WICED Studio

WICED™ Development System Factory Programming

Associated Part Family: BT CYW2070x

Doc. No.: 002-19004 Rev. *A

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
www.cypress.com

Contents

About this Document

Purpose and Scope

This document describes the process to program device specific information into a Cypress Wireless Internet Connectivity for Embedded Devices (WICED™; pronounced “wicked”) device. The process is typically used at the time of device manufacture.

Note: This document applies to **WICED SDK 4.2** or higher.

Audience

This document is for software developers who are using the WICED Development System to create a manufacturing process for WICED-based embedded wireless networked devices.

Acronyms and Abbreviations

In most cases, acronyms and abbreviations are defined on first use.

For a comprehensive list of acronyms and other terms used in Cypress documents, go to www.cypress.com/glossary.

IoT Resources and Technical Support

Cypress provides a wealth of data at www.cypress.com/internet-things-iot to help you to select the right IoT device for your design, and quickly and effectively integrate the device into your design. Cypress provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates. Customers can acquire technical documentation and software from the Cypress Support Community website (community.cypress.com/)

1 General Overview

During the manufacturing process, individual WICED devices require a limited set of unique parameters to be programmed into onboard flash. The WICED SDK provides the necessary tools to aid the customization and programming process.

By default, the WICED build system generates three separate firmware elements for each WICED device when an application is built: the Bootloader, the Application and the Device Configuration Table (DCT). The bootloader and application are common to all devices, however the DCT may contain device specific information including, but not limited to, a unique device serial number, WLAN MAC address and security certificate.

This document describes how to generate a unique per-device DCT image, and how to program the DCT image together with the bootloader and application images, into a WICED device during manufacture.

For the example in this document, the WICED SDK is used from a command line (rather than the WICED IDE) since a typical manufacturing environment is run with a script. The WICED IDE may alternately be used to issue build commands.

Examples are provided for the Windows® operating system, the same procedure using equivalent commands may also be used on OS X and Linux since the WICED SDK runs on all major operating systems.

2 Creating a Unique DCT

2.1 Overview

The process to create a unique DCT image is described here by way of an example. Developers are free to customize the process to suit individual manufacturing requirements. The WICED SDK `temp_control` demonstration application provides two unique DCT parameter files. The text in this section shows how to use these parameter files to create two unique DCT images suitable for programming into two individual WICED devices.

Using the WICED IDE (or a command shell), navigate to the WICED SDK temp control directory located in the SDK at `<WICED-SDK>/apps/demo/temp_control`. In addition to the usual application files, the directory includes a file called `factory_reset_dct.c` and a sub-directory called `mfg` which contains two files `0001.txt` and `0002.txt`.

The `factory_reset_dct.c` file contains a `factory_reset_dct_t` structure similar to that shown in Figure 2-1. The structure contains static information that is populated into the DCT for all devices, as well as placeholders for dynamic information that is populated on a per-device basis by the WICED build system. Dynamic information is prepended with the keyword `_DYNAMIC_`, for example `_DYNAMIC_WLAN_MAC_ADDRESS`.

IMPORTANT NOTE: ALL variables defined in the generic WICED SDK `platform_dct.h` header file (located in the `<WICED-SDK>/Platform/include` directory) but not listed in the `factory_reset_dct_t` structure will be initialized to 0 by the WICED SDK build system when a unique DCT is generated! For the default reference see `<WICED-SDK>/internal/dct.c.`

