# CYW208XX Feature and Peripheral Guide

## About this document

### Scope and purpose

This document introduces the CYW208XX ultra-low power, dual-mode Bluetooth v5.0 wireless MCU with an Arm® Cortex®-M4 CPU by exploring the CYW208XX device architecture along with the development tools to write applications using ModusToolbox™ software. To help you develop these applications, this technical guide focuses on the use of building blocks of the device and the tools that can aid during the development of the applications. This technical guide is intended to provide you with a brief overview of the features and peripherals of CYW208XX.

## Table of contents

# 1 Introduction

CYW208XX is a Bluetooth 5.0-compliant SoC with support for Bluetooth Basic Rate (BR), Enhanced Data Rate (EDR), and Bluetooth Low Energy. CYW208XX supports all optional LE features as per Bluetooth core specification v4.2 and the LE 2 Mbps feature as per specification v5.0. Manufactured using an advanced 40-nm CMOS low-power process, CYW208XX employs the highest level of integration to eliminate all critical external components, thereby minimizing the device's footprint and the costs associated with implementing Bluetooth solutions.

CYW208XX integrates a 96-MHz Arm® Cortex®-M4 CPU that offers significant processing power but still maintains a smaller footprint. CYW208XX offers 160 KB of RAM, 16 KB of Patch RAM, 8 KB of Cache, 256 KB of on-chip secure flash for Bluetooth applications along with 1 MB of ROM space dedicated for low-level Bluetooth stack and drivers. CYW208XX also hosts several hardware peripheral blocks such as ADC, PWM, I$^2$C, SPI, UART and keyboard scanner to aid in interfacing with other components. This guide will help you to explore CYW208XX device features and guide you in using the SoC peripherals.

# 2 Resources

A wealth of data is available at **www.cypress.com** to help you to select the right device, and quickly and effectively integrate the device into your design. The following is an abbreviated list of resources related to this application note:

- **Overview: Bluetooth Low Energy and Bluetooth**
- **Product Selectors: Bluetooth Product Selector**
- **CYW20819 Datasheet:** Describes and provides electrical specifications for CYW20819 Bluetooth SoC.
- **CYW20820 Datasheet:** Describes and provides electrical specifications for CYW20820 Bluetooth SoC. CYW20820 has an integrated power amplifier, which can transmit at 10 dBm transmit power.
- **IDE: ModusToolbox IDE for Bluetooth-SDK**
- **Application Notes:** Cover a broad range of topics, from basic to advanced level.
- **Code Examples**: **Bluetooth-SDK code examples** available in GitHub repository.
- **Bluetooth-SDK Documents**: Up-to-date Bluetooth SDK documents, which includes firmware development, debugging guides, test tools, library guides, and API documentation.
- **Development Kits:** Some examples include**:**
  - **CYW920819EVB-02 Evaluation Kit** enables you to evaluate and develop single-chip Bluetooth applications using CYW20819, an ultra-low-power dual-mode Bluetooth 5.0 wireless MCU device.
  - **CYW920820EVB-02 Evaluation Kit** enables you to evaluate and develop single-chip Bluetooth applications using CYW20820, an ultra-low-power dual-mode Bluetooth 5.0 wireless MCU device. CYW20820 has an integrated power amplifier, which can transmit at 10 dBm transmit power.
  - **CYBT-213043-MESH EZ-BT™ Module Mesh Evaluation Kit** enables you to evaluate and develop the Bluetooth SIG mesh functionality using the EZ-BT Bluetooth 5.0-qualified module CYBT-213043-02.

## 2.1 Hardware EVB

Three evaluation kits are available for evaluating the CYW208XX Bluetooth SoC. You can use:

- The **CYW920819EVB-02** Evaluation Kit to evaluate and develop single-chip Bluetooth applications using CYW20819, an ultra-low-power dual-mode Bluetooth 5.0 wireless MCU device.
- The **CYW920820EVB-02** Evaluation Kit to evaluate and develop single-chip Bluetooth applications using CYW20820, an ultra-low-power dual-mode Bluetooth 5.0 wireless MCU device.  CYW20820 has an integrated power amplifier, which can transmit at 10 dBm transmit power.
- The EZ-BT Mesh Evaluation kit (**CYBT-213043-MESH**) to evaluate the Bluetooth SIG mesh functionality using the CYBT-213043-02 EZ-BT Bluetooth 5.0-qualified module.

# 3 CYW208XX Bluetooth SoC

**Figure 1** shows the block diagram of the CYW208XX device. See the **CYW20819 datasheet** and **CYW20820 datasheet** for more details on the CYW208XX SoC features.

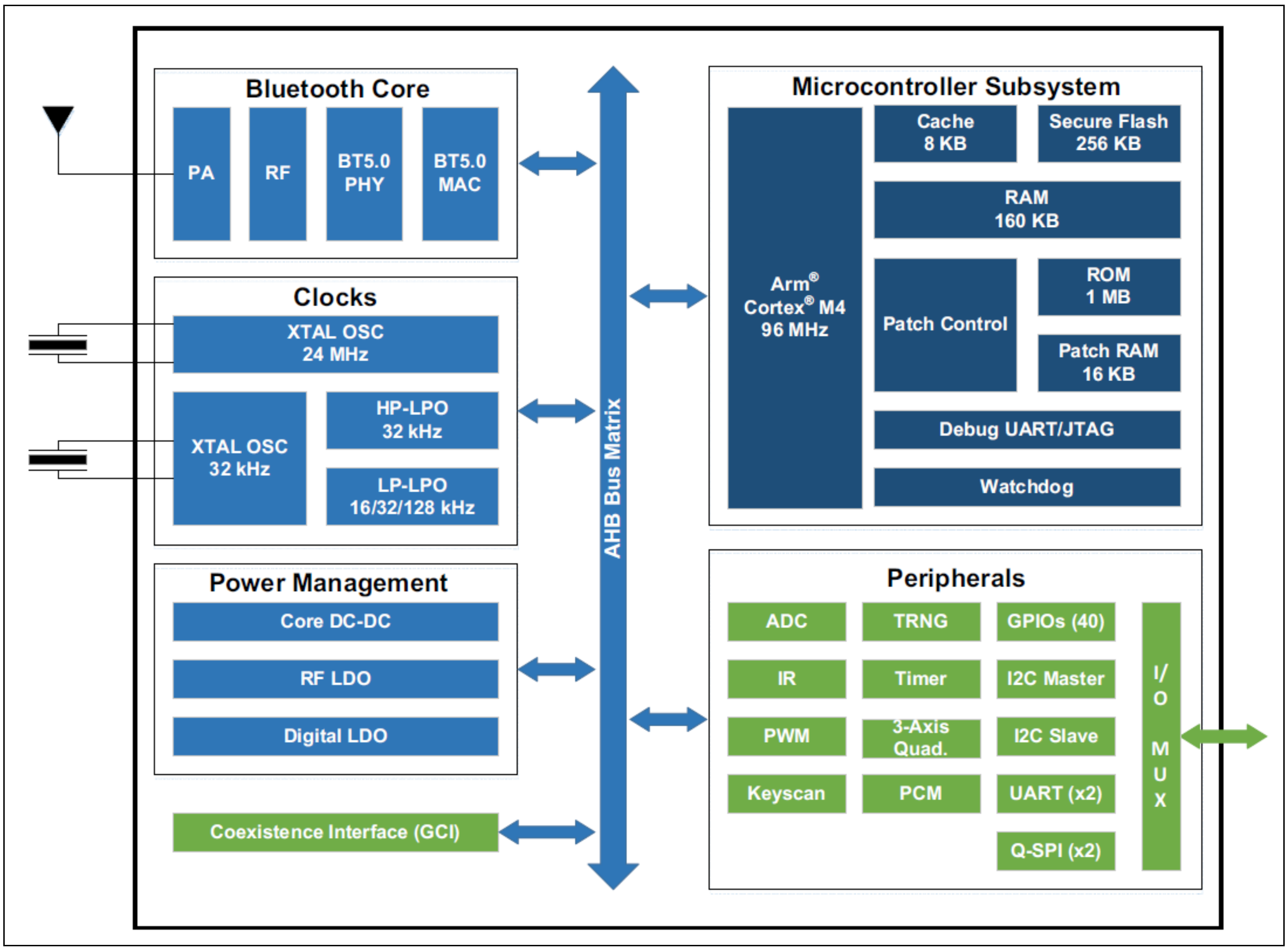


**Figure 1 Block Diagram of CYW208XX Bluetooth SoC**

# 4 CYW208XX SoC Peripherals

Hardware Abstraction Layer (HAL) drivers are available in the Bluetooth SDK to make use of several SoC peripherals such as ADC and I2C in CYW208XX.  HAL drivers in the Bluetooth SDK are blocking functions. To know more about the API implementation in the Bluetooth SDK, click the **API Documentation** link in the *README.md* file of wiced_btsdk or go to **Github Documentation**. Some of the SoC peripherals have dedicated code examples that demonstrate the functionality of the hardware blocks and are available to use in the Bluetooth SDK. More code examples are available in the **GitHub repository**.

## 4.1 General Purpose Input/Output (GPIO)

GPIOs are digital signal pins which can act as input or output depending on configuration at run time by the user. The Bluetooth SDK provides GPIO drivers in *wiced_hal_gpio.h*. CYW208XX supports two types of GPIOs: LHL GPIOs (up to 40) and Arm GPIOs (up to 10). The number of LHL GPIOs and Arm GPIOs available in the device depends on the package.

- The CYW20819A1KFB1G (112-ball FBGA) device package has 40 LHL GPIOs and 6 Arm GPIOs. The CYW20819A1KFBG (62-ball FBGA) device package has 22 LHL GPIOs available and 2 Arm GPIOs (DEV_WAKE and HOST_WAKE). Both LHL and Arm GPIOs are enumerated in `wiced_bt_gpio_numbers_t()` list in *wiced_hal_gpio.h* .
- The CYW20820A1KFB1G (112-ball FBGA) device package has 40 LHL GPIOs and 6 Arm GPIOs. The CYW20820A1KFBG (62-ball FBGA) device package has 22 LHL GPIOs available and 2 Arm GPIOs (DEV_WAKE and HOST_WAKE). Both LHL and Arm GPIOs are enumerated in  wiced_bt_gpio_numbers_t() list in *wiced_hal_gpio.h*.

