AN216761

CYW2070x Hardware Interface Selection And Programming

Associated Part Family: CYW2070x
WICED™ Studio 4
To get the latest version of this application note, please visit www.cypress.com/AN216761

This document primarily identifies the full complement of interfaces available on the CYW2070x (20706 and 20707) and interface- selection restrictions and consequences. In addition, it provides:

- The interface options pertinent to the Cypress® CYW92070xV3_EVAL reference design.
- API programming information examples for configuring various CYW92070xV3_EVAL interfaces (SPI, I2C, peripheral UART, and more).

Contents

1 Introduction

The CYW2070x (CYW20706 or CYW20707) is an embedded Bluetooth (BT) 4.2 device that supports a wide range of products, including home automation gateways, monitoring devices (heart rate, blood pressure, etc.), wearables, and more. The CYW20706 is the fully embedded version of the device running an embedded BT stack, while the CYW20707 is the same device but operates as a standalone BT Controller, and communicates with an external MCU with external BT stack via the HCI UART. This document applies to both versions except specific sections where noted.

To accommodate its various applications, the CYW2070x supports some fixed interfaces plus a set of selectable interfaces. The selectable interfaces are chosen from a superset of chip-supported interfaces. Interface selection is accomplished through a combination of signal routing to the CYW2070x and programming.

2 CYW2070x Interfaces

The CYW2070x interfaces are presented as two interface sets: fixed and selectable. The fixed interfaces are those with dedicated pins and, thus, always available. The selectable interfaces are those that a board designer chooses to use from a superset of possible interfaces that the CYW2070x supports.

See Section 2.1 “Fixed Interfaces” for more information on the fixed interfaces. See Section 2.2 “Selectable Interfaces” for more information on the set of interfaces from which board designers can choose.

2.1 Fixed Interfaces

Table 1 shows the fixed straps and digital I/O interfaces of the CYW2070x. In contrast to the Selectable Interfaces, the fixed interfaces of the CYW2070x have dedicated pins.

Table 1. Fixed Straps and Digital I/O Interfaces

<table>
<tr><th>Strap or Interface Purpose</th><th>Signals</th><th>Pins</th><th colspan="3">Notes</th></tr>
<tr><td rowspan="5">XTAL frequency selection</td><td>BT_XTAL_STRAP_0</td><td>G3</td><td>Strap_1</td><td>Strap_0</td><td>XTAL Option</td></tr>
<tr><td rowspan="4">BT_XTAL_STRAP_1</td><td rowspan="4">F2</td><td>0</td><td>0</td><td>40 MHz</td></tr>
<tr><td>0</td><td>1</td><td>24 MHz</td></tr>
<tr><td>1</td><td>0</td><td>20 MHz</td></tr>
<tr><td>1</td><td>1</td><td>Read from NV memory</td></tr>
<tr><td>Device reset</td><td>RST_N</td><td>A6</td><td colspan="3">Active low reset input</td></tr>
<tr><td>OTP usage</td><td>BT_OTP_3P3V_ON</td><td>G2</td><td colspan="3">Pull high if OTP is used; otherwise, pull low.</td></tr>
<tr><td rowspan="4">HCI UART</td><td>BT_UART_RXD</td><td>F5</td><td colspan="3">HCI UART receive data</td></tr>
<tr><td>BT_UART_TXD</td><td>F4</td><td colspan="3">HCI UART transmit data</td></tr>
<tr><td>BT_UART_RTS_N</td><td>F3</td><td colspan="3">HCI UART request-to-send input</td></tr>
<tr><td>BT_UART_CTS_N</td><td>G4</td><td colspan="3">HCI UART clear-to-send input</td></tr>
</table>

Strap or Interface Purpose	Signals	Pins	Notes
Serial Peripheral Interface (SPI) or 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

2.2 Selectable Interfaces

The CYW2070x supports several other interfaces besides those identified in Fixed Interfaces. Although the CYW2070x 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 get multiplexed to 12 digital I/O pins. In pin order, 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 the CYW2070x:

- Multiple general purpose I/Os (GPIOs), referred to as GPIO_Pxx or the LHL GPIOs.
- For more information on GPIO signals, see Section 3 “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).
- A 60 Hz input (60Hz_main) to a zero-crossing detector.
- Two triac control outputs.
- Two external T/R switch control outputs (TX_PD and ~TX_PD).
- Multiple inputs from quadrature detectors (QDX0, QDX1, QDY0, QDY1, QDZ0, and QDZ1).

