APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

ViSi Connecting the CC3000 to an AP

DOCUMENT DATE:	**28th May 2019**
DOCUMENT REVISION:	**1.1**

Description

This application note shows how to connect to an access point using the CC3000 WiFi module. This documentation is limited to initializing the CC3000 and connecting to a user defined access point.

Before getting started, the following are required:

- The target module can also be a Diablo16 display

  uLCD-35DT uLCD-70DT

  Visit www.4dsystems.com.au/products to see the latest display module products that use the Diablo16 processor.

- 4D Programming Cable or µUSB-PA5
- micro-SD (µSD) memory card
- Workshop 4 IDE (installed according to the installation document)
- When downloading an application note, a list of recommended application notes is shown. It is assumed that the user has read or has a working knowledge of the topics presented in these recommended application notes.
- CC3000 WiFi Shield

Content

Application Overview

Nowadays, wireless connections is one the most common mode of interconnectivity. One the most utilized mode of wireless connection is the internet connection. This application note show how to create the hardware connection from the display module to the cc3000 WiFi module via SPI connection.

Setup Procedure

For instructions on how to launch Workshop 4, how to open a **ViSi** project, and how to change the target display, kindly refer to the section "**Setup Procedure**" of the application note

ViSi Getting Started - First Project for Picaso and Diablo16

Create a New Project

For instructions on how to create a new **ViSi** project, please refer to the section "**Create a New Project**" of the application note

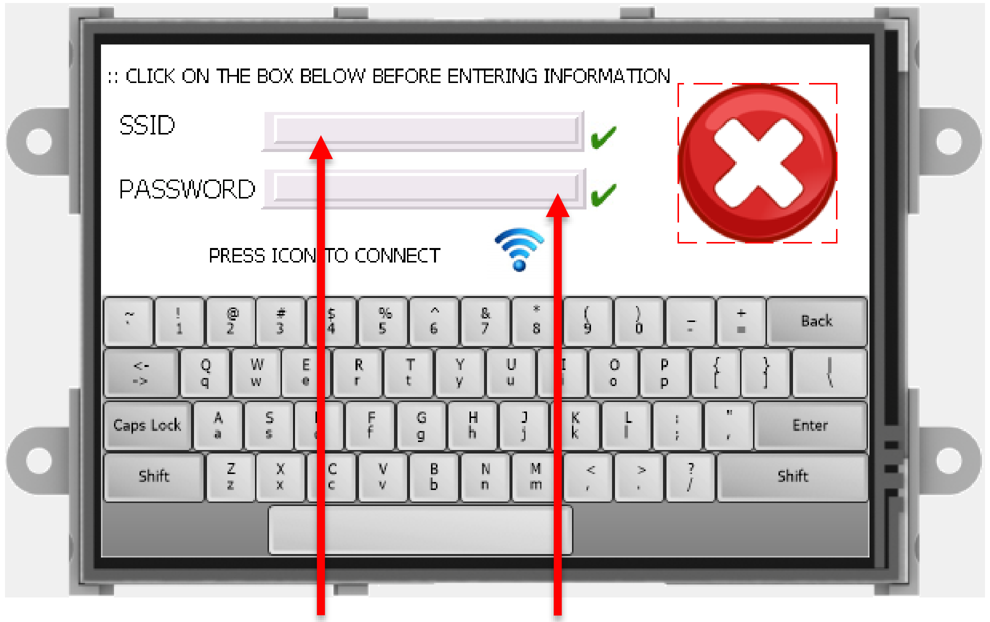
ViSi Getting Started - First Project for Picaso and Diablo16

Design of the Project

Layout of the Project

In this project couple of objects were used to provide a simple user interface form for the CC3000. The user interface accepts string inputs which are manually entered by the user through the keyboard object. The characters can be typed in the space provided in the project form.

ACCESS POINT SSID AND PASSKEY INPUT LOCATIONS



Spaces for the **ACCESS POINT SSID** AND **PASSKEY**.

CONNECTION ACCESS POINT STATUS ICONS

These icons show the status of connecting to the network.

