Internet of Things
Department of Computer Engineering
Santa Clara University, California
Behnam Dezfouli, PhD

Santa Clara University
Internet of Things
RESEARCH LAB
SIOTLAB

Internet of Things

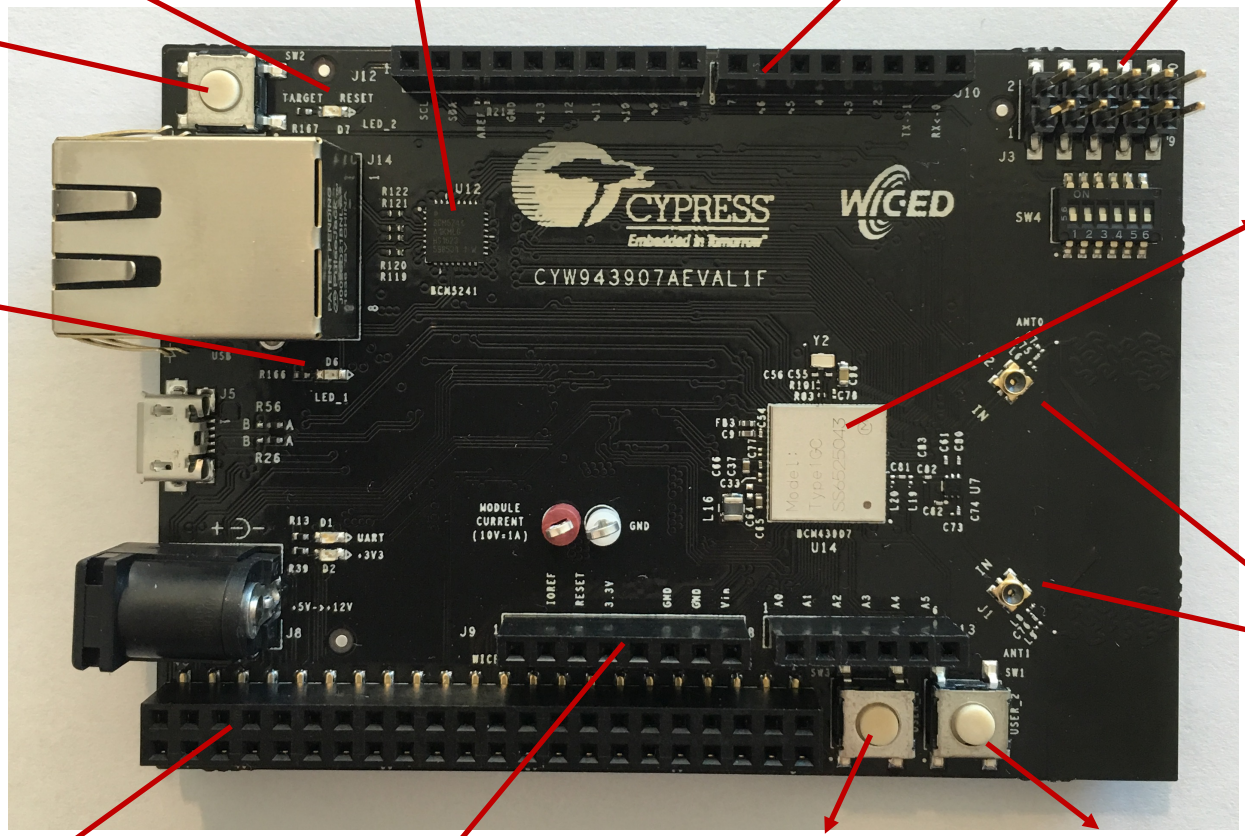
Lab 5
Getting Started with RTOS

Hardware Platform

Base Board and Shield Board

Hardware Platform

Cypress CYW943907



External JTAG Header

BCM5241 Ethernet transceiver

Arduino headers

LED 2