LHL GPIOs are different from Arm GPIOs in a few ways:

- LHL GPIOs can operate in low-power modes and support muxable peripherals. Peripherals such as I2C, SPI, PUART, and ACLK, whose functionalities can be brought out to any LHL GPIOs are said to be Super-Muxable. See the datasheet for the list of peripherals that are Super-Muxable. LHL GPIOs are the ones usually used by most user applications.
- Arm GPIOs can operate only in active mode and they do not support muxable peripherals. DEV_WAKE, HOST_WAKE, and four BT_GPIOs are the Arm GPIOs available in the CYW20819A1KFB1G package, whereas DEV_WAKE and HOST_WAKE are the Arm GPIOs available in the CYW20819A1KFBG package. The DEV_WAKE can be used to wake the Bluetooth device with a signal from the host. The HOST_WAKE signal can be used to wake the host device with a signal from the Bluetooth device. BT_GPIOs can be configured as a GCI (Global Co-existence Interface) pin.

CYW208XX supports the following GPIO capabilities in Bluetooth SDK:

- Input/Output selection
- Pull-up/Pull-down selection
- Drive strength selection
- Hysteresis control
- Level/Edge triggered interrupt selection

The following are some of the most commonly used GPIO APIs and their descriptions:

- The `wiced_hal_gpio_configure_pin()` API is used to configure the GPIO capabilities on a specific GPIO .

- The `wiced_hal_gpio_register_pin_for_interrupt()` can be used to register an interrupt for GPIOs and based on edge or level changes it will trigger the callback function (i.e., interrupt service routine or interrupt handler).
- The `wiced_hal_gpio_clear_pin_interrupt_status()` should be used to clear the interrupt in the GPIO interrupt callback function.
- The `wiced_hal_gpio_select_function()` API can be used to map the functionality to specific pins (see the following code snippet as an example on how to use it). The first parameter to this function refers to the GPIO pin (`wiced_bt_gpio_numbers_t`) and the second parameter refers to the GPIO functionality (`wiced_bt_gpio_function_t`).

```
    wiced_hal_gpio_select_function (WICED_P03, WICED_PWM0);
```

For more details on these GPIO APIs, click **Components** > **Hardware Drivers** > **GPIO** and navigate to GPIO API reference in WICED API Reference Guide.

Though the SDK offers to configure all I/Os, the selection of pins in an application needs to be made based on the following constraints with respect to the device:

- Certain functionalities (for example, ADC input) are supported only on specific LHL I/Os and cannot be routed to other pins. See the Pin Assignments and GPIOs section in the **datasheet** for more details.
- GPIOs such as P26, P27, P28, and P29 can drive higher current compared to other pins. See the **datasheet** for more details.

## 4.2 Analog-to-Digital Converter (ADC)

The ADC block in CYW208XX is a single-ended switched-capacitor sigma-delta ADC with 12 bits for Audio1 measurements and 10 bits for DC measurements. The ADC has 32 DC input channels including 28 GPIO inputs. The ADC does not support Super-Muxing capability. For more details on the ADC APIs described, click **Components** > **Hardware Drivers** > **Analog-to-Digital Converter (ADC)** and navigate to ADC API reference in WICED API Reference Guide.

- The following 15 pins can be programmed as ADC inputs in CYW20819A1KFBG / CYW20820A1KFBG (62-ball FBGA):
  - P0, P1, P8-P15, P17, P28, P29, P32, P37
- The following 28 pins can be programmed as ADC inputs in CYW20819A1KFB1G / CYW20820A1KFB1G (112-ball FBGA):
  - P0, P1, P8-P19, P21-P23, P28-P38

The Bluetooth SDK provides APIs to read voltage/raw samples of VDDIO (Input / Output Voltage), VDD_CORE (Core Voltage), ADC_BGREF (Band-gap reference), and ADC_REFGND (Reference ground) in *wiced_hal_adc.h*. See the ADC Electrical Characteristics in the **datasheet** for more details.

CYW208XX supports two modes of operation: DC measurement mode and Audio[1] mode. For Audio[1] applications, the ADC uses a Programmable-Gain Amplifier (PGA) to control the gain.

1 The micn and micp pins are not available in CYW20819A1KFBG/CYW20820A1KFBG (62-ball FBGA) but are available in CYW20819A1KFB1G/CYW20820A1KFB1G (112-ball FBGA). DMA is available only for audio mode; DMA handling is done by ROM firmware. Audio mode is currently supported only in the HID profile library of the Bluetooth SDK. The ROM firmware takes care of handling HID voice commands.

### 4.2.1 DC Measurement Mode

The DC Measurement mode refers to single-ended voltage measurement on an input pin referenced to the ADC ground. From the ADC's block diagram, 28 GPIO channels, VDDIO, VDD_CORE, ADC_BGREF, and ADC_REFGND are multiplexed and fed into the switched-capacitor sigma-delta ADC.

The `wiced_hal_adc_init()` function initializes the necessary registers required for ADC operation in DC measurement mode. The ADC needs to be re-initialized if it has been powered down in specific low-power modes.

#### 4.2.1.1 ADC Output Data and Clock

The ADC clock input is 24 MHz, which is divided internally to 12 MHz in the ADC hardware. The typical conversion rate is 100 kHz and the minimum conversion rate is 50 kHz for static measurements of the ADC. The practical time for reading DC voltages after ADC initialization is approximately 150 microseconds. Similarly, the practical time for reading DC raw samples from the ADC is approximately 50 microseconds. The reduction in conversion rate is because the raw sample read function does not do any post processing on the samples, whereas when reading voltages, the firmware does move average filtering (low-pass FIR) on the sampled values for better accuracy. This conversion rate roughly translates to a range between 16 ksps and 20 ksps.

#### 4.2.1.2 ADC Input MUX Clocking in DC Measurement Mode

The ADC requires up to 2 microseconds to settle after switching of the DC input channels. It takes approximately 20 microseconds for the ADC REF to settle after powering up the ADC. The acquisition time for DC measurement mode is 10 µs. This is handled internally by the API functions. The ADC_DC_MUX_clk is the timing clock on the digital side to synchronize the DC input channel switching and measurement. **Figure 2** shows the MUX clocking and timing.

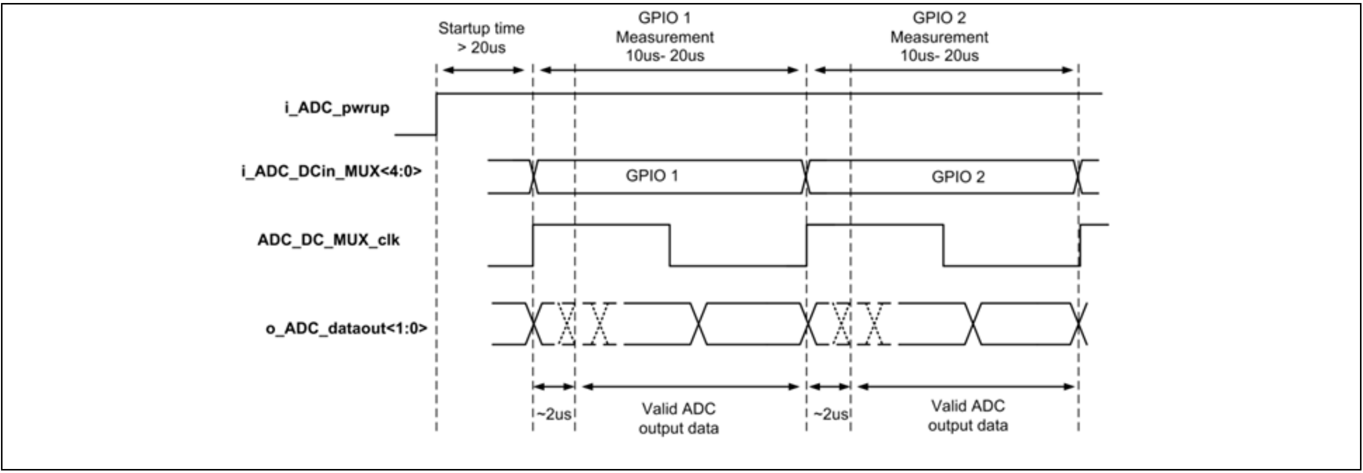


**Figure 2 Mux Clocking and Timing Diagram of ADC**

#### 4.2.1.3 ADC Sampling and Gain Calculation

The `wiced_hal_adc_read_voltage`() API function allows the program to read the analog voltage from the ADC. The ADC must be initialized before reading the voltage or else the function will return '0'. All read function calls to ADC registers are blocking. The ADC input and ADC output gain relationship in DC measurement mode can vary due to the supply voltages (ADC_AVDDBAT, MIC_AVDD, VDDC). For changes in supply voltages, the following API functions must be invoked to get more accurate ADC samples. These functions, when invoked, recalibrate the internal band gap and reference voltages.

- When the supply voltage is set to 1.8 V, use `wiced_hal_adc_set_input_range(ADC_RANGE_0_1P8V).`
- When the supply voltage is set to 3.3 V, use `wiced_hal_adc_set_input_range(ADC_RANGE_0_3P6V).`

## 4.3 Clocks

CYW208XX has multiple clock dividers and PLLs routed to several high-speed and low-speed peripherals. The figure below shows the simplified clock tree in CYW208XX.

The basic source for all the high-frequency clocks in CYW208XX is the 24-MHz external crystal (xtal). This is supplied to the ADPLL, which converts the frequency to 96 MHz. This clock is then supplied to most of the peripherals in the CYW208XX device like the Bluetooth core and CPU. Some peripherals like PWM can also take the 24-MHz clock directly. These peripherals have internal dividers that can divide the input clock according to its requirement.

The low frequency clock is 32 kHz and can be derived from multiple sources. See Low-Frequency Clock Sources for more details.

