INTERNET OF THINGS | COEN 243
DEPARTMENT OF COMPUTER ENGINEERING, SANTA CLARA UNIVERSITY, CALIFORNIA
BEHNAM DEZFOULI, PHD

Internet of Things

Hands-On: Wireless Networks

Software Development using WICED® Studio

Device Configuration Table (DCT)

- **Device configuration table (DCT)** is a section of the flash with a predefined format that is used to store fundamental information about the system (i.e., client AP SSID, AP passphrase, etc.)
- It can also be used to store your application information
- The DCT is used by the operating system to perform the right operation
  - For example, `wiced_network_up()` reads the network information from the DCT and connects to the specified network

- **The table is built during the make process and written into the flash along with your application**
- The DCT **can also be modified (and written) on the fly** by your application

Device Configuration Table (DCT)

- When developing an application, you can either use the default DCT or you can make a custom one or a custom section of one

- To **preconfigure the Wi-Fi section** of the DCT table you need to:
  - Create a "`*.h`" file (generally called `wifi_config_dct.h` with the correct #defines
  - Add the following line to the makefile so that the your custom DCT is built `WIFI_CONFIG_DCT_H := wifi_config_dct.h`

- You can get a template for the file in the directory `include/default_wifi_config_dct.h`

Device Configuration Table (DCT)

```
/* This is the soft AP used for device configuration */
#define CONFIG_AP_SSID        "IoT_Config"
#define CONFIG_AP_CHANNEL     1
#define CONFIG_AP_SECURITY    WICED_SECURITY_WPA2_AES_PSK
#define CONFIG_AP_PASSPHRASE  "12345678"
```

This mode is used for devices that want to allow other devices to connect to them to perform configuration of the system over Wi-Fi

```
/* This is the soft AP available for normal operation (if used)*/
#define SOFT_AP_SSID          "IoT AP"
#define SOFT_AP_CHANNEL       1
#define SOFT_AP_SECURITY      WICED_SECURITY_WPA2_AES_PSK
#define SOFT_AP_PASSPHRASE    "PASSPHRASE"
```

This mode is used for devices that will act as a Wi-Fi access point during normal operation

```
/* This is the default AP the device will connect to (as a client)*/
#define CLIENT_AP_SSID        "IOT-CPS-COURSE-2G"
#define CLIENT_AP_PASSPHRASE  "------------"
#define CLIENT_AP_BSS_TYPE    WICED_BSS_TYPE_INFRASTRUCTURE
#define CLIENT_AP_SECURITY    WICED_SECURITY_WPA2_MIXED_PSK
#define CLIENT_AP_CHANNEL     1
#define CLIENT_AP_BAND        WICED_802_11_BAND_2_4GHZ
```

This mode is used for connection to an AP

Device Configuration Table (DCT)

- All of the DCT information is mapped into flash by the SDK
- **If you want to read/modify some of the DCT settings from the firmware you will need to understand how the values are stored in flash**
- The WICED SDK provides a predefined structure for the DCT mapping in the file `platform_dct.h`
  - This file can be found in the `WICED/platform/include` folder

```
typedef struct
{
    platform_dct_header_t               dct_header;
    platform_dct_mfg_info_t             mfg_info;
    platform_dct_security_t             security_credentials;
    platform_dct_wifi_config_t          wifi_config;
    platform_dct_ethernet_config_t      ethernet_config;
    platform_dct_network_config_t       network_config;
    platform_dct_bt_config_t            bt_config;
    platform_dct_p2p_config_t           p2p_config;
    platform_dct_ota2_config_t          ota2_config;
    platform_dct_version_t              dct_version;
    platform_dct_misc_config_t          dct_misc;
    /* If you need to add anything to the DCT, add it here, in a new structure*/

} platform_dct_data_t;
```

`wifi_config` is a structure of type `platform_dct_wifi_config_t` that contains information about the Wi-Fi configuration
(please see next page)