Reset

Cypress CYW43907

LED 1

WiFi antenna

WICED header

Arduino headers

User Switch 1 SW3

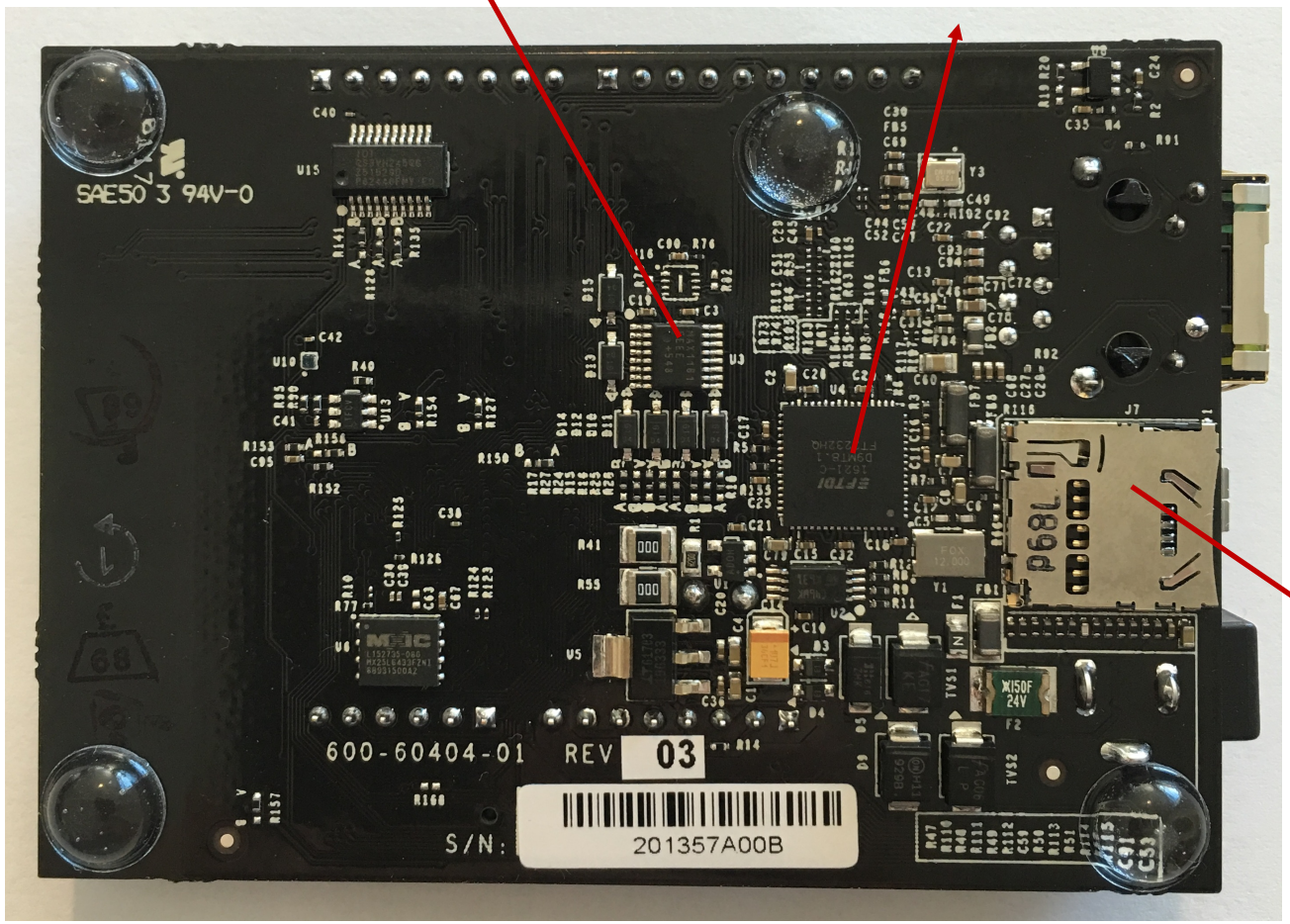
User Switch 2 SW1

Hardware Platform

Cypress CYW943907

MAX11615 ADC
Helps to achieve Analog functionality on the Arduino headers

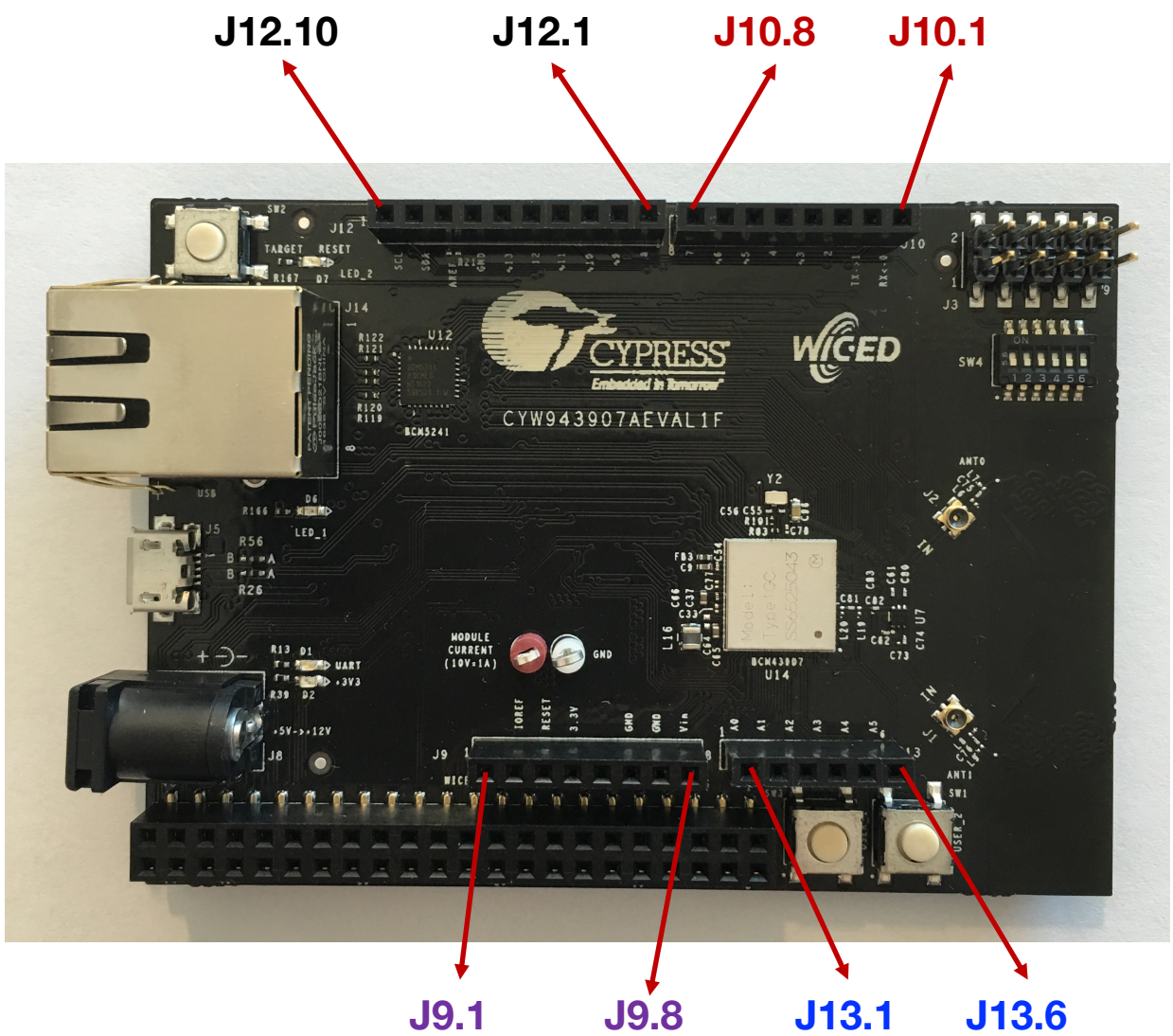
- **FT2232H is FTDI's 5th generation of USB devices**
- The FT2232H is a USB 2.0 Hi-Speed (480Mb/s) to UART/FIFO IC



