, IDD_REFRESH_DATA, 0 );
                break;

            /* reset the read pointer on fifo   */
            case IDD_RESET_RD:
                dso_reset_rd();
                PostMessage( window_handle, WM_COMMAND, IDD_REFRESH_DATA, 0 );
                break;

            /* close dialog box and return      */
            case IDCANCEL:
                EndDialog( window_handle, LOWORD(wParam) );
                break;
            }
        break;
    }

return( 0 );
}
```

```
/* message handler for hardware io dialog box       */
static LRESULT CALLBACK prefs_message_handler( HWND window_handle, UINT message, WPARAM wParam, LPARAM lParam )
    {
    /* what sort of message have we been sent       */
    switch( message )
        {
        /* init dialog box with current values      */
        case WM_INITDIALOG:
            if( prefs.zoom_pan_mode ) SendDlgItemMessage( window_handle, IDD_ZOOM_PAN, BM_CLICK, 0, 0 );
            if( prefs.x_axis_time )   SendDlgItemMessage( window_handle, IDD_X_AXIS_TIME, BM_CLICK, 0, 0 );
            if( prefs.show_pre_trig ) SendDlgItemMessage( window_handle, IDD_SHOW_PRE_TRIG, BM_CLICK, 0, 0 );
            if( prefs.show_trig_level ) SendDlgItemMessage( window_handle, IDD_SHOW_TRIG_LEVEL, BM_CLICK,0, 0);
            return( 1 );
            break;

        case WM_COMMAND:
            /* what command have we been sent       */
            switch( LOWORD(wParam) )
                {
                /* clicked ok, change the settings  */
                case IDOK:
                    /* read in the zoom flag        */
                    if( IsDlgButtonChecked( window_handle, IDD_ZOOM_PAN ) == BST_CHECKED )
                        prefs.zoom_pan_mode=TRUE;
                    else
                        prefs.zoom_pan_mode=FALSE;
                    /* read in the x axis time flaf */
                    if( IsDlgButtonChecked( window_handle, IDD_X_AXIS_TIME ) == BST_CHECKED )
                        prefs.x_axis_time=TRUE;
                    else
                        prefs.x_axis_time=FALSE;
                    /* read in the pre-trig falg    */
                    if( IsDlgButtonChecked( window_handle, IDD_SHOW_PRE_TRIG ) == BST_CHECKED )
                        prefs.show_pre_trig=TRUE;
                    else
                        prefs.show_pre_trig=FALSE;
```

```
                /* read in the trig level flag  */
                if( IsDlgButtonChecked( window_handle, IDD_SHOW_TRIG_LEVEL ) == BST_CHECKED )
                    prefs.show_trig_level=TRUE;
                else
                    prefs.show_trig_level=FALSE;
                /* refresh main window, go home */
                PostMessage(  GetParent( window_handle ), WM_COMMAND, IDM_REFRESH, 0 );
                EndDialog( window_handle, LOWORD(wParam) );
                break;

            /* cancel and go home               */
            case IDCANCEL:
                EndDialog( window_handle, LOWORD(wParam) );
                break;
            }
        break;
    }

return( 0 );
}
```

8.1.4) Gui.h

```
/* check if we have included this header files before */
#ifndef FILE_GUI_HEADER
    /* define FILE_GUI_HEADER so we dont include this file again */
    #define FILE_GUI_HEADER

    /* include file statments */
    #include <windows.h>

    /* user defined events         */
    #define IDD_DISABLE                   WM_USER + 1
    #define IDD_DSO_DOWNLOAD            WM_USER + 2
    #define IDD_DSO_PROGRESS            WM_USER + 3
    #define IDD_DSO_SETUP_VALIDATE        WM_USER + 4

    /* Check whether if this file is being used for gui.c or another file */
    #ifdef FILE_GUI
        /* File used for gui.c */
        #define FILE_GUI_EXTERN

        /* internal define statments */
        #define MIN_WINDOW_X 300
        #define MIN_WINDOW_Y 400

        /* internal func declirations */
        static HWND setup_window( int nCmdShow );
        static LRESULT CALLBACK main_window_message_handler( HWND window_handle, UINT message, WPARAM wParam,
                                                             LPARAM lParam );
        static LRESULT CALLBACK view_range_message_handler( HWND window_handle, UINT message, WPARAM wParam,
                                                            LPARAM lParam );
        static LRESULT CALLBACK view_mode_message_handler( HWND window_handle, UINT message, WPARAM wParam,
                                                           LPARAM lParam );
        static LRESULT CALLBACK about_message_handler( HWND window_handle, UINT message, WPARAM wParam,
                                                       LPARAM lParam );
```

```
        static LRESULT CALLBACK set_port_message_handler( HWND window_handle, UINT message, WPARAM wParam,
                                                          LPARAM lParam );
        static LRESULT CALLBACK data_capture_message_handler( HWND window_handle, UINT message, WPARAM wParam,
                                                              LPARAM lParam );
        static LRESULT CALLBACK progress_message_handler( HWND window_handle, UINT message, WPARAM wParam,
                                                          LPARAM lParam );
        static LRESULT CALLBACK hardware_io_message_handler( HWND window_handle, UINT message, WPARAM wParam,
                                                             LPARAM lParam );
        static LRESULT CALLBACK prefs_message_handler( HWND window_handle, UINT message, WPARAM wParam,
                                                       LPARAM lParam );
    #else
        /* File used for other source file */
        #define FILE_GUI_EXTERN extern
    #endif

    /* internal and external func definitions */
    FILE_GUI_EXTERN int show_gui( int nCmdShow );
#endif
```

8.1.5) Plot_Data.c

```
/* local define statments */
#define FILE_PLOT_DATA

/* local include statments */
#include "plot_data.h"
#include "dso_io.h"
#include "define.h"
#include "main.h"
#include "numeric.h"
#include <stdio.h>
#include <string.h>
#include <windows.h>


/* file level veriable decliration */
static long trace_start=0, trace_stop=FIFO_SIZE;
static PAINTSTRUCT paint_struct;


/* function used to set the trace start and stop values  */
BOOL set_range( long new_trace_start, long new_trace_stop )
    {
    /* check the new values are valid                    */
    if( new_trace_start < 0 ) new_trace_start = 0;
    if( new_trace_stop  < 0 ) new_trace_stop  = 0;
    if( scope_data.scope_mode == 1 )
        {
        if( new_trace_stop >= ( FIFO_SIZE * 2 ) ) new_trace_stop = ( FIFO_SIZE * 2 ) - 1;
        }
    else
        {
        if( new_trace_stop >= FIFO_SIZE ) new_trace_stop = FIFO_SIZE - 1;
        }
```

```
    if( new_trace_start >= new_trace_stop ) return( FALSE );

    /* update file level veriables, and return TRUE      */
    trace_start=new_trace_start;
    trace_stop=new_trace_stop;

    return( TRUE );
    }


/* gets the trace start and stop vales                   */
void get_range( long *send_range_start, long *send_range_stop )
    {
    /* send the parent func the range                    */
    *send_range_start = trace_start;
    *send_range_stop  = trace_stop;
    return;
    }


/* goes to the max range                                 */
void goto_max_range( void )
    {
    /* set the range to its max for this dso mode        */
    trace_start=0;
    if( scope_data.scope_mode == 1 ) trace_stop=( FIFO_SIZE * 2 ) - 1;
    else                             trace_stop=FIFO_SIZE - 1;
    }


/* pans or changes the current wave form                 */
void change_view( HWND window_handle, int mode )
    {
    long view_range, scale_factor;

    /* pre calc a few things                             */
```

```
view_range=trace_stop-trace_start;
/* calc pan or zoom factor                              */
if( mode == 0 || mode == 1 ) scale_factor=(long) ( view_range * ZOOM_RATIO );
else                         scale_factor=(long) ( view_range * PAN_RATIO  );
/* corrent int round down error                         */
if( scale_factor == 0 ) scale_factor=1;

/* how are we going to change the view                  */
switch( mode )
    {
    /* zoom in on the current view                      */
    case 0:
        trace_start = trace_start + scale_factor;
        trace_stop  = trace_stop  - scale_factor;
        break;
    /* zoom out on the current view                     */
    case 1:
        trace_start = trace_start - scale_factor;
        trace_stop  = trace_stop  + scale_factor;
        break;
    /* pan to the right on the current view             */
    case 2:
        trace_start = trace_start + scale_factor;
        trace_stop  = trace_stop  + scale_factor;
        break;
    /* pan to the left on the current view              */
    case 3:
        trace_start = trace_start - scale_factor;
        trace_stop  = trace_stop  - scale_factor;
        break;
    }

/* make sure trace values are still in range            */
if( trace_start < 0 ) trace_start = 0;
if( trace_stop  < 0 ) trace_stop  = 0;
if( scope_data.scope_mode == 1 )
```

```
    {
    /* single channel mode, can be twice as large    */
    if( trace_stop > ( ( FIFO_SIZE * 2 ) - 1 ) ) trace_stop=( FIFO_SIZE * 2 ) - 1;
    }
else
    {
    /* upto FIFO_SIZE                                */
    if( trace_stop > ( FIFO_SIZE - 1 ) )         trace_stop=FIFO_SIZE - 1;
    }
if( trace_start >= trace_stop )
    {
    if( trace_start != 0 ) trace_start--;
    if( scope_data.scope_mode == 1 )
        {
        if( trace_stop < ( ( FIFO_SIZE * 2 ) - 1 ) ) trace_stop++;
        }
    else
        {
        if( trace_stop < ( FIFO_SIZE - 1 ) ) trace_stop++;
        }
    }

/* depending on define alter pan behaviour           */
if( !prefs.zoom_pan_mode )
    {
    /* panning make sure we dont zoom at end         */
    switch( mode )
        {
        /* check we cant pan more than the end       */
        case 2:
            if( ( trace_stop - trace_start ) < view_range )
                trace_start=trace_stop-view_range;
            break;
        /* check we cant pan more than the start     */
        case 3:
            if( ( trace_stop - trace_start ) < view_range )
```

```
                trace_stop=view_range;
            break;
        }
    }

    /* invalidate client area, so a redraw takes place   */
    InvalidateRect( window_handle, NULL, TRUE );
    return;
    }


/* plots the scope data to the screen                    */
void plot_scope_data( HWND window_handle, PAINTSTRUCT new_paint_struct )
    {
    int i;
    HPEN pen_handle, old_pen_handle;
    HBRUSH brush_handle, old_brush_handle;
    RECT window_coordinates, plot_area_1, plot_area_2;

    /* setup the global paint struct                     */
    paint_struct=new_paint_struct;
    /* get the coordinates of the client window          */
    GetClientRect( window_handle, &window_coordinates );

    /* set pen color for background                      */
    pen_handle=CreatePen( PS_SOLID, 1, BACK_COLOR );
    old_pen_handle=SelectObject( paint_struct.hdc, pen_handle );
    /* setup brush                                       */
    brush_handle=CreateSolidBrush( BACK_COLOR );
    old_brush_handle=SelectObject( paint_struct.hdc, brush_handle );
    /* paint background                                  */
    Rectangle( paint_struct.hdc, window_coordinates.left, window_coordinates.top, window_coordinates.right,
              window_coordinates.bottom );
    /* delete brush used for background                  */
    SelectObject( paint_struct.hdc, old_brush_handle );
    DeleteObject( brush_handle );
```

```
/* set pen color for plotting                          */
pen_handle=CreatePen( PS_SOLID, 1, PLOT_COLOR );
pen_handle=SelectObject( paint_struct.hdc, pen_handle );
DeleteObject( pen_handle );