Device Configuration Table (DCT)

```
typedef struct
{
    wiced_bool_t                device_configured;
    wiced_config_ap_entry_t     stored_ap_list[CONFIG_AP_LIST_SIZE];
    wiced_config_soft_ap_t      soft_ap_settings;
    wiced_config_soft_ap_t      config_ap_settings;
    wiced_country_code_t        country_code;
    wiced_mac_t                 mac_address;
    uint8_t                     padding[2];  /* ensure 32bit aligned size */
} platform_dct_wifi_config_t;
```

- The second entry `stored_ap_list` is an array of type `wiced_config_ap_entry_t`
- The first element (i.e. index 0) of this array contains information for the access point that the STA connects to as a client

```
typedef struct
{
    wiced_ap_info_t details;
    uint8_t         security_key_length;
    char            security_key[ SECURITY_KEY_SIZE ];
} wiced_config_ap_entry_t;
```

Device Configuration Table (DCT)

```
typedef struct wiced_ap_info
{
    //Service Set Identification (i.e. Name of Access Point)
    wiced_ssid_t    SSID;
    // Basic Service Set Identification (i.e. MAC address of Access Point)
    wiced_mac_t     BSSID;


    //Receive Signal Strength Indication in dBm <-90=Very poor,>-30=Excellent
    int16_t         signal_strength;

    uint32_t        max_data_rate;  // Maximum data rate in kilobits/s
    wiced_bss_type_t    bss_type;   // Network type
    wiced_security_t    security;   // Security type


    //Radio channel that the AP beacon was received on
    uint8_t                 channel;
    wiced_802_11_band_t     band;       //Radio band
    struct wiced_ap_info*   next;       //Pointer to the next scan result


} wiced_ap_info_t;
```

Important APIs

Brings up a network interface

```
 * @param[in] interface     : The interface to bring up
 * @param[in] config        : The network IP configuration
 * @param[in] ip_settings   : Static IP settings that are mandatory
 * for the AP interface, but are optional for the STA interface
 *
 * @return @ref wiced_result_t
 */
extern wiced_result_t wiced_network_up( wiced_interface_t interface,
                wiced_network_config_t config,
                      const wiced_ip_setting_t* ip_settings );
```

Brings down a network interface

```
 *
 * @param[in] interface : The interface to bring down
 *
 * @return @ref wiced_result_t
 */
extern wiced_result_t wiced_network_down(
                                      wiced_interface_t interface );
```

WiFi SDK

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

- An App that attaches to a WPA2 AES PSK network
- LED0 blinks on failure, LED1 blinks on success
- The application retries connecting when unsuccessful

Steps:

1. Copy the template `default_wifi_config_dct.h` into your application folder and name it `wifi_config_dct.h`
2. Modify `wifi_config_dct.h` with the network configuration information
3. Create and edit the makefile
   - Don't forget to add the line for `WIFI_CONFIG_DCT_H`
4. Use function `wiced_network_up()` to read the DCT and start the network
5. Use a serial terminal emulator to look at messages from the device as it boots and connects

Software Development using WICED Studio

Example: Attach WLAN (`attach_wlan`)
Part: 1/2

```
#include "wiced.h"
#define THREAD_PRIORITY      (10)
#define THREAD_STACK_SIZE    (10000)

uint8_t ledToBlink = WICED_SH_LED0;
wiced_bool_t led = WICED_FALSE;

void connectThread(wiced_thread_arg_t arg)
{
    wiced_result_t connectResult;
    wiced_bool_t connected = WICED_FALSE;

    while(connected == WICED_FALSE)
    {
        wiced_network_down(WICED_STA_INTERFACE);
        connectResult = wiced_network_up(WICED_STA_INTERFACE,
                                         WICED_USE_EXTERNAL_DHCP_SERVER, NULL);

        if (connectResult == WICED_SUCCESS) {
            connected = WICED_TRUE;
            ledToBlink = WICED_SH_LED1;
        }
        else  {
             wiced_rtos_delay_milliseconds( 5000 );
        }
    }
}
```