SD Card

Hardware Platform

Cypress CYW943907



Hardware Platform

Cypress CYW943907

Eval Board Header	CYW43907 Pin Name/ Kit Signal Name	ARDUINO Header Name	WICED Enumeration	Alternate Enumeration
J10.1	GPIO_0	D0	WICED_GPIO_1	N/A
J10.2	GPIO_1	D1	WICED_GPIO_2	N/A
J10.3	GPIO_13	D2	WICED_GPIO_9	N/A
J10.4	GPIO_7	D3	WICED_GPIO_3	WICED_PWM_6
J10.5	GPIO_14	D4	WICED_GPIO_10	N/A
J10.6	GPIO_16	D5	WICED_GPIO_12	WICED_PWM_3
J10.7	GPIO_15	D6	WICED_GPIO_11	WICED_PWM_4
J10.8	I2S0_SD_IN	D7	WICED_GPIO_31	WICED_I2S_1
J12.1	I2S1_SD_IN	D8	WICED_GPIO_36	WICED_I2S_3
J12.2	PWM_4	D9	WICED_GPIO_17	WICED_PWM_5
J12.3	GPIO_11	D10	WICED_GPIO_7	WICED_PWM_2
J12.4	GPIO_10	D11	WICED_GPIO_6	WICED_PWM_1
J12.5	GPIO_12	D12	WICED_GPIO_8	N/A
J12.6	GPIO_9	D13	WICED_GPIO_5	WICED_LED2
J12.7	GND	GND	N/A	N/A
J12.8	ARD_AREF	AREF	N/A	N/A

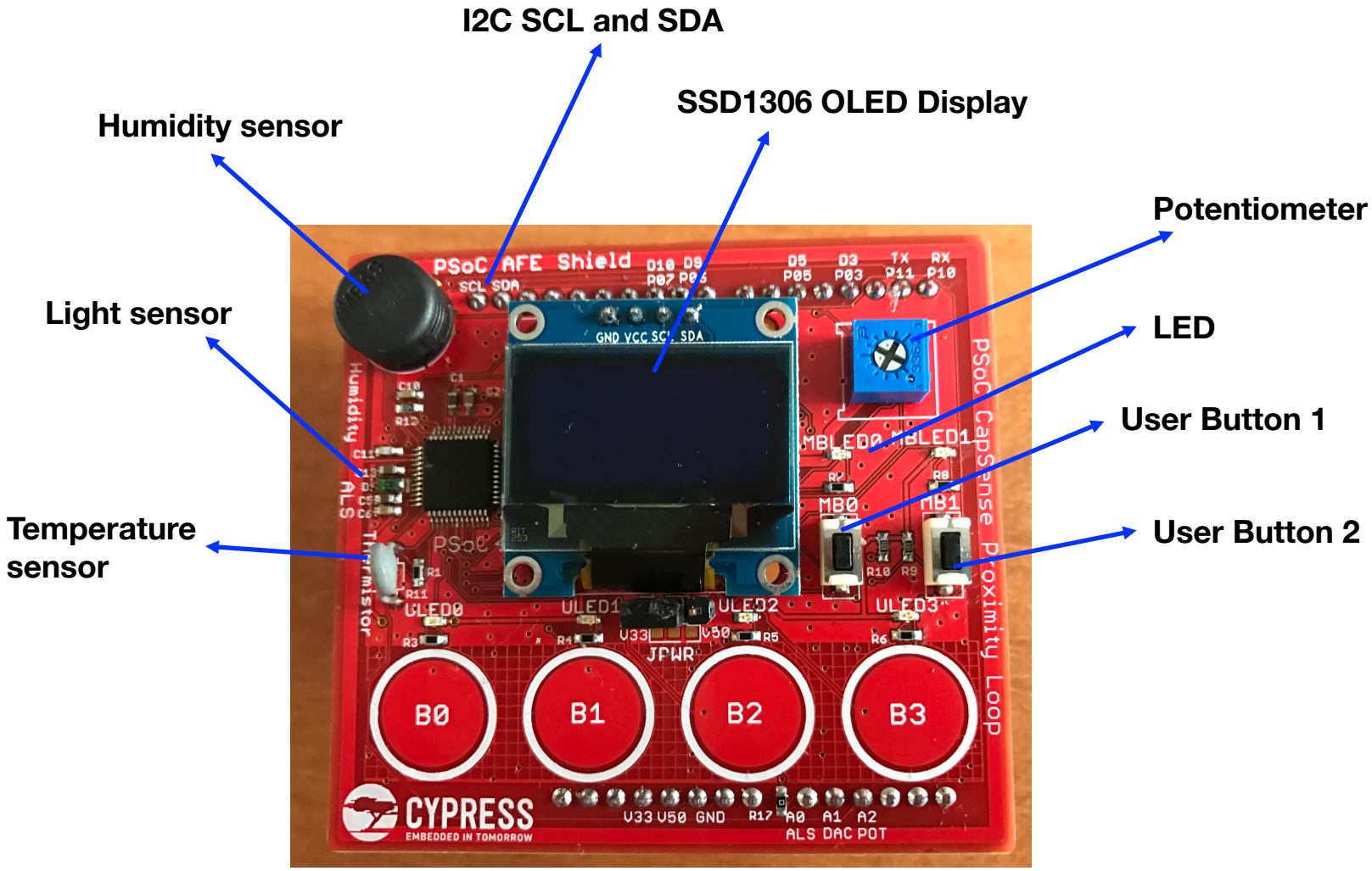
Hardware Platform

Cypress CYW943907

Eval Board Header	CYW43907 Pin Name/ Kit Signal Name	ARDUINO Header Name	WICED Enumeration	Alternate Enumeration
J12.9	I2C_1_SDA	SDA	WICED_GPIO_50	WICED_I2C_2
J12.10	I2C_1_SCL	SCL	WICED_GPIO_51	WICED_I2C_2
J13.1	ARD_AD0	A0	N/A	N/A
J13.2	ARD_AD1	A1	N/A	N/A
J13.3	ARD_AD2	A2	N/A	N/A
J13.4	ARD_AD3	A3	N/A	N/A
J13.5	ARD_AD4_SDA	A4	N/A	N/A
J13.6	ARD_AD5_SCL	A5	N/A	N/A
J9.1	NC	NC	N/A	N/A
J9.2	ARD_IOREF	IOREF	N/A	N/A
J9.3	ARD_RESET	RESET	N/A	N/A
J9.4	3V3	3.3V	N/A	N/A
J9.5	NC	5V	N/A	N/A
J9.6	GND	GND	N/A	N/A
J9.7	GND	GND	N/A	N/A
J9.8	VIN_EXT	VIN	N/A	N/A