### 4.3.1 High-Frequency Clock Sources

The high-frequency clocks are derived from an external 24-MHz crystal oscillator using internal PLLs to upscale the frequency. These frequencies are determined by the hardware automatically, and are not user-configurable:

- HCLK: This clock powers the Resource Processing Unit (RPU). The RPU includes the CM4 CPU and DMA Controller. HCLK's frequency can range from 1 MHz to 96 MHz.
- Bluetooth Core Clock: This clock powers the Bluetooth RF hardware. The Bluetooth Core clock frequency can range from 1 MHz to 48 MHz.
- PTU Clock: This clock powers the Peripheral Transport Unit (PTU), which includes SPI, I2C, UART, PCM, and Random Number Generator. The PTU clock frequency can either be 24 MHz or 48 MHz. The peripherals will internally divide the clock frequency for their specific purpose. For example, the I2C master clock (SCL) can range from 100 kHz to 1 MHz.
- Timer Clock*:* The application timer (dual-input 32-bit timers) is clocked at 1 MHz.

### 4.3.2 Low-Frequency Clock Sources

The 32-kHz low-frequency clock (lhl_lpo_32kHz on the following figure) can be obtained from multiple sources. There are two internal low-power oscillators (LPOs) called the LP-LPO and HP-LPO and external crystal (OSC32K). The firmware determines the clock source to use among the available LPOs depending on the accuracy and power requirements. The preferred source is the external LPO (OSC32K) because it has good accuracy with the lowest current consumption. Internal LP-LPO has low current consumption and low accuracy whereas HP-LPO has higher accuracy and higher current consumption. The firmware assumes the external LPO has less than 250 PPM error with little or no jitter.

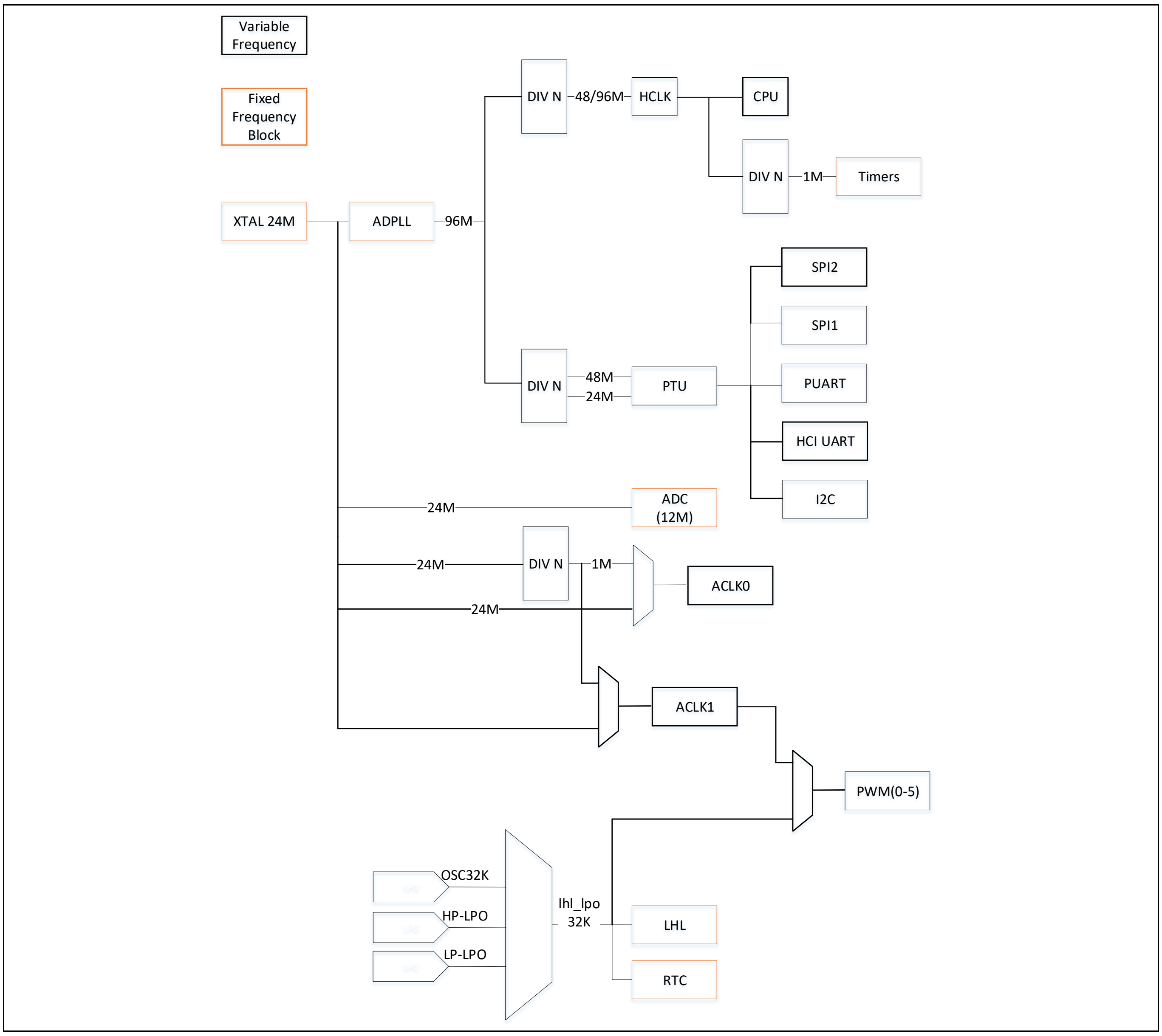


**Figure 3 Clock Diagram of CYW208XX**

## 4.4 Real-Time Clock (RTC)

CYW208XX supports a 48-bit RTC timer running on the 32-kHz crystal (XTAL32K) LPO. RTC supports a clock input from either an external or internal LPO. If an external LPO is not connected to CYW208XX, the firmware takes the clock input from the internal LPO for the RTC. The RTC timer is represented in the structure `wiced_rtc_time_t` with members representing seconds, minutes, hour, day, month and year. By default, the date and time are set to January 1, 2010 with a time of 00:00:00 denoting HH: MM: SS.

The Bluetooth SDK provides API functions to:

- Set the current time (`wiced_set_rtc_time()`).
- Get the current time (`wiced_rtc_get_time()`).
- Convert the current time value to a string (`wiced_rtc_ctime()`).

For more details on RTC APIs described above, click **Components** > **Hardware Drivers** > **RTCDriver** and navigate to RTCDriver API reference in the **Bluetooth-SDK Documentation page**.

## 4.5 Watchdog Timer (WDT)

The Watchdog Timer module in CYW208XX is based on a 32-bit down counter that is used to detect and recover from malfunctions. During normal operation, the device regularly resets the watchdog timer before the count value reaches zero to prevent it from elapsing, or timing out. If, due to a hardware fault or program error, the device fails to reset the watchdog, the timer will elapse and generate a system reset on time out. The process of resetting the watchdog timer's counter is referred to as "petting the dog". The default watchdog timeout is set to 2 seconds and watchdog petting is done in the idle thread. Production applications need not pet the watchdog. Any application that attempts to hold the CPU longer than 2 seconds will trigger the watchdog and reboot the system. To disable the watchdog timer, the `wiced_hal_wdog_disable()` API is available in *wiced_hal_wdog.h*.

The Bluetooth SDK provides limited debugging functionalities via the watchdog timer using *wiced_hal_wdog.h*. When the watchdog timer expires, the system will reset after a core-dump. The core-dump contains the following information: device firmware or hardware version, warning-flag, memory info, CPU/HW registers, SRAM, and Patch RAM image. The core dump is sent over the HCI-UART as Bluetooth-HCI vendor-specific events.

## 4.6 Application Timer

CYW208XX provides one general-purpose 32-bit dual-input timer. The firmware uses the 2 x 32-bit timers as a 1 x 64-bit timer. The Bluetooth SDK provides APIs to use the timer functionality in *wiced_timer.h*. At first, the application timers should be initialized to a specific seconds or milliseconds timer value, then the timer should be started and upon reaching the timeout, the callback function is executed. This callback function can hold any application code like LED blinking, serial data transfer, reading RTC value, reading ADC samples, and so on. The timer supports two modes of operation:

- Periodic Interrupt mode
- Single-shot mode

The Bluetooth SDK provides four timer types:

- WICED_SECONDS_TIMER
- WICED_MILLI_SECONDS_TIMER
- WICED_SECONDS_PERIODIC_TIMER
- WICED_MILLI_SECONDS_PERIODIC_TIMER

The two periodic timers trigger the timer callback function every time the timer count reaches zero (down counter) whereas the other two timers are single-shot. The timer callbacks execute only once and exit on every timer expiration. Some of the APIs which can help to get started with Timer APIs are as follows:

- The `wiced_init_timer()` function initializes the timer. It allows you to specify the callback function to be invoked when the timer expires. This function also allows you to specify the type of timer that should be used. This function does not start the timer.
- The `wiced_start_timer()` function is used to start the count down and to specify the count value. The `wiced_stop_timer()` function can be used to stop the timer.
- The `wiced_is_timer_in_use()` function allows you to know if the specified timer is currently in use by returning a Boolean value.

For more details on these Application Timer APIs, click **Components** > **Hardware Drivers** > **Timer Management Services** and navigate to Timer API reference in WICED API Reference Guide.

