Chipsmall
深圳市津利帝科技有限公司
Geelead Technology Limited
采购部

CYRF69313

Programmable Radio-on-Chip LPstar

Features

- Radio System-on-Chip, with built-in 8-bit MCU in a single device.
- Operates in the unlicensed worldwide Industrial, Scientific, and Medical (ISM) band (2.400 GHz to 2.483 GHz).
- On Air compatible with second generation radio WirelessUSB™ LP and PRoC LP.
- Pin-to-pin compatible with PRoC LP except the pin 31 and pin 37.

Intelligent

- M8C based 8-bit CPU, optimized for human interface devices (HID) applications
- 256 bytes of SRAM
- 8 Kbytes of flash memory with EEPROM emulation
- In-System reprogrammable through D+/D– pins
- CPU speed up to 12 MHz
- 16-bit free running timer
- Low power wakeup timer
- 12-bit programmable interval timer with interrupts
- Watchdog timer

Low Power

- 21 mA operating current (Transmit at –5 dBm)
- Sleep current less than 1 µA
- Operating voltage from 4.0 V to 5.25 V DC
- Fast startup and fast channel changes
- Supports coin-cell operated applications

Reliable and Robust

- Receive sensitivity typical –90 dBm
- AutoRate™ – dynamic data rate reception
  - Enables data reception for any of the supported bit rates automatically.
  - DSSS (250 Kbps), GFSK (1 Mbps)
- Operating temperature from 0 °C to 70 °C
- Closed-loop frequency synthesis for minimal frequency drift

Simple Development

- Auto transaction sequencer (ATS): MCU can stay sleeping longer to save power
- Framing, length, CRC16, and Auto ACK
- Separate 16 byte transmit and receive FIFOs
- Receive signal strength indication (RSSI)
- Built-in serial peripheral interface (SPI) control while in sleep mode
- Advanced development tools based on Cypress's PSoC® Tools
- Flexible I/O
- 2 mA source current on all GPIO pins. Configurable 8 mA or 50 mA/pin current sink on designated pins
- Each GPIO pin supports high impedance inputs, configurable pull-up, open-drain output, CMOS/TTL inputs, and CMOS output
- Maskable interrupts on all I/O pins

BOM Savings

- Low external component count
- Small footprint 40-pin QFN (6 mm × 6 mm)
- GPIOs that require no external components
- Operates off a single crystal
- Integrated 3.3 V regulator
- Integrated pull-up on D–

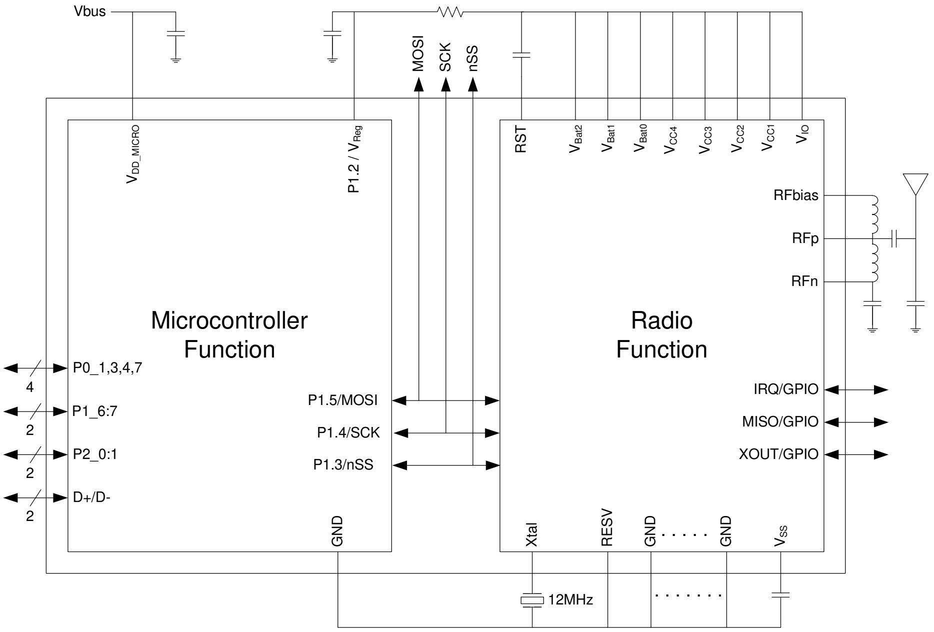
USB Specification Compliance

- Conforms to USB specification version 2.0
- Conforms to USB HID specification version 1.1
- Supports one low speed USB device address
- Supports one control endpoint and two data end points
- Integrated USB transceiver

Applications

- Wireless keyboards and mice
- Presentation tools
- Wireless gamepads
- Remote controls
- Toys
- Fitness

Logic Block Diagram



Contents

Functional Description **5**
Functional Overview **5**
2.4 GHz Radio Function 5
USB Microcontroller Function 5
Backward Compatibility 5
Pinouts **6**
Pin Configuration **6**
PRoC LPstar Functional Overview **7**
Functional Block Overview **8**
2.4 GHz Radio 8
Frequency Synthesizer 8
Baseband and Framer 8
Packet Buffers 9
Auto Transaction Sequencer (ATS) 9
Interrupts 9
Clocks 9
GPIO Interface 9
Power-on Reset 10
Power Management 10
Timers 10
USB Interface 10
Low Noise Amplifier (LNA) and Received Signal Strength Indication (RSSI) 10
SPI Interface **10**
Three-Wire SPI Interface 10
Four-Wire SPI Interface 11
SPI Communication and Transactions 11
SPI I/O Voltage References 11
SPI Connects to External Devices 11
CPU Architecture **12**
CPU Registers **13**
Flags Register 13
Accumulator Register 13
Index Register 13
Stack Pointer Register 14
CPU Program Counter High Register 14
CPU Program Counter Low Register 14
Addressing Modes **15**
Source Immediate 15
Source Direct 15
Source Indexed 15
Destination Direct 15
Destination Indexed 16
Destination Direct Source Immediate 16
Destination Indexed Source Immediate 16
Destination Direct Source Direct 16
Source Indirect Post Increment 17
Destination Indirect Post Increment 17
Instruction Set Summary **18**
Memory Organization **19**
Flash Program Memory Organization 19
Data Memory Organization 20
Flash 20
SROM 20
SROM Function Descriptions 21
SROM Table Read Description **24**
Clocking **25**
Clock Architecture Description 26
CPU Clock During Sleep Mode 32
Reset **32**
Power-on Reset **34**
Watchdog Timer Reset 34
Sleep Mode **34**
Sleep Sequence 34
Wakeup Sequence 35
Low Power in Sleep Mode 35
Power-on Reset Control **37**
POR Compare State 37
ECO Trim Register 37
General-Purpose I/O Ports **38**
Port Data Registers 38
GPIO Port Configuration 39
GPIO Configurations for Low Power Mode 43
Serial Peripheral Interface (SPI) **44**
SPI Data Register 45
SPI Configure Register 45
Timer Registers **47**
Registers 47
Interrupt Controller **50**
Architectural Description 50
Interrupt Processing 51
Interrupt Latency 51
Interrupt Registers 51
USB Transceiver **56**
USB Transceiver Configuration 56
USB Serial Interface Engine (SIE) **56**
USB Device **57**
Endpoint 0 Mode 58
Endpoint Data Buffers 60
USB Mode Tables **61**
Mode Column 61
Encoding Column 61
SETUP, IN, and OUT Columns 61
Details of Mode for Differing Traffic Conditions **62**
Register Summary **64**
Radio Function Register Descriptions **66**
Absolute Maximum Ratings **67**
DC Characteristics **67**
RF Characteristics **69**
AC Test Loads and Waveforms for Digital Pins **70**
AC Characteristics **71**
Switching Waveforms **72**
Ordering Information **76**
Ordering Code Definitions 76
Package Handling **77**
Package Diagrams **77**
Acronyms **79**
Document Conventions **79**
Units of Measure 79
Document History Page **80**

Sales, Solutions, and Legal Information 81
Worldwide Sales and Design Support 81
Products 81
PSoC® Solutions 81
Cypress Developer Community 81
Technical Support 81

Functional Description

PRoC LPstar devices are integrated radio and microcontroller functions in the same package to provide a dual role single-chip solution.

Communication between the microcontroller and the radio is via the SPI interface between both functions.

Functional Overview

The CYRF69313 is a complete Radio System-on-Chip device, providing a complete RF system solution with a single device and a few discrete components. The CYRF69313 is designed to implement low cost wireless systems operating in the worldwide 2.4 GHz Industrial, Scientific, and Medical (ISM) frequency band (2.400 GHz–2.4835 GHz).

2.4 GHz Radio Function

The SoC contains a 2.4 GHz, 1 Mbps GFSK radio transceiver, packet data buffering, packet framer, DSSS baseband controller, Received Signal Strength Indication (RSSI), and SPI interface for data transfer and device configuration.

The radio supports 98 discrete 1 MHz channels (regulations may limit the use of some of these channels in certain jurisdictions).

The baseband performs DSSS spreading/despreading, Start of Packet (SOP), End of Packet (EOP) detection, and CRC16 generation and checking. The baseband may also be configured to automatically transmit Acknowledge (ACK) handshake packets whenever a valid packet is received.

When in receive mode, with packet framing enabled, the device is always ready to receive data transmitted at any of the supported bit rates. This enables the implementation of mixed-rate systems in which different devices use different data rates. This also enables the implementation of dynamic data rate systems that use high data rates at shorter distances or in a low-moderate interference environment or both. It changes to lower data rates at longer distances or in high interference environments or both.

USB Microcontroller Function

The microcontroller function is based on the powerful CYRF69313 microcontroller. It is an 8-bit Flash programmable microcontroller with integrated low speed USB interface.