Hardware Platform

Shield Board



Software Development using WICED® Studio

Understanding the Environment

Behnam Dezfouli | Santa Clara University | California

Installation

- The WICED software tool is called "**WICED Studio**" and it is based on Eclipse

You can run this tool on ECC Linux machines using the following commands:

- `setup WICED`
- `WICED-Studio`

- Once the program is loaded, you need to:
  - Double click the menu entry for **43xxx_wi-fi** on the left panel in WICED studio to select the platform

SDK Structure

- WICED Studio supports multiple RTOSs
- ThreadX (by Express Logic) is built into the device ROM and the license is included for anyone using WICED chips so that is by far the best choice
- **It also supports FreeRTOS**
- In order to simplify using multiple RTOSs, **the WICED SDK has a built in abstraction layer that provides a unified interface to the fundamental RTOS functions**

- Find the **documentation** for the WICED RTOS APIs under:

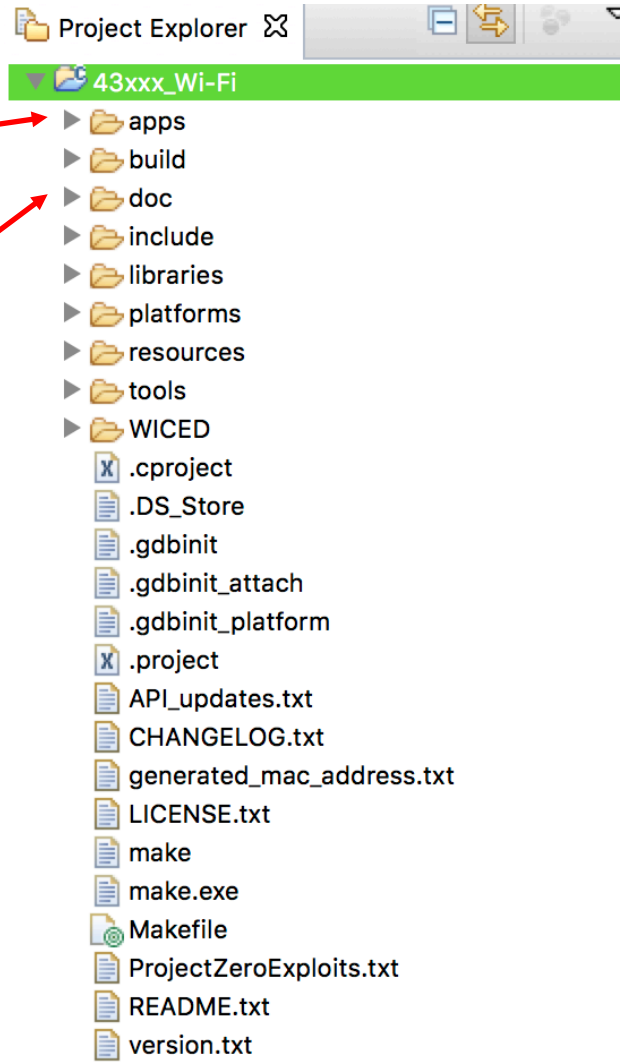
  Cypress WICED API Reference -> Components -> RTOS

- The WICED RTOSs provide important OS mechanisms such as mutexes, semaphores, queues and timers

SDK Structure

Your APPs

Documentation



SDK Structure

`apps`: where all of the **example projects** reside, as well as where you will put your own projects

- **`snip`**: These are short examples that typically demonstrate one feature
  - ❑ For example:
    - **`snip/gpio`** demonstrates GPIO use by reading buttons and blinking LEDs
    - **`snip/scan`** scans for Wi-Fi access points every 5 seconds and displays the results to a terminal window
- **`demo`**: These are more complex and complete demonstrations
  - ❑ For example:
    - **`demo/temp_control`** demonstrates an application for controlling and reporting temperatures
    - **`demo/bt_smartbridge`** demonstrates a Bluetooth to Wi-Fi bridge
- **`test`**: These are test and utility programs such as a console that allows you to scan for and connect to Wi-Fi access points
  - ❑ For example:
    - **`test/console`** provides a console application on a terminal window Type “help” in the console for a list of all supported commands

SDK Structure

`doc`: The doc folder contains the documentation for the SDK Workspace

- Of particular interest is the API html file which documents all of the WICED API functions
- Also contains other useful documents such as the QSG (Quick Start Guide), how to use DCT (Device Configuration Tables), FLAC (Free Lossless Audio Compression), and OTA (Over the Air) updates

  https://community.cypress.com/welcome

`platform`: contains information on different kits (i.e. hardware platforms)

- These files are necessary in order to program a given project into specific hardware

SDK Structure

- In our case, **the kit we are using is called *CYW943907AEVAL1F***
- This kit has a platform folder, but since we are also using a shield attached to it, we will use a custom set of platform files (***CYW943907AEVAL1F_WW101***) that also includes the peripherals on the shield

- **You will have to copy over the custom platform files before using the shield and kit**

- Use git and clone the following repository:

  `https://github.com/SIOTLAB/cypress_cyw43907_shield.git`

- **Copy the entire folder "`CYW943907AEVAL1F_WW101`" from the class materials into the "platforms" directory in the `43xxx-wi-fi` folder**

SDK Structure

`libraries`: contains various sets of library function files

- **`audio`:** Contains support for Apollo (a streaming audio standard), and codecs including Free Lossless Audio compression.
- **`crypto`:** Elliptic curve cryptography (ECC)
- **`daemons`:** Contains some typical "Unix" daemons to provide networking support including an HTTP Server, 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, console, printf, buffers, etc.