/* calc common axis coordinates                        */
plot_area_1.top   = window_coordinates.top   + PLOT_BORDER_GAP;
plot_area_1.right = window_coordinates.right - PLOT_BORDER_GAP;
plot_area_1.left  = window_coordinates.left  + PLOT_BORDER_GAP + PLOT_Y_AXIS_GAP;
/* is this a dual plot                                 */
if( scope_data.scope_mode == 0 || scope_data.scope_mode == 3 )
    {
    plot_area_2.right = plot_area_1.right;
    plot_area_2.left  = plot_area_1.left;
    }

/* what sort(s) of plots are we doing                  */
switch( scope_data.scope_mode )
    {
    /* plot normal dual channel mode                   */
    case 0:
        /* calc mode specific coordinates              */
        plot_area_2.bottom=window_coordinates.bottom - PLOT_BORDER_GAP;
        i=( plot_area_2.bottom - plot_area_1.top - PLOT_BORDER_GAP ) / 2;
        plot_area_1.bottom = plot_area_1.top    + i;
        plot_area_2.top    = plot_area_2.bottom - i;
        /* plot the two channels                       */
        plot_trace( scope_data.data.dual_channel.a , plot_area_1 );
        plot_trace( scope_data.data.dual_channel.b , plot_area_2 );
        break;
    /* plot single channel mode                        */
    case 1:
        /* calc mode specific coordinates              */
        plot_area_1.bottom=window_coordinates.bottom - PLOT_BORDER_GAP;
        /* plot the channel                            */
```

```
        plot_trace( scope_data.data.single_channel, plot_area_1 );
        break;
    /* plot 16 bit logic                              */
    case 2:
        /* calc mode specific coordinates             */
        plot_area_1.bottom=window_coordinates.bottom - PLOT_BORDER_GAP - PLOT_X_AXIS_GAP;
        /* plot the logic                             */
        plot_logic( NULL, scope_data.data.logic, plot_area_1 );
        break;
    /* Mixed logic and analogue plot                  */
    case 3:
        /* calc mode specific coordinates             */
        plot_area_2.bottom=window_coordinates.bottom - PLOT_BORDER_GAP - PLOT_X_AXIS_GAP;
        i=( plot_area_2.bottom - plot_area_1.top - PLOT_BORDER_GAP ) / 2;
        plot_area_1.bottom = plot_area_1.top    + i;
        plot_area_2.top    = plot_area_2.bottom - i;
        /* plot analugoe and logic channels           */
        plot_trace(   scope_data.data.dual_channel.b, plot_area_1 );
        plot_logic( scope_data.data.dual_channel.a, NULL, plot_area_2 );
        break;
    }

/* change color for pre_trig plot                     */
pen_handle=CreatePen( PS_SOLID, 1, TRIG_COLOR );
pen_handle=SelectObject( paint_struct.hdc, pen_handle );
DeleteObject( pen_handle );
/* plot the pre trig line                             */
switch( scope_data.scope_mode )
    {
    case 0:
        draw_pre_trig(   plot_area_1.left, plot_area_1.right, plot_area_1.top, plot_area_1.bottom    );
        draw_pre_trig(   plot_area_2.left, plot_area_2.right, plot_area_2.top, plot_area_2.bottom    );
        draw_trig_level( plot_area_1.left, plot_area_1.right, plot_area_1.top, plot_area_1.bottom, 1 );
        draw_trig_level( plot_area_2.left, plot_area_2.right, plot_area_2.top, plot_area_2.bottom, 2 );
        break;
    case 1:
```

```
        draw_pre_trig(   plot_area_1.left, plot_area_1.right, plot_area_1.top, plot_area_1.bottom    );
        draw_trig_level( plot_area_1.left, plot_area_1.right, plot_area_1.top, plot_area_1.bottom, 1 );
        break;
    case 2:
        draw_pre_trig(   plot_area_1.left, plot_area_1.right, plot_area_1.top, plot_area_1.bottom    );
        break;
    case 3:
        draw_pre_trig(   plot_area_1.left, plot_area_1.right, plot_area_1.top, plot_area_1.bottom    );
        draw_pre_trig(   plot_area_2.left, plot_area_2.right, plot_area_2.top, plot_area_2.bottom    );
        draw_trig_level( plot_area_1.left, plot_area_1.right, plot_area_1.top, plot_area_1.bottom, 2 );
        break;
    }

/* change pen colour for axis                          */
pen_handle=CreatePen( PS_SOLID, 1, AXIS_COLOR );
pen_handle=SelectObject( paint_struct.hdc, pen_handle );
DeleteObject( pen_handle );
/* draw the axis                                       */
switch( scope_data.scope_mode )
    {
    case 0:
        draw_x_axis(      plot_area_1.left, plot_area_1.right, ( i / 2 ) + plot_area_1.top );
        draw_x_axis(      plot_area_1.left, plot_area_1.right, ( i / 2 ) + plot_area_2.top );
        draw_y_axis_anal( plot_area_1.left, plot_area_1.top,   plot_area_1.bottom );
        draw_y_axis_anal( plot_area_2.left, plot_area_2.top,   plot_area_2.bottom );
        break;
    case 1:
        draw_x_axis( plot_area_1.left, plot_area_1.right, ((plot_area_1.bottom-plot_area_1.top )/2)+
                     plot_area_1.top );
        draw_y_axis_anal( plot_area_1.left, plot_area_1.top,   plot_area_1.bottom );
        break;
    case 2:
        draw_x_axis(      plot_area_1.left, plot_area_1.right, plot_area_1.bottom );
        draw_y_axis_logi( plot_area_1.left, plot_area_1.top,   plot_area_1.bottom, 16 );
        break;
    case 3:
```

```
            draw_x_axis(      plot_area_1.left, plot_area_1.right, ( i / 2 ) + plot_area_1.top );
            draw_x_axis(      plot_area_1.left, plot_area_1.right, plot_area_2.bottom );
            draw_y_axis_anal( plot_area_1.left, plot_area_1.top,   plot_area_1.bottom );
            draw_y_axis_logi( plot_area_2.left, plot_area_2.top,   plot_area_2.bottom, 8 );
            break;
        }

    /* restore original pen and return home              */
    SelectObject( paint_struct.hdc, old_pen_handle );
    DeleteObject( pen_handle );
    return;
    }


/* this function plots a data set as a scope trace       */
static void plot_trace( unsigned char dso_data[], RECT plot_area )
    {
    long i;
    unsigned char min, max;
    int x_axis, j, k;
    float sample_range_start, sample_range_stop, x_scale, y_scale;

    /* calculate the x & y scale value, and axis value    */
    x_scale=( trace_stop - trace_start ) / (float) ( plot_area.right - plot_area.left );
    i=plot_area.bottom - plot_area.top;
    y_scale=i/ (float) FIFO_RANGE;
    x_axis=( ( i / 2 ) + plot_area.top );


    /* what sort of plot are we doing                     */
    if( x_scale <= DOT_PLOT_THRESHOLD )
        {
        /* invert x_scale as we are scaling pixels:-      */
        /* and NOT the sampled data                       */
        x_scale=1 / x_scale;
        /* since this is the first dot do a move to       */
```

```
    k=(int)( ( 128 - dso_data[trace_start] ) * y_scale ) + x_axis;
    MoveToEx( paint_struct.hdc, plot_area.left, k, NULL );
    /* plot data use dot to represent sample          */
    for( i=trace_start ; i<=trace_stop ; i++ )
        {
        /* pre calc values so we dont do it twice     */
        j=(int)( ( i - trace_start   ) * x_scale ) + plot_area.left;
        k=(int)( ( 128 - dso_data[i] ) * y_scale ) + x_axis;
        /* draw the pixel                             */
        LineTo(   paint_struct.hdc, j, k );
        }
    }
else
    {
    /* plot data line to represent range of samples   */
    if( scope_data.scope_mode == 1) k=FIFO_SIZE * 2;
    else                            k=FIFO_SIZE;

    sample_range_start=(float)trace_start;
    for( i=plot_area.left ; i<=plot_area.right ; i++ )
        {
        /* NOTE                                       */
        /* sample range is calculated to avoid an     */
        /* int multiply every line that is drawn      */
        sample_range_stop=sample_range_start+x_scale;
        /* find min and max values in pixel range     */
        min=dso_data[ (int)sample_range_start ];
        max=dso_data[ (int)sample_range_start ];
        for( j=(int)sample_range_start ; j<sample_range_stop && j<k ; j++ )
            {
            if( dso_data[j] < min ) min=dso_data[j];
            if( dso_data[j] > max ) max=dso_data[j];
            }
        sample_range_start=sample_range_stop;
        /* draw vertical line for value range         */
        MoveToEx( paint_struct.hdc, i, (int)( ( 128 - min ) * y_scale ) + x_axis, NULL );
```

```
            LineTo(   paint_struct.hdc, i, (int)( ( 128 - max ) * y_scale ) + x_axis + 1 );
            }
        }
    }


/* plot DSO data in 8 bit and 16bit logic form           */
static void plot_logic( char dso_data8[], short dso_data16[],  RECT plot_area )
    {
    long i, j;
    BOOL last_bit, low_bit, high_bit;
    char bit_number, num_bits;
    unsigned short bit_mask, current_sample;
    int vert_pos_on, vert_pos_off;
    float bit_hight, bit_length, hori_pos, gap_hight;

    /* how many bits are we plotting                     */
    if( dso_data8 == NULL ) num_bits=16;
    else                    num_bits=8;
    /* calculate space for single channel & bit          */
    bit_hight=( plot_area.bottom - plot_area.top ) / (float) ( num_bits + LOGIC_CHANNEL_GAP );
    gap_hight=( bit_hight * LOGIC_CHANNEL_GAP ) / (float) ( num_bits + 1 );
    bit_length=( plot_area.right - plot_area.left ) / (float) ( trace_stop - trace_start );

    /* plot the channels                                 */
    for( bit_number=0 ; bit_number<num_bits ; bit_number++ )
        {
        /* calculate a few variables to speed things up  */
        bit_mask=( unsigned short ) pow( (double) 2, (double) bit_number );
        vert_pos_on=(int) ( ( ( gap_hight + bit_hight ) * bit_number ) + plot_area.top + gap_hight );
        vert_pos_off=(int) ( vert_pos_on + bit_hight );

        /* init setup, last_bit and pen possition        */
        if( num_bits == 8 ) current_sample=dso_data8[ trace_start ];
        else                current_sample=dso_data16[ trace_start ];
        if( ( bit_mask & current_sample ) == 0 )
```

```
    {
    last_bit=0;
    MoveToEx( paint_struct.hdc, plot_area.left, vert_pos_off, NULL );
    }
else
    {
    last_bit=1;
    MoveToEx( paint_struct.hdc, plot_area.left, vert_pos_on, NULL );
    }

/* check if we can use the quick plot method      */
if( bit_length < 1 )
    {
    /* more data to plot than pixels so use quick plot    */
    /* set through the pixcels ploting them              */
    for( i=plot_area.left ; i<plot_area.right ; i++ )
        {
        /* what values are there over pixcel range      */
        low_bit=0;
        high_bit=0;
        for( j=(long) ( ( i - plot_area.left ) / bit_length ) + trace_start ; j<( ( i + 1 –
             plot_area.left ) / bit_length ) + trace_start ; j++ )
            {
            /* get the sample we are deeling with       */
            if( num_bits == 8 ) current_sample=dso_data8[ j ];
            else                current_sample=dso_data16[ j ];
            /* check state of bit and update variables  */
            if( ( bit_mask & current_sample ) == 0 )
                low_bit=1;
            else
                high_bit=1;
            /* if we found high & low bits, end search  */
            if( low_bit && high_bit )
                break;
            }
```