NOT CONNECTED

CONNECTED

EXECUTE CONNECTION TO ACCESS POINT ICON

If the access point SSID and PASSKEY have all been entered as represented by the check mark beside the spaces. The next icon to consider is the connect command execute icon.

This icon contains the command that takes the input strings from the SSID and PASSKEY which was entered by the use and sends it to the CC3000 via the SPI port 2 of the Diablo 16.

** NOTE: A child program is run in flash bank 2. The program included in the zip file with this project.

The ViSi-based CC3000 Program

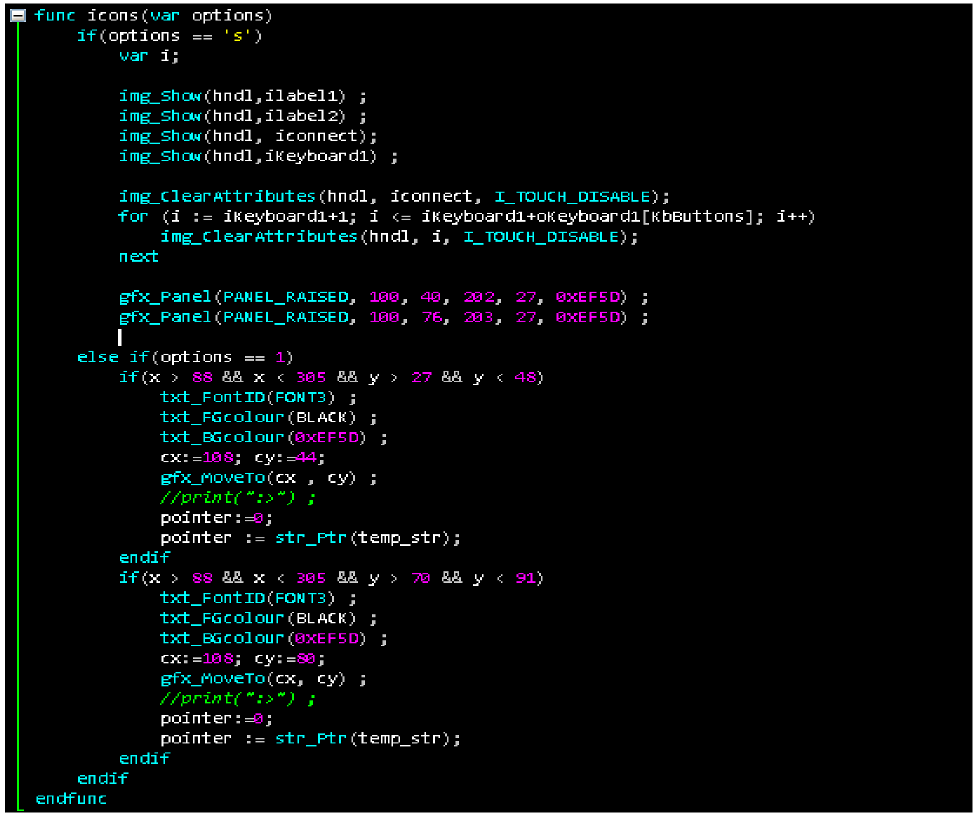
In this part of the application note statements that were written in 4DGL will be shown. These statements are used to produce the functionality desired for this project application. Each of the sub-routines has a group of statements that are intended for a designated purpose only.

The sub-routines

The statements presented here are called upon in the main function later in this document. It is best to present the purpose of each sub-routine for us to be able to easily understand the main program statements.

Icons() sub-routine

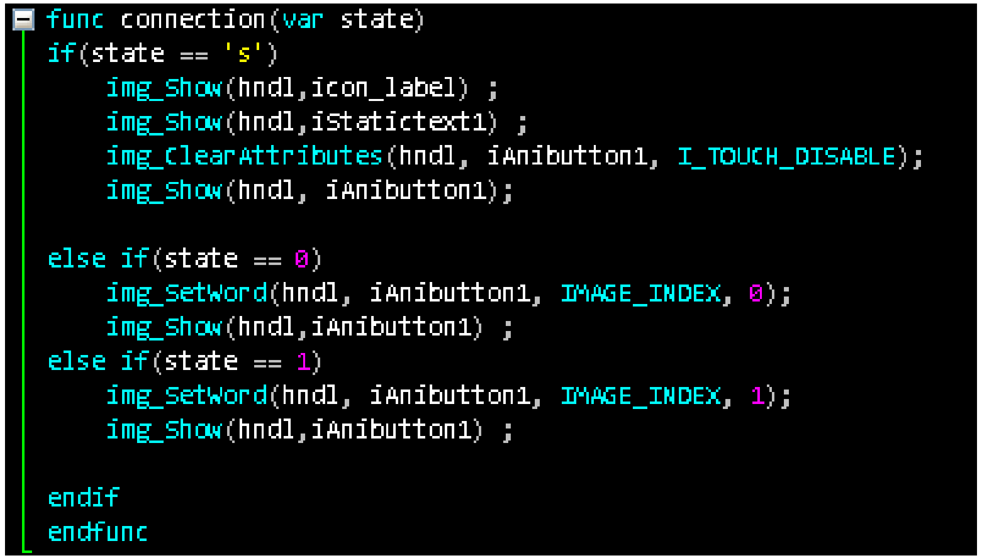
To initially display the ViSi project objects is the purpose of this sub-routine. It also contains the statements that clear the attributes for the objects there enabling the object to be valid for touch feature of the display.



```
func icons(var options)
    if(options == 's')
        var i;

        img_Show(hndl,ilabel1) ;
        img_Show(hndl,ilabel2) ;
        img_Show(hndl, iconnect);
        img_Show(hndl,iKeyboard1) ;

        img_ClearAttributes(hndl, iconnect, I_TOUCH_DISABLE);
        for (i := iKeyboard1+1; i <= iKeyboard1+oKeyboard1[KbButtons]; i++)
            img_ClearAttributes(hndl, i, I_TOUCH_DISABLE);
        next

        gfx_Panel(PANEL_RAISED, 100, 40, 202, 27, 0xEF5D) ;
        gfx_Panel(PANEL_RAISED, 100, 76, 203, 27, 0xEF5D) ;

    else if(options == 1)
        if(x > 88 && x < 305 && y > 27 && y < 48)
            txt_FontID(FONT3) ;
            txt_FGcolour(BLACK) ;
            txt_BGcolour(0xEF5D) ;
            cx:=108; cy:=44;
            gfx_MoveTo(cx , cy) ;
            //print(":>") ;
            pointer:=0;
            pointer := str_Ptr(temp_str);
        endif
        if(x > 88 && x < 305 && y > 70 && y < 91)
            txt_FontID(FONT3) ;
            txt_FGcolour(BLACK) ;
            txt_BGcolour(0xEF5D) ;
            cx:=108; cy:=80;
            gfx_MoveTo(cx, cy) ;
            //print(":>") ;
            pointer:=0;
            pointer := str_Ptr(temp_str);
        endif
    endif
endfunc
```

Connection(state) sub-routine

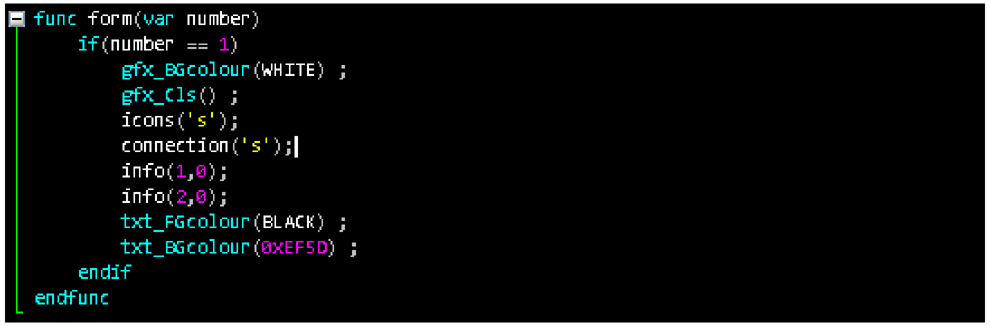
This sub-routine is made to initially display the animated button for the connection status indicator and also it is used to change the index of the animated button according to the status of the connection to an access point.



```
func connection(var state)
if(state == 's')
    img_Show(hndl,icon_label) ;
    img_Show(hndl,iStatictext1) ;
    img_ClearAttributes(hndl, iAnibutton1, I_TOUCH_DISABLE);
    img_Show(hndl, iAnibutton1);

else if(state == 0)
    img_SetWord(hndl, iAnibutton1, IMAGE_INDEX, 0);
    img_Show(hndl,iAnibutton1) ;
else if(state == 1)
    img_SetWord(hndl, iAnibutton1, IMAGE_INDEX, 1);
    img_Show(hndl,iAnibutton1) ;

endif
endfunc
```

form(number) sub-routine

This routine was written to simply the method of showing the objects in the main program it is simply a compilation of all the other individual components that forms the page displayed.



```
func form(var number)
    if(number == 1)
        gfx_BGcolour(WHITE) ;
        gfx_Cls() ;
        icons('s');
        connection('s');
        info(1,0);
        info(2,0);
        txt_FGcolour(BLACK) ;
        txt_BGcolour(0xEF5D) ;
    endif
endfunc
```

String_handle() sub-routine

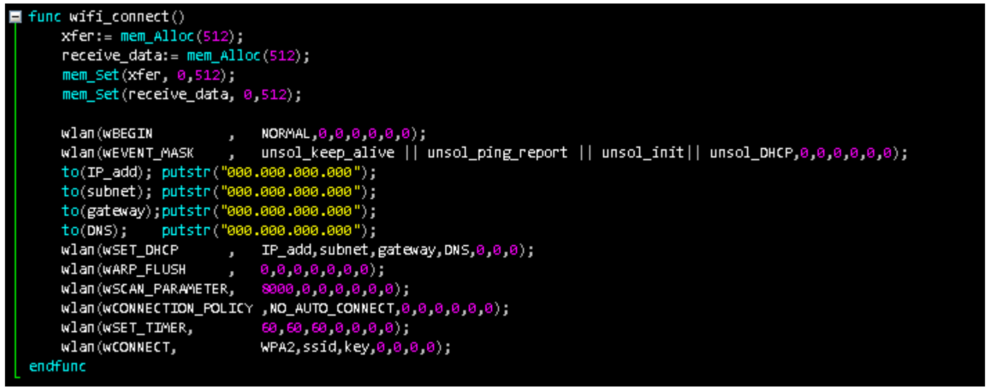
This sub-routine is dedicated to processing the output of the keyboard object. It has several statement that perform an equality comparison to identify the character that was pressed. Whenever a character is pressed this is added to the location of the global array pointer. The character is continually added to the string but the SSID and the PASSKEY can only be up to 32 characters.

In addition to this editing of the string can only be done while the user is currently typing the SSID or the password. Once the ENTER KEY is pressed the string is passed to their respective array containers. The data set to the SSID and passkey will be overwritten once the user presses the location of the spaces provided to display the inputs.

The content of the routine does not include detection of existing password or any history. The access point information is erased once the system restarts.

Wifi_connect() subroutine

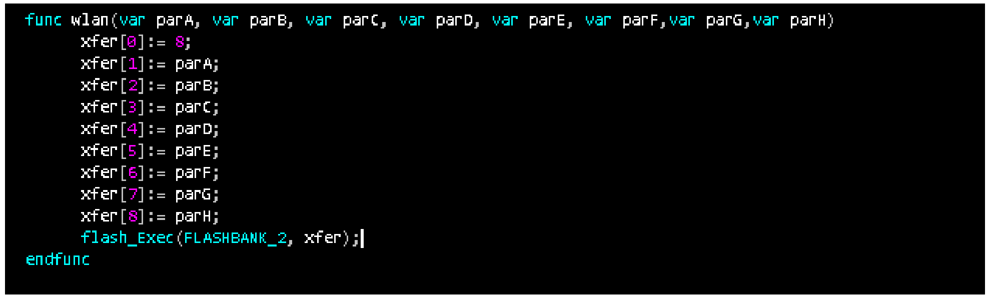
This sub-routine contains the command to transport data to flashbank_2. Notice that there is memory allocation in the statements. The memory set allows data to be saved to the location while a program is run in the other flash bank locations.



```
func wifi_connect()
    xfer:= mem_Alloc(512);
    receive_data:= mem_Alloc(512);
    mem_Set(xfer, 0,512);
    mem_Set(receive_data, 0,512);

    wlan(wBEGIN           ,   NORMAL,0,0,0,0,0,0);
    wlan(wEVENT_MASK      ,   unsol_keep_alive || unsol_ping_report || unsol_init|| unsol_DHCP,0,0,0,0,0,0);
    to(IP_add); putstr("000.000.000.000");
    to(subnet); putstr("000.000.000.000");
    to(gateway);putstr("000.000.000.000");
    to(DNS);    putstr("000.000.000.000");
    wlan(wSET_DHCP        ,   IP_add,subnet,gateway,DNS,0,0,0);
    wlan(wARP_FLUSH       ,   0,0,0,0,0,0,0);
    wlan(wSCAN_PARAMETER,     8000,0,0,0,0,0,0);
    wlan(wCONNECTION_POLICY ,NO_AUTO_CONNECT,0,0,0,0,0,0);
    wlan(wSET_TIMER,          60,60,60,0,0,0,0);
    wlan(wCONNECT,            WPA2,ssid,key,0,0,0,0);
endfunc
```

Wlan() subroutine

This sub routine is dedicated to transferring information in the form of individual data in an array. For this application 8 bytes that correspond to a particular value is written to flash bank 2.

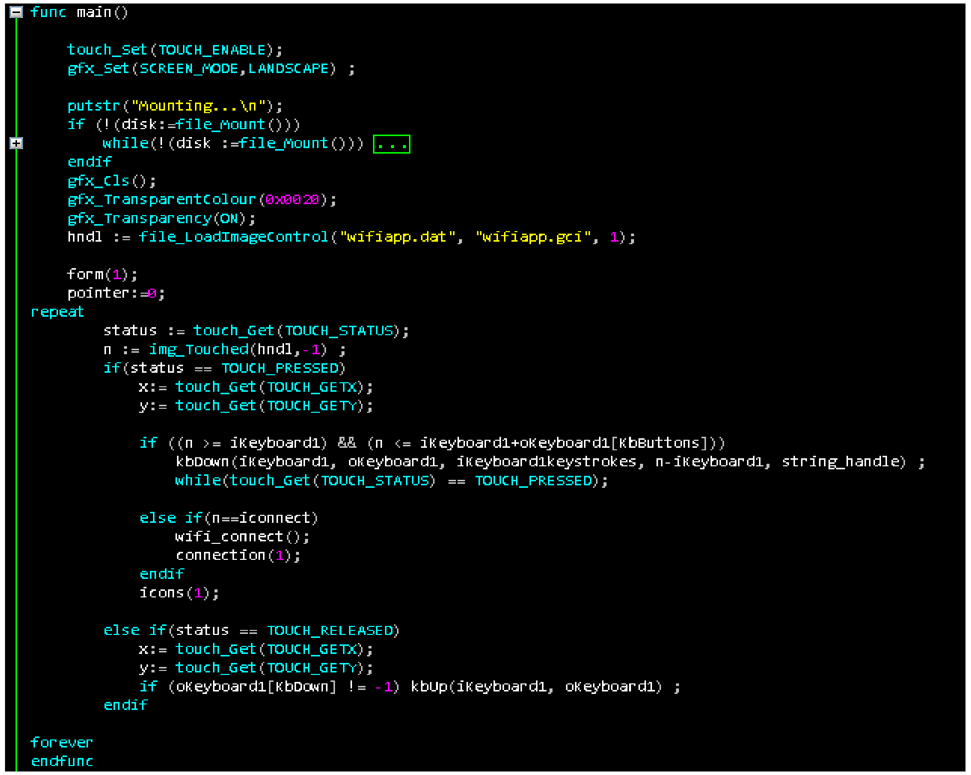


```
func wlan(var parA, var parB, var parC, var parD, var parE, var parF,var parG,var parH)
      xfer[0]:= 8;
      xfer[1]:= parA;
      xfer[2]:= parB;
      xfer[3]:= parC;
      xfer[4]:= parD;
      xfer[5]:= parE;
      xfer[6]:= parF;
      xfer[7]:= parG;
      xfer[8]:= parH;
      flash_Exec(FLASHBANK_2, xfer);
endfunc
```

The data transferred to flash bank 2 is translated by a child program which should be downloaded to the aforementioned flash bank. The content of the child program is a compilation of libraries to support the CC3000 WiFi Module. The child program is included in the project files.

The main() program

The statement included initializes the microSD and then accessing the GCI and DAT files from there on. The images contained in the GCI/DAT files are then read and displayed using the sub-routine calls. Finally a repetitive loop checks and monitors the touch related events.

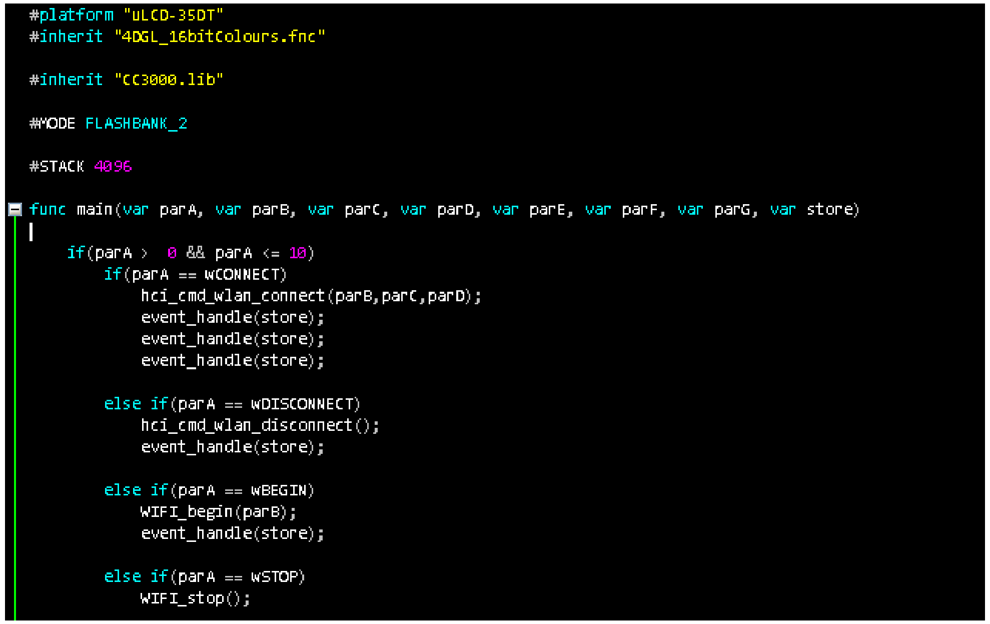


```
func main()

    touch_Set(TOUCH_ENABLE);
    gfx_Set(SCREEN_MODE,LANDSCAPE) ;

    putstr("Mounting...\n");
    if (!(disk:=file_Mount()))
        while(!(disk :=file_Mount())) ...
    endif
    gfx_Cls();
    gfx_TransparentColour(0x0020);
    gfx_Transparency(ON);
    hndl := file_LoadImageControl("wifiapp.dat", "wifiapp.gci", 1);

    form(1);
    pointer:=0;
repeat
        status := touch_Get(TOUCH_STATUS);
        n := img_Touched(hndl,-1) ;
        if(status == TOUCH_PRESSED)
            x:= touch_Get(TOUCH_GETX);
            y:= touch_Get(TOUCH_GETY);

            if ((n >= iKeyboard1) && (n <= iKeyboard1+oKeyboard1[KbButtons]))
                kbDown(iKeyboard1, oKeyboard1, iKeyboard1keystrokes, n-iKeyboard1, string_handle) ;
                while(touch_Get(TOUCH_STATUS) == TOUCH_PRESSED);

            else if(n==iconnect)
                wifi_connect();
                connection(1);
            endif
            icons(1);

        else if(status == TOUCH_RELEASED)
            x:= touch_Get(TOUCH_GETX);
            y:= touch_Get(TOUCH_GETY);
            if (oKeyboard1[KbDown] != -1) kbUp(iKeyboard1, oKeyboard1) ;
        endif

forever
endfunc
```

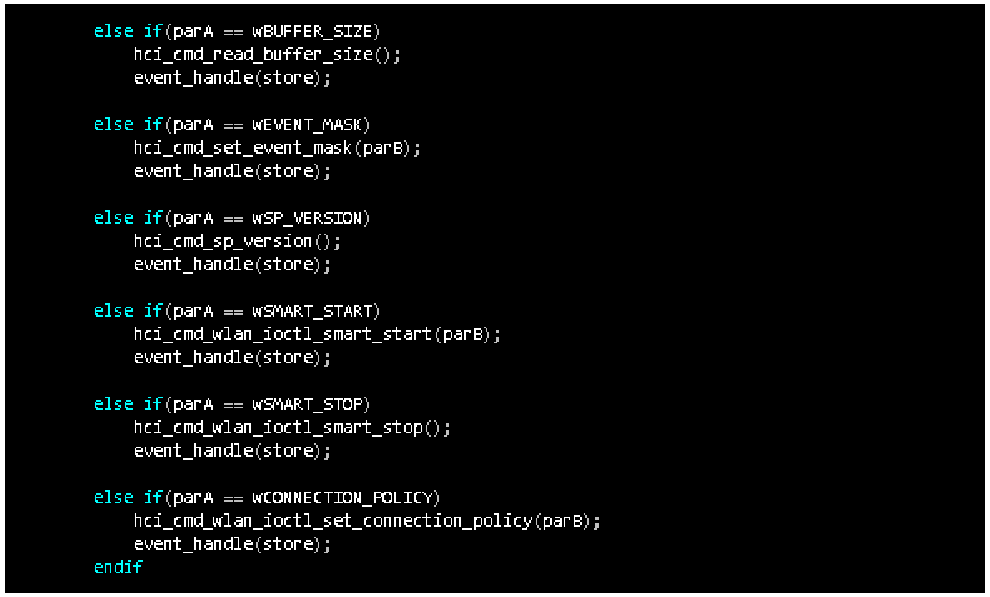
The WiFi support program on FLASHBANK_2

As mentioned from the previous part of this application note, a program is downloaded into the second flash bank location. The child program is named “SECONDARY”. THIS SHOULD BE DOWNLOADED TO THE DISPLAY TOGETHER WITH THE PROJECT DISCUSSED IN PRECEEDING SECTIONS.

The files contains host controller commands that are needed to instruct the CC3000. The commands are used in a blocking mode, meaning a certain set of time is allocated to await the response of the CC3000. The child program takes the values passed the main bank or flash bank 0 and these values are used in the commands in function main() of flash bank 2.



```
#platform "uLCD-35DT"
#inherit "4DGL_16bitColours.fnc"

#inherit "CC3000.lib"

#MODE FLASHBANK_2

#STACK 4096

func main(var parA, var parB, var parC, var parD, var parE, var parF, var parG, var store)

    if(parA >  0 && parA <= 10)
        if(parA == WCONNECT)
            hci_cmd_wlan_connect(parB,parC,parD);
            event_handle(store);
            event_handle(store);
            event_handle(store);

        else if(parA == WDISCONNECT)
            hci_cmd_wlan_disconnect();
            event_handle(store);

        else if(parA == WBEGIN)
            WIFI_begin(parB);
            event_handle(store);

        else if(parA == WSTOP)
            WIFI_stop();
```



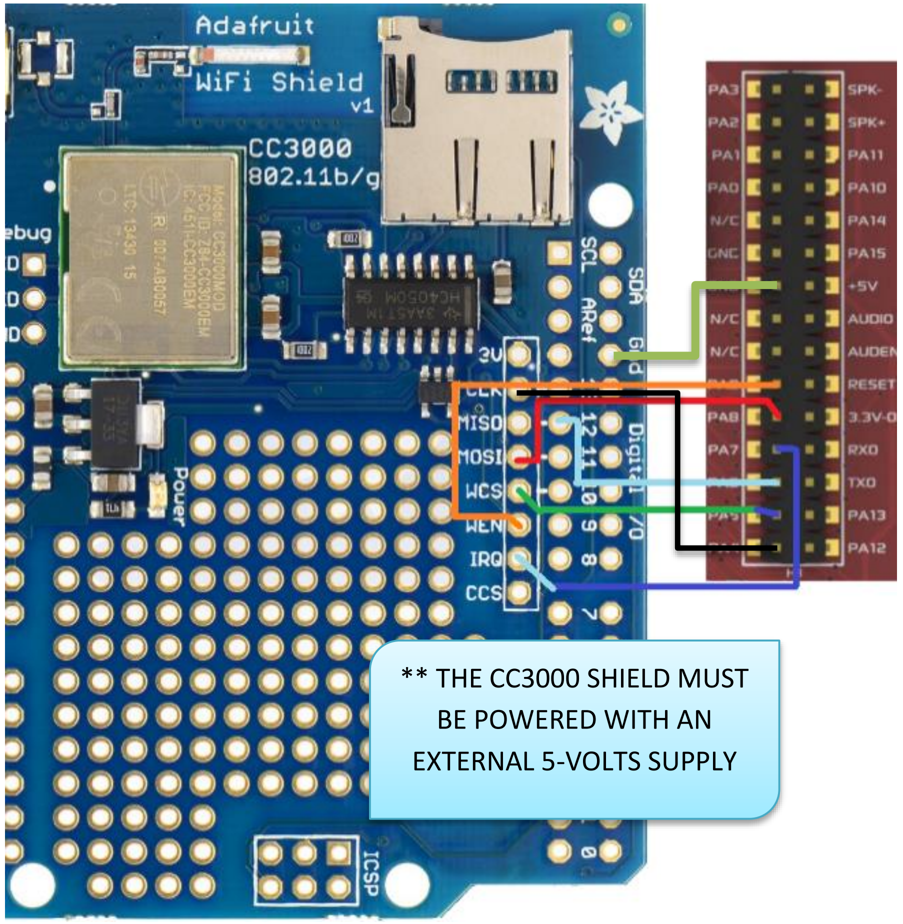
```
        else if(parA == WBUFFER_SIZE)
            hci_cmd_read_buffer_size();
            event_handle(store);

        else if(parA == WEVENT_MASK)
            hci_cmd_set_event_mask(parB);
            event_handle(store);

        else if(parA == WSP_VERSION)
            hci_cmd_sp_version();
            event_handle(store);

        else if(parA == WSMART_START)
            hci_cmd_wlan_ioctl_smart_start(parB);
            event_handle(store);

        else if(parA == WSMART_STOP)
            hci_cmd_wlan_ioctl_smart_stop();
            event_handle(store);

        else if(parA == WCONNECTION_POLICY)
            hci_cmd_wlan_ioctl_set_connection_policy(parB);
            event_handle(store);
        endif
```

From the previous function wlan(…), parameters are passed to the 2nd Flashbank. These parameters are the subjected to a series of if-else condition to identify which command is equivalent to parameters passed. The passed command is then executed while running in Flashbank 2. A special value is passed in the wlan function, this is the parameter ‘store’. The ‘store’ parameter identifies the start of the array where the result will be saved using the event_handle(store) function.

Included in the child program on Flashbank 2 is a long list of commands all dedicated to supporting the CC3000. This part of the project will be described on a separate application note to avoid confusion.

Connect the Adafruit CC3000 Shield to the uLCD-35DT

The image below shows a colour guided wiring connections for the CC3000 shield and uLCD-35DT.



Run the Program

For instructions on how to save a **ViSi** project, how to connect the target display to the PC, how to select the program destination, and how to compile and download a program, please refer to the section "**Run the Program**" of the application note

ViSi Getting Started - First Project for Picaso and Diablo16

Proprietary Information

Disclaimer of Warranties & Limitation of Liability