Software Development using WICED Studio

Example: Attach_WLAN
Part: 2/2

```
void application_start( )
{
    wiced_thread_t cnctThreadHandle;
    wiced_init();/* Initialize the WICED device */

    wiced_rtos_create_thread(&cnctThreadHandle, THREAD_PRIORITY,
         "connectionThread", connectThread, THREAD_STACK_SIZE, NULL);

    while ( 1 )
    {
        if ( led == WICED_TRUE )    {
            wiced_gpio_output_low( ledToBlink );
            led = WICED_FALSE;
        }
        else     {
            wiced_gpio_output_high( ledToBlink );
            led = WICED_TRUE;
        }

        wiced_rtos_delay_milliseconds(250);
    }
}
```

WiFi SDK

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

- Attach to a WPA2 network named IOT-CPS-COURSE
- Network info is printed to the UART upon connection:
  - local IP address (`wiced_ip_get_ipv4_address`)
  - Netmask (`wiced_ip_get_netmask`)
  - Gateway IP address (`wiced_ip_get_gateway_address`)
  - IP address for `www.scu.edu` (`wiced_hostname_lookup`)
  - Local device MAC address (`wwd_wifi_get_mac_address`)

- Look at the API guide sections *Components* > *IP Communication* > *Raw IP* and *Components* > *IP Communication* > *DNS lookup*

WiFi SDK

- **The IP addresses** are returned as a structure of type `wiced_ip_address_t`
- One element in the structure (called `ip.v4`) is a `uint32_t` which contains the IPV4 address as 4 hex bytes
- You can mask off each of these bytes individually and print them as decimal values separated by periods to get the format that is typically seen

- The **MAC address** is returned as a structure of type `wiced_mac_t`
- This structure contains an element called `octet` which is an array of 6 octets (bytes)
- You can print each of these bytes individually separated by ":" to see the MAC address in the typical format

Software Development using WICED Studio

Example: WLAN_Info (`wlan_info`)
Part: 1/3

```
#include "wiced.h"

void printIp(wiced_ip_address_t ipV4address)
{
    WPRINT_APP_INFO(("%d.%d.%d.%d\r\n",
            (int)((ipV4address.ip.v4 >> 24) & 0xFF),
                 (int)((ipV4address.ip.v4 >> 16) & 0xFF),
                     (int)((ipV4address.ip.v4 >> 8) & 0xFF),
                          (int)(ipV4address.ip.v4 & 0xFF)));
}


void application_start( )
{
    /* Variables */
    wiced_result_t connectResult;
    wiced_bool_t led = WICED_FALSE;
    uint8_t ledToBlink;
    wiced_ip_address_t ipAddress;
    wiced_mac_t mac;

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

    connectResult = wiced_network_up(WICED_STA_INTERFACE,
                                     WICED_USE_EXTERNAL_DHCP_SERVER, NULL);
```

Software Development using WICED Studio

Example: WLAN_Info (`wlan_info`)
Part: 2/3

```
/* Print out network info */
WPRINT_APP_INFO(("NETWORK INFO\r\n"));
/* IP address */
wiced_ip_get_ipv4_address(WICED_STA_INTERFACE, &ipAddress);
WPRINT_APP_INFO(("IP addr: "));
printIp(ipAddress);

/* Netmask */
wiced_ip_get_netmask(WICED_STA_INTERFACE, &ipAddress);
WPRINT_APP_INFO(("Netmask: "));
printIp(ipAddress);

/* Gateway */
wiced_ip_get_gateway_address(WICED_STA_INTERFACE, &ipAddress);
WPRINT_APP_INFO(("Gateway: "));
printIp(ipAddress);

/* Cypress.com Address */
wiced_hostname_lookup("www.scu.edu", &ipAddress, WICED_NEVER_TIMEOUT,
                                                 WICED_STA_INTERFACE);
WPRINT_APP_INFO(("SCU: "));
printIp(ipAddress);
```