```
        /* if found high & low draw a line              */
        if( low_bit && high_bit )
            {
            MoveToEx( paint_struct.hdc, i, vert_pos_off, NULL );
            LineTo(   paint_struct.hdc, i, vert_pos_on );
            }
        /* if found low bit only draw pixcel            */
        if( low_bit )
            {
            MoveToEx( paint_struct.hdc, i, vert_pos_off, NULL );
            LineTo(   paint_struct.hdc, i, vert_pos_off - 1 );
            }
        /* if found high bit only draw pixcel           */
        if( high_bit )
            {
            MoveToEx( paint_struct.hdc, i, vert_pos_on, NULL );
            LineTo(   paint_struct.hdc, i, vert_pos_on + 1 );
            }
        }
    }
else
    {
    /* data must be displayed normaly, so do it         */
    /* start the draw process                           */
    for( i=trace_start ; i<trace_stop ; i++ )
        {
        /* pre calc to save doing it twice              */
        hori_pos=plot_area.left + ( ( i - trace_start ) * bit_length );

        /* get the sample we are deeling with           */
        if( num_bits == 8 ) current_sample=dso_data8[ i ];
        else                current_sample=dso_data16[ i ];
        /* what is the state fo the bit                 */
        if( ( bit_mask & current_sample ) != 0 )
            {
            /* bit is on                                */
```

```
                    /* check if the last bit is also off        */
                    if(  last_bit == 0 )
                        {
                        /* last bit not same, so draw tran line */
                        LineTo( paint_struct.hdc, (int) hori_pos,  vert_pos_on );
                        }
                    /* update the last bit varaible             */
                    last_bit=1;
                    /* draw bit line                            */
                    LineTo( paint_struct.hdc, (int) ( hori_pos + bit_length ), vert_pos_on );
                    }
                else
                    {
                    /* bit is off                               */
                    /* check if the last bit is also on         */
                    if(  last_bit != 0 )
                        {
                        /* last bit not same, so draw tran line */
                        LineTo( paint_struct.hdc, (int) hori_pos, vert_pos_off );
                        }
                    /* update the last bit varaible             */
                    last_bit=0;
                    /* draw bit line                            */
                    LineTo( paint_struct.hdc, (int) ( hori_pos + bit_length ), vert_pos_off );
                    }
                }
            }
        }
    }


/* draws the x axis on the plots                               */
static void draw_x_axis( int left, int right, int top )
    {
    int tick_pos;
    char num_string[20], unit_string[5];
```

```
float bit_length, f;
long i, j, step_size, axis_start, axis_stop;

/* setup the text alighnment                              */
SetTextAlign( paint_struct.hdc, TA_CENTER );
/* first draw the axis line                               */
MoveToEx( paint_struct.hdc, left,  top, NULL );
LineTo(   paint_struct.hdc, right, top );

/* check if we are plotting time                          */
if( prefs.x_axis_time )
    {
    /* calc sample speed in MHz                           */
    if( scope_data.scope_mode == 1 ) bit_length=DSO_CLOCK_FREQ * 2;
    else                             bit_length=DSO_CLOCK_FREQ;
    for( i=0 ; i<scope_data.clock_div ; i++ ) bit_length=bit_length/2;
    /* calc time per sample in seconds                    */
    bit_length=(float) 0.000001/bit_length;
    /* calc time for displayed range                      */
    f=bit_length;
    f=( trace_stop - trace_start ) * bit_length;
    /* calc units and scale time per sample               */
    for( i=0 ; f<10 ; i++ )
        {
        f=f*1000;
        bit_length=bit_length*1000;
        }
    switch( i )
        {
        case 0: strcpy( unit_string, "S"  ); break;
        case 1:    strcpy( unit_string, "mS" ); break;
        case 2:    strcpy( unit_string, "uS" ); break;
        case 3:    strcpy( unit_string, "nS" ); break;
        }
    /* calc axis start and stop values                    */
    axis_start = (long) ( trace_start * bit_length );
```

```
    axis_stop  = (long) ( trace_stop  * bit_length );
    }
else
    {
    /* setup the internal axis start & stop values      */
    axis_start=trace_start;
    axis_stop=trace_stop;
    }

/* calc max number of chars in lable and gap            */
i=axis_stop;
for( j=1 ; i>10 ; j++ ) i=i/10;
if( prefs.x_axis_time ) i=j + 4;
else                    i=j + 1;
/* work out the number of lables we can have            */
/* get the width of a single char in current font       */
GetCharWidth( paint_struct.hdc, 'a', 'a', &j );
i=( right - left ) / ( i * j );

/* calculate large step size                            */
step_size=( axis_stop - axis_start ) / i;
/* now round up the step size                           */
if( step_size < 10 ) step_size=10;
else                 step_size=l_round_up( step_size );
/* calc the number of bits per pixel                    */
bit_length=( right - left ) / (float) ( axis_stop - axis_start );

/* draw all the large ticks and lable them              */
/* first calc start sample                              */
i=( axis_start / step_size ) * step_size;
/* if resultant is less than start increment one tick   */
if( i < axis_start ) i=i + step_size;
/* now get down to drawing those ticks                  */
for( ; i<=axis_stop ; i=i + step_size )
    {
    /* calc tick position and draw it                   */
```

```
        tick_pos=(int) ( bit_length * ( i - axis_start ) ) + left;
        MoveToEx( paint_struct.hdc, tick_pos, top, NULL );
        LineTo(   paint_struct.hdc, tick_pos, top + LARGE_TICK_LEN );
        /* lable the tick                                   */
        if( prefs.x_axis_time )    sprintf( num_string, "%d %s", i, unit_string );
        else                     sprintf( num_string, "%d", i );
        TextOut( paint_struct.hdc, tick_pos, top + LARGE_TICK_LEN, num_string, strlen( num_string ) );
        }

    /* draw all the small ticks                             */
    /* first calc start sample                              */
    step_size=step_size/10;
    i=( axis_start / step_size ) * step_size;
    /* if resultant is less than start increment one tick   */
    if( i < axis_start ) i=i + step_size;
    /* now get down to drawing those ticks                  */
    for( ; i<=axis_stop ; i=i + step_size )
        {
        /* calc tick position and draw it                   */
        tick_pos=(int) ( bit_length * ( i - axis_start ) ) + left;
        MoveToEx( paint_struct.hdc, tick_pos, top, NULL );
        LineTo(   paint_struct.hdc, tick_pos, top + SMALL_TICK_LEN );
        }

    return;
    }

/* draws the y axis on the plots                            */
static void draw_y_axis_anal( int left, int top, int bottom )
    {
    int lable_num;
    char num_string[10];
    float i, spaceing;

    /* setup the text alighnment                            */
    SetTextAlign( paint_struct.hdc, TA_RIGHT );
```

```
    /* calc a few things like tick interval                */
    spaceing=( bottom - top ) / (float) 8;

    /* first draw the axis line                            */
    MoveToEx( paint_struct.hdc, left,  top, NULL );
    LineTo(   paint_struct.hdc, left,  bottom );

    /* draw the ticks                                      */
    lable_num=128;
    for( i=(float) top ; i<=bottom ; i=i+spaceing )
        {
        /* draw the tick                                   */
        MoveToEx( paint_struct.hdc, left, (int) i, NULL );
        LineTo(   paint_struct.hdc, left - LARGE_TICK_LEN,  (int) i );
        /* output the lable for the tick                   */
        itoa( lable_num, num_string, 10 );
        TextOut( paint_struct.hdc, left - LARGE_TICK_LEN, (int) i - 8, num_string ,strlen(num_string) );
        /* decriment lable_num so we display next lable    */
        lable_num=lable_num-32;
        }

    return;
    }

/* draw the axis for the logic plots                       */
static void draw_y_axis_logi( int left, int top, int bottom, int num_bits )
    {
    char lable_string[10];
    float i, j, tick_spaceing, tick_offset;

    /* setup the text alighnment                           */
    SetTextAlign( paint_struct.hdc, TA_RIGHT );

    /* pre calc a few things like tick spaceing and offset  */
    j=( bottom - top ) / (float) ( num_bits + LOGIC_CHANNEL_GAP );
```

```
i=( j * LOGIC_CHANNEL_GAP ) / ( num_bits + 1 );
tick_spaceing = i + j;
tick_offset   = i + ( j / 2 );

/* first draw the axis line                              */
MoveToEx( paint_struct.hdc, left,  top, NULL );
LineTo(   paint_struct.hdc, left,  bottom );

j=0;
for( i=tick_offset + top ; i<bottom ; i=i + tick_spaceing )
    {
    /* draw the tick                                     */
    MoveToEx( paint_struct.hdc, left, (int) i, NULL );
    LineTo(   paint_struct.hdc, left - LARGE_TICK_LEN,  (int) i );
    /* generate the text for the lable                   */
    if( j == 0 )
        strcpy( lable_string, "LSB 0" );
    else
        {
        if( j == num_bits - 1 )
            sprintf( lable_string, "MSB %d", (int) j );
        else
            itoa( (int) j, lable_string, 10 );
        }
    /* now output the label                              */
    TextOut( paint_struct.hdc, left - LARGE_TICK_LEN, (int) i - 8, lable_string ,strlen(lable_string) );
    /* increment j, the bit number                       */
    j++;
    }

return;
}
```

```
/* draw line to represent pre trig                          */
static void draw_pre_trig(    int left, int right, int top, int bottom )
    {
    int position;
    float bit_length;

    /* check if we should draw pre-trig                     */
    if( prefs.show_pre_trig == FALSE) return;
    /* check if the pre-trig value is in display range      */
    if( scope_data.pre_trigger > (unsigned) trace_stop  ) return;
    if( scope_data.pre_trigger < (unsigned) trace_start ) return;

    /* calc number of pixcels per sample                    */
    bit_length=( right - left ) / (float)( trace_stop - trace_start );

    /* calc position of pre_trig                            */
    position=(int)( ( scope_data.pre_trigger - trace_start ) * bit_length );
    position=position + left;

    /* draw the line                                        */
    MoveToEx( paint_struct.hdc, position, top, NULL );
    LineTo(   paint_struct.hdc, position, bottom );

    return;
    }


/* draw trig level                                          */
static void draw_trig_level( int left, int right, int top, int bottom, int channel )
    {
    int position;
    float bit_hight;

    /* check if we should draw the trig line                */
    if( prefs.show_trig_level == FALSE) return;
    switch( scope_data.trig_sel )
```

```
    {
    /* no trig line beause ext trig or no trig          */
    case 0:
    case 3:
        return;
        break;
    /* trig channel a selected                          */
    case 1:
        if( scope_data.scope_mode > 1 || channel != 1 ) return;
        break;
    /* trig off channel b                               */
    case 2:
        if( channel != 2 ) return;
        break;
    }

/* work out where the line is to be drawn               */
bit_hight=( bottom - top ) / (float) FIFO_RANGE;

/* calc position of trig level                          */
position=(int) ( bit_hight * ( FIFO_RANGE - scope_data.trig_value - 1 ) );
position=position + top;

/* draw the trig level                                  */
MoveToEx( paint_struct.hdc, left,  position, NULL );
LineTo(   paint_struct.hdc, right, position );

return;
}
```

8.1.6) Plot_Data.h

```
/* check if we have included this header files before */
#ifndef FILE_PLOT_DATA_HEADER
    /* define FILE_PLOT_DATA_HEADER so we dont include this file again */
    #define FILE_PLOT_DATA_HEADER

    /* internal and external include statments */
    #include "windows.h"