Here is a sample code snippet to use the application timer:

```
wiced_timer_t my_timer_handle; /* Typically defined as a global */
.
.
/* Typically, inside the BTM_ENABLED_EVT */
wiced_init_timer (&my_timer_handle,
   myTimer,
   0,
   WICED_MILLI_SECONDS_PERIODIC_TIMER);
wiced_start_timer (&my_timer_handle, 100);
.
. .
/* The timer function */
void myTimer (uint32_t arg)
{
/* Put timer code here */
}
```

## 4.7 Pulse-Width Modulator (PWM)

There are six 16-bit Hardware PWM channels available in CYW208XX, which support the Super-Muxable functionality. The *wiced_hal_pwm.h* file in the Bluetooth SDK defines PWM-specific drivers. LHL_CLK or PMU_CLK can be used as the clock source for each PWM channel. If the clock source is LHL_CLK, the clock frequency will be 32 kHz. PMU_CLK requires an auxiliary clock to be configured first. When configuring the auxiliary clock, ACLK0 is not available for use with the PWMs. Therefore, ACLK1 is the only available PMU_CLK running at 24 MHz or 1 MHz (using internal clock division). Each PWM channel can be routed to GPIO pins using the SuperMUX. The `wiced_hal_gpio_select_function()` is used to map any LHL GPIO to the specified PWM channel (`WICED_PWM`<x>) where 'x' represents the PWM channels. Glitches may occur during runtime change of PWM configuration parameters. This API was explained earlier in General Purpose Input/Output (GPIO). For more details on PWM APIs, clickn **Components** > **Hardware Drivers** > **Pulse Width Modulation (PWM)** and navigate to PWM API reference in the **Bluetooth-SDK Documentation page**.

- The `wiced_hal_aclk_enable()` function in *wiced_hal_aclk.h* enables the auxiliary clock and allows the functionality of choosing either 1 MHz or 24 MHz for routing to the PWMs.
- **Note:** CYW20820 does not support ACLK at 1 MHz as of Bluetooth SDK 1.3.
- The `wiced_hal_pwm_start()` function configures, enables, and starts the PWM and also routes it to a preconfigured GPIO pin. The desired GPIO pin must be configured as an output before assigning it for use with a PWM. This function takes in five parameters (i.e., channel, clock source, toggle count, initial count, and invert signal).
  - The channel refers to PWM channels 0 to 5.
  - The clock source can be either `LHL_CLK` or `ACLK1` (`PMU_CLK`).
  - The toggle count refers to the number of ticks to wait before toggling the signal.
  - The initial count is the initial value of the register.

  - The invert signal can be either 1 or 0. If the invert signal is 1, the PWM output starts at logic HIGH and if the invert signal is 0, the PWM output starts at logic LOW.
- The `wiced_hal_pwm_change_values()` function changes the PWM settings such as toggle count and initial count of a specific PWM channel after the PWM hardware block has already been started.
- The `wiced_hal_pwm_get_params()` is a utility function that calculates the PWM parameters. This function takes in three parameters: input clock frequency, duty cycle, and desired PWM output frequency to determine the required initial count and toggle count. The initial count and toggle count are determined by the function using the following expression:
  - Initial count = 0xFFFF - (Input clock frequency/PWM frequency output)
  - Toggle count = 0xFFFF - ((duty cycle in percentage) x (Input clock frequency/PWM frequency output)/ 100)

## 4.8 Random Number Generator (RNG)

The Bluetooth SDK provides an API to use the RNG hardware module. It provides functions to generate either a single 32-bit random number or fill a given array with 32-bit random numbers. These functions are useful for security-related applications such as authentication. These functions generate random numbers from an input called a “seed” that is taken from either the generating hardware block’s temperature when it is warm or from the Bluetooth clock when it is cold. The *wiced_hal_rand.h* file defines functions such as `wiced_hal_rand_gen_num()` to generate a 32-bit random integer and `wiced_hal_rand_gen_num_array()` to fill a given array with randomly generated 32-bit integers. The RNG hardware block takes 1 ms to generate random numbers (either a single number or an array) including the warm-up time. For more details on RNG APIs, click **Components** > **Hardware Drivers** > **Random Number Generator (RNG)** and navigate to PWM API reference in the **Bluetooth-SDK Documentation page**.

## 4.9 Inter-Integrated Circuit (I$^2$C)

CYW208XX provides one I2C-compatible Master interface to communicate serially with I2C Slave devices. The I2C block supports the Super-Muxable functionality.

The I2C module features include support for:

- 7-bit addressing mode
- Clock stretching as a Master
- SCL clock frequencies: 100 kHz (Standard Mode), 400 kHz (Fast Mode), 800 kHz (Proprietary), and 1 MHz (Fast Mode+)
- Multi-slave operation

### 4.9.1 I2C Clock

The I2C clock (SCL) is provided by the Master on the I2C bus. When the I2C module is in Master mode, the serial clock generator generates the SCL clock from the transport clock of 24 MHz. The `wiced_hal_i2c_set_speed()` function allows you to set the clock rate of SCL to any of the speeds listed in the introduction. The I2C block has 64 bytes of RX and TX FIFO for the transaction buffer, and thus each I2C transaction payload can be a maximum of 64 bytes.

The following enum constants are provided in *wiced_hal_i2c.h* to set the desired SCL clock frequency:

- `I2CM_SPEED_100KHZ`
- I2CM_SPEED_400KHZ
- I2CM_SPEED_800KHZ
- I2CM_SPEED_1000KHZ

For example, the enumerated value of `I2CM_SPEED_100KHz` is set to 240, which is the dividing parameter, that is, when 24,000,000 divided by 240, it results in 100 kHz. Trying to use Multi-master will result in undefined behavior because the clocks from two or more masters will not be synchronized. Also, note that the proprietary 800-kHz mode is not expected to work with all I2C Slaves. The `wiced_hal_i2c_get_speed()` function allows you to get the current clock frequency of the I2C block.

### 4.9.2 I2C Initialization

I2C module registers are initialized by calling `wiced_hal_i2c_init()`, which is defined in *wiced_hal_i2c.h*. This function call initializes the I2C driver and its private values. You must call this function before using I2C because some modules will turn the block OFF for power saving. This function call ensures that the clock signal to the hardware block is turned on with the default SCL clock frequency of 100 kHz. Therefore, the `wiced_hal_i2c_set_speed()` function must be called after the `wiced_hal_i2c_init()` function if you want to use a speed other than 100 kHz.

### 4.9.3 I2C Operations

The I2C Master module supports three operations: master write, master read, and combined write followed by read. These operations are discussed in the following sections.

#### 4.9.3.1 I2C Master Write

The `wiced_hal_i2c_write()` function writes data to a specific Slave address. Although any arbitrary length of data may be written to the Slave, atomic transactions greater than the hardware's capability are not possible and the data will be split into multiple transactions. This is a blocking call. When the I2C slave device acknowledges a successful I2C master write operation, the function returns I2CM_SUCCESS, and if the I2C master does not receive any acknowledgement from the I2C slave, the function returns I2CM_OP_FAILED.

#### 4.9.3.2 I2C Master Read

The `wiced_hal_i2c_read()` function reads data from a particular Slave address register into the given buffer. This is a blocking call. This I2C function returns I2CM_OP_FAILED when there is NACK (No Acknowledgement) from the I2C slave device or when the I2C slave returns 0 bytes. For more details, see the **Bluetooth-SDK Documentation**.

#### 4.9.3.3 I2C Master Combined Write and Then Read

The `wiced_hal_i2c_combined_read()` function executes a write transaction followed by a read transaction with a repeated start condition between the transactions. In the first transaction, data is written to the Slave address and after the repeated start, data is read from the Slave in the second transaction. This operation is usually used to read a Slave device's registers with the first write transaction specifying the Slave's register address that needs to be read. This I2C function returns I2CM_OP_FAILED when there is NACK (No Acknowledgement) from the I2C slave device or when the I2C slave returns 0 bytes.

## 4.10 Serial Peripheral Interface (SPI)

CYW208XX contains two SPI blocks used to communicate with other SPI Master and Slave devices. The CYW208XX SPI interface supports the Super-Muxable functionality. This block can be used to communicate with SPI-based sensors such as temperature sensors and motion sensors. SPI blocks support the following features:

- Three-wire (Master) and four-wire (Master and Slave) SPI interface
- Master and Slave modes

- Configurable SCK polarity and phase
- Configurable LSB- or MSB-first transfer

1024-byte transmit buffer and 1024 byte receive buffer for SPI 12 (shared with HCI UART)

- 64-byte transmit buffer and 64-byte receive buffer for SPI 2[2] (shared with PUART)

CYW208XX provides functions to choose between two SPI hardware blocks. `spi_interface_t` allows you to choose between `SPI1` and `SPI2`. Each SPI Utility function will have an argument to choose between the two SPI blocks.

The SPI block can be used as a generic Master or a generic Slave. Any LHL GPIOs can be used for the SPI interface. The Bluetooth SDK provides a list of parameters and functions to access the SPI driver. For more details on SPI APIs, click **Components** > **Hardware Drivers** > **PeripheralSpiDriver** and navigate to SPI API reference in the Bluetooth-SDK Documentation page.

The SPI block can be initialized in the required configuration using the `wiced_hal_pspi_init()` function. In generic Master mode, it is up to the firmware to assert the CS to various peripherals. The CS and INT pins can be used as GPIOs when in Master mode. The SPI clock and pins can be changed during application runtime when there is no activity in the SPI block. There are three types of Master transactions and two types of Slave transactions as described below.

### 4.10.1 SPI Master TX-Only Operation

When data only needs to be sent, the SPI block is in TX-Only mode.  In this mode:

- TxFIFO must be enabled
- RxFIFO must be disabled

In this configuration, the SPI block will transmit any data placed in the TxFIFO. The `wiced_hal_pspi_tx_data()` function is used to place the data in the TxFIFO to transmit the data as a Master.

When the TxFIFO becomes empty, SPI_CLK is paused at a byte boundary until more data is placed in the TxFIFO using the `wiced_hal_pspi_tx_data()` function.  No data is received in the RxFIFO.

### 4.10.2 SPI Master RX-Only Operation

When data only needs to be received, the SPI block is in RX-Only mode.  In this mode:

- TxFIFO must be disabled
- RxFIFO must be enabled

In this configuration, the SPI block will retrieve specified bytes of data from the Slave. The `wiced_hal_pspi_rx_data()` function is used to receive the data as a Master.

The data stream will be paused if the RxFIFO becomes full. The data stream will resume when there is space in the RxFIFO. No data is sent in this mode.

