Si4010

CRYSTAL-LESS SOC RF TRANSMITTER

Single Coin-cell Battery Transmitter
- Supply voltage: 1.8 to 3.6 V
- Standby current < 10 nA
- Crystal-less operation
- Temperature range –40 to +85 °C
- Automotive quality option, AEC-Q100
- 10-pin MSOP/14-pin SOIC
- Pb free/RoHS compliant

RF Transmitter
- Frequency range: 27—960 MHz
- +10 dBm output power, adjustable
- Automatic antenna tuning
- Symbol rate up to 100 kbps
- FSK/OOK modulation
- Manchester, NRZ, 4/5 encoder

Analog Peripherals
- LDO regulator with POR circuit
- Integrated temperature sensor
- Battery voltage monitor

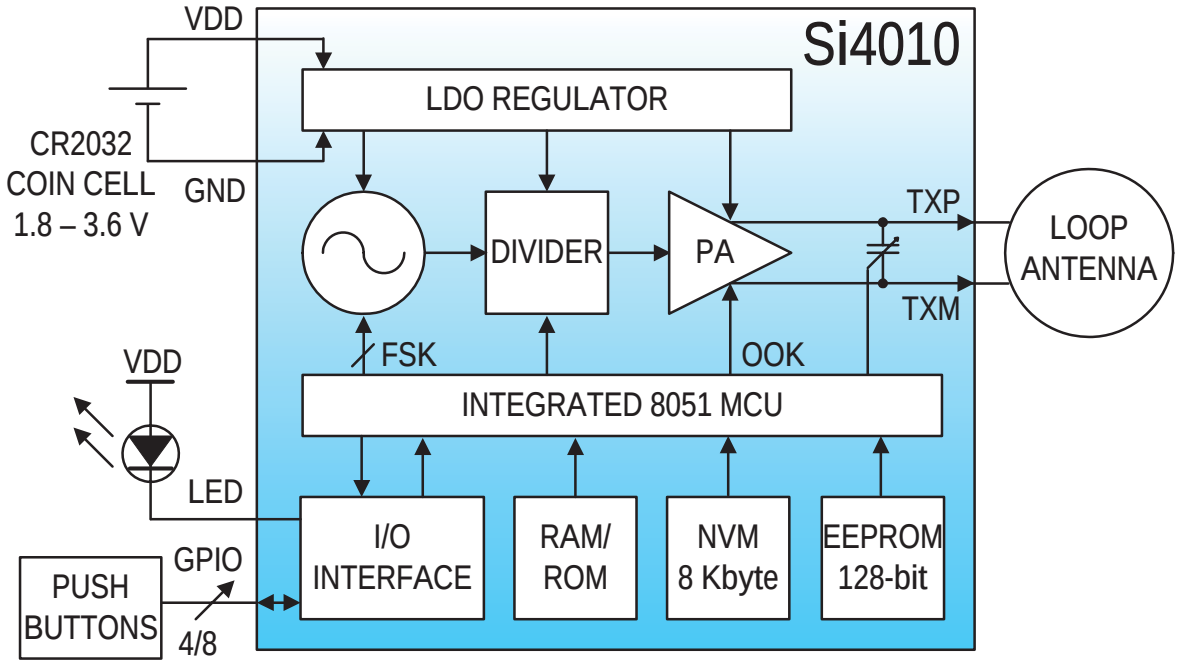
High-Speed 8051 µC Core
- Pipeline instruction architecture
- 70% of instructions in 1 or 2 clocks
- Up to 24 MIPs with 24 MHz clock

Memory
- 4 kB RAM/8kB NVM
- 128 bit EEPROM
- 256 byte of internal data RAM
- 256 byte of external data RAM (XREG)
- 12 kB ROM embedded functions
- 8 byte low leakage RAM

Digital Peripherals
- 128 bit AES Accelerator
- 4/8 GPIO with wakeup functionality
- 1 LED driver
- Data serializer
- High-speed frequency counter
- RTC, Timers 2, 3
- On-chip debugging - C2

Clock Sources
- High-speed crystal-less VCO
- Programmable low-power osc - LPOSC
- Ultra low-power sleep timer
- Optional crystal oscillator input

Applications
- Garage and gate door openers
- Home automation and security
- Remote keyless entry



Copyright © 2010 by Silicon Laboratories

This information applies to a product under development. Its characteristics and specifications are subject to change without notice.

TABLE OF CONTENTS

1. System Overview. 11
2. Test Circuit. 13
3. Typical Application Schematic 14
3.1. Si4010 Used in a 5-Button RKE System with LED Indicator 14
3.2. Si4010 with an External Crystal in a 4-Button RKE System with LED Indicator 14
4. Ordering Information 15
5. Pin Definitions 16
5.1. MSOP, Application 16
5.2. MSOP, Programming/Debug Mode 17
5.3. SOIC Package, Application 18
5.4. SOIC Package, Programming/debug Mode 19
6. Package Specifications 20
6.1. 10-Pin MSOP 20
6.2. 14-pin SOIC Package 21
7. PCB Land Pattern 10-Pin MSOP 22
8. PCB Land Pattern 14-pin SOIC Package. 24
9. Electrical Characteristics 26
10. System Description. 33
10.1. Overview 33
10.2. Setting Basic Si4010 Transmit Parameters 35
10.2.1. Package Type. 35
10.2.2. Output Power. 35
10.2.3. Modulation, Encoding, and Data Rate. 37
10.2.4. Output Frequency. 37
10.2.5. Battery Life Calculation. 38
10.3. Applications Programming Interface (API) Commands. 39
11. Power Amplifier. 42
11.1. Register Description. 44
12. Output Data Serializer (ODS) 46
12.1. Description 46
12.2. Timing 46
12.3. Register Description. 47
13. LC Oscillator (LCOSC) 52
13.1. Register Description. 52
14. Low Power Oscillator and System Clock Generator 53
14.1. Register Description. 53
15. Crystal Oscillator (XO) 55
15.1. Register Description. 55
16. Frequency Counter 56
16.1. Register Description. 58
17. Sleep Timer 60
18. Bandgap and LDO. 60
19. Low Leakage HVRAM 60

20. Temperature Sensor 60
21. CIP-51 Microcontroller 61
21.1. Instruction Set 62
21.1.1. Instruction and CPU Timing 62
21.2. CIP-51 Register Descriptions 67
22. Memory Organization 71
22.1. Program Memory 72
22.2. Internal Data Memory 72
22.3. External Data Memory 72
22.4. General Purpose Registers 72
22.5. Bit Addressable Locations 73
22.6. Stack 73
22.7. Special Function Registers (SFR) 73
22.8. Registers Mapped to XDATA Address Space (XREG) 73
22.9. NVM (OTP) Memory 73
22.10. MTP (EEPROM) Memory 74
23. System Boot and NVM Programming 75
23.1. Startup Overview 75
23.2. Reset 76
23.3. Chip Program Levels 76
23.4. NVM Organization 77
23.5. Device Boot Process 78
23.6. Error Handling During Boot 79
23.7. CODE/XDATA RAM Address Map 79
23.8. Boot Status Variables 81
23.9. Boot Routine Destination Address Space 84
23.10. NVM Programming 85
23.11. Retest and Retest Configuration 86
23.12. Boot and Retest Protection Control Register 88
23.13. Chip Protection Control Register 89
24. On-Chip Registers 90
24.1. Special Function Registers 90
24.2. XREG Registers 93
25. Interrupts 96
25.1. MCU Interrupt Sources and Vectors 97
25.2. Interrupt Priorities 97
25.3. Interrupt Latency 97
25.4. Interrupt Register Descriptions 98
25.5. External Interrupts 104
26. Power Management Modes 106
26.1. Idle Mode 106
26.2. Stop Mode 106
27. AES Hardware Accelerator 108
27.1. AES SFR Registers 108
28. Reset Sources 111
28.1. Device Boot Outline 111

28.2. External Reset 111
28.3. Software Reset. 111
29. Port Input/Output. 112
29.1. GPIO Pin Special Roles 115
29.2. Pullup Roff Option 116
29.3. Matrix Mode Option 116
29.4. Pullup Roff and Matrix Mode Option Control 118
29.5. Special GPIO Modes Control 119
29.6. LED Driver on GPIO[5]. 121
30. Clock Output Generation 127
30.1. Register Description 128
31. Control and System Setting Registers 130
32. Real Time Clock Timer 132
32.1. RTC Interrupt Flag Time Uniformity 133
32.2. Register Description 133
33. Timers 2 and 3 135
33.1. Interrupt Flag Generation 136
33.2. 16-bit Timer with Auto Reload (Wide Mode) 137
33.3. 16-bit Capture Mode (Wide Mode). 137
33.4. 8-bit Timer/Timer Mode (Split Mode) 138
33.5. 8-bit Capture/Capture Mode (Split Mode) 139
33.6. 8-bit Timer/Capture Mode (Split Mode) 140
34. C2 Interface 152
34.1. C2 Pin Sharing 152
35. IDE Development Environment and Debugging Chain 155
35.1. Functionality Limitations While Using IDE Development Environment 155
35.2. Chip Shutdown Limitation 156
35.3. LED Driver Usage while Using IDE Debugging Chain 156
35.4. LED Driver and Application Development Issues 157
36. Additional Reference Resources 158
Document Change List 159
Contact Information 160

LIST OF FIGURES

Figure 1.1. Si4010 Block Diagram 12
Figure 2.1. Test Block Diagram with 10-pin MSOP 13
Figure 3.1. Si4010 Used in a 5-button RKE System with LED Indicator 14
Figure 3.2. Si4010 with an External Crystal in a 4-button RKE System with LED Indicator 14
Figure 6.1. 10-pin MSOP Package 20
Figure 6.2. 14-pin SOIC Package 21
Figure 7.1. 10-Pin MSOP Recommended PCB Land Pattern 22
Figure 8.1. 14-pin SOIC Recommended PCB Land Pattern 24
Figure 10.1. Functional Block Diagram 33
Figure 11.1. Simplified PA Block Diagram 42
Figure 12.1. OOK Timing Example 46
Figure 12.2. FSK Timing Example 46
Figure 16.1. Frequency Counter Block Diagram 56
Figure 21.1. CIP-51 Block Diagram 61
Figure 22.1. Address Space Map after the Boot 71
Figure 23.1. NVM Address Map 78
Figure 23.2. CODE/XDATA RAM Address Map 80
Figure 23.3. Boot Routine Destination CPU Address Space for Copy from NVM 84
Figure 29.1. Device Package and Port Assignments 112
Figure 29.2. GPIO[3:1] Functional Diagram 114
Figure 29.3. Other GPIO Functional Diagram 114
Figure 29.4. Push Button Organization in Matrix Mode 117
Figure 29.5. GPIO[5] LED Driver Block Diagram 121
Figure 30.1. Output Clock Generator Block Diagram 127
Figure 32.1. RTC Timer Block Diagram 132
Figure 33.1. Timer Interrupt Generation 136
Figure 33.2. Timer 16-bit Mode Block Diagram (Wide Mode) 137
Figure 33.3. Capture 16-bit Mode Block Diagram (Wide Mode) 138
Figure 33.4. Two 8-bit Timers in Timer/Timer Configuration (Split Mode) 139
Figure 33.5. Two 8-bit Timers in Capture/Capture Configuration (Split Mode) 140
Figure 33.6. Two 8-bit TImers in Timer/Capture Configuration (Split Mode) 141
Figure 33.7. Two 8-bit Timers In Capture/Timer Configuration (Split Mode) 142
Figure 34.1. 10-pin C2 USB Debugging Adapter Connection to Device 152
Figure 34.2. 14-pin C2 ToolStick Connection to Device 154

LIST OF TABLES

Table 4.1. Product Selection Guide 15
Table 6.1. Package Dimensions 20
Table 6.2. Package Dimensions 21
Table 7.1. 10-Pin MSOP Dimensions 23
Table 8.1. PCB Land Pattern Dimensions 25
Table 9.1. Recommended Operating Conditions 26
Table 9.2. Absolute Maximum Ratings[1,2] 26
Table 9.3. DC Characteristics 27
Table 9.4. Si4010 RF Transmitter Characteristics 28
Table 9.5. Low Battery Detector Characteristics 31
Table 9.6. Optional Crystal Oscillator Characteristics 31
Table 9.7. EEPROM Characteristics 31
Table 9.8. Low Power Oscillator Characteristics 32
Table 9.9. Sleep Timer Characteristics 32
Table 21.1. CIP-51 Instruction Set Summary 63
Table 23.1. Boot XDATA Status Variables 81
Table 23.2. Run Chip Retest Protection Flags: NVM Programmer 86
Table 24.1. Special Function Register (SFR) Memory Map 90
Table 24.2. Special Function Registers 91
Table 24.3. XREG Register Memory Map in External Memory 94
Table 24.4. XREG Registers 95
Table 25.1. Interrupt Summary 98
Table 29.1. 10–Pin Mode 113
Table 29.2. 14–Pin Mode 113
Table 29.3. GPIO Special Roles 115
Table 29.4. GPIO Special Roles Control and Order 120

LIST OF XREG REGISTERS

XREG Definition 11.2. wPA_CAP 44
XREG Definition 11.3. bPA_TRIM 45
XREG Definition 14.1. bLPOSC_TRIM 53
XREG Definition 15.1. bXO_CTRL 55
XREG Definition 16.3. lFC_COUNT 59
XREG Definition 22.1. abMTP_RDATA[16] 74

LIST OF SFR REGISTERS

SFR Definition 11.1. PA_LVL 44
SFR Definition 12.1. ODS_CTRL 47
SFR Definition 12.2. ODS_TIMING 48
SFR Definition 12.3. ODS_DATA 49
SFR Definition 12.4. ODS_RATEL 49
SFR Definition 12.5. ODS_RATEH 50
SFR Definition 12.6. ODS_WARM1 50
SFR Definition 12.7. ODS_WARM2 51
SFR Definition 13.1. LC_FSK 52
SFR Definition 14.2. SYSGEN 54
SFR Definition 16.1. FC_CTRL 58
SFR Definition 16.2. FC_INTERVAL 59
SFR Definition 21.1. DPL 67
SFR Definition 21.2. DPH 67
SFR Definition 21.3. SP 68
SFR Definition 21.4. ACC 68
SFR Definition 21.5. B 69
SFR Definition 21.6. PSW 70
SFR Definition 23.1. BOOT_BOOTSTAT 82
SFR Definition 23.2. BOOT_FLAGS 83
SFR Definition 23.3. PROT3_CTRL 88
SFR Definition 23.4. PROT0_CTRL 89
SFR Definition 25.1. IE 99
SFR Definition 25.2. IP 100
SFR Definition 25.3. EIE1 101
SFR Definition 25.4. EIP1 102
SFR Definition 25.5. INT_FLAGS 103
SFR Definition 25.6. PORT_INTCFG 105
SFR Definition 26.1. PCON 107
SFR Definition 27.1. GFM_DATA 109
SFR Definition 27.2. GFM_CONST 109
SFR Definition 27.3. SBOX_DATA 110
SFR Definition 29.1. P0 122
SFR Definition 29.2. P0CON 123
SFR Definition 29.3. P1 123
SFR Definition 29.4. P1CON 124
SFR Definition 29.5. P2 124
SFR Definition 29.6. PORT_CTRL 125
SFR Definition 29.7. PORT_SET 126
SFR Definition 30.1. CLKOUT_SET 128
SFR Definition 31.1. GPR_CTRL 130
SFR Definition 31.2. GPR_DATA 130
SFR Definition 31.3. RBIT_DATA 131

SFR Definition 32.1. RTC_CTRL 134
SFR Definition 33.1. TMR_CLKSEL 143
SFR Definition 33.2. TMR2CTRL 144
SFR Definition 33.3. TMR2RL 146
SFR Definition 33.4. TMR2RH 146
SFR Definition 33.5. TMR2L 147
SFR Definition 33.6. TMR2H 147
SFR Definition 33.7. TMR3CTRL 148
SFR Definition 33.8. TMR3RL 150
SFR Definition 33.9. TMR3RH 150
SFR Definition 33.10. TMR3L 151
SFR Definition 33.11. TMR3H 151

1. System Overview

The Si4010 is a fully integrated crystal-less CMOS SoC RF transmitter with an embedded CIP-51 8051 MCU designed for the sub 1 GHz ISM frequency bands. This chip is optimized for battery powered applications with operating voltages from 1.8 to 3.6 V and ultra-low current consumption with a standby current of less than 10 nA. The high power amplifier can supply up to +10 dBm output power with 19.5 dB of programmable range. Moreover, the SoC transmitter includes a patented antenna tuning circuit that automatically fine tunes the resonance frequency and impedance matching between the PA output and the connected antenna for optimum transmit efficiency and low harmonic content. FSK and OOK modulation is supported with symbol rates up to 100 kbps. Like all wireless devices, users are responsible for complying with applicable local regulatory requirements for radio transmissions.

The embedded CIP-51 8051 MCU provides the core functionality of the Si4010. User software has complete control of all peripherals, and may individually shut down any or all peripherals for power savings. A space of 8 kB of on-chip one-time programmable NVM memory is available to store the user program and can also store unique transmit IDs. In case of power outages due to battery removal, 128 bits of EEPROM is available for counter or other operations providing non-volatile storage capability. A library of useful software functions such as AES encryption, a patented 32-bit counter providing 1 M cycles of read/write endurance, and many other functions are included in the 12 kB of ROM to reduce user design time and code space. General purpose input/output pins with push button wake-on touch capability, a programmable system clock, and ultra low power timers are also available to further reduce current consumption.

The Si4010 includes Silicon Laboratories' 2-wire C2 Debug and Programming interface. This debug logic supports memory inspection, viewing and modification of special function registers (SFR), setting break points, single stepping, and run and halt commands. All analog and digital peripherals are fully functional while debugging using C2. The two C2 interface pins can be shared with user functions, allowing in-system debugging without occupying package pins.

The device leverages Silicon Labs' patented and proven crystal-less oscillator technology and offers better than ±150 ppm carrier frequency stability over the temperature range of 0 to + 70 °C and ±250 ppm carrier frequency stability over the industrial range of –40 to + 85 °C without the use of an external crystal or frequency reference. The internal MCU automatically calibrates the on-chip voltage controlled oscillator (LCOSC) which forms the output carrier frequency for process and temperature variations. An external 1-pin crystal oscillator option is available for applications requiring tighter frequency tolerances.

Digital integration reduces the amount of required external components compared to traditional offerings, resulting in a solution that only requires a printed circuit board (PCB) implementation area of approximately 25 by 50 mm (including battery, switches, and 25 mm^2 antenna). The high integration of the Si4010 improves the system manufacturing reliability and quality and minimizes costs. This chip offers industry leading RF performance, high integration, flexibility, low BOM, small board area, and ease of design. No production alignment is necessary as all RF functions are integrated into the device.