    /* Check whether if this file is being used for plot_data.c or another file */
    #ifdef FILE_PLOT_DATA
        /* File used for plot_data.c */
        #define FILE_PLOT_DATA_EXTERN

        /* internal defines    */
        #define PLOT_BORDER_GAP           20
        #define PLOT_Y_AXIS_GAP           55
        #define PLOT_X_AXIS_GAP           25
        #define DOT_PLOT_THRESHOLD        10
        #define LOGIC_CHANNEL_GAP         2
        #define BACK_COLOR                RGB( 255, 255, 255 )
        #define PLOT_COLOR                RGB(   0,   0, 255 )
        #define AXIS_COLOR                RGB( 255,   0,   0 )
        #define TRIG_COLOR                RGB(   0, 255,   0 )
        #define SMALL_TICK_LEN            5
        #define LARGE_TICK_LEN            10
        #define ZOOM_RATIO                0.2
        #define PAN_RATIO                 0.2

        /* internal func declirations */
        static void plot_logic( char dso_data8[], short dso_data16[],  RECT plot_area );
        static void plot_trace( unsigned char dso_data[],              RECT plot_area );
        static void draw_x_axis(      int left, int right, int top );
        static void draw_pre_trig(    int left, int right, int top, int bottom );
```

```
        static void draw_trig_level(  int left, int right, int top, int bottom, int channel );
        static void draw_y_axis_anal( int left, int top,   int bottom );
        static void draw_y_axis_logi( int left, int top,   int bottom, int num_bits );
    #else
        /* File used for other source file */
        #define FILE_PLOT_DATA_EXTERN extern
    #endif

    /* internal and external function declirations */
    FILE_PLOT_DATA_EXTERN void goto_max_range( void );
    FILE_PLOT_DATA_EXTERN void change_view( HWND window_handle, int mode );
    FILE_PLOT_DATA_EXTERN BOOL set_range( long   new_trace_start, long new_trace_stop );
    FILE_PLOT_DATA_EXTERN void get_range( long *send_range_start, long *send_range_stop );
    FILE_PLOT_DATA_EXTERN void plot_scope_data( HWND window_handle, PAINTSTRUCT new_paint_struct );
#endif
```

8.1.7) Error.c

```
/* Local Define Statments */
#define FILE_ERROR

/* General header files */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>

/* Specific header files */
#include "error.h"
#include "main.h"

/* error list                                                 */
static struct error_struct error_list[ ERROR_LIST_SIZE ] =
    {
        /* error_number, error_level, error_message        */
        {  1, 10, "Unable To Allocate Memory" },
        {  2, 20, "Buffer/Array Overflow" },
        {  3, 20, "Invalid Linked List Node" },
        {  4,  5, "Could Not Find Any Directory Enteries ( Including . And .. )" },
        {  5,  5, "Could Not Change To Directory" },
        {  6,  5, "Could Not Create Directory" },
        {  7,  5, "Could Not Write To File" },
        {  8,  5, "Could Not Read From File" },
        {  9,  5, "Could Not Open File" },
        { 10, 10, "Parallel Port NOT Bi-Directional" },
        { 11,  1, "Not In Valid Range" },
        { 12, 10, "Unable To Create Thread" },
        { 13, 10, "Windows Error" },
        { 14,  1, "\"Start_Point\" Not Valid, Check Cable" },
        { 15,  1, "Offset Correct Can Only Be Used In Single Channel Mode" }
    };
```

```
/* Main error handler code                                    */
void error( int error_code, char function_name[ FUNC_NAME_LEN ], char extra_info[ ERROR_MESS_LEN ] )
    {
    char error_string[ ERROR_MESS_LEN ];
    int i, error_level;

    /* find the error in the error table                      */
    for( i=0 ; i<ERROR_LIST_SIZE && error_list[i].error_number!=error_code ; i++ );

    /* generate error messate                                 */
    /* check if we found the error or not                     */
    if( i == ERROR_LIST_SIZE )
        {
        /* unknow error message output                        */
        sprintf
            (
            error_string,
            "The Error Handler Was Unable To Identify The Error Code\n\n"
            "Error Infomation:-\n"
            "     Error Code : %d\n"
            "    In Function : %s\n"
            "Additional Info : %s",
            error_code,
            function_name,
            extra_info
            );
        error_level=20;
        }
    else
        {
        /* setup the local error level                        */
        error_level=error_list[i].error_level;
        /* what is the severity of the error message          */
        if( error_level < 5 )
            {
            /* just a warbing to user                         */
```

```
        strcpy( error_string, error_list[i].error_message );
        }
    else
        {
        /* generate standard error message            */
        sprintf
            (
            error_string,
            "Error Infomation:-\n"
            "              Error : %s\n"
            "    Error Level : %d\n"
            "    In Function : %s\n"
            "Additional Info : %s",
            error_list[i].error_message,
            error_level,
            function_name,
            extra_info
            );
        }
    }
/* add author info if serious error                   */
if( error_level >= 20 )
    {
    sprintf
        (
        &error_string[ strlen( error_string ) ],
        "\n\n"
        "Please Inform the Author At:-\n"
        "       %s\n"
        "Giving Him A Copy Of Your Prefs File And The Circumstances Of The Error\n"
        "Also Inculde The Name And Version Of The Program, And The Above Infomation\n"
        "Thank You For Your Help\n\n",
        AUTHOR_MAIL_ADDRESS
        );
    }
```

```
/* show the corrent error box and return               */
if( error_level < 5 )
    {
    MessageBox( GetForegroundWindow(), error_string, "Error", MB_ICONWARNING );
    return;
    }
else if( error_level <= 10 )
    {
    MessageBox( GetForegroundWindow(), error_string, "Fatal Error - Program Halted", MB_ICONERROR );
    quit( error_level );
    }
else
    {
    MessageBox( GetForegroundWindow(), error_string, "Fatal Error - Program Halted", MB_ICONERROR );
    exit( error_level );
    }
}
```

8.1.8) Error.h

```
/* check if we have included this header files before */
#ifndef FILE_ERROR_HEADER
    /* define FILE_ERROR_HEADER so we dont include this file again */
    #define FILE_ERROR_HEADER

    /* include file statments */
    #include "define.h"

    /* Check whether if this file is being used for error.c or another file */
    #ifdef FILE_ERROR
        /* File used for error.c */
        #define FILE_ERROR_EXTERN
        #define ERROR_LIST_SIZE   15

        /* internal structure declirations */
        static struct error_struct
            {
            int error_number;
            int error_level;
            char error_message[ ERROR_MESS_LEN ];
            };
    #else
        /* File used for other source file */
        #define FILE_ERROR_EXTERN extern
    #endif

    /* function declirations */
    FILE_ERROR_EXTERN void error( int error_code, char function_name[ FUNC_NAME_LEN ], char extra_info[
ERROR_MESS_LEN ] );
#endif
```

8.1.9) File_IO.c

```
/* local define statments */
#define FILE_FILE_IO


/* Include Statments */
#include "file_io.h"
#include "error.h"
#include "dso_io.h"
#include <stdio.h>
#include <windows.h>


/* save the dso structure to a file      */
int save_dso_structure( char file_name[ FILE_NAME_LEN ] )
    {
    int i;
    HANDLE file_handle;

    /* open the file with error check    */
    file_handle=CreateFile( file_name, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if( file_handle    == INVALID_HANDLE_VALUE )
        error( 9, "save_dso_structure", file_name );

    /* write the data out to the file    */
    WriteFile( file_handle, &scope_data, sizeof( struct scope_data_struct ), &i, NULL );
    if( i != sizeof( struct scope_data_struct ) )
        error( 7, "save_dso_structure", file_name );

    /* close the file, and return        */
    CloseHandle( file_handle );
    return(0);
    }
```

```
/* load the dso structure to a file      */
int load_dso_structure( char file_name[ FILE_NAME_LEN ] )
    {
    int i;
    char string[30];
    HANDLE file_handle;

    /* open the file with error check    */
    file_handle=CreateFile( file_name, GENERIC_READ, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if( file_handle    == INVALID_HANDLE_VALUE )
        error( 9, "load_dso_structure", file_name );

    /* read the data out to the file     */
    ReadFile( file_handle, &scope_data, sizeof( struct scope_data_struct ), &i, NULL );
    if( i != sizeof( struct scope_data_struct ) )
        {
        ltoa(GetLastError(), string, 10 );
        error( 8, "load_dso_structure", string);
        }

    /* close the file, and return        */
    CloseHandle( file_handle );
    return(0);
    }

/* save the raw data out as a CSV file          */
int export_csv( char file_name[ FILE_NAME_LEN ] )
    {
    long i;
    FILE *fp;
    union convert_union convert;

    /* open the file with error check           */
    fp=fopen( file_name, "w" );
    if( fp == NULL )
        error( 9, "save_dso_structure", file_name );
```

```
/* what format is the data in               */
switch( scope_data.scope_mode )
    {
    /* output standard dual channel mode    */
    case 0:
        for( i=0 ; i<FIFO_SIZE ; i++ )
            fprintf( fp, "%d,%d\n", scope_data.data.dual_channel.a[i], scope_data.data.dual_channel.b[i] );
        break;

    /* output in single channel mode        */
    case 1:
        for( i=0 ; i<( FIFO_SIZE * 2 ) ; i++ )
            fprintf( fp, "%d\n", scope_data.data.single_channel[i] );
        break;

    /* 16 bit logic mode                    */
    case 2:
        for( i=0 ; i<FIFO_SIZE ; i++ )
            {
            /* union so can get bit pattern */
            convert.logic=scope_data.data.logic[i];

            /* output the string to file    */
            fprintf( fp, "%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d\n",
                convert.bits.p,
                convert.bits.o,
                convert.bits.n,
                convert.bits.m,
                convert.bits.l,
                convert.bits.k,
                convert.bits.j,
                convert.bits.i,
                convert.bits.h,
                convert.bits.g,
                convert.bits.f,
```

```
                convert.bits.e,
                convert.bits.d,
                convert.bits.c,
                convert.bits.b,
                convert.bits.a
                );
            }
        break;

    /* mixed mode, logic & analogue         */
    case 3:
        for( i=0 ; i<FIFO_SIZE ; i++ )
            {
            /* union so can get bit pattern */
            convert.logic=scope_data.data.dual_channel.a[i];

            /* output the string to file    */
            fprintf( fp, "%d,%d,%d,%d,%d,%d,%d,%d,%d\n",
                scope_data.data.dual_channel.b[i],
                convert.bits.h,
                convert.bits.g,
                convert.bits.f,
                convert.bits.e,
                convert.bits.d,
                convert.bits.c,
                convert.bits.b,
                convert.bits.a
                );
            }
        break;
    }

/* close the file, and return               */
fclose( fp );
return(0);
}
```

8.1.10) File_IO.h

```
/* check if we have included this header files before */
#ifndef FILE_ERROR_HEADER
    /* define FILE_ERROR_HEADER so we dont include this file again */
    #define FILE_ERROR_HEADER

    /* include file statments */
    #include "define.h"

    /* Check whether if this file is being used for error.c or another file */
    #ifdef FILE_ERROR
        /* File used for error.c */
        #define FILE_ERROR_EXTERN
        #define ERROR_LIST_SIZE   15

        /* internal structure declirations */
        static struct error_struct
            {
            int error_number;
            int error_level;
            char error_message[ ERROR_MESS_LEN ];
            };
    #else
        /* File used for other source file */
        #define FILE_ERROR_EXTERN extern
    #endif

    /* function declirations */
    FILE_ERROR_EXTERN void error( int error_code, char function_name[ FUNC_NAME_LEN ], char extra_info[
ERROR_MESS_LEN ] );
#endif
```

8.1.11) Numeric.c