Figure 2-1. Example Factory Reset DCT Structure

```
static const factory_reset_dct_t initial_dct =
{
...
    /* Manufacturing Section _________________________________________________________*/
    .platform.mfg_info = _DYNAMIC_MFG_INFO,

    /* Security Credentials for Config Section _______________________________________*/
    .platform.security_credentials.certificate = _DYNAMIC_CERTIFICATE_STORE,
    .platform.security_credentials.cooee_key   = COOEE_KEY_STRING,
...
    .platform.wifi_config.stored_ap_list[0]                = _DYNAMIC_STORED_AP_INFO,
    .platform.wifi_config.soft_ap_settings.SSID            = _DYNAMIC_SOFT_AP_SSID,
    .platform.wifi_config.soft_ap_settings.security_key    = _DYNAMIC_SOFT_AP_PASSPHRASE,
    .platform.wifi_config.config_ap_settings.SSID          = _DYNAMIC_CONFIG_AP_SSID,
    .platform.wifi_config.config_ap_settings.security_key  = _DYNAMIC_CONFIG_AP_PASSPHRASE,
...
    .platform.wifi_config.mac_address       = _DYNAMIC_WLAN_MAC_ADDRESS,
...
};
```

The `0001.txt` and `0002.txt` files are unique per-device DCT parameter files. Each file includes unique parameters that are programmed into an individual device. Notice there is a unique parameter in each file called `WLAN_MAC_ADDRESS`. The assigned value of this unique parameter replaces the corresponding `_DYNAMIC_WLAN_MAC_ADDRESS` placeholder in the `factory_reset_dct_t` structure when individual DCT images are generated by the WICED SDK build system. Similarly, each parameter in the unique DCT parameter file is used to replace the matching dynamic placeholder in the `factory_reset_dct` structure. The unique DCT parameter file `0001.txt` is reproduced in Figure 2-2 for reference.

Figure 2-2. Example of a unique DCT parameter file : 0001.txt

```
SOFT_AP_SSID       = {sizeof("WICED_SOFT_AP-0001")-1,"WICED_SOFT_AP-0001"}

SOFT_AP_PASSPHRASE = "abcd1234"

CONFIG_AP_SSID       = {sizeof("WICED-0001")-1,"WICED-0001"}

CONFIG_AP_PASSPHRASE = "12345678"

WLAN_MAC_ADDRESS     = {"\x02\x0A\xF7\x00\x00\x01"}

STORED_AP_INFO =

    {

        .details.SSID        = {sizeof("YOUR_AP_SSID")-1,"YOUR_AP_SSID"},

        .security_key        = "YOUR_AP_PASSPHRASE",

        .security_key_length = sizeof("YOUR_AP_PASSPHRASE")-1,

        .details.security    = WICED_SECURITY_WPA2_MIXED_PSK,

    }

MFG_INFO=

    {

       .manufacturer           = "Cypress",

       .product_name           = "Wiced Device",

       .BOM_name               = "100-123793-0000",

       .BOM_rev                = "P100",

       .serial_number          = "0001",

       .manufacture_date_time  = "2013/10/30 12:30:15",

       .manufacture_location   = "USA",

       .bootloader_version     = "1.0",

    }

CERTIFICATE_STORE="-----BEGIN CERTIFICATE-----\r\n"

                  "MIIDdTCCAl2gAwIBAgILBAAAAAABFUtaw5QwDQYJKoZIhvcNAQEFBQAwVzELMAkG\r\n"

                  "A1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExEDAOBgNVBAsTB1Jv\r\n"

                  "b3QgQ0ExGzAZBgNVBAMTEkdsb2JhbFNpZ24gUm9vdCBDQTAeFw05ODA5MDExMjAw\r\n"

                  "MDBaFw0yODAxMjgxMjAwMDBaMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9i\r\n"

                  "YWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENBMRswGQYDVQQDExJHbG9iYWxT\r\n"

                  "aWduIFJvb3QgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDaDuaZ\r\n"

                  "jc6j40+Kfvvxi4Mla+pIH/EqsLmVEQS98GPR4mdmzxzdzxtIK+6NiY6arymAZavp\r\n"

                  "xy0Sy6scTHAHoT0KMM0VjU/43dSMUBUc71DuxC73/OlS8pF94G3VNTCOXkNz8kHp\r\n"

                  "1Wrjsok6Vjk4bwY8iGlbKk3Fp1S4bInMm/k8yuX9ifUSPJJ4ltbcdG6TRGHRjcdG\r\n"

                  "snUOhugZitVtbNV4FpWi6cgKOOvyJBNPc1STE4U6G7weNLWLBYy5d4ux2x8gkasJ\r\n"

                  "U26Qzns3dLlwR5EiUWMWea6xrkEmCMgZK9FGqkjWZCrXgzT/LCrBbBlDSgeF59N8\r\n"

                  "9iFo7+ryUp9/k5DPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMBAf8E\r\n"

                  "BTADAQH/MB0GA1UdDgQWBBRge2YaRQ2XyolQL30EzTSo//z9SzANBgkqhkiG9w0B\r\n"

                  "AQUFAAOCAQEA1nPnfE920I2/7LqivjTFKDK1fPxsnCwrvQmeU79rXqoRSLblCKOz\r\n"

                  "yj1hTdNGCbM+w6DjY1Ub8rrvrTnhQ7k4o+YviiY776BQVvnGCv04zcQLcFGUl5gE\r\n"

                  "38NflNUVyRRBnMRddWQVDf9VMOyGj/8N7yy5Y0b2qvzfvGn9LhJIZJrglfCm7ymP\r\n"

                  "AbEVtQwdpf5pLGkkeB6zpxxxYu7KyJesF12KwvhHhm4qxFYxldBniYUr+WymXUad\r\n"

                  "DKqC5JlR3XC321Y9YeRq4VzW9v493kHMB65jUr9TU/Qr6cf9tveCX4XSQRjbgbME\r\n"

                  "HMUfpIBvFSDJ3gyICh3WZlXi/EjJKSZp4A==\r\n"

                  "-----END CERTIFICATE-----\r\n"

                  "\0"

                  "\0"
```