`resources`: where you store files that are required by your application

- For example, if your application contains a web server, the html files for the server would be in the resources folder under `apps/https_server`

SDK Structure

- The **bases board** contains two LEDs and two mechanical buttons

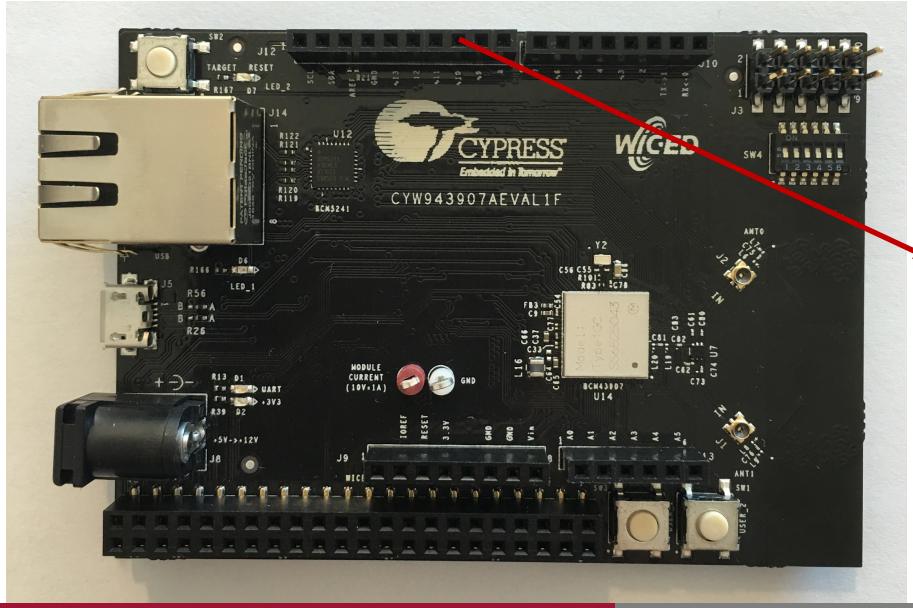
```
/* LEDs and buttons on the base board */
#define WICED_LED1        ( WICED_GPIO_16 )
#define WICED_LED2        ( WICED_GPIO_5 )
#define WICED_BUTTON1     ( WICED_GPIO_18 )
#define WICED_BUTTON2     ( WICED_GPIO_4 )
```

Switch	CYW43907 Pin Name	WICED_ENUM_ID	Alternate Enumeration in WICED
LED_1	PWM_3	WICED_GPIO_16	WICED_LED1
LED_2	GPIO_9	WICED_GPIO_5	WICED_LED2

SDK Structure

- The **shield board** contains two LEDs and two mechanical buttons

```
/* LEDs and buttons on the shield */
#define WICED_SH_LED0                    ( WICED_GPIO_7 )
#define WICED_SH_LED1                    ( WICED_GPIO_17 )
#define WICED_SH_LED0_ON_STATE           ( WICED_ACTIVE_HIGH )
#define WICED_SH_LED1_ON_STATE           ( WICED_ACTIVE_HIGH )
#define WICED_SH_MB0                     ( WICED_GPIO_12 )
#define WICED_SH_MB1                     ( WICED_GPIO_3 )
```

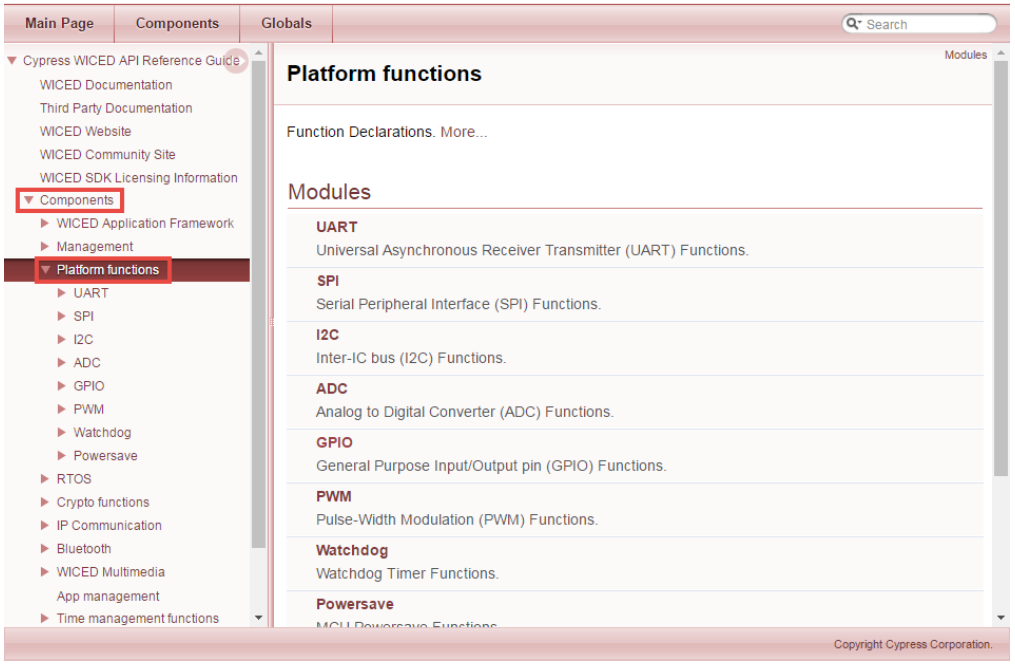


J12.3 = **WICED_SH_LED0** = WICED_GPIO_7 = D10 (Arduino) = GPIO_11 (CYW43907)