```
/* local define statments */
#define FILE_NUMERIC


/* local include files */
#include "dso_io.h"


/* rounds a number up to the form #000000000    */
long l_round_up( long num )
    {
    int i, j;
    long r_num;

    /* round the number                          */
    r_num=num;
    for( j=0 ; r_num>=10 ; j++ ) r_num=r_num/10;
    for( i=j; i>0 ; i-- ) r_num=r_num*10;

    /* round up to the next seg if neaded        */
    if( r_num < num)
        r_num=r_num + (long) pow( 10, j );

    return( r_num );
    }


/* re-aligns single channel interleve data      */
BOOL offset_correct( void )
    {
    char offset;
    long pos, point_count=0;
    float chan_a_average=0, chan_b_average=0;
```

```
/*  check dso in single channel mode        */
if( scope_data.scope_mode != 1 ) return( TRUE );

/* first get average of the two channels    */
for( pos=0 ; pos < ( FIFO_SIZE * 2 ) ; pos=pos + 2 )
    {
    /* check neither channels is clipped    */
    if( scope_data.data.single_channel[ pos     ] == 0 || scope_data.data.single_channel[ pos     ] == 255
      ) continue;
    if( scope_data.data.single_channel[ pos + 1 ] == 0 || scope_data.data.single_channel[ pos + 1 ] == 255
      ) continue;

    /* values are valid so add to averate   */
    chan_a_average=chan_a_average + scope_data.data.single_channel[ pos ];
    chan_b_average=chan_b_average + scope_data.data.single_channel[ pos + 1 ];
    /* are adding point so increment count  */
    point_count++;
    }

/* calculate the two averages               */
chan_a_average=chan_a_average / point_count;
chan_b_average=chan_b_average / point_count;
/* calculate channel a offset               */
offset=(char)( chan_b_average - chan_a_average );

/* correct offset in data                   */
for( pos=0 ; pos < ( FIFO_SIZE * 2 ) ; pos=pos + 2 )
    {
    /* check neither channels is clipped    */
    if( scope_data.data.single_channel[ pos     ] == 0 || scope_data.data.single_channel[ pos     ] == 255
      ) continue;
    if( scope_data.data.single_channel[ pos + 1 ] == 0 || scope_data.data.single_channel[ pos + 1 ] == 255
      ) continue;
```

```
        /* will offset push data out of range   */
        if( (int)scope_data.data.single_channel[ pos ] + (int)offset > 255 ) continue;

        /* corrent offset                       */
        scope_data.data.single_channel[ pos ]=scope_data.data.single_channel[ pos ] + offset;
        }

    /* finished so go home                      */
    return( FALSE );
    }
```

8.1.12) Numeric.h

```
/* check if we have included this header files before */
#ifndef FILE_NUMERIC_HEADER
    /* define FILE_ERROR_HEADER so we dont include this file again */
    #define FILE_NUMERIC_HEADER

    /* Check whether if this file is being used for error.c or another file */
    #ifdef FILE_NUMERIC
        /* File used for error.c */
        #define FILE_NUMERIC_EXTERN
    #else
        /* File used for other source file */
        #define FILE_NUMERIC_EXTERN extern
    #endif

    /* function declirations */
    FILE_NUMERIC_EXTERN long l_round_up( long num );
    FILE_NUMERIC_EXTERN BOOL offset_correct( void );
#endif
```

8.1.13) DSO_IO.c

```
/* local define statments */
#define FILE_DSO_IO


/* include statments */
#include "dso_io.h"
#include "p_port_io.h"
#include "main.h"
#include "gui.h"
#include "error.h"
#include "resource.h"
#include "define.h"


/* file level verables */
static BOOL cancel_trig_wait;
static HWND prog_wind_pointer;
static long start_point=0;


/* records the data from the selected source    */
DWORD WINAPI capture_thread_func( LPVOID progress_window_handler )
    {
    /* setup global window handler              */
    prog_wind_pointer=progress_window_handler;

    /* init the dso                             */
    dso_setup();

    /* tell the DSO to capture data             */
    dso_capture();
```

```
    /* tell user now downloading                */
    SendMessage( prog_wind_pointer, WM_COMMAND, IDD_DSO_DOWNLOAD, 0 );
    /* download DSO data                        */
    get_dso_data();

    /* get rid of progress box                  */
    PostMessage( prog_wind_pointer, WM_COMMAND, IDOK, 0 );
    ExitThread( 0 );
    return( 0);
    }


/* cancels the wait for trigger pulse           */
void cancel_capture( void )
    {
    cancel_trig_wait=TRUE;
    }


/* setup for record                             */
static void dso_setup( void )
    {
    char i;

    /* make sure that both fifo's are tri-state */
    set_reg( 0x3, 0x0 );
    set_dso_control( 0x6 | CONTROL_BIDI_BIT );
    set_dso_control( 0x7 | CONTROL_BIDI_BIT );
    set_dso_control( 0x0 );

    /* first set the first 8 bits               */
    set_reg( 0x2, (char)( scope_data.pre_trigger & 0xFF ) );
    /* pulse the pre_trig_load0 bit             */
    set_reg( 0x3, 0x0 );
    set_reg( 0x3, 0x1 );
    set_reg( 0x3, 0x0 );
```

```
/* now set the second 8 bits                */
set_reg( 0x2, (char)( ( scope_data.pre_trigger & 0xFF00 ) >>8 ) );
/* pulse the pre_trig_load1 bit             */
set_reg( 0x3, 0x0 );
set_reg( 0x3, 0x2 );
set_reg( 0x3, 0x0 );
/* now set the final 3 bits                 */
set_reg( 0x2, (char)( ( scope_data.pre_trigger & 0x070000 ) >>16 ) );
/* pulse the pre_trig_load2 bit             */
set_reg( 0x3, 0x0 );
set_reg( 0x3, 0x4 );
set_reg( 0x3, 0x0 );

/* set the opperating mode of the DSO       */
switch( scope_data.scope_mode )
    {
    case 0: i=0; break;
    case 1:    i=1; break;
    case 2:    i=6; break;
    case 3:    i=2; break;
    }
/* setup the clock_div value                */
i=i | ( scope_data.clock_div << 5 );
/* setup the trig source value              */
i=i | ( scope_data.trig_sel  << 3 );
/* set the register                         */
set_reg( 0x4, i );

/* setup the trig value reg                 */
if( scope_data.scope_mode == 2 )
    {
    /* 16 bit trig value                    */
    /* program the low byte                 */
    set_reg( 0x2, (char) scope_data.trig_value );
    set_reg( 0x5, 0x0 );
    set_reg( 0x5, 0x1 );
```

```
        set_reg( 0x5, 0x0 );
        /* program the high byte                */
        set_reg( 0x2, (char) ( scope_data.trig_value >> 8 ) );
        set_reg( 0x5, 0x0 );
        set_reg( 0x5, 0x2 );
        set_reg( 0x5, 0x0 );
        }
    else
        {
        /* 8 bit trig value, put in both registers      */
        set_reg( 0x2, (char) scope_data.trig_value );
        /* pulse both load reg lines at same time       */
        set_reg( 0x5, 0x0 );
        set_reg( 0x5, 0x3 );
        set_reg( 0x5, 0x0 );
        }
    /* set the greater than bit                         */
    if( scope_data.trig_falling ) set_reg( 0x5, 0x4 );

    /* init the dso control and FIFO mem wrt pointers   */
    set_reg( 0x3, 0x0  );
    set_reg( 0x3, 0x10 );
    set_reg( 0x3, 0x0  );

    /* DSO ready to role so return                      */
    return;
    }


/* tells the DSO to capture the data                                    */
static void dso_capture( void )
    {
    int i;
    float bit_length;

    /* wait, that we can garantee the fifos have been filled once       */
```

```
    /* calc sample speed in MHz                                          */
    bit_length=DSO_CLOCK_FREQ;
    for( i=0 ; i<scope_data.clock_div ; i++ ) bit_length=bit_length/2;
    /* calc time per sample in seconds                                   */
    bit_length=(float) 0.000001/bit_length;
    /* calc time for pre trig in mS, add one incase of round down        */
    i=(int) ( ( bit_length * scope_data.pre_trigger ) * 1000 ) + 1;
    /* now wait for that about of time                                   */
    Sleep( i );

    /* set the rec enable bit                                            */
    set_reg( 0x3, 0x8 );

    /* wait until the DSO signals that it has finished the capture       */
    cancel_trig_wait=FALSE;
    while( ( p_port_get_status() & 0x40 ) == 0x0 )
        {
        /* check if we should cancel trig wait                           */
        if( cancel_trig_wait == TRUE ) ExitThread( 0 );
        /* sleep so we dont chew up lots of cpu time                     */
        Sleep( TRIGGER_CHECK_TIME );
        }

    /* disable rec_enable bit                                            */
    set_reg( 0x3, 0x0 );
    return;
    }


/* downloads the data from the dso into the correct type of array        */
static void get_dso_data( void )
    {
    long i;
    unsigned char fifo_a[ FIFO_SIZE ], fifo_b[ FIFO_SIZE ];
```

```
/* slow fifo_wrt_clk down to 1.6MHz to reduce noice during download */
    set_reg( 0x4, 0xE0 );

    /* read in the start point for alignment                              */
    start_point=get_start_point();
    if( start_point > MAX_PRE_TRIG || start_point < 0 )
        {
        error( 14, "get_dso_data", "" );
        return;
        }

    /* get the data from the two fifo's                                   */
    get_fifo_data( fifo_a, TRUE,  TRUE );
    get_fifo_data( fifo_b, FALSE, TRUE );

    /* copy data from buffers to data structure. this is dependant on DSO mode  */
    switch( scope_data.scope_mode )
        {
        /* normal, mixed mode so no alteration in format                  */
        case 0:
        case 3:
            for( i=0 ; i<FIFO_SIZE ; i++  )
                {
                scope_data.data.dual_channel.a[i]=fifo_a[i];
                scope_data.data.dual_channel.b[i]=fifo_b[i];
                }
            break;

        /* single channel mode so must do interleve                       */
        case 1:
            for( i=0 ; i<FIFO_SIZE ; i++  )
                {
                scope_data.data.single_channel[ i + i ]=fifo_a[i];
                scope_data.data.single_channel[ i + i + 1 ]=fifo_b[i];
                }
            break;
```

```
        /* 16 logic mode so combine to a single short                          */
        case 2:
            for( i=0 ; i<FIFO_SIZE ; i++  )
                scope_data.data.logic[i]= fifo_a[i] | ( fifo_b[i] << 8 );
            break;
        }

    /* make sure that both fifo's are tri-state                                */
    set_reg( 0x3, 0x0 );
    set_dso_control( 0x6 | CONTROL_BIDI_BIT );
    set_dso_control( 0x7 | CONTROL_BIDI_BIT );
    set_dso_control( 0x0 );

    /* all done, dso data now correctly aligned, and stored, so return home    */
    return;
    }


/* sets a internal register to a given value                                   */
void set_reg( unsigned char reg_num, unsigned char value )
    {
    /* output the value to the data bus                                        */
    p_port_put_data( value );

    /* pulse the control latch line                                            */
    set_dso_control( 0x0     );
    set_dso_control( reg_num );
    set_dso_control( 0x0     );

    return;
    }
```

```
/* setup a value to the internal control register                          */
void set_dso_control( unsigned char reg_num )
    {
    char new_reg_num;

    /* calc the shifted reg num                                            */
    new_reg_num=reg_num<<1;