### 4.10.3 SPI Master Full-Duplex Operation

When data needs to be sent and received simultaneously, the SPI block is in full-duplex mode.  In this mode:

2 The SPI 1 block and HCI UART use the same buffer, so they cannot be used at the same time. The SPI 2 block and PUART use the same buffer, so they cannot be used at the same time.

- TxFIFO is enabled
- RxFIFO is enabled

In this configuration, the SPI block will transfer data to/from both FIFOs if there is:

- Space in RxFIFO
- Data in TxFIFO

If either of these conditions becomes false, SPI_CLK is paused until both conditions become true again. The `wiced_hal_pspi_exchange_data()` function can be used to send and receive data simultaneously.

### 4.10.4 SPI Slave TX-Only Operation

As a generic slave, data is transferred to the host when:

- There is data in the TxFIFO
- The TxFIFO is enabled
- The host toggles SPI_CLK and SPI_CSN is asserted

If condition (1) is false, but conditions (2) and (3) are true, a data underflow condition occurs. The `wiced_hal_pspi_slave_tx_data()` function can be used to put the data in the TxFIFO.

### 4.10.5 SPI Slave RX-Only Operation

As a generic slave, data is received from the host when:

- There is space in the RxFIFO
- The RxFIFO is enabled
- The host toggles SPI_CLK and SPI_CSN is asserted

If condition (1) is false, but conditions (2) and (3) are true, a data overflow condition occurs. The `wiced_hal_pspi_slave_rx_data()` function can be used to receive the data.

### 4.10.6 SPI Slave Full-Duplex Operation

When data needs to be sent and received simultaneously, the SPI block is in full-duplex mode.  In this mode:

- TxFIFO is enabled
- RxFIFO is enabled

In this configuration, the SPI block will transfer data to/from both FIFOs if there is:

- Space in RxFIFO
- Data in TxFIFO

To send and receive data simultaneously, enable the Tx and Rx buffers, use the `wiced_hal_pspi_slave_tx_data()` API to place data in the buffer and read the data from the Rx buffer using `wiced_hal_pspi_slave_rx_data()` after the Master transfers the data.

## 4.11 UART

This device has two UART blocks: Peripheral UART (PUART) and HCI UART (Bluetooth UART)

### 4.11.1 Peripheral UART (PUART)

CYW208XX's PUART can be used as a generic Tx/Rx UART block in addition to the debug printing functions. CYW208XX can map the peripheral UART to any LHL GPIOs. PUART is clocked at 24 MHz with configurable baud rate and parity[3]. Both PUART RX and PUART TX have a 256-byte FIFO.

PUART has the following features:

- 8-bit transfer up to 3 Mbps
- 9-bit transfer including stop bit up to 2.5 Mbps
- Configurable flow control
- Interrupt functionality on receive operation

#### 4.11.1.1 PUART Initialization

The `wiced_hal_puart_init()` function defined in *wiced_hal_puart.h* is used to initialize the PUART block.

The `wiced_hal_puart_select_uart_pads()` function can be used to select TX/RX and optional CTS/RTS pins for the PUART hardware to use.

#### 4.11.1.2 PUART Baud Rate

The `wiced_hal_puart_set_baudrate()` function is used to set the baud rate. The default PUART baud rate is 115200. Typical baud rates include 115200, 921600, 1500000, and 3000000 bps, although intermediate speeds are also available. For more details on the baud rate, see the datasheet.

#### 4.11.1.3 PUART Transmit

The `wiced_hal_puart_enable_tx()` and `wiced_hal_puart_disable_tx()` functions enable or disable transmit capability of the PUART, respectively. To send data over PUART, transmit must be enabled.

The `wiced_hal_puart_print()` function is used to send a string of characters via the TX line.

The `wiced_hal_puart_write()` function is used to send one byte via the TX line.

#### 4.11.1.4 PUART Receive

The `wiced_hal_puart_enable_rx()` function is used to enable receive capability.

The `wiced_hal_puart_rx_fifo_not_empty()` function is used to check if there is any data available in the RX FIFO.

The `wiced_hal_puart_register_interrupt()` function provides an interrupt callback on receive of data.

For receiving data, register an interrupt callback function, set the watermark to determine how many bytes should be received before an interrupt is triggered, and enable Rx. A code snippet for receiving data on PUART is given below.

```
wiced_hal_puart_register_interrupt(rx_interrupt_callback);
/* Set watermark level to 1 to receive interrupt up on receiving each byte */
```

[3] Subject to availability of drivers in Bluetooth SDK. As of Bluetooth SDK v1.3, the SDK does not support parity selection.

```
wiced_hal_puart_set_watermark_level (1);
wiced_hal_puart_enable_rx ();
```

The Rx processing is done inside the interrupt callback function. The PUART interrupt must be cleared inside the callback function to receive additional characters.

```
void rx_interrupt_callback (void* unused)
{
      uint8_t readbyte;
      /* Read one byte from the buffer and then clear the interrupt */
      wiced_hal_puart_read (&readbyte);
      wiced_hal_puart_reset_puart_interrupt ();

      /* Add your processing here */
}
```

#### 4.11.1.5 PUART Flow Control

The `wiced_hal_puart_flow_on()` and `wiced_hal_puart_flow_off()` functions enable or disable flow control on the  PUART, respectively.

### 4.11.2 Host-Controller Interface UART

This UART is used for HCI transport for controller mode and is also used for programming the device. The HCI UART signals are fixed to specific I/O pads. DMA is used for HCI UART packets when the length of the data to be received is greater than around 64 bytes.

To understand the programming sequence over the HCI UART interface, see *WICED-HCI-Control-Protocol.pdf* in the GitHub **btsdk-docs** repository.

## 4.12 Keyboard Scanner (Keyscan)

The keyboard scanner is a matrix scanner hardware peripheral which can process key press and release events from the GPIOs. The keyboard scanner is implemented as a firmware queue on top of a Hardware FIFO. For the architecture of the keyboard scanner, see **Figure 4**.

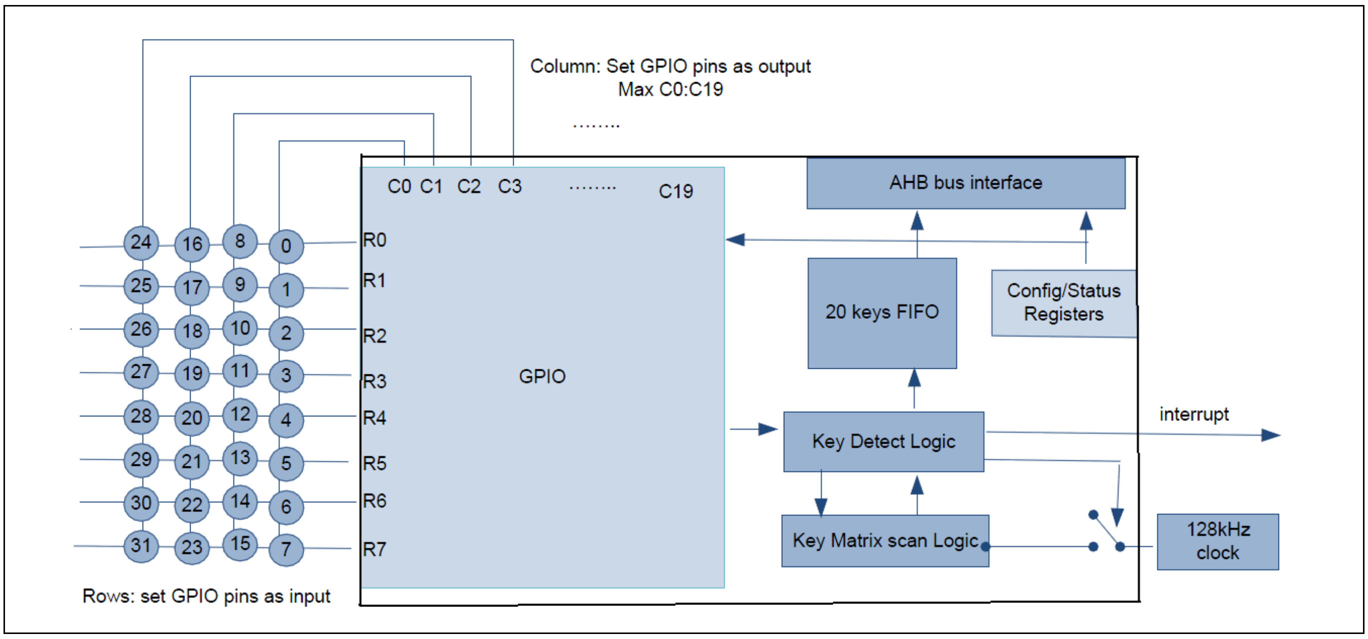


**Figure 4 Keyboard Scanner for CYW208XX**

Following are the specifications for the keyboard scanner hardware block:

- The maximum number of rows (Key Scan Inputs (KSIs)) that can be configured is 8.
- The maximum number of columns (Key Scan Outputs (KSOs)) that can be configured is 20.
- All the column pins to the keyboard scanner are Super-Muxable.
- The keyboard scanner block samples the entire 160 keys at 128 KHz.
- Hardware debouncing and ghost key detection are available.

The keyboard scanner in CYW208XX is designed to autonomously sample keys and store them into buffer registers without the need for the host microcontroller to intervene. Key up/down events are a byte stream coming from the API driver. The ROM firmware takes care of the state machine handling of the keyboard scanner hardware. The state machine begins in idle state and will enter scan state if any key is pressed. During the scan state, a row counter counts from 0 to programmable number of rows minus 1 and programmable number of columns minus 1. See the CYW208xx datasheet to know the maximum number of rows and columns that are available in the specific package.

The APIs needed for configuring and utilizing the Keyboard scanner are available in *wiced_hal_keyscan.h*. The Keyboard scanner API provides the functions for following capabilities:

- Configures the number of rows and columns for the keyboard scanner
- Checks for any new key press events and pending key press events
- Enables/disables ghost key detection
- Resets the keyboard scanner hardware block
- Provides wake up source for the device

See the API reference guide for more details on the function prototypes and parameters. The application developer can use the device configurator tool in the ModusToolbox to configure the keyboard scanner hardware.