[1]Use the SPI2 interface for applications where the CYW2070x 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 I^2C-compatible interface via pins D8 and E8, an I^2C-compatible interface is not available via these pins. The CYW2070x does support an I^2C-compatible interface (BSC interface) via pins A8 and B7 (see Section 4.5 “Broadcom Serial Control”).

- 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.
- A Broadcom Serial Control (BSC) interface.

The above set of selectable interfaces are mapped to the 12 digital I/O pins shown in Figure 1. The figure shows that many signal functions are supported at each pin, but a system designer must select a single function per pin.

Note: For help determining signal-function assignments to the 12 available I/O pins, see Section 3 “GPIO Information”.

Figure 1: Interface Mapping to the 12 Available Digital I/O Pins

Pins	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	P3 P29 P35				P3 P35	P3						P29 P35									P29				P29							P3	P35		A8								P35	
B5	P15											P15			P15											P15																		
B6	P11 P26						P26					P11					P11	P26				P26					P26																	
B7	P2 P28 P37	P2					P2	P2*			P37	P28 P37		P37						P28				P28					P2					P37		B7								P37
C5	P33 P27	P33						P27		P33		P33		P33					P27				P27				P27					P33												
C6	P30			P30								P30																																
C7	P12											P12																									C7							
C8	P0 P34	P34	P0					P0				P0 P34			P0											P0		P34		P34								C8						
D6	P6 P31		P31	P6			P6					P31														P6					P6											D6		
F7	P25 P32	P25	P32				P32		P25			P32	P32																P32												F7			
F8	P36 P38					P36		P38				P36 P38	P36			P38												P36			P36									F8				
G8	P4 P24	P4	P24			P24		P4								P4														P4									G8					

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

** The red rectangle indicates that the associated signal functions do not involve any GPIO signals.

To help understand Figure 1, consider the shaded portion of the table, which pertains to the signal-function selection for pin D6. If an alphanumeric string appears in a signal-function column for a given pin, then that pin can support the signal function. Using this logic, D6 can be configured to support one signal function from the following set of signal functions:

1. GPIO_**P6** (a general-purpose user-defined I/O)
2. GPIO_**P31** (a general-purpose user-defined I/O)
3. PUART_TX (a peripheral UART TX output multiplexed to GPIO_**P31**)
4. PUART_RTS (a peripheral UART request-to-send output multiplexed to GPIO_**P6**)
5. SPI1_CS (a SPI interface chip-select input multiplexed to GPIO_**P6**)
6. A/D converter input (an A/D converter input multiplexed to GPIO_**P31**)
7. 60Hz_main input (a zero-crossing detector input multiplexed to GPIO_**P6**)
8. Quadrature decoder signal (the output of a quadrature decoder multiplexed to GPIO_**P6** as an input)
9. External LPO input (LPO_IN) (an external LPO input connected to pin **D6**). No GPIOs are used in this case.
10. 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 to pin D6 (using GPIO_P31).

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 5 “Interface Programming Information and Examples”.

3 GPIO Information

3.1 GPIO_Pxx

The Cypress WICED Studio API provides configuration support for up to 40 multiplexed GPIOs (represented as GPIO_P0 through GPIO_P39 or more simply as P0 through P39). The CYW2070x uses 23 of these multiplexed GPIOs, which are represented as GPIO_Pxx and sometimes referred to as the LHL GPIOs.

Table 2 shows CYW2070x interfaces and signal functions that, if used in a design, will restrict the set of digital I/O pins available for assigning GPIOs.

Table 2. Key Interface (or Signal Function) Pin Assignments

If This Interface or Signal Function Is Used	***Then Do Not Use These Pins for GPIO_Pxx Assignments***
I2S/PCM	A8, B7, C7, and C8
BT_CLK_REQ	G8
BT_DEV_WAKE[1]	F8
BT_HOST_WAKE	F7
LPO_IN	D6

[1]This signal function can be used to wake the CYW2070x. For those who do not want to use this signal function (and the digital I/O pin associated with it) to wake the CYW2070x, see Appendix A.3 “Wake Options”.

Table 3 provides an interface summary of the 23 multiplexed GPIOs used by the CYW2070x. The Description column provides the potential signal functions of each GPIO and the associated I/O type of each signal function.