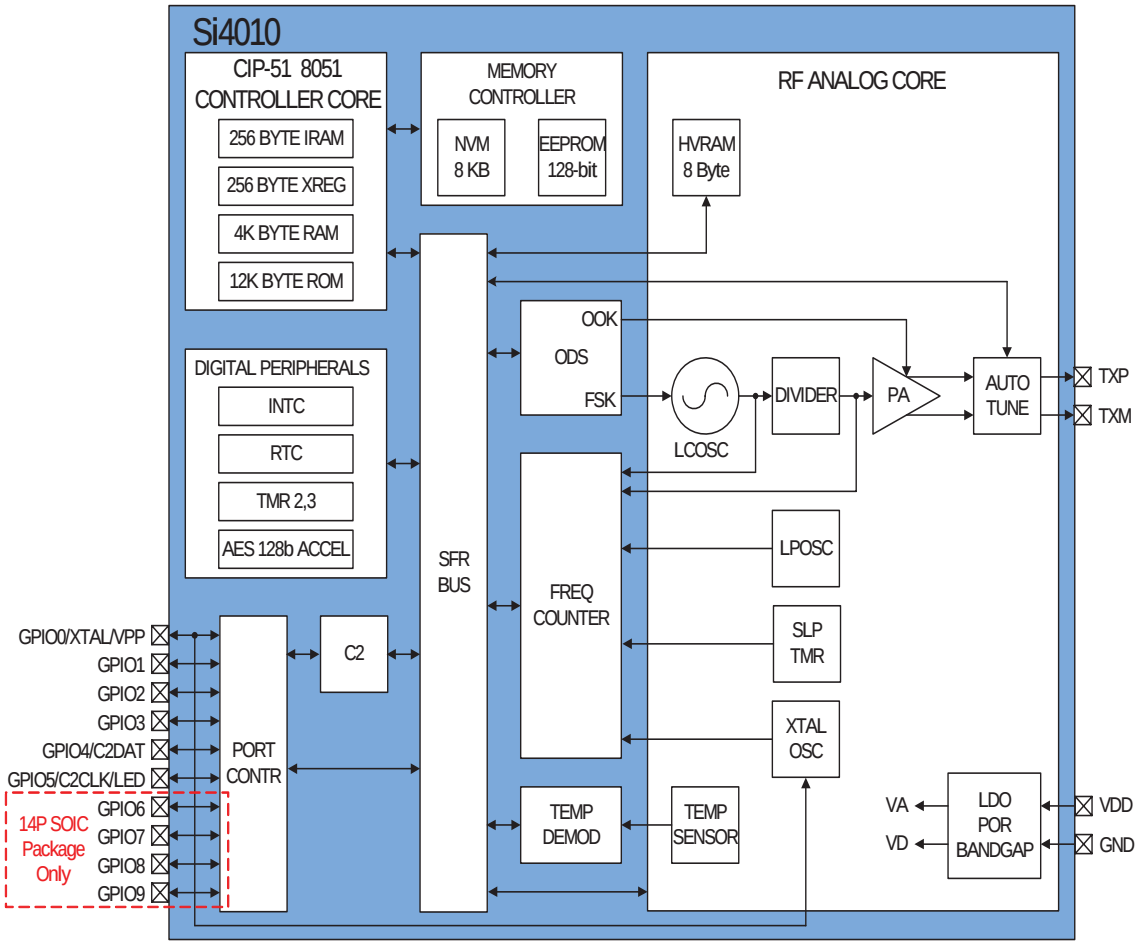


Figure 1.1. Si4010 Block Diagram

2. Test Circuit

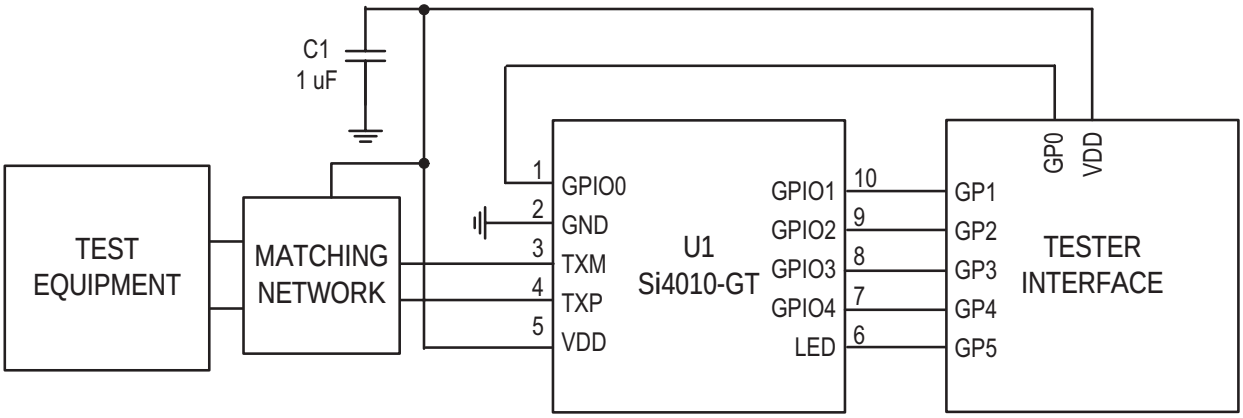


Figure 2.1. Test Block Diagram with 10-Pin MSOP

3. Typical Application Schematic

3.1. Si4010 Used in a 5-Button RKE System with LED Indicator

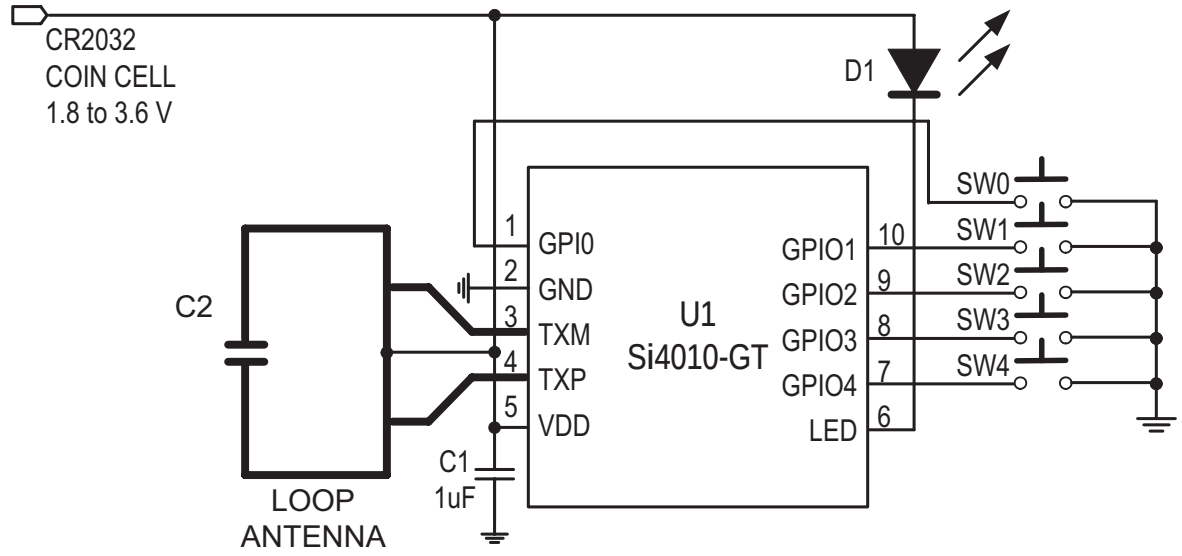


Figure 3.1. Si4010 Used in a 5-button RKE System with LED Indicator

3.2. Si4010 with an External Crystal in a 4-Button RKE System with LED Indicator

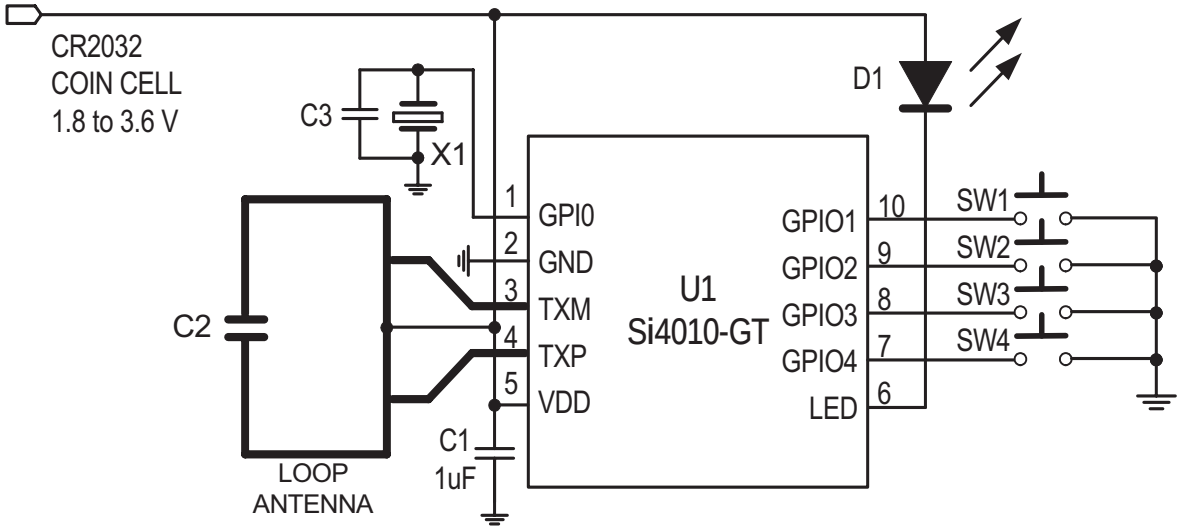


Figure 3.2. Si4010 with an External Crystal in a 4-button RKE System with LED Indicator

4. Ordering Information

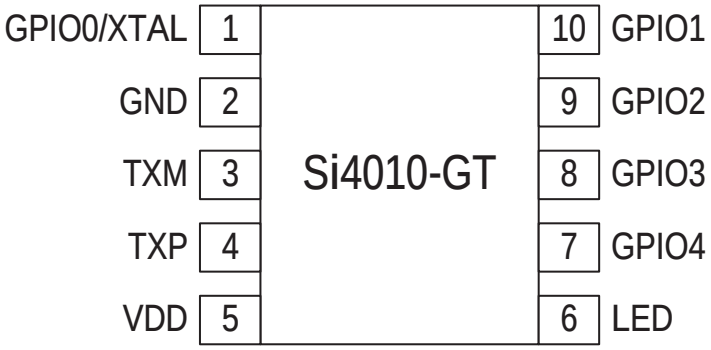
Table 4.1. Product Selection Guide

Ordering Part Number[1]	MIPS (Peak)	NVM (OTP) Memory (Bytes)	RAM (Bytes)	Embedded ROM Functions	Internal Data RAM (Bytes)	HVRAM (Bytes)	EEPROM (Bits)	128-bit AES Accelerator	GPIO with Wakeup[2]	LED Driver	Sleep Timer	+10 dBm RF Transmitter	LDO with POR Circuit	Integrated Temperature Sensor	Low Battery Detector	Automotive Qualified	Lead-free (RoHS Compliant)	Package
Si4010-C2-GT	24	8k	4k	Y	256	8	128	Y	**4**	1	Y	Y	Y	Y	Y	—	Y	MSOP-10
Si4010-C2-GS	24	8k	4k	Y	256	8	128	Y	**8**	1	Y	Y	Y	Y	Y	—	Y	SOIC-14
Si4010-C2-AT	24	8k	4k	Y	256	8	128	Y	**4**	1	Y	Y	Y	Y	Y	**Y**	Y	MSOP-10
Si4010-C2-AS	24	8k	4k	Y	256	8	128	Y	**8**	1	Y	Y	Y	Y	Y	**Y**	Y	SOIC-14

Notes:
1. Add an “(R)” at the end of the device part number to denote tape and reel option.
2. Assumes LED driver is used and no external crystal.

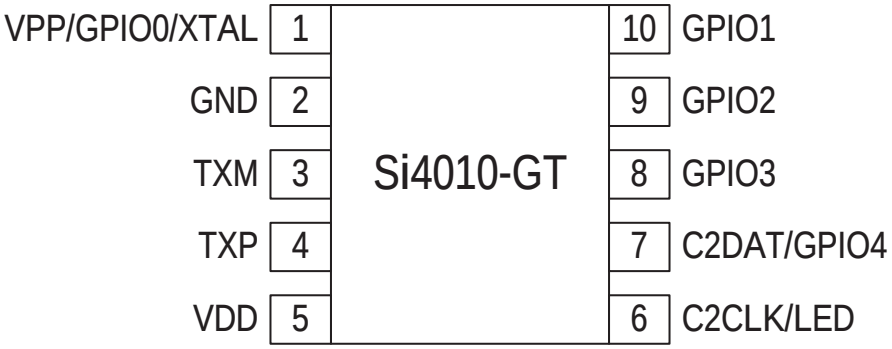
5. Pin Definitions

5.1. MSOP, Application



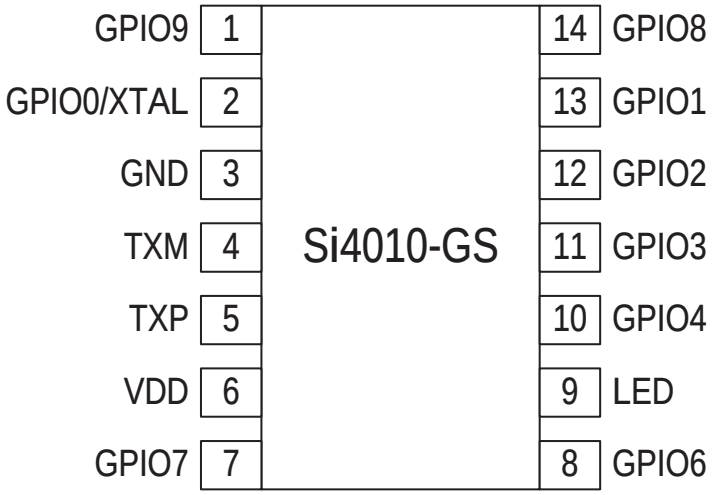
Pin Number(s)	Name	Description
1	GPIO0/XTAL	General purpose input pin. Can be configured as an input pin for a crystal.
2	GND	Ground. Connect to ground plane on PCB.
3, 4	TXM, TXP	Transmitter differential outputs.
5	VDD	Power.
6	LED	Dedicated LED driver.
7, 8, 9, 10	GPIO[4:1]	General purpose input/output pins.

5.2. MSOP, Programming/Debug Mode



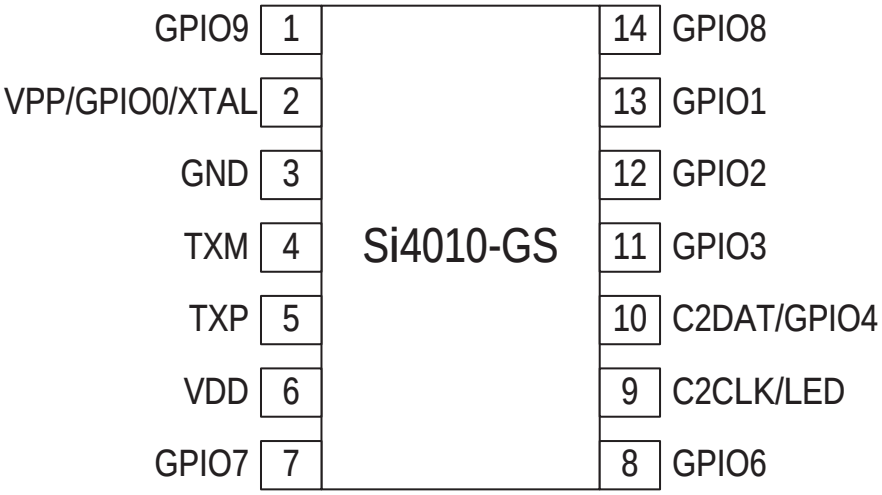
Pin Number(s)	Name	Description
1	VPP	+6.5 V required for NVM (OTP) Memory programming.
2	GND	Ground. Connect to ground plane on PCB.
3	TXM	Transmitter differential output.
4	TXP	Transmitter differential output.
5	VDD	Power.
6	C2CLK	C2 clock interface.
7	C2DAT	C2 data input/output pin.
8, 9, 10	GPIO[3:1]	General purpose input/output pins.

5.3. SOIC Package, Application



Pin Number(s)	Name	Description
1	GPIO9	General purpose input/output pin
2	GPIO0/XTAL	General purpose input pin. Can be configured as an input pin for a crystal
3	GND	Ground. Connect to ground plane on PCB
4,5	TXM, TXP	Transmitter differential outputs
6	VDD	Power
7,8	GPIO[7:6]	General purpose input/output pins
9	LED	Dedicated LED driver
10,11,12,13	GPIO[4:1]	General purpose input/output pins
14	GPIO8	General purpose input/output pin

5.4. SOIC Package, Programming/debug Mode



Pin Number(s)	Name	Description
1	GPIO9	General purpose input/output pin
2	VPP	+6.5 V required for NVM (OTP) Memory programming
3	GND	Ground. Connect to ground plane on PCB
4,5	TXM, TXP	Transmitter differential outputs
6	VDD	Power
7,8	GPIO[7:6]	General purpose input/output pins
9	C2CLK	C2 clock interface
10	C2DAT	C2 data input/output pin
11,12,13	GPIO[4:1]	General purpose input/output pins
14	GPIO8	General purpose input/output pin

6. Package Specifications

6.1. 10-Pin MSOP

Figure 6.1 illustrates the package details for the Si4010, 10-pin MSOP package. Table 6.1 lists the values for the dimensions shown in the illustration.

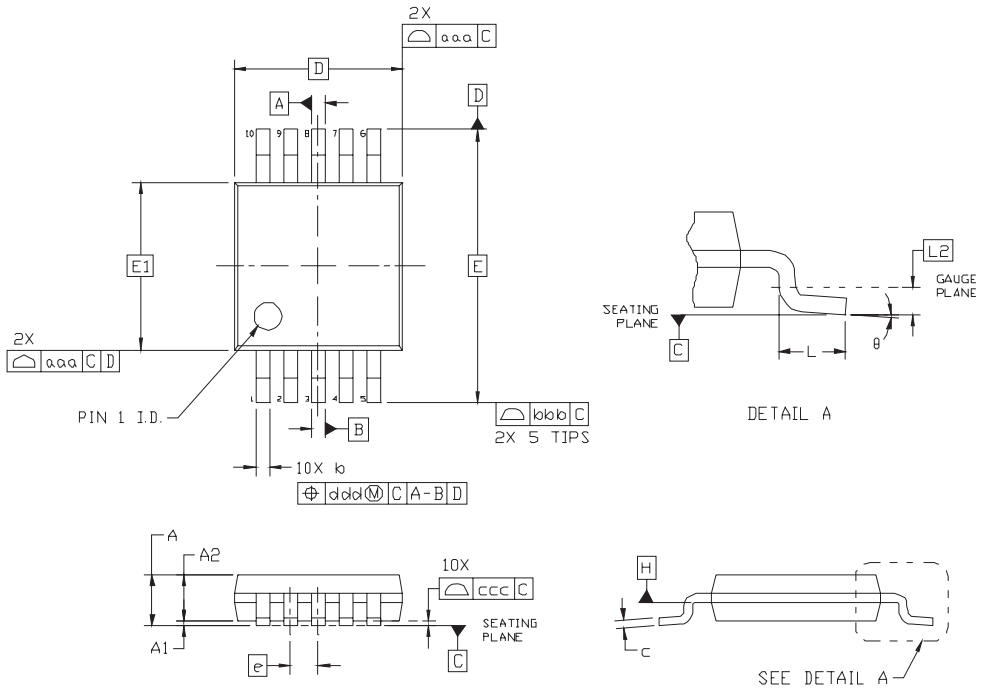


Figure 6.1. 10-Pin MSOP Package

Table 6.1. Package Dimensions

Symbol	Millimeters		
	Min	Nom	Max
A	—	—	1.10
A1	0.00	—	0.15
A2	0.75	0.85	0.95
b	0.17	—	0.33
c	0.08	—	0.23
D	3.00 BSC		
E	4.90 BSC		
E1	3.00 BSC		

Symbol	Millimeters		
	Min	Nom	Max
e	0.50 BSC		
L	0.40	0.60	0.80
L2	0.25 BSC		
q	0°	—	8°
aaa	—	—	0.20
bbb	—	—	0.25
ccc	—	—	0.10
ddd	—	—	0.08

Notes:

1. All dimensions are shown in millimeters (mm).
2. Dimensioning and tolerancing per ASME Y14.5M-1994.
3. This drawing conforms to JEDEC Outline MO-187, Variation “BA.”
4. Recommended card reflow profile is per the JEDEC/IPC J-STD-020 specification for Small Body Components.

6.2. 14-pin SOIC Package

Figure 6.2 illustrates the package details for the Si4010, 14-pin SOIC package. Table 6.2 lists the values for the dimensions shown in the illustration.

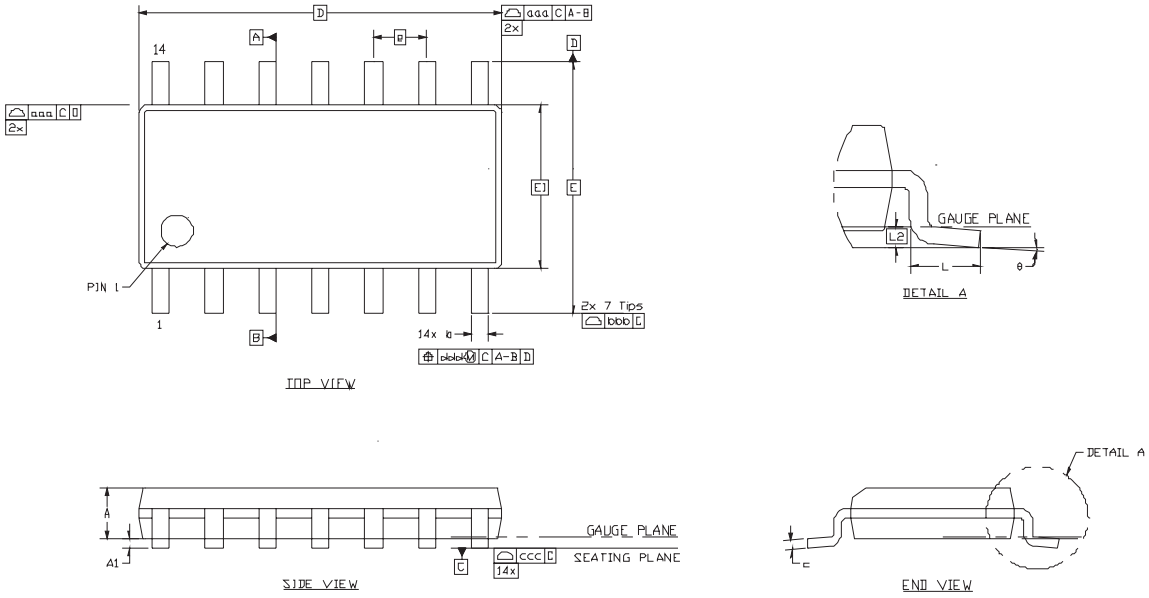


Figure 6.2. 14-Pin SOIC Package

Table 6.2. Package Dimensions

Symbol	Min	Max
A	—	1.75
A1	0.10	0.25
b	0.33	0.51
c	0.17	0.25
D	8.65 BSC	
E	6.00 BSC	
E1	3.90 BSC	
e	1.27 BSC	

Symbol	Min	Max
L	0.40	1.27
L2	0.25 BSC	
Q	0°	8°
aaa	0.10	
bbb	0.20	
ccc	0.10	
ddd	0.25	

Notes:

1. All dimensions are shown in millimeters (mm).
2. Dimensioning and tolerancing per ASME Y14.5M-1994.
3. This drawing conforms to JEDEC Outline MS012, variation AB."
4. Recommended card reflow profile is per the JEDEC/IPC J-STD-020 specification for Small Body Components.