The microcontroller has up to 14 GPIO pins to support USB, PS/2 and other applications. Each GPIO port supports high impedance inputs, configurable pull-up, open drain output, CMOS/TTL inputs and CMOS output. Up to two pins support programmable drive strength of up to 50 mA. Additionally each I/O pin can be used to generate a GPIO interrupt to the microcontroller. Each GPIO port has its own GPIO interrupt vector with the exception of GPIO Port 0.

The microcontroller features an internal oscillator. With the presence of USB traffic, the internal oscillator can be set to precisely tune to USB timing requirements (24 MHz ± 1.5%).

The PRoC LPstar has up to 8 Kbytes of Flash for user’s firmware code and up to 256 bytes of RAM for stack space and user variables.

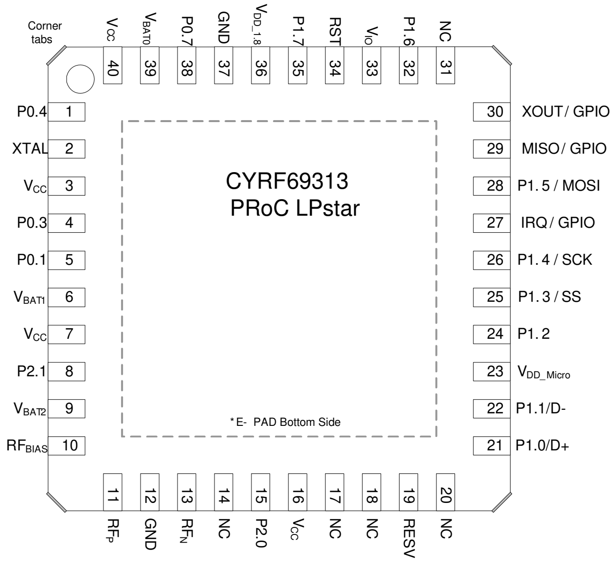
Backward Compatibility

The CYRF69313 IC is fully interoperable with the main modes of the second generation Cypress radio SoC namely the CYRF6936, CYRF69103 and CYRF69213.

CYRF69313 IC device may transmit data to or receive data from a second generation device, or both.

Pinouts

Figure 1. 40-pin QFN pinout



Pin Configuration

Pin	Name	Function
1	P0.4	Individually configured GPIO
2	Xtal_in	12 MHz Crystal. External clock in
3, 7, 16, 40	V_{CC}	Connected to pin 24 via 0.047 µF capacitor
4	P0.3	Individually configured GPIO
5	P0.1	Individually configured GPIO
6, 9, 39	V_{bat}	Connected to pin 24 via 0.047 µFshunt capacitor
8	P2.1	GPIO. Port 2 Bit 1
10	RF Bias	RF pin voltage reference
11	RF_p	Differential RF input to/from antenna
12	GND	Ground
13	RF_n	Differential RF to/from antenna
14, 17, 18, 20, 36	NC	
15	P2.0	GPIO. Port 2 Bit 0
19	RESV	Reserved. Must connect to GND
21	P1.0 / D+ / ISSP-SCLK	GPIO 1.0 / Low speed USB I/O / ISSP-SCLK
22	P1.1 / D– / ISSP-SDATA	GPIO 1.1 / Low speed USB I/O/ISSP-SDATA
23	V_{DD_micro}	4.0–5.5 for 12 MHz CPU/4.75–5.5 for 24 MHz CPU
24	P1.2	Must be configured as 3.3 V output. It must have a 1–2 µF output capacitor

Pin Configuration (continued)

Pin	Name	Function
25	P1.3 / nSS	Slave select SPI Pin
26	P1.4 / SCK	Serial Clock Pin from MCU function to radio function
27	IRQ	Interrupt output, configure high/low or GPIO
28	P1.5 / MOSI	Master Out Slave In
29	MISO	Master In Slave Out, from radio function. Can be configured as GPIO
30	XOUT	Bufferd CLK or GPIO
31	NC	Must be floating
32	P1.6	GPIO. Port 1 Bit 6
33	V_{IO}	I/O interface voltage. Connected to pin 24 via 0.047 µF
34	Reset	Radio Reset. Connected to V_{DD} via 0.47 µF capacitor or to microcontroller GPIO pin. Must have a RESET = HIGH event the very first time power is applied to the radio otherwise the state of the radio function control registers is unknown
35	P1.7	GPIO. Port 1 Bit 7
36	$V_{DD\_1.8}$	Regulated logic bypass. Connected via 0.47 µF to GND
37	GND	Must be connected to ground
38	P0.7	GPIO. Port 0 Bit 7
41	E-pad	Must be connected to GND
42	Corner Tabs	Do not connect corner tabs

PRoC LPstar Functional Overview

The SoC contains a 2.4 GHz 1 Mbps GFSK radio transceiver, packet data buffering, packet framer, DSSS baseband controller, Received Signal Strength Indication (RSSI), and SPI interface for data transfer and device configuration.

The radio supports 98 discrete 1 MHz channels (regulations may limit the use of some of these channels in certain jurisdictions). In DSSS modes the baseband performs DSSS spreading/despreading, while in GFSK Mode (1 Mb/s - GFSK) the baseband performs Start of Frame (SOF), End of Frame (EOF) detection and CRC16 generation and checking. The baseband may also be configured to automatically transmit Acknowledge (ACK) handshake packets whenever a valid packet is received.

When in receive mode, with packet framing enabled, the device is always ready to receive data transmitted at any of the supported bit rates. This enables the implementation of mixed-rate systems in which different devices use different data rates. This also enables the implementation of dynamic data rate systems that use high data rates at shorter distances or in a low-moderate interference environment or both. It changes to lower data rates at longer distances or in high interference environments or both.

The MCU function is an 8-bit Flash programmable microcontroller with integrated low speed USB interface. The instruction set has been optimized specifically for USB operations, although it can be used for a variety of other embedded applications.

The MCU function has up to eight Kbytes of Flash for user’s code and up to 256 bytes of RAM for stack space and user variables.

In addition, the MCU function includes a Watchdog timer, a vectored interrupt controller, a 16-bit Free-Running Timer, and 12-bit Programmable Interrupt Timer.

The MCU function supports in-system programming by using the D+ and D– pins as the serial programming mode interface. The programming protocol is not USB.

Functional Block Overview

All the blocks that make up the PRoC LPstar are presented here.

2.4 GHz Radio

The radio transceiver is a dual conversion low IF architecture optimized for power and range/robustness. The radio employs channel matched filters to achieve high performance in the presence of interference. An integrated Power Amplifier (PA) provides up to 0 dBm transmit power, with an output power control range of 30 dB in six steps. The supply current of the device is reduced as the RF output power is reduced.

Table 1. Internal PA Output Power Step Table

PA Setting	Typical Output Power (dBm)
6	0
5	–5
4	–10
3	–15
2	–20
1	–25
0	–30

Frequency Synthesizer

Before transmission or reception may commence, it is necessary for the frequency synthesizer to settle. The settling time varies depending on channel; 25 fast channels are provided with a maximum settling time of 100 µs.

The ‘fast channels’ (<100 µs settling time) are every third frequency, starting at 2400 MHz up to and including 2472 MHz (for example, 0,3,6,9…….69 and 72).

Baseband and Framer

The baseband and framer blocks provide the DSSS encoding and decoding, SOP generation and reception and CRC16 generation and checking, and EOP detection and length field.

Data Rates and Data Transmission Modes

The SoC supports two different data transmission modes:

- In GFSK mode, data is transmitted at 1 Mbps, without any DSSS.
- In DSSS mode eight bits (8DR, 32 chip) are encoded in each derived code symbol transmitted, resulting in effective 250 Kbps data rate.

32 chip Pseudo Noise (PN) codes are supported. The two data transmission modes apply to the data after the SOP. In particular the length, data, and CRC16 are all sent in the same mode. In general, DSSS reduce packet error rate in any environment.

Link Layer Modes

The CYRF69313 IC device supports the following data packet framing features:

SOP

Packets begin with a two-symbol SoP marker. If framing is disabled then an SOP event is inferred whenever two successive correlations are detected. The SOP_CODE_ADR code used for the SOP is different from that used for the “body” of the packet, and if desired may be a different length. SOP must be configured to be the same length on both sides of the link.

Length

Length field is the first eight bits after the SOP symbol, and is transmitted at the payload data rate. An EoP condition is inferred after reception of the number of bytes defined in the length field, plus two bytes for the CRC16.

CRC16

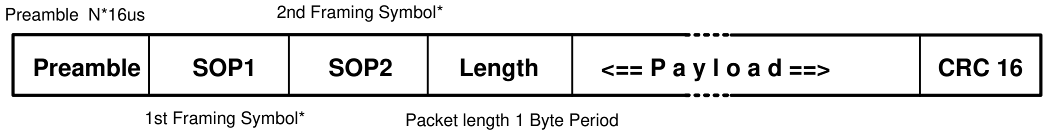
The device may be configured to append a 16-bit CRC16 to each packet. The CRC16 uses the USB CRC polynomial with the added programmability of the seed. If enabled, the receiver verifies the calculated CRC16 for the payload data against the received value in the CRC16 field. The starting value for the CRC16 calculation is configurable, and the CRC16 transmitted may be calculated using either the loaded seed value or a zero seed; the received data CRC16 is checked against both the configured and zero CRC16 seeds.

CRC16 detects the following errors:

- Any one bit in error
- Any two bits in error (irrespective of how far apart, which column, and so on)
- Any odd number of bits in error (irrespective of the location)
- An error burst as wide as the checksum itself

Figure 2 shows an example packet with SOP, CRC16 and lengths fields enabled.

Figure 2. Example Default Packet Format



*Note: 32 us

Packet Buffers

Packet data and configuration registers are accessed through the SPI interface. All configuration registers are directly addressed through the address field in the SPI packet. Configuration registers are provided to allow configuration of DSSS PN codes, data rate, operating mode, interrupt masks, interrupt status, and others.