2.2 Generating DCT Images

The following description provides an example of how to use the WICED SDK to generate two unique DCT images suitable for use with the `temp_control` application. A command prompt is used for this example since most manufacturing processes are run using a script (rather than a GUI).

Open a command prompt and cd to the top level WICED SDK directory.

Enter the following command to build the `temp_control` application and bootloader for the `BCM943362WCD4` platform using the platform default RTOS, Network Stack and WLAN-MCU bus.

Figure 2-3. Windows cmd shell command to build temp_control application

```
> .\make demo.temp_control-BCM943362WCD4
```

Figure 2-4. Bash shell command to build temp_control application

```
> ./make demo.temp_control-BCM943362WCD4
```

After the build completes, use the following commands to generate two unique DCT. Note that the only difference between these commands is the appended unique DCT parameter file `0001.txt` vs. `0002.txt`. Each command must be entered on a single line.

NOTE: You need to change `<platform>` to the platform you are using (ex: `43364WCD1` or `BCM943362WCD4`)!

NOTE: Windows uses '\' separators between sub-directories in the path. However, the filename is handed to the OpenOCD driver, which always uses '/' separators.

Figure 2-5. Windows cmd shell command to build separate unique factory reset DCT files

```
> .\make demo.temp_control-<platform> factory_reset_dct apps/demo/temp_control/mfg/0001.txt
> .\make demo.temp_control-<platform> factory_reset_dct apps/demo/temp_control/mfg/0002.txt
```

Figure 2-6. Bash shell command to build separate unique factory reset DCT files

```
> ./make demo.temp_control-<platform> factory_reset_dct apps/demo/temp_control/mfg/0001.txt
> ./make demo.temp_control-<platform> factory_reset_dct apps/demo/temp_control/mfg/0002.txt
```

The WICED build system writes each of the generated `factory_reset_dct_000X` images to the build directory `<Wiced-SDK>/build/demo.temp_control-<platform>/factory_reset`: The *.elf files are used in the next step to program the files into FLASH, as they contain the correct offset to program the data.

As previously discussed, each generated unique DCT image contains common, as well as device specific, information.