7. PCB Land Pattern 10-Pin MSOP

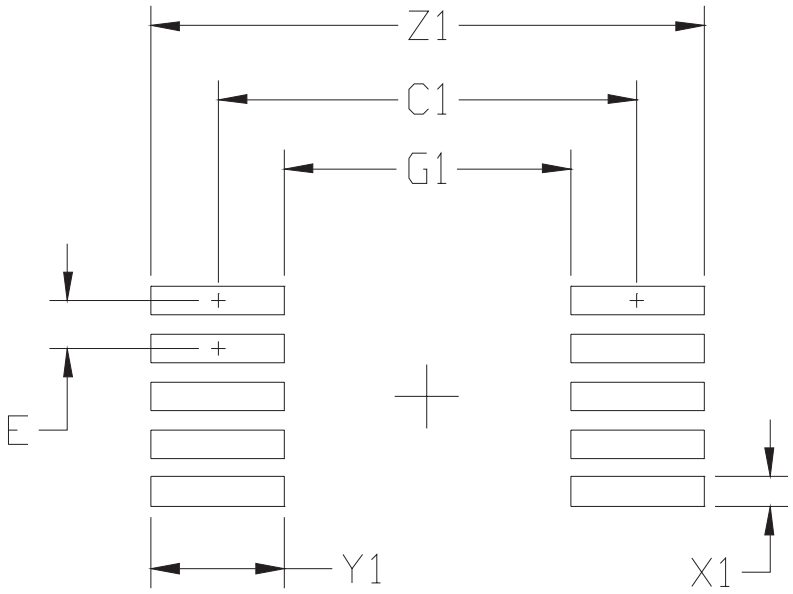


Figure 7.1. 10-Pin MSOP Recommended PCB Land Pattern

Table 7.1. 10-Pin MSOP Dimensions

Dimension	MIN	MAX
C1	4.40 REF	
E	0.50 BSC	
G1	3.00	—
X1	—	0.30
Y1	1.40 REF	
Z1	—	5.80

Notes:

General

1. All dimensions shown are in millimeters (mm) unless otherwise noted.
2. Dimensioning and Tolerancing per ASME Y14.5M-1994.
3. This Land Pattern Design is based on the IPC-7351 guidelines.
4. All dimensions shown are at Maximum Material Condition (MMC). Least Material Condition (LMC) is calculated based on a Fabrication Allowance of 0.05mm.

Solder Mask Design

1. All metal pads are to be non-solder mask defined (NSMD). Clearance between the solder mask and the metal pad is to be 60 m minimum, all the way around the pad.

Stencil Design

1. A stainless steel, laser-cut and electro-polished stencil with trapezoidal walls should be used to assure good solder paste release.
2. The stencil thickness should be 0.125mm (5 mils).
3. The ratio of stencil aperture to land pad size should be 1:1.

Card Assembly

1. A No-Clean, Type-3 solder paste is recommended.
2. The recommended card reflow profile is per the JEDEC/IPC J-STD-020 specification for Small Body Components.

8. PCB Land Pattern 14-pin SOIC Package

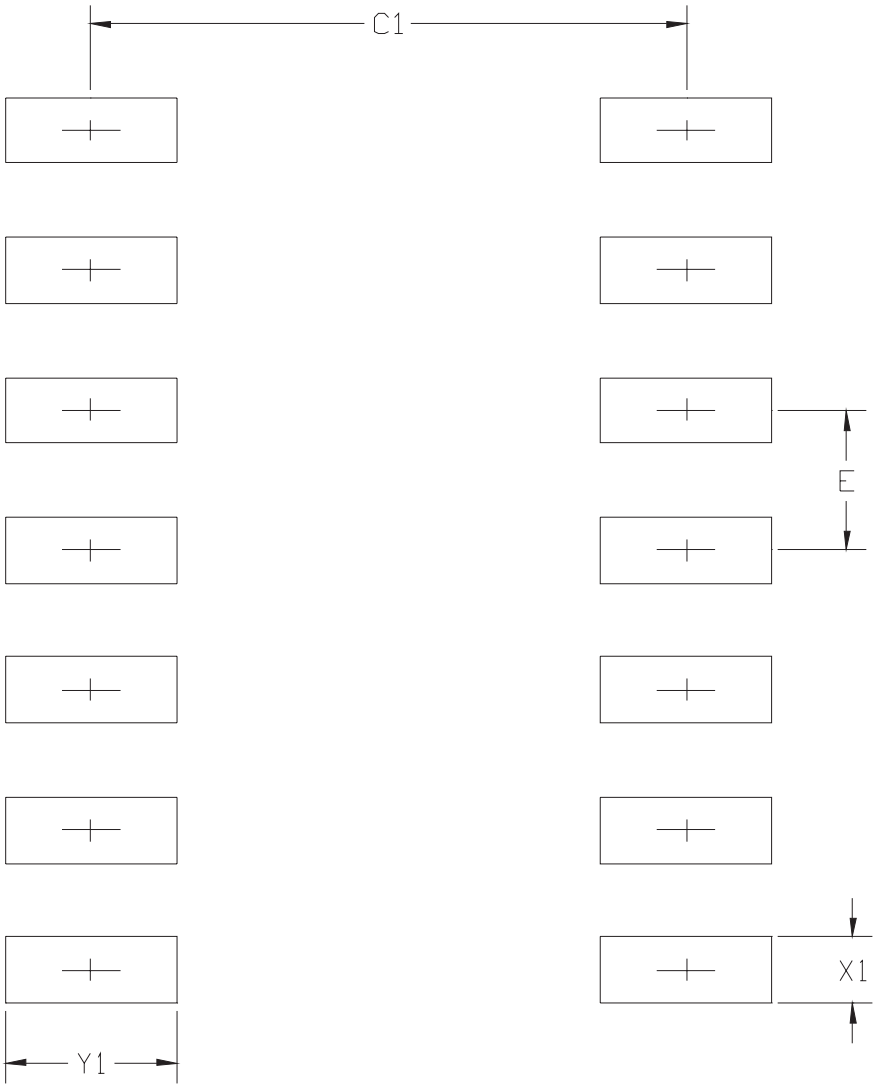


Figure 8.1. 14-Pin SOIC Recommended PCB Land Pattern

Table 8.1. PCB Land Pattern Dimensions

Dimension	MIN	MAX
C1	5.30	5.40
E	1.27 BSC	
X1	0.50	0.60
Y1	1.45	1.55

Notes:

General

1. All dimensions shown are in millimeters (mm) unless otherwise noted.
2. This land pattern design is based on the IPC-7351 guidelines.

Solder Mask Design

1. All metal pads are to be non-solder mask defined (NSMD). Clearance between the solder mask and the metal pad is to be 60 µm minimum, all the way around the pad.

Stencil Design

1. A stainless steel, laser-cut and electro-polished stencil with trapezoidal walls should be used to assure good solder paste release.
2. The stencil thickness should be 0.125 mm (5 mils).
3. The ratio of stencil aperture to land pad size should be 1:1 for all perimeter pads.

Card Assembly

1. A No-Clean, Type-3 solder paste is recommended.
2. The recommended card reflow profile is per the JEDEC/IPC J-STD-020 specification for Small Body Components.

9. Electrical Characteristics

Table 9.1. Recommended Operating Conditions

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Supply Voltage	V_{DD}		1.8	—	3.6	V
Supply Voltage Slew Rate		Initial Battery Insertion*	20	—	650	mV/us
Ambient Temperature	T_A		–40	25	85	°C
Digital Input Range		Digital Input Signals	–0.3	—	V_{DD} + 0.3	V

***Note:** Recommend bypass capacitor = 1 µF; slew rate measured 1 V < V_{DD} ,< 1.7 V.

Table 9.2. Absolute Maximum Ratings[1,2]

Parameter	Symbol	Value	Unit
Supply Voltage	V_{DD}	–0.5 to 3.9	V
Input Current[3]	I_{IN}	10	mA
Input Voltage[4]	V_{IN}	–0.3 to (V_{DD} + 0.3)	V
Junction Temperature	T_{OP}	–40 to 90	°C
Storage Temperature	T_{STG}	–55 to 125	°C

Notes:

1. Permanent device damage may occur if the absolute maximum ratings are exceeded. Functional operation should be restricted to the conditions as specified in the operational sections of this data sheet. Exposure beyond recommended operating conditions for extended periods may affect device reliability.
2. Handling and assembly of these devices should only be done at ESD-protected workstations.
3. All input pins besides V_{DD}.
4. For GPIO pins configured as inputs.

Table 9.3. DC Characteristics
(TA = 25° C, VDD = 3.3 V, RL = 550 Ω, unless otherwise noted)

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Supply Current	I_{VDD}	+10 dBm output, OOK, Manchester	—	14.2	—	mA
		+6.5 dBm output, OOK, Manchester	—	11.3	—	mA
		+10 dBm, FSK	—	19.8	—	mA
		+6.5 dBm output, FSK	—	14.1	—	mA
Sleep Timer Mode	I_{ST}	Only sleep timer is enabled	—	700	—	nA
Standby Supply Current	I_{SB}	All GPIO floating or held high	—	10	—	nA
LED Sink Current	I_{LED}	V_{OUT} > 200 mV	—	0.68	—	mA
GPIO[0-9] Pull Up Resistance	R_{PU}		48	55	62	kΩ
High Level Input Voltage[1]	V_{IH}	Trip point at 0.45 x V_{DD}		0.506 x V_{DD}		V
Low Level Input Voltage[1]	V_{IL}	Trip point at 0.45 x V_{DD}		0.42 x V_{DD}		V
High Level Input Current[1]	I_{IH}	V_{IN} = V_{DD}	—	TBD	—	µA
Low Level Input Current[1]	I_{IL}	V_{IN} = 0	—	TBD	—	µA
High Level Output Voltage[2]	V_{OH}	I_{SOURCE} = TBD	—	TBD	—	V
Low Level Output Voltage[2]	V_{OL}	I_{SINK} = TBD	—	TBD	—	V

Notes:
1. For GPIO pins configured as inputs.
2. For GPIO pins configured as outputs.

Table 9.4. Si4010 RF Transmitter Characteristics

(TA = 25° C, VDD = 3.3 V, RL = 550 Ω,, SOIC package unless otherwise noted)

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Frequency Range[1]	F_{RF}		27	—	960	MHz
Frequency Noise (rms)[2]		Allen deviation, measured across 1 ms interval	—	0.3	—	ppm
Phase Noise @ 915 MHz		10 kHz offset	—	–70	—	dBc/Hz
		100 kHz offset	—	–100	—	dBc/Hz
		1 MHz offset	—	–105	—	dBc/Hz
Frequency Tuning Time			—	5	—	ms
Selected Frequencies in Range of 27–960 MHz		Discrete frequencies	—	100	—	MHz
			—	315	—	MHz
			—	390	—	MHz
			—	433.92	—	MHz
			—	868	—	MHz
			—	915	—	MHz
Carrier Frequency Accuracy		$0°C \le T_A \le 70° C$	–150	—	+150	ppm
		$–40°C \le T_A \le 85° C$	–250	—	+250	ppm
		F_{RF} = 100 MHz $0°C \le T_A \le 70° C$	–15	—	15	kHz
		F_{RF} = 100 MHz $–40°C \le T_A \le 85° C$	–25	—	25	kHz
		F_{RF} = 315 MHz $0°C \le T_A \le 70° C$	–47.3	—	47.3	kHz
		F_{RF} = 315 MHz $–40°C \le T_A \le 85° C$	–78.8	—	78.8	kHz
		F_{RF} = 433.92 MHz $0°C \le T_A \le 70° C$	–65.1	—	65.1	kHz
		F_{RF} = 433.92 MHz $–40°C \le T_A \le 85° C$	–108	—	108	kHz
		F_{RF} = 868 MHz $0°C \le T_A \le 70° C$	–130	—	130	kHz
		F_{RF} = 868 MHz $–40°C \le T_A \le 85° C$	–217	—	217	kHz
		F_{RF} = 915 MHz $0°C \le T_A \le 70° C$	–137	—	137	kHz
		F_{RF} = 915 MHz $–40°C \le T_A \le 85° C$	–229	—	229	kHz

Notes:

1. The frequency range is continuous over the specified range.
2. The frequency step size is limited by the frequency noise.
3. Optimum differential load is equal to 4 V/(11.5mA/2 * 4/PI) = 550 Ω. Therefore the antenna load resistance in parallel with the Si4010 differential output resistance should equal 50 Ω.
4. Total NVM copy time = 2 ms + (NVM copy Boot Time per kB) x (NVM data in kB).

Table 9.4. Si4010 RF Transmitter Characteristics(Continued)
(TA = 25° C, VDD = 3.3 V, RL = 550 Ω,, SOIC package unless otherwise noted)

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Frequency Error Contribution with External Crystal			–10	—	+10	ppm
Transmit Power[3]		Maximum programmed Tx power, with optimum differential load, Vdd > 2.2 V	—	10	—	dBm
		Minimum programmed TX power, with optimum differential load, V_{DD} > 2.2 V	—	–13	—	dBm
		Power variation vs temp and supply, with optimum differential load, V_{DD} > 2.2 V	–1.0	—	0.5	dB
		Power variation vs temp and supply, with optimum differential load, V_{DD} > 1.8 V	–2.5	—	0.5	dB
		Transmit power step size from –13 to 10 dBm	—	0.25	—	dB
PA Edge Ramp Rate Programmable Range		OOK mode	0.34	—	10.7	us
Data Rate		OOK	0.1	—	50	kBaud
		FSK	0.1	—	100	kBaud

Notes:
1. The frequency range is continuous over the specified range.
2. The frequency step size is limited by the frequency noise.
3. Optimum differential load is equal to 4 V/(11.5mA/2 * 4/PI) = 550 Ω. Therefore the antenna load resistance in parallel with the Si4010 differential output resistance should equal 50 Ω.
4. Total NVM copy time = 2 ms + (NVM copy Boot Time per kB) x (NVM data in kB).

Table 9.4. Si4010 RF Transmitter Characteristics(Continued)
(TA = 25° C, VDD = 3.3 V, RL = 550 Ω,, SOIC package unless otherwise noted)

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
FSK Deviation		Max frequency deviation	—	300	—	ppm
		Deviation resolution	—	2	—	ppm
		Deviation accuracy		TBD		ppm
		Max frequency deviation, 100 MHz	—	30	—	kHz
		Deviation resolution, 100 MHz	—	200	—	Hz
		Max frequency deviation, 315 MHz	—	95	—	kHz
		Deviation resolution, 315 MHz	—	630	—	Hz
		Max frequency deviation, 433.92 MHz	—	130	—	kHz
		Deviation resolution, 433.92 MHz	—	868	—	Hz
		Max frequency deviation, 868 MHz	—	260	—	kHz
		Deviation resolution, 868 MHz	—	1740	—	Hz
		Max frequency deviation, 915 MHz	—	275	—	kHz
		Deviation resolution, 915 MHz	—	1830	—	Hz
OOK Modulation depth			60	—	—	dB
Antenna Tuning Capacitive Range (Differential)		315 MHz	2.4	—	12.5	pF
NVM Copy Boot Time per kB[4]			—	3.6	—	ms/ kB

Notes:
1. The frequency range is continuous over the specified range.
2. The frequency step size is limited by the frequency noise.
3. Optimum differential load is equal to 4 V/(11.5mA/2 * 4/PI) = 550 Ω. Therefore the antenna load resistance in parallel with the Si4010 differential output resistance should equal 50 Ω.
4. Total NVM copy time = 2 ms + (NVM copy Boot Time per kB) x (NVM data in kB).

Table 9.5. Low Battery Detector Characteristics

(TA = 25° C, VDD = 3.3 V, RL = 550 Ω, unless otherwise noted)

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Battery Voltage Measurement Accuracy			—	2	—	%

Table 9.6. Optional Crystal Oscillator Characteristics

(TA = 25° C, VDD = 3.3 V, RL = 600 Ω, unless otherwise noted)

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
Crystal Frequency Range		GPI0 configured as crystal oscillator	10	—	13	MHz
Input Capacitance (GPIO0)		GPI0 configured as crystal oscillator	—	5	—	pF
Crystal ESR		GPI0 configured as crystal oscillator	—	—	50	Ω
Start-up Time		Crystal oscillator only, 60 mH motional arm inductance	—	9	—	ms

Table 9.7. EEPROM Characteristics

Parameter	Conditions	Min	Typ	Max	Units
Program Time	Independent of number of bits changing values	—	8	40	ms
Maximum Count per Counter	Using API		1000000		cycles
Write Endurance (per bit)*		50000	—	—	cycles
Note: *API uses coding technique to achieve write endurance of 1M cycles per bit.					

Table 9.8. Low Power Oscillator Characteristics

V_{DD} = 1.8 to 3.6 V; T_A = –40 to +85 °C unless otherwise specified. Use factory-calibrated settings.

Parameter	Conditions	Min	Typ	Max	Units
Programmable Frequency Range	Programmable divider in powers of 2 up to 128	.1875	—	24	MHz
Frequency Accuracy		–1	—	+1	%

Table 9.9. Sleep Timer Characteristics

V_{DD} = 1.8 to 3.6 V; T_A = –40 to +85 °C unless otherwise specified. Use factory-calibrated settings.

Parameter	Conditions	Min	Typ	Max	Units
Maximum Programmable Time		—	—	6800	s
Time Accuracy	Using API to program timer	–1.5	—	1.5	%

10. System Description

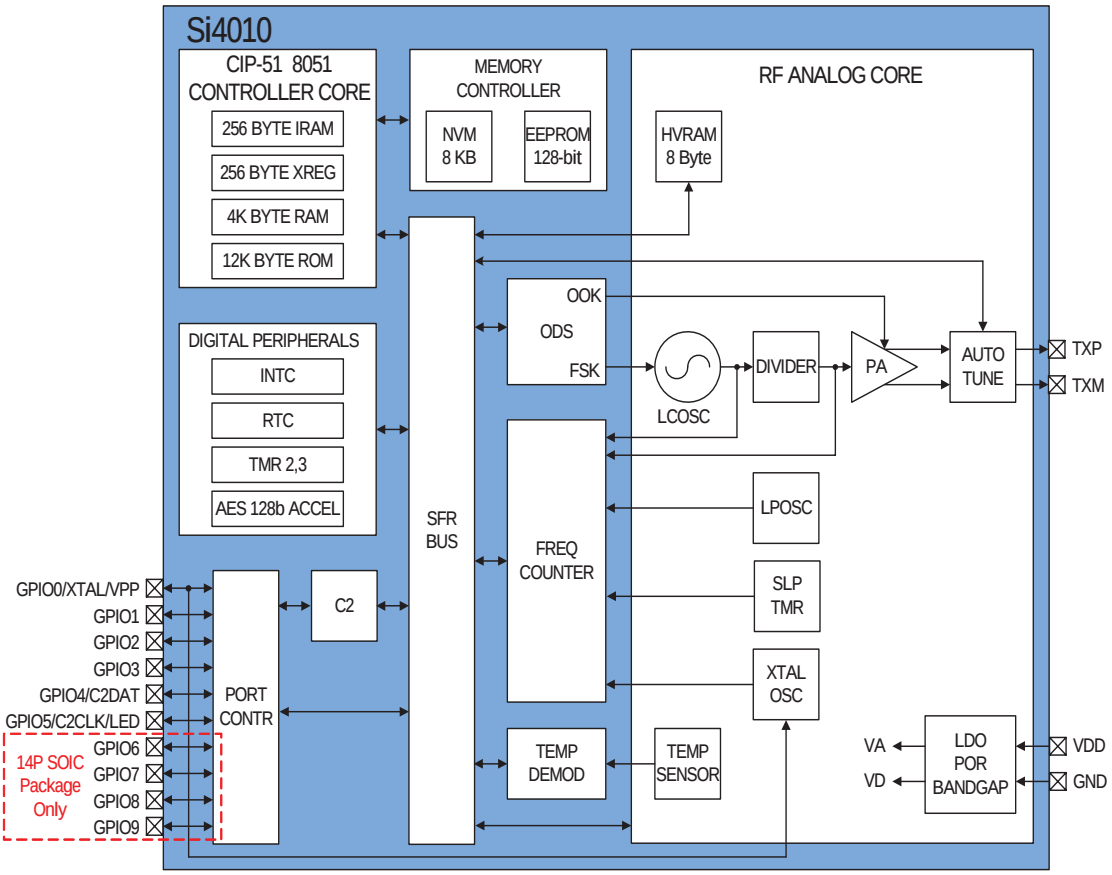


Figure 10.1. Functional Block Diagram

10.1. Overview

The Si4010 is a fully integrated crystal-less CMOS SoC RF transmitter with an embedded CIP-51 8051 MCU as the core processor of the system. The device is designed for low power battery applications with standby currents of less than 10 nA to optimize battery life. Upon power up, the device immediately enters standby mode. In this mode, all blocks are powered down except for the low leakage high-voltage RAM (HVRAM) which provides 8 bytes of memory that retains its state as long as the battery voltage is applied and above 1.8 V. The Si4010 is awakened from standby mode by a falling edge to ground on any one of the GPIO pins. In addition, the Si4010 has a low-power sleep timer for applications where the device is required to wake up and periodically check for events instead of being wakened by a GPIO falling edge.

Upon wake up, the boot loader copies data from the one time programmable (OTP) NVM to CODE/XDATA RAM (4 kB) because the MCU can only operate with programs stored in RAM or ROM. The copy process occurs on each wake-up event and requires approximately 2 ms of fixed time plus 3.6 ms per kB of data or 16.4 ms to fill the full 4 kB of CODE/XDATA RAM. After the NVM boot copy process is completed, the MCU runs the user program in RAM and can also run functions from ROM that are called by the user program such as button service routines to facilitate button debouncing, button time stamps, etc. A complete list of all the API functions is given in Section 10.3 and a detailed description is given in application note “AN370: Si4010 Software Programming Guide.”