Packet Buffers

All data transmission and reception uses the 16-byte packet buffers — one for transmission and one for reception.

The transmit buffer allows a complete packet of up to 16 bytes of payload data to be loaded in one burst SPI transaction. This is then transmitted with no further MCU intervention. Similarly, the receive buffer allows an entire packet of payload data up to 16 bytes to be received with no firmware intervention required until packet reception is complete.

The CYRF69313 IC supports packet length of up to 40 bytes; interrupts are provided to allow an MCU to use the transmit and receive buffers as FIFOs. When transmitting a packet longer than 16 bytes, the MCU can load 16 bytes initially, and add further bytes to the transmit buffer as transmission of data creates space in the buffer. Similarly, when receiving packets longer than 16 bytes, the MCU function must fetch received data from the FIFO periodically during packet reception to prevent it from overflowing.

Auto Transaction Sequencer (ATS)

The CYRF69313 IC provides automated support for transmission and reception of acknowledged data packets.

When transmitting a data packet, the device automatically starts the crystal and synthesizer, enters transmit mode, transmits the packet in the transmit buffer, and then automatically switches to receive mode and waits for a handshake packet — and then automatically reverts to sleep mode or idle mode when either an ACK packet is received, or a timeout period expires.

Similarly, when receiving in transaction mode, the device waits in receive mode for a valid packet to be received, then automatically transitions to transmit mode, transmits an ACK packet, and then switches back to receive mode to await the next packet. The contents of the packet buffers are not affected by the transmission or reception of ACK packets.

In each case, the entire packet transaction takes place without any need for MCU firmware action; to transmit data the MCU simply needs to load the data packet to be transmitted, set the length, and set the TX GO bit. Similarly, when receiving packets in transaction mode, firmware simply needs to retrieve the fully received packet in response to an interrupt request indicating reception of a packet.

Interrupts

The radio function provides an interrupt (IRQ) output, which is configurable to indicate the occurrence of various different events. The IRQ pin may be programmed to be either active high or active low, and be either a CMOS or open drain output. The IRQ pin can be multiplexed on the SPI if routed to an external pin.

The radio function features three sets of interrupts: transmit, receive, and system interrupts. These interrupts all share a single pin (IRQ), but can be independently enabled/disabled. In transmit mode, all receive interrupts are automatically disabled, and in receive mode all transmit interrupts are automatically disabled. However, the contents of the enable registers are preserved when switching between transmit and receive modes.

If more than one radio interrupt is enabled at any time, it is necessary to read the relevant status register to determine which event caused the IRQ pin to assert. Even when an interrupt source is disabled, the status of the condition that would otherwise cause an interrupt can be determined by reading the appropriate status register. It is therefore possible to use the devices without making use of the IRQ pin by polling the status register(s) to wait for an event, rather than using the IRQ pin.

The microcontroller function supports 23 maskable interrupts in the vectored interrupt controller. Interrupt sources include a USB bus reset, POR, a programmable interval timer, a 1.024-ms output from the Free Running Timer, three USB endpoints, two capture timers, five GPIO Ports, three GPIO pins, two SPI, a 16-bit free running timer wrap, an internal wakeup timer, and a bus active interrupt. The wakeup timer causes periodic interrupts when enabled. The USB endpoints interrupt after a USB transaction complete is on the bus. The capture timers interrupt whenever a new timer value is saved due to a selected GPIO edge event. A total of eight GPIO interrupts support both TTL or CMOS thresholds. For additional flexibility, on the edge sensitive GPIO pins, the interrupt polarity is programmable to be either rising or falling.

Clocks

The radio function has a 12 MHz crystal (30-ppm or better) directly connected between XTAL and GND without the need for external capacitors. A digital clock out function is provided, with selectable output frequencies of 0.75, 1.5, 3, 6, or 12 MHz. This output may be used to clock an external microcontroller (MCU) or ASIC. This output is enabled by default, but may be disabled.

Following are the requirements for the crystal to be directly connected to XTAL pin and GND:

- Nominal Frequency: 12 MHz
- Operating Mode: Fundamental Mode
- Resonance Mode: Parallel Resonant
- Frequency Stability: ±30 ppm
- Series Resistance: $\leq$60 ohms
- Load Capacitance: 10 pF
- Drive Level:100 μW

The MCU function features an internal oscillator. With the presence of USB traffic, the internal oscillator can be set to precisely tune to USB timing requirements (24 MHz ±1.5%). The clock generator provides the 12 MHz and 24 MHz clocks that remain internal to the microcontroller.

GPIO Interface

The MCU function features up to 20 general purpose I/O (GPIO) pins to support USB, PS/2, and other applications. The I/O pins are grouped into five ports (Port 0 to 4). The pins on Port 0 and Port 1 may each be configured individually while the pins on Ports 2, 3, and 4 may only be configured as a group. Each GPIO port supports high impedance inputs, configurable pull-up, open drain output, CMOS/TTL inputs, and CMOS output with up to five pins that support programmable drive strength of up to 50 mA sink current. GPIO Port 1 features four pins that interface at a voltage level of 3.3 volts. Additionally, each I/O pin can be used to generate a GPIO interrupt to the microcontroller. Each GPIO port has its own GPIO interrupt vector with the exception of GPIO Port 0. GPIO Port 0 has three dedicated pins that have independent interrupt vectors (P0.3–P0.4).

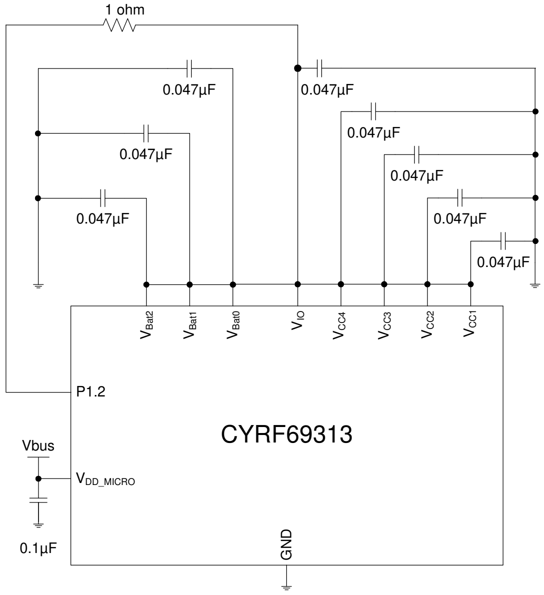
Power-on Reset

The power-on reset (POR) circuit detects logic when power is applied to the device, resets the logic to a known state, and begins executing instructions at Flash address 0x0000. When power falls below a programmable trip voltage, it generates reset or may be configured to generate interrupt. The Watchdog timer can be used to ensure the firmware never gets stalled in an infinite loop.

Power Management

The device draws its power supply from the USB V_{bus} line. The V_{bus} supplies power to the MCU function, which has an internal 3.3 V regulator. This 3.3 V is supplied to the radio function via P1.2 after proper filtering as shown in Figure 3.

Figure 3. Power Management From Internal Regulator



Timers

The free-running 16-bit timer provides two interrupt sources: the programmable interval timer with 1 µs resolution and the 1.024 ms outputs. The timer can be used to measure the duration of an event under firmware control by reading the timer at the start and at the end of an event, then calculating the difference between the two values.

USB Interface

The MCU function includes an integrated USB serial interface engine (SIE) that allows the chip to easily interface to a USB host. The hardware supports one USB device address with three endpoints.

Low Noise Amplifier (LNA) and Received Signal Strength Indication (RSSI)

The gain of the receiver may be controlled directly by clearing the AGC EN bit and writing to the low noise amplifier (LNA) bit of the RX_CFG_ADR register. When the LNA bit is cleared, the receiver gain is reduced by approximately 20 dB, allowing accurate reception of very strong received signals (for example when operating a receiver very close to the transmitter). An additional 20 dB of receiver attenuation can be added by setting the Attenuation (ATT) bit; this allows data reception to be limited to devices at very short ranges. Disabling AGC and enabling LNA is recommended unless receiving from a device using external PA.

The RSSI register returns the relative signal strength of the on-channel signal power.

When receiving, the device may be configured to automatically measure and store the relative strength of the signal being received as a 5-bit value. When enabled, an RSSI reading is taken and may be read through the SPI interface. An RSSI reading is taken automatically when the start of a packet is detected. In addition, a new RSSI reading is taken every time the previous reading is read from the RSSI register, allowing the background RF energy level on any channel to be easily measured when RSSI is read when no signal is being received. A new reading can occur as fast as once every 12 µs.

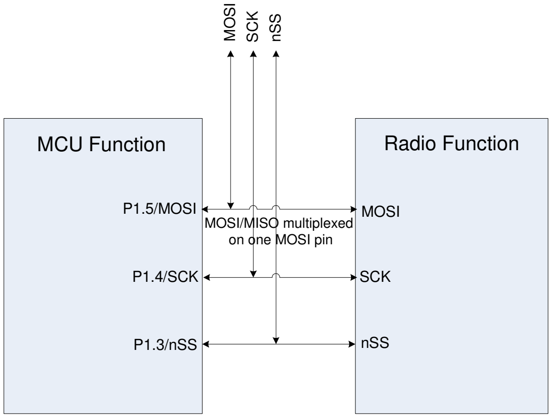
SPI Interface

The SPI interface between the MCU function and the radio function is a 3-wire SPI Interface. The three pins are MOSI (Master Out Slave In), SCK (Serial Clock), SS (Slave Select). There is an alternate 4-wire MISO Interface that requires the connection of two external pins. The SPI interface is controlled by configuring the SPI Configure Register (SICR Address: 0x3D).