3 Writing an Image to Microprocessor Flash

The WICED SDK build system uses the OpenOCD utility to download images via USB JTAG to the microprocessor flash on the WICED device. By default, the SDK hides these commands, full output is otherwise available by adding "`VERBOSE=1`" to an application build string. For factory programming, a review of just the commands required to write images to the flash is provided in this section.

3.1 OCD Command-line Configuration

In addition to the image to be written, OpenOCD requires configuration information about the platform to assist with flash programming. This includes the USBIO device type used to emulate JTAG, the microcontroller type and information about how to access the flash. This information is provided by the SDK in various OpenOCD configuration files.

For convenience, we suggest setting up environment variables to identify each configuration file using the Windows set command as shown in Figure 2-3 (use an equivalent command for your operating system if you are not using Windows). It is also possible to provide each of these files as direct arguments to OpenOCD if you choose not to setup environment variables. This example uses the `BCM943362WCD4` microprocessor. If your WICED platform does not use the `BCM943362WCD4` platform, locate and use the correct OpenOCD configuration files to suit your microprocessor family. Run the commands in Figure 2-3 now to setup the necessary OpenOCD configuration environment variables.

NOTE: The easiest way to determine which configuration files to use for your platform is to add "`VERBOSE=1`" to the application build string.

Figure 3-1. Set Windows Environment variables to identify OpenOCD configuration files

```
> set JTAG_CFG=.\tools\OpenOCD\BCM9WCD1EVAL1.cfg


> set MCU_CFG=.\tools\OpenOCD\stm32f2x.cfg


> set FLASH_CFG=.\tools\OpenOCD\stm32f2x-flash-app.cfg
```

Figure 3-2. Set bash Environment variables to identify OpenOCD configuration files

```
> export JTAG_CFG=./tools/OpenOCD/BCM9WCD1EVAL1.cfg


> export MCU_CFG=./tools/OpenOCD/stm32f2x.cfg


> export FLASH_CFG=./tools/OpenOCD/stm32f2x-flash-app.cfg
```

3.2 OCD Command-line Internal Flash Programming

This section describes the method for platforms that use an internal FLASH to store program and data. For platforms with external SFLASH, read this section, then read the next section which describes the differences.

Images are loaded into the flash using the 'elf' format since elf files natively include the physical address in flash to locate the image. If a binary file is otherwise used, OpenOCD also requires the physical address in flash to write the image.

The commands shown in Figure 2-4 provide examples showing how to use OpenOCD to program the Bootloader, Application and DCT images to flash. To program a unique DCT image into a device, the factory programming script invokes OpenOCD using a unique DCT elf file for each new device.

The final command, which is used to download the unique DCT image, includes logging and error reporting that may also be used with the other commands to download the bootloader and application if desired.

Run the commands in Figure 2-4 now to program the Bootloader, Application and unique DCT images to the microprocessor flash memory. The 'echo' commands may be safely ignored, they are provided for illustrative purposes.

NOTE: You need to change `<platform>` to the platform you are using (ex: `43364WCD1` or `BCM943362WCD4`)!

NOTE: Windows uses '\' separators between sub-directories in the path. However, the filename is handed to the OpenOCD driver, which always uses '/' separators.

Figure 3-3. Windows cmd shell command sequences to write Bootloader, Application and DCT images to flash

```
> echo "Downloading Bootloader ..."

> .\tools\OpenOCD\Win32\openocd-all-brcm-libftdi.exe -s .\tools\OpenOCD\scripts -f %JTAG_CFG%
-f %MCU_CFG% -f %FLASH_CFG% -c "flash write_image erase build/waf.bootloader-NoOS-
<platform>/binary/waf.bootloader-NoOS-<platform>.stripped.elf" -c shutdown


> echo "Downloading Application ..."

> .\tools\OpenOCD\Win32\openocd-all-brcm-libftdi.exe -s .\tools\OpenOCD\scripts -f %JTAG_CFG%
-f %MCU_CFG% -f %FLASH_CFG% -c "flash write_image erase build/demo.temp_control-
<platform>/binary/demo.temp_control-<platform>.stripped.elf" -c shutdown


> echo "Downloading DCT ..."

> .\tools\OpenOCD\Win32\openocd-all-brcm-libftdi.exe -s .\tools\OpenOCD\scripts -f %JTAG_CFG%
-f %MCU_CFG% -f %FLASH_CFG% -c "flash write_image erase build/demo.temp_control-
<platform>/factory_reset/factory_reset_dct_0001.stripped.elf" -c shutdown
```

