CYW920706WCDEVAL Hardware User Guide

Associated Part Family: CYW20706

Doc. No.: 002-16535 Rev. **

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

Contents

About This Document

Purpose and Audience

This document describes the Cypress Wireless Internet Connectivity for Embedded Devices (WICED; pronounced "wick-ed") CYW920706WCDEVAL evaluation board and provides various pins, jumpers, switches, ports, and test points to access CYW20706 to perform development, debug, evaluation, and troubleshooting. It also identifies the full complement of interfaces available on CYW20706 and interface-selection restrictions and consequences. In addition, it provides:

- CYW20706 interface options pertinent to the Cypress CYW920706WCDEVAL reference design
- API programming information examples for configuring various CYW20706 interfaces (SPI, I^2C, peripheral UART, and more).

This document is intended for designers and developers using CYW20706 WICED devices and the CYW920706WCDEVAL board as a reference design for CYW20706 embedded solutions with an embedded Bluetooth (BT) stack and user-developed applications using WICED Studio.

Scope

This document details the hardware aspects of the CYW920706WCDEVAL board and interface selection considerations for the CYW20706 device. For more information on the CYW20706 hardware, see the CYW20706 datasheet (*CYW20706 Embedded Bluetooth 4.2 SoC with MCU, Bluetooth Transceiver, and Baseband Processor [1]*). For information on using the board with WICED Studio to develop embedded custom applications, see the *WICED CYW920706WCDEVAL Kit Guide [2].*

Several paragraphs in the document refer the reader to WICED Studio files, variables and data structures that are not described in this document. For information on the variables and data structures mentioned in this document, see the WICED Studio API documentation that is provided with WICED Studio.

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 http://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 (http://community.cypress.com/).

1 Product Description

CYW20706 is a monolithic, single chip, Bluetooth (BT) dual-mode System-on-a-Chip (SoC) that includes a baseband processor, an ARM® Cortex™-M3 processor and an integrated transceiver. It is a fully embedded device running an embedded BT stack with support for embedded user applications developed with WICED Studio.

The Cypress CYW920706WCDEVAL board (Figure 1-1) is an evaluation board that provides various pins, jumpers, switches, ports, and test points to access CYW20706 to perform debug, evaluation, and troubleshooting.

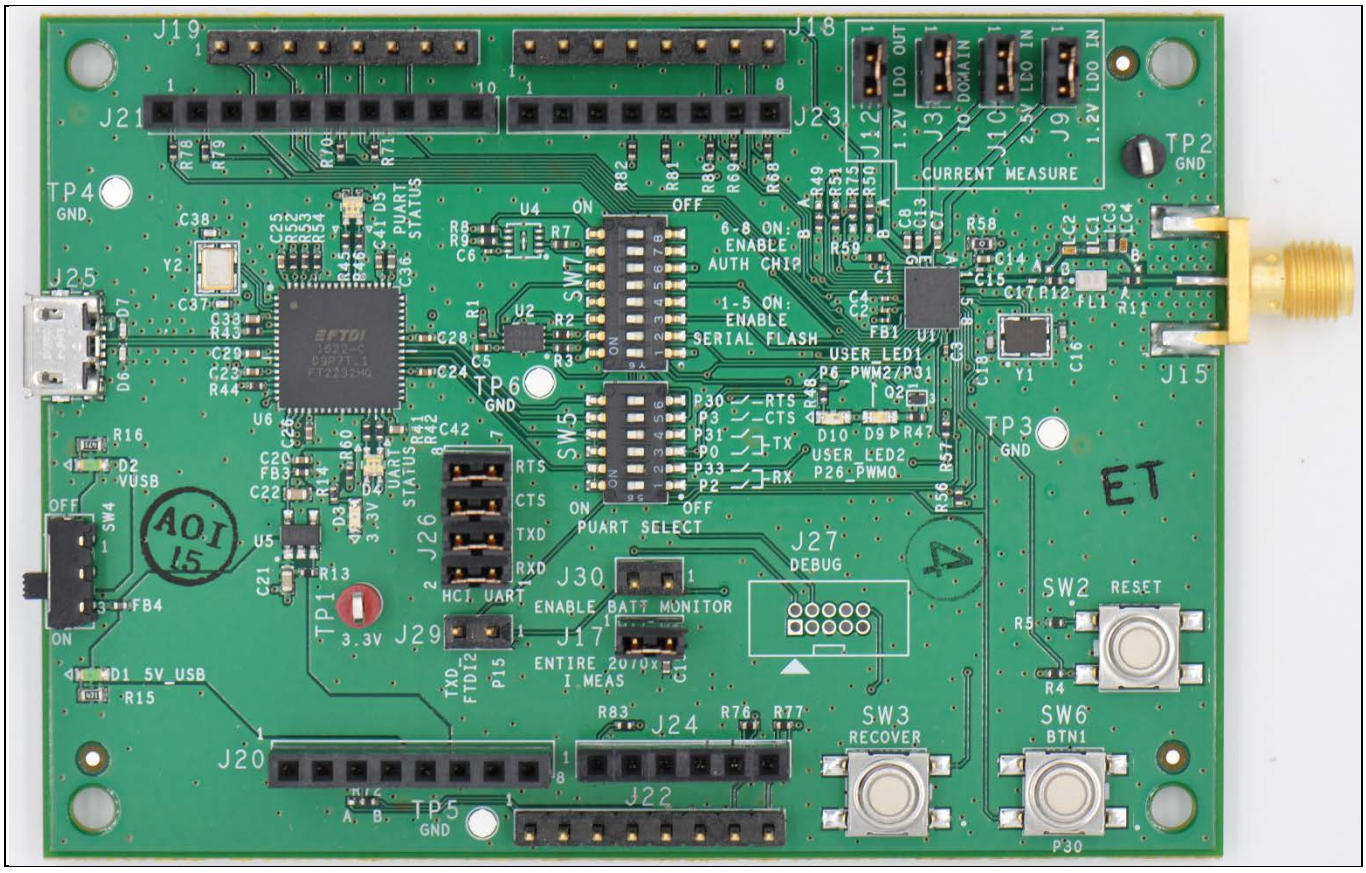


Figure 1-1. CYW920706WCDEVAL Board

Figure 1-2 shows the pins connected to the user accessible headers on the board.

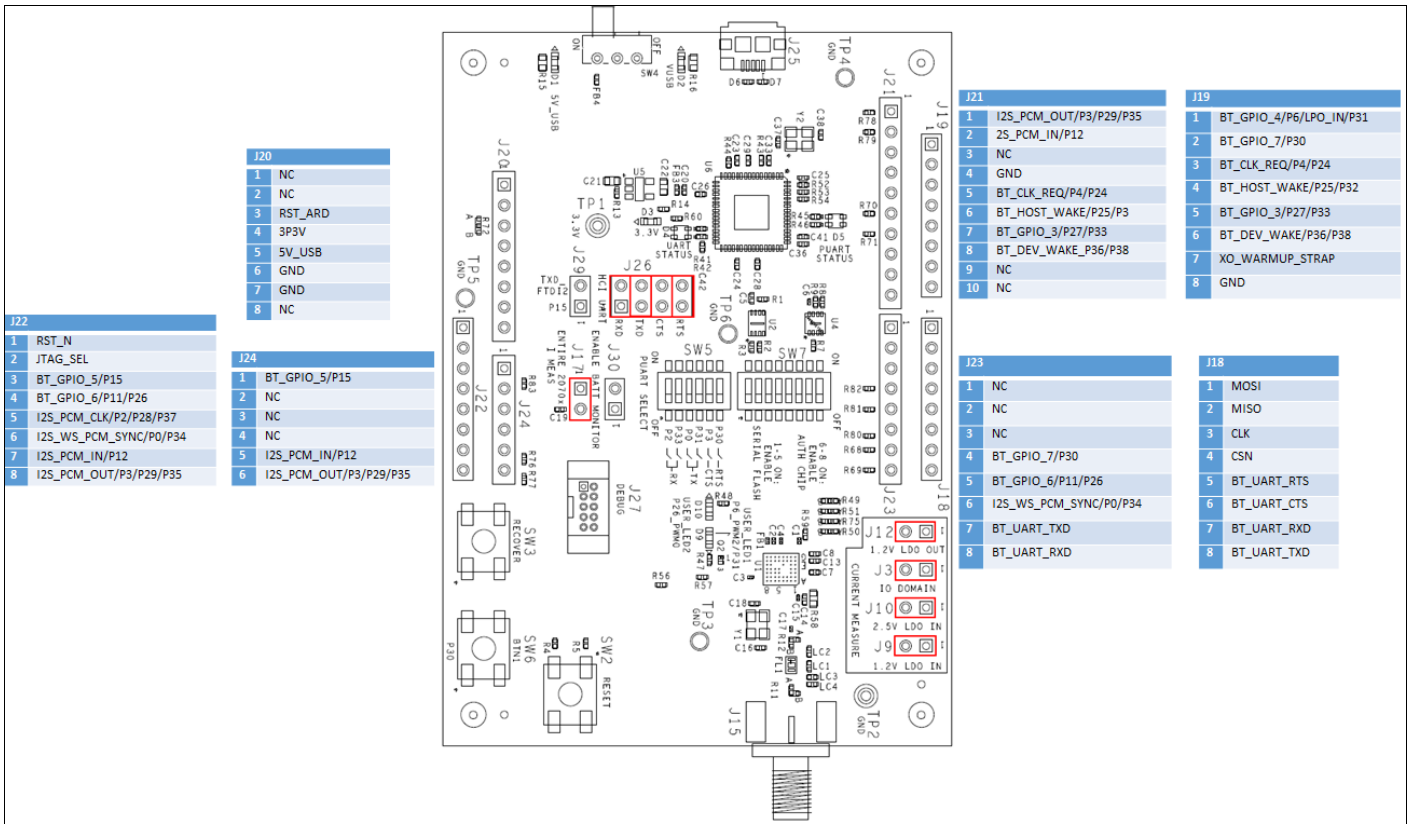


Figure 1-2. User Accessible headers

2 Board Layout

Figure 2-1 shows the location of key jumpers and switches on the CYW920706WCDEVAL board.

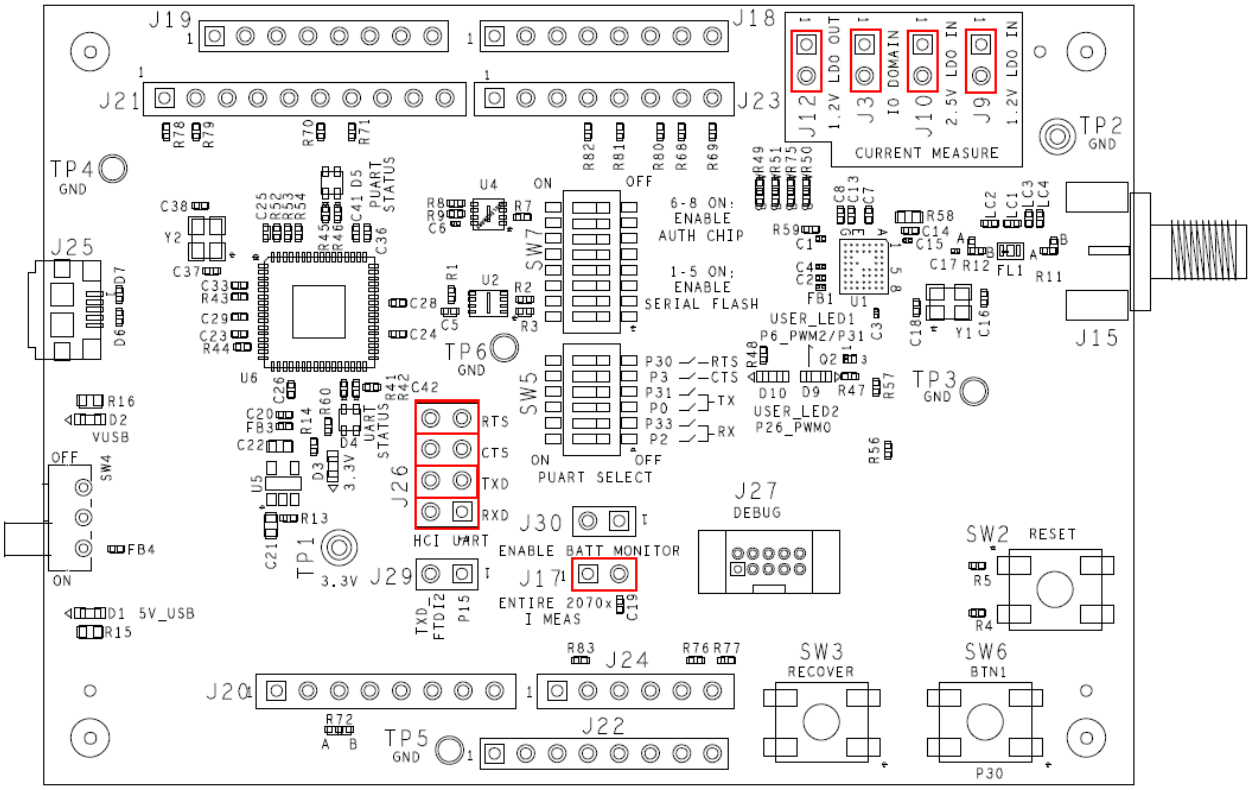


Figure 2-1. CYW920706WCDEVAL Evaluation Board Layout and Component Locations

Note: Jumpers which are shorted by default are identified in red.