Software Development using WICED Studio

Example: WLAN_Info (`wlan_info`) **Part:** 3/3

```
    /* Device MAC Address */
    wwd_wifi_get_mac_address(&mac , WICED_STA_INTERFACE);
    WPRINT_APP_INFO(("MAC Address: "));
    WPRINT_APP_INFO(("%X:%X:%X:%X:%X:%X\r\n",
                mac.octet[0], mac.octet[1], mac.octet[2],
                              mac.octet[3], mac.octet[4], mac.octet[5]));

    if(connectResult == WICED_SUCCESS)    {
        ledToBlink = WICED_SH_LED1; /* Blink LED1 if successful */
    }
    else     {
        ledToBlink = WICED_SH_LED0; /* Blink LED0 if unsuccessful */
    }

    while ( 1 )     {
        /* Blink appropriate LED */
        if ( led == WICED_TRUE )          {
            wiced_gpio_output_low( ledToBlink );
            led = WICED_FALSE;
        }
        else          {
            wiced_gpio_output_high( ledToBlink );
            led = WICED_TRUE;
        }
        wiced_rtos_delay_milliseconds(250);
    }
}
```

Device Configuration Table (DCT) | Advanced Topics

- The DCT may exist as **a series of flash rows** inside of the application processor (i.e. if it has internal flash), or it may exist in a serial flash attached to the Wi-Fi chip
- In order to read from the DCT you need to call the function `wiced_dct_read_lock()` which will read the DCT into a RAM buffer which you can then modify and then write back to the flash with the function `wiced_dct_write()`
- You provide the `wiced_dct_read_lock()` call with a pointer to a pointer to an empty structure which will be filled with the DCT Wi-Fi data
- The type of structure depends on which section of the DCT that you want to read (the section is a parameter to the `wiced_dct_read_lock()` function)
- For example, if you want to read the DCT_WIFI_CONFIG_SECTION, then the pointer type would be `platform_dct_wifi_config_t`

Raspberry Pi Network Configuration

Raspberry Pi Network Configuration

IP Addressing

- **Identify Interfaces**
  - To quickly identify all available Ethernet interfaces, you can use the `ifconfig` command as follows: `ifconfig -a`

- **To temporarily configure an IP address**, you can use the `ifconfig` command in the following manner
  - Just modify the IP address and subnet mask to match your network requirements

```
sudo ifconfig wlan0 192.168.1.100 netmask 255.255.255.0
```

- To **configure a default gateway**, you can use the `route` command in the following manner
- Modify the default gateway address to match your network requirements

```
sudo route add default gw 10.0.0.1 eth0
```

- To verify your default gateway configuration, you can use the `route` command in the following manner: `route -n`

Raspberry Pi Network Configuration

IP Addressing

- To turn on an interface: `sudo ifup wlan0`
- To turn off an interface: `sudo ifdown wlan0`

- We can configure static or dynamic IP assignment for interfaces in `/etc/network/interfaces`

- To configure an interface to use static address assignment:

```
auto wlan0
iface wlan0 inet static
address 10.0.0.100
netmask 255.255.255.0
gateway 10.0.0.1
```

- To configure an interface to use DHCP for dynamic address assignment:

```
auto wlan0
iface wlan0 inet dhcp
```

Raspberry Pi Network Configuration

IP Addressing

- The **loopback interface** is identified by the system as `lo`
- It has a default IP address of 127.0.0.1
- It can be viewed using: `ifconfig lo`
- By default, there should be two lines in `/etc/network/interfaces` responsible for automatically configuring your loopback interface:

```
auto lo
iface lo inet loopback
```

Raspberry Pi Network Configuration

Interface Configuration Options

More details

Layer-2 options:

- `auto interface`: Start the interface(s) at boot
- `allow-auto interface`: Same as auto
- `allow-hotplug interface`: Start the interface when a "hotplug" event is detected

Layer-3 options:

- All options below are a suffix to a defined interface (`iface <Interface_family>`)
- Basically, `iface wlan0` creates a **stanza** called `wlan0` on an wireless device
  - `inet static`: Defines a static IP address
  - `inet manual`: Does not define an IP address for an interface. Generally used by interfaces that are bridge or aggregation members, interfaces that need to operate in promiscuous mode
  - `inet dhcp`: Acquire IP address through DHCP protocol.
  - `inet6 static`: Defines a static IPv6 address

Raspberry Pi Network Configuration

Interface Configuration Options

- `address`: IP address for a static IP configured interface
- `netmask`: netmask
- `gateway`: The default gateway of a server
- `wpa-ssid:` Wireless: Set a wireless WPA SSID
- `mtu`: MTU size
- `wpa-psk`: Set a hexadecimal encoded PSK for your SSID