Table 3. CYW2070x Multiplexed GPIO_Pxx Interface Summary

		Description	
CYW2070x Pin	***GPIO***	***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
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)

[1]For a device that is not running an application, the default I/O type for each GPIO is input (I) at boot up. For a device that is running an application, unused GPIOs will be high-Z (I/O disabled) after application initialization.

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

CYW2070x Pin	GPIO	Description	
		I/O Type[1]	Signal Function Options
		O	Optical control output (QOC0)
		O	Triac control 1
		O	PWM output (PWM0)
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
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)

CYW2070x Pin	GPIO	Description	
		I/O Type[1]	Signal Function Options
		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
		I	A/D converter input (A/D input 23)
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)

CYW2070x Pin	GPIO	Description	
		I/O Type[1]	Signal Function Options
	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)
		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)

Note: Table 3 provides GPIO information for the full capabilities of the CYW2070x. Some CYW2070x-based modules may not support all GPIOs. Check the module schematic and other documentation to determine which GPIOs are available.

3.2 GPIO_Pxx Capabilities

The GPIO_Pxx have the following capabilities:

- Each can be programmed to serve one of the signal functions associated with it (see Figure 1 or Table 3).
- All can be input and output disabled (high-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 Sleep and Deep Sleep modes.
- All GPIOs can be configured for edge (rising/falling/both) or level interrupts.

Note: 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 Sleep and Deep Sleep modes. For more information on the Sleep and Deep Sleep modes, see Appendix A "Low-Power Options".

- All GPIOs, excluding P26–P29, can source or sink up to 8 mA at 3.3V and 4 mA at 1.8V.
- GPIOs P26–P29 can source or sink 16 mA at 3.3V and 8 mA at 1.8V.

Note: See Section 5.1 "GPIO Programming Example", for GPIO programming information, including the enabling and disabling of inputs and outputs, edge triggering, and interrupt configuration.

3.3 GPIO_Pxx Ports and Pad Banks

As previously stated, the Cypress WICED Studio API provides configuration support for up to 40 multiplexed GPIOs (represented as GPIO_P0 through GPIO_P39 or more simply as P0 through P39). A fully featured CYW2070x device uses 23 of these multiplexed GPIOs.

The 40 GPIOs are arranged in three ports. Some of the GPIOs are also a part of one of two pad banks.

Table 4 shows the GPIO ranges for the GPIO ports and pad banks.

Table 4. GPIO Ports and Pad Banks

GPIO Port	*GPIO Range*
1	P0 to P15
2	P16 to P31
3	P32 to P39

GPIO Pad Bank	*GPIO Range*
1 (Lower)	P0 to P7
--	--
2 (Upper)	P24 to P39

See Section 4 "Interface Signal Function Selection Restrictions and Considerations" for specific information on using and configuring the GPIO_Pxx interfaces.

See Section 5 "Interface Programming Information and Examples" for information on programming the GPIO_Pxx interfaces and some GPIO_Pxx interface programming examples.

4 Interface Signal Function Selection Restrictions and Considerations

This section provides some signal-function selection restrictions and/or general information associated with the following CYW2070x interfaces:

- I^{2}S and PCM)
- SPI1
- SPI2

- HCI UART
- Peripheral UART
- I^2C (see Broadcom Serial Control)
- NVRAM

4.1 I^2S and PCM

The CYW2070x contains I^2S and PCM circuit blocks that share a common signal-routing interface.

Table 5 shows the shared I^2S and PCM interface in pin order. It also shows which GPIO_Pxx can't be used if the I^2S or PCM circuit blocks are used in a board design.

Table 5. Shared I^2S and PCM Interface and GPIO_Pxx Usage Restrictions

CYW2070x Pin	***Signal***	***When Interface Used, Do Not Use the Following GPIO_Pxx***
A8	I2S_DO or PCM_OUT	GPIO_P0, GPIO_P2, GPIO_P3, GPIO_P12, GPIO_P28, GPIO_P29, GPIO_P34, GPIO_P35, and GPIO_P37
B7	I2S_CLK or PCM_CLK	
C7	I2S_DI or PCM_IN	
C8	I2S_WS or PCM_SYNC	

4.2 Serial Peripheral Interfaces

The CYW2070x 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 the CYW2070x (see Section 3.1 "GPIO_Pxx").

As shown in Table 1, the SPI2 interface has a fixed signal routing to CYW2070x pins; therefore, it does not require any device GPIO support.