3 Board Block Diagram

Figure 3-1 shows the block diagram of CYW20706 connections on the CYW920706WCDEVAL board.

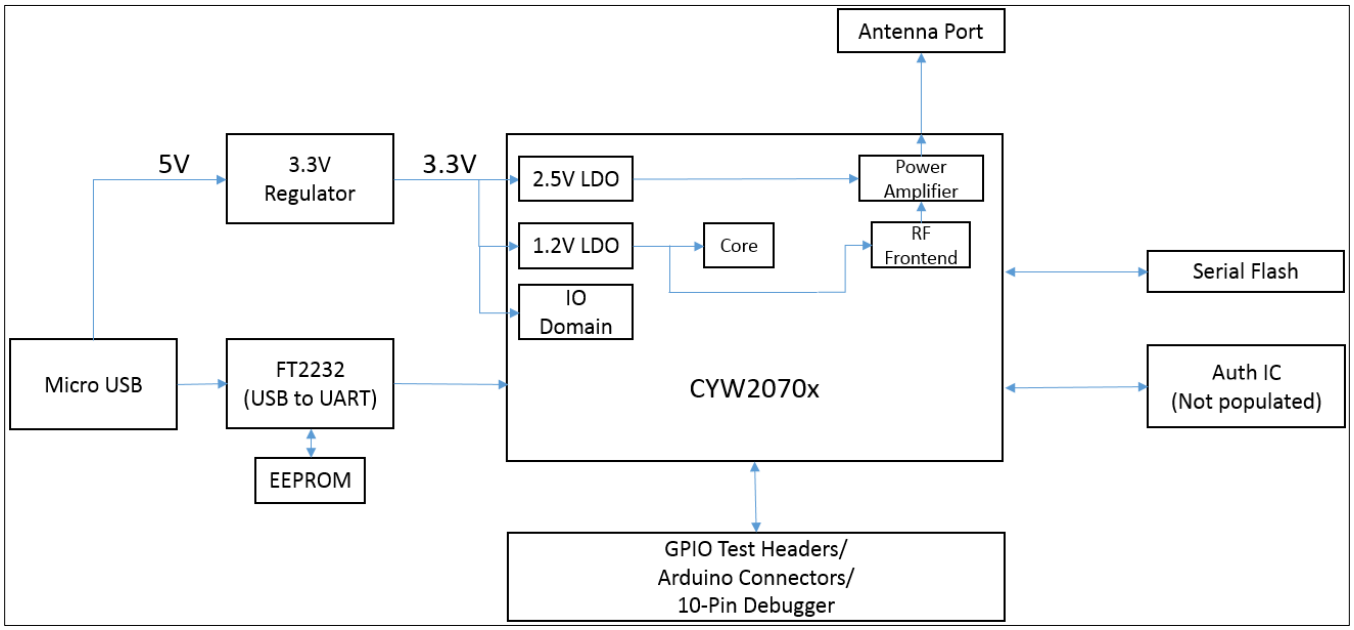


Figure 3-1 Board Block Diagram

4 Jumper and DIP Switch Settings

Figure 2-1 shows the location for SW5, a switch that is used to configure Peripheral UART (PUART). Settings are shown in Table 4-1.

DIP	Default State	Description
1	OFF	Set this switch ON to use P2 as PUART_RX
2	ON	Set this switch ON to use P33 as PUART_RX
3	OFF	Set this switch ON to use P0 as PUART_TX
4	ON	Set this switch ON to use P31 as PUART_TX
5	OFF	Set this switch ON to use P3 as PUART_CTS
6	OFF	Set this switch ON to use P30 as PUART_RTS
Note: Only one of DIP switches 1 or 2 should be turned ON at once (but never both), and only one of DIP switches 3 or 4 should be turned ON at once (but never both).		

Table 4-1. SW5 DIP Switch Settings

Figure 2-1 shows the location for SW7, a switch that is used to configure serial flash and authentication IC connections. Settings are shown in Table 4-2.

DIP	Default State	Description
Serial Flash		
1	ON	Set this switch ON to power the serial flash from VDDIO
2	ON	Set this switch ON to enable the SPI MISO connection between CYW20706 and serial flash
3	ON	Set this switch ON to enable the SPI MOSI connection between CYW20706 and serial flash
4	ON	Set this switch ON to enable the SPI CS connection between CYW20706 and serial flash
5	ON	Set this switch ON to enable the SPI CLK connection between CYW20706 and serial flash
Auth IC		
6	OFF	Set this switch ON to power the authentication IC from VDDIO
7	OFF	Set this switch ON to enable the I^2C SDA connection between CYW20706 and authentication IC
8	OFF	Set this switch ON to enable the I^2C SCL connection between CYW20706 and authentication IC

Table 4-2. SW7 DIP Switch Settings

Authentication IC U4 is DNI by default. If the user decides to install this, then SW7 positions 6-8 must be placed in the ON position for proper connection. If authentication IC is not used, set these to the OFF position.

See Figure 2-1 for the jumpers and switch locations. Table 4-3 shows the CYW920706WCDEVAL board jumper and switch settings.

Jumper/Switch	Default State	Comment
J3	Shorted	Short this Jumper to supply 3.3V to the IO Domain (VDDIO) of the 0270x. Also use this jumper to measure the current consumption of the IO Domain.
J9	Shorted	Input of the internal 1.2V LDO, use this jumper to measure current consumption at the input of the 1.2V LDO
J10	Shorted	Input of the internal 2.5V LDO, use this jumper to measure current consumption at the input of the 2.5V LDO
J12	Shorted	Output of the internal 1.2V LDO, use this jumper to measure current consumption at the output of the 1.2V LDO
J17	Shorted	Supplies power to all the power rails in the 20706, use this jumper to measure current consumption of the entire 20706 chip.
J26 1 and 2	Shorted	Connects CYW20706 BT_UART_RXD to FTDI TX
J26 3 and 4	Shorted	Connects CYW20706 BT_UART_TXD to FTDI RX
J26 5 and 6	Shorted	Connects CYW20706 BT_UART_CTS to FTDI RTS
J26 7 and 8	Shorted	Connects CYW20706 BT_UART_RTS to FTDI CTS
J29	Open	Short this jumper to connect GPIO P15 to the RX of the second FTDI com port for debugging purposes
J30	Open	Short this jumper to connect GPIO P15 to VDDIO for battery monitoring applications.
SW2	–	Reset Button
SW3	–	Recovery Button
SW4	3 ON	Power switch positions: 1: OFF, 3: ON
SW5	See Table 4-1	Serial Flash Connections
SW6	–	User button
SW7	See Table 4-2	Authentication IC Connections

Table 4-3. CYW920706WCDEVAL Board Jumper and Switch Settings

Table 4-4 shows the CYW920706WCDEVAL board headers.

Header	Description
J18	CYW test header - SPI, UART
J19	CYW test header – GPIO
J20	Arduino shield connection
J21	Arduino shield connection
J22	CYW test header: GPIO, Reset, I^2S, PWM
J23	Arduino shield connection
J24	Arduino shield connection
J27	10 Pin JTAG connector

Table 4-4. CYW920706WCDEVAL Board Headers

5 Current Consumption Measurement

Table 5-1 shows the low-power Bluetooth classic mode current measured for different sleep modes in three different scenarios.

Mode	J9 (1.2V Core + Radio)	J3 (IO Domain)	J9 +J3	Units
No scans enabled + sleep mode	0.114	0.084	0.198	mA
Page scan enabled + sleep mode	0.328	0.083	0.411	mA
Sniff link 1.28s, 4 attempts, 0 timeout, no scans + sleep mode	1.250	0.084	1.334	mA

Table 5-1. Low-Power Bluetooth Current

Table 5-2 lists the jumper locations for measuring current.

Note: Remove the listed jumper and measure the current across the exposed pins.

To Measure...	Remove the Jumper and Measure Across...
IO Domain (VDDIO)	J3
1.2V LDO input (1.2V Core + Radio)	J9
2.5V LDO input (2.5V LDO + On Chip Power Amplifier)	J10
1.2V LDO output (1.2V Core only)	J12
Total Current consumed by CYW20706	J17

Table 5-2. Current Measurements

6 Schematics

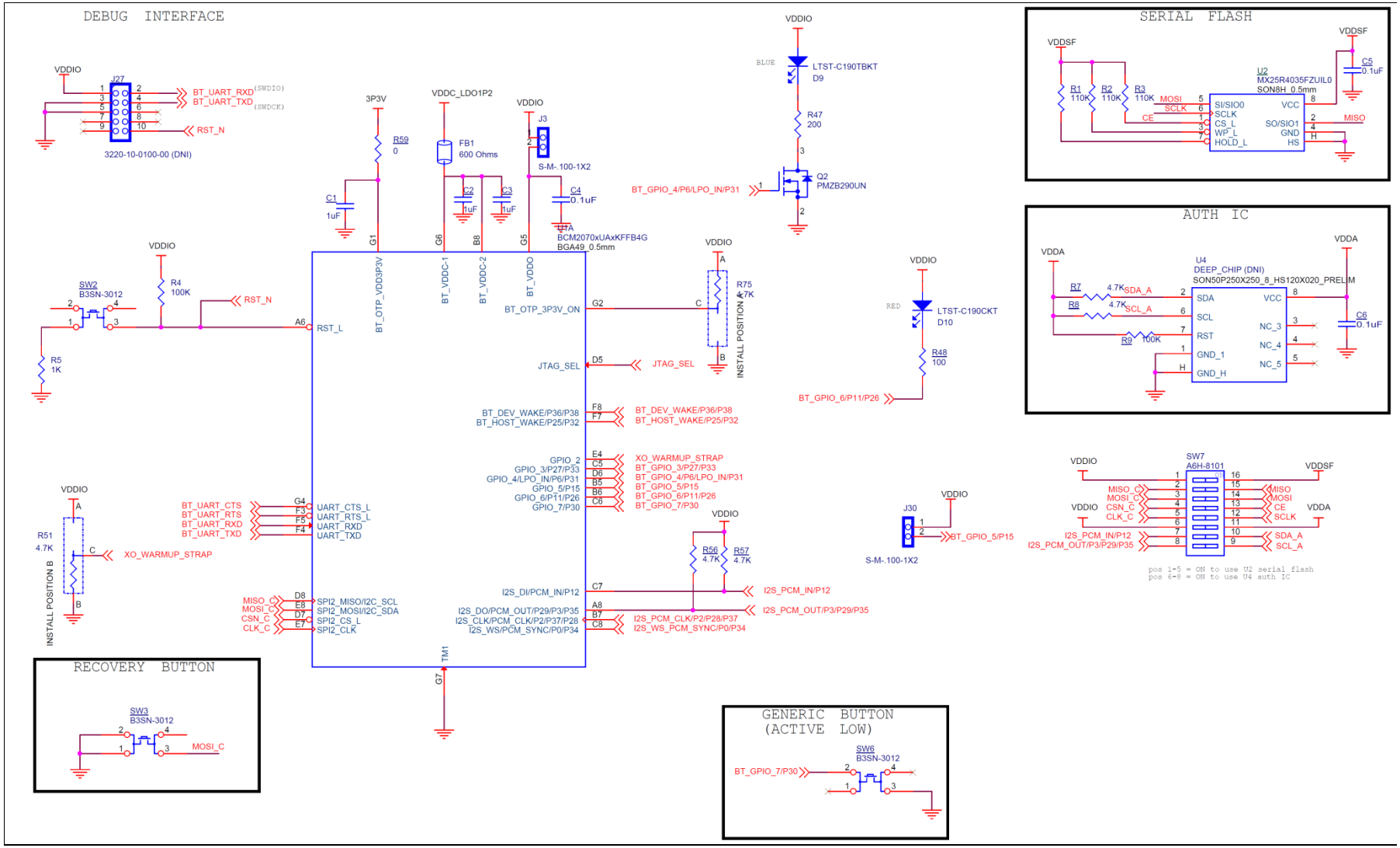
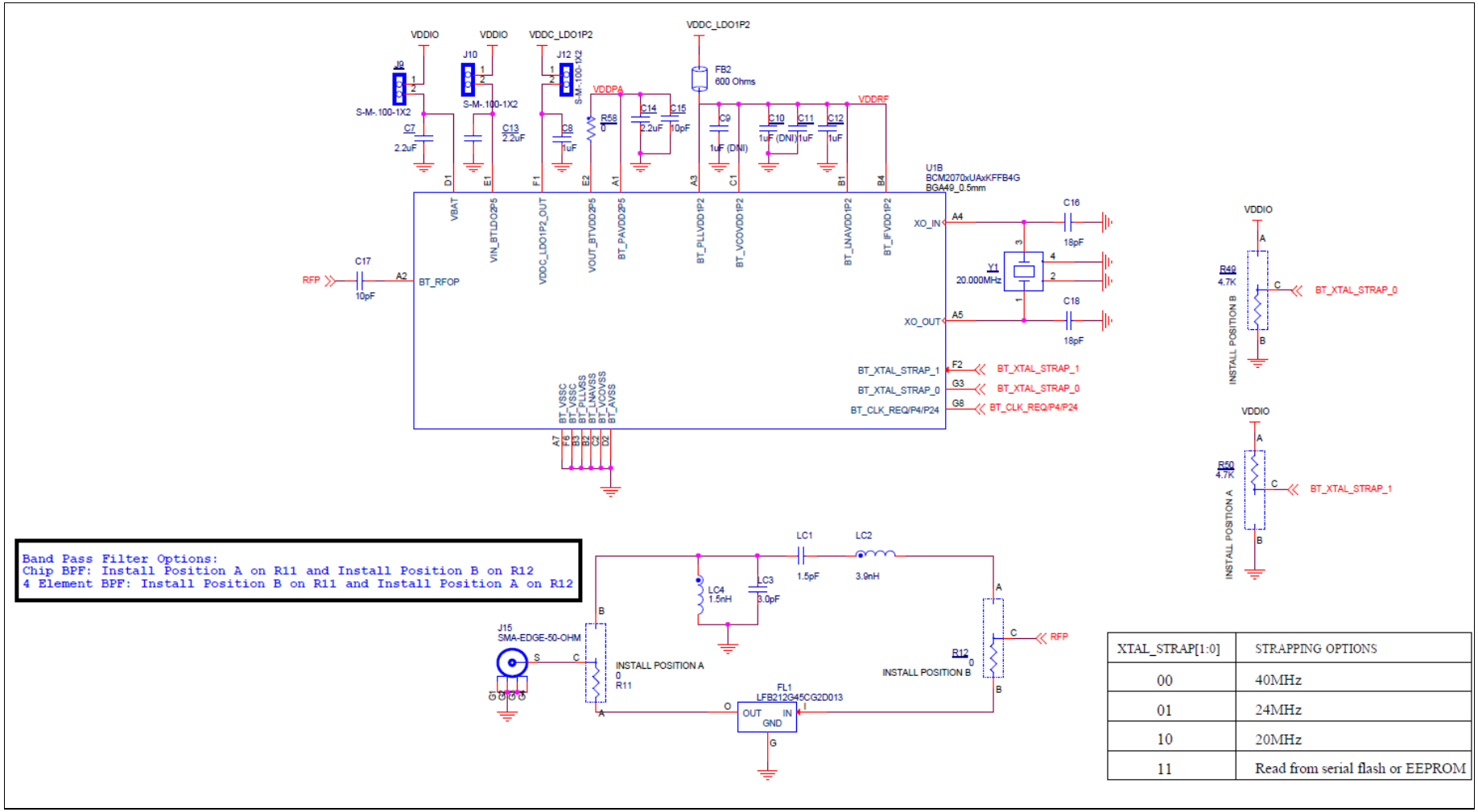