    /* output data to internal control reg                                 */
    p_port_put_control( new_reg_num );
    /* pulse clock pin to latch control data into control reg              */
    p_port_put_control( (char) ( new_reg_num | 0x1 ) );
    /* recover from clock pulse                                            */
    p_port_put_control( new_reg_num );
    }


/* gets data from one of the internal altera registers                     */
static unsigned char get_reg( unsigned char reg_num )
    {
    char new_reg_num;

    /* calc the shifted reg num    & OR in the bi-di bit on the parrallel port  */
    new_reg_num=reg_num | CONTROL_BIDI_BIT;

    /* output data to internal control reg                                 */
    set_dso_control( new_reg_num );
    /* now get the data from the data bus                                  */
    new_reg_num=p_port_get_data();
    /* now set internal control reg to 0 to reverse data bus direction     */
    set_dso_control( 0x0 );

    /* return the num we got from the reg                                  */
    return( new_reg_num );
    }
```

```
/* inverts the byte passed to it                */
static unsigned char invert_byte( unsigned char input_byte )
    {
    unsigned char i, output_byte=0;

    /* cycle through all 8 bits                 */
    for( i=0 ; i<8 ; i++ )
        {
        /* shift output so lsb is free          */
        output_byte=output_byte << 1;
        /* add the l significant bit to output  */
        output_byte=output_byte | ( input_byte & 0x1 );
        /* shit input so we can test next bit   */
        input_byte=input_byte   >> 1;
        }

    return( output_byte );
    }


/* reads in start point in the fifo from altera                    */
static long get_start_point( void )
    {
    long start_point;

    /* read in the data from the START_POINT register              */
    /* get the lowest byte                                         */
    set_reg( 0x5, 0x8 );
    start_point=get_reg( 0x1 );
    /* get the bits 8-15                                           */
    set_reg( 0x5, 0x10 );
    start_point=start_point | ( get_reg( 0x1 ) << 8 );
    /* get the bits 16-18                                          */
    set_reg( 0x5, 0x20 );
    start_point= start_point | ( ( get_reg( 0x1 ) & 0x7 ) << 16 );
```

```
    /* decriment start point because HW counter reset to 1 not 0,   */
    /* this saves a lot of complexe logic in altera                 */
    start_point--;

    return( start_point );
    }


/* get fifo data                                                                */
void get_fifo_data( unsigned char fifo_buf[], BOOL sel_fifo_a, BOOL show_progress )
    {
    long j, i;

    /* reset fifo and sel fifo                                                  */
    dso_reset_rd();
    fifo_select( sel_fifo_a );

    /* do we enable messaging back to progress window                           */
    if( !show_progress ) j=UPDATE_INTERVAL + 1;
    else                 j=0;
    /* setup the oe pin so when clock goes high output is enabled               */
    set_dso_control( 0x6 | CONTROL_BIDI_BIT );
    /* step through the FIFO reading it in and doing alighnemt at the same time */
    for( i=0 ; i<FIFO_SIZE ; i++ )
        {
        /* is it about time we updated the progress indicator                   */
        if( j == UPDATE_INTERVAL )
            {
            /* calc % done                                                      */
            j=( i * 50 ) / FIFO_SIZE;
            if( !sel_fifo_a ) j=j+50;
            /* update the progress message                                      */
            SendMessage( prog_wind_pointer, WM_COMMAND, IDD_DSO_PROGRESS, j );
            /* reset counter                                                    */
            j=0;
```

```
        }
    else j++;

    /* pulse the read clock to advance the mem pointer                    */
    set_dso_control( 0x7 | CONTROL_BIDI_BIT );
    /* read in the data from the fifo                                     */
    if( i < start_point )
        {
        /* before the start point so place the data at the end of the array */
        #ifdef BYTE_INVERT
            fifo_buf[ FIFO_SIZE - start_point + i ]=invert_byte( p_port_get_data() );
        #else
            fifo_buf[ FIFO_SIZE - start_point + i ]=p_port_get_data();
        #endif
        }
    else
        {
        /* after the start point so place the data at the begining of array */
        #ifdef BYTE_INVERT
            fifo_buf[ i - start_point ]=invert_byte( p_port_get_data() );
        #else
            fifo_buf[ i - start_point ]=p_port_get_data();
        #endif
        }
    /* recover from read clock pulse                                      */
    set_dso_control( 0x6 | CONTROL_BIDI_BIT );
    }

return;
}
```

```
/* reset the read pointer to the begining of fifo   */
void dso_reset_rd( void )
    {
    /* do acctual fifo sel and reset read pointer   */
    set_reg( 0x3, 0x0 );
    set_dso_control( 0x7 | CONTROL_BIDI_BIT );
    set_dso_control( 0x6 | CONTROL_BIDI_BIT );
    set_reg( 0x3, 0x20 );
    }


/* selects a fifo for output                        */
static void fifo_select( BOOL sel_fifo_a )
    {
    /* select the correct FIFO for output           */
    if( sel_fifo_a )
        set_reg( 0x3, 0xE0 );
    else
        set_reg( 0x3, 0xA0 );
    return;
    }


/* copy a to b                                      */
void copy_a_b( void )
    {
    long i;

    for( i=0 ; i<FIFO_SIZE ; i++ )
        scope_data.data.dual_channel.b[i]=scope_data.data.dual_channel.a[i];
    return;
    }
```

```
/* copy b to a                                          */
void copy_b_a( void )
    {
    long i;

    for( i=0 ; i<FIFO_SIZE ; i++ )
        scope_data.data.dual_channel.a[i]=scope_data.data.dual_channel.b[i];
    return;
    }

/* swaps the data for channels a and b                  */
void swap_a_b( void )
    {
    long i;
    unsigned char buf;

    for( i=0 ; i<FIFO_SIZE ; i++ )
        {
        buf=scope_data.data.dual_channel.a[i];
        scope_data.data.dual_channel.a[i]=scope_data.data.dual_channel.b[i];
        scope_data.data.dual_channel.b[i]=buf;
        }
    return;
    }


/* clears the contence of the data arrays               */
void clear_dso_data( void )
    {
    long i;

    for( i=0 ; i < ( FIFO_SIZE * 2 ) ; i++ )
        scope_data.data.single_channel[i]=128;

    return;
    }
```

8.1.14) DSO_IO.h

```
/* check if we have included this header files before */
#ifndef FILE_DSO_IO_HEADER
    /* define FILE_IO_HEADER so we dont include this file again */
    #define FILE_DSO_IO_HEADER

    /* include file statments */
    #include <windows.h>

    /* Check whether if this file is being used for io.c or another file */
    #ifdef FILE_DSO_IO
        /* File used for io.c */
        #define FILE_DSO_IO_EXTERN

        /* internal functions */
        static void dso_setup( void );
        static void dso_capture( void );
        static void get_dso_data( void );
        static long get_start_point( void );
        static void fifo_select( BOOL sel_fifo_a );
        static unsigned char get_reg( unsigned char reg_num );
        static unsigned char invert_byte( unsigned char input_byte );
    #else
        /* File used for other source file */
        #define FILE_DSO_IO_EXTERN extern
    #endif

    /* external and internal define statments */
    #define FIFO_SIZE            393216
    #define FIFO_RANGE           256
    #define MAX_PRE_TRIG         FIFO_SIZE-1
    #define TRIGGER_CHECK_TIME   250
    #define UPDATE_INTERVAL      4000
    #define CONTROL_BIDI_BIT     0x10
```

```
/* scope data structure */
struct scope_data_struct
    {
    /* pre trigger value, ie number of samples */
    unsigned long pre_trigger;
    /* select trig source           */
    /* 0 = ext trig                 */
    /* 1 = A, A&B, A or B           */
    /* 2 = B                        */
    /* 3 = None                     */
    char trig_sel;
    /* value used by trig reg       */
    unsigned short trig_value;
    /* trig on falling slop         */
    //BOOL trig_greater_than;
    BOOL trig_falling;
    /* valid values are 0-5         */
    /* representing 1-32 div on     */
    /* dso clock signal             */
    char clock_div;
    /* available scopy modes are    */
    /* 0 = normal dual channel mode */
    /* 1 = single channel interleve */
    /* 2 = 16bit logic analyser     */
    /* 3 = a=logic, b=analogue      */
    char scope_mode;
    /* combine all data structes so we dont use excess ram */
    union data_union
        {
        /* dual channel data structures */
        struct dual_channel_struct
            {
            unsigned char a[ FIFO_SIZE ];
            unsigned char b[ FIFO_SIZE ];
            } dual_channel;
```

```
    /* 16 bit logic analayser mode storage */
    short logic[ FIFO_SIZE ];
    /* interleave mode storage struct */
    unsigned char  single_channel[ FIFO_SIZE * 2 ];
    } data;
} scope_data;

/* convert struct used to access individual bits */
union convert_union
    {
    /* short access varaible                    */
    short logic;
    /* bit access structure                     */
    struct bits_struct
        {
        unsigned a : 1;
        unsigned b : 1;
        unsigned c : 1;
        unsigned d : 1;
        unsigned e : 1;
        unsigned f : 1;
        unsigned g : 1;
        unsigned h : 1;
        unsigned i : 1;
        unsigned j : 1;
        unsigned k : 1;
        unsigned l : 1;
        unsigned m : 1;
        unsigned n : 1;
        unsigned o : 1;
        unsigned p : 1;
        } bits;
    };
```

```
    /* internal and external function declirations */
    FILE_DSO_IO_EXTERN void swap_a_b( void );
    FILE_DSO_IO_EXTERN void copy_a_b( void );
    FILE_DSO_IO_EXTERN void copy_b_a( void );
    FILE_DSO_IO_EXTERN void cancel_capture( void );
    FILE_DSO_IO_EXTERN void clear_dso_data( void );
    FILE_DSO_IO_EXTERN void dso_reset_rd( void );
    FILE_DSO_IO_EXTERN void set_dso_control( unsigned char reg_num );
    FILE_DSO_IO_EXTERN void set_reg( unsigned char reg_num, unsigned char value );
    FILE_DSO_IO_EXTERN DWORD WINAPI capture_thread_func( LPVOID progress_window_handler );
    FILE_DSO_IO_EXTERN void get_fifo_data( unsigned char fifo_buf[], BOOL sel_fifo_a, BOOL show_progress );
#endif
```

8.1.15) P_Port_IO.c

```
/* local define statments */
#define FILE_P_PORT_IO
#define STATUS_XOR_MASK     0x80
#define STATUS_BIT_MASK     0xFC
#define CONTROL_XOR_MASK    0x0B
#define CONTROL_BIT_MASK    0x3F


/* local include files */
#include "p_port_io.h"
#include "define.h"
#include "error.h"
#include <stdio.h>
#include <conio.h>


/* file level veraibles */
static unsigned short port_address;


/* do we include all the I/O code or use dummy code */
#ifdef DISABLE_IO_CODE
    /* define varaibles to emulate regs             */
    static unsigned char data_reg, control_reg;

    /* compile dummy code only                      */
    void p_port_put_control( char control_data )
        {
        control_reg=control_data;
        return;
        }
```

```
    void p_port_put_data( char data )
        {
        data_reg=data;
        return;
        }
    unsigned char p_port_get_status( void )
        {
        return( 0 );
        }
    unsigned char p_port_get_data( void )
        {
        return( data_reg );
        }
    unsigned char p_port_get_control( void )
        {
        return( control_reg );
        }
    char p_port_setup( void )
        {
        return( 1 );
        }
#else
    /* compile the real I/O code                                                                    */