The WICED Studio API supports configuring and controlling both SPI interfaces. It provides services for:

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

For more information on the SPI1 interface, see SPI1 below.

For more information on the SPI2 interface, see SPI2 below.

4.2.1 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.

SP11 Master and SPI1 Slave provide SPI1 bus-configuration options.

4.2.1.1 SP11 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.

Note: All SPI bus signals must be on the same pad bank (see Section 3.3 “GPIO_Pxx Ports and Pad Banks”)

A GPIO_Pxx signal will need to be assigned to source each required slave chip-select output.

The application controls CS line assertion and deassertion 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 5 “Interface Programming Information and Examples” for more information.

Table 6 shows the CYW2070x SPI1 master bus-configuration options.

Table 6. CYW2070x SPI1 Master Bus-Configuration Options

	SPI1_CLK		***SPI1_MOSI***		***SPI1_MISO***	
Option	***Pin***	***GPIO_Pxx***	***Pin***	***GPIO_Pxx***	***Pin***	***GPIO_Pxx***
1	G8	P24	C5	P27	F7	P25
2	G8	P24	F8	P38	F7	P25
3	F8	P36	C5	P27	F7	P25

Note: The RX signal of a peripheral UART (PUART), if included in a system design, must be on the same pad bank as the SPI1 bus signals. See Section 4.4 “Peripheral UART” to understand how this restriction affects the bus configuration options of both interfaces.

For a SPI1 master programming example, see Section 5.2 “SPI1 Master Programming Example”.

4.2.2 SPI1 Slave

Table 7 shows the CYW2070x SPI1 slave bus-configuration options.

Table 7. CYW2070x SPI1 Slave Bus-Configuration Options

	SPI1_CLK		***SPI1_MOSI***		***SPI1_MISO***	
Option	***Pin***	***GPIO_Pxx***	***Pin***	***GPIO_Pxx***	***Pin***	***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

Option	SPI1_CLK		SPI1_MOSI		SPI1_MISO	
	Pin	GPIO_Pxx	Pin	GPIO_Pxx	Pin	GPIO_Pxx
8	F8	P36	C5	P27	B7	P37
9	F8	P36	C5	P33	F7	P25
10	F8	P36	C5	P33	B7	P37

Note: The bus signals of a peripheral UART (PUART), if included in a system design, must be on the same pad bank as the SPI1 bus signals. See Section 4.4 "Peripheral UART" to understand how this restriction affects the bus configuration options of both interfaces.

For a SPI1 slave programming example, see Section 5.3 "SPI1 Slave Programming Example".

Note: Table 6 and Table 7 apply to the CYW2070x. They may not apply to future modules that are based on the CYW2070x.

4.2.3 SPI2

Note: The SPI2 interface is provided as a CYW20706 connection to nonvolatile memory for accessing configuration, patches, application code, and application data. It is not applicable to the CYW20707 as the MCU will load these items into the device's SRAM over the HCI interface.

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 1.

The firmware controls CS line assertion and deassertion 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_hal_sflash.h or wiced_hal_seeprom.h, respectively.

4.3 HCI UART

The CYW2070x supports an HCI 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 debugging.
- It is a fixed (nonmultiplexed) interface (see Table 1).
- In the CYW20706, it is available only for application-defined HCI commands and events; it is not for Bluetooth standard HCI commands. In the CYW20707, 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:

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

- Debug messages can be routed to it. 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.

4.4 Peripheral UART

The CYW2070x 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 maximum RX and TX hardware buffer size is 256 bytes.
- Table 8 shows the CYW2070x PUART bus-configuration options when a system design also uses the SPI1 interface provided by the CYW2070x. In such a system, PUART_RX must be on the same pad bank as the SPI1 bus signals. Since only the upper pad bank supports a full SPI1 bus, all SPI1 bus signals and PUART_RX must be on the upper pad bank (P24 through P39).

If the CYW2070x SPI1 interface is not used, then several more PUART bus-configuration options (in addition to those shown in Table 8) are available. See Figure 1 to determine the other available PUART bus-configuration options.