The Si4010 has three timing sources. The LCOSC is the most accurate timing source native to the chip. Each device is factory trimmed and programmed at Silicon Labs to produce a frequency accuracy of better than ±150 ppm over the temperature range of 0 to + 70 °C and ±250 ppm over the industrial range of –40 to +85 °C. The LCOSC is fitted to a multiple-degree polynomial to compensate for temperature variations both from the on-chip power amplifier (PA) and also from the external environment. This LCOSC oscillates around 3.9 GHz and provides the clock (via the DIVIDER) used to modulate the PA for OOK and FSK transmission. The low power oscillator (LPOSC) is the second timing source and operates at 24 MHz. The LPOSC is always the source of clocking for the MCU and is turned off only in standby mode. The system clock is programmable allowing the MCU to operate with lower clock frequencies while waiting between packets to save power. The RTC and timers 2 and 3 are derived from the LPOSC. The last clock source is the crystal oscillator (XTALOSC). This crystal oscillator is unused in many customer applications and used only when a highly accurate carrier frequency is desired. When enabled, it is used before the beginning of a transmission to correct the frequency of the LCOSC and is then shutdown to save power. An internal frequency counter is implemented in hardware to allow for quick frequency ratio measurements to calibrate the different clock sources.

The high efficiency PA is a CMOS open drain output driver capable of producing 4 Vpk differential output swing with a supply voltage of 2.2 V or higher. The PA output has 2.4 to 12.5 pF of differential variable capacitance that is automatically adjusted to resonate the antenna at the start of each packet transmission. This automatic adjustment is realized with a firmware algorithm in the ROM and some additional hardware in the PA. Maximum power can be transferred to the inductive antenna load when the antenna and output driver are at resonance and the real component of the load is equal to the optimum load resistance of $V_{pk}/(4/Pi * I_{tail}/2)$ where V_{pk} is the peak differential voltage and I_{tail} is the tail current of the PA. At higher resistances the PA is voltage limited and at lower resistances the PA is current limited. The PA tail current is programmable from 810 uA up to 7.67 mA in 0.25 dB steps and there is a boost current bit that multiplies the tail current by 1.5 times allowing it to go up to 11.5 mA. With an antenna load resistance of about 550 Ω an output power of +10 dBm is achievable. Edge rate control is also included for OOK mode to reduce harmonics that may otherwise violate government regulations.

The on-chip temperature sensor (TEMP SENSOR) measures the internal temperature of the chip and temperature demodulator (TEMP DEMOD) converts the TEMP SENSORs' output into a binary number representing temperature and is used to compensate the frequency of the LCOSC when the temperature changes. Each device is frequency and temperature calibrated in the factory.

The output data serializer (ODS) is responsible for synchronizing the output data to the required data rate and maintaining a steady data flow when data is available. This block produces the edge rate control for the PA in OOK mode and the frequency deviation in FSK mode. The block also schedules the power on/off times of the LCOSC, DIVIDER, and PA to conserve battery power during transmission.

Power management is provided on chip with low-drop-out (LDO) regulators for the internal analog and digital supplies, VA and VD, respectively. The power-on reset (POR) circuit monitors the power applied to the chip and generates a reset signal to set the chip into a known state. The bandgap produces voltage and current references for the analog blocks in the chip and can be shut down when the analog blocks are not used.

The embedded CIP-51 8051 MCU provides the core functionality of the Si4010. User software has complete control of all peripherals, and may individually shut down any or all peripherals for power savings. 8K bytes of on-chip one-time programmable NVM memory is available to store the user program and can also store unique transmit IDs. 128 bits of EEPROM is available for counter or other operations providing non-volatile storage capability in case of power outages due to battery removal. A library of useful software functions such as AES encryption, a patented 32-bit counter providing 1M cycles of read/write endurance, and many other functions are included in the 12 kB of ROM to reduce user design time and code space. General purpose input/output pins with push button wake-on touch capability are available to further reduce current consumption.

The Si4010 includes Silicon Laboratories' 2-wire C2 Debug and Programming interface. This debug logic supports inspection memory, viewing and modification of special function registers (SFR), setting break points, single stepping, and run and halt commands. All analog and digital peripherals are fully functional while debugging using C2. The two C2 interface pins can be shared with user functions, allowing in-system debugging without occupying package pins.

10.2. Setting Basic Si4010 Transmit Parameters

The basic transmit parameters such as output power, modulation type, data rate, and operating frequency are set by using applications programming interface (API) function commands. When using these functions certain parameters are determined by using a calculator spread sheet. The Si4010 development kit (part number 4010-DKKF 434) includes a calculator spread sheet that helps developers set the API function arguments to meet their desired design requirements. A summary of the calculator operations are given below and more detailed descriptions are given in the individual sections of this data sheet and in AN370: Si4010 Software Programming Guide.

10.2.1. Package Type

The Si4010 has two package type options: 10-pin MSOP or 14-pin SOIC. The customer should choose the package type they are using to properly model the Si4010 RF behavior.

10.2.2. Output Power

The output power of the Si4010 depends on many parameters including the antenna impedance, the output impedance of the PA, the nominal varactor setting, the battery supply voltage, and the bias current of the PA. The calculator spreadsheet can calculate the required antenna impedance needed to achieve the desired output power or it can estimate the output power given the antenna impedance. It has the following input parameters:

Power Setup:

- Power Target (dBm): This is the desired output power in dBm. The spreadsheet will always try and hit this target.
- Choose One of the Following: Maximize Radiated Power or Minimize PA current while Maximizing Radiated Power. If only radiated power is to be maximized, the PA current is maximized and an antenna impedance is found that maximizes the possible radiated power. Usually, this tends to minimize the antenna impedance relative to the chip impedance. If the PA current is to be minimized while still maximizing radiated power, the solution tends to equalize the antenna and on-chip impedances. This increases the effective impedance of the system, which saves PA current at the expense of radiation efficiency (as more power will now be consumed on-chip).
- Frequency (MHz): The RF frequency of operation, range is 27 to 960 MHz.
- Nominal Cap Word: This is the nominal setting of the power amplifier varactor that is part of the antenna tuning circuit, range is 0 to 511.
- External Diff Cap (pF): This is an external capacitor placed across the TXP and TXM pins. Assuming this has a much larger quality factor than the on-chip varactor, there may be antenna efficiency advantages of using this external component.
- Q-Factor External Cap: This is the quality factor of the external capacitor. Typical values would be 250-300.

Antenna Setup:

- Alpha (bLevel/deg C): The sensitivity of the antenna resistance vs temperature change. If constant radiated power vs temperature is desired, this constant may be used to compensate the PA drive strength. See the API section on power control.
- Approx Efficiency (%): The approximate antenna efficiency used to estimate radiated power.

- Manual Impedance Entry: Determines if the antenna impedance is calculated to meet a desired output power or if the antenna impedance is entered and the spread sheet calculates the resulting impedance. The current drive is adjusted to meet the power target (if possible).
- Antenna Real(Z) (Ohms): The antenna resistance at the operating frequency.
- Antenna Imag(Z) (Ohms): The antenna reactance at the operating frequency.

These parameters are discussed in more detail in the Power Amplifier section of the data sheet. Based on these input parameters the calculator will provide the following outputs:

PA Design Values:

- Iout Target (mA): Theoretical output current that meets the power target.
- Attenuation Factor: Theoretical attenuation factor due to losses from the chip
- Actual Iout (mA): The actual output current delivered to the antenna that accounts for quantization effects and chip losses.
- Rdif at PA (Ohms): Theoretical optimum differential load resistance that includes chip, antenna, and external capacitance loading.
- Total Power (dBm): The estimated output power based on all loss mechanisms.
- Max Diff Vpk at PA (V): The calculated peak differential voltage swing.

Antenna Targets:

- Real_Z (Ohms): The required resistance of the antenna at the frequency of operation to meet the desired output power.
- Imag_Z (Ohms): The required reactance of the antenna at the frequency of operation to meet the desired output power.
- Power dissipated in Antenna (dBm): The expected power delivered to the antenna.
- Expected Radiated Power (dBm): The expected radiated power of the device given the antenna efficiency

Chip Impedance:

- Total Diff Cap due to Chip + External Load (pF): The equivalent differential capacitance seen looking into the package pins. It includes the on-chip varactor, the package and external differential capacitor (if used).
- Real_Z (Ohms): The resistance of the chip at the frequency of operation to meet the desired output power
- Imag_Z (Ohms): The reactance of the chip at the frequency of operation to meet the desired output power

API: PA Setup:

- bMaxDrv—value for this API parameter
- bLevel—value for this API parameter
- wCap—value for this API parameter
- fAlpha—value for this API parameter
- fBeta—value for this API parameter. The sensitivity of the antenna resistance vs capacitance change. If constant radiated power vs tuning capacitance change is desired this constant may be used to compensate the PA drive strength. See the API section on power control. The algorithm attempts to keep the PA output voltage multiplied by the PA capacitance constant due to fluctuations in the external component values of the loop antenna.

10.2.3. Modulation, Encoding, and Data Rate

The output data serializer (ODS) API function commands set the modulation type, encoding method, and data rate of the transmitter. The calculator has the following inputs:

Serializer Setup:

- Bit (or Data) Rate (Kbits/s): This is the bit or data rate of the transmitter
- Encoding: The encoding methods supported are Manchester, NRZ+4b/5b, and NRZ encoding.
- Modulation: OOK or FSK.
- FSK Deviation (kHz): This is the FSK frequency deviation of the carrier frequency in response to a data signal.
- Manual Ramp Rate Entry: If Yes, used to set the ramp rate for turning on and off the PA, otherwise the ramp rate will be automatically calculated
- Target Ramp Rate (us): This parameter is the target ramp rate. Only used if the Manual Ramp Rate Entry is Yes

The outputs of the calculator are the following:

Serializer Control:

- Ramp Time (µs): The actual ramp time of turning on and off the PA. If Manual Rate Entry is Yes, this will represent the closest possible match to the user entry. If Manual Rate Entry is No, this is automatically calculated based on the target bit rate. The chosen rate insures the resulting spectrum will be FCC/ETSI compliant.
- Actual Symbol Rate (Ksym/s): The actual symbol rate produced by the chip after taking into account encoding and quantization effects due to the timers.
- Ramp Time/Symbol Rate: The ratio of the ramp time divided by the symbol rate.

API: ODS Setup:

- bModulation Type—value for this API parameter
- bClkDiv—value for this API parameter
- bEdgeRate—value for this API parameter
- bGroupWidth—value for this API parameter
- wBitRate—value for this API parameter
- bDivWarmInt, bLcWarmInt, and bPaWarmInt—value for this API parameter

API: FSK Controls:

- biFskDev—value for this API parameter
- Expected FSK Deviation (kHz): The expected FSK deviation with quantization error

10.2.4. Output Frequency

The output frequency does not require the use of the calculator and is set by using the following API commands:

- vFCast_Setup()
- vFCast_Tune(desired frequency)

10.2.5. Battery Life Calculation

The calculator also estimates battery life of a system given the packet setup and number of button pushes per day. The inputs to the calculator are all of the above inputs plus the following:

Packet Setup:

- Number of bits in Packet—Number of bits in the packet excluding the preamble bits
- Preamble bits—Number of bits in the preamble
- Time Prior to Transmit (ms)—The time required to boot the chip and send a packet out
- Number of Packets—The number of packets sent out per button press
- Time Between Packets (ms)—The time between repeating packets
- Button Pushes/Day—The number of button pushes per day

The outputs of the calculator are the following:

Battery Life:

- Avg Transmit Current (mA)—The average transmit current
- Peak Transmit Current (mA)—The peak transmit current
- Charge/Year (mAH)—The charge per year in mAH
- 220 mAH Battery Life (Years)—The estimated battery life of a 220 mAH battery

10.3. Applications Programming Interface (API) Commands

The following is a list of API commands for the Si4010. For detailed descriptions of the API commands see the application note AN370: Si4010 Software Programming Guide.

AES Module Functions:

```
vAes_Cipher
vAes_InvGenKey
vAes_InvCipher
```

Button Service Module Functions:

```
vBsr_Setup
wBsr_Pop
wBsr_GetCurrentButton
vBsr_InitPts
bBsr_GetPtsItemCnt
vBsr_Service
bBsr_GetTimestamp
```

Demodulator Temperature Sensor Module Functions:

```
vDmdTs_Setup
iDmdTs_GetData
iDmdTs_GetLatestDmdSample
iDmdTs_GetLatestTemp
vDmdTs_ClearDmd
vDmdTs_ClearDmdIntFlag
vDmdTs_IsrCall
bDmdTs_GetSamplesTaken
vDmdTs_Enable
vDmdTs_RunForTemp
vDmdTs_ResetCounts
```

Encoding Module Functions:

```
vEnc_4b5bEncode
vEnc_Set4b5bLastBit
bEnc_ManchesterEncode
```

Frequency Counter Module Functions:

```
vFc_Setup
vFc_StartCount
vFc_PollDone
lFc_GetCount
lFc_StartPollGetCount
```

Frequency Casting Module Functions:

```
vFCast_Setup
vFCast_XoSetup
vFCast_Tune
vFCast_FineTune
vFCast_FskAdj
```

HVRAM Module Functions:

```
vHvram_Write
bHvram_Read
```

Multi-Time Programmable Module Functions:

```
lMtp_GetDecCount
vMtp_IncCount
vMtp_SetDecCount
bMtp_Write
vMtp_Strobe
pbMtp_Read
```

Battery Measurement Module Functions:

```
iMVdd_Measure
```

Non-Volatile Memory Copy Module Functions:

```
vNvm_SetAddr
wNvm_GetAddr
bNvm_CopyBlock
vNvm_McEnableRead
vNvm_McDisableRead
```

Output Data Serializer Module Functions:

```
vOds_Setup
vOds_Enable
vOds_WriteData
```

Power Amplifier Module Functions:

```
vPa_Setup
vPa_Tune
```

Single Transmission Loop Module Functions:

```
vStl_EncodeSetup
vStl_EncodeByte
vStl_PreLoop
vStl_SingleTxLoop
vStl_PostLoop
```

System Module Functions:

```
vSys_Setup
vSys_BandGapLdo
vSys_ForceLc
wSys_GetRomId
wSys_GetChipId
bSys_GetRevId
lSys_GetProdId
wSys_GetKeilVer
vSys_SetClkSys
lSys_GetMasterTime
vSys_IncMasterTime
vSys_SetMasterTime
vSys_LedIntensity
vSys_LpOscAdj
vSys_Shutdown
bSys_GetBootStatus
vSys_FirstPowerUp
vSys_16BitDecLoop
vSys_8BitDecLoop
```

Sleep Timer Module Functions:

```
lSleepTim_GetCount
vSleepTim_SetCount
bSleepTim_CheckDutyCycle
vSleepTim_AddTxTimeToCounter
lSleepTim_GetOneHourValue
```

11. Power Amplifier

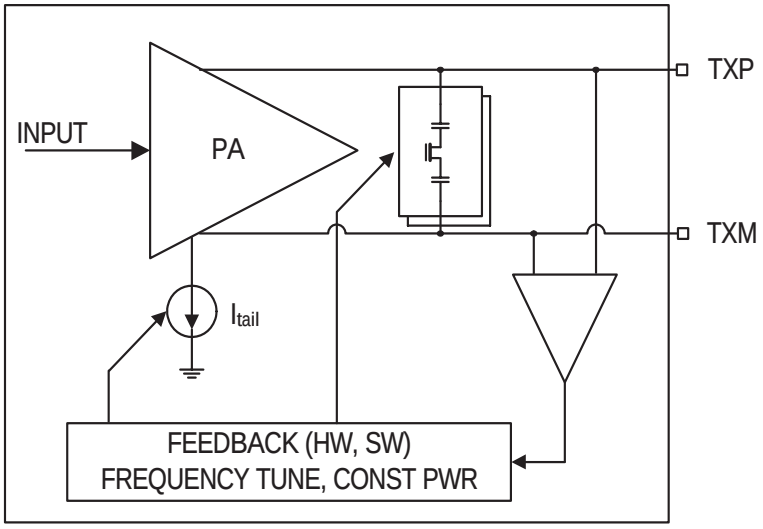


Figure 11.1. Simplified PA Block Diagram

The CMOS power amplifier (PA) is a differential open drain amplifier capable of delivering +10 dBm of output power. Maximum power can be transferred to an inductive antenna load when the antenna and output driver of the PA are at resonance and the real component of the combined load is equal to the optimum load resistance of $V_{pk}/(4/Pi \times I_{tail}/2)$ where V_{pk} is the peak differential voltage of the PA and I_{tail} is the tail current of the PA. This optimum load resistance is the parallel combination of the PA output resistance and the differential antenna resistance. At higher resistances the PA is voltage limited and at lower resistances the PA is current limited. The PA tail current is programmable from 810 µA up to 7.67 mA (SFR register PA_LVL) in 0.25 dB steps and there is a boost current bit (XREG PA_TRIM.PA_MAX_DRV) that multiplies the tail current by 1.5 times allowing it to go up to 11.5mA. The maximum differential peak-to-peak voltage is 4 V when the supply is 2.2 to 3.6 V and drops linearly down to 3.4V when the supply is at 1.8V

The calculator spreadsheet tool computes the required antenna impedance and API settings to achieve the user desired output power. Proper layout and matching techniques are all necessary to ensure optimal performance. Figure 9.1 shows a typical application schematic of the Si4010 for a differential loop antenna. Application note "AN369: Antenna Interface for the Si401x Transmitters" provides detailed information about designing the antenna interface for the Si401X transmitters. With proper filtering and layout techniques, the Si4010 can conform to US FCC part 15.231 and European EN 300 220 regulations. Edge rate control is also included for OOK mode to reduce harmonics that may otherwise violate government regulations. Edge shaping is accomplished by gradually turning on and off the driver transistors of the PA. The edge shaping parameters are controlled by the ODS block and is automatically determined by the calculator spread sheet based on the desired data rate and encoding method. Users must comply with local radio frequency transmission regulations.

Off-chip capacitor tolerances, loop antenna manufacturing tolerances, and environmental variations can lead to impedance mismatch at the PA output causing reduced radiated power level. The Si4010 includes an automatic antenna tuning circuit to reduce the mismatch by adjusting the on-chip variable capacitor to resonate with the inductance of the antenna. The PA output has 2.4 to 12.5 pF of variable capacitance that is adjusted to tune the antenna to the correct frequency using a firmware assisted algorithm and on-chip hardware.The variable capacitance is adjusted at the start of each packet transmission during the preamble. The switching network in the capacitor array is compensated over process, voltage, and temperature

(PVT) to keep its quality factor (Q) nearly constant at 50 (at 434 MHz). The starting value of the 9-bit capacitor word (XREG PA_CAP) is chosen with the help of the calculator spreadsheet. In general, a high operating frequency requires a smaller capacitance and hence a low value capacitive word. The output resistance of the PA is a strong function of the capacitive word because the variable capacitor is implemented with a capacitor and a MOS switch. When more capacitance is turned on (higher capacitive word), more switches turn on and with a constant Q design, the output resistance of the PA decreases and has more loss. Thus another consideration for the nominal capacitive word besides the operating frequency is how the resistive loading of the varactor affects the optimum load resistance and the required antenna resistance. The calculator illustrates how the nominal value of the capacitive word affects the desired antenna resistance.

In addition to the algorithm used to tune the antenna for resonance, a software control loop using the Power Amplifier Module API can keep the transmit radiated power constant due to changes in temperature and/or capacitance of the antenna. For example, if changes in the temperature of the transmitter and/or the capacitance of the antenna cause the impedance of the load (the parallel combination of the PA and antenna resistances) to decrease, this will cause a decrease in the output voltage of the PA and hence the radiated power. Both the operating temperature and the capacitor tuning word are monitored by the chip and may be used to increase the nominal drive current to bring the product of the output voltage and driver capacitance back to what it was prior to the environmental change. In order for this loop to operate correctly, the parameters Alpha and Beta need to be determined from measured antenna characteristics. Alpha represents the required change in bLevel (the nominal power level programmed through the API interface) given changes in temperature. Beta represents the required change in bLevel given changes in programmed driver capacitance. Remember that each LSB change in bLevel corresponds to a 0.25 dBm change in power. For example, if experimental measurement shows that the radiated power changes by 1 dBm over a 50 °C change in temperature, alpha would be set to 4/50=0.08. In this alpha equation, the 4 is derived from 1 dBm/0.25 dBm per step in bLevel. Thus, the units of alpha are (LSB steps in bLevel)/(change in temp). Beta can be measured by forcing the external antenna capacitance to change by some small amount and measuring the corresponding change in tuning capacitance and radiated power. For example, if the antenna capacitor is changed by 5%, it is seen that the resulting capacitor word changed by 30 LSBs and the power decreased by 2dBm. In this case, Beta would be calculated as 8/30=0.27 and has the units of (LSB steps in bLevel)/(LSB steps in capacitor word). These two parameters can be measured and entered as parameters to the API to provide accurate adjustments to the radiated power. In addition to these parameters, the differential peak voltage and current drive of the PA should not be maximized prior to using this loop so adjustments in the current drive, which affects the differential peak voltage, can be made by the feedback loop. If either the current or voltage is maximized prior to using the loop, the loop would not be able to further adjust the current or voltage and hence fail to operate properly.

11.1. Register Description

SFR Definition 11.1. PA_LVL

Bit	7	6	5	4	3	2	1	0
Name	PA_LVL_NSLICE[4:0]					PA_LVL_BIAS[2:0]		
Type	R/W					R/W		
Reset	0					0		

SFR Address = 0xCE

Bit	Name	Function
7:3	PA_LVL_ NSLICE [4:0]	**Number of Slices Enabled in the PA Driver.** +-This parameter determines the output current drive of the PA. The values entered into this register come from the Power Amplifier Module API.
2:0	PA_LVL_ BIAS [2:0]	**PA Level Bias.** This parameter determines the bias current per slice of the PA. The values entered into this register come from the Power Amplifier Module API.