Three-Wire SPI Interface

The radio function receives a clock from the MCU function on the SCK pin. The MOSI pin is multiplexed with the MISO pin. Bidirectional data transfer takes place between the MCU function and the radio function through this multiplexed MOSI pin. When using this mode the user firmware should ensure that the MOSI pin on the MCU function is in a high impedance state, except when the MCU is actively transmitting data. Firmware must also control the direction of data flow and switch directions between MCU function and radio function by setting the SWAP bit [Bit 7] of the SPI Configure Register. The SS pin is asserted prior to initiating a data transfer between the MCU function and the radio function. The IRQ function may be optionally multiplexed with the MOSI pin; when this option is enabled the IRQ function is not available while the SS pin is low. When using this configuration, user firmware should ensure that the MOSI function on MCU function is in a high impedance state whenever SS is high.

Figure 4. Three-Wire SPI Mode

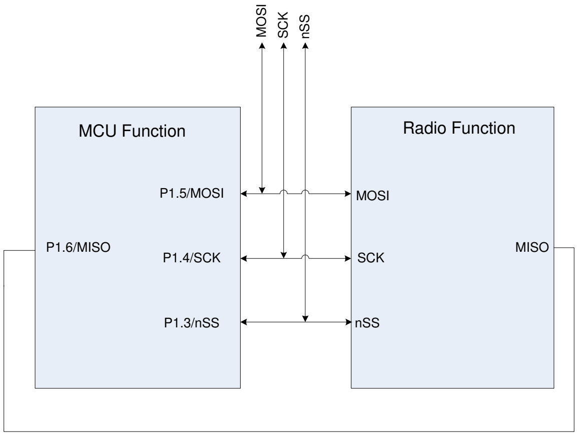


Four-Wire SPI Interface

The four-wire SPI communications interface consists of MOSI, MISO, SCK, and SS.

The device receives SCK from the MCU function on the SCK pin. Data from the MCU function is shifted in on the MOSI pin. Data to the MCU function is shifted out on the MISO pin. The active low SS pin must be asserted for the two functions to communicate. The IRQ function may be optionally multiplexed with the MOSI pin; when this option is enabled the IRQ function is not available while the SS pin is low. When using this configuration, user firmware should ensure that the MOSI function on MCU function is in a high impedance state whenever SS is high.

Figure 5. Four-Wire SPI Mode



This connection is external to the PRoC LPstar Chip

SPI Communication and Transactions

The SPI transactions can be single byte or multi-byte. The MCU function initiates a data transfer through a Command/Address byte. The following bytes are data bytes. The SPI transaction format is shown in Table 2 on page 12.

The DIR bit specifies the direction of data transfer. 0 = Master reads from slave. 1 = Master writes to slave.

The INC bit helps to read or write consecutive bytes from contiguous memory locations in a single burst mode operation.

If Slave Select is asserted and INC = 1, then the master MCU function reads a byte from the radio, the address is incremented by a byte location, and then the byte at that location is read, and so on. If Slave Select is asserted and INC = 0, then the MCU function reads/writes the bytes in the same register in burst mode, but if it is a register file then it reads/writes the bytes in that register file.

The SPI interface between the radio function and the MCU is not dependent on the internal 12 MHz oscillator of the radio. Therefore, radio function registers can be read from or written into while the radio is in sleep mode.

SPI I/O Voltage References

The SPI interfaces between MCU function and the radio and the IRQ and RST have a separate voltage reference V_{IO}, enabling the radio function to directly interface with the MCU function, which operates at higher supply voltage. The internal SPIO pins between the MCU function and radio function should be connected with a regulated voltage of 3.3 V (by setting [bit4] of Registers P13CR, P14CR, P15CR, and P16CR of the MCU function) and the internal 3.3 V regulator of the MCU function should be turned on.

SPI Connects to External Devices

The three SPI wires, MOSI, SCK, and SS are also drawn out of the package as external pins to allow the user to interface their own external devices (such as optical sensors and others) through SPI. The radio function also has its own SPI wires MISO and IRQ, which can be used to send data back to the MCU function or send an interrupt request to the MCU function. They can also be configured as GPIO pins.

Table 2. SPI Transaction Format

	Byte 1			Byte 1+N
Bit#	7	6	[5:0]	[7:0]
Bit Name	DIR	INC	Address	Data

CPU Architecture

This family of microcontrollers is based on a high performance, 8-bit, Harvard-architecture microprocessor. Five registers control the primary operation of the CPU core. These registers are affected by various instructions, but are not directly accessible through the register space by the user.

Table 3. CPU Registers and Register Names

Register	Register Name
Flags	CPU_F
Program Counter	CPU_PC
Accumulator	CPU_A
Stack Pointer	CPU_SP
Index	CPU_X

The 16-bit Program Counter Register (CPU_PC) allows for direct addressing of the full eight Kbytes of program memory space.

The Accumulator Register (CPU_A) is the general purpose register that holds the results of instructions that specify any of the source addressing modes.

The Index Register (CPU_X) holds an offset value that is used in the indexed addressing modes. Typically, this is used to address a block of data within the data memory space.

The Stack Pointer Register (CPU_SP) holds the address of the current top-of-stack in the data memory space. It is affected by the PUSH, POP, LCALL, CALL, RETI, and RET instructions, which manage the software stack. It can also be affected by the SWAP and ADD instructions.

The Flag Register (CPU_F) has three status bits: Zero Flag bit [1]; Carry Flag bit [2]; Supervisory State bit [3]. The Global Interrupt Enable bit [0] is used to globally enable or disable interrupts. The user cannot manipulate the Supervisory State status bit [3]. The flags are affected by arithmetic, logic, and shift operations. The manner in which each flag is changed is dependent upon the instruction being executed (for example, AND, OR, XOR). See Table 20 on page 18.

CPU Registers

Flags Register

The Flags Register can only be set or reset with logical instruction.

Table 4. CPU Flags Register (CPU_F) [R/W]

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field	Reserved			XIO	Super	Carry	Zero	Global IE
Read/Write	–	–	–	R/W	R	RW	RW	RW
Default	0	0	0	0	0	0	1	0

Bits 7:5 Reserved

Bit 4 XIO

Set by the user to select between the register banks
0 = Bank 0
1 = Bank 1

Bit 3 Super

Indicates whether the CPU is executing user code or Supervisor Code. (This code cannot be accessed directly by the user.)
0 = User Code
1 = Supervisor Code

Bit 2 Carry

Set by CPU to indicate whether there has been a carry in the previous logical/arithmetic operation
0 = No Carry
1 = Carry

Bit 1 Zero

Set by CPU to indicate whether there has been a zero result in the previous logical/arithmetic operation
0 = Not Equal to Zero
1 = Equal to Zero

Bit 0 Global IE

Determines whether all interrupts are enabled or disabled
0 = Disabled
1 = Enabled

Note CPU_F register is only readable with explicit register address 0xF7. The *OR F, expr and AND F, expr* instructions must be used to set and clear the CPU_F bits

Accumulator Register

Table 5. CPU Accumulator Register (CPU_A)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field	CPU Accumulator [7:0]							
Read/Write	–	–	–	–	–	–	–	–
Default	0	0	0	0	0	0	0	0

Bits 7:0 CPU Accumulator [7:0]

8-bit data value holds the result of any logical/arithmetic instruction that uses a source addressing mode

Index Register

Table 6. CPU X Register (CPU_X)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field	X [7:0]							
Read/Write	–	–	–	–	–	–	–	–
Default	0	0	0	0	0	0	0	0

Bits 7:0 X [7:0]

8-bit data value holds an index for any instruction that uses an indexed addressing mode

Stack Pointer Register

Table 7. CPU Stack Pointer Register (CPU_SP)

Bit #	7	6	5	4	3	2	1	0
Field	Stack Pointer [7:0]							
Read/Write	–	–	–	–	–	–	–	–
Default	0	0	0	0	0	0	0	0

Bits 7:0 Stack Pointer [7:0]
8-bit data value holds a pointer to the current top-of-stack

CPU Program Counter High Register

Table 8. CPU Program Counter High Register (CPU_PCH)

Bit #	7	6	5	4	3	2	1	0
Field	Program Counter [15:8]							
Read/Write	–	–	–	–	–	–	–	–
Default	0	0	0	0	0	0	0	0

Bits 7:0 Program Counter [15:8]
8-bit data value holds the higher byte of the program counter

CPU Program Counter Low Register

Table 9. CPU Program Counter Low Register (CPU_PCL)

Bit #	7	6	5	4	3	2	1	0
Field	Program Counter [7:0]							
Read/Write	–	–	–	–	–	–	–	–
Default	0	0	0	0	0	0	0	0

Bits 7:0 Program Counter [7:0]
8-bit data value holds the lower byte of the program counter

Addressing Modes

Examples of the different addressing modes are discussed in this section and example code is given.

Source Immediate

The result of an instruction using this addressing mode is placed in the A register, the F register, the SP register, or the X register, which is specified as part of the instruction opcode. Operand 1 is an immediate value that serves as a source for the instruction. Arithmetic instructions require two sources. Instructions using this addressing mode are two bytes in length.

Table 10. Source Immediate

Opcode	Operand 1
Instruction	Immediate Value

Examples

```
ADD   A,   7   ;In this case, the immediate value
               ;of 7 is added with the Accumulator,
               ;and the result is placed in the
               ;Accumulator.

MOV   X,   8   ;In this case, the immediate value
               ;of 8 is moved to the X register.

AND   F,   9   ;In this case, the immediate value
               ;of 9 is logically ANDed with the F
               ;register and the result is placed
               ;in the F register.
```

Source Direct

The result of an instruction using this addressing mode is placed in either the A register or the X register, which is specified as part of the instruction opcode. Operand 1 is an address that points to a location in either the RAM memory space or the register space that is the source for the instruction. Arithmetic instructions require two sources; the second source is the A register or X register specified in the opcode. Instructions using this addressing mode are two bytes in length.