Table 8. CYW2070x PUART Bus-Configuration Options When PUART_RX Is on the Upper Pad Bank

	PUART_RX		***PUART_TX***		***PUART_RTS***		***PUART_CTS***	
Option	***Pin***	***GPIO_Pxx***	***Pin***	***GPIO_Pxx***	***Pin***	***GPIO_Pxx***	***Pin***	***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

Option	PUART_RX		PUART_TX		PUART_RTS		PUART_CTS	
	Pin	GPIO_Pxx	Pin	GPIO_Pxx	Pin	GPIO_Pxx	Pin	GPIO_Pxx
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
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

The PUART bus-configuration options presented in Table 8 will be further reduced by the selected SPI1 bus-configuration option.

For example, SPI1 master bus-configuration option 2 (from Table 6) requires digital I/O pins G8, F8, and F7. Therefore, all PUART bus-configuration options (in Table 8) 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.

4.5 Broadcom Serial Control

The CYW2070x supports a Broadcom Serial Control (BSC) interface.

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

The following information applies to the BSC interface:

- Its use is optional.
- If used, then GPIO_P37 (pin B7) must be programmed as the BSC clock signal (BSC_CLK) and GPIO_P35 (pin A8) must be programmed as the BSC data signal (BSC_SDA).
- 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 the transaction.

- 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, the CYW2070x 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.

For a BSC programming example, see Section 5.4 “BSC Programming Example”.

4.6 NVRAM

The 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.

The CYW20707 does not have or need NVRAM, the MCU should store all non-volatile data.

5 Interface Programming Information and Examples

Note: All code and code references in this section pertain to WICED Studio.

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

- Disable the inputs and outputs of all unused GPIO_Pxx that are internally bonded (in the CYW2070x) or externally bonded (on a board) to GPIO_Pxx used in a design. Refer to wiced_hal_gpio.h for the pertinent API functions.
- When the CYW2070x is in the Sleep or Deep Sleep modes, no data can be received from any of its interfaces (SPI, PUART, etc.). To wake the CYW2070x, use a GPIO_Pxx configured as an interrupt. Refer to wiced_hal_gpio.h and gpiodriver.h for the pertinent API functions and contstants. Se Appendix A “Low-Power Options” for more information about the low-power options.
- Use the `wiced_hal_gpio_pin_to_port_pin` function to convert GPIO_Pxx to an internal port and pin number.
- To access GPIO driver services invoke the `wiced_hal_gpio_init` function during application initialization. This is independent of other functions and must be one of the first to be initialized.
- For a sample application that pertains to GPIO API function usage refer to hal_gpio_app.c.
- 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

5.1 GPIO Programming Example

The example in this section demonstrates some aspects of GPIO programming. Refer to wiced_hal_gpio.h for the complete set of API functions that support GPIO programming.