# 5 Bluetooth Low Energy Radio Features

This section describes some of the Bluetooth Low Energy radio features available in CYW208XX. The CYW208XX devices provide a highly integrated low-power 2.4-GHz radio transceiver with support for multiple modulations and packet formats. CYW20820 adds an integrated power amplifier which can transmit at 10 dBm transmit power. The MCU subsystem provides an interface between the peripherals and the radio, which makes it possible to issue commands, read status, and automate and sequence radio events. The Arm Cortex-M4 processor handles the data to and from the analog RF and baseband circuitries and assembles the information bits in a given packet structure.

## 5.1 TX Power Control

CYW20819 has a maximum power output is +4dbm for class 2 operation. It supports TX power from +4dBm to -16dBm. The resolution is 4dBm, that is, the configurable TX power levels are {+4 dBm, 0 dBm, -4 dBm, -8 dBm, -12 dBm, -16 dBm}.

CYW20820 has an internal power amplifier with a maximum power output of +10dbm for class 2.5 operation. It supports TX power from +4dBm to -24dBm. The resolution is 4dBm, that is, the configurable TX power levels are {+4 dBm, 0dBm, -4 dBm, -8 dBm, -12 dBm, -16 dBm, -20 dBm, -24 dBm}.

The Bluetooth SDK provides the following function for controlling the TX power:

- `wiced_bt_set_tx_power()`: This function defined in *wiced_bt_dev.h* can be used to set the TX power on data channels for a particular connection. It takes the peer device's BD (Bluetooth Device) address as an input parameter. This API can also be used to set the TX power on Advertisement channels by passing the `BD_ADDR` as 0.

## 5.2 Coexistence - SECI

Serial Enhanced Coexistence Interface (SECI) is a proprietary Cypress interface in Cypress-to-Cypress Coexistence solutions. The `WICED_GCI_SECI_IN` and `WICED_GCI_SECI_OUT` functionality need to be routed to any of the available GPIOs in CYW208XX. The `wiced_bt_coex_enable()` and `wiced_bt_coex_disable()` functions defined in *wiced_bt_dev.h* allow you to enable and disable the Coexistence interface. See **AN214852** for Coexistence interfaces and how to use them with CYW208XX.

# 6 Software

This section describes the application code build and download process in CYW208XX Bluetooth devices.

## 6.1 Application Code

All Bluetooth wireless systems work the same basic way. The programmer writes the application firmware that calls Bluetooth APIs in the network stack. The stack then talks to the radio hardware, which in turn sends and receives data. When an event happens in the radio hardware, the stack will initiate actions in the application firmware by creating events (for example, when it receives a message from the other side of the Bluetooth connection). The application code is responsible for processing these events that are pushed up by the stack and doing the necessary post processing for each event. The ROM is preprogrammed with a boot sequence, device driver functions, low-level protocol stack components, and a bootloader. The ROM firmware in CYW208XX handles the Bluetooth stack activities and processes the radio events. The ROM firmware works like a Finite-State Machine (FSM) in handling the Bluetooth events. As an application developer, you can ignore the low-level Bluetooth functionalities.

A minimal C file for an application will look something like this:

```
#include "wiced.h"
#include "wiced_platform.h"
#include "sparcommon.h"
#include "wiced_bt_dev.h"
/****************************** Function Prototypes *******************/
wiced_result_t bt_cback (wiced_bt_management_evt_t event,
wiced_bt_management_evt_data_t *p_event_data);
/****************************** Functions *****************************/
/* Main application. This just starts the Bluetooth stack and provides the
callback function.
* The actual application initialization will happen when stack reports that
Bluetooth device is
* ready.
*/
APPLICATION_START ()
{
      /*Add initialization required before starting Bluetooth stack here */
      wiced_bt_stack_init (bt_cback, NULL, NULL); /* Register Bluetooth stack
      callback */
}
/* Callback function for Bluetooth events */
wiced_result_t bt_cback (wiced_bt_management_evt_t event,
wiced_bt_management_evt_data_t *p_event_data)
{
      wiced_result_t result = WICED_SUCCESS;
      switch (event)
      {
            /* Bluetooth stack enabled */
            case BTM_ENABLED_EVT:
            /* Initialize and start your application here once the Bluetooth
            stack is running */
                  break;

            default:
                  break;
        }
        return result;
}
```

The purpose of an RTOS is to reduce the complexity of writing embedded firmware that has multiple asynchronous, response-time critical tasks that have overlapping resource requirements. The RTOS maintains a list of tasks known as threads that are in different states such as active, inactive, or scheduled, and executes these tasks based on their priorities. Multi-threaded applications that make use of SoC peripherals should handle thread synchronization in the application when there are resource conflicts such as shared memory, mutual exclusion, hold and wait, and circular wait.

The RTOS in the ROM firmware creates multiple threads immediately after the bootup for handling the Bluetooth functionality and then gives control to the application thread. Currently, ModusToolbox supports multiple RTOSs. The device ROM has ThreadX by Express Logic built in and the license included, which makes this the best choice for developing applications with CYW208XX. To simplify using multiple RTOSs, the Bluetooth SDK has a built-in abstraction layer that provides a unified interface to the fundamental RTOS functions. You can find the documentation for the WICED RTOS APIs in **Components** > **RTOS** in the **Bluetooth-SDK Documentation page**.

## 6.2 Low-Power Capabilities

**AN225270** provides guidelines on designing and developing a low-powered application using CYW208XX and the parameters to consider for a power-efficient design.

## 6.3 Firmware Architecture

The CYW208XX device can either be used in controller mode or embedded mode.

In controller mode, the CYW208XX device runs the Bluetooth controller stack and the Bluetooth host stack is run on an external host MCU. The CYW208XX device uses the Host Controller Interface (HCI) to communicate with the host controller.

In embedded mode, both the Bluetooth host stack and the controller stack are run on the CYW208XX device. All the components of the controller stack and most of the components in the host stack reside in the device ROM. The user application, which can call APIs to access the ROM code, is programmed into the flash. If no code is programmed into the flash and the device is powered ON, it behaves like a Bluetooth controller that is controlled by HCI.

The CYW208XX device uses “patches” to ROM code to modify the behavior and fix bugs. These patches can be built into a core patch file (called general patch in the remaining part of this document) that is included in all ModusToolbox builds, or they can be built into optional patch libraries (called patch libraries in the remaining part of this document) that are application-specific, such as patches that pertain to audio or mesh network applications only. This approach optimizes the use of RAM by using only the patch libraries that are needed for a given application.

Cypress Bluetooth devices support Over-The-Air (OTA) updates of the application code. More details on OTA can be found in the *WICED-Firmware-Upgrade-Library.pdf* and *WICED-Secure-Over-the-Air-Firmware-Upgrade.pdf* in the GitHub **btsdk-docs** repository.

Programming the device does more than just copy the patch and application code to flash. It also uses commands to configure the structured data, cause the CPU to make function calls, configure hardware registers, etc. The default way to download firmware is through boot loading over the HCI UART. The device ROM contains a piece of code which can accept data through HCI commands from an external device (PC or MCU) and store it in flash. When the application code is downloaded, the host MCU (or PC) first transfers a piece of code called minidriver to SRAM; the minidriver then accepts further data through HCI commands and stores it to on-chip flash (OCF). More details about the minidriver can be found later in this document.

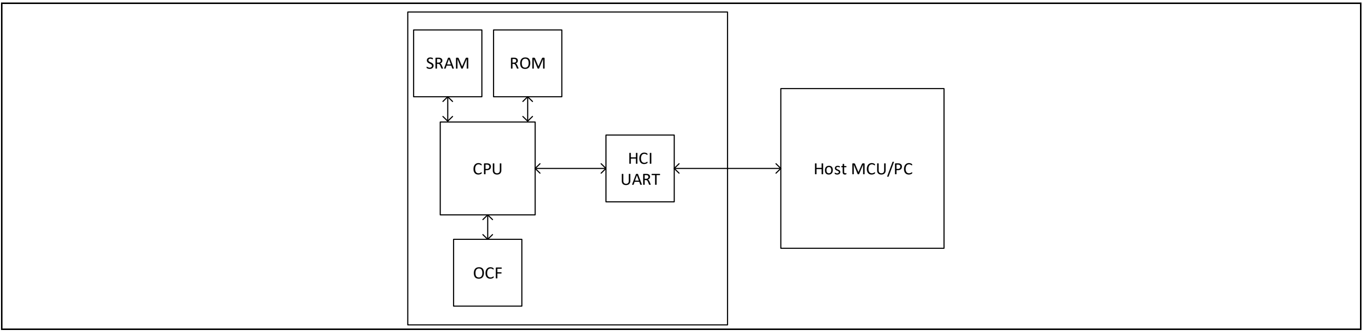


**Figure 5 CYW208XX Firmware Download Flow**

## 6.4 Memory Layout

The CYW208xx SoCs consist of various physical memories as discussed in Introduction. This section discusses in detail the various memories of the CYW208xx SoCs. ROM and Flash are the nonvolatile physical memories whereas SRAM provides the significant portion of the volatile memory in the CYW208xx. The utilization of patch RAM to modify the ROM code was discussed in Firmware Architecture.

The nonvolatile memory for the Bluetooth device (on-chip flash or external serial flash) is divided into five logical sections as shown in **Figure 8**. **Table 1** gives details about each section of the memory layout. See **AN216403 - ModusToolbox Application Buffer** for details on application buffer pool requirements of the Bluetooth Stack.

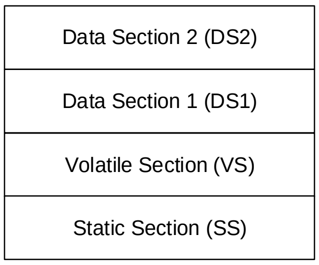


**Figure 6 Memory Organization of Nonvolatile Memory**

**Table 1 Non-Volatile Memory Organization, On-chip Flash**

| Section Name | Offset | Length | Description |
|---|---|---|---|
| Static Section (SS) | 0x500000 | 0x2000 | Static section used internally by chip firmware. This typically contains configuration records that don't change after factory programming. |
| Volatile Section (VS) | 0x500400 | 0x1000 | First volatile section used by the application and the stack to store data in the external or on-chip flash memory |
| Data Section (DS1) | 0x501400 | 0x1F600 | First partition. Used to store application code |
| Data Section (DS2) | 0x520A00 | 0x1F600 | Second partition. Used to store application code |

These values are indicative, and the actual values can be found in the BTP file which is discussed next.

