Chapter 4: Using the WICED-SDK Library

Objective

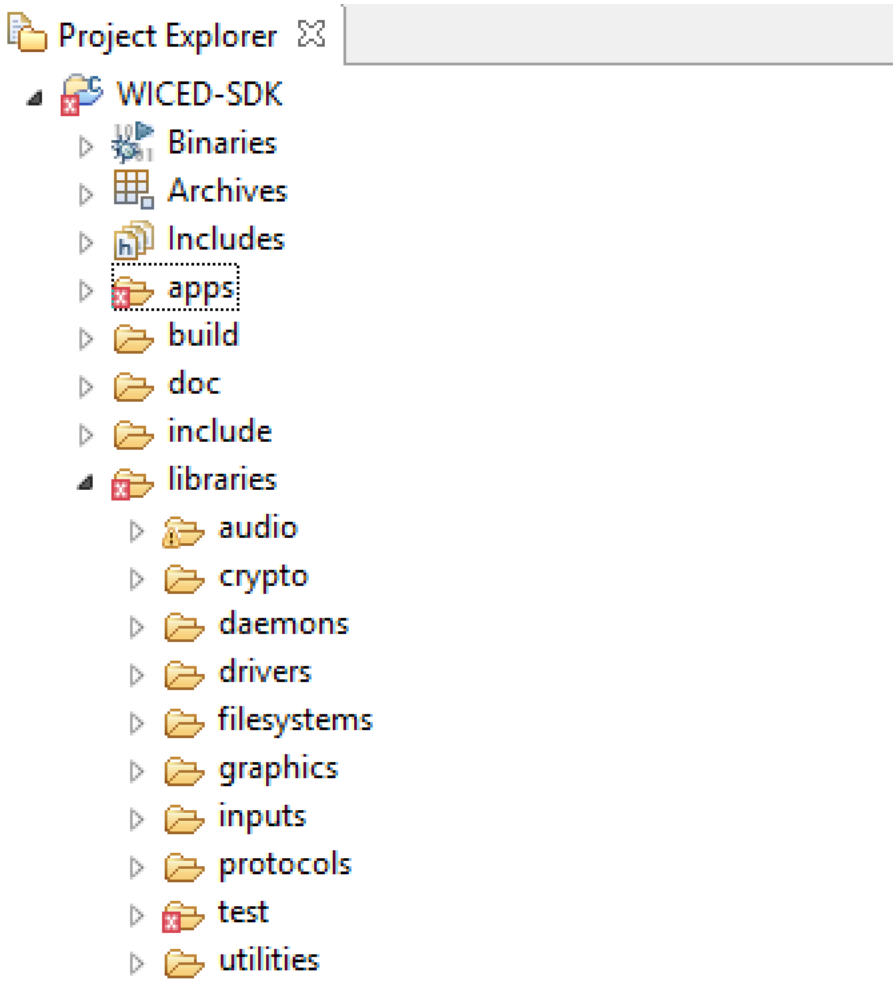
At the end of this chapter you should understand what is contained in the WICED-SDK library.

Time: 1 Hour

Fundamentals

WICED-SDK Library

At this point life is too short to develop all of the "stuff" that you might want to include in your IoT project. In order to accelerate your development cycle, the WICED SDK includes a bunch of code to handle many tasks that you might want to use in your design. If you look in the "libraries" folder in the SDK Workspace you will find the following sub-folders:



- **Audio:** Contains support for Apollo (a streaming audio standard), and codecs including Free Lossless Audio compression.
- **Crypto:** ?
- **Daemons:** Contains some typical "Unix" daemons to provide networking support including an HTTP Server, Gedday, TFTP, DHCP, DNS etc.
- **Drivers:** Contains hardware support files for SPI flash, USB etc.
- **Filesystems:** FAT, FILE and other file systems that could be written to an SPI flash.
- **Graphics:** Support for the U8G OLED displays.
- **Inputs:** Drivers for buttons and GPIOs.
- **Protocols**: Support for application layer protocols including HTTP, COAP, MQTT etc.
- **Test**: Tools to test network performance, iPerf, malloc, TraceX, audio.
- **Utilities**: Support for JSON, linked lists, console, printf, buffers, etc.
    - **Note**: There are 2 JSON parsers: cJSON and JSON_parser. The cJSON functions are simpler to use and a README file and examples are provided.

In the exercises, we will be using the graphics library to display information on the OLED display present on the shield board.