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.

```
#include "wiced_hal_gpio.h"

void test_gpio_driver( void )
{
    /* Initializes the GPIO driver */
    wiced_hal_gpio_init( );
    test_gpio_output( );
}

/* Sample function configures GPIO as output.
 * GPIO pins can be monitored to observe the output pattern */
void test_gpio_output( void )
{
    int32_t gpio_num , output_val;


    WICED_BT_TRACE("gpio_set_output\n\r");

    for( gpio_num = 0; gpio_num < GPIO_NUM_PINS; gpio_num++)
    {
        /* Configure GPIO PIN# as output and outvalue as high */
        wiced_hal_gpio_configure_pin( gpio_num, GPIO_OUTPUT_ENABLE,
            GPIO_PIN_OUTPUT_HIGH);
    }

    /* Send a pattern on GPIO output pins */
    while(1)
    {
        for( gpio_num = GPIO_NUM_PINS-1 ; gpio_num >= 0; gpio_num-- )
        {
            /* Set the output value of output pin P# to HIGH */
            wiced_hal_gpio_set_pin_output( gpio_num, GPIO_PIN_OUTPUT_HIGH);

            /* Get the output value of output pin */
            output_val = wiced_hal_gpio_get_pin_output( gpio_num);
            WICED_BT_TRACE("\n\rOutput val on %d: %d\n\r", gpio_num, output_val);
        }

        for( gpio_num = 0; gpio_num < GPIO_NUM_PINS; gpio_num++ )
        {
            /* Set the output value of output pin P# to LOW */
```

```
            wiced_hal_gpio_set_pin_output( gpio_num, GPIO_PIN_OUTPUT_LOW);

            /* Get the output value of output pin */
            output_val = wiced_hal_gpio_get_pin_output( gpio_num);
            WICED_BT_TRACE("Output val on %d: %d\n\r", gpio_num, output_val);
        }
    }
}

/* Sample function configures GPIO as input.
 * GPIO pins can be polled to observe the input pattern */
void test_gpio_input( void )
{
    int32_t gpio_num, input_val;

    for( gpio_num = 0; gpio_num < GPIO_NUM_PINS; gpio_num++)
    {
        /* if pin is used for other fucntions like UART debugging it should not be
         * used as GPIO I/O
         * EX: If P31  is configured to use as PUART TXD */
        if(gpio_num==31)
        {
            continue;
        }

        /* Configure GPIO PIN# as input and initial outvalue as high */
        wiced_hal_gpio_configure_pin( gpio_num, GPIO_INPUT_ENABLE,
            GPIO_PIN_OUTPUT_HIGH );
    }

    /* Get the input on GPIOs */
    while(1)
    {
        for( gpio_num = GPIO_NUM_PINS-1 ; gpio_num > 0; gpio_num-- )
        {
            /* Get the status of the input pin P# */
            input_val = wiced_hal_gpio_get_pin_input_status( gpio_num );
            WICED_BT_TRACE("Input val on %d: %d\n\r", gpio_num, input_val);
        }
    }
}

void gpio_interrrupt_handler(void *data, UINT8 port_pin)
{
    WICED_BT_TRACE("gpio_interrupt_handler %d\n\r", port_pin);

     /* Get the status of interrupt on P20 */
    if ( wiced_hal_gpio_get_pin_interrupt_status( (BYTE)data ) )
    {
        WICED_BT_TRACE( "Interrupt occured\n\r" );

        /* Clear the gpio interrupt */
        wiced_hal_gpio_clear_pin_interrupt_status( (BYTE)data );
    }

    /*  Set all GPIOs to be input disabled*/
    wiced_hal_gpio_disable_all_inputs();
}
```

```
/* Sample function configures GPIO as input. Enable interrupt.
 * Register a call back function to handle on interrupt*/
void test_gpio_input_interrupt( void )
{
    uint32_t gpio_num = 20;
    uint16_t masks[GPIO_NUM_PORTS]= {0, 1 << (20%16), 0};
    uint16_t pin_config;

/* Register the GPIO interrupt handler */
wiced_hal_gpio_register_pin_for_interrupt( masks, gpio_interrrupt_handler,
    &gpio_num );

    /* Configure GPIO PIN# as input, pull up and interrupt on falling edge */
    wiced_hal_gpio_configure_pin( gpio_num, (GPIO_INPUT_ENABLE | GPIO_PULL_UP |
        GPIO_EN_INT_FALLING_EDGE), GPIO_PIN_OUTPUT_HIGH);

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

5.2 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 */
    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 );

}
```

5.3 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 */
    // 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 );

}
```

5.4 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 );
}
```

5.5 PUART Programming Example

The following example shows how to initialize the peripheral UART. The code in this example can be found in hal_puart_app.c in WICED Studio.

```
#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. Initialises 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 avaliable for use on your
    // specific hardware platform.
    // Please see the User Documentation to reference the valid pins.
    wiced_hal_puart_select_uart_pads( 2, 31, 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 );

#if PUART_INTERRUPT_DISABLE
    /* Enable to read 5 bytes sequentially and hold it in destination buffer */
    //wiced_hal_puart_synchronous_read( read_5_bytes, 5);

    /* Enable to write 5 bytes sequentially held in source buffer */
    //wiced_hal_puart_synchronous_write( read_5_bytes, 5);
#endif
}
```

5.6 NVRAM Programming Example

The following code sample provides an example of writing a Bluetooth device address to NVRAM and reading it back. The code in this example can be found in hal_nvram_app.c in WICED Studio. This is applicable only to the CYW20706, see Section 4.6 “NVRAM”.

```
#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);
}
```

Appendices

A Low-Power Options

The WICED Studio API provides two power-save options for the CYW2070x, each initiated by a function call. The power-save options (or low-power options) are:

- Deep Sleep
- Sleep

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 Power-Save Option 1 — Deep-Sleep Mode

Use `wiced_power_save_start` to allow the CYW2070x to enter the Deep-Sleep mode. The CYW2070x power consumption is minimized when applications allow for Deep-Sleep operation.

A transition to Deep Sleep mode causes the currently running application to exit, so care must be taken to save all state information that will be needed by the application upon waking up. When waking from this power-save mode, the application will be restarted.