XREG Definition 11.2. wPA_CAP

Bit	8	7	6	5	4	3	2	1	0
Name	PA_CAP[8:0]								
Type	R/W								
Reset	0	0	0	0	0	0	0	0	0

XREG Address = 0x400C

Bit	Name	Function
8:0	PA_CAP [8:0]	**PA Variable Capacitance.** Linear control of the output capacitance of the PA. Range: 2.4–-12.5 pF (not exact values). The resonance frequency and impedance matching between the PA output and the connected antenna can be tuned by changing this value. This register is set by the Power Amplifier Module API.

XREG Definition 11.3. bPA_TRIM

Bit	7	6	5	4	3	2	1	0
Name				PA_MAX_DRV	Reserved	Reserved	Reserved	Reserved
Type				R/W				
Reset				0				

XREG Address = 0x4012

Bit	Name	Function
7:5	Unused	
4	PA_MAX_DRV	**PA MAX Drive Bit.** This parameter boost the bias current of the PA by 1.5 times up to 10.5 mA. The values entered into this register come from the Power Amplifier Module API. This bit should be set without changing the other bits.
3:0	Reserved	**Reserved.**

12. Output Data Serializer (ODS)

12.1. Description

The ODS block is responsible for synchronizing the output data to the required data rate and maintaining a steady data flow during transmission. The serializer accomplishes the following functions:

- Controls the edge rate of the PA on/off transitions.
- Schedules PA, DIVIDER, LCOSC on/off power transitions for minimal power consumption.
- Controls the serial data rate.
- Provides handshake interface and a 1 byte pipeline to allow a software process to maintain steady dataflow.
- Modulates a 7 bit “frequency deviation” bus to the LC oscillator to allow for FSK operation.
- Provides test features to force on the power state of the LCOSC, DIVIDER, and PA; recirculating a fixed pattern; forcing the FSK offset frequency.

The SFR and XREG settings of this block are determined from the desired modulation, data rate, and encoding method and are automatically set by the ODS API in conjunction with the calculator. Users are recommended to use the ODS API module functions for setting these registers.

12.2. Timing

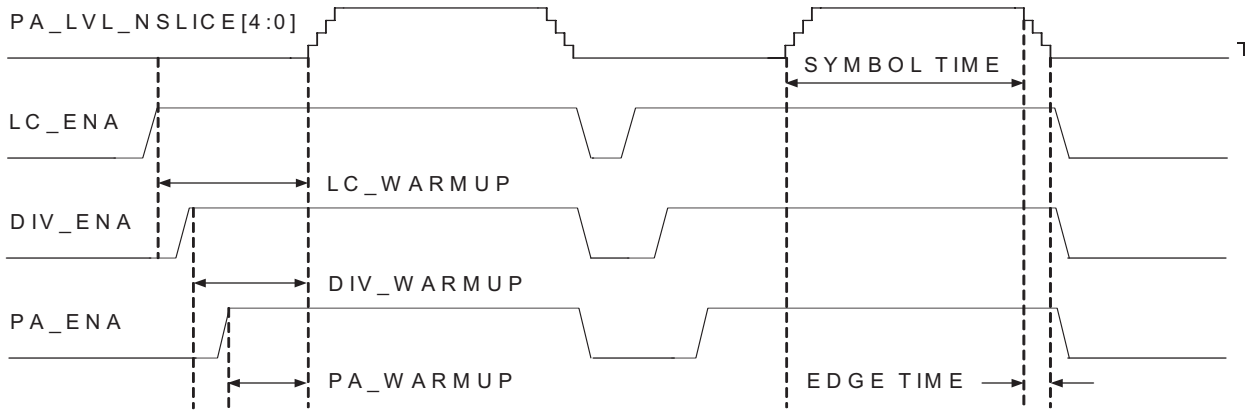


Figure 12.1. OOK Timing Example

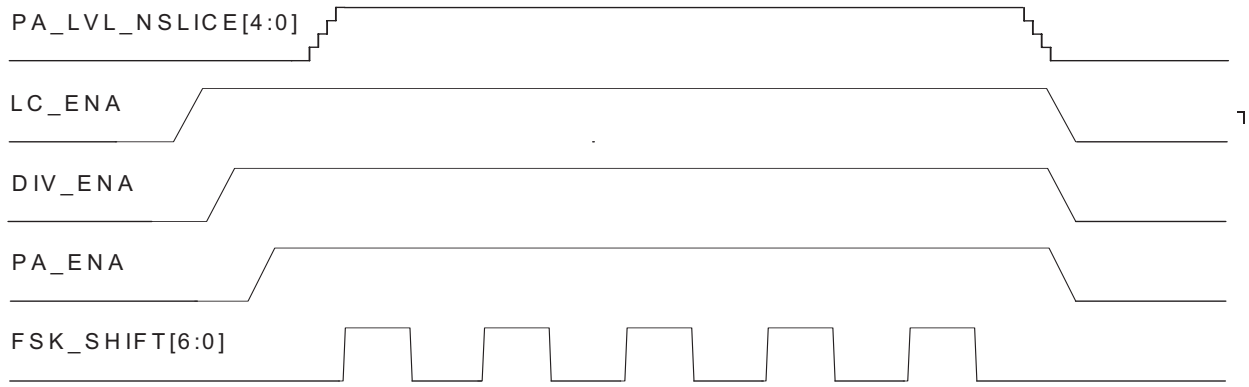


Figure 12.2. FSK Timing Example

12.3. Register Description

SFR Definition 12.1. ODS_CTRL

Bit	7	6	5	4	3	2	1	0
Name	ODS_SHIFT_CTRL [1:0]		FSK_FORCE_DEV	FSK_MODE	FORCE_LC	FORCE_DIV	FORCE_PA	ODS_EN
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xA9

Bit	Name	Function
7:6	ODS_SHIFT_CTRL[1:0]	**ODS Output Control on Last Bit.** Controls behavior of serializer when data runs out. 00: The PA, DIVIDER, and LCOSC shutdown after last bit. 01: Reuse the last symbol group for transmission. 10: All 0s data. 11: All 1s data.
5	FSK_FORCE_DEV	**Force FSK Deviation.** 0: Normal operation. 1: Force the LCOSC to frequency deviate regardless of data pattern or FSK_MODE.
4	FSK_MODE	**Selects Modulation Mode.** 0: OOK mode. 1: FSK mode.
3	FORCE_LC	**Force LCOSC On.** .0: Normal operation. 1: Force LSCOSC on.
2	FORC_DIV	**Force DIVIDER On.** .0: Normal operation. 1: Force DIVIDER on.
1	FORCE_PA	**Force PA On.** .0: Normal operation. 1: Force PA on. In addition, PA_LVL_NSLICE[4:0] in PA_LVL register is passed directly through the serializer, unchanged.
0	ODS_EN	**Enable the Serializer.** 0: Disable the ODS. 1: Enable the ODS.

SFR Definition 12.2. ODS_TIMING

Bit	7	6	5	4	3	2	1	0
Name	ODS_GROUP_WIDTH[2:0]			ODS_EDGE_TIME [1:0]		ODS_CK_DIV[2:0]		
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xAA

Bit	Name	Function
7:5	ODS_ GROUP_ WIDTH[2:0]	**Controls Symbol Group width, from 2–8 Symbols.** Set to 4 to transmit 5 symbol groups obtained from 4/5 encoding. Or set to 7 to send 8 symbol group obtained from Manchester encoding of 4 bits. Note that ods_group_width can be changed dynamically prior to writing the ODS_DATA register, should you want to (for example) add 2 more symbols to the end of a transmission which was previously using 8 symbol groups.
4:3	ODS_ EDGE_ TIME [1:0]	**Controls PA Edge Time.** Additional division factor in range 1-4 (ods_edge time+1). PA controlled edge rates are: 8*(ods_ck_div+1)*(ods_edge_time+1)/25 MHz. When clk_ods is in range of 3-8 MHz, edge rate can be selected from 1us to 10.7us. Study has indicated that in the worst case (20Kbps Manchester), edge rates somewhat higher than 4us are needed.
2:0	ODS_CK_ DIV[2:0]	**Controls the Clock of the ODS.** Sets the division factor of the 24 MHz system clock to produce clk for the ODS module. Division factors are 1–8 (ods_ck_div+1). Generally should select factor which produces serializer clock in range of ~ 3-8 MHz

SFR Definition 12.3. ODS_DATA

Bit	7	6	5	4	3	2	1	0
Name	ODS_DATA[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xAB

Bit	Name	Function
7:0	ODS_DATA [7:0]	**ODS Input Data.** Symbol group register. Side effect of writing is clearing of ODS_EMPTY flag. It generates a single pulse for the ODS to notify the Tx ODS data SFR holding register been written to and contains new data. The pulse is a registered write pulse, so it will be generated when the data is stable in the holding register. ODS data format is little endian.

SFR Definition 12.4. ODS_RATEL

Bit	7	6	5	4	3	2	1	0
Name	ODS_RATEL[7:0]							
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xAC

Bit	Name	Function
7:0	ODS_RATEL [7:0]	**Lower Byte of the 15-bit Wide ODS Data Rate Field.** Symbol rate produced by the serializer is 24MHz/(ods_datarate*(ods_ck_div+1))

SFR Definition 12.5. ODS_RATEH

Bit	7	6	5	4	3	2	1	0
Name	Reserved	ODS_RATEH[6:0]						
Type	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xAD

Bit	Name	Function
7	Reserved	Read as 0. Write has no effect.
6:0	ODS_ RATEH [6:0]	**Upper Bits of 15-bit ODS Data Rate Field.** See the ODS_RATEL for description of the serializer data rates.

SFR Definition 12.6. ODS_WARM1

Bit	7	6	5	4	3	2	1	0
Name	ODS_WARM_DIV[3:0]				ODS_WARM_PA[3:0]			
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xAE

Bit	Name	Function
7:4	ODS_ WARM_ DIV[3:0]	**Sets Warm-Up Time for DIVIDER.** Sets the "warm up" interval for the DIVIDER, where it is biased up prior to transmission or on the transition from OOK Zero bit to OOK One bit. Interval is in 4 * clk_ods cycles resolution Interval = 4*ods_warm_pa*(ods_ck_div+1)/24 MHz When clk_ods is in range of 3-8 MHz, warm-up interval range is from 7.6 to 20 µs.
3:0	ODS_ WARM_ PA[3:0]	**Sets Warm-Up Time for PA.** Sets the "warm up" interval for the PA, where it is biased up prior to transmission or on the transition from OOK Zero bit to OOK One bit. Interval is directly in clk_ods cycles. Interval = ods_warm_pa x (ods_ck_div+1)/24 MHz When clk_ods is in range of 3–8 MHz, warm-up interval range is from 1.9 to 5 µs.

SFR Definition 12.7. ODS_WARM2

Bit	7	6	5	4	3	2	1	0
Name	Reserved				ODS_WARM_LC[3:0]			
Type	R				R/W			
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xAF

Bit	Name	Function
7:4	Reserved	Read as 0x0. Write has no effect.
3:0	ODS_ WARM_ LC[3:0]	**Sets Warm-Up Time for the LCOSC.** Sets the "warm up" interval for the LC oscillator, where it is biased up prior to transmission or on the transition from OOK. Zero bit to OOK One bit. Interval is in 64*clk_ods cycles resolution Interval = 64 x ods_warm_pa x (ods_ck_div+1)/24 MHz When clk_ods is in range of 3-8 MHz, warm-up interval range is from 30 to 80 µs.

13. LC Oscillator (LCOSC)

The Si4010 VCO is a fully integrated CMOS LC oscillator that operates at approximately 3.9 GHz. This block in conjunction with a programmable frequency divider generates the transmit carrier frequency. The technology behind the VCO is based on the Silicon Laboratories Si500 crystal-less oscillator chip and forms the core of the Si4010s' crystal-less operation. After this device is factory trimmed, the VCO frequency is the most accurate frequency on the chip and sets the chips transmit frequency stability unless an external crystal oscillator is used. The device achieves ±150 ppm frequency stability over the commercial temperature range of 0 to 70°C and ±250 ppm frequency stability over the industrial temperature range of –40 to 85 °C.

The transmit carrier frequency is set by using the API functions vFCast_Tune (desired carrier) and vFCast_Setup(). For FSK modulation, the frequency deviation is also a parameter to the freq_adjustment function. Users are recommended to use the API functions to set the corresponding SFR registers.

13.1. Register Description

SFR Definition 13.1. LC_FSK

Bit	7	6	5	4	3	2	1	0
Name	Reserved	FSK_DEVIATION[6:0]						
Type	R/W	R/W						
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xE4

Bit	Name	Function
7	Reserved	Do not write to this bit.
6:0	FSK_ DEVIATION [6:0]	**FSK Deviation.** These bits determine the FSK deviation. The values of these bits are calculated and entered from the API vFCast_FskAdj.

14. Low Power Oscillator and System Clock Generator

The source of all digital system clocks is derived from the low power oscillator (LPOSC) and system clock generator. The LPOSC produces a 24 MHz clock signal and is used by the system clock generator to produce the system clock. This system clock is applied to all digital blocks including the MCU and is programmable via the SYSGEN SFR register which is useful for power savings. Users are recommended to use the System Module Function API to set the registers.

14.1. Register Description

XREG Definition 14.1. bLPOSC_TRIM

Bit	7	6	5	4	3	2	1	0
Name	LPOSC_TRIM[7:0]							
Type	R/W							
Reset	1	1	1	1	1	1	1	1

XREG Address = 0x4002

Bit	Name	Function
7:0	LPOSC_ TRIM[7:0]	**Low Power Oscillator Trimming.** ±16% range with 0.14 % resolution. Setting all the bits to low will maximize the frequency of operation.

SFR Definition 14.2. SYSGEN

Bit	7	6	5	4	3	2	1	0
Name	SYSGEN_ SHUT-DOWN	Re-served	PWR_1ST _TIME	RTC_ TICKCLR	PORT_ HOLD	SYSGEN_DIV[2:0]		
Type	R/W	R	R	W	R/W	R/W		
Reset	0	0	—	0	0	0	0	0

SFR Address = 0xBE

Bit	Name	Function
7	SYSGEN_ SHUT-DOWN	**System General Shutdown.** Setting this bit causes shutdown of MCU and most analog. Recovery from this is via falling edge on any GPIO, which results in a power up and a power on reset. This is THE bit that shuts down the power to nearly everything. 0: Normal operation 1: Shutdown. Do not use this bit directly. It is recommended to use the vSys_Shutdown() API call.
6	Reserved	Read as 0. Write has no effect.
5	PWR_1ST_ TIME	**Initial Powerup Indicator.** Read only register. It will get set when power up was caused by a battery insertion.
4	RTC_ TICKCLR	**Real Time Clock Clear.** 0: Normal operation 1: Clears the real time clock 5.12us counter.
3	PORT_ HOLD	**Port Hold.** This bit needs to be set before shutting down, it delays any button pushes that occur between this bit setting and shutdown until the chip completes shutdown, to ensure the shutdown process cannot be interrupted. 0: Normal operation 1: Holds GPIO port values until shutdown is complete
2:0	SYSGEN_ DIV[2:0]	**System Clock Generator Divider.** System clock divider control to generate the system clock. 000: 24 MHz; div = 1 001: 12 MHz; div = 2 010: 6.0 MHz; div = 4 011: 3.0 MHz; div = 8 100: 1.5 MHz; div = 16 101: 0.75 MHz; div = 32 110: 0.375 MHz; div = 64 111: 0.1875 MHz; div = 128

15. Crystal Oscillator (XO)

The crystal oscillator produces an accurate clock reference for applications demanding a high-accuracy transmit carrier frequency. It uses a 1-pin crystal oscillator circuit (Colpitt's oscillator) and the output is connected to the frequency counter.

15.1. Register Description

XREG Definition 15.1. bXO_CTRL

Bit	7	6	5	4	3	2	1	0
Name	Reserved	Reserved	Reserved	Reserved	XO_TST[1:0]		XO_LOW CAP	XO_ENA
Type					R/W		R/W	R/W
Reset	0	0	0	0	0	0	0	0

XREG Address = 0x4016

Bit	Name	Function
7:4	Reserved	**Reserved.**
3:2	XO_ TST[1:0]	**Measurement of the XO Regenerative Amplifier Bias Current.** 0: No connection 1: Sense 2: Force 3: Sense and force
1	XO_ LOWCAP	**XO Low Capacitance.** Bit should be set for crystals that require less than 14 pF of total capacitance. 0: Crystals with 14 pF or more of total capacitance. 1: Crystals with less than 14 pF of total capacitance.
0	XO_ENA	**Enable XO.** Note that operation of the XO requires that the bandgap be enabled with the System Module Function API. The input XO_CKGOOD status bit is in the SFR SYSTEM register. 0: Crystal oscillator disabled. 1: Crystal oscillator enabled.

16. Frequency Counter

The frequency counter allows the measurement of the ratio of two selected clock sources: a low frequency clock which defines a counting interval, and a high frequency clock which is counted.

The frequency counter consists of an interval counter, driven by one of the six clock sources. Programming of the interval counter determines how long the main counter will count one of the two high speed clocks, LC oscillator or DIVIDER output.

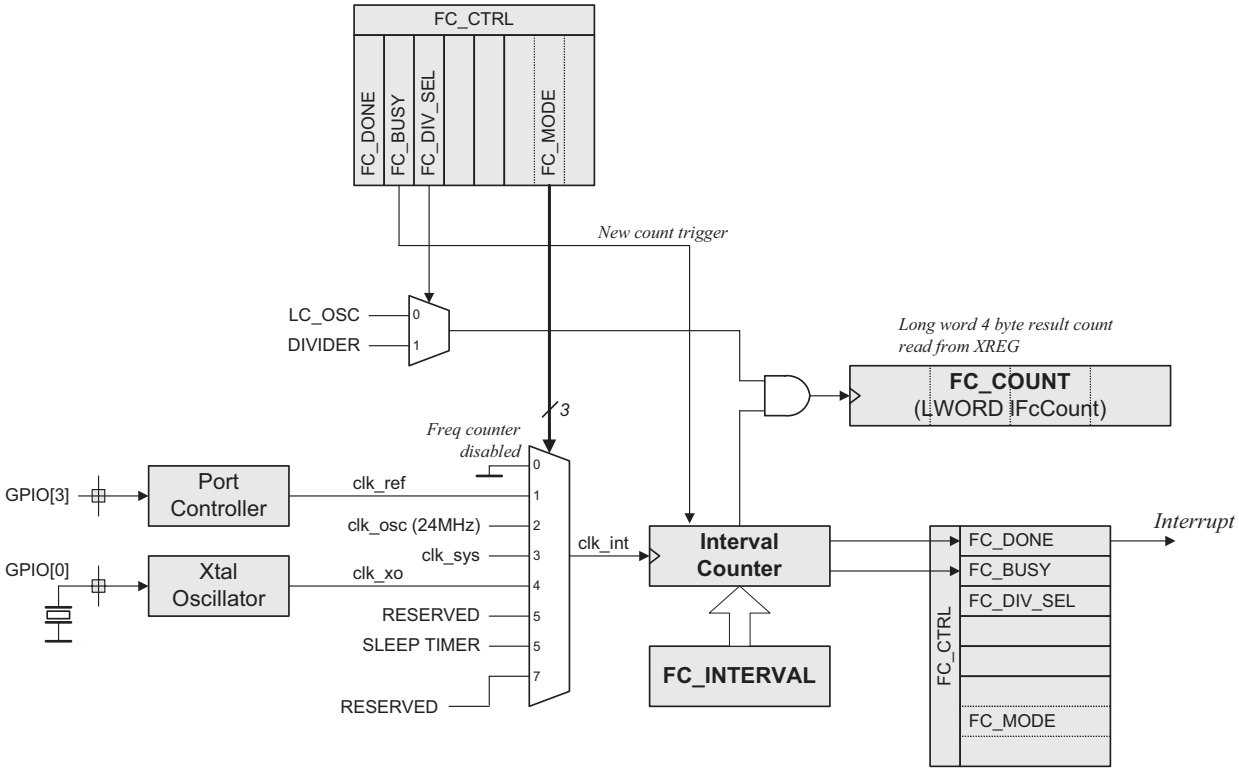


Figure 16.1. Frequency Counter Block Diagram

The block diagram of the frequency counter is in Figure 16.1. When the FC_MODE=0, the frequency counter is disabled. The only way to disable the frequency counter is to set the FC_MODE=0. The frequency counter stops counting immediately, so it can be restarted by setting FC_MODE to some functional mode immediately.

If the frequency counter is enabled by setting FC_MODE to other than the 0 value, it enters the idle state. To start the counter, the interval counter has to be triggered by writing 1 to the FC_BUSY bit. By writing FC_BUSY=1, the FC_DONE bit gets cleared as well. The user can also clear the FC_DONE bit in software after reading the main FC_COUNT value.

Once the interval counter is triggered, and after several **clk_sys** cycles synchronization delay it waits for the first rising edge of the **clk_int** clock, which is the output of the interval counter clock selector mux. It then enables the main frequency counter FC_COUNT clock. After the interval counter counts the interval specified by FC_INTERVAL SFR register, another rising edge of the **clk_int** stops the clocks to the main FC_COUNT counter. The interval counter edge to edge counting and main FC_COUNT clock enable is measured very accurately in between the **clk_int** rising edges.

When the interval counter is finished with the interval count, it clears the FC_BUSY=0 bit and after a few cycles of **clk_sys** synchronization delay it sets the FC_DONE=1 bit. Both interval counter and main FC_COUNT counter are stopped and the main FC_COUNT keeps the accumulated value until the frequency counter is disabled or triggered again. The 23 bit FC_COUNT value can be read as a 4 byte long word, **IFcCount**, from the XREG register in XDATA. When the counter is counting and FC_BUSY=1, then reading the FC_COUNT value returns the on the fly changing value of the FC_COUNT counter.