Documentation

- Documentation can be found in the SDK Workspace **doc** folder
- The file **API.html** contains the documentation of the APIs that we will be using

✓ Open this file by right-clicking on it and selecting *Open With* > *System Editor* and then expand "Components" and "Platform Functions" to see the list of supported components



SDK Structure

- The `platform.c` file contains several constant arrays and structures that are used to configure the peripherals
- These files map the chip pins to WICED pins
- This file also contains the functions used to initialize and control the peripherals

❑ For example, the LED pins are initialized as outputs and the button pins are initialized as inputs with a resistive pullup

Creating a New Project

Creating a New Project

- A WICED Studio project can be located anywhere **within the `apps` folder** of the SDK Workspace
- **The key parts of a project are:**
  - A folder with the name of the project
  - A C source file called `<project>.c` inside the project folder
  - A makefile called `<project>.mk` inside the project folder

➢ **IMPORTANT**: The `<project>` name must be the same for the folder name, C file name, and makefile

- ***IMPORTANT: Do NOT use "File -> New -> Project" to create a new project***
  - *You can use "File -> New -> Folder" and "File -> New -> Source File" to create the folder structure and files for a new project*
  - *You can also right click on an existing folder and use "New -> Folder" or "New -> File"*

Creating a New Project

- **The makefile contains:**
  - The application name (any **unique string**)
  - The list of all source files (including `<project>.c`)
  - A list of valid and/or invalid platforms for the given project
  - Makefile macros to provide access to libraries
  - Other resources such as images, web pages, etc.

Creating a New Project

- In order to download the project to your board, **you will need to create a new make target** of the form:

```
<folder1>.[<folder2>...].<project>-<platform> download run
```

  - `<folder1>` is the name of the folder below the apps folder
  - `<folder2>, <folder3>`, etc., are the rest of the path down to the project name; use a period to separate the folder names
  - `<project>` is the name of the project
    - The folder, main C file, and makefile must all have the same name
  - `<platform>` is the name of the hardware platform
    - Example: **CYW943907AEVAL1F_WW101**
    - There must be an entry in the platforms directory that matches the name provided here

Creating a New Project

- **Example:** If we create a folder called "`scu_iot_projects`" for our class projects and a subfolder called "`os`", and call the first project "`blinkled`", the build target for our board would be:

  `scu_iot_projects.os.blinkled-CYW943907AEVAL1F download run`

- The make targets that are defined can be seen in the "Make Target" window along the right side of WIKED Studio
- Expand "43xxx_Wi-Fi" to see the existing make targets

✓ To create a new make target:

- Right click on an existing make target that is similar to what you want to create and select *New...*
- Delete "*Copy of* " (don't forget to remove the space!) and change the name as necessary for your new make target

Creating a New Project

- You must `#include "wiced.h"` at the top of the main C file

- You must also call the `wiced_init()` function in the initialization section of the main C file
  - This function **does all of the initialization** required to get the other WICED APIs to work properly and calls the functions that initialize the peripherals for the kit

GPIO

GPIO

- As explained previously, GPIOs must be initialized before they are used
- The IOs on the kit that are connected to specific peripherals such as LEDs and buttons **are often automatically initialized for you as part of the platform files** (`platform.h` and `platform.c`)

- Once initialized:
  - **outputs** can be driven using `wiced_gpio_output_high()` *and* `wiced_gpio_output_low()`
  - **input** pins can be read using `wiced_gpio_input_get()`

- The **parameter for these functions** is the WICED pin name such as `WICED_GPIO_1` or a peripheral name for your platform such as `WICED_LED1`

GPIO

- In the next example, we use **GPIO** and **`wiced_rtos_delay_milliseconds`**

```
wiced_result_t  wiced_rtos_delay_milliseconds (uint32_t  milliseconds)
```

- **Sleep for a given period of milliseconds**
- Causes the **current thread** to sleep for **at least** the specified number of milliseconds?
  - param[in] `milliseconds`: The time to sleep in milliseconds
  - return `WICED_SUCCESS`: on success
  - return `WICED_ERROR`: if an error occurred

Deliverable

Student Work

- **Task 5-1**
  - **In the previous slide, why we say "*at least*"?**

❑**Example**: This example shows how we blink LED1 on the base board and shield with a frequency of 2 Hz

Example: Blink LED (`blinkled`)
Part: 1/1

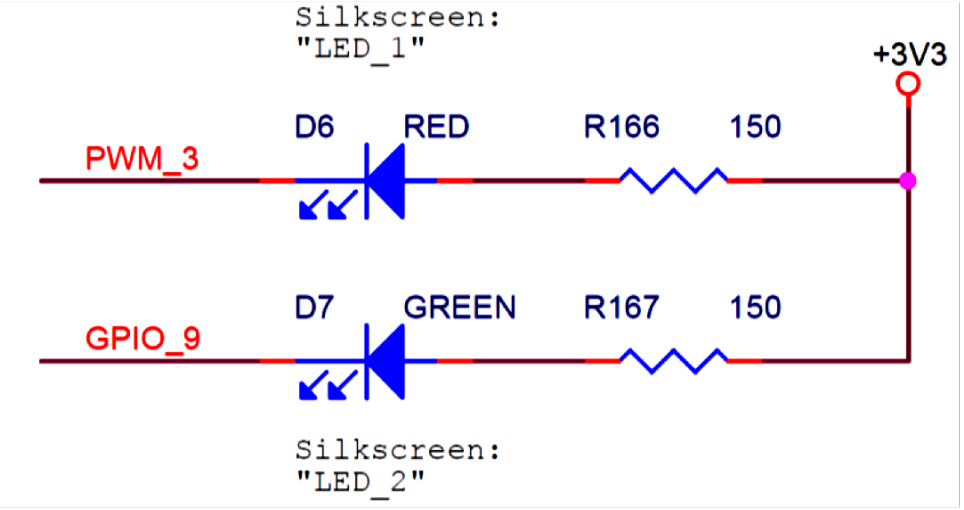
```
#include "wiced.h"
void application_start( )
{
    wiced_init();        /* Initialize the WICED device */

    /* The LED is initialized in platform.c.
     * If it was not, you would need the following:
     * wiced_gpio_init(WICED_SH_LED1, OUTPUT_PUSH_PULL); */

    while ( 1 )
    {
        /* LED on */
        wiced_gpio_output_low( WICED_LED1 );
        wiced_gpio_output_low( WICED_LED2 );
        wiced_gpio_output_low( WICED_SH_LED1 );
        wiced_rtos_delay_milliseconds( 250 );
        /* LED off */
        wiced_gpio_output_high( WICED_LED1 );
        wiced_gpio_output_high( WICED_LED2 );
        wiced_gpio_output_high( WICED_SH_LED1 );
        wiced_rtos_delay_milliseconds( 250 );
    }
}
```