Table 11. Source Direct

Opcode	Operand 1
Instruction	Source Address

Examples

```
ADD   A,   [7]      ;In this case, the value in
                    ;the RAM memory location at
                    ;address 7 is added with the
                    ;Accumulator, and the result
                    ;is placed in the Accumulator.

MOV   X,   REG[8]   ;In this case, the value in
                    ;the register space at address
                    ;8 is moved to the X register.
```

Source Indexed

The result of an instruction using this addressing mode is placed in either the A register or the X register, which is specified as part of the instruction opcode. Operand 1 is added to the X register forming an address that points to a location in either the RAM memory space or the register space that is the source for the instruction. Arithmetic instructions require two sources; the second source is the A register or X register specified in the opcode. Instructions using this addressing mode are two bytes in length.

Table 12. Source Indexed

Opcode	Operand 1
Instruction	Source Index

Examples

```
ADD   A,   [X+7]      ;In this case, the value in
                      ;the memory location at
                      ;address X + 7 is added with
                      ;the Accumulator, and the
                      ;result is placed in the
                      ;Accumulator.

MOV   X,   REG[X+8]   ;In this case, the value in
                      ;the register space at
                      ;address X + 8 is moved to
                      ;the X register.
```

Destination Direct

The result of an instruction using this addressing mode is placed within either the RAM memory space or the register space. Operand 1 is an address that points to the location of the result. The source for the instruction is either the A register or the X register, which is specified as part of the instruction opcode. Arithmetic instructions require two sources; the second source is the location specified by Operand 1. Instructions using this addressing mode are two bytes in length.

Table 13. Destination Direct

Opcode	Operand 1
Instruction	Destination Address

Examples

```
ADD   [7],      A   ;In this case, the value in
                    ;the memory location at
                    ;address 7 is added with the
                    ;Accumulator, and the result
                    ;is placed in the memory
                    ;location at address 7. The
                    ;Accumulator is unchanged.

MOV   REG[8],   A   ;In this case, the Accumula-
                    ;tor is moved to the regis-
                    ;ter space location at
                    ;address 8. The Accumulator
                    ;is unchanged.
```

Destination Indexed

The result of an instruction using this addressing mode is placed within either the RAM memory space or the register space. Operand 1 is added to the X register forming the address that points to the location of the result. The source for the instruction is the A register. Arithmetic instructions require two sources; the second source is the location specified by Operand 1 added with the X register. Instructions using this addressing mode are two bytes in length.

Table 14. Destination Indexed

Opcode	Operand 1
Instruction	Destination Index

Example

```
ADD  [X+7],    A    ;In this case, the value in the
                    ;memory location at address X+7
                    ;is added with the Accumulator,
                    ;and the result is placed in
                    ;the memory location at address
                    ;x+7. The Accumulator is
                    ;unchanged.
```

Destination Direct Source Immediate

The result of an instruction using this addressing mode is placed within either the RAM memory space or the register space. Operand 1 is the address of the result. The source for the instruction is Operand 2, which is an immediate value. Arithmetic instructions require two sources; the second source is the location specified by Operand 1. Instructions using this addressing mode are three bytes in length.

Table 15. Destination Direct Immediate

Opcode	Operand 1	Operand 2
Instruction	Destination Address	Immediate Value

Examples

```
ADD  [7],      5   ;In this case, value in the
                   ;memory location at address 7 is
                   ;added to the immediate value of
                   ;5, and the result is placed in
                   ;the memory location at address 7.

MOV  REG[8],   6   ;In this case, the immediate
                   ;value of 6 is moved into the
                   ;register space location at
                   ;address 8.
```

Destination Indexed Source Immediate

The result of an instruction using this addressing mode is placed within either the RAM memory space or the register space. Operand 1 is added to the X register to form the address of the result. The source for the instruction is Operand 2, which is an immediate value. Arithmetic instructions require two sources; the second source is the location specified by Operand 1 added with the X register. Instructions using this addressing mode are three bytes in length.

Table 16. Destination Indexed Immediate

Opcode	Operand 1	Operand 2
Instruction	Destination Index	Immediate Value

Examples

```
ADD   [X+7],      5     ;In this case, the value in
                        ;the memory location at
                        ;address X+7 is added with
                        ;the immediate value of 5,
                        ;and the result is placed
                        ;in the memory location at
                        ;address X+7.

MOV   REG[X+8],   6     ;In this case, the immedi-
                        ;ate value of 6 is moved
                        ;into the location in the
                        ;register space at
                        ;address X+8.
```

Destination Direct Source Direct

The result of an instruction using this addressing mode is placed within the RAM memory. Operand 1 is the address of the result. Operand 2 is an address that points to a location in the RAM memory that is the source for the instruction. This addressing mode is only valid on the MOV instruction. The instruction using this addressing mode is three bytes in length.

Table 17. Destination Direct Source Direct

Opcode	Operand 1	Operand 2
Instruction	Destination Address	Source Address

Example

```
MOV   [7],   [8]  ;In this case, the value in the
                  ;memory location at address 8 is
                  ;moved to the memory location at
                  ;address 7.
```

Source Indirect Post Increment

The result of an instruction using this addressing mode is placed in the Accumulator. Operand 1 is an address pointing to a location within the memory space, which contains an address (the indirect address) for the source of the instruction. The indirect address is incremented as part of the instruction execution. This addressing mode is only valid on the MVI instruction. The instruction using this addressing mode is two bytes in length. Refer to the *PSoC Designer: Assembly Language User Guide* for further details on MVI instruction.

Table 18. Source Indirect Post Increment

Opcode	Operand 1
Instruction	Source Address Address

Example

```
MVI   A,    [8]   ;In this case, the value in the
                  ;memory location at address 8 is
                  ;an indirect address. The memory
                  ;location pointed to by the indi-
                  ;rect address is moved into the
                  ;Accumulator. The indirect
                  ;address is then incremented.
```

Destination Indirect Post Increment

The result of an instruction using this addressing mode is placed within the memory space. Operand 1 is an address pointing to a location within the memory space, which contains an address (the indirect address) for the destination of the instruction. The indirect address is incremented as part of the instruction execution. The source for the instruction is the Accumulator. This addressing mode is only valid on the MVI instruction. The instruction using this addressing mode is two bytes in length.

Table 19. Destination Indirect Post Increment

Opcode	Operand 1
Instruction	Destination Address Address

Example

```
MVI    [8],   A      ;In this case, the value in
                     ;the memory location at
                     ;address 8 is an indirect
                     ;address. The Accumulator is
                     ;moved into the memory loca-
                     ;tion pointed to by the indi-
                     ;rect address. The indirect
                     ;address is then incremented.
```

Instruction Set Summary

The instruction set is summarized in Table 20 numerically and serves as a quick reference. If more information is needed, the Instruction Set Summary tables are described in detail in the *PSoC Designer Assembly Language User Guide* (available on www.cypress.com).

Table 20. Instruction Set Summary Sorted Numerically by Opcode Order [1, 2]