The frequency counter is restartable. If 1 is written to FC_BUSY while the frequency counter is busy then the current FC_COUNT result is discarded, main FC_COUNT is reset, and the interval counter is triggered, waiting for the first rising edge of the **clk_int** clock.

The count interval is chosen with the FC_INTERVAL SFR register. The number of interval count cycles (count cycles of the low frequency clock) = (2+FC_INTERVAL[0])*(2^FC_INTERVAL[5:1]).

Note: FC_INTERVAL is not allowed to take on numbers higher than 43. If the number is higher than 43, then the interval counted is forced to 1.

The output of the frequency counter is in the XREG FC_COUNT. The user is recommended to use the Frequency Counter Module Function API to set the following registers.

16.1. Register Description

SFR Definition 16.1. FC_CTRL

Bit	7	6	5	4	3	2	1	0
Name	FC_DONE	FC_BUSY	FC_DIV_SEL	Reserved		FC_MODE[2:0]		
Type	R/W	R/W	R/W	R		R/W		
Reset	0	0	0	0		0		

SFR Address = 0x9B

Bit	Name	Function
7	FC_DONE	**Frequency Counter Done.** Counting done, interrupt generation level signal. Must be cleared by software ISR. It is also cleared if 1 is written to fc_busy, which denotes the start of the next count. Any value can be written here, so one can invoke interrupt just writing 1 here. 0: Frequency counter is counting 1: Frequency counter done counting, must be cleared by software ISR
6	FC_BUSY	**Frequency Counter Busy.** Frequency counter is busy counting. Falling edge of the fc_busy signal sets the FC_DONE=1. Writing 1 to this bit triggers a new FC counting cycle. FC is restartable, so any Wr 1 to this bit restarts the FC and discards what the FC was currently doing. 0: Frequency counter is not busy, falling edge sets FC_DONE=1 1: Writing 1 restarts the Frequency Counter
5	FC_DIV_SEL	**Frequency Counter Divider Select.** Selection control of source of clock. It chooses between LC and DIVIDER. If the frequency counter is not enabled, FC_MODE=0, then both signals mentioned above are in their inactive states. 0: LCOSC 1: DIVIDER
4:3	Reserved	Read as 0x0. Write has no effect.
2:0	FC_MODE [2:0]	**Frequency Counter Mode Control Register.** 000: Frequency counter disabled 001: Interval: clk_ref .. reference clock from GPIO 010: Interval: clk_osc .. undivided output of Low Power Osc (24 MHz) 011: Interval: clk_sys .. system clock, divided output of Low Power Osc 100: Interval: clk_xo .. XO oscillator 101: Reserved 110: Interval: Sleep Timer output 111: Reserved

SFR Definition 16.2. FC_INTERVAL

Bit	7	6	5	4	3	2	1	0
Name	Reserved	Reserved	FC_INTERVAL[5:0]					
Type			R/W					
Reset	0		0					

SFR Address = 0x9D

Bit	Name	Function
7:6	Reserved	**Reserved.**
5:0	FC_ INTERVAL [5:0]	**Frequency Counter Interval.** Controls number of interval clock cycles in an interval. n_cycles = (2+fcnt_interval[0])*(2^fcnt_interval[5:1]) Note that fcnt_interval is allowed to take on values no higher than 43. If the number higher than 43 is used then the the interval counted is forced to n_cycles = 1.

XREG Definition 16.3. IFC_COUNT

Bit	22	21	...	1	0
Name	FC_COUNT[22:0]				
Type	R				
Reset	0	0	...	0	0

XREG Address = 0x4008

Bit	Name	Function
22:0	FC_COUNT [22:0]	**Frequency Counter Output.** Counter output value. When the counter is running and the value is read then the current on the fly value will be read

17. Sleep Timer

The Si4010 includes a very low-power sleep timer that can be used to support the transmit duty cycle requirements of the ETSI specification or self-wakeup for button independent applications. It consist of a low speed (~2.1 kHz), very low power oscillator with a 24 bit down counter. When programmed to its maximum interval it takes ~2.1 hours to count down to zero. When it counts down to zero, it automatically powers down completely. The sleep timer can also be programmed to wake up the chip if the chip was powered down. Control of the sleep timer is done with the API Sleep Timer Module functions.

18. Bandgap and LDO

Power management is provided on chip with LDO regulators for the internal analog and digital supplies, VA and VD, respectively. The power-on reset circuit monitors the power applied to the chip and generates a reset signal to set the chip into a known state. The bandgap produces voltage and current references for the analog blocks in the chip and can be shut down when the analog blocks are not used. Control of the bandgap and LDO is done with the System Module Function API vSys_BandGapLdo.

19. Low Leakage HVRAM

The low-leakage HVRAM provides 8 bytes of RAM memory which keeps its contents in all states including standby mode as long as the supply voltage is applied to the chip. Control of the HVRAM is done with the API HVRAM Module Functions.

20. Temperature Sensor

The on-chip temperature sensor measures the internal temperature of the chip and the temperature demodulator converts the temperature sensors' output into a binary number representing temperature and is used to compensate the frequency of the LCOSC when the temperature changes. Temperature compensation of the LCOSC is automatically taken care of by the Single Transmission Loop Module Function API. The Demodulator Temperature Sensor Module Function APIs can be used to get samples of the current temperature when not transmitting. Each device is frequency and temperature calibrated in the factory.

21. CIP-51 Microcontroller

The MCU system controller core is the CIP-51 microcontroller. The CIP-51 is fully compatible with the MCS-51™ instruction set; standard 803x/805x assemblers and compilers can be used to develop software. The MCU family has a superset of all the peripherals included with a standard 8051. The CIP-51 also includes on-chip debug hardware, and interfaces directly with the analog and digital subsystems providing a complete RF transmitter solution in a single integrated circuit.

The CIP-51 Microcontroller core implements the standard 8051 organization and peripherals as well as additional custom peripherals and functions to extend its capability. The CIP-51 includes the following features:

- Fully Compatible with MCS-51 Instruction Set
- 24 MIPS Peak Throughput with 24 MHz Clock
- 0 to 24 MHz Clock Frequency
- Extended Interrupt Handler
- Power Management Modes
- On-chip Debug Logic
- Program and Data Memory Security

Performance

The CIP-51 employs a pipelined architecture that greatly increases its instruction throughput over the standard 8051 architecture. In a standard 8051, all instructions except for MUL and DIV take 12 or 24 system clock cycles to execute, and usually have a maximum system clock of 12 MHz. By contrast, the CIP-51 core executes 70% of its instructions in one or two system clock cycles, with no instructions taking more than eight system clock cycles.

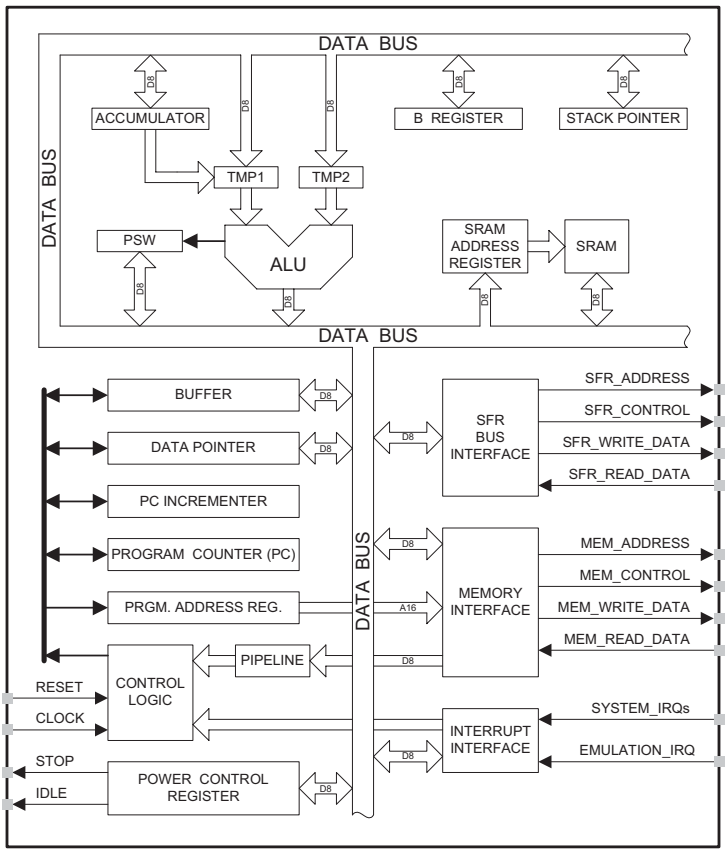


Figure 21.1. CIP-51 Block Diagram

With the CIP-51's maximum system clock at 24 MHz, it has a peak throughput of 24 MIPS. The CIP-51 has a total of 109 instructions. The table below shows the total number of instructions in the function of the required clock cycles.

Clocks to Execute	1	2	2/3	3	3/4	4	4/5	5	8
Number of Instructions	26	50	5	14	7	3	1	2	1

21.1. Instruction Set

The instruction set of the CIP-51 System Controller is fully compatible with the standard MCS-51™ instruction set. Standard 8051 development tools can be used to develop software for the CIP-51. All CIP-51 instructions are the binary and functional equivalent of their MCS-51™ counterparts, including opcodes, addressing modes and effect on PSW flags. However, instruction timing is different than that of the standard 8051.

21.1.1. Instruction and CPU Timing

In many 8051 implementations, a distinction is made between machine cycles and clock cycles, with machine cycles varying from 2 to 12 clock cycles in length. However, the CIP-51 implementation is based solely on clock cycle timing. All instruction timings are specified in terms of clock cycles.

Due to the pipelined architecture of the CIP-51, most instructions execute in the same number of clock cycles as there are program bytes in the instruction. Conditional branch instructions take one less clock cycle to complete when the branch is not taken as opposed to when the branch is taken. Table 21.1 is the CIP-51 Instruction Set Summary, which includes the mnemonic, number of bytes, and number of clock cycles for each instruction.

Table 21.1. CIP-51 Instruction Set Summary

Mnemonic	Description	Bytes	Clock Cycles
Arithmetic Operations			
ADD A, Rn	Add register to A	1	1
ADD A, direct	Add direct byte to A	2	2
ADD A, @Ri	Add indirect RAM to A	1	2
ADD A, #data	Add immediate to A	2	2
ADDC A, Rn	Add register to A with carry	1	1
ADDC A, direct	Add direct byte to A with carry	2	2
ADDC A, @Ri	Add indirect RAM to A with carry	1	2
ADDC A, #data	Add immediate to A with carry	2	2
SUBB A, Rn	Subtract register from A with borrow	1	1
SUBB A, direct	Subtract direct byte from A with borrow	2	2
SUBB A, @Ri	Subtract indirect RAM from A with borrow	1	2
SUBB A, #data	Subtract immediate from A with borrow	2	2
INC A	Increment A	1	1
INC Rn	Increment register	1	1
INC direct	Increment direct byte	2	2
INC @Ri	Increment indirect RAM	1	2
DEC A	Decrement A	1	1
DEC Rn	Decrement register	1	1
DEC direct	Decrement direct byte	2	2
DEC @Ri	Decrement indirect RAM	1	2
INC DPTR	Increment Data Pointer	1	1
MUL AB	Multiply A and B	1	4
DIV AB	Divide A by B	1	8
DA A	Decimal adjust A	1	1
Logical Operations			
ANL A, Rn	AND Register to A	1	1
ANL A, direct	AND direct byte to A	2	2
ANL A, @Ri	AND indirect RAM to A	1	2
ANL A, #data	AND immediate to A	2	2
ANL direct, A	AND A to direct byte	2	2
ANL direct, #data	AND immediate to direct byte	3	3
ORL A, Rn	OR Register to A	1	1
ORL A, direct	OR direct byte to A	2	2
ORL A, @Ri	OR indirect RAM to A	1	2
ORL A, #data	OR immediate to A	2	2
ORL direct, A	OR A to direct byte	2	2
ORL direct, #data	OR immediate to direct byte	3	3
XRL A, Rn	Exclusive-OR Register to A	1	1
XRL A, direct	Exclusive-OR direct byte to A	2	2
XRL A, @Ri	Exclusive-OR indirect RAM to A	1	2
XRL A, #data	Exclusive-OR immediate to A	2	2
XRL direct, A	Exclusive-OR A to direct byte	2	2

Table 21.1. CIP-51 Instruction Set Summary (Continued)

Mnemonic	Description	Bytes	Clock Cycles
XRL direct, #data	Exclusive-OR immediate to direct byte	3	3
CLR A	Clear A	1	1
CPL A	Complement A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through Carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through Carry	1	1
SWAP A	Swap nibbles of A	1	1
Data Transfer			
MOV A, Rn	Move Register to A	1	1
MOV A, direct	Move direct byte to A	2	2
MOV A, @Ri	Move indirect RAM to A	1	2
MOV A, #data	Move immediate to A	2	2
MOV Rn, A	Move A to Register	1	1
MOV Rn, direct	Move direct byte to Register	2	2
MOV Rn, #data	Move immediate to Register	2	2
MOV direct, A	Move A to direct byte	2	2
MOV direct, Rn	Move Register to direct byte	2	2
MOV direct, direct	Move direct byte to direct byte	3	3
MOV direct, @Ri	Move indirect RAM to direct byte	2	2
MOV direct, #data	Move immediate to direct byte	3	3
MOV @Ri, A	Move A to indirect RAM	1	2
MOV @Ri, direct	Move direct byte to indirect RAM	2	2
MOV @Ri, #data	Move immediate to indirect RAM	2	2
MOV DPTR, #data16	Load DPTR with 16-bit constant	3	3
MOVC A, @A+DPTR	Move code byte relative DPTR to A	1	3
MOVC A, @A+PC	Move code byte relative PC to A	1	3
MOVX A, @Ri	Move external data (8-bit address) to A	1	3
MOVX @Ri, A	Move A to external data (8-bit address)	1	3
MOVX A, @DPTR	Move external data (16-bit address) to A	1	3
MOVX @DPTR, A	Move A to external data (16-bit address)	1	3
PUSH direct	Push direct byte onto stack	2	2
POP direct	Pop direct byte from stack	2	2
XCH A, Rn	Exchange Register with A	1	1
XCH A, direct	Exchange direct byte with A	2	2
XCH A, @Ri	Exchange indirect RAM with A	1	2
XCHD A, @Ri	Exchange low nibble of indirect RAM with A	1	2
Boolean Manipulation			
CLR C	Clear Carry	1	1
CLR bit	Clear direct bit	2	2
SETB C	Set Carry	1	1
SETB bit	Set direct bit	2	2
CPL C	Complement Carry	1	1
CPL bit	Complement direct bit	2	2

Table 21.1. CIP-51 Instruction Set Summary (Continued)

Mnemonic	Description	Bytes	Clock Cycles
ANL C, bit	AND direct bit to Carry	2	2
ANL C, /bit	AND complement of direct bit to Carry	2	2
ORL C, bit	OR direct bit to carry	2	2
ORL C, /bit	OR complement of direct bit to Carry	2	2
MOV C, bit	Move direct bit to Carry	2	2
MOV bit, C	Move Carry to direct bit	2	2
JC rel	Jump if Carry is set	2	2/3
JNC rel	Jump if Carry is not set	2	2/3
JB bit, rel	Jump if direct bit is set	3	3/4
JNB bit, rel	Jump if direct bit is not set	3	3/4
JBC bit, rel	Jump if direct bit is set and clear bit	3	3/4
Program Branching			
ACALL addr11	Absolute subroutine call	2	3
LCALL addr16	Long subroutine call	3	4
RET	Return from subroutine	1	5
RETI	Return from interrupt	1	5
AJMP addr11	Absolute jump	2	3
LJMP addr16	Long jump	3	4
SJMP rel	Short jump (relative address)	2	3
JMP @A+DPTR	Jump indirect relative to DPTR	1	3
JZ rel	Jump if A equals zero	2	2/3
JNZ rel	Jump if A does not equal zero	2	2/3
CJNE A, direct, rel	Compare direct byte to A and jump if not equal	3	4/5
CJNE A, #data, rel	Compare immediate to A and jump if not equal	3	3/4
CJNE Rn, #data, rel	Compare immediate to Register and jump if not equal	3	3/4
CJNE @Ri, #data, rel	Compare immediate to indirect and jump if not equal	3	4/5
DJNZ Rn, rel	Decrement Register and jump if not zero	2	2/3
DJNZ direct, rel	Decrement direct byte and jump if not zero	3	3/4
NOP	No operation	1	1

Notes on Registers, Operands and Addressing Modes:
Rn—Register R0–R7 of the currently selected register bank.
@Ri—Data RAM location addressed indirectly through R0 or R1.
rel—8-bit, signed (twos complement) offset relative to the first byte of the following instruction. Used by SJMP and all conditional jumps.
direct—8-bit internal data location's address. This could be a direct-access Data RAM location (0x00–0x7F) or an SFR (0x80–0xFF).
#data—8-bit constant
#data16—16-bit constant
bit—Direct-accessed bit in Data RAM or SFR
addr11—11-bit destination address used by ACALL and AJMP. The destination must be within the same 2 kB page of program memory as the first byte of the following instruction.
addr16—16-bit destination address used by LCALL and LJMP. The destination may be anywhere within the 8 kB program memory space.
There is one unused opcode (0xA5) that performs the same function as NOP. All mnemonics copyrighted © Intel Corporation 1980.

21.2. CIP-51 Register Descriptions

Following are descriptions of SFRs related to the operation of the CIP-51 System Controller. Reserved bits should always be written to the value indicated in the SFR description. Future product versions may use these bits to implement new features in which case the reset value of the bit will be the indicated value, selecting the feature's default state. Detailed descriptions of the remaining SFRs are included in the sections of the data sheet associated with their corresponding system function.

SFR Definition 21.1. DPL

Bit	7	6	5	4	3	2	1	0
Name	DPL[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0x82

Bit	Name	Function
7:0	DPL[7:0]	**Data Pointer Low.** The DPL register is the low byte of the 16-bit DPTR.

SFR Definition 21.2. DPH

Bit	7	6	5	4	3	2	1	0
Name	DPH[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0x83

Bit	Name	Function
7:0	DPH[7:0]	**Data Pointer High.** The DPH register is the high byte of the 16-bit DPTR.

SFR Definition 21.3. SP

Bit	7	6	5	4	3	2	1	0
Name	SP[7:0]							
Type	R/W							
Reset	0	0	0	0	0	1	1	1

SFR Address = 0x81

Bit	Name	Function
7:0	SP[7:0]	**Stack Pointer.** The Stack Pointer holds the location of the top of the stack. The stack pointer is incremented before every PUSH operation. The SP register defaults to 0x07 after reset.

SFR Definition 21.4. ACC

Bit	7	6	5	4	3	2	1	0
Name	ACC[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xE0; Bit-Addressable

Bit	Name	Function
7:0	ACC[7:0]	**Accumulator.** This register is the accumulator for arithmetic operations.

SFR Definition 21.5. B

Bit	7	6	5	4	3	2	1	0
Name	B[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xF0; Bit-Addressable

Bit	Name	Function
7:0	B[7:0]	**B Register.** This register serves as a second accumulator for certain arithmetic operations.

SFR Definition 21.6. PSW

Bit	7	6	5	4	3	2	1	0
Name	CY	AC	F0	RS[1:0]		OV	F1	PARITY
Type	R/W	R/W	R/W	R/W		R/W	R/W	R
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xD0; Bit-Addressable

Bit	Name	Function
7	CY	**Carry Flag.** This bit is set when the last arithmetic operation resulted in a carry (addition) or a borrow (subtraction). It is cleared to logic 0 by all other arithmetic operations.
6	AC	**Auxiliary Carry Flag.** This bit is set when the last arithmetic operation resulted in a carry into (addition) or a borrow from (subtraction) the high order nibble. It is cleared to logic 0 by all other arithmetic operations.
5	F0	**User Flag 0.** This is a bit-addressable, general purpose flag for use under software control.
4:3	RS[1:0]	**Register Bank Select.** These bits select which register bank is used during register accesses. 00: Bank 0, Addresses 0x00-0x07 01: Bank 1, Addresses 0x08-0x0F 10: Bank 2, Addresses 0x10-0x17 11: Bank 3, Addresses 0x18-0x1F
2	OV	**Overflow Flag.** This bit is set to 1 under the following circumstances: - An ADD, ADDC, or SUBB instruction causes a sign-change overflow. - A MUL instruction results in an overflow (result is greater than 255). - A DIV instruction causes a divide-by-zero condition. The OV bit is cleared to 0 by the ADD, ADDC, SUBB, MUL, and DIV instructions in all other cases.
1	F1	**User Flag 1.** This is a bit-addressable, general purpose flag for use under software control.
0	PARITY	**Parity Flag.** This bit is set to logic 1 if the sum of the eight bits in the accumulator is odd and cleared if the sum is even.

22. Memory Organization

The memory organization of the Si4010 is similar to that of a standard 8051. There are two separate memory spaces: program memory and data memory. Program and data memory share the same address space but are accessed via different instruction types. However, this device is unique since it has the program and data memory spaces combined into one. This is called a unified CODE and XDATA memory.

The device has a standard 8051 program and data address configuration. It includes 256 bytes of data RAM, with the upper 128 bytes dual-mapped. Indirect addressing accesses the upper 128 bytes of general purpose RAM, and direct addressing accesses the 128 byte SFR address space. The lower 128 bytes of RAM are accessible via direct and indirect addressing. The first 32 bytes are addressable as four banks of general purpose registers, and the next 16 bytes can be byte addressable or bit addressable.