Figure 6-1. CYW920706WCDEVAL Baseband Schematic



XTAL_STRAP[1:0]	STRAPPING OPTIONS
00	40MHz
01	24MHz
10	20MHz
11	Read from serial flash or EEPROM

Figure 6-2. CYW920706WCDEVAL RF Schematic

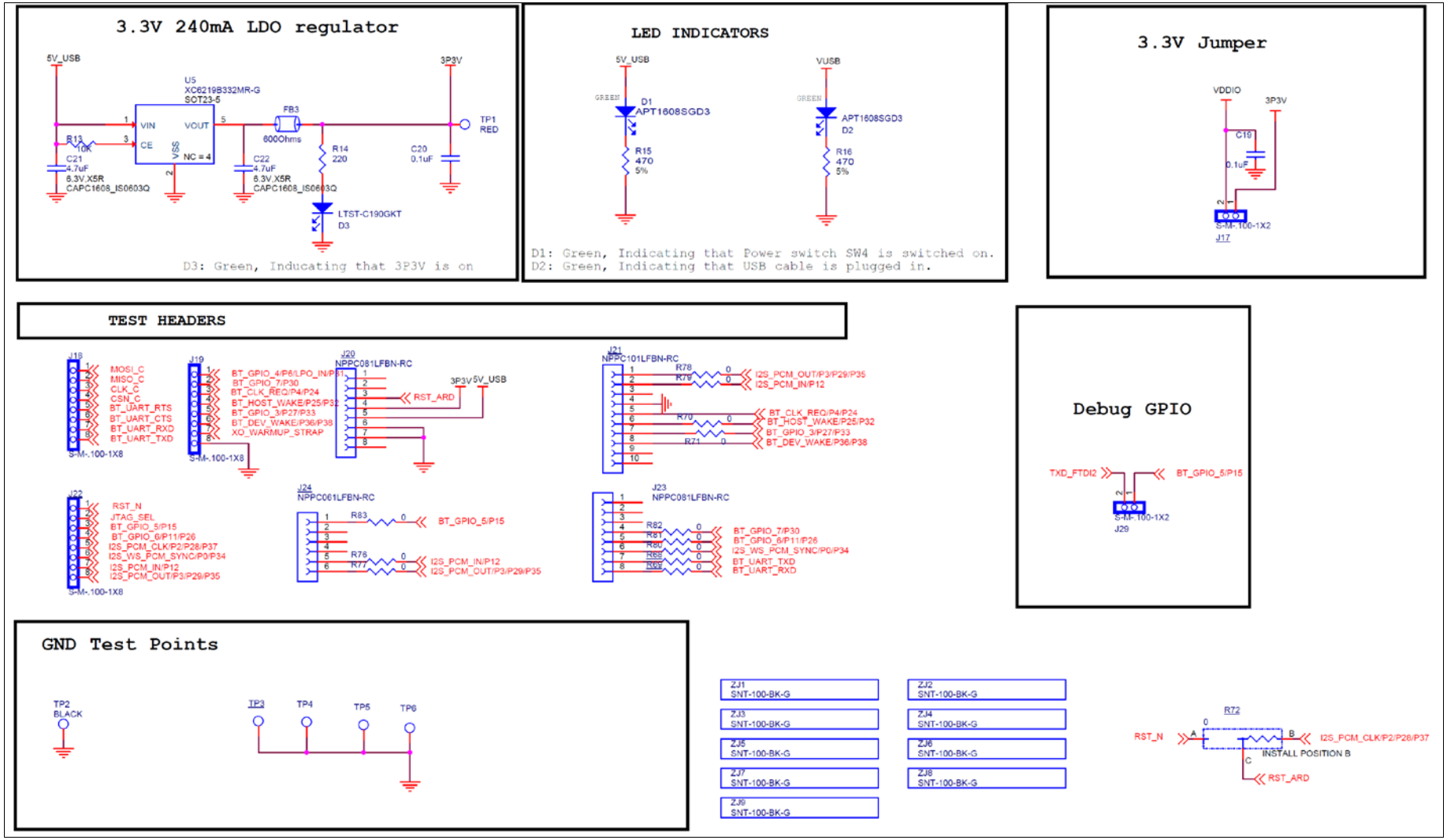


Figure 6-3. CYW920706WCDEVAL Power & Headers Schematic

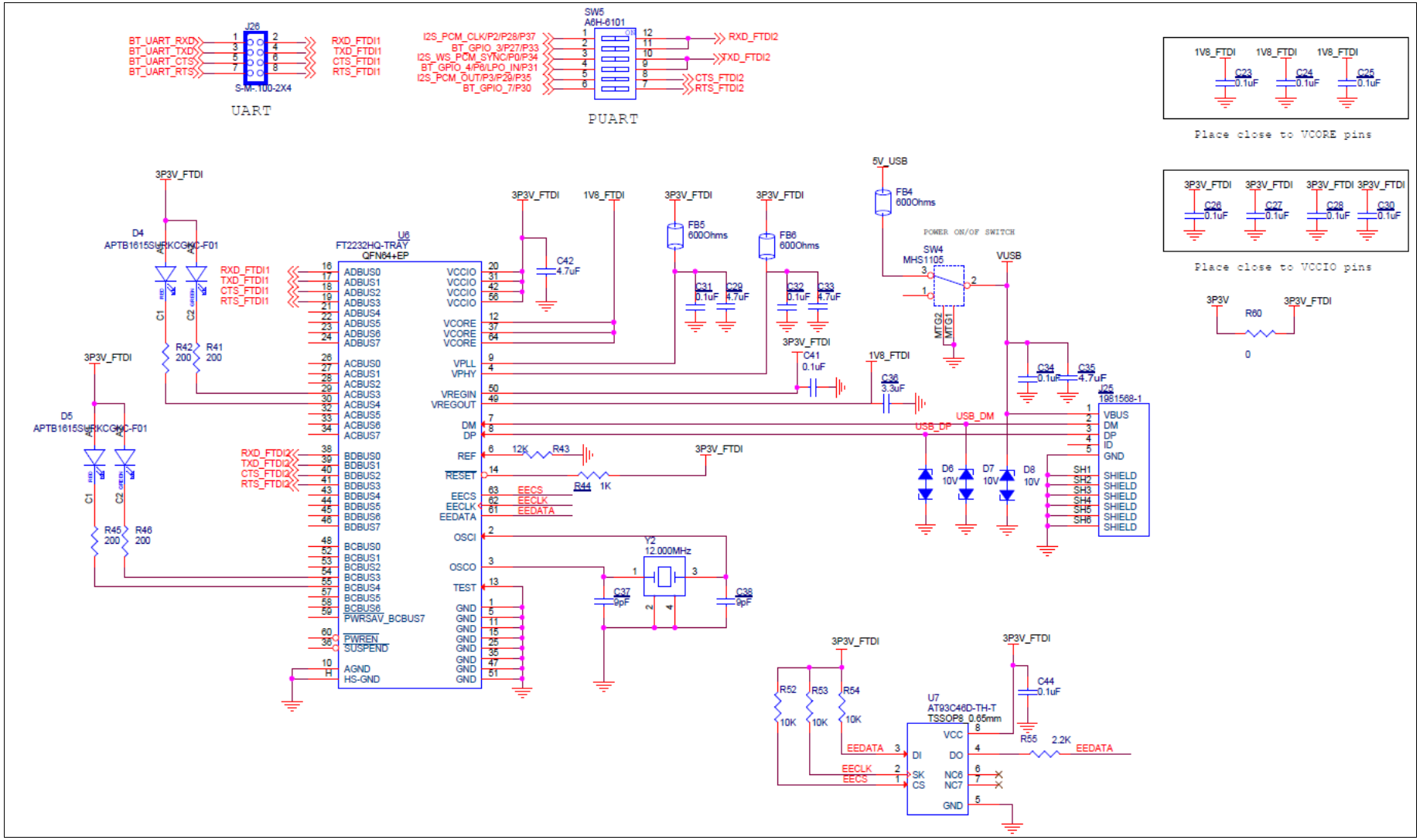


Figure 6-4. CYW920706WCDEVAL USB to UART Interface Schematic

7 CYW20706 Interfaces

The package CYW20706 is available in may have fewer GPIOs than the die. Some of these GPIOs are fixed function IOs while others are reconfigurable for different interfaces. To accommodate its various applications, interface selection is accomplished through a combination of signal routing to CYW20706 and programming using the WICED API described in later sections.

CYW20706 interfaces are presented as two interface sets: fixed and selectable. The fixed interfaces are those with dedicated pins and, thus, are always available. The selectable interfaces are those that a board designer chooses to use from a superset of possible interfaces that CYW20706 supports.

See Section 7.1 "Fixed Interfaces" for more information on the fixed interfaces. See Section 7.2 "Selectable Interfaces" for more information on the set of interfaces from which board designers can choose.

7.1 Fixed Interfaces

Table 7-1 shows the fixed straps and digital I/O interfaces of CYW20706. In contrast to the Selectable Interfaces, the fixed interfaces of CYW20706 have dedicated pins.