Toggle mainboard and shield LEDs

GPIO

- Note that the **on-board LEDs** are connected as follows:

Switch	CYW43907 Pin Name	WICED_ENUM_ID	Alternate Enumeration in WICED
LED_1	PWM_3	WICED_GPIO_16	WICED_LED1
LED_2	GPIO_9	WICED_GPIO_5	WICED_LED2



Deliverable

Student Work

- **Task 5-2**
  - **Please explain why on-board LEDs turn on and shield's LEDs turn off when we call the following API?**

```
wiced_gpio_output_low()
```

Debug Printing

- The SDK includes **built-in debug print functions** which can be used to display messages via the USB-UART Bridge built into the kit
- The file "`wwd_debug.h`" defines all of the different message types
- We will use one called "`WPRINT_APP_INFO`" which is meant for printing application information
- This is a macro that uses standard `printf()` formatting
- It is enabled by default in the SDK ("`wiced_defaults.h`")

❑ **For example, to print a variable called "test"**

```
WPRINT_APP_INFO( ("The value of test is: %d\n", test) );
```

- Note: **<u>The extra set of parenthesis is required</u>** due to the way the macro is defined

✓ Use a serial port tool (e.g., screen on Linux or SerialTools on MAC OS) with **baudrate 115200 bps**

GPIO Interrupts

- GPIO **interrupts** are controlled using:
  - `wiced_gpio_input_irq_enable()`
  - `wiced_gpio_input_irq_disable()`

```
wiced_result_t wiced_gpio_input_irq_enable
        ( wiced_gpio_t gpio, wiced_gpio_irq_trigger_t trigger,
                wiced_gpio_irq_handler_t handler, void* arg )
```

Enables an interrupt trigger for an input GPIO pin
Using this function on an *uninitialized gpio pin* or a *gpio pin which is set to output* mode is undefined.

- param[in] **`gpio`**: The gpio pin which will provide the interrupt trigger
- param[in] **`trigger`**: The type of trigger (rising/falling edge, high/low level)
- param[in] **`handler`**: A function pointer to the interrupt handler
- param[in] **`arg`**: An argument that will be passed to the interrupt handler
- return **`WICED_SUCCESS`**: on success
- return **`WICED_ERROR`**: if an error occurred with any step

Software Development using WICED Studio

Example: Interrupt-LED (`interrupt_led`)
Part: 1/1

```
#include "wiced.h"
/* Interrupt service routine for the button */
void button_isr(void* arg)
{
        static wiced_bool_t led1 = WICED_FALSE;
        if ( led1 == WICED_TRUE ) {
                wiced_gpio_output_low( WICED_LED1 );
                led1 = WICED_FALSE;
        }
        else {
                wiced_gpio_output_high( WICED_LED1 );
                led1 = WICED_TRUE;
        }
}

void application_start( ) // Main application
{
    wiced_init();/* Initialize the WICED device */

    wiced_gpio_input_irq_enable(WICED_BUTTON2, IRQ_TRIGGER_FALLING_EDGE,
                                button_isr, NULL); /* Setup interrupt */

    while ( 1 ) {
        /* No main loop code required - just wait for interrupt */
    }
}
```

set up a **falling edge interrupt** for the GPIO connected to the button

GPIO Interrupts

- The onboard user buttons are as follows

Switch	CYW43907 Pin Name	WICED_ENUM_ID	Alternate Enumeration in WICED
USER_1 (SW3)	PWM_5	WICED_GPIO_18	WICED_BUTTON1
USER_2 (SW1)	GPIO_8	WICED_GPIO_4	WICED_BUTTON2

- The chip itself provides pull-up resistor
- So the board does not require to use any pull-up

Deliverable

Student Work

- **Task 5-3**
  - **Please extend the previous example to use `WICED_BUTTON1` to toggle `WICED_LED2`**
  - **Demo your project to the TA**

UART

UART

❑ **Example:** We are interested to implement the following application:

- **The number of times the button has been pressed is sent out over the UART interface whenever the button is pressed**
- For simplicity, just count from 0 to 9 and then wrap back to 0 so that you only have to send a single character each time
- We want to setup a UART configuration structure for a baud rate of 115200, data width 8, no parity, 1 stop bit, and no flow control

- **In order to modify UART configuration, <u>we need to avoid automatic UART configuration by WICED</u>**

✓ **To this end, we need to <u>add the following like to the make file</u>**

```
GLOBAL_DEFINES := WICED_DISABLE_STDIO
```