Figure 3-4. Bash shell command sequences to write Bootloader, Application and DCT images to flash

```
> echo "Downloading Bootloader ..."

> ./tools/OpenOCD/Win32/openocd-all-brcm-libftdi.exe -s ./tools/OpenOCD/scripts -f $JTAG_CFG
-f $MCU_CFG -f $FLASH_CFG -c "flash write_image erase build/waf.bootloader-NoOS-
<platform>/binary/waf.bootloader-NoOS-<platform>.stripped.elf" -c shutdown


> echo "Downloading Application ..."

> ./tools/OpenOCD/Win32/openocd-all-brcm-libftdi.exe -s ./tools/OpenOCD/scripts -f $JTAG_CFG
-f $MCU_CFG -f $FLASH_CFG -c "flash write_image erase build/demo.temp_control-
<platform>/binary/demo.temp_control-<platform>.stripped.elf" -c shutdown


> echo "Downloading DCT ..."

> ./tools/OpenOCD/Win32/openocd-all-brcm-libftdi.exe -s ./tools/OpenOCD/scripts -f $JTAG_CFG
-f $MCU_CFG -f $FLASH_CFG -c "flash write_image erase build/demo.temp_control-
<platform>/factory_reset/factory_reset_dct_0001.stripped.elf" -c shutdown
```

NOTE: The commands in Figure 3-3 (or Figure 3-4) may be run at any time, in any order and more than once on a particular device that is using internal FLASH after the corresponding elf file is available.

NOTE: Each command must be provided on a single line. Using copy-paste to grab the command line text may insert additional unwanted carriage returns that should be removed.

3.3 OCD Command-line External Flash Programming

For some platform builds where the program and data is stored in an external SFLASH, the WICED Filesystem and the Application Look Up Table (LUT) are needed as well. As discussed in the beginning of this section, when using the WICED IDE , add “VERBOSE=1” to the build command to show which files are downloaded to the Flash during your build and their destination addresses.

These examples show the download sequence for the snip.scan application built for the BCM943907WAE_1. To provide different unique DCTs, copy the method shown above, and replace the “DCT.bin” file in the following examples with your factory_reset_dct files.

NOTE: It is important to note that external FLASH writing script requires the destination address. Most addresses are fixed due to the nature of the FLASH Layout: Bootloader, DCT, Apps LUT and Filesystem. The location for the Application is dependent on the size of the Resource Filesystem. If the Resource Filesystem size changes, the destination address for the Application will change.

NOTE: These download examples use the script file =.\apps\waf\sflash_write\sflash_write.tcl and the first download (bootloader) specifies that the FLASH be erased (note the “1” in the string passed to the script). This should only be done once. You can also only write each element once (different than writing to an internal FLASH).

NOTE: These download examples use the script file =.\apps\waf\sflash_write\sflash_write.tcl and define the destination address.

NOTE: Change <platform>, <chip> and <platform>-<board>-<bus> to the platform you are using. You can determine these values for your platform by looking at the output in the Console tab of the IDE when you build your project during development with “VERBOSE=1” as a command argument.