Strap or Interface Purpose	Signals	Pins	Notes
XTAL frequency selection[1]	BT_XTAL_STRAP_0	G3	
	BT_XTAL_STRAP_1	F2	F2 / G3 / XTAL 0 / 0 / 40 MHz 0 / 1 / 24 MHz 1 / 0 / 20 MHz 1 / 1 / Read from NV memory
Device reset	RST_N	A6	Active LOW reset input
OTP usage	BT_OTP_3P3V_ON	G2	Pull HIGH if OTP is used; otherwise, pull LOW.
HCI UART	BT_UART_RXD	F5	HCI UART receive data (or SWDIO, see 9.3.1)
	BT_UART_TXD	F4	HCI UART transmit data (or SWDCLK, see 9.3.1)
	BT_UART_RTS_N	F3	HCI UART request-to-send input
	BT_UART_CTS_N	G4	HCI UART clear-to-send input
Serial Peripheral Interface (SPI) or I²C, sometimes known as Broadcom Serial Control (BSC)[1]	SPI2_MISO_I2C_SCL[2]	D8	SPI MISO
	SPI2_MOSI_I2C_SDA[2]	E8	SPI MOSI
	SPI2_CLK	E7	SPI clock output
	SPI2_CSN	D7	SPI active LOW chip select output

Table 7-1. Fixed Straps and Digital I/O Interfaces

[1] Use the SPI2 interface for applications where the CYW20706 firmware image is to be loaded from onboard serial flash. The SPI2 interface is sometimes referred to as Spiffy2.

[2] Although the SPI2_MISO_I2C_SCL and SPI2_MOSI_I2C_SDA signal names imply support for an I²C-compatible interface via pins D8 and E8, an I2C-compatible interface is not usable via these pins since the SPI signals are connected to the serial flash on this hardware. CYW20706 does support an I²C-compatible interface (BSC interface) via pins A8 and C7 (including pull ups) or B7 and C7 (see Section 9.5 "Broadcom Serial Control").

7.2 Selectable Interfaces

CYW20706 supports several other interfaces besides those identified in Fixed Interfaces. Although CYW20706 supports several other interfaces, it cannot support all of its interfaces in a single hardware board design. Therefore, board designers must select which interfaces to use in a given design.

The key limitation on the selectable interfaces is the number of available digital I/O pins. The selectable interfaces are multiplexed to 12 digital I/O pins. The 12 digital I/O pins are: A8, B5, B6, B7, C5, C6, C7, C8, D6, F7, F8, and G8.

The following interfaces represent the superset of interfaces supported by CYW20706:

- Multiple general purpose I/Os (GPIOs), identified as GPIO_Pxx, and also referred to as the low leakage LHL3 GPIOs (LHL is a low leakage power island for GPIOs on the chip). For more information on GPIO signals, see Section 8 “GPIO Information”.
- Four PWM outputs
- A peripheral UART (or PUART). This PUART is for attachment to microcontroller units (MCUs) or onboard peripherals.
- A Serial Peripheral Interface (SPI). This is a second SPI interface that is identified as SPI1 (and sometimes called Spiffy1). It can be a master or a slave.
- Multiple A/D converter inputs
- Multiple auxiliary clock outputs
- An infrared learning (IR_RX) input and playback (IR_TX) output
- A keyboard scan output (KSO3)
- Four optical control outputs (QOC0 through QOC3) for use in quadrature or rotary encoders.\
- A 60 Hz input (60Hz_main) to a zero-crossing detector
- Two triac control outputs
- Two external Transmit/Receive (T/R) switch control outputs (TX_PD and ~TX_PD)
- Multiple inputs from quadrature detectors (QDX0, QDX1, QDY0, QDY1, QDZ0, and QDZ1)
- A shared PCM or I^2S interface
- A Bluetooth clock request (BT_CLK_REQ) for a shared-clock application
- A Low Power Oscillator (LPO) input
- An Inter-Integrated Circuit (I^2C) serial interface, also called Broadcom Serial Control (BSC) interface

These selectable interfaces are mapped to the 12 digital I/O pins. Many signal functions are supported at each pin, but a system designer must select a single function per pin. See Section 8 “GPIO Information” for detailed information on signal-function to I/O pin assignments.

7.3 Selectable Interfaces Supported by CYW920706WCDEVAL

The CYW920706WCDEVAL board supports all of the fixed interfaces (see Section 7.1 ”Fixed Interfaces”) but only supports a subset of the selectable interfaces (see Section 7.2 “Selectable Interfaces”) supported by CYW20706.

The CYW920706WCDEVAL board supports the following selectable interfaces:

- SPI1 master
- PUART
- I^2S, I^2C, PWM, PCM
- BT_GPIO_P5/P15 connected via jumper to VDDIO which can be used for voltage detect
- BT_GPIO_P4/P6 to control external LEDs
- BT_GPIO_P7 to monitor SW6 button depression

Table 7-2 shows which of the selectable CYW20706 interfaces are supported on the CYW920706WCDEVAL board.

CYW20706 Pin	Schematic Signal Name	Bonded GPIOs	Description		
			Default State	I/O Type	Signal Function and Notes
A8	I2S_PCM_OUT/P3/P29/P35	I2S_PCM_OUT	LOW[3]	O	I2S_DO/PCM_OUT. To use I^2S, set SW5-5 to the OFF position, from the SDK you need to configure the *pcm_config* structure pointer with the mode role set and use the function *wiced_hal_set_pcm_config* to select the I^2S function. I/Os P3, P29, and P35 must remain disabled. This pin can be configured as I^2C SCL by calling *wiced_hal_i2c_init(WICED_I2C_SDA_I2S_DOUT_PCM_OUT_SCL_I2S_DIN_PCM_IN)* which selects the I2C_SCL on I2S_PCM_OUT.
		P3	Floating	I/O	This can be used as PUART_CTS by setting SW5-5 to the ON position. It can be monitored at J22-8 if the I^2S/PCM function is deselected, and I/Os P29 and P35 are disabled.
		P29	Floating	I/O	No specific function defined. It can be monitored at J22-8 if the I^2S/PCM function is deselected, and I/Os P3 and P35 are disabled.
		P35	Floating	I/O	No specific function defined. It can be monitored at J22-8 if the I^2S/PCM function is deselected, and P3 and P29 are I/O disabled. This pin can be configured as I^2C SDA by calling *wiced_hal_i2c_init(WICED_I2C_SDA_P35_SCL_P37)* which selects the I2C_SDA on P35
B5	BT_GPIO_5/P15	BT_GPIO_5	HIGH[3]	I/O	General Purpose I/O
		P15	Floating	O	GPIO to monitor battery or general-purpose I/O
B6	BT_GPIO_6/P11/P26	BT_GPIO_6	LOW[3]	I/O	General Purpose I/O
		P11	Floating	I/O	No specific function defined. It can be monitored at J22-4 if I/O P26 is disabled.
		P26	Floating	O	LED D10 on/off control that can sink 16 mA and be modulated using the PWM0 circuit on CYW20706. To use, disable I/O P11.
B7	I2S_PCM_CLK/P2/P28/P37	I2S_PCM_CLK	LOW[3]	O	I2S_CLK. To use, set SW5-1 to the OFF position, select the I^2S function, and disable I/Os P2, P28, and P37.
		P2	Floating	I/O	PUART_RX. To use, set SW5-1 to the ON position, SW5-2 to the OFF position, and disable I/Os P28 and P37. It can be monitored at J22-5 if the I^2S/PCM function is deselected with the mentioned SW5 settings.
		P28	Floating	I/O	No specific function defined. It can be monitored at J22-5 if the I^2S/PCM function is deselected, and I/Os P2 and P37 are disabled.

[3] CYW20706 has an internal 40K pull down for LOW state and 40K pull up for HIGH state. If the pin needs to be driven to the opposite HIGH state for an application, one of the other connected GPIOs should be configured to drive it as needed.

CYW20706 Pin	Schematic Signal Name	Bonded GPIOs	Description		
			Default State	I/O Type	Signal Function and Notes
		P37	Floating	I/O	No specific function defined. It can be monitored at J22-5 if the I^2S/PCM function is deselected, and I/Os P2 and P28 are disabled. This pin can be configured as I^2C SCL by configuring *wiced_hal_i2c_init(WICED_I2C_SDA_P35_SCL_P37)* which selects the I2C_SCL on P37
C5	BT_GPIO_3/P27/P33	BT_GPIO_3	LOW[3]	I/O	General Purpose I/O
		P27	Floating	O (master) I (slave)	SPI1_MOSI (master or slave) To use, set SW5-2 to the OFF position and disable I/O P33.
		P33	Floating	I	PUART_RX. To use, set SW5-1 to the OFF position, SW5-2 to the ON position, and disable I/O P27.
C6	BT_GPIO_7/P30	BT_GPIO_7	LOW[3]	I/O	General Purpose I/O
		P30	Floating	I	SW6 button input. This can also be configured as PUART_RTS by setting SW5-6 to the ON position since it is internally muxed on the same pin.
C7	I2S_PCM_IN/P12	I2S_PCM_IN	LOW[3]	I	I2S_DI. To use, select the I^2S function, and disable I/O P12. This pin can be configured as I2C_SDA by calling *wiced_hal_i2c_init(WICED_I2C_SDA_I2S_DOUT_PCM_OUT_SCL_I2S_DIN_PCM_IN)* which selects the I2C_SDA on I2S_PCM_IN.
		P12	Floating	I/O	No specific function defined. It can be monitored at J22-7 if the I^2S/PCM function is deselected.
C8	I2S_WS_PCM_SYNC/P0/P34	I2S_WS_PCM_SYNC	LOW[3]	I/O	I2S_WS. To use, set SW5-3 to the OFF position, select the I^2S function, and disable I/Os P0 and P34.
		P0	Floating	I/O	PUART_TX. To use, set SW5-3 to the ON position, SW5-4 to the OFF position, and disable I/O P34. It can be monitored at J22-6 if the I^2S/PCM function is deselected and I/O P34 is disabled.
		P34	Floating	I/O	No specific function defined. It can be monitored at J22-6 if the I^2S/PCM function is deselected and I/O P0 is disabled.
D6	BT_GPIO_4/P6/LPO_IN/P31	BT_GPIO_4	HIGH[3]	I/O	General Purpose I/O
		P6	Floating	I/O	LED D9 on/off control that can be modulated using the PWM2 circuit on CYW20706. To use, disable I/O P31.
		P31	Floating	O	PUART_TX. To use, set SW5-3 to the OFF position, set SW5-4 to the ON position, and disable I/O P6.
F7	BT_HOST_WAKE/P25/P32	BT_HOST_WAKE	LOW[3]	O	A signal from CYW20706 device to the host indicating that the Bluetooth device requires attention

CYW20706 Pin	Schematic Signal Name	Bonded GPIOs	Description		
			Default State	I/O Type	Signal Function and Notes
		P25	Floating	I (master) O (slave)	SPI1_MISO (master or slave). To use, disable I/O P32.
		P32	Floating	I/O	No specific function defined. It can be monitored at J19-4 if I/O P25 is disabled.
F8	BT_DEV_WAKE/P36/P38	BT_DEV_WAKE	LOW[3]	I	A signal from the host to CYW20706 that the host requires attention.
		P36	Floating	O (master) I (slave)	GPIO_P36 programmed as SPI1_CS. To use, disable I/O P38.
		P38	Floating	I/O	No specific function defined. It can be monitored at J19-6 if I/O P36 is disabled.
G8	BT_CLK_REQ/P4/P24	P4	Floating	I/O	No specific function defined. It can be monitored at J19-3 if I/O P24 is disabled.
		P24	Floating	I/O	SPI1_CLK (master or slave). To use, disable I/O P4.

Table 7-2. CYW920706WCDEVAL Board Interface Summary

8 GPIO Information

8.1 GPIO_Pxx

The Cypress WICED Studio API provides configuration support for up to 40 multiplexed GPIOs (represented as WICED_GPIO_P0 through WICED_GPIO_P39). CYW20706 uses 23 of these multiplexed GPIOs. The assignment of the multiplexed GPIOs to physical device pins is restricted from certain digital I/O pins when certain interfaces or signal functional are used.

Table 8-1 shows CYW20706 interfaces and signal functions that, if used in a design, will restrict the set of digital I/O pins available for assigning GPIOs.