Use the following two API functions to save and restore 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)

The `wake_source` parameter in `wiced_power_save_start` indicates the power-save wake-up source. The three available wake-up sources are:

- `WICED_WAKE_SOURCE_GPIO`

  Setting this wake-up source causes a wake-up from power-save if any of the multiplexed GPIOs, also referred to as the LHL GPIOs (or GPIO_Pxx), transition to an active state.
- `WICED_WAKE_SOURCE_TIMEOUT`

  Setting this wake-up source causes a wake-up from power-save based on a timeout.
- `WICED_WAKE_SOURCE_ALL` (this is `WICED_WAKE_SOURCE_GPIO` | `WICED_WAKE_SOURCE_TIMEOUT`)

In this power-save option, an application can register callback functions using the following functions:

- `wiced_power_save_register_approve_cback`

  The callback registered with this function lets an application approve or disapprove of the CYW2070x transitioning to the power-save mode.
- `wiced_power_save_register_enter_cback`

  The callback registered with this function is called just before the CYW2070x transitions to the power-save mode.
- `wiced_power_save_register_abort_cback`

  The callback registered with this function is invoked when a transition to the power-save mode is aborted for unknown reasons. An application can react to a power-save abort by once again invoking `wiced_power_save_start` or the second power-save option, `wiced_sleep_config`.

A.2 Power-Save Option 2 — Sleep Mode

Use `wiced_sleep_config` to enter (and exit) Sleep mode if Deep-Sleep mode is not required or feasible. In Sleep mode, power savings are achieved by suspending the application instead of exiting the application.

Upon waking from Sleep mode, the application resumes.

The application can use `wiced_sleep_config` to:

- Enable Sleep mode
- Disable Sleep mode
- Activate the appropriate pin (the BT_DEV_WAKE pin (F8), for example) so that an attached host can wake the CYW2070x. See Appendix A.3 "Wake Options" for more information on waking the CYW2070x.
- Activate the BT_HOST_WAKE pin so that the CYW2070x can wake an attached host.

A.3 Wake Options

The CYW2070x can be awakened using one or more of the following three approaches:

1. Assign pin F8 as the BT_DEV_WAKE signal.

   The drawback to this approach: it dedicates what might be a valuable digital I/O pin to waking the device.
2. Program one or more of the GPIO_Pxx as wake-up sources.
3. Program the CYW2070x to wake based on a timeout.

For approaches 2 and 3, see the wake-up source options in Appendix A.1 "Power-Save Option 1 — Deep-Sleep Mode".

B CYW92070xV3_EVAL Board

B.1 CYW2070x Interfaces Used

Besides using the fixed interfaces (see Section 2.1 ”Fixed Interfaces”), the CYW92070xV3_EVAL board uses some of the selectable interfaces (see Section 2.2 “Selectable Interfaces”) supported by the CYW2070x.

The CYW92070xV3_EVAL board uses the following selectable interfaces:

- SPI1 master
- PUART
- I^2S
- A GPIO_Pxx to control an external switch
- A GPIO_Pxx to control an external LED
- A GPIO_Pxx to monitor an SW6 button depression

The CYW2070x has 12 digital I/O pins that can be used to support the selectable interfaces listed above.

Table 9 shows which of the selectable CYW2070x interfaces are used on the CYW92070xV3_EVAL board.

Table 9. CYW92070xV3_EVAL Board Interface Summary

CYW2070x Pin	***Schematic Signal Name***	***Bonded GPIOs***	***Description***	
			I/O Type	***Signal Function and Notes***
A8	I2S_DO/SCL/P3	–	O	I2S_DO/PCM_OUT. To use, set SW5-5 to the off position, select the I^2S function, and I/O disable P3, P29, and P35.
		P3	I/O	This can be used as PUART_CTS by setting SW5-5 to the on position. It can be tested and monitored at J22-8 if the I^2S/PCM function is deselected, and P29 and P35 are I/O disabled.
		P29	I/O	No specific function defined. It can be tested and monitored at J22-8 if the I^2S/PCM function is deselected, and P3 and P35 are I/O disabled.
		P35	I/O	No specific function defined. It can be tested and monitored at J22-8 if the I^2S/PCM function is deselected, and P3 and P29 are I/O disabled.
B5	P15	P15	O	GPIO switch-control output.
B6	P26_PWM0	P11	I/O	No specific function defined. It can be tested and monitored at J22-4 if P26 is I/O disabled.
		P26	O	LED D10 on/off control that can sink 16 mA and be modulated using the PWM0 circuit on the CYW2070x. To use, I/O disable P11.
B7	I2S_CLK/P2	–	O	I2S_CLK. To use, set SW5-1 to the off position, select the I^2S function, and I/O disable P2, P28, and P37.
		P2	I/O	PUART_RX. To use, set SW5-1 to the on position, SW5-2 to the off position, and I/O disable P28 and P37. It can be tested and monitored at J22-5 if the I^2S/PCM function is deselected, and P28 and P37 are I/O disabled.
		P28	I/O	No specific function defined. It can be tested and monitored at J22-5 if the I^2S/PCM function is deselected, and P2 and P37 are I/O disabled.