Opcode Hex	Cycles	Bytes	Instruction Format	Flags	Opcode Hex	Cycles	Bytes	Instruction Format	Flags	Opcode Hex	Cycles	Bytes	Instruction Format	Flags
00	15	1	SSC		2D	8	2	OR [X+expr], A	Z	5A	5	2	MOV [expr], X	
01	4	2	ADD A, expr	C, Z	2E	9	3	OR [expr], expr	Z	5B	4	1	MOV A, X	Z
02	6	2	ADD A, [expr]	C, Z	2F	10	3	OR [X+expr], expr	Z	5C	4	1	MOV X, A	
03	7	2	ADD A, [X+expr]	C, Z	30	9	1	HALT		5D	6	2	MOV A, reg[expr]	Z
04	7	2	ADD [expr], A	C, Z	31	4	2	XOR A, expr	Z	5E	7	2	MOV A, reg[X+expr]	Z
05	8	2	ADD [X+expr], A	C, Z	32	6	2	XOR A, [expr]	Z	5F	10	3	MOV [expr], [expr]	
06	9	3	ADD [expr], expr	C, Z	33	7	2	XOR A, [X+expr]	Z	60	5	2	MOV reg[expr], A	
07	10	3	ADD [X+expr], expr	C, Z	34	7	2	XOR [expr], A	Z	61	6	2	MOV reg[X+expr], A	
08	4	1	PUSH A		35	8	2	XOR [X+expr], A	Z	62	8	3	MOV reg[expr], expr	
09	4	2	ADC A, expr	C, Z	36	9	3	XOR [expr], expr	Z	63	9	3	MOV reg[X+expr], expr	
0A	6	2	ADC A, [expr]	C, Z	37	10	3	XOR [X+expr], expr	Z	64	4	1	ASL A	C, Z
0B	7	2	ADC A, [X+expr]	C, Z	38	5	2	ADD SP, expr		65	7	2	ASL [expr]	C, Z
0C	7	2	ADC [expr], A	C, Z	39	5	2	CMP A, expr	if (A=B) Z=1 if (A<B) C=1	66	8	2	ASL [X+expr]	C, Z
0D	8	2	ADC [X+expr], A	C, Z	3A	7	2	CMP A, [expr]		67	4	1	ASR A	C, Z
0E	9	3	ADC [expr], expr	C, Z	3B	8	2	CMP A, [X+expr]		68	7	2	ASR [expr]	C, Z
0F	10	3	ADC [X+expr], expr	C, Z	3C	8	3	CMP [expr], expr		69	8	2	ASR [X+expr]	C, Z
10	4	1	PUSH X		3D	9	3	CMP [X+expr], expr		6A	4	1	RLC A	C, Z
11	4	2	SUB A, expr	C, Z	3E	10	2	MVI A, [[expr]++]	Z	6B	7	2	RLC [expr]	C, Z
12	6	2	SUB A, [expr]	C, Z	3F	10	2	MVI [[expr]++], A		6C	8	2	RLC [X+expr]	C, Z
13	7	2	SUB A, [X+expr]	C, Z	40	4	1	NOP		6D	4	1	RRC A	C, Z
14	7	2	SUB [expr], A	C, Z	41	9	3	AND reg[expr], expr	Z	6E	7	2	RRC [expr]	C, Z
15	8	2	SUB [X+expr], A	C, Z	42	10	3	AND reg[X+expr], expr	Z	6F	8	2	RRC [X+expr]	C, Z
16	9	3	SUB [expr], expr	C, Z	43	9	3	OR reg[expr], expr	Z	70	4	2	AND F, expr	C, Z
17	10	3	SUB [X+expr], expr	C, Z	44	10	3	OR reg[X+expr], expr	Z	71	4	2	OR F, expr	C, Z
18	5	1	POP A	Z	45	9	3	XOR reg[expr], expr	Z	72	4	2	XOR F, expr	C, Z
19	4	2	SBB A, expr	C, Z	46	10	3	XOR reg[X+expr], expr	Z	73	4	1	CPL A	Z
1A	6	2	SBB A, [expr]	C, Z	47	8	3	TST [expr], expr	Z	74	4	1	INC A	C, Z
1B	7	2	SBB A, [X+expr]	C, Z	48	9	3	TST [X+expr], expr	Z	75	4	1	INC X	C, Z
1C	7	2	SBB [expr], A	C, Z	49	9	3	TST reg[expr], expr	Z	76	7	2	INC [expr]	C, Z
1D	8	2	SBB [X+expr], A	C, Z	4A	10	3	TST reg[X+expr], expr	Z	77	8	2	INC [X+expr]	C, Z
1E	9	3	SBB [expr], expr	C, Z	4B	5	1	SWAP A, X	Z	78	4	1	DEC A	C, Z
1F	10	3	SBB [X+expr], expr	C, Z	4C	7	2	SWAP A, [expr]	Z	79	4	1	DEC X	C, Z
20	5	1	POP X		4D	7	2	SWAP X, [expr]		7A	7	2	DEC [expr]	C, Z
21	4	2	AND A, expr	Z	4E	5	1	SWAP A, SP	Z	7B	8	2	DEC [X+expr]	C, Z
22	6	2	AND A, [expr]	Z	4F	4	1	MOV X, SP		7C	13	3	LCALL	
23	7	2	AND A, [X+expr]	Z	50	4	2	MOV A, expr	Z	7D	7	3	LJMP	
24	7	2	AND [expr], A	Z	51	5	2	MOV A, [expr]	Z	7E	10	1	RETI	C, Z
25	8	2	AND [X+expr], A	Z	52	6	2	MOV A, [X+expr]	Z	7F	8	1	RET	
26	9	3	AND [expr], expr	Z	53	5	2	MOV [expr], A		8x	5	2	JMP	
27	10	3	AND [X+expr], expr	Z	54	6	2	MOV [X+expr], A		9x	11	2	CALL	
28	11	1	ROMX	Z	55	8	3	MOV [expr], expr		Ax	5	2	JZ	
29	4	2	OR A, expr	Z	56	9	3	MOV [X+expr], expr		Bx	5	2	JNZ	
2A	6	2	OR A, [expr]	Z	57	4	2	MOV X, expr		Cx	5	2	JC	
2B	7	2	OR A, [X+expr]	Z	58	6	2	MOV X, [expr]		Dx	5	2	JNC	
2C	7	2	OR [expr], A	Z	59	7	2	MOV X, [X+expr]		Ex	7	2	JACC	
										Fx	13	2	INDEX	Z

Notes

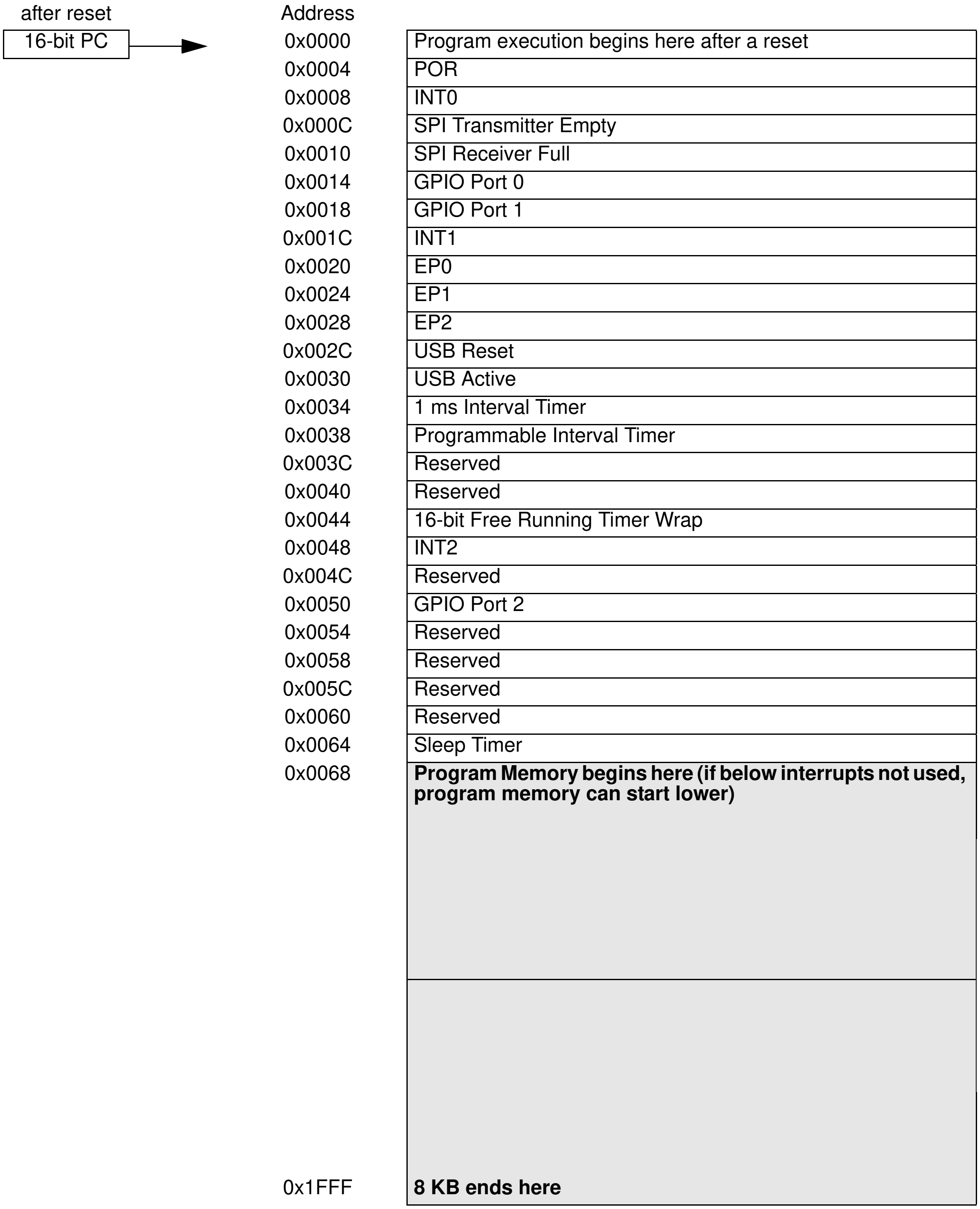
[1] Interrupt routines take 13 cycles before execution resumes at interrupt vector table.

[2] The number of cycles required by an instruction is increased by one for instructions that span 256-byte boundaries in the Flash memory space.

Memory Organization

Flash Program Memory Organization

Figure 6. Program Memory Space with Interrupt Vector Table

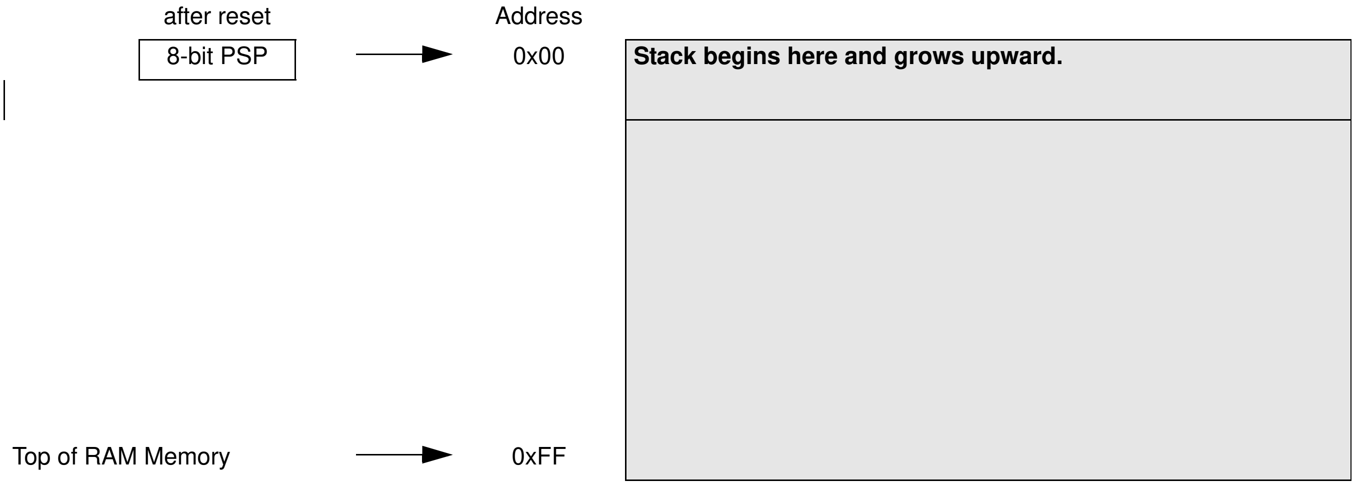


Address	
0x0000	Program execution begins here after a reset
0x0004	POR
0x0008	INT0
0x000C	SPI Transmitter Empty
0x0010	SPI Receiver Full
0x0014	GPIO Port 0
0x0018	GPIO Port 1
0x001C	INT1
0x0020	EP0
0x0024	EP1
0x0028	EP2
0x002C	USB Reset
0x0030	USB Active
0x0034	1 ms Interval Timer
0x0038	Programmable Interval Timer
0x003C	Reserved
0x0040	Reserved
0x0044	16-bit Free Running Timer Wrap
0x0048	INT2
0x004C	Reserved
0x0050	GPIO Port 2
0x0054	Reserved
0x0058	Reserved
0x005C	Reserved
0x0060	Reserved
0x0064	Sleep Timer
0x0068	**Program Memory begins here (if below interrupts not used, program memory can start lower)**
0x1FFF	**8 KB ends here**