UART

```
/** Initializes a UART interface
 * Prepares an UART hardware interface for communications
 * @param[in]  uart   : The interface which should be initialized
 * @param[in]  config: UART configuration structure defined in
 *                     WICED/platform/include/platform_peripheral.h
 * @param[in] optional_rx_buffer : Pointer to an optional RX ring buffer
 * @return    WICED_SUCCESS : on success.
 * @return    WICED_ERROR   : if an error occurred with any step


wiced_result_t wiced_uart_init( wiced_uart_t uart,
                  const wiced_uart_config_t* config,
                           wiced_ring_buffer_t* optional_rx_buffer );
```

```
/** Transmit data on a UART interface
 * @param[in] uart : The UART interface
 * @param[in]  data     : Pointer to the start of data
 * @param[in]  size     : Number of bytes to transmit
 * @return    WICED_SUCCESS : on success.
 * @return    WICED_ERROR   : if an error occurred with any step

wiced_result_t wiced_uart_transmit_bytes( wiced_uart_t uart, const void* data,
                                                            uint32_t size );
```

Software Development using WICED Studio

Example: UART-Send (uartsend)
Part: 1/2

```
#include "wiced.h"
volatile wiced_bool_t newPress = WICED_FALSE;

void button_isr(void* arg) // Interrupt service routine for the button
{
        static wiced_bool_t led1 = WICED_FALSE;

        /* Toggle LED1 */
        if ( led1 == WICED_TRUE ) {
                wiced_gpio_output_low( WICED_LED1 );
                led1 = WICED_FALSE;
        }
        else {
                wiced_gpio_output_high( WICED_LED1 );
                led1 = WICED_TRUE;
        }
        newPress = WICED_TRUE; /* Need to send new button count value */
}

/* Main application */
void application_start( )
{
    uint8_t pressCount = 0;
    char    printChar;

    wiced_init();/* Initialize the WICED device */
```

Software Development using WICED Studio

```
wiced_gpio_input_irq_enable(WICED_SH_MB1, IRQ_TRIGGER_FALLING_EDGE,
                                button_isr, NULL); /* Setup interrupt */

    /* Configure and start the UART */
    wiced_uart_config_t uart_config =
    {
        .baud_rate    = 115200,
        .data_width   = DATA_WIDTH_8BIT,
        .parity       = NO_PARITY,
        .stop_bits    = STOP_BITS_1,
        .flow_control = FLOW_CONTROL_DISABLED,
    };

    wiced_uart_init(WICED_UART_1, &uart_config, NULL); /* Setup UART */

    while ( 1 ) {
        if(newPress) {
                pressCount ++; /* Increment counter */
                if(pressCount > 9) pressCount = 0;

                printChar = pressCount + '0';

                wiced_uart_transmit_bytes(WICED_UART_1, &printChar , 1);

                newPress = WICED_FALSE;   /* Reset for next press */
        }
    }
}
```

For the configuration to work, we need to use `GLOBAL_DEFINES := WICED_DISABLE_STDIO` in the makefile

We use NULL for the receive buffer since we will only be transmitting values

Alternatively, we could write `pressCount + 0x30` for integer to ASCII conversion

Example: UART-Send (`uartsend`)
Part: 2/2

UART

- **Example:** We are interested in implementing the following application:
- **Receive characters from the UART interface**
- **0 turns the LED off, 1 turns the LED on**

UART

Initialize a ring buffer

```
 * @param[out] ring_buffer: Pointer to the ring buffer structure to be
initialized.
 * @param[in]  buffer      : Pointer to the buffer to use for the ring buffer.
 * @param[in]  buffer_size : Size of the buffer
 * (maximum buffer_size - 1 bytes will be stored)

wiced_result_t ring_buffer_init ( wiced_ring_buffer_t* ring_buffer,
                                  uint8_t* buffer, uint32_t buffer_size );
```

Receive data on a UART interface

```
 * @param[in] uart: The UART interface
 * @param[out]  data : Pointer to the buffer which will store incoming data
 * @param[in,out] size : Number of bytes to receive, function return in same
 * parameter number of actually received bytes
 * @param[in] timeout : Timeout in milliseconds WICED_WAIT_FOREVER and
 * WICED_NO_WAIT can be specified for infinite and no wait.
 * @return    WICED_SUCCESS : on success.
 * @return    WICED_ERROR   : if an error occurred with any step

wiced_result_t wiced_uart_receive_bytes( wiced_uart_t uart, void* data,
                                          uint32_t* size, uint32_t timeout );
```

UART

Example: UART-Receive (`uartreceive`)
Part: 1/2

```
#include "wiced.h"
/* Main application */
void application_start( )
{
    char     receiveChar;
    uint32_t  expected_data_size = 1;

    wiced_init(); /* Initialize the WICED device */

    /* Configure and start the UART. */
    #define  RX_BUFFER_SIZE (5)
    wiced_ring_buffer_t  rx_buffer;
    uint8_t  rx_data[RX_BUFFER_SIZE]; // A 5 byte circular array

    // Initialize ring buffer to hold receive data
    ring_buffer_init(&rx_buffer, rx_data, RX_BUFFER_SIZE );

    wiced_uart_config_t uart_config =
    {
        .baud_rate    = 115200,
        .data_width   = DATA_WIDTH_8BIT,
        .parity       = NO_PARITY,
        .stop_bits    = STOP_BITS_1,
        .flow_control = FLOW_CONTROL_DISABLED,
    };
    wiced_uart_init( WICED_UART_1, &uart_config, &rx_buffer); /* Setup UART */
```

For the configuration to work, we need to use `GLOBAL_DEFINES := WICED_DISABLE_STDIO` in the makefile

UART

Example: UART-Receive (`uartreceive`)
Part: 2/2

```
while ( 1 )
    {
        if ( wiced_uart_receive_bytes(WICED_UART_1, &receiveChar,
                                      &expected_data_size, WICED_NEVER_TIMEOUT )
                                                                  == WICED_SUCCESS )
        {
                /* If we get here then a character has been received */
                 if(receiveChar == '0') /* Turn LED off */
                 {
                         wiced_gpio_output_low( WICED_LED1 );
                 }
                 if(receiveChar == '1') /* Turn LED on */
                 {
                         wiced_gpio_output_high( WICED_LED1 );
                 }
        }

    }
}
```

Deliverable

Student Work

- **Task 5-4**
  - **Please demo the UART project to the TA**