If This Interface or Signal Function Is Used	Then These Pins Cannot Be Used for GPIO_Pxx Assignments
I^2S/PCM	A8, B7, C7, and C8
BT_CLK_REQ	G8
BT_DEV_WAKE[1]	F8
BT_HOST_WAKE[1]	F7
LPO_IN	D6

Table 8-1. Key Interface (or Signal Function) Pin Assignments

These signal functions can be used to wake CYW20706 from sleep modes by an external MCU and for CYW20706 to wake an external MCU. Embedded applications have better options for sleep and wake operations rather than using a dedicated digital I/O pin, see Appendix A “Power-Save Options”.

[1] Unused GPIOs will be HI-Z (I/O disabled) after application initialization.

8.1.1 Multiplexed GPIO_Pxx Interface Summary

Table 8-2 provides an interface summary of the 23 multiplexed GPIOs used by CYW20706. The description column provides the potential signal functions of each GPIO and the associated I/O type of each signal function.

CYW20706 Pin	GPIO	Description	
		I/O Type[1]	Signal Function Options
A8	P3	I/O	General purpose, user-defined GPIO
		I	X-coordinate output from a quadrature detector (QDX1)
		I	Peripheral UART CTS input (PUART_CTS)
		O	SPI1_CLK (master)
		I	SPI1_CLK (slave)
	P29[2]	I/O	General purpose, user-defined GPIO
		O	Optical control output (QOC3)
		I	A/D converter input 10
		O	PWM output (PWM3)
	P35	I/O	General purpose, user-defined GPIO
		I	A/D converter input (A/D input 4)
		I	Y-coordinate output from a quadrature detector (QDY1)
		I	Peripheral UART CTS input (PUART_CTS)
		I/O	BSC SDA (I2C_SDA)
B5	P15	I/O	General purpose, user-defined GPIO
		I	A/D converter input (A/D input 20)
		I	Infrared learning input (IR_RX)
		I	60 Hz input (60Hz_main) to a zero-crossing detector.
B6	P11	I/O	General purpose, user-defined GPIO
		I	Keyboard scan output (KSO3)
		I	A/D converter input (A/D input 24)
	P26[2]	I/O	General purpose, user-defined GPIO
		I	SPI1_CS (slave)
		O	Optical control output (QOC0)
		O	Triac control 1
		O	PWM output (PWM0)

[1] Unused GPIOs will be HI-Z (I/O disabled) after application initialization.

[2] This GPIO can source and sink 16 mA at 3.3 V or 8 mA at 1.8 V. For all other GPIOs, the maximum is 8 mA at 3.3 V and 4 mA at 1.8 V.

CYW20706 Pin	GPIO	Description	
		I/O Type[1]	Signal Function Options
B7	P2	I/O	General purpose, user-defined GPIO
		I	X-coordinate output from a quadrature detector (QDX0)
		I	Peripheral UART RX input (PUART_RX)
		I	SPI1_CS (slave)
		O	SPI1_MOSI (master)
	P28[2]	I/O	General purpose, user-defined GPIO
		O	Optical control output (QOC2)
		I	A/D converter input (A/D input 11)
		O	PWM output (PWM2)
	P37	I/O	General purpose, user-defined GPIO
		I	A/D converter input (A/D input 2)
		I	Z-coordinate output from a quadrature detector (QDZ1)
		O	SPI1_MISO (slave)
		O	Auxiliary clock output (ACLK1)
		O	BSC SCL (I2C_SCL)
C5	P27[2]	I/O	General purpose, user-defined GPIO
		O	SPI1_MOSI (master)
		I	SPI1_MOSI (slave)
		O	Optical control output (QOC1)
		O	Triac control 2
		O	PWM output (PWM1)
	P33	I/O	General purpose, user-defined GPIO
		I	A/D converter input (A/D input 6)
		I	X-coordinate output from a quadrature detector (QDX1)
		I	SPI1_MOSI (slave)
		O	Auxiliary clock output (ACLK1)
		I	Peripheral UART RX input (PUART_RX)
C6	P30	I/O	General purpose, user-defined GPIO
		I	A/D converter input (A/D input 9)
		O	Peripheral UART RTS input (PUART_RTS)
C7	P12	I/O	General purpose, user-defined GPIO

CYW20706 Pin	GPIO	Description	
		I/O Type[1]	Signal Function Options
		I	A/D converter input (A/D input 23)
		O	BSC SCL (I2C_SCL)
C8	P0	I/O	General purpose, user-defined GPIO
		I	A/D converter input (A/D input 29)
		O	Peripheral UART TX output (PUART_TX)
		O	SPI1_MOSI (master output)
		I	SPI1_MOSI (slave input)
		I	Infrared learning input (IR_RX)
		I	60 Hz input (60Hz_main) to a zero-crossing detector.
	P34	I/O	General purpose, user-defined GPIO
		I	A/D converter input (A/D input 5)
		I	Y-coordinate output from a quadrature detector (QDY0)
		I	Peripheral UART RX input (PUART_RX)
		O	T/R switch control (TX_PD)
D6	P6	I/O	General purpose, user-defined GPIO
		I	Z-coordinate output from a quadrature detector (QDZ0)
		O	Peripheral UART RTS input (PUART_RTS)
		I	SPI1_CS (slave)
		I	60 Hz input (60Hz_main) to a zero-crossing detector.
	P31	I/O	General purpose, user-defined GPIO
		I	A/D converter input (A/D input 8)
		O	Peripheral UART TX output (PUART_TX)
F7	P25	I/O	General purpose, user-defined GPIO
		I	SPI1_MISO (master)
		O	SPI1_MISO (slave)
		I	Peripheral UART RX input (PUART_RX)
	P32	I/O	General purpose, user-defined GPIO
		I	A/D converter input (A/D input 7)
		I	X-coordinate output from a quadrature detector (QDX0)
		I	SPI1_CS (slave only)
		O	Auxiliary clock output (ACLK0)

CYW20706 Pin	GPIO	Description	
		I/O Type[1]	Signal Function Options
		O	Peripheral UART TX output (PUART_TX)
F8	P36	I/O	General purpose, user-defined GPIO
		I	A/D converter input (A/D input 3)
		I	Z-coordinate output from a quadrature detector (QDZ0)
		O	SPI1_CLK (master)
		I	SPI1_CLK (slave)
		O	Auxiliary clock output (ACLK0)
		O	T/R switch control (~TX_PD)
	P38	I/O	General purpose, user-defined GPIO
		I	A/D converter input (A/D input 1)
		O	SPI1_MOSI (master)
		I	SPI1_MOSI (slave)
		O	Infrared learning output playback (IR_TX)
G8	P4	I/O	General purpose, user-defined GPIO
		I	Y-coordinate output from a quadrature detector (QDY0)
		I	Peripheral UART RX input (PUART_RX)
		O	SPI1_MOSI (master)
		I	SPI1_MOSI (slave)
		O	Infrared learning output playback (IR_TX)
	P24	I/O	General purpose, user-defined GPIO
		O	SPI1_CLK (master)
		I	SPI1_CLK (slave)
		O	Peripheral UART TX output (PUART_TX)

Table 8-2. CYW20706 Multiplexed GPIO_Pxx Interface Summary

Note: Table 8-2 provides GPIO information for the full capabilities of CYW20706. Some CYW20706-based modules may not support all GPIOs. Check the module schematic and other documentation to determine which GPIOs are available.

8.1.2 Digital I/O Pin Interface Mapping

Table 8-3 shows the mapping of CYW20706 interfaces to the 12 digital I/O pins and the available GPIO_Pxx assignments. This is the same information explained in Table 8-2, but in a more compact format for easier reference with the following description of the selection choices available for an example pin.

To help understand Table 8-3 consider the shaded portion of the table, which pertains to the signal-function selection for pin D6. If an alphanumeric string or an X appears in a signal-function column for a given pin, then that pin can support the signal function. Using this logic, pin D6 can be configured to support one signal function from the following set of signal functions:

- GPIO_P6 (a general-purpose user-defined I/O)
- GPIO_P31 (a general-purpose user-defined I/O)
- PUART_TX (a peripheral UART TX output multiplexed to GPIO_P31)
- PUART_RTS (a peripheral UART request-to-send output multiplexed to GPIO_P6)
- SPI1_CS (a SPI interface chip-select input multiplexed to GPIO_P6)
- A/D converter input (an A/D converter input multiplexed to GPIO_P31)
- 60Hz_mains input (a zero-crossing detector input multiplexed to GPIO_P6)
- Quadrature decoder signal (the output of a quadrature decoder multiplexed to GPIO_P6 as an input)
- External LPO input (LPO_IN) (an external LPO input connected to pin D6). No GPIOs are used in this case.

A system designer will select one signal function from the set of possible signal functions available at pin D6. For example, if the system design requires a peripheral UART, the PUART_TX signal function might be assigned and configured to pin D6 (using GPIO_P31) and the other LHL GPIO, P6 in this case, is configured as input and output disabled See Section 9.4 “Peripheral UART” for details.

For a more detailed example that assigns pins to the other PUART signal functions as well as signal function assignments for the remaining pins in a system design, see Section 10 “Interface Programming Information and Examples”.

Pins		Jumper Position	GPIO_Pxx	PUART_RX	PUART_TX	PUART_RTS	PUART_CTS	SPI1_CLK (Master and Slave)	SPI1_CS (Slave)	SPI1_MOSI (Master and Slave)	SPI1_MISO (Master and Slave)	SPI1_MOSI (Slave)	SPI1_MISO (Slave)	A/D Input	Auxiliary Clock Output (ACLK0)	Auxiliary Clock Output (ACLK1)	IR_RX	IR_TX	KSO3	PWM0	PWM1	PWM2	PWM3	QOC0	QOC1	QOC2	QOC3	60Hz_Main	Triac Control	TX_PD and ~TX_PD	QDX0	QDY0	QDZ0	QDX1	QDY1	QDZ1	I2S_DO/PCM_OUT	I2S_CLK/PCM_CLK	I2S_DI/PCM_IN	I2S_WS/PCM_SYNC	BT_CLK_REQ	BT_DEV_WAKE	BT_HOST_WAKE	LPO_IN	BSC_SDA	BSC_CLK
	A8	J22.8/ J23.6/ J21.2	P3 P29 P35				P3 P35	P3						P29 P35									P29			P29							P3	P35		X							P35	
	B5	J22.3/ J24.1	P15											P15				P15									P15																	
	B6	J22.4/ J23.5	P11 P26						P26					P11					P11	P26				P26					P26																
	B7	J22.5	P2 P28 P37	P2					P2	P2*			P37	P28 P37		P37						P28				P28					P2				P37		X							P37
	C5	J19.5/ J21.7	P33 P27	P33						P27		P33		P33		P33					P27				P27		P27					P33												
	C6	J19.2/ J23.4	P30			P30								P30																														
	C7	J22.7/ J23.5/ J21.1	P12											P12																							X							
	C8	J22.6/ J23.6	P0 P34	P34	P0					P0				P0 P34			P0										P0		P34		P34							X						
	D6	J19.1	P6 P31		P31	P6			P6					P31													P6					P6									X			
	F7	J19.4/ J21.6	P25 P32	P25	P32				P32		P25			P32	P32															P32											X			
	F8	J19.6/ J21.8	P36 P38					P36		P38				P36 P38	P36			P38											P36			P36									X				
	G8	J19.3/ J21.5	P4 P24	P4	P24			P24		P4								P4													P4										X				

Table 8-3. Interface Mapping to the 12 Available Digital I/O Pins

* This instance of P2 is SPI1_MOSI (master only).

An X in the red rectangle indicates that the associated signal functions do not involve any LHL GPIO signals.

The LHL GPIOs in the green rectangle indicate the associated signal functions are available only in those LHL GPIOs.

8.2 LHL GPIO Capabilities

The LHL GPIOs have the following capabilities:

- Each can be programmed to serve one of the signal functions associated with it (see Multiplexed GPIO_Pxx Interface Summary or Table 8-2).
- All can be input and output disabled (HI-Z), input enabled, or output enabled.
- An internal pull-up or pull-down can be configured on input-enabled GPIOs.
- An output-enabled GPIO that is driven HIGH or LOW will remain driven while in the Low Power and Deep Sleep modes.
- All GPIOs, excluding P26–P29, can source or sink up to 8 mA at 3.3 V and 4 mA at 1.8 V.
- GPIOs P26–P29 can source or sink 16 mA at 3.3 V and 8 mA at 1.8 V.
- All GPIOs can be configured for edge (rising/falling/both) or level interrupts.

Below is an example GPIO configuration for an input with a resistive pulldown that is enabled, and a rising edge interrupt:

```
wiced_hal_gpio_configure_pin( gpio_num, (GPIO_INPUT_ENABLE | GPIO_PULL_DOWN |
                              GPIO_EN_INT_RISING_EDGE),GPIO_PIN_OUTPUT_LOW );
```

Application-level interrupt handlers can be configured to handle interrupts in the application thread context and interrupts can be configured to wake the system from the Low Power and Deep Sleep modes. For more information on the Low Power and Deep Sleep modes, see Appendix A “Power-Save Options”.