Apart from the CPU core related internal memory, the device has the following memories:

- 4.5 kB of RAM .. it can be used both as program CODE and external data XDATA memory
- 12 kB of ROM .. it holds the Silicon Labs provided API (Application Programming Interface) routines. The ROM is not readable by the user.
- 256B hardware control registers mapped to XDATA address space (XREG)
- 8 kB of one time programmable (OTP) non-volatile memory (NVM)
- 128 bits of multiple time programmable (MTP) EEPROM. Each bit can change value at most 50,000 times.

See Figure 22.1 for the MCU system memory map:

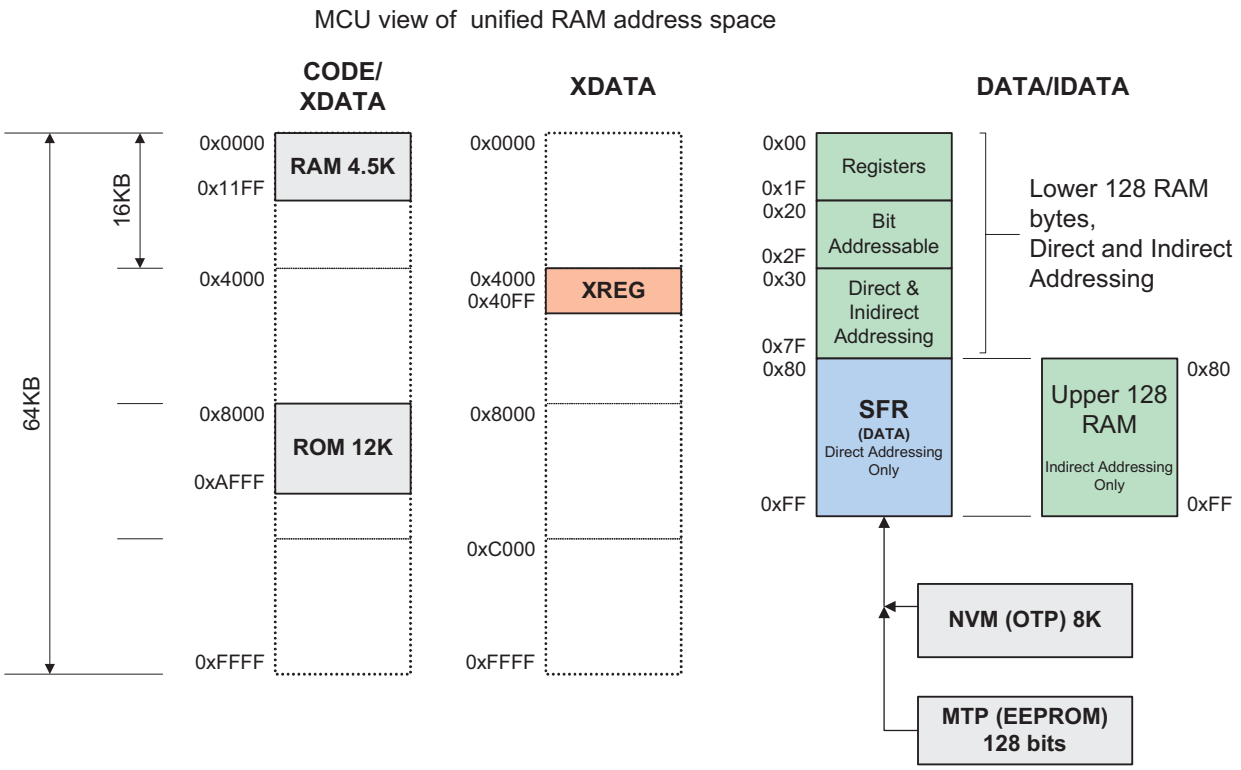


Figure 22.1. Address Space Map after the Boot

22.1. Program Memory

Program memory consists of 4.5 kB for RAM and 12 kB of ROM. The device employs a unified CODE/XDATA RAM memory. On 8051 architecture the external data memory (XDATA) space is physically different from the program memory (CODE); they can be accessed with different instructions. On this device the RAM can store both CODE and XDATA at any location. The program memory is commonly called CODE memory, residing in CODE address space.

Both MOVC and MOVX instructions can be used to read data from the CODE/XDATA address space.

The ROM holds the Silicon Labs proprietary code and cannot be read by a user. Only code can be executed from ROM. If read is attempted by MOVC or MOVX instructions from ROM area the read value is undetermined. The NVM and MTP memories are not mapped to the CPU address space.

22.2. Internal Data Memory

The device implements 256 bytes of internal RAM mapped into the data memory space from 0x00 through 0xFF. The lower 128 bytes of data memory are used for general purpose registers and memory. Either direct or indirect addressing may be used to access the lower 128 bytes of data memory. Locations 0x00 through 0x1F are addressable as four banks of general purpose registers, each bank consisting of eight byte-wide registers. The next 16 bytes, locations 0x20 through 0x2F, may either be addressed as bytes or as 128 bit locations accessible with the direct addressing mode.

The upper 128 bytes of data memory are accessible only by indirect addressing. This region occupies the same address space as the Special Function Registers (**SFR**) but is physically separate from the SFR space. The addressing mode used by an instruction when accessing locations above 0x7F determines whether the CPU accesses the upper 128 bytes of data memory space or the SFRs. Instructions that use direct addressing will access the SFR space. Instructions using indirect addressing above 0x7F access the upper 128 bytes of data memory. Figure 22.1 illustrates the data memory organization.

22.3. External Data Memory

Even though it is called external memory, it resides on the chip. This is the data memory, up to 64 kB in size, which is accessible by MOVX instructions. For the original MCS-51™ architecture this memory resided physically external to the chip. This memory is commonly referred as XDATA memory.

The device implements shared CODE/XDATA memory. The 4.5 kB of RAM is shared between the CODE and XDATA. The CPU can run code from any location of that RAM, can read any location using MOVC and MOVX instructions, and can write any location by using MOVX instruction.

Important note: Linker of the user application has to be given proper regions of CODE and XDATA memory, which are mutually exclusive. Therefore, for example, the user cannot set the CODE region to be 0x0000 .. 0x1000 and XDATA region to be the very same at the same time. One has to specify two non-overlapping regions in the RAM area instead.

22.4. General Purpose Registers

The lower 32 bytes of data memory, locations 0x00 through 0x1F, may be addressed as four banks of general-purpose registers. Each bank consists of eight byte-wide registers designated R0 through R7. Only one of these banks may be enabled at a time. Two bits in the program status word, RS0 (PSW.3) and RS1 (PSW.4), select the active register bank. This allows fast context switching when entering subroutines and interrupt service routines. Indirect addressing modes use registers R0 and R1 as index registers.

22.5. Bit Addressable Locations

In addition to direct access to data memory organized as bytes, the sixteen data memory locations at 0x20 through 0x2F are also accessible as 128 individually addressable bits. Each bit has a bit address from 0x00 to 0x7F. Bit 0 of the byte at 0x20 has bit address 0x00 while bit 7 of the byte at 0x20 has bit address 0x07. Bit 7 of the byte at 0x2F has bit address 0x7F. A bit access is distinguished from a full byte access by the type of instruction used (bit source or destination operands as opposed to a byte source or destination).

The MCS-51™ assembly language allows an alternate notation for bit addressing of the form XX.B where XX is the byte address and B is the bit position within the byte.

For example, the instruction:

```
MOV C, 22.3h
```

moves the Boolean value at 0x13 (bit 3 of the byte at location 0x22) into the Carry flag.

22.6. Stack

A programmer's stack can be located anywhere in the 256 byte data memory. The stack area is designated using the Stack Pointer (SP, address 0x81) SFR. The SP will point to the last location used. The next value pushed on the stack is placed at SP+1 and then SP is incremented. A reset initializes the stack pointer to location 0x07; therefore, the first value pushed on the stack is placed at location 0x08, which is also the first register (R0) of register bank 1. Thus, if more than one register bank is to be used, the SP should be initialized to a location in the data memory not being used for data storage. The stack depth can extend up to 256 bytes.

22.7. Special Function Registers (SFR)

The direct-access data memory locations from 0x80 to 0xFF constitute the special function registers (SFRs). The SFRs provide control and data exchange with the CIP-51's resources and peripherals. The CIP-51 duplicates the SFRs found in a typical 8051 implementation as well as implementing additional SFRs used to configure and access the sub-systems unique to the MCU. This allows the addition of new functionality while retaining compatibility with the MCS-51™ instruction set. Table 24.1 lists the SFRs implemented in the device.

The SFR registers are accessed whenever the direct addressing mode is used to access memory locations from 0x80 to 0xFF. SFRs with addresses ending in 0x0 or 0x8 (e.g. P0, P1, IE, etc.) are bit-addressable as well as byte-addressable. All other SFRs are byte-addressable only. Unoccupied addresses in the SFR space are reserved for future use. Accessing these areas will have an indeterminate effect and should be avoided. Refer to the corresponding pages of the data sheet for a detailed description of each register.

22.8. Registers Mapped to XDATA Address Space (XREG)

Given the extensive requirement for the numerous hardware registers some of the registers are mapped to the XDATA space as shown in Figure 22.1. Those registers are accessible only by MOVX instructions and are viewed from the CPU as a regular external XDATA memory. Registers which are more than single byte wide are organized in big endian fashion (most significant byte on the lowest address) to comply with the Keil development toolchain. They can be declared as regular variables in higher level languages, like C.

Map of user accessible XREG registers is in Table 24.3.

22.9. NVM (OTP) Memory

NVM memory is only accessible indirectly through Silicon Labs provided API functions for NVM access initialization and read of formatted blocks of data generated by the NVM programmer. Programming of the NVM can be only done by Silicon Labs provided tools. It is not possible to program the NVM by writing to registers.

22.10. MTP (EEPROM) Memory

The MTP memory is a special block not organized as a usual memory. The memory output is mapped to the XDATA address space as a XREG register (abMTP_RDATA[16]) 16 byte **read only** array at addresses 0x4040 .. 0x404F. Writing to the MTP memory can be done only indirectly by using the Silicon Labs provided API ROM functions.

To write to MTP the user must prepare an array of all 16 bytes in CODE/XDATA RAM. There is no byte access to MTP. Even if only a single bit is to be changed in MTP, the current content must be copied to the CODE/XDATA RAM in full, all 16 bytes. Then the desired bit has to be changed in that RAM copy and an API function has to be called to program the 16 byte changed data from RAM to MTP. The user can use the API MTP copy call to get the current content of MTP into CODE/XDATA RAM for modifications. If the MTP bit is not changing value the programming cycle is not counted against the maximum bit change durability of MTP. Therefore, programming the 16 byte MTP content unchanged from the current value has no effect on the longevity of the MTP.

There is no direct write access to MTP through registers. Silicon Labs API ROM functions must be used.

XREG Definition 22.1. abMTP_RDATA[16]

Byte	15	14	...	1	0
Name	abMTP_RDATA[0:15]				
Type	R				
Reset	—	—	...	—	—

XREG Address = 0x4040

Byte	Name	Function
15:0	abMTP_ RDATA[0:15]	**MTP Read Data.** MTP 16-byte read only array.

23. System Boot and NVM Programming

The device does not include a Flash memory for permanent code or data storage. Instead, the device contains 4. 5kB of RAM, which can serve as a unified CODE and XDATA RAM memory. The device contains of 8 kB of NVM (OTP) memory for user code and data storage. Small part of the NVM is reserved for Silicon Labs factory use and is not available to a user. In general more than 7 kB of NVM will be available for user application use.

23.1. Startup Overview

The code cannot be run directly from NVM, since it is not mapped directly to the CPU address space. Instead, upon device reset, the device goes through a boot process during which the factory chip configuration and the user application code and data is copied from NVM to the CODE/XDATA RAM. Only after the boot process finishes the user code starts being executed from CODE/XDATA RAM address 0x0000.

Therefore upon reset the device does not execute the user code immediately, but only after the boot process finishes. The time in between the device wakeup, either caused by cycling the power or waking up from the shutdown mode by button press, depends on the size of the user code load.

In general the startup time is about 2ms of fixed time plus 3.6 ms per 1 kB of user application code. For example, 4 kB application will incur

Tstartup = 2 ms + 3.6 ms x **User**_KB = 2 ms + 3.6 ms x 4 = 16.4 ms

startup time before the user application starts being executed.

For debugging purposes user will not program the NVM, but will use the RAM for code development. In that case the device will only contain factory settings and go through much shorter startup routine, which would take less than 2 ms to finish.

23.2. Reset

Reset circuitry allows the controller to be easily placed in a predefined default condition. There is only one external reset source for the device, which is power on reset. It get invoked at two occasions:

1. Power is supplied to the device. This means connecting the power supply to disconnected device.
2. The device is waking up from a shutdown mode. The power supply was connected before, but the device was put into the shutdown mode. When it is awaken the power is supplied internally to all the device systems.

On entry to this reset state, the following occur:

- CIP-51 halts program execution
- Special Function Registers (**SFR**) are initialized to their defined reset values
- XDATA registers (**XREG**) are initialized to their defined reset values
- External Port pins are forced to a known state
- Interrupts and timers are disabled

All SFRs are reset to the predefined values noted in the SFR detailed descriptions. The contents of internal data memory is lost, since the power got cycled.

The Port I/O latches are reset to 0xFF (all logic ones) in open-drain mode.

On exit from the reset state, the program counter (PC) is reset, and the system clock defaults to the internal oscillator. Device starts its boot sequence. See other sections for description of the boot sequence.

23.3. Chip Program Levels

The boot process starts by reading the NVM configuration bytes in the **Factory** region of NVM. The information about the programmed level of the chip is read first and the boot process acts accordingly.

After boot, the program level of the chip can be read as **NVM_BLOWN**[2:0] field in the **PROT0_CTRL** register.

From user point of view there are 3 program levels of the chip:

1. **Factory** .. blank part leaving the factory. The factory chip calibration is written into NVM. ROM and NVM **Factory** region is not readable by the user. Part can be used with debugging chain for software development and **User** load can be programmed to the part. Boot process initializes the part based on the **Factory** settings.
2. **User** .. same as **Factory** (blank) part, but with the **User** region in the NVM programmed with user code. The boot process will initialize the part according to the **Factory** settings and then (see Note 1. in section “23.5. Device Boot Process”) copies the **User** load to the CODE/XDATA or IRAM based on the **User** load. The code is not automatically run (see Note 2. in section “23.5. Device Boot Process”). The part can be used with IDE for further software development. The part is still opened for further NVM programming and the user can add additional data to the **User** region in the NVM. Debugging of the code loaded from NVM is possible. The user can modify the boot behavior of the **User** part by controlling two bits described later in the boot sequence description.
   This program level can be used two ways:
   - User programs the **User** code to check the load before finalizing the product.
   - Silicon Labs program most of the **User** code into the chip. Then the customer will add additional information specific for each chip on his own. For example, the customer may chose to let Silicon Labs program all the application data, but wants to program security keys into each chip on their own. This **User** level would be the chip program level delivered to a customer.
3. **Run** .. mission mode part, fully programmed for use in the field. No further NVM programming possible, no C2 interface access enabled, with the exception of special mode for retest. No possibility of IDE debug. The boot process is the same as in the case of **User** part, but after the user load is copied from

NVM to RAM, the boot loader executes a jump to RAM address 0x0000 and the user application is executed. The C2 is not enabled in this mode with the retest exception, briefly described in this document.

The IDE debugging environment can be used only with the **Factory** and **User** program chip levels, not with the **Run** part.

23.4. NVM Organization

The 8 kB NVM (OTP) memory is virtually mapped to the device address space 0xE000 .. 0xFFFF. However, CPU can access NVM only indirectly using the predefined API calls in ROM.

The NVM address region is organized in the following fashion:

1. **Factory** region .. factory settings critical for chip functions. Size is variable based on the device configuration.
2. **User** region .. region available for User application load at boot time. If the user application is not going to use overlays, then this will be the only user data region used.
3. **User App** optional region .. optional region not visible at boot time. If the user application is using overlays, then the overlay code will be stored in this region. It will be up to the user to load the application code from the NVM to CODE/XDATA RAM at runtime based on the user application request. Application note will be devoted to this technique.
4. **Reserved** region .. last 64 bytes of NVM are reserved for factory use and not available for user load.

The **User** load can occupy the rest of the NVM. The user may decide that he will use overlays. That means that the boot routine will not copy all the data from NVM to RAM upon boot, but during the runtime of the user program the program itself will load data from NVM to the RAM as desired. Only the **User** region is known to boot routine and will be loaded during boot.

The **User App** region is the data region available to the user for a load to be loaded at runtime by the user program. The user will have to call the API NVM copy routine in that case. The application note will describe this process in detail. In such a scenario, this NVM region will not be loaded by boot, but by the user application. That region of NVM is labeled as **User App** region in Figure 23.1, “NVM Address Map”. Boot routine will not know about the data there.

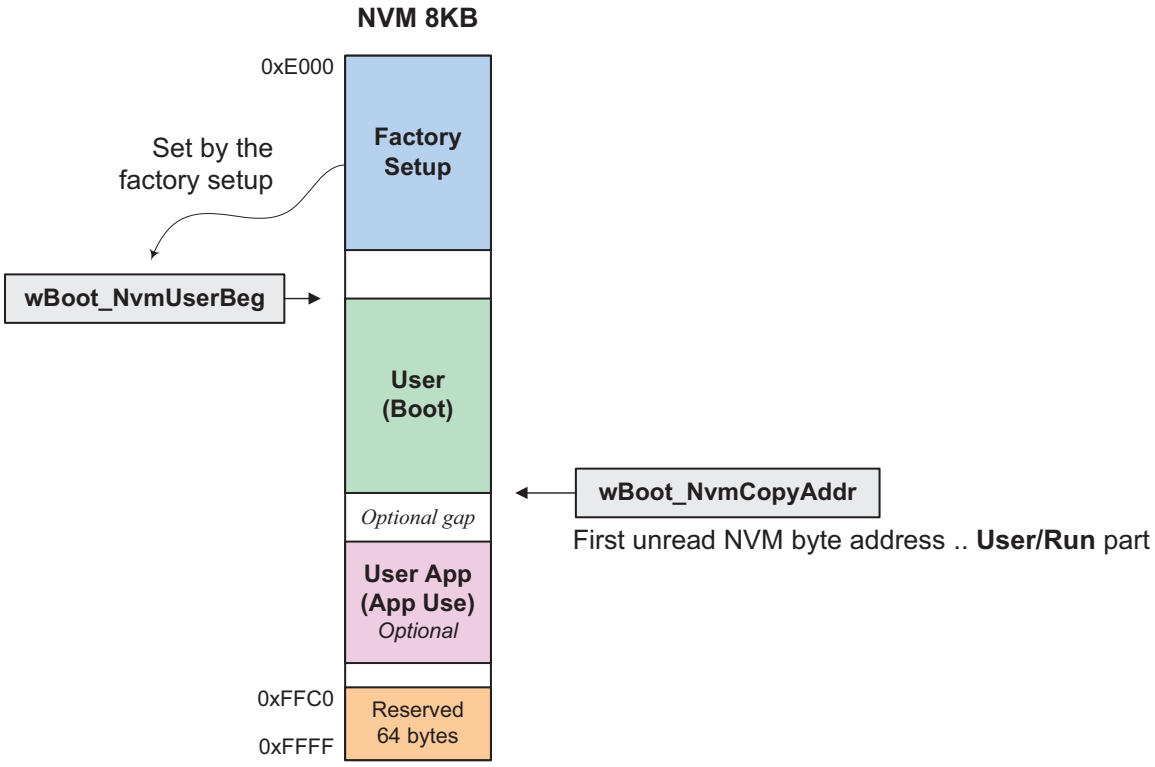


Figure 23.1. NVM Address Map

23.5. Device Boot Process

The boot process works in the following sequence:

1. Boot is invoked by cycling power to the internals of the chip (which includes power cycle to the whole chip) or waking up by button press, or by pressing a **Reset** button in the IDE development platform.
2. The device will read the **Factory** part of the NVM to determine the device configuration and load the configuration values to appropriate registers and CODE/XDATA memory locations. Part of this process is setting the boot variable block at the end of the CODE/XDATA memory.
3. If the program level is **Factory** then the boot process will stop and will not execute any code. It will wait in an infinite loop for the debugging chain to load a user application to CODE/XDATA RAM and to allow that code execution from the IDE. More specifically, the boot hardware waits for the CODE_RUN_POR or CODE_RUN_SYS bits to be set in the BOOT_FLAGS register. When using debugging chain and IDE, this is taken care of automatically by the IDE and there is no user intervention required.
4. If the program level is **User** then the same procedure is followed as for the **Factory** device. After that the boot procedure automatically (see Note 1.) continues to load **User** region from NVM to CODE/XDATA RAM and IRAM. After it finishes the device does not execute any code (see Note 2.) and goes to the same waiting infinite loop as described in item 3. for **Factory** device.
   The user can modify the boot behavior of the **User** part by controlling the following two bits:

   Note 1. BOOT_TRIM_POR bit in BOOT_FLAGS .. Register cleared on power on reset. If this bit is 1, the boot loader will not load the **User** load but enables C2 and goes to the **boot_flags** waiting loop. The part will behave as a **Factory** part. This bit has higher priority than the one below. Convenient for debugging until the power is cycled.

   Note 2. USER_CONT bit in PROT3_CTRL in NVM .. Bit in the NVM protection register. Once set it cannot be cleared. When this bit is 1, then after the Factory and User loads are loaded from

NVM the boot loader enables C2 and runs the user code immediately, without any wait, by executing long jump to RAM address 0x0000. The IDE can still halt the chip and connect to it in a usual fashion. From the debug point of view there is no change. This bit corresponds to the **Exe User Boot** checkout on the NVM programmer GUI application.