Figure 3-5. Windows cmd shell command sequences

```
> .\make snip.scan-BCM943907WAE_1

> set JTAG_CFG=.\tools\OpenOCD\BCM9WCD1EVAL1.cfg
> set MCU_CFG=.\tools\OpenOCD\BCM4390x.cfg
> set FLASH_TCL=.\apps\waf\sflash_write\sflash_write.tcl

> echo Downloading Bootloader ...
> .\tools\OpenOCD\Win32\openocd-all-brcm-libftdi.exe -s .\tools\OpenOCD\scripts -f $JTAG_CFG
-f $MCU_CFG -f $FLASH_TCL -c "sflash_write_file build/waf.bootloader-NoOS-NoNS--<platform>-
<board>-<bus>/binary/waf.bootloader-NoOS-NoNS-<platform>-<board>-<bus>.trx.bin 0x00000000
<platform>-<board>-<bus> 1 <chip>" -c shutdown

> echo Downloading resources filesystem ...
> .\tools\OpenOCD\Win32\openocd-all-brcm-libftdi.exe -s .\tools\OpenOCD\scripts -f $JTAG_CFG
-f $MCU_CFG -f $FLASH_TCL -c "sflash_write_file build/snip.scan-<platform>/filesystem.bin
69632 <platform>-<board>-<bus> 0 <chip>" -c shutdown

> echo Downloading APP0 ...
> .\tools\OpenOCD\Win32\openocd-all-brcm-libftdi.exe -s .\tools\OpenOCD\scripts -f $JTAG_CFG
-f $MCU_CFG -f $FLASH_TCL -c "sflash_write_file build/snip.scan-<platform>/binary/snip.scan-
<platform>.stripped.elf 589824 <platform>-<board>-<bus> 0 <chip>" -c shutdown

> echo Downloading apps lookup table ...
> .\tools\OpenOCD\Win32\openocd-all-brcm-libftdi.exe -s .\tools\OpenOCD\scripts -f $JTAG_CFG
-f $MCU_CFG -f $FLASH_TCL  -c "sflash_write_file build/snip.scan-<platform>/APPS.bin 0x10000
<platform>-<board>-<bus> 0 <chip>" -c shutdown

> echo Downloading DCT ...
> .\tools\OpenOCD\Win32\openocd-all-brcm-libftdi.exe -s .\tools\OpenOCD\scripts -f $JTAG_CFG
-f $MCU_CFG -f $FLASH_TCL -c "sflash_write_file build/snip.scan--<platform>/DCT.bin
0x00008000 <platform>-<board>-<bus> 0 <chip>" -c shutdown
```

Figure 3-6. Bash shell command sequences

```
> ./make snip.scan-BCM943907WAE_1


> export JTAG_CFG=./tools/OpenOCD/BCM9WCD1EVAL1.cfg

> export MCU_CFG=./tools/OpenOCD/BCM4390x.cfg

> export FLASH_TCL=./apps/waf/sflash_write/sflash_write.tcl


> echo Downloading Bootloader ...

> ./tools/OpenOCD/Win32/openocd-all-brcm-libftdi.exe -s ./tools/OpenOCD/scripts -f $JTAG_CFG
-f $MCU_CFG -f $FLASH_TCL -c "sflash_write_file build/waf.bootloader-NoOS-NoNS-<platform>-
<board>-<bus>/binary/waf.bootloader-NoOS-NoNS-<platform>-<board>-<bus>.trx.bin 0x00000000
<platform>-<board>-<bus> 1 <chip>" -c shutdown


> echo Downloading resources filesystem ...

> ./tools/OpenOCD/Win32/openocd-all-brcm-libftdi.exe -s ./tools/OpenOCD/scripts -f $JTAG_CFG
-f $MCU_CFG -f $FLASH_TCL -c "sflash_write_file build/snip.scan-<platform>/filesystem.bin
69632 <platform>-<board>-<bus> 0 <chip>" -c shutdown


> echo Downloading APP0 ...

> ./tools/OpenOCD/Win32/openocd-all-brcm-libftdi.exe -s ./tools/OpenOCD/scripts -f $JTAG_CFG
-f $MCU_CFG -f $FLASH_TCL -c "sflash_write_file build/snip.scan-<platform>/binary/snip.scan-
BCM943907WAE_1.stripped.elf 589824 <platform>-<board>-<bus> 0 <chip>" -c shutdown


> echo Downloading apps lookup table ...

> ./tools/OpenOCD/Win32/openocd-all-brcm-libftdi.exe -s ./tools/OpenOCD/scripts -f $JTAG_CFG
-f $MCU_CFG -f $FLASH_TCL  -c "sflash_write_file build/snip.scan-<platform>/APPS.bin 0x10000
<platform>-<board>-<bus> 0 <chip>" -c shutdown


> echo Downloading DCT ...

> ./tools/OpenOCD/Win32/openocd-all-brcm-libftdi.exe -s ./tools/OpenOCD/scripts -f $JTAG_CFG
-f $MCU_CFG -f $FLASH_TCL -c "sflash_write_file build/snip.scan-<platform>/DCT.bin
0x00008000 <platform>-<board>-<bus> 0 <chip>" -c shutdown
```