Note: See Section 10.1 “GPIO Programming Example”, for GPIO programming information, including the enabling and disabling of inputs and outputs, edge triggering, and interrupt configuration.

9 Interface Signal Function Selection Restrictions and Considerations

This section provides signal-function selection restrictions and/or general information associated with the following CYW20706 interfaces:

- I2S and PCM
- SPI1
- SPI2
- HCI UART
- Peripheral UART
- Broadcom Serial Control (BSC) (Compatible with I2C)
- NVRAM

9.1 I²S and PCM

CYW20706 contains I²S and PCM circuit blocks that share a common signal-routing interface.

Table 9-1 shows the shared I²S and PCM interface in pin order. It also shows which GPIO_Pxx cannot be used if the I²S or PCM circuit blocks are used in a board design.

CYW20706 Pin	Signal	When Interface Used, Do Not Use the Following GPIO_Pxx
A8	I2S_DO or PCM_OUT	P0, P2, P3, P12, P28, P29, P34P35, and P37
B7	I2S_CLK or PCM_CLK	
C7	I2S_DI or PCM_IN	
C8	I2S_WS or PCM_SYNC	

Table 9-1. Shared I²S and PCM Interface and GPIO_Pxx Usage Restrictions

9.2 Serial Peripheral Interfaces

CYW20706 supports two serial peripheral interface (SPI) blocks: SPI1 (also known as Spiffy1) and SPI2 (also known as Spiffy2).

The routing of the SPI1 interface is multiplexed using the GPIO_Pxx supported by CYW20706 (see Section 8.1 “GPIO_Pxx”).

As shown in Table 7-1, the SPI2 interface has a fixed signal routing to CYW20706 pins; therefore, it does not require any device GPIO support.

The WICED Studio API allows configuring and controlling both SPI interfaces by providing functions for:

- Clock control
- Mode control
- Data transfer method (half or full duplex)

For more information on the SPI1 interface, see SPI1.

For more information on the SPI2 interface, see SPI2.

9.2.1 SPI1

The application has full control of the SPI1 interface. The SPI1 interface supports:

- SPI clock modes 0 through 4.
- A maximum transaction size of 254 bytes.
- A maximum clock speed of 12 MHz for all I/O supply levels.

  Note: Running the SPI clock at speeds above 12 MHz can lead to undesired behavior.
- Configuration as either a master or a slave.

SPI1 bus-configuration options are detailed in the SPI1 Master and SPI1 Slave sections.

9.2.1.1 SPI1 Master

With SPI1 configured as a master, multiple slaves can be connected to the same bus, where the clock (SPI1_CLK), data input (SPI1_MISO), and data output (SPI1_MOSI) lines comprise the bus.

An LHL GPIO signal will need to be assigned to source each required slave chip-select output.

The application controls Chip Select (CS) line assertion and de-assertion and can use the CS line to optimize transactions greater than 254 bytes.

Note: The software application controls the chip selects using a GPIO driver provided by the WICED Studio API. See Section 10 “Interface Programming Information and Examples” for more information.

Table 9-2 shows CYW20706 SPI1 master bus-configuration options.

Option	SPI1_CLK		SPI1_MOSI		SPI1_MISO	
	Pin	LHL GPIOPxx	Pin	LHL GPIO Pxx	Pin	LHL GPIO Pxx
1	G8	P24	C5	P27	F7	P25
2	G8	P24	F8	P38	F7	P25
3	F8	P36	C5	P27	F7	P25

Table 9-2. CYW20706 SPI1 Master Bus-Configuration Options

Note: The RX signal of a peripheral UART (PUART) has certain restrictions if included in a system design that also uses the SPI1 interface. See Section 9.4 “Peripheral UART” to understand how this restriction affects the bus configuration options of both interfaces.

For a SPI1 master programming example, see Section 10.2 “SPI1 Master Programming Example”.

9.2.1.2 SPI1 Slave

Table 9-3 shows CYW20706 SPI1 slave bus-configuration options.

Option	SPI1_CLK		SPI1_MOSI		SPI1_MISO	
	Pin	LHL GPIO Pxx	Pin	LHL GPIO Pxx	Pin	LHL GPIO Pxx
1	G8	P24	C5	P27	F7	P25
2	G8	P24	C5	P27	B7	P37
3	G8	P24	C5	P33	F7	P25
4	G8	P24	C5	P33	B7	P37
5	G8	P24	F8	P38	F7	P25
6	G8	P24	F8	P38	B7	P37
7	F8	P36	C5	P27	F7	P25
8	F8	P36	C5	P27	B7	P37
9	F8	P36	C5	P33	F7	P25
10	F8	P36	C5	P33	B7	P37

Table 9-3. CYW20706 SPI1 Slave Bus-Configuration Options

Note: The bus signals of a peripheral UART (PUART) has certain restrictions if included in a system design that also uses the SPI1 interface. See Section 9.4 "Peripheral UART" to understand how this restriction affects the bus configuration options of both interfaces.

For a SPI1 slave programming example, see Section 10.3 "SPI1 Slave Programming Example".

9.2.2 SPI2

The SPI2 interface is provided as a CYW20706 connection to nonvolatile memory for accessing configuration, patches, application code, and application data.

The SPI2 interface supports:

- SPI clock modes 0 through 4.
- A maximum transaction size of 254 bytes.
- A maximum clock speed of 12 MHz for all I/O supply levels.
- Operation as a master only.
- The fixed-signal interface shown in Table 7-1.

The firmware controls CS line assertion and de-assertion and can use the CS line to optimize transactions greater than 254 bytes.

For information on the API functions that support SPI2 access to attached serial flash or EEPROM, see WICED Studio header files *wiced_hal_sflash.h* or *wiced_hal_seeprom.h*, respectively.

9.3 HCI UART

CYW20706 supports an HCI UART (also referred to as BT_UART). The following information applies to the HCI UART:

- It is primarily used for programming and factory testing, but can also be used to support SWD debugging (see below).
- It is a fixed (non-multiplexed) interface (see Table 7-1).

- In CYW20706, it is available only for application-defined HCI commands and events; it is not for Bluetooth standard HCI commands. In CYW20706, it is used to communicate with the external MCU and does support Bluetooth standard HCI commands.
- It supports a maximum baud rate of 3 Mbps. To configure for 3 Mbps use WICED Studio API:

  ```
  uart_SetBaudrate(0, 0, 3000000)
  ```

- Debug messages can be routed to the HCI UART. To do so, use:

  ```
  wiced_set_debug_uart(wiced_debug_uart_types_t uart) // with
                                                      // uart=WICED_ROUTE_DEBUG_TO_HCI_UART
  ```

The API functions that support the HCI UART are defined in *wiced_hci.h*.

9.3.1 SWD Debugging with the HCI UART

The HCI UART interface may be used for SWD debugging. When used for debugging, the normal HCI UART function may not be used. The device FW will auto-detect the presence of an SWD debugger if the SWDIO signal is connected to HCI UART pin BT_UART_RXD and the SWDCLK signal is connected to BT_UART_TXD. In order to do this on the CYW920706WCDEVAL board, the FTDI chip must be bypassed as described below in order to connect the HCI UART pins to the SWD debugger.

9.3.2 Bypassing the FTDI chip

Bypassing the FTDI chip on the CYW920706WCDEVAL board can be necessary in scenarios where the HCI UART interface needs to be used for another purpose, such as connecting to an external MCU, or if an SWD debugger is connected to the HCI UART. In those cases, the UART must be disconnected from the FTDI chip to avoid contention. The FTDI chip can be bypassed and the board can be powered in 2 different ways:

- Powering the board from USB
  1. Remove All jumpers on J26
  2. Plug in the USB cable to power up the board as usual
- Powering the board using External supply
  1. Remove All jumpers on J26
  2. Supply 3.3V to TP1

9.4 Peripheral UART

CYW20706 supports a peripheral UART (PUART). The following information applies to the peripheral UART:

- It is primarily used to interface with peripheral devices (for example sensors) or a microcontroller, but can also be used to support debugging.
- It supports a standard two-wire serial interface (RX and TX) without hardware flow control and a four-wire serial interface (RX, TX, RTS, and CTS) with hardware flow control.

  Note: When hardware flow control is not used, the application is responsible for servicing the TX FIFO before it empties and the RX FIFO before it fills. If the application does not service the FIFOs, receive-data can be lost and transmit-gaps can occur.
- The default baud rate is 115200 bps, but other baud rates, such as 19200 bps, 38400 bps, 57600 bps, and 3 Mbps, are also supported.
- The RX and TX FIFO hardware buffer size is 256 bytes.
- Table 9-4 shows CYW20706 PUART bus-configuration options when a system design also uses the SPI1 interface provided by CYW20706. In such a system, PUART_RX must be on the same group of pins (referred to as a “pad bank”) as the SPI1 bus signals. Since a full SPI1 bus can only be supported on a certain group of pins, all SPI1 bus signals and PUART_RX must be on the “upper pad bank”, which is P24 through P39.

If the CYW20706 SPI1 interface is not used, then several more PUART bus-configuration options (in addition to those shown in Table 9-4) are available. See Peripheral UART to determine the other available PUART bus-configuration options.

Option	PUART_RX		PUART_TX		PUART_RTS		PUART_CTS	
	Pin	LHL GPIO Pxx	Pin	LHL GPIO Pxx	Pin	LHL GPIO Pxx	Pin	LHL GPIO Pxx
1	C5	P33	C8	P0	C6	P30	A8	P3
2	C5	P33	C8	P0	C6	P30	A8	P35
3	C5	P33	C8	P0	D6	P6	A8	P3
4	C5	P33	C8	P0	D6	P6	A8	P35
5	C5	P33	D6	P31	C6	P30	A8	P3
6	C5	P33	D6	P31	C6	P30	A8	P35
7	C5	P33	F7	P32	C6	P30	A8	P3
8	C5	P33	F7	P32	C6	P30	A8	P35
9	C5	P33	F7	P32	D6	P6	A8	P3
10	C5	P33	F7	P32	D6	P6	A8	P35
11	C5	P33	G8	P24	C6	P30	A8	P3
12	C5	P33	G8	P24	C6	P30	A8	P35
13	C5	P33	G8	P24	D6	P6	A8	P3
14	C5	P33	G8	P24	D6	P6	A8	P35
15	C8	P34	D6	P31	C6	P30	A8	P3
16	C8	P34	D6	P31	C6	P30	A8	P35
17	C8	P34	F7	P32	C6	P30	A8	P3
18	C8	P34	F7	P32	C6	P30	A8	P35
19	C8	P34	F7	P32	D6	P6	A8	P3
20	C8	P34	F7	P32	D6	P6	A8	P35
21	C8	P34	G8	P24	C6	P30	A8	P3
22	C8	P34	G8	P24	C6	P30	A8	P35
23	C8	P34	G8	P24	D6	P6	A8	P3
24	C8	P34	G8	P24	D6	P6	A8	P35
25	F7	P25	C8	P0	C6	P30	A8	P3
26	F7	P25	C8	P0	C6	P30	A8	P35
27	F7	P25	C8	P0	D6	P6	A8	P3
28	F7	P25	C8	P0	D6	P6	A8	P35
29	F7	P25	D6	P31	C6	P30	A8	P3
30	F7	P25	D6	P31	C6	P30	A8	P35
31	F7	P25	G8	P24	C6	P30	A8	P3

Option	PUART_RX		PUART_TX		PUART_RTS		PUART_CTS	
	Pin	LHL GPIO Pxx	Pin	LHL GPIO Pxx	Pin	LHL GPIO Pxx	Pin	LHL GPIO Pxx
32	F7	P25	G8	P24	C6	P30	A8	P35
33	F7	P25	G8	P24	D6	P6	A8	P3
34	F7	P25	G8	P24	D6	P6	A8	P35

Table 9-4. CYW20706 PUART Bus-Configuration Options When PUART_RX Is on the Upper Pad Bank

PUART bus-configuration options presented in Table 9-4 will be further reduced by the selected SPI1 bus-configuration option.

For example, SPI1 master bus-configuration option 2 (from Table 9-2) requires digital I/O pins G8, F8, and F7. Therefore, all PUART bus-configuration options (in Table 9-4) that use G8, F8, or F7 cannot be used if SPI1 master bus-configuration option 2 is used. So the remaining PUART bus-configuration options become 1–6, 15, and 16.

9.5 Broadcom Serial Control (BSC) (Compatible with I²C)

CYW20706 supports a Broadcom Serial Control (BSC) interface.

Note: BSC is a proprietary Cypress interface that is compatible with I²C.