Data Memory Organization

The MCU function has 256 bytes of data RAM

Figure 7. Data Memory Organization



Flash

This section describes the Flash block of the CYRF69313. Much of the user-visible Flash functionality, including programming and security, are implemented in the M8C Supervisory Read Only Memory (SROM). CYRF69313 Flash has an endurance of 1000 cycles and 10 year data retention.

Flash Programming and Security

All Flash programming is performed by code in the SROM. The registers that control the Flash programming are only visible to the M8C CPU when it is executing out of SROM. This makes it impossible to read, write, or erase the Flash by bypassing the security mechanisms implemented in the SROM.

Customer firmware can only program the Flash via SROM calls. The data or code images can be sourced by way of any interface with the appropriate support firmware. This type of programming requires a 'bootloader' — a piece of firmware resident on the Flash. For safety reasons this bootloader should not be overwritten during firmware rewrites.

The Flash provides four auxiliary rows that are used to hold Flash block protection flags, boot time calibration values, configuration tables, and any device values. The routines for accessing these auxiliary rows are documented in the SROM section. The auxiliary rows are not affected by the device erase function.

In-System Programming

Most designs that include an CYRF69313 part have a USB connector attached to the USB D+/D– pins on the device. These designs require the ability to program or reprogram a part through these two pins alone.

CYRF69313 device enables this type of in-system programming by using the D+ and D– pins as the serial programming mode interface. This allows an external controller to cause the CYRF69313 part to enter serial programming mode and then to use the test queue to issue Flash access functions in the SROM. The programming protocol is not USB.

SROM

The SROM holds code that is used to boot the part, calibrate circuitry, and perform Flash operations. (Table 21 lists the SROM functions.) The functions of the SROM may be accessed in normal user code or operating from Flash. The SROM exists in a separate memory space from user code. The SROM functions are accessed by executing the Supervisory System Call instruction (SSC), which has an opcode of 00h. Prior to executing the SSC, the M8C's accumulator needs to be loaded with the desired SROM function code from Table 21. Undefined functions causes a HALT if called from user code. The SROM functions are executing code with calls; therefore, the functions require stack space. With the exception of Reset, all of the SROM functions have a *parameter block* in SRAM that must be configured before executing the SSC. Table 22 on page 21 lists all possible parameter block variables. The meaning of each parameter, with regards to a specific SROM function, is described later in this section.

Table 21. SROM Function Codes

Function Code	Function Name	Stack Space
00h	SWBootReset	0
01h	ReadBlock	7
02h	WriteBlock	10
03h	EraseBlock	9
05h	EraseAll	11
06h	TableRead	3
07h	CheckSum	3

Two important variables that are used for all functions are KEY1 and KEY2. These variables are used to help discriminate between valid SSCs and inadvertent SSCs. KEY1 must always have a value of 3Ah, while KEY2 must have the same value as the stack pointer when the SROM function begins execution. This would be the Stack Pointer value when the SSC opcode is executed, plus three. If either of the keys do not match the expected values, the M8C halts (with the exception of the SWBootReset function). The following code puts the correct value in KEY1 and KEY2. The code starts with a halt, to force the program to jump directly into the setup code and not run into it.

```
halt
SSCOP: mov [KEY1], 3ah
mov X, SP
mov A, X
add A, 3
mov [KEY2], A
```

Table 22. SROM Function Parameters

Variable Name	SRAM Address
Key1/Counter/Return Code	0,F8h
Key2/TMP	0,F9h
BlockID	0,FAh
Pointer	0,FBh
Clock	0,FCh
Mode	0,FDh
Delay	0,FEh
PCL	0,FFh

The SROM also features Return Codes and Lockouts.

Return Codes

Return codes aid in the determination of success or failure of a particular function. The return code is stored in KEY1's position in the parameter block. The CheckSum and TableRead functions do not have return codes because KEY1's position in the parameter block is used to return other data.

Table 23. SROM Return Codes

Return Code	Description
00h	Success
01h	Function not allowed due to level of protection on block
02h	Software reset without hardware reset
03h	Fatal error, SROM halted

Read, write, and erase operations may fail if the target block is read or write protected. Block protection levels are set during device programming.

The EraseAll function overwrites data in addition to leaving the entire user Flash in the erase state. The EraseAll function loops through the number of Flash macros in the product, executing the following sequence: erase, bulk program all zeros, erase. After all the user space in all the Flash macros are erased, a second loop erases and then programs each protection block with zeros.

SROM Function Descriptions

All SROM functions are described in the following sections.

SWBootReset Function

The SROM function, SWBootReset, is the function that is responsible for transitioning the device from a reset state to running user code. The SWBootReset function is executed whenever the SROM is entered with an M8C accumulator value of 00h; the SRAM parameter block is not used as an input to the function. This happens, by design, after a hardware reset, because the M8C's accumulator is reset to 00h or when user code executes the SSC instruction with an accumulator value of 00h. The SWBootReset function does not execute when the SSC instruction is executed with a bad key value and a nonzero function code. A CYRF69313 device executes the HALT instruction if a bad value is given for either KEY1 or KEY2.

The SWBootReset function verifies the integrity of the calibration data by way of a 16-bit checksum, before releasing the M8C to run user code.

ReadBlock Function

The ReadBlock function is used to read 64 contiguous bytes from Flash — a block.

The first thing this function does is to check the protection bits and determine if the desired BLOCKID is readable. If read protection is turned on, the ReadBlock function exits, setting the accumulator and KEY2 back to 00h. KEY1 has a value of 01h, indicating a read failure. If read protection is not enabled, the function reads 64 bytes from the Flash using a ROMX instruction and store the results in SRAM using an MVI instruction. The first of the 64 bytes is stored in SRAM at the address indicated by the value of the POINTER parameter. When the ReadBlock completes successfully, the accumulator, KEY1, and KEY2 all have a value of 00h.

Table 24. ReadBlock Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value, when SSC is executed
BLOCKID	0,FAh	Flash block number
POINTER	0,FBh	First of 64 addresses in SRAM where returned data should be stored

WriteBlock Function

The WriteBlock function is used to store data in the Flash. Data is moved 64 bytes at a time from SRAM to Flash using this function. The first thing the WriteBlock function does is to check the protection bits and determine if the desired BLOCKID is writable. If write protection is turned on, the WriteBlock function exits, setting the accumulator and KEY2 back to 00h. KEY1 has a value of 01h, indicating a write failure. The configuration of the WriteBlock function is straightforward. The BLOCKID of the Flash block, where the data is stored, must be determined and stored at SRAM address FAh.

The SRAM address of the first of the 64 bytes to be stored in Flash must be indicated using the POINTER variable in the parameter block (SRAM address FBh). Finally, the CLOCK and DELAY values must be set correctly. The CLOCK value determines the length of the write pulse that is used to store the data in the Flash. The CLOCK and DELAY values are dependent on the CPU speed. Refer to 'Clocking' Section for additional information.

Table 25. WriteBlock Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value, when SSC is executed
BLOCKID	0,FAh	8 KB Flash block number (00h–7Fh) 4 KB Flash block number (00h–3Fh) 3 KB Flash block number (00h–2Fh)
POINTER	0,FBh	First of 64 addresses in SRAM, where the data to be stored in Flash is located prior to calling WriteBlock
CLOCK	0,FCh	Clock divider used to set the write pulse width
DELAY	0,FEh	For a CPU speed of 12 MHz set to 56h

EraseBlock Function

The EraseBlock function is used to erase a block of 64 contiguous bytes in Flash. The first thing the EraseBlock function does is to check the protection bits and determine if the desired BLOCKID is writable. If write protection is turned on, the EraseBlock function exits, setting the accumulator and KEY2 back to 00h. KEY1 has a value of 01h, indicating a write failure. The EraseBlock function is only useful as the first step in programming. Erasing a block does not cause data in a block to be one hundred percent unreadable. If the objective is to obliterate data in a block, the best method is to perform an EraseBlock followed by a WriteBlock of all zeros.

To setup the parameter block for the EraseBlock function, correct key values must be stored in KEY1 and KEY2. The block number to be erased must be stored in the BLOCKID variable and the CLOCK and DELAY values must be set based on the current CPU speed.

Table 26. EraseBlock Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value when SSC is executed
BLOCKID	0,FAh	Flash block number (00h–7Fh)
CLOCK	0,FCh	Clock divider used to set the erase pulse width
DELAY	0,FEh	For a CPU speed of 12 MHz set to 56h

ProtectBlock Function

The CYRF69313 device offers Flash protection on a block-by-block basis. Table 27 lists the protection modes available. In the table, ER and EW are used to indicate the ability to perform external reads and writes. For internal writes, IW is used. Internal reading is always permitted by way of the ROMX instruction. The ability to read by way of the SROM ReadBlock function is indicated by SR. The protection level is stored in two bits according to Table 27. These bits are bit packed into the 64 bytes of the protection block. Therefore, each protection block byte stores the protection level for four Flash blocks. The bits are packed into a byte, with the lowest numbered block's protection level stored in the lowest numbered bits.