    /* Get status                                                                                   */
    unsigned char p_port_get_status( void )
        {
        /* read in the data from the port masking the reserved bits and inverting the HW inverted bits */
        return( ( STATUS_BIT_MASK & inp( (unsigned short)( port_address + 0x1 ) ) ) ^ STATUS_XOR_MASK );
        }
```

```
/* output control reg                                                                    */
void p_port_put_control( char control_data )
    {
    /* output the data to the port masking res bits and inverting HW inv bits           */
    outp( (unsigned short)( port_address + 0x2 ), ( control_data & CONTROL_BIT_MASK ) ^ CONTROL_XOR_MASK );

    /* return to parent func                                                            */
    return;
    }

/* get control reg                                                                       */
unsigned char p_port_get_control( void )
    {
    return( ( CONTROL_BIT_MASK & inp( (unsigned short)( port_address + 0x2 ) ) ) ^ CONTROL_XOR_MASK );
    }

/* output data reg                                                                       */
void p_port_put_data( char data )
    {
    /* output the data to the port                                                      */
    outp( port_address, data );

    /* return to parent func                                                            */
    return;
    }


/* input data reg                                                                        */
unsigned char p_port_get_data( void )
    {
    /* input the data from the DATA reg                                                 */
    return( inp( port_address ) );
    }
```

```
    /* init parallel port                                                          */
    char p_port_setup( void )
        {
        /* Setup Registers Ready For IO                                            */
        outp( (unsigned short)( port_address + 0x402 ), 0x20 );    /* Set ECR So We Are In BYTE mode   */

        /* test if the port is bidirectional or note                               */
        /* Basic Idea Is To Write Data Out To The DATA Port And See If The Read     */
        /* Value Matches What We Have Read Or The Instantanious Value On The Port  */
        p_port_put_control( 0x2B );         /* Try To Put The Port In Bi-Directional Mode              */
        p_port_put_data( (char) 0xAA );     /* Set DATA To 0xAA                                        */
        if( p_port_get_data()==0xAA )       /* Check That What Was Writen Matches What Was Read        */
            return( 0 );                    /* Writen Matches Read Therefore Its Not Bi-Directional    */
        else
            return( 1 );                    /* Writen Dosen't Match Read Therefore Its Bi-Directional  */
        }
#endif


/* Sets the base addesss of the parallel port                  */
void p_port_set_address( unsigned short base_address, char do_setup )
    {
    port_address=base_address;

    /* do the port setup, if we have been asked to             */
    if( do_setup == 1 )
        {
        if( p_port_setup() == 0 )
            {
            /* exit with error since port is not bi-directional */
            error( 10, "Main", NULL );
            }
        }

    return;
    }
```

```
/* returns the current address of the port */
unsigned short p_port_get_address( void )
    {
    return( port_address );
    }
```

8.1.16) P_Port_IO.h

```
/* check if we have included this header files before */
#ifndef FILE_P_PORT_IO_HEADER
    /* define FILE_P_PORT_IO_HEADER so we dont include this file again */
    #define FILE_P_PORT_IO_HEADER

    /* Check whether if this file is being used for source file or another file */
    #ifdef FILE_P_PORT_IO
        /* File used for source file */
        #define FILE_P_PORT_IO_EXTERN
    #else
        /* File used for other source file */
        #define FILE_P_PORT_IO_EXTERN extern
    #endif

    /* internal and external function declirations */
    FILE_P_PORT_IO_EXTERN unsigned char p_port_get_data( void );
    FILE_P_PORT_IO_EXTERN unsigned char p_port_get_status( void );
    FILE_P_PORT_IO_EXTERN unsigned char p_port_get_control( void );
    FILE_P_PORT_IO_EXTERN char p_port_setup( void );
    FILE_P_PORT_IO_EXTERN void p_port_put_data( char data );
    FILE_P_PORT_IO_EXTERN void p_port_put_control( char control_data );
    FILE_P_PORT_IO_EXTERN void p_port_set_address( unsigned short base_address, char do_setup );
    FILE_P_PORT_IO_EXTERN unsigned short p_port_get_address( void );
#endif
```

8.1.17) Define.h

```
/* check if we have included this header files before */
#ifndef FILE_DEFINE_HEADER
    /* define FILE_DEFINE_HEADER so we dont include this file again */
    #define FILE_DEFINE_HEADER

    /* comment out to enable I/O code */
    #define DISABLE_IO_CODE
    /* comment out to disable byte invert */
    //#define BYTE_INVERT

    /* version definition */
    #ifdef DISABLE_IO_CODE
        #define APP_VERSION "Version 2.08 (I/O Disabled)"
    #else
        #define APP_VERSION "Version 2.08 (I/O Enabled)"
    #endif

    /* Define statments */
    #define FILE_NAME_LEN           512
    #define FUNC_NAME_LEN           100
    #define ERROR_MESS_LEN          1000
    #define DEFAULT_P_PORT          0x378
    #define DSO_CLOCK_FREQ          50
    #define APP_NAME                "DSO Control Software"
    #define GUI_CLASS_NAME          "DSOSoftware"
    #define AUTHOR_MAIL_ADDRESS     "696GrocuttT@chocbar.demon.co.uk"
#endif
```

8.1.18) Resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by gui.rc
//
#define IDM_GUI_MENU                    104
#define IDI_DSO_ICON                    105
#define IDK_GUI_KEYS                    113
#define IDD_VIEW_RANGE                  114
#define IDD_ABOUT                       115
#define IDD_DSO_CAPTURE                 116
#define IDD_SET_PORT                    117
#define IDI_DSO_ICON_SMALL              121
#define IDD_PROGRESS                    124
#define IDD_VIEW_MODE                   125
#define IDD_HARDWARE_IO                 127
#define IDD_PREFS                       128
#define IDB_RISE                        132
#define IDB_FALL                        133
#define IDD_VIEW_RANGE_START            1001
#define IDD_VIEW_RANGE_STOP             1002
#define IDD_DSO_MODE_0                  1003
#define IDD_DSO_MODE_1                  1004
#define IDD_DSO_MODE_2                  1005
#define IDD_DSO_MODE_3                  1006
#define IDD_PORT_ADDRESS                1007
#define IDD_PRE_TRIG                    1009
#define IDD_EMAIL_ADDRESS               1014
#define IDD_APP_NAME                    1016
#define IDD_APP_VERSION                 1017
#define IDD_CLOCK_FREQ                  1023
#define IDD_TRIG_SOURCE_1               1032
#define IDD_TRIG_SOURCE_2               1033
#define IDD_TRIG_SOURCE_0               1034
#define IDD_TRIG_SOURCE_3               1035
#define IDD_PROGRESS_TEXT               1037
#define IDD_PROGRESS_PERSENTAGE         1038
#define IDD_REFRESH_DATA                1047
#define IDD_STATUS                      1049
#define IDD_CONTROL                     1051
#define IDD_DATA                        1052
#define IDD_REG_NUM                     1053
#define IDD_REG_VALUE                   1054
#define IDD_SET_REG                     1055
#define IDD_RESET_RD                    1056
#define IDD_TRIG_VALUE                  1057
#define IDD_INT_CTRL_REG                1057
#define IDD_TRIG_VALUE_MESSAGE          1058
#define IDD_GREATER_THAN                1059
#define IDD_CTRL_SET_REG                1059
#define IDD_GREATER_THAN_TEXT           1060
#define IDD_ZOOM_PAN                    1064
#define IDD_X_AXIS_TIME                 1065
#define IDD_TRIG_SLOPE_RISE             1066
#define IDD_TRIG_SLOPE_FALL             1067
#define IDD_SHOW_PRE_TRIG               1068
#define IDD_SHOW_TRIG_LEVEL             1069
```

```
#define IDM_EXIT                        40001
#define IDM_FILE_SAVE_AS                40002
#define IDM_FILE                        40003
#define IDM_DATA_CAPTURE                40005
#define IDM_FILE_OPEN                   40006
#define IDM_ABOUT                       40008
#define IDM_EXPORT_CSV                  40011
#define IDM_FILE_EXPORT                 40012
#define IDM_ZOOM_IN                     40013
#define IDM_ZOOM_OUT                    40014
#define IDM_PAN_RIGHT                   40015
#define IDM_PAN_LEFT                    40016
#define IDM_REFRESH                     40021
#define ID_VIEW                         40022
#define IDM_VIEW_RANGE                  40023
#define IDM_SET_PORT                    40025
#define IDM_VIEW_MODE                   40026
#define IDM_A_TO_B                      40027
#define IDM_B_TO_A                      40028
#define IDM_SWAP_AB                     40029
#define IDM_REREAD_A                    40030
#define IDM_REREAD_B                    40031
#define IDM_HW_IO                       40032
#define IDM_PREFS                       40033
#define IDM_OFFSET_CORRECT              40034

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NO_MFC                     1
#define _APS_NEXT_RESOURCE_VALUE        134
#define _APS_NEXT_COMMAND_VALUE         40035
#define _APS_NEXT_CONTROL_VALUE         1070
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

8.2) Pattern Generation Software

8.2.1) Pat.c

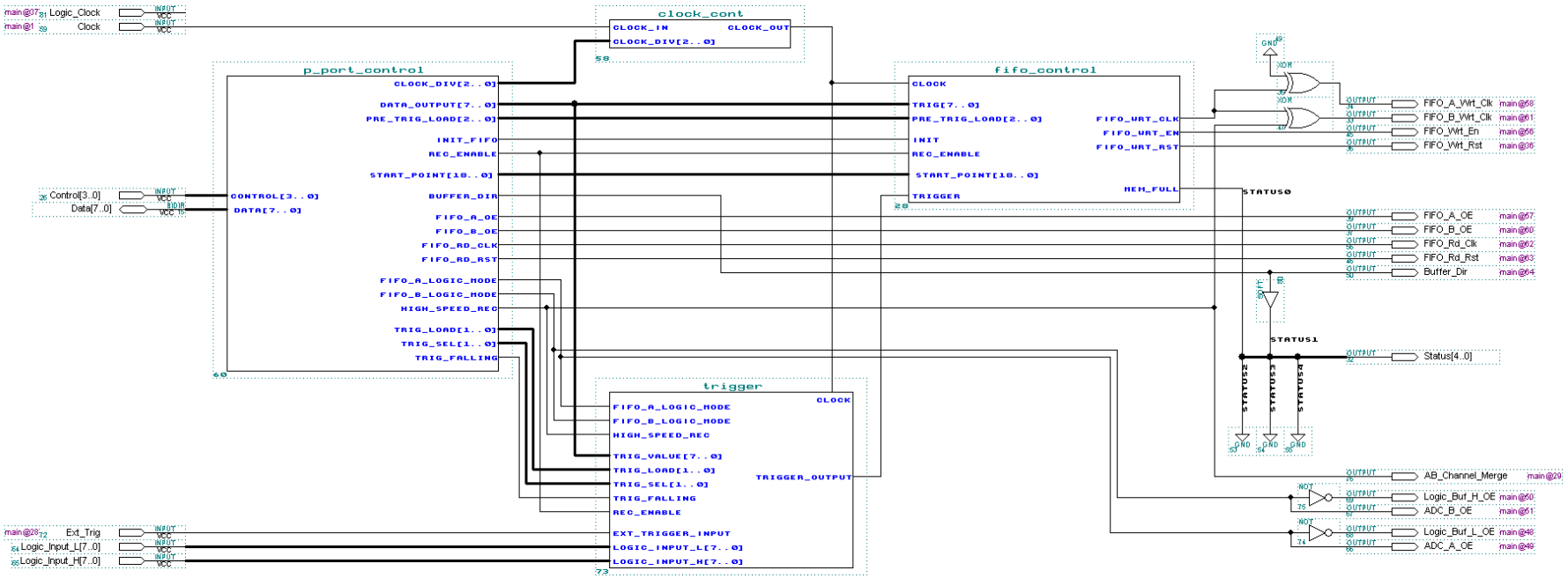
```
/* pattern generator */
/* by thomas grocutt */