The following information applies to the BSC interface:

- There are three possible choices for the mapping of BSC signals (SDA and SCL) to CYW20706 pins as shown in the table below. The parameter used for the *wiced_hal_i2c_init* function call for each selection of pins is also provided.

SDA	SCL	Configuration Parameter Name
(Pin C7) I2S_DIN/PCM_IN/SDA[1]	(Pin A8) I2S_DOUT/PCM_OUT/SCL[1]	WICED_I2C_SDA_I2S_DOUT_PCM_OUT_SCL_I2S_DIN_PCM_IN
(Pin A8) P35	(Pin B7) P37	WICED_I2C_SDA_P35_SCL_P37
(Pin E8) SFLASH_MOSI [2]	(Pin D8) SFLASH_MISO[2]	WICED_I2C_SDA_SFLASH_MOSI_SCL_SFLASH_MISO

Table 9-5. BSC Signals Pins and Parameters

- The interface supports the following transfer types:
  - ☐ Read
  - ☐ Write
  - ☐ Combined write then read

    Note: The BSC block generates a repeated START condition between the two parts of a combined transfer.
- The maximum transaction length is 64 bytes.
- Both low-speed and fast-mode slaves are supported at a maximum clock speed of 4000 kbps. The maximum clock speed is 2400 kbps for slaves that use clock-cycle stretching.

  Note: Clock speeds may be less than the speeds indicated above (for example, if an external pull-up resistor is used and it affects transmission time).
- Multi-master bus mode is not supported and, thus, CYW20706 must be the only bus master.
- Only 7-bit slave addresses are supported.
- Information on the API functions that support the interface is in *wiced_hal_i2c.h*.

[1] These pins have a 4.7K pull up on the CYW920706WCDEVAL kit.

[2] These pins cannot be used for BSC by default on the CYW920706CDEVAL kit because they are used for SPI communication with the serial flash.

For a BSC programming example, see Section 10.4 “BSC Programming Example”.

9.6 NVRAM

CYW20706 has NVRAM, which can be used to save the state of the device while power is off.

For example, an application can use the NVRAM to save the Bluetooth Device address of a paired device. During a future connection establishment, the application can check whether the connecting device is paired or not.

10 Interface Programming Information and Examples

Note: All code and code references in this section pertain to WICED Studio. Source C files may be found under the appropriate application folder in the Project Explorer window under 20706-A2_Bluetooth\apps, and header H files which contain definitions for the #define's used in the example code may be found under 20706-A2_Bluetooth\include. See [2] for detailed information on using WICED Studio to build and run the sample applications mentioned in the subsections below, including viewing application trace message output generated from WICED_BT_TRACE().statements shown in the example code.

Consider the following information after physically assigning GPIO_Pxx to digital I/O pins:

- When CYW20706 is in the Low Power or Deep Sleep modes, no data can be received from any of its interfaces (SPI, PUART, etc.). To wake CYW20706, use a GPIO_Pxx configured as an interrupt. Refer to *wiced_hal_gpio.h* and *gpiodriver.h* for the pertinent API functions and constants. See Appendix A "Power-Save Options" for more information about the sleep modes and power-save options.
- To access GPIO driver services invoke the `wiced_hal_gpio_init` function during application initialization. This is independent of other functions and must be called before any other GPIO functions.
- For a sample application that pertains to GPIO API function usage refer to the *hal_gpio_app.c* file in WICED Studio.
- For information on the API functions that support SPI1 configuration, control, and data exchange, refer to *wiced_hal_pspi.h*. To generate chip-select (CS) signals to attached slaves, refer to *wiced_hal_gpio.h*.
- For information on the API functions that pertain to PUART access, refer to *wiced_hal_puart.h*.
- The remaining paragraphs in this section provide the following programming examples:
  - ☐ GPIO Programming Example
  - ☐ SPI1 Master Programming Example
  - ☐ SPI1 Slave Programming Example
  - ☐ BSC Programming Example
  - ☐ PUART Programming Example
  - ☐ NVRAM Programming Example

10.1 GPIO Programming Example

The example in this section demonstrates some aspects of GPIO programming, including GPIO driver initialization, configuring GPIOs for input and output, and registering interrupt handlers. Refer to *wiced_hal_gpio.h* for the complete set of API functions that support GPIO programming. See the `hal_gpio_app` sample application in WICED Studio under *20706-A2_Bluetooth\apps\snip\hal_gpio* for a complete working program using these APIs.

To initialize the GPIO driver, invoke the `wiced_hal_gpio_init` function. The GPIO driver should be one of the first to be initialized.

The following example code is similar to the code found in *hal_gpio_app.c*.

```
APPLICATION_START( )
{
    wiced_set_debug_uart( WICED_ROUTE_DEBUG_TO_PUART);
    wiced_hal_puart_select_uart_pads( WICED_PUART_RXD, WICED_PUART_TXD, 0, 0);

    WICED_BT_TRACE( "GPIO application start\n\r" );

    wiced_bt_stack_init( sample_gpio_app_management_cback,NULL,NULL);
}

wiced_result_t sample_gpio_app_management_cback( wiced_bt_management_evt_t event,
                                                 wiced_bt_management_evt_data_t *p_event_data)
{
    wiced_result_t result = WICED_SUCCESS;
```

```
    WICED_BT_TRACE( "sample_gpio_app_management_cback %d\n\r", event );

    switch( event )
    {
        /* Bluetooth  stack enabled */
        case BTM_ENABLED_EVT:
            /* Initializes the GPIO driver */
            wiced_hal_mia_init( );
            wiced_hal_gpio_init( );
            wiced_hal_mia_enable_mia_interrupt( TRUE );
            wiced_hal_mia_enable_lhl_interrupt( TRUE );

            /* Sample function configures LED pin as output
             * sends a square wave on it
             * if testing pin 31 then disable puart*/
            gpio_test_led( );

            /* Sample function configures GPIO as input. Enable interrupt.
             * Register a call back function to handle on interrupt*/
            gpio_set_input_interrupt( );

            break;
        default:
            break;
    }
    return result;
}

void gpio_interrrupt_handler(void *data, uint8_t port_pin)
{
    static uint32_t prev_blink_interval = APP_TIMEOUT_IN_SECONDS_A;
    uint32_t curr_blink_interval = 0;

     /* Get the status of interrupt on P# */
    if ( wiced_hal_gpio_get_pin_interrupt_status( WICED_GPIO_BUTTON ) )
    {
        /* Clear the gpio interrupt */
        wiced_hal_gpio_clear_pin_interrupt_status( WICED_GPIO_BUTTON );
    }

    /* stop the led blink timer */
    wiced_stop_timer( &seconds_timer );

    /* toggle the blink time interval */
    curr_blink_interval = (APP_TIMEOUT_IN_SECONDS_A == prev_blink_interval)
                          ?APP_TIMEOUT_IN_SECONDS_B:APP_TIMEOUT_IN_SECONDS_A;

    wiced_start_timer( &seconds_timer, curr_blink_interval );

    WICED_BT_TRACE("gpio_interrupt_handler : %d\n\r", curr_blink_interval);

    /* update the previous blink interval */
    prev_blink_interval = curr_blink_interval;
}

void gpio_set_input_interrupt( )
{
    uint16_t pin_config;

    /* Configure GPIO PIN# as input, pull up and interrupt on rising edge and output
     * value as high (pin should be configured before registering interrupt handler )*/
    wiced_hal_gpio_configure_pin( WICED_GPIO_BUTTON, WICED_GPIO_BUTTON_SETTINGS(
                          GPIO_EN_INT_RISING_EDGE ), WICED_GPIO_BUTTON_DEFAULT_STATE );
```

```
    wiced_hal_gpio_register_pin_for_interrupt( WICED_GPIO_BUTTON,
                          gpio_interrrupt_handler, NULL );

    /* Get the pin configuration set above */
    pin_config = wiced_hal_gpio_get_pin_config( WICED_GPIO_BUTTON );
    WICED_BT_TRACE( "Pin config of P%d is %d\n\r", WICED_GPIO_BUTTON, pin_config );
}

/* The function invoked on timeout of app seconds timer. */
void seconds_app_timer_cb( uint32_t arg )
{
    wiced_timer_count++;
    WICED_BT_TRACE( "seconds periodic timer count: %d s\n", wiced_timer_count );

    if(wiced_timer_count & 1)
    {
        wiced_hal_gpio_set_pin_output( LED_GPIO_1, GPIO_PIN_OUTPUT_LOW);
    }
    else
    {
        wiced_hal_gpio_set_pin_output( LED_GPIO_1, GPIO_PIN_OUTPUT_HIGH);
    }
}

void gpio_test_led( )
{
    WICED_BT_TRACE( "gpio_test_led\n\r" );

    /* Configure LED PIN as input and initial outvalue as high */
    wiced_hal_gpio_configure_pin( LED_GPIO_1, GPIO_OUTPUT_ENABLE, GPIO_PIN_OUTPUT_HIGH );

    if ( wiced_init_timer( &seconds_timer, &seconds_app_timer_cb, 0,
                           WICED_SECONDS_PERIODIC_TIMER )== WICED_SUCCESS )
    {
        if ( wiced_start_timer( &seconds_timer, APP_TIMEOUT_IN_SECONDS_A )
                           !=  WICED_SUCCESS )
        {
            WICED_BT_TRACE( "Seconds Timer Error\n\r" );
        }
    }
}
```

10.2 SPI1 Master Programming Example

The following code shows how to initialize SPI1 as a master, write a byte to a SPI slave, and read a byte from a SPI slave.

```
#include "wiced_hal_gpio.h"
#include "wiced_hal_pspi.h"

#define SPIFFY_SPEED            1000000         /* Use 1M speed */
#define SPIFFY_CS_ASSERT        1
#define SPIFFY_CS_DEASSSERT     0
uint8_t test_spiffy1_master_send_receive_byte( uint8_t byteToSend );

void test_pspi_driver( void )
{
    /* Reset spi hardware block and set to default config*/
    wiced_hal_pspi_reset( );

    /* Initialize spiffy1 in master role */
    /* available pin combinations are in spiffydriver.h */
```

```
    wiced_hal_pspi_init( MASTER, GPIO_PULL_UP, MASTER1_P24_CLK_P27_MOSI_P25_MISO,
        SPIFFY_SPEED, SPI_MSB_FIRST, SPI_SS_ACTIVE_LOW, SPI_MODE_3, WICED_GPIO_33 );

    /* Send a byte and receive a byte from slave*/
    test_spiffy1_master_send_receive_byte( 1 );
}


/* Sends one byte and receives one byte from the SPI slave.
 * byteToSend - The byte to send to the slave.
 * Returns the byte received from the slave.
 */
uint8_t test_spiffy1_master_send_receive_byte( uint8_t byteToSend )
{
    uint8_t byteReceived;

    /* Assert chipselect by driving O/P on the GPIO */
    wiced_hal_gpio_set_pin_output( WICED_GPIO_15, SPIFFY_CS_ASSERT);

    /* Tx one byte of data */
    wiced_hal_pspi_tx_data( 1, &byteToSend );

    /* Rx one byte of data */
    wiced_hal_pspi_rx_data( 1, &byteReceived );

    /* Deassert chipselect */
    wiced_hal_gpio_set_pin_output( WICED_GPIO_15, SPIFFY_CS_DEASSSERT );

}
```

10.3 SPI1 Slave Programming Example

The following code shows how to initialize SPI1 as a slave, write a byte to a SPI master, and read a byte from a SPI master.

```
#include "wiced_hal_pspi.h"

#define SLAVE1_P36_CS_P24_CLK_P27_MOSI_P25_MISO         0x24181b19      /* Refer Hardware
peripherals doc */
#define SPIFFY_SPEED                                    1000000         /* Use 1M speed */

void test_pspi_driver( void )
{
    /* Initialize spiffy1 in slave role, available pin combinations are in spiffydriver.h
    DO NOT CONFIGURE csPin in SLAVE mode - the HW takes care of this.  There is no need
    to configure the speed too - the master selects the speed.
    */

    wiced_hal_pspi_init( SLAVE, GPIO_PULL_DOWN,
        SLAVE1_P36_CS_P24_CLK_P27_MOSI_P25_MISO,
        SPIFFY_SPEED, SPI_MSB_FIRST,
        SPI_SS_ACTIVE_LOW, SPI_MODE_3,
        WICED_GPIO_03 );


    /* Rx a byte from master, increment and Tx it back to master*/
    test_spiffy1_slave_send_receive_byte( );
}

/* Receives a byte from the SPI master, increments the byte and sends it back
 * byteToSend - The byte to send to the slave.
 * Returns the byte received from the slave.
```

```
 */
uint8_t test_spiffy1_slave_send_receive_byte( void )
{
    uint8_t byteReceived;

    /* Rx one byte of data */
    wiced_hal_pspi_rx_data( 1, &byteReceived );

    /* Send back byteReceived + 1 */
    byteReceived++;

    /* Tx one byte of data */
    wiced_hal_pspi_tx_data( 1, &byteReceived );

}
```