The logical sections mentioned in **Table 1** are aligned with the erase sector boundaries of the memory. These details can be found in *.btp files located at *\<mtw_path>\wiced_btsdk\dev-kit\baselib\<device>\platforms\208XX_OCF.btp*. More details on the memory layout can be found in the *WICED-Firmware-Upgrade-Library.pdf* file in the GitHub **btsdk-docs** repository.

### 6.4.1 Significance of the *.btp File

The **.btp* file contains information used by the download tools such as the minidriver location, baud rate, memory section addresses and other details. It also contains information such as the BD ADDRESS which can be changed while programming the device. Some of the parameters are:

- DLConfigCrystalFreqMHz X 10000 – Frequency of the crystal connected to the device
- DLConfigSerialControlBaudRate – UART baud rate used for programming
- ConfigDSLocation – DS1 location offset
- ConfigDS2Location – DS2 location offset
- DLConfigSSLocation – SS location offset
- DLConfigBD_ADDRBase – BD address

## 6.5 Programming

As explained earlier, the CYW208XX device can typically be used in two configurations – Controller mode or Embedded mode.

In controller mode, the host typically downloads some code like the general patch to the Bluetooth device's SRAM or OCF. If the code is downloaded to SRAM, it needs to happen on each boot as the SRAM contents are lost on reset.

In embedded mode, the device embedded stack is used for some of the host functionality and the application code needs to be downloaded to the device's OCF.

### 6.5.1 File Formats

Application code is usually kept in *.hex* or *.hcd* format. The **.hcd* file is typically used for SRAM downloads and **.hex* files are typically used for flash downloads. As an intermediate step, the source code along with some configuration values are converted to *.cgs* (configuration source) format and then converted to .hex or .hcd format. The *.hcd* file is useful when an external MCU needs to program the Bluetooth device in controller mode as the file size is smaller and the payload format is similar to the HCI commands. Intel *.hex* file is used when programming to flash. The following diagram shows the file conversion process.

These files must be parsed and converted to HCI commands to download to the device. In ModusToolbox this is done by the ChipLoad tool.

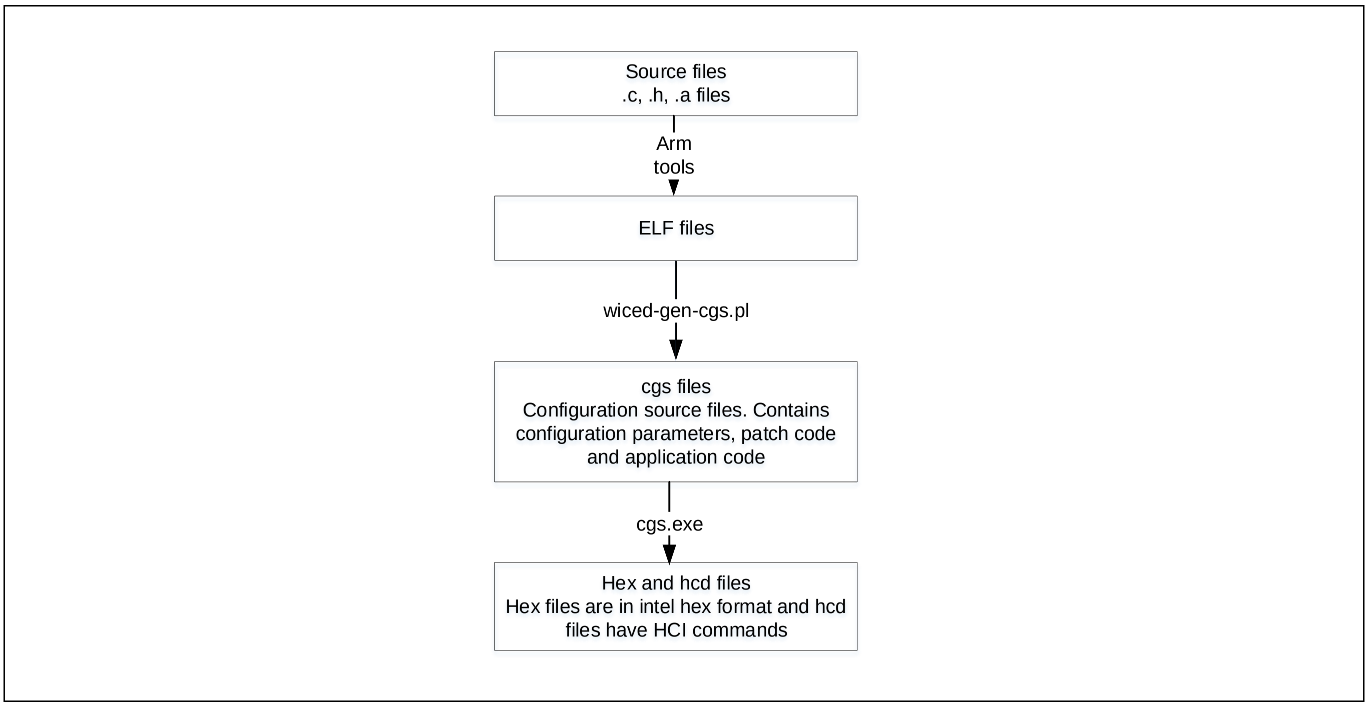

Source files
.c, .h, .a files
Arm
tools
ELF files
wiced-gen-cgs.pl
cgs files
Configuration source files. Contains configuration parameters, patch code and application code
cgs.exe
Hex and hcd files
Hex files are in intel hex format and hcd files have HCI commands

**Figure 7 File Formats**

### 6.5.1.1 Details of *.hcd Format

The *hcd* files are binary files that can be parsed and interpreted directly as HCI commands. The *hcd* files have all the commands in binary format except the packet type which needs to be added by the host. The format of the *hcd* files are as follows:

- The first two bytes are the command identifier. For example, the HCI_WRITE command is represented by 0x4C, 0xFC.
- The next byte is the command payload length. For example, 0x06 indicates that six more bytes will follow to complete the command.
- The command payload.

To convert the file to HCI commands, only the HCI packet type needs to be added. For example, when sending a command from the host, 0x01 should precede any HCI command to indicate that it is a command rather than an event.

The following WRITE_RAM command is an example:

01 4C FC nn xx xx xx xx yy yy yy …

In this WRITE_RAM command:

- nn is 4 + N, which represents four address bytes plus N payload bytes.
- xx xx xx xx is the 4-byte, absolute RAM address.
- yy yy yy … are the N payload bytes to be loaded into the addressed RAM location.

The following response to each WRITE_RAM command is expected within 200 ms:

04 0E 04 01 4C FC 00

#### 6.5.1.2 Details of *.hex Format

The *.hex* file follows the Intel I32HEX conventions. The format consists of records delimited by ASCII carriage return and line feed (0x0D, 0x0A). The format of the *.hex* files is as follows:

- First character is the start code ':' (ASCII 0x3A)
- Next byte indicates payload length. For example, 'FF' indicates 255.
- Next four bytes indicates 16 address bits. For example, '1000' represents address 0x1000. The type of address depends on the type of command.
- Next byte indicates the type of command:
  - '00' is for data and the address field represents the lower 16 bits of the destination address.
  - '01' indicates end of file. The payload is 0 and the address field is set as '0000'.
  - '04' indicates extended address. The address field indicates the lower 16 bits of the address and the payload represents the higher 16 bits of the address.
  - '05' for a 32-bit address payload. The address field is set as '0000', the payload length is set as '04' and the payload is interpreted as a 32-bit address. This is often used to indicate a LAUNCH_RAM (described later) destination.

#### 6.5.1.3 Details of *.cgs Format

The *.cgs* file contains configuration parameters, patch code, and application code. The *.elf* file generated from the application source code is converted to .cgs format and is appended to the patch *.cgs* file located at *\<mtw_path>\wiced_btsdk\dev-kit\baselib\208XXA1\internal\<device>\patches\patch.cgs*.

The definition of the configuration parameters in the *.cgs* file can be found in *\<mtw_path>\wiced_btsdk\dev-kit\baselib\208XXA1\internal\<device>\configdef<device>.hdf.*

*Note: <device> - The device in the directory path stands for 20819A1 or 20820A1.*

Parameters such as the crystal frequency, UART configuration, BD_ADDR, local name, and RSSI configuration can be modified using the patch .cgs file.

### 6.5.2 File Generation During Build

The file conversion process is as follows:

1. The source files (*.c*, *.h*) are compiled and converted to *.o* (object) files using GNU tools.
2. Library sources are linked to library *.a* files.
3. These along with prebuilt library files are then linked to a .elf (executable and linkable format) file by GNU tools.
4. The *.elf* file is converted to *.cgs* (configuration source) file using the *\<mtw_path>\wiced_btsdk\dev-kit\baselib\<device>\make\scripts\wiced-gen-cgs.pl script.* The ROM patch code and platform *.cgs* files are also appended.
5. The *\<mtw_path>\wiced_btsdk\dev-kit\btsdk-tools\<OS>\CGS\cgs.exe* tool is then used to convert the *.cgs* file to Intel hex format. The *cgs.exe* application takes inputs such as the *.cgs* file created in the previous step and .btp file. The *cgs.exe* application can take runtime arguments such as BD ADDRESS and overwrite the BD ADDRESS in the *.btp* file. This can be useful while programming multiple devices with different addresses.
6. The hex file is converted to an *hcd* file using the tool *\<mtw_path>\wiced_btsdk\dev-kit\btsdk-tools\<OS>\IntelHexToBin\IntelHexToHCD.exe*.