The first address of the protection block contains the protection level for blocks 0 through 3; the second address is for blocks 4 through 7. The 64th byte stores the protection level for blocks 252 through 255.

Table 27. Protection Modes

Mode	Settings	Description	Marketing
00b	SR ER EW IW	Unprotected	Unprotected
01b	$\overline{SR}$ $\overline{ER}$ EW IW	Read protect	Factory upgrade
10b	$\overline{SR}$ $\overline{ER}$ $\overline{EW}$ $\overline{IW}$	Disable external write	Field upgrade
11b	$\overline{SR}$ $\overline{ER}$ $\overline{EW}$ $\overline{IW}$	Disable internal write	Full protection

7	6	5	4	3	2	1	0
Block n+3		Block n+2		Block n+1		Block n	

The level of protection is only decreased by an EraseAll, which places zeros in all locations of the protection block. To set the level of protection, the ProtectBlock function is used. This function takes data from SRAM, starting at address 80h, and ORs it with the current values in the protection block. The result of the OR operation is then stored in the protection block. The EraseBlock function does not change the protection level for a block. Because the SRAM location for the protection data is fixed and there is only one protection block per Flash macro, the ProtectBlock function expects very few variables in the parameter block to be set prior to calling the function. The parameter block values that must be set, besides the keys, are the CLOCK and DELAY values.

Table 28. ProtectBlock Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value when SSC is executed
CLOCK	0,FCh	Clock divider used to set the write pulse width
DELAY	0,FEh	For a CPU speed of 12 MHz set to 56h

EraseAll Function

The EraseAll function performs a series of steps that destroy the user data in the Flash macros and resets the protection block in each Flash macro to all zeros (the unprotected state). The EraseAll function does not affect the three hidden blocks above the protection block in each Flash macro. The first of these four hidden blocks is used to store the protection table for its eight Kbytes of user data.

The EraseAll function begins by erasing the user space of the Flash macro with the highest address range. A bulk program of all zeros is then performed on the same Flash macro, to destroy all traces of the previous contents. The bulk program is followed by a second erase that leaves the Flash macro in a state ready for writing. The erase, program, erase sequence is then performed on the next lowest Flash macro in the address space if it exists. Following the erase of the user space, the protection block for the Flash macro with the highest address range is erased. Following the erase of the protection block, zeros are written into every bit of the protection table. The next lowest Flash macro in the address space then has its protection block erased and filled with zeros.

The end result of the EraseAll function is that all user data in the Flash is destroyed and the Flash is left in an unprogrammed state, ready to accept one of the various write commands. The protection bits for all user data are also reset to the zero state.

The parameter block values that must be set, besides the keys, are the CLOCK and DELAY values.

Table 29. EraseAll Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value when SSC is executed
CLOCK	0,FCh	Clock divider used to set the write pulse width
DELAY	0,FEh	For a CPU speed of 12 MHz set to 56h

TableRead Function

The TableRead function gives the user access to part specific data stored in the Flash during manufacturing. It also returns a Revision ID for the die (not to be confused with the Silicon ID).

Table 30. Table Read Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value when SSC is executed
BLOCKID	0,FAh	Table number to read

The table space for the CYRF69313 is simply a 64-byte row broken up into eight tables of eight bytes. The tables are numbered zero through seven. All user and hidden blocks in the CYRF69313 parts consist of 64 bytes.

An internal table holds the Silicon ID and returns the Revision ID. The Silicon ID is returned in SRAM, while the Revision ID is returned in the CPU_A and CPU_X registers. The Silicon ID is a value placed in the table by programming the Flash and is controlled by Cypress Semiconductor Product Engineering. The Revision ID is hard coded into the SROM. The Revision ID is discussed in more detail later in this section.

An internal table holds alternate trim values for the device and returns a one-byte internal revision counter. The internal revision counter starts out with a value of zero and is incremented each time one of the other revision numbers is not incremented. It is reset to zero each time one of the other revision numbers is incremented. The internal revision count is returned in the CPU_A register. The CPU_X register is always set to FFh when trim values are read. The BLOCKID value, in the parameter block, is used to indicate which table should be returned to the user. Only the three least significant bits of the BLOCKID parameter are used by the TableRead function for the CYRF69313. The upper five bits are ignored. When the function is called, it transfers bytes from the table to SRAM addresses F8h–FFh.

The M8C’s A and X registers are used by the TableRead function to return the die’s Revision ID. The Revision ID is a 16-bit value hard coded into the SROM that uniquely identifies the die’s design.

Checksum Function

The Checksum function calculates a 16-bit checksum over a user specifiable number of blocks, within a single Flash macro (Bank) starting from block zero. The BLOCKID parameter is used to pass in the number of blocks to calculate the checksum over. A BLOCKID value of 1 calculates the checksum of only block 0, while a BLOCKID value of 0 calculates the checksum of all 256 user blocks. The 16-bit checksum is returned in KEY1 and KEY2. The parameter KEY1 holds the lower eight bits of the checksum and the parameter KEY2 holds the upper eight bits of the checksum.

The checksum algorithm executes the following sequence of three instructions over the number of blocks times 64 to be checksummed.

```
romx
  add [KEY1], A
adc [KEY2], 0
```

Table 31. Checksum Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value when SSC is executed
BLOCKID	0,FAh	Number of Flash blocks to calculate checksum on

SROM Table Read Description

Figure 8. SROM Table

	F8h	F9h	F8h	F8h	F8h	F8h	F8h	F8h
Table 0	Silicon ID [15-8]	Silicon ID [7-0]						
Table 1								
Table 2								
Table 3								
Table 4								
Table 5								
Table 6								
Table 7								

The Silicon IDs for enCoRe II devices are stored in SROM tables in the part, as shown in Figure 8 on page 24.

The Silicon ID can be read out from the part using SROM Table reads. This is demonstrated in the following pseudo code. As mentioned in the section SROM on page 20, the SROM variables occupy address F8h through FFh in the SRAM. Each of the variables and their definition in given in the section SROM on page 20.

```
AREA SSCParmBlkA(RAM,ABS)

    org  F8h // Variables are defined starting at address F8h

SSC_KEY1:                 ; F8h  supervisory key
SSC_RETURNCODE:     blk 1 ; F8h  result code
SSC_KEY2 :          blk 1  ;F9h  supervisory stack ptr key
SSC_BLOCKID:        blk 1 ; FAh  block ID
SSC_POINTER:        blk 1 ; FBh  pointer to data buffer
SSC_CLOCK:          blk 1 ; FCh  Clock
SSC_MODE:           blk 1 ; FDh  ClockW ClockE multiplier
SSC_DELAY:          blk 1 ; FEh  flash macro sequence delay count
SSC_WRITE_ResultCode: blk 1 ; FFh  temporary result code

_main:
           mov   A, 0
           mov   [SSC_BLOCKID], A // To read from Table 0 - Silicon ID is stored in Table 0
//Call SROM operation to read the SROM table
          mov   X, SP      ; copy SP into X
          mov   A, X       ; A temp stored in X
          add   A, 3               ; create 3 byte stack frame (2 + pushed A)
          mov   [SSC_KEY2], A      ; save stack frame for supervisory code

   ; load the supervisory code for flash operations
          mov   [SSC_KEY1], 3Ah  ;FLASH_OPER_KEY - 3Ah

          mov   A,6        ; load A with specific operation. 06h is the code for Table read Table 21
          SSC              ; SSC call the supervisory ROM

// At the end of the SSC command the silicon ID is stored in F8 (MSB) and F9(LSB) of the SRAM
.terminate:
    jmp .terminate
```

Clocking

The CYRF69313 internal oscillator outputs two frequencies, the Internal 24 MHz Oscillator and the 32 kHz Low power Oscillator.

The Internal 24 MHz Oscillator is designed such that it may be trimmed to an output frequency of 24 MHz over temperature and voltage variation. With the presence of USB traffic, the Internal 24 MHz Oscillator can be set to precisely tune to USB timing requirements (24 MHz ± 1.5%). Without USB traffic, the Internal 24 MHz Oscillator accuracy is 24 MHz ± 5% (between 0 °C–70 °C). No external components are required to achieve this level of accuracy.

The internal low speed oscillator of nominally 32 kHz provides a slow clock source for the CYRF69313 in suspend mode, particularly to generate a periodic wakeup interrupt and also to provide a clock to sequential logic during power-up and power-down events when the main clock is stopped. In addition, this oscillator can also be used as a clocking source for the Interval Timer clock (ITMRCLK) and Capture Timer clock (TCAPCLK). The 32 kHz Low power Oscillator can operate in low power mode or can provide a more accurate clock in normal mode. The Internal 32 kHz Low power Oscillator accuracy ranges (between 0 °C–70° C) as follows:

5 V Normal mode: –8% to + 16%

5 V LPstar mode: +12% to + 48%

When using the 32 kHz oscillator the PITMRL/H should be read until two consecutive readings match before sending/receiving data. The following firmware example assumes the developer is interested in the lower byte of the PIT.

```
Read_PIT_counter:
mov A, reg[PITMRL]
mov [57h], A
mov A, reg[PITMRL]
mov [58h], A
mov [59h], A
mov A, reg{PITMRL]
mov [60h], A
;;;Start comparison
mov A, [60h]
mov X, [59h]
sub A, [59h]
jz done
mov A, [59h]
mov X, [58h]
sub A, [58h]
jz done
mov X, [57h]
;;;correct data is in memory location 57h
done:
mov [57h], X
ret
```