4 Assigning a MAC Address to your Device

A Medium Access Control (MAC) address is used to uniquely identify a Wi-Fi device on the wireless network. Each Wi-Fi device **must** be assigned a unique Wi-Fi MAC address. The WICED SDK provides various options to set the Wi-Fi MAC address of a WICED device.

4.1 WLAN OTP Memory

Each WLAN chip includes a small amount of One-Time Programmable (OTP) memory. Many Wi-Fi module manufacturers program a MAC address into the OTP during the module manufacturing process. In most cases, it is best to leave WICED to use the MAC address in OTP. If your devices each require a MAC address other than that assigned by the module manufacturer, you may use one of the methods described in Sections 4.2 or 4.3 to set the MAC address (and override the MAC address in OTP).

4.2 DCT

A unique per-device MAC address may be specified in the DCT as described in Section 2.1. To direct WICED and the WLAN chip to use the MAC addressed located in the DCT, it is necessary to define a global variable in the application makefile. Using the `temp_control` app as an example, add a `MAC_ADDRESS_SET_BY_HOST` global define to the `temp_control` application makefile located in the WICED SDK at `<WICED-SDK>/App/demo/temp_control/temp_control.mk` as follows:

```
GLOBAL_DEFINES += MAC_ADDRESS_SET_BY_HOST
```

4.3 Custom

If you do not want to use the MAC address in the WLAN OTP or in the DCT, you may redefine the WICED API function `host_platform_get_mac_address()` to provide a MAC address when the WICED Wi-Fi driver initializes the WLAN chip.

For most architectures, `host_platform_get_mac_address()` is located in the file:

`<WICED-SDK>/WICED/platform/MCU/wwd_platform_separate_mcu.c`

For `BCM94390x` architectures, `host_platform_get_mac_address()` is located in the file:

`<WICED-SDK>/WICED/platform/MCU/BCM4390x/BCM4390x_platform.c`

The global variable `MAC_ADDRESS_SET_BY_HOST` must also be configured as described in Section 5.2.

4.4 NVRAM (Development ONLY)

A text file known as NVRAM is provided for each WICED platform. The NVRAM provides platform specific information related to the Wi-Fi chip including, but not limited to, the frequency of the crystal used, transmit power limits and a MAC address. In general, changing NVRAM variables is not recommended since it is possible to adversely impact the performance of the Wi-Fi chip.

The MAC address specified in the NVRAM is used by the Wi-Fi chip if the OTP does not contain a MAC address. Setting the MAC address in NVRAM is only useful during development since the NVRAM is compiled into the final application, and the final application is common to all devices.

Document Revision History

Document Title: WICED™ Development System Factory Programming

Document Number: 002-19004

Revision	ECN	Issue Date	Description of Change
**		11/05/2013	WICED-AN800-R: Initial Doc.
		02/17/2017	WICED-AN800-R 0.2: Bring up to date with latest SDK 4.0
		02/24/2017	WICED-AN800-R 0.3: Correct some mistakes, add changes to indicate changes needed per platform
*A		03/21/2017	Converted to Cypress template format.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

t

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2013-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.