### 6.5.3 Recovery Process

Sometimes, the CYW208XX device may enter an unknown state or the HCI baud rate might have been changed from default. In such cases, it is not possible to program the device using HCI UART. To program the device correctly, the device needs to be put into the recovery mode. This is done by asserting the HCI UART CTS line while resetting the device.

When the Bluetooth device is reset, it checks for status of the HCI UART CTS pin (recovery pin) during power-ON. If the recovery pin is asserted (LOW) during reset, then the device enters what is called recovery mode. This mode will attempt to detect the UART baud rate by checking the RX line for the bit pattern of an HCI_RESET command. When detected, the HCI_RESET response is given at the same baud rate.

In this mode, most of the HCI commands will have no response. To download to the device in this mode, ignore the 'no response' to HCI_DOWNLOAD_MINIDRIVER and proceed with the download procedures. If the CTS pin is high after reset, then the device will check the OCF and apply any stored configuration, typically ending in a mode ready to accept all HCI commands at a default baud rate. If no configuration is available, the device will enter autobaud mode. Note that the application code can be directly loaded to the SRAM and executed or to the serial flash and then executed on next boot up.

### 6.5.4 Minidriver

The minidriver is separate FW independent of the ROM code. It is used to download the application code to the on-chip flash. The minidriver contains code to interpret certain HCI commands such as WRITE_RAM and READ_RAM and execute those commands. To download the code to the on-chip flash, first the minidriver needs to be written and executed from SRAM, which will interpret further HCI commands and write the code to the correct location in flash. As soon as the code download is complete, the minidriver is discarded because it is no longer needed. The minidriver will differ if the interface changes from UART to SPI or something else. The default minidriver can be found at *\<mtw_path>\wiced_btsdk\dev-kit\baselib\<device>\platforms\minidriver-<device>-uart-patchram.hex*.

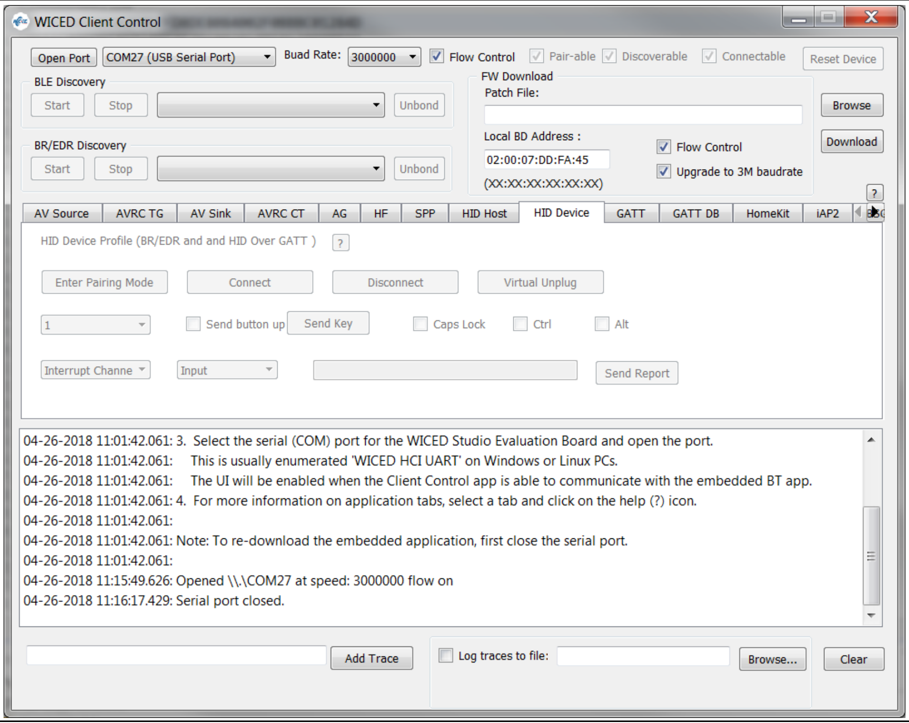
## 6.6 Tools for Programming

To understand the download process and HCI commands flow in detail, see *WICED-HCI-Control-Protocol.pdf* in the GitHub **btsdk-docs** repository.

**ClientControl**

ClientControl is a WICED application that acts as a Bluetooth host on a PC. It connects to the Bluetooth device using HCI UART and communicates via the WICED HCI protocol. This application can be used to test the CYW208XX device by sending commands to start/stop advertisements, send connection request, test HID profile, etc. as well as download the firmware to the device SRAM. Note that because this tool downloads to the device SRAM, it doesn't require any minidriver.

To launch the ClientControl application for Windows, macOS, and Linux select the Bluetooth-SDK application in ModusToolbox, and then click **Quick Panel** > **Tools** > **ClientControl**.



```
04-26-2018 11:01:42.061: 3.  Select the serial (COM) port for the WICED Studio Evaluation Board and open the port.
04-26-2018 11:01:42.061:     This is usually enumerated 'WICED HCI UART' on Windows or Linux PCs.
04-26-2018 11:01:42.061:     The UI will be enabled when the Client Control app is able to communicate with the embedded BT app.
04-26-2018 11:01:42.061: 4.  For more information on application tabs, select a tab and click on the help (?) icon.
04-26-2018 11:01:42.061:
04-26-2018 11:01:42.061: Note: To re-download the embedded application, first close the serial port.
04-26-2018 11:01:42.061:
04-26-2018 11:15:49.626: Opened \\.\COM27 at speed: 3000000 flow on
04-26-2018 11:16:17.429: Serial port closed.
```

**Figure 8 ClientControl Window**

### 6.6.1 ChipLoad

ChipLoad is a command-line utility used to download the firmware to the Bluetooth device. It takes inputs such as the *.hex* file and *.btp* file to get the download data and download configuration respectively. It can be found at \\<*mtw_path*>\*wiced_btsdk*\*dev-kit*\*btsdk-tools*\\<*OS*>\*ChipLoad*\\.

Note that the ChipLoad tool supports the Intel hex format.

### 6.6.2 DetectAndId

The DetectAndId tool performs two tasks: Detect and ID. At first, this tool tries to detect the serial ports on which Cypress Bluetooth devices are present. This is done by sending an HCI_RESET command on available ports at various baud rates and waiting for appropriate response. In case the serial port is already given as an input to this tool, it skips the port detect part.

The second task is to ID the connected device. This is done by sending a WICED HCI command to read the chip ID. The received ID is then compared to the expected chip ID present in the chip specific ID file in this path <*mtw_path*>\*wiced_btsdk*\*dev-kit*\*baselib*\\<*device*>\*platforms*\*CYW208XXA1_IDFILE.txt*. This ensures that only the right device is programmed. This can be used by other tools to download the application code.

### 6.6.3 Cypress Programmer

Cypress Programmer is a GUI application that can be used to program the CYW208xx based evaluation boards. It provides options to program, erase, and verify the flash on the CYW208xx. The Cypress Programmer expects the **.hex* file that is generated by the build process of the Bluetooth application. On selecting the HCI UART port, you can program the board. Please refer to the Cypress Programmer reference guide and release notes for more details.

# 7 Summary

CYW208XX is the optimal solution for applications in wireless input, wearables, medical and beacon applications for Internet of Things (IoT) devices. To facilitate design of cost-sensitive devices, the Arm Cortex-M4 processor implements tightly coupled system components that reduce processor area while significantly improving Bluetooth packet handling and peripheral system capabilities. It provides unique differentiating features that allow it to address multiple market verticals while keeping the same chassis. This guide is useful for programmers who are exploring the CYW208XX device features and peripherals. This guide teaches how to use various CYW208XX SoC peripherals and how to overcome hardware level constraints when developing applications and designing products.

# 8 Related Documents

| Application Notes | |
|---|---|
| **AN225684 – Getting Started with CYW208XX** | Describes CYW208XX Bluetooth SoC and how to build your first Bluetooth Low Energy application using the device in ModusToolbox IDE |
| **AN225948 - CYW20819/20820 Hardware Design Guideline** | This document includes basic layout guidelines which include the board stack up and the impedance control requirements, component placements and recommended parts for the critical components and routing guidelines and the recommended trace width for various power traces. |
| **AN225270 - CYW208XX Low Power Guidelines** | This document gives the guidelines on designing and developing a low-powered application using CYW208XX and the parameters to consider for power-efficient design. |
| **AN216403 – Application Buffer Pools** | This document provides a description of buffers used by the application and the upper layer stack of the WICED Bluetooth Stack. |
| **AN214852 - Collaborative Coexistence Interface Between Cypress-to Cypress Solutions and Cypress-to-third-party Chips** | This application note describes collaborative coexistence hardware mechanisms and algorithms of the CYW43XX and how to connect the COEX wiring between Cypress-to-Cypress solutions or Cypress to a third-party chip. |
| **Code Examples** | |
| Visit the **GitHub repository** for a comprehensive collection of code examples using ModusToolbox IDE | |
| Device and Evaluation Kit Documentation | |
| **CYW20819 Device Datasheet** | **CYW920819EVB-02 Evaluation Kit** |
| **CYW20820 Device Datasheet** | **CYW920820EVB-02 Evaluation Kit** |
| **CYBT-213043-02: EZ-BT™ Module** | **CYBT-213043-MESH EZ-BT™ Module Mesh Evaluation Kit** |
| **Tool Documentation** | |
| **ModusToolbox IDE** | The IDE for IoT designers |
| **Cypress Programmer Documentation** | This page will provide Cypress Programmer GUI User Guide and and Cypress Programmer release notes. |

## Revision history

| Document version | Date of release | Description of changes |
|---|---|---|
| ** | 2019-03-27 | Initial release |
| *A | 2020-04-20 | Updated the entire document for CYW20820.<br>Added Keyboard Scanner (Keyscan) section |
| *B | 2021-03-23 | Updated in Infineon template |

**Edition 2021-03-23**

**Published by**

**Infineon Technologies AG**

**81726 Munich, Germany**

**Do you have a question about this document?**

**Go to www.cypress.com/support**

**Document reference**

**002-26546 Rev. *B**