10.4 BSC Programming Example

The following example shows how to initialize the BSC as a master, and how to write, read, and use the combination write-then-read transactions.

```
#include "wiced_hal_i2c.h"

#define I2C_SLAVE_ADDRESS               (0x1A)   /* Use driver slave address 0x1A =
                                                  * (7'b0001101 << 1) | 1'bR|W */
#define I2C_SLAVE_OPERATION_READ        0        /* Read operation to the lower level
                                                  * driver is 0 */
#define I2C_SLAVE_OPERATION_WRITE       1        /* Write operation to the lower
                                                  * level driver is 1 */

/* Sample code to test i2c driver */
void test_i2c_driver( void )
{
    uint8_t data_array[] = "123456";

    /* Initializes the I2C driver and its private values. This initialization
     * sets the bus speed to 100KHz by default (I2CM_SPEED_100KHZ)*/
    wiced_hal_i2c_init( );

    /* current I2C bus speed */
    wiced_hal_i2c_get_speed( );

    /* Sets the I2C bus speed
     *(I2CM_SPEED_100KHZ/ I2CM_SPEED_400KHZ/ I2CM_SPEED_800KHZ/ I2CM_SPEED_1000KHZ)*/
    wiced_hal_i2c_set_speed( I2CM_SPEED_400KHZ );

    /* Writes the given data to the I2C HW addressing a particular slave address */
    wiced_hal_i2c_write( data_array, sizeof( data_array ), I2C_SLAVE_ADDRESS);

    /* Reads data into given buffer from the I2C HW addressing
     * a particular slave address */
    wiced_hal_i2c_read( data_array, sizeof( data_array ), I2C_SLAVE_ADDRESS);
    WICED_BT_TRACE( "Read bytes %s\n\r", data_array );
}
```

10.5 PUART Programming Example

The following example shows how to initialize the peripheral UART. The complete working project for this example can be found in WICED Studio under Project Explorer *20706-A2_Bluetooth\apps\snip\hal_puart*. Pertinent sections of source file *hal_puart_app.c* are shown here:

```
#include "wiced_hal_puart.h"

void puar_rx_interrupt_callback(void* unused)
{
    // There can be at most 16 bytes in the HW FIFO.
    uint8_t  readbyte;

    wiced_hal_puart_read( &readbyte );

    /* send one byte via the TX line. */
    wiced_hal_puart_write( readbyte+1 );

    if( readbyte == 'S' )
    {
        /* send a string of characters via the TX line */
        wiced_hal_puart_print( "\nYou typed 'S'.\n" );
    }
    wiced_hal_puart_reset_puart_interrupt( );

}

/* Sample code to test puart driver. Initializes puart, selects puart pads,
 * turn off flow control, and enables Tx and Rx.
 * Echoes the input byte with increment by 1.
 */
void test_puart_driver( void )
{
    uint8_t read_5_bytes[5];

    wiced_hal_puart_init( );

    // Possible uart tx and rx combination.
    // Pin for Rx: p2, Pin for Tx: p0
    // Note that p2 and p0 might not be available for use on your
    // specific hardware platform.
    // Please see the User Documentation to reference the valid pins.
    wiced_hal_puart_select_uart_pads( WICED_PUART_RXD, WICED_PUART_TXD, 0, 0);
    /* Turn off flow control */
    wiced_hal_puart_flow_off( );  // call wiced_hal_puart_flow_on(); to turn on flow
                                  // control

    // BEGIN - puart interrupt
    wiced_hal_puart_register_interrupt(puar_rx_interrupt_callback);

    /* Turn on Tx */
    wiced_hal_puart_enable_tx( ); // call wiced_hal_puart_disable_tx to disable
                                  // transmit capability.
    wiced_hal_puart_print( "Hello World!\r\nType something! Keystrokes are echoed to
        the terminal ...\r\n");

    /* Enable to change puart baud rate. eg: 9600, 19200, 38200 */
    //wiced_hal_puart_set_baudrate( 115200 );

}
```

10.6 NVRAM Programming Example

The following code sample provides an example of writing a Bluetooth device address to NVRAM and reading it back.

```
#define APP_VS_ID      WICED_NVRAM_VSID_START
BD_ADDR bd_addr_write = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06};
BD_ADDR bd_addr_read;

void test_nvram_read_write_app( )
{
    uint8_t written_bytes, read_bytes;
    wiced_result_t status;

    /* Write BD ADDR to NVRAM */
    written_bytes = wiced_hal_write_nvram( APP_VS_ID, BD_ADDR_LEN, bd_addr_write,
        &status );
    WICED_BT_TRACE("status of nvram write %d, number of bytes written %d\n", status,
        written_bytes);

    /* Read BD ADDR from NVRAM */
    read_bytes = wiced_hal_read_nvram( APP_VS_ID, BD_ADDR_LEN, bd_addr_read,
        &status );
    WICED_BT_TRACE( "status of nvram read %d, number of bytes read %d\n", status,
        read_bytes);
    wiced_bt_trace_array( "BD_ADDR read back : \n", bd_addr_read, BD_ADDR_LEN);
}
```

Appendix A. Power-Save Options

The WICED Studio API provides two power-save options for CYW20706, each initiated by a function call. The power-save options are:

- Low Power Sleep Mode – reduced power consumption, device configured to sleep when idle, application is suspended.
- Deep Sleep Power Save Mode - minimal power consumption, device largely inactive, application exits.
- Applications can make use of either or both power save modes. Typically, the Deep Sleep Power Save Mode is used when the battery level falls below a critical threshold. See below for details on each power save mode.

  Note: Refer to *wiced_power_save.h* in WICED Studio for all API functions and constants related to the available power-save options.

A.1 Low Power Sleep Mode

Use `wiced_sleep_config` to configure CYW20706 to automatically enter and exit Low Power Sleep Mode operation. In this mode, the device achieves power savings by suspending the application and allowing the device to sleep between BLE advertising intervals.

Configuring Low Power Sleep Mode does not immediately put the device in low power mode, it simply enables the device to transition to Low Power mode whenever idle. The application is suspended and resumed transparently by the device when it enters and exits sleep.

The device wakeup can be triggered by any activity on an active peripheral, GPIO, the BT radio, or an interrupt from a timer started by the application.

To enable Low Power Sleep Mode operation, call:

```
wiced_sleep_config(WICED_TRUE, NULL, NULL);
```

To disable Low Power Sleep Mode operation, call:

```
wiced_sleep_config(WICED_FALSE, NULL, NULL);
```

The second and third arguments are not needed for embedded SoC applications, they are only used in external MCU scenarios where BT_DEV_WAKE (F8) can be used by an external MCU to wake the device, and BT_HOST_WAKE can be used by the device as a wake signal to an external MCU.

A.2 Deep Sleep Power Save Mode

The APIs used to utilize Deep Sleep Power Save Mode are described below, followed by code snippets showing their usage.

Use `wiced_power_save_start` to instruct CYW20706 to enter the Deep Sleep Power Save Mode when idle.

The `wake_source` parameter in `wiced_power_save_start` indicates the wake-up source. The available choices are:

- `WICED_WAKE_SOURCE_GPIO` - wake if any of the multiplexed GPIOs, also referred to as the LHL GPIOs (or GPIO_Pxx), transition to an active state (positive edge trigger).
- `WICED_WAKE_SOURCE_TIMEOUT` - wake based on a timeout
- `WICED_WAKE_SOURCE_ALL` - wake either based on GPIOs or a timeout

The transition to Deep Sleep mode causes the currently running application to exit, so care must be taken to save any state information that will be needed by the application upon waking up. When the device wakes from this power-save mode, the application will be restarted. Therefore, an application utilizing this mode should retrieve any such saved state information from a previous incarnation on every startup.

Use the following two API functions to save and restore application state information:

- `wiced_power_save_store_state` (use before transitioning into power-save mode)
- `wiced_power_save_retrieve_state` (use after waking from power-save mode)

Configuring Deep Sleep Power Save Mode does not immediately put the device to sleep, rather, it instructs the device that it should transition to deep sleep if possible. The device will try to transition to deep sleep as soon as it determines the system is idle, or until the operation has been aborted by an internal component.

Applications can register callback functions to be notified of deep sleep transition attempts, possible transition aborts, and to allow or disallow the actual transition to deep sleep:

- `wiced_power_save_register_approve_cback` – register callback to approve or disapprove transition

The callback registered with this function is called when the device has decided to transition to deep sleep, to allow an application to approve or disapprove of CYW20706 transitioning to the power-save mode. The application may allow the transition by returning a non-zero value from the callback, or return zero to disallow. If the transition is disallowed, the original request to transition to deep sleep remains active, and the device will continue to request the transition with further invocations to this callback. Applications may choose to disallow to temporarily delay the sleep transition until the application is ready.

- `wiced_power_save_register_enter_cback` – register callback to be notified of imminent transition

The callback registered with this function is called just before CYW20706 transitions to the deep sleep power save mode to allow the application to save state information with `wiced_power_save_store_state`.

- `wiced_power_save_register_abort_cback` – register callback to be notified if a pending transition is aborted.

The callback registered with this function is invoked if a transition to the power-save mode is aborted by the device. In this state, the original request to transition to deep sleep is no longer active. An application can react to a power-save abort by once again invoking `wiced_power_save_start`.

Use `wiced_power_save_stop` to instruct CYW20706 to cancel a previously requested deep sleep transition. This can be called by the application any time after the call to `wiced_power_save_start` and before the callback registered for transition entry notification is called (from `wiced_power_save_register_enter_cback`). Once that notification callback has been invoked, it is too late to try to stop the transition.

The following code examples show how to use the `wiced_power_save_*` API functions:

```
uint16_t persistent_state_info = 0; /* application specific info to survive sleep */

/* normal application initializations (GPIOs, etc) occur from the callback function
 * registered with wiced_bt_stack_init() handling BTM_ENABLED_EVT notification.
 * call this function from that context after those initialization */
void deep_sleep_init(void)
{
    /* retrieve stored state from previous execution */
    stored_state_info = wiced_power_save_retrieve_state());

    /* register callbacks */
    wiced_power_save_register_approve_cback(power_save_approve);
    wiced_power_save_register_enter_cback(power_save_enter);
    wiced_power_save_register_abort_cback(power_save_abort);
}

/* function to be called when app wants to init deep sleep,
 * e.g. battery monitor or idle time handler */
void sample_interrupt_handler(void)
{
    /* request deep sleep, to be woken either on GPIO interrupt or a 10000ms timeout */
    wiced_power_save_start(WICED_WAKE_SOURCE_ALL, 10000);
}

uint32_t power_save_approve(void)
{
    if( isAppWillingToSleepNow )
    {
        WICED_BT_TRACE("Approving deep sleep request...\n");
        return ~0;
    }
    else /* not willing to approve */
    {
        /* app will return 0 to disapprove, which will allow device to retry */
        /* unless we want to stop it now */
        if ( doesAppWantToCancelSleep )
        {
```

```
            WICED_BT_TRACE("Cancelling deep sleep mode now...\n");
            wiced_power_save_stop();
        }
        else
        {
            WICED_BT_TRACE("Disapproving deep sleep request for now to delay it...\n");
        }
        return 0;
    }
}

void power_save_enter(void)
{
    /* too late to try to stop it now, can only store state */
    wiced_power_save_store_state(persistent_state_info);
    WICED_BT_TRACE("Entering deep sleep now...\n");
}

void power_save_abort(void)
{
    /* device has aborted the deep sleep, it will not retry further unless requested */
    WICED_BT_TRACE("Deep Sleep request aborted.\n");

    if ( doesAppStillWantToSleep )
    {
        WICED_BT_TRACE("Re-init deep sleep request...\n");
        wiced_power_save_start(WICED_WAKE_SOURCE_ALL, 10000);
    }
}
```

References

Document (or Item) Name	Number	Source
[1] CYW20706 Embedded Bluetooth 4.2 SoC with MCU, Bluetooth Transceiver, and Baseband Processor	002-14790	community.cypress.com
[2] WICED CYW920706WCDEVAL Kit Guide	002-18191	community.cypress.com

Document Revision History

Document Title: CYW920706WCDEVAL Hardware User Guide

Document Number: 002-16535

Revision	ECN	Issue Date	Description of Change
**	5829476	07/26/2017	Initial release

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 | PSoC 6

Cypress Developer Community

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

Technical Support

cypress.com/support

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