/* include statements */
#include "c:\djgpp\include\stdio.h"
#include "c:\djgpp\include\conio.h"

/* define statments */
#define BASE_ADDRESS    0x3BC
#define DATA_REG        BASE_ADDRESS
#define CONTROL_REG     BASE_ADDRESS + 2
#define STATUS_REG      BASE_ADDRESS + 1

main()
    {
    unsigned char i,j;

    while(1)
        {
        /* trigger pulse        */
        outp( DATA_REG, 0x0  );
        outp( DATA_REG, 0xff );
        outp( DATA_REG, 0x0  );

        /* 7 bit count          */
        for( i=0 ; i<128 ; i++ )
            {
            outp( DATA_REG, i );
            }

        /* ripple               */
        for( i=1    ; i!=0x80 ; i=i<<1 ) outp( DATA_REG, i );
        for( i=0x80 ; i!=0x1  ; i=i>>1 ) outp( DATA_REG, i );
        }

    exit(0);
    }
```

8.2.2) Reset.c

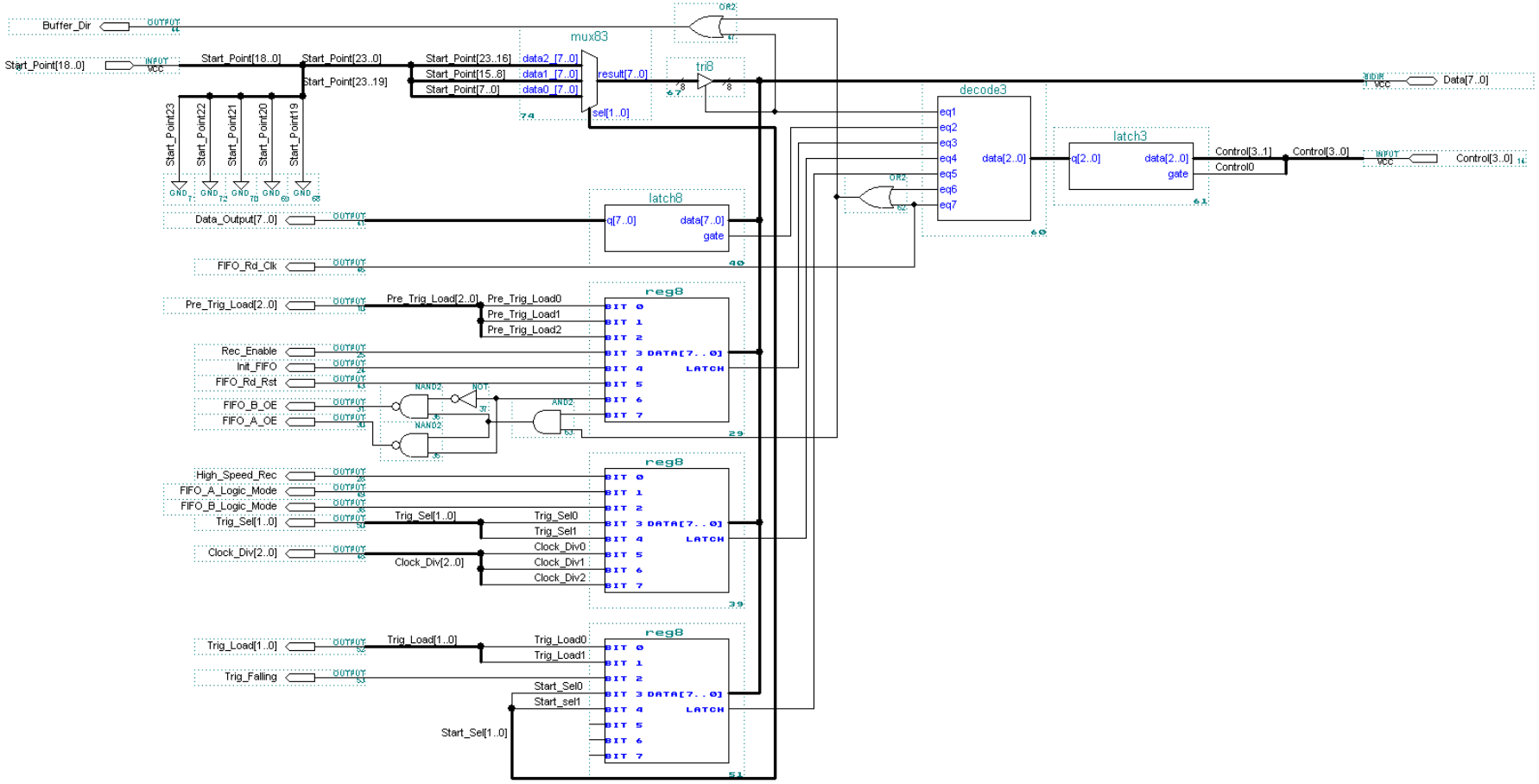
```
/* pattern generator */
/* by thomas grocutt */

/* include statements */
#include "c:\djgpp\include\stdio.h"
#include "c:\djgpp\include\conio.h"

/* define statments */
#define BASE_ADDRESS    0x3BC
#define DATA_REG        BASE_ADDRESS
#define CONTROL_REG     BASE_ADDRESS + 2
#define STATUS_REG      BASE_ADDRESS + 1

main()
    {
    outp( DATA_REG, 0x0 );

    exit(0);
    }
```

8.3) Altera Block Diagrams

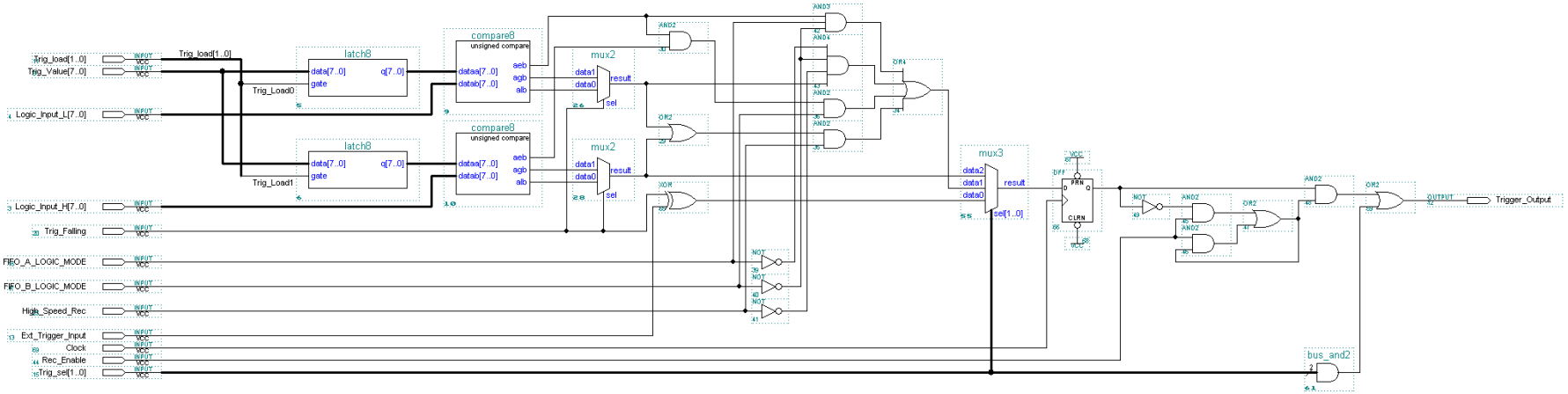
8.3.1) Functional Overview



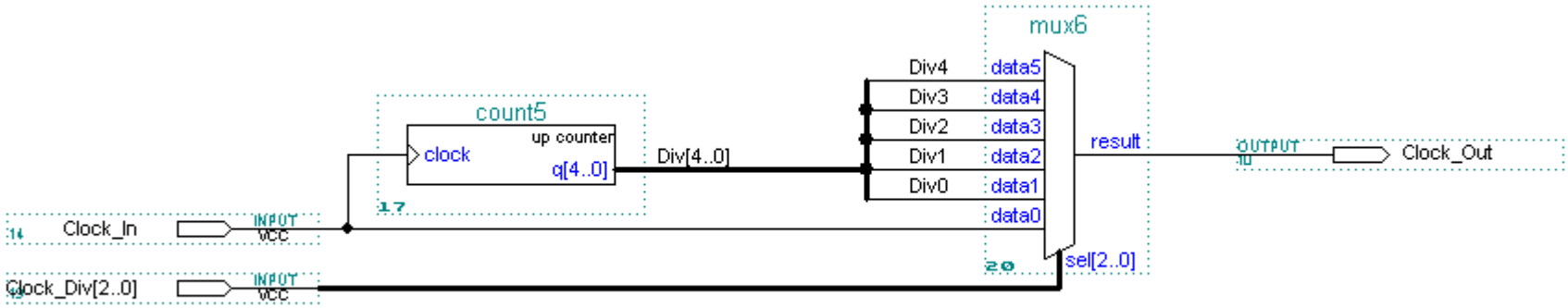
8.3.2) Paralell Port Controler



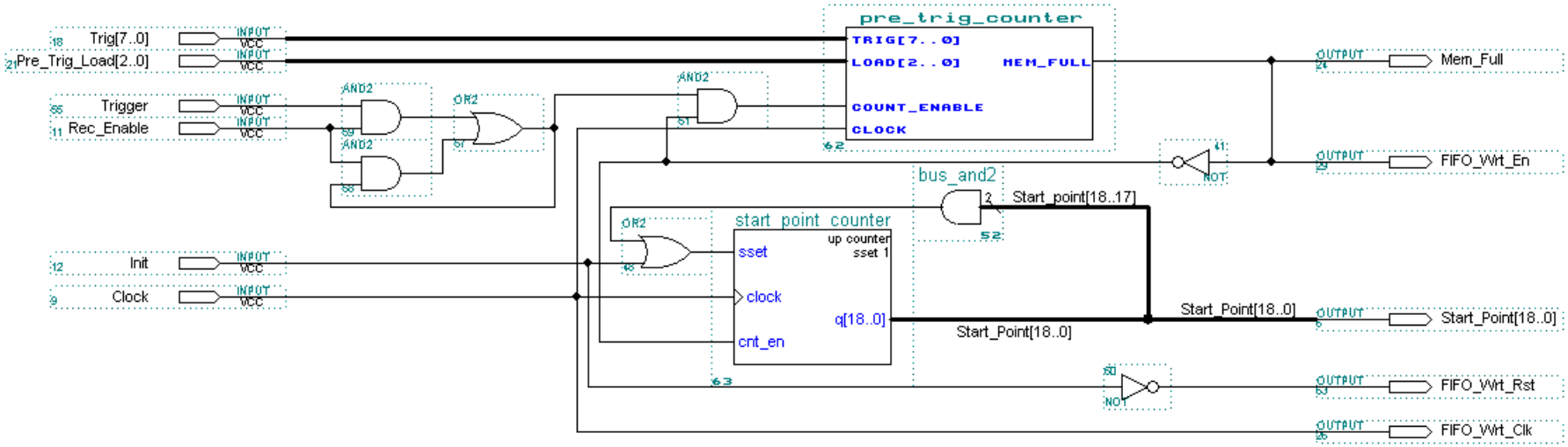
8.3.3) Trigger Unit



8.3.4) Clock Controler



8.3.5) FIFO Controler



8.3.6) Pre Trigger Counter