5. If the program level is **Run** then the same boot procedures is followed as for the **User** device. When loading the **User** region is done, the user code is run by jumping to the 0x0000 address in CODE/XDATA RAM. The C2 interface is disabled and the chip can no longer be used with debug chain and IDE. The **Run** chip can be opened for retest, but the user has an option to limit **Retest** access or lock the chip out completely. See Section “23.11. Retest and Retest Configuration”.

Note: If the **Factory** or **User** part is powered up, the part will wait in an infinite loop, consuming power. Only the **Run** part executes code in CODE/XDATA RAM automatically. The user can also optionally make the **User** part to execute loaded code automatically as described above.

23.6. Error Handling During Boot

At the end of the boot process the **bBoot_BootStat** byte variable contains the final status of the whole boot process. Bit field meanings are summarized in SFR Definition 23.1. The user application code should read that variable and if its value is other then 0x00 or 0x80, then it should decide whether it is safe to run the application at all. The boot success/fail single bit information is also contained in the BOOT_FLAGS SFR register for easier access.

23.7. CODE/XDATA RAM Address Map

The 4.5 kB for internal RAM at the address range 0x0000 .. 0x11FF is the main area for the user program (CODE) and external data (XDATA). It is a unified memory, referred to as CODE/XDATA RAM in this document, so both CPU code (CODE) can be executed there and external data (XDATA) can reside there. External data are the data accessible by MOVX instructions. MOVC instructions can also be used to access data in that region.

After the boot of a **Run** part the CPU starts executing code from address 0x0000 in RAM. Therefore, user code must occupy the beginning of the RAM, followed by the XDATA.

Important: Linker of the user application has to be given proper regions of CODE and XDATA memory, which are mutually exclusive. Therefore, for example, the user cannot set the CODE region to be 0x0000 .. 0x1000 and XDATA region to be the very same at the same time. One has to specify two non-overlapping regions for CODE and XDATA in the CODE/XDATA RAM area instead.

The end of the CODE/XDATA RAM is reserved for internal Silicon Labs use. The CODE/XDATA RAM address space is divided into three parts:

1. **User** CODE/XDATA .. user application load. The boot process copies the user code and external initialized data from NVM to this region.
2. **Factory** data values .. variable length. Reserved for Silicon Labs use. The actual beginning of the Silicon Labs reserved area in RAM can be obtained by reading the boot WORD (2 byte) variable **wBoot_DpramTrimBeg**. In big endian fashion it contains an address of the first reserved byte of the RAM. User can use the range 0x0000 .. (**wBoot_DpramTrimBeg**) - 1 for application CODE and XDATA
3. Boot status variables .. variables in the region 0x11F3 .. 0x11FF are boot status variables set at the end of the boot process to inform the user application about the RAM size available for user application and about the final status of the boot process.

The visual representation of the RAM is in Figure 23.2. The detailed explanation of the boot control data variables are in Table 23.1 to SFR Definition 23.1.

The user code or user development environment need to pay attention to the content of the following variables. All are stored in big endian fashion (MSB at the lower address):

- **wBoot_DpramTrimBeg** .. this variable points to the first occupied (by factory data) address of RAM. Therefore, the user development platform needs to read this variable to determine what the available RAM area for user CODE/XDATA is.
- **bBoot_BootStat** .. boot status result. User code should check this value at its beginning. If the value is different than 0x00 then the user could decide not to run its application since there was a problem with the boot.

Critical registers and variables corresponding to the NVM programming:

- **PROT0_CTRL** .. this register, described in SFR Definition 23.4, contains the value of the current program level of chip. Depending on that value, the NVM programming utility will decide what can and cannot be programmed into the NVM.
- **PROT3_CTRL** .. internal byte in the **Factory** region of the NVM controlling the boot process. It contains all the user code protection bits and modification of the User part boot process.
- **wBoot_NvmUserBeg** .. address in NVM of the beginning of the **User** load. For programming the User load into the NVM, the NVM programming utility has to be properly configured by using this value. The value is read automatically by the NVM programming utility, and also is available through the IDE. Depending on the size of the **Factory** load the value of this variable can vary in between chip revisions. It could also vary from chip to chip, but that is unlikely.
- **wBoot_NvmCopyAddr** .. first unread address of the NVM during boot. This address contains the NVM address the boot routine would read next. The last byte of the last data block read is at the address that is one less than the content of this variable: (**wBoot_NvmCopyAddr**) - 1. The NVM programmer will use this information when additional block **User** data is needed to be programmed. As long as the part is in a program state **User** additional blocks can be added to the **User** load.

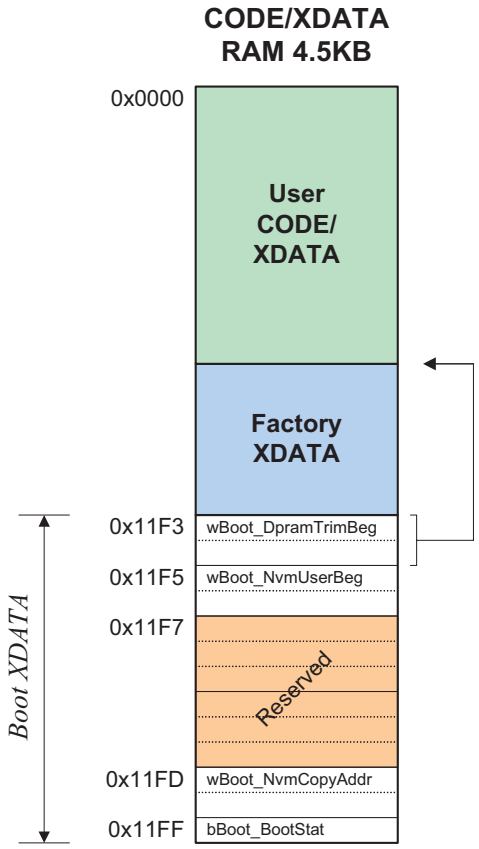


Figure 23.2. CODE/XDATA RAM Address Map

23.8. Boot Status Variables

End of the CODE/XDATA RAM are reserved for boot status variables.

The user must pay attention to the content of the **wBoot_DpramTrimBeg** variable. Its content points to the first reserved address for **Factory** Silicon Labs use.

Important: The CODE/XDATA area from this address on (increasing address) is reserved and must not be overwritten by **User** NVM load at boot time nor by user application at runtime.

If this area is accidentally overwritten by user application the chip will behave unpredictably. There is no hardware protection for this region.

Note that depending on the revision of the chip the **Factory** XDATA area can vary in size. The area is refreshed when reset is issued.

Table 23.1. Boot XDATA Status Variables

Register	Addr	Type	Description
wBoot_DpramTrimBeg	0x11F3	WORD*	Address of the first occupied byte by the Silicon Labs factory data in CODE/XDATA RAM. This variable is set after the boot. User must read the variable to determine where is the end of the usable CODE/XDATA RAM memory for user's use. The address is stored in big endian fashion; address MSB byte at the variable address location, followed by LSB byte on the next (address + 1) location.
wBoot_NvmUserBeg	0x11F5	WORD	Byte address of the first byte of the User load in the NVM memory. It is set by the Factory load. The User load MUST start at that address in NVM. Boot routine reads this variable before loading the User code after it finished loading the Factory load.
wBoot_NvmCopyAddr	0x11FD	WORD	First unread data address in NVM by the NVM copy routine **bNvm_CopyBlock**. After the boot is done this variable contains, in big endian, the NVM address of the first NVM byte not read by NVM copy routine. This is the first "empty" byte in NVM which is available for new data. The value of this variable is essential when the user wants to add more data to NVM later on.
bBoot_BootStat	0x11FF	BYTE	Boot status. User program can read this byte and decide whether the boot finished correctly. If not, then it can blink LED or not to continue with running the code. See the **bBoot_BootStat** bit description table.
***Note:** WORD is an unsigned 16 bit value, BYTE is an unsigned 8 bit value.			

Boot status byte can or should be read by the user application at the very beginning to determine whether the copying of the **Factory** and **User** data from NVM to desired RAM destination was successful or not. When there are no errors, the value the **bBoot_BootStat** variable should be 0x00 or 0x80. Any other value denotes a boot error. The user application then can decide whether to run or stall, if the user application was actually loaded to RAM. If the boot fails and the user application is not loaded to RAM, then unpredictable results may occur. The bit 7 of this variable contains a read value of GPIO[0] at the very beginning of the boot before the XO was optionally turned on.

SFR Definition 23.1. BOOT_BOOTSTAT

Bit	7	6	5	4	3	2	1	0
Name	BS_GPIO_XTAL	RESERVED		BS_ERR_FACTORY[2:0]			BS_ERR_USER_NEXT	BS_ERR_USER_FIRST
Type	R	R	R	R			R	R
Reset	0/1	0	0	0	0	0	0	0

XREG Address = 0x11FF

Bit	Name	Function
7	BS_GPIO_XTAL	**GPIO0 Read before Boot.** Read GPIO0 value at the very beginning of the boot prior to optionally turning on the XO (crystal oscillator).
6:5	Reserved	**Reserved.**
4:2	BS_ERR_FACTORY[2:0]	**Load of the Factory Data.** Load of the Factory data failed if value is other than 0x0
1	BS_ERR_USER_NEXT	**Load of the Second or Subsequent User block.** Load of the second or subsequent user block failed if other than 0.
0	BS_ERR_USER_FIRST	**Load of the First User block.** Load of the first user block failed if other than 0.

Apart from the CODE/XDATA RAM memory region there is a boot control and status SFR register, BOOT_FLAGS. It controls the end of the boot and has error status bit, which is set when **bBoot_BootStat** variable has other than 0x00 value. That is added for convenience so the user code can just check a single bit in SFR register rather than reading XDATA variable to determine whether boot finished successfully or not. If the **bBoot_BootStat** XDATA variable is not 0x00, the boot fail flag is set in the BOOT_FLAGS SFR.

The other bits control whether the user code will run after the boot. If the debugging chain is used and user code is loaded through IDE, this process is transparent to the user. Whenever the IDE connects to the device, it resets and halts the device, awaiting user. The user will generally not write to the **BOOT_FLAGS** register.

However, if the user wants to make the **User** part to behave as a **Factory** part, then it is possible to write value 0x20 to the BOOT_FLAGS register through IDE (see View -> Debug Windows -> SFR -> Boot window). Don't forget to press the **`Refresh`** IDE button for the change to take effect. Then until the power to the part is cycled the part would behave as a **Factory** part.

SFR Definition 23.2. BOOT_FLAGS

Bit	7	6	5	4	3	2	1	0
Name	Reserved	Reserved	BOOT_ TRIM_ POR	CODE_ RUN_ POR	Reserved	BOOT_ FAIL_ SYS	BOOT_ DONE_ SYS	CODE_ RUN_SYS
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xDD

Bit	Name	Function
7:6	Reserved	**Reserved.**
5	BOOT_TRIM_POR	**Force User Part to Act as a Factory Part.** For **User** part only: During the boot process load only **Factory** values and stop. By other words, act like a **Factory** part. Must be set for additional programming of the **User** part or for loading user test code to RAM when the part is programmed as **User** part. This bit has higher priority then the PROT3_CTRL.USER_CONT bit.
4	CODE_ RUN_ POR	**Run User Code in RAM.** Same functionality as CODE_RUN_SYS.
3	Reserved	**Reserved.**
2	BOOT_ FAIL_ SYS	**Boot Loading Process Failed.** This is an information flag, independent of the BOOT_DONE_SYS. This bit is set when the boot status XDATA variable **bBoot_BootStat** is not equal to 0x00, signalling error during boot. It is recommended that the user code reads this bit and possibly make decisions whether to continue with the execution of the loaded RAM code, which might not be complete, or signal to a user a problem, by, for example, blinking LED in some not-ordinary fashion.
1	BOOT_ DONE_ SYS	**Boot Routine Finished Flag.** Always set to 1 at the end of the boot.
0	CODE_ RUN_SYS	**Run User Code in RAM.** Used for Factory and User program states, ignored in Run state. When this bit is set the boot routine will jump to CODE address 0x0000. Forced by the debugging chain if the device is connected to the IDE.

23.9. Boot Routine Destination Address Space

The boot process reads the formatted data from NVM and writes it to the desired destination. The format supports different address regions based on the destination (write) address. The destination address is part of the NVM content data frame format.

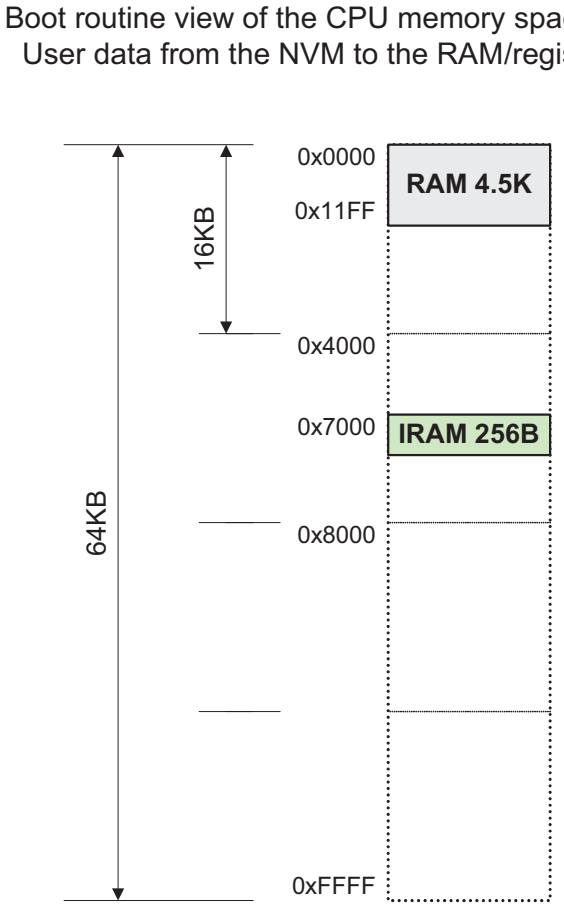


Figure 23.3. Boot Routine Destination CPU Address Space for Copy from NVM

The address space of the NVM image destinations depend on the program level of the chip and is shown in Figure 23.3:

- 0x0000 .. 0x11FF .. CODE/XDATA RAM. The end of the RAM is reserved for the boot control data.
- 0x7000 .. 0x70FF .. virtually mapped 256 byte of IRAM for DATA/IDATA indirect access. Whenever the destination address in the NVM image is in this region the data destination is going to be DATA/IDATA IRAM space. However, only region 0x7020 .. 0x70EF is writable. That means that the first 32 and last 16 bytes of the IRAM are not writable by a boot process. Note that the mapping is for indirect internal IRAM access (DATA/IDATA), so SFR registers cannot be initialized by this process.

It is up to the user to generate IntelHEX files to be passed to the NVM programmer. The NVM programmer will ensure that the NVM gets programmed with a proper data structures such that the data values provided in the IntelHEX files will appear at the RAM and IRAM addresses specified in the IntelHEX input file after the boot is done.

Note that by using the unified CODE/XDATA memory and by mapping the IRAM to the boot process address space the user can initialize both XDATA and IRAM variables directly from the **User** NVM load without the need for running any startup code to do variable initializations, resulting in the saving of a code size.

One application of the data initialization by a boot process could be copying of keys from the NVM to fixed locations without any code intervention. The user can program all the chips with the same application in the factory and then add only a very small, per chip, User block with keys, specifying where to the XDATA and/or IRAM memories the boot process should copy the values of the keys.

For example, to initialize IRAM location 0x56 to 0xA4 value the user will provide and IntelHEX file specifying that at the address 0x7056 the data value should be 0xA4.

23.10. NVM Programming

The NVM programming can be done only by the Silicon Labs provided data preparer and programmer. The data preparer will take user generated application IntelHEX files, user directives, and will generate data to be programmed into the NVM. The NVM programmer then programs the data into the NVM. In the end the NVM programming will be handled by a single stand alone application.

During the programming process the user will have control of the following:

1. Make **Factory** part a **User** part .. program User data into the NVM
2. Update **User** part .. add additional User data block to the existing User data already in NVM. This process can be done many times as long as there is a space in NVM.
3. Make **User** part a **Run** part .. mark a part as a final mission mode part. When making the part a Run part the user can decide whether the part retest will be allowed and if so, then what protection restrictions the user is going to impose during the retest process.

These steps can be combined into a single programming step. Step 2. is optional and is convenient when part specific data needs to be added later to the NVM load.

23.11. Retest and Retest Configuration

When the part is programmed as a **Run** part, the C2 interface is disabled and nobody can access the part externally. However, Silicon Labs needs to be able to retest the part in case it returns as a failed part from a customer application. Silicon Labs understands that customer may have programmed sensitive information into the NVM which should not be revealed to anybody, not even to Silicon Labs, during the retest process. During the process of making the part a **Run** part the user will have one time option to control the access to the chip during the retest process.

To be able to retest the fully programmed **Run** part, a special sequence of pin values needs to be applied at a particular time during the boot process. Once that sequence is recognized by the part, the boot process loads only the **Factory** region of the NVM and will not load any of the **User** regions from the NVM.

Then *before* the boot process opens the C2 interface for factory retest communication, it consults the user retest protection control flags programmed into the PROT3_CTRL byte in NVM when the part was made a **Run** part and acts on the values immediately. Only after all the actions prescribed by the flags settings are completed can the chip open for retest communication.

When making a **Run** part, the user can set the following retest protection flags when using the NVM programmer.

Note that if the bits are set into the PROT3_CTRL NVM byte before the part is programmed as **Run** part (for example, those bits are set when making a **User** part), the settings are ignored. The boot process will monitor these values only after the part is programmed to be the **Run** part.

Table 23.2. Run Chip Retest Protection Flags: NVM Programmer

Flag Name	Description
c2_off	Disable the C2 interface for good. No retest possible. **Warning**: When set then the part is locked out, C2 interface is disabled forever, and SiLabs cannot retest the chip. There is no back door to the part. All other settings below are ignored, since they have no effect. This bit is set in PROT0_CTRL.C2_OFF and it corresponds to **C2 Disable** checkbox on the NVM programmer GUI.
mem_c2_prot	Protect CODE/XDATA and IRAM RAM memories. When set then the boot process clears CODE/XDATA and IRAM RAM's when the **Run** chip is opened for retest. CODE/XDATA and IRAM RAM's get cleared with 0, excluding the **Factory** region at the end of CODE/XDATA. The IRAM gets also cleared completely outside of the register bank 0 (bottom 8 registers). This ensures that there is no lingering **User** code or data values, like keys, in any of the RAM's. This bit is in PROT3_CTRL.MEM_C2_PROT and it corresponds to **RAM Clear** checkbox on the NVM programmer GUI.

Table 23.2. Run Chip Retest Protection Flags: NVM Programmer

Flag Name	Description
mtp_c2_prot	Protect MTP. When set then both Wr and Rd access to MTP is disabled. Forces boot process to set MTP_PROT=1 to disable MTP communication completely. Reading from MTP returns 0x00 values, writing is not possible. Customer may want to set this option if there is a sensitive information written into the MTP EEPROM during the lifetime of the part. This bit is in PROT3_CTRL.MTP_C2_PROT and it corresponds to **MTP Disable** checkbox on the NVM programmer GUI.
nvm_c2_prot	Protect NVM. When set then both Wr and Rd access to NVM is disabled. It forces boot process to write NVM_PROT=1 at the end of the boot process to disable NVM access. This protects **User** load in NVM from being read by SiLabs. If this option is used then the SiLabs can still do the following with NVM content during retest: 1. Calculate CRC32 over the **Factory** region of NVM. 2. Calculate CRC32 over the user portion of the NVM, which is the whole NVM excluding the **Factory** region and the last 64 bytes of NVM. 3. Read the end 64 bytes of the NVM, which is a reserved NVM region for SiLabs use. When this option is set then SiLabs cannot do anything else with NVM during retest. This bit is in PROT3_CTRL.NVM_C2_PROT and it corresponds to **NVM Disable** checkbox on the NVM programmer GUI.

Once these options are programmed to the part they cannot be undone or changed. Additional setting of these options *after* the part is made a **Run** part is not possible either.

23.12. Boot and Retest Protection Control Register

The boot process monitors the value of an NVM byte called **PROT3_CTRL**. There is not a corresponding hardware register to this byte. It is a value in the **Factory** region at the beginning of NVM. The register contains **Retest** protection flags described above and modification of the boot for **User** part.

Each bit is write 1 once. Once the bit is programmed it cannot be cleared. The bits are programmable though the checkboxes in the NVM programmer. Once the bit is set, there is no way to monitor the current status of the bit in the PROT3_CTRL NVM byte on the device.

SFR Definition 23.3. PROT3_CTRL

Bit	7	6	5	4	3	2	1	0
Name	NVM_C2_PROT	MTP_C2_PROT	MEM_C2_PROT	BOOT_XO_ENA	Reserved		USER_CONT	Reserved
Type	W	W	W	W	R		W	R
Reset	0	0	0	0	0x0		0	0

SFR Address = 0xDA

Bit	Name	Function
7	NVM_C2_PROT	**NVM Protection (Disable) When Entering Retest Mode.** This bit corresponds to **NVM Disable** checkbox on the NVM programmer GUI.
6	MTP_C2_PROT	**MTP Protection (Disable) When Entering Retest Mode.** This bit corresponds to **MTP Disable** checkbox on the NVM programmer GUI.
5	MEM_C2_PROT	**RAM Clearing (Content Protection) When Entering Retest