CYW2070x Pin	Schematic Signal Name	Bonded GPIOs	Description: I/O Type	Description: Signal Function and Notes
		P37	I/O	No specific function defined. It can be tested and monitored at J22-5 if the I²S/PCM function is deselected, and P2 and P28 are I/O disabled.
C5	P27_SP1_MOSI/P33	P27	O (master) I (slave)	SPI1_MOSI (master or slave) . To use, set SW5-2 to the off position and I/O disable P33.
		P33	I	PUART_RX. To use, set SW5-1 to the off position, SW5-2 to the on position, and I/O disable P27.
C6	P30	P30	I	SW6 button-depression input. This can also be configured as PUART_RTS by setting SW5-6 to the on position.
C7	I2S_DI/SDA	–	I	I2S_DI. To use, select the I²S function, and I/O disable P12.
		P12	I/O	No specific function defined. It can be tested and monitored at J22-7 if the I²S/PCM function is deselected.
C8	I2S_WS/P0	–	I/O	I2S_WS. To use, set SW5-3 to the off position, select the I²S function, and I/O disable P0 and P34.
		P0	I/O	PUART_TX. To use, set SW5-3 to the on position, SW5-4 to the off position, and I/O disable P34. It can be tested and monitored at J22-6 if the I²S/PCM function is deselected and P34 is I/O disabled.
		P34	I/O	No specific function defined. It can be tested and monitored at J22-6 if the I²S/PCM function is deselected and P0 is I/O disabled.
D6	P6_PWM2/P31	P6	I/O	LED D9 on/off control that can be modulated using the PWM2 circuit on the CYW2070x. To use, I/O disable P31.
		P31	O	PUART_TX. To use, set SW5-3 to the off position, set SW5-4 to the on position, and I/O disable P6.
F7	P25_SP1_MISO	P25	I (master) O (slave)	SPI1_MISO (master or slave). To use, I/O disable P32.

CYW2070x Pin	Schematic Signal Name	Bonded GPIOs	Description	
			I/O Type	Signal Function and Notes
		P32	I/O	No specific function defined. It can be tested and monitored at J19-4 if P25 is I/O disabled.
F8	P36_SP1_CS	P36	O (master) I (slave)	GPIO_P36 programmed as SPI1_CS. To use, I/O disable P38.
		P38	I/O	No specific function defined. It can be tested and monitored at J19-6 if P36 is I/O disabled.
G8	P24_SP1_SCK	P4	I/O	No specific function defined. It can be tested and monitored at J19-3 if P24 is I/O disabled.
		P24	I/O	SPI1_CLK (master or slave). To use, I/O disable P4.

Document History

Document Title: AN216761 - CYW2070x Hardware Interface Selection and Programming

Document Number: 002-16761

Revision	Submission Date	Description of Change
*B	01/13/2017	Fixed page numbers, updated references, and applied template updates
*A	11/08/2016	Product number changes and fixes
**	10/17/2016	New Spec.

References

[1] 002-14790 - CYW20706 Datasheet

[2] AN218191 - WICED Quick Start Guide for BT CYW2070x

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.

Cypress 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
Lighting & Power Control	cypress.com/powerpsoc
Memory	cypress.com/memory
PSoC	cypress.com/psoc
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless/RF	cypress.com/wireless

PSoC® Solutions

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

Cypress Developer Community

Forums | Projects | Videos | Blogs | Training | Components

WICED IoT

Uniting CDC and WICED Solutions

Technical Support

cypress.com/support

PSoC is a registered trademark and WICED and PSoC Creator are trademarks of Cypress Semiconductor Corporation. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com