Note: <u>If the display is currently showing information from the PSoC analog co-processor, you must hold down button MB0 until the display clears (5-10 seconds). This will turn off the I2C master in the PSoC analog co-processor so that you can control the OLED from the WICED base board.</u>

In order to draw text to the display you must:

1. Setup a structure of type *u8g_t*. A pointer to this structure will be the first argument in almost all of the u8g function calls that we use.
2. Setup and initialize an I2C structure for the OLED display. For our hardware:
    a. I2C port = WICED_I2C_2
    b. I2C address = 0x3C
    c. I2C address with = I2C_ADDRESS_WIDTH_7BIT
    d. Flags = 0
    e. Speed mode = I2C_STANDART_MODE
3. Initialize the I2C device using u8g_init_wiced_i2c_device. This function takes a pointer to the I2C structure from step 2.
4. Initialize the communication functions by calling *u8g_InitComFn*. It takes a pointer to the u8g structure created in step 1, a pointer to a u8g_dev_t structure which specifies the type of display, and a communication function pointer. For our hardware, if you have a display structure called "display", the call looks like this:

    u8g_InitComFn(&display, &u8g_dev_ssd1306_128x64_i2c, u8g_com_hw_i2c_fn);

5. Select a font using *u8g_SetFont*. It takes a pointer to the u8g structure and the name of the font.
    a. The fonts are all listed in the file u8g_font_data.c in the graphics library directory. The examples use u8g_font_unifont, but feel free to experiment with others if you want.
6. Set a position using *u8g_SetFontPosTop*, *u8g_SetFontPosBottom*, or *u8g_SetFontPosCenter*.

a. These functions determine where the characters are drawn relative to the starting coordinates specified in the DrawStr function described below. *u8g_SetFontPosTop* means that the top of the first character will be at the coordinate specified.

7. Each time you want to display a string you:
   a. Select the page to display the string using *u8g_FirstPage*.
   b. Draw the string using *u8g_DrawStr*. You must call this repeatedly until *u8g_NextPage* returns a 0. The u8g_DrawStr function takes a pointer to the u8g structure, X coordinate, Y coordinate, and the string to be printed.

As an example, assuming a display structure called "display" and an I2C structure called "display_i2c" the following will print the string "Cypress":

```
u8g_init_wiced_i2c_device(&display_i2c);
u8g_InitComFn(&display, &u8g_dev_ssd1306_128x64_i2c, u8g_com_hw_i2c_fn);
u8g_SetFont(&display, u8g_font_unifont);
u8g_SetFontPosTop(&display);

u8g_FirstPage(&display);
do
{
    u8g_DrawStr(&display, 0, 10, "Cypress");
} while (u8g_NextPage(&display));
```

In addition, you must include "u8g_arm.h" in the .c file and you must include the u8g library in the .mk file to have access to the library functions:

In <project>.c : #include u8g_arm.h

In <project>.mk: $(NAME)_COMPONENTS := graphics/u8g

Note: u8g_arm.h includes wiced.h so you don't need to include wiced.h separately.

Exercise(s)

01 Browse the library directory to see what functions are available

02 Review the graphics library documentation and run the examples

1. Go to the documentation directory in the SDK (43xxx_Wi-Fi/doc) and open the WICED-LED_Display.pdf file. Review the documentation.
2. Copy the project from snip/graphics/hello to ww101/04/02_hello. Rename files and update the make file as necessary.
3. Verify that the I2C port is set to WICED_I2C_2.
4. Update the I2C speed_mode to I2C_STANDARD_SPEED_MODE.
5. Review the rest of the project to understand what it is doing.
6. Create a make target for your project and run it.
7. Repeat the above steps for the graphicstest project.
    a. Hint: you will have to remove the VALID_PLATFORMS line from the make file (or add CYW943907*) in order to build the project.

03 (Advanced) Display sensor information on the OLED display

1. Copy 02_hello to 03_sensorData. Update the names and make target as necessary.
2. Update the code so that the temperature, humidity, ambient light, and potentiometer values are read from the analog co-processor and displayed to the screen every ½ second.
    a. Hint: see the I2C read exercise in chapter 2 for information on reading the sensor values using I2C.
    b. Hint: you will need to create two different I2C structures and initialize two I2C devices – one for the analog co-processor and one for the OLED display. They will both use the same physical interface (WICED_I2C_2) but not at the same time.
    c. Hint: use *snprintf* to format the strings. This safer than sprint because you tell it the max number of characters to output – there is no chance of over-running the buffer which can cause all sorts of odd behavior. The prototype is:

        int snprintf(char *buffer, size_t n, const char *format-string, argument-list);

        Note that the string produced includes a terminating null character so the size parameter must be large enough to hold the string plus the terminating null.

    d. Hint: If you are using threads, this would be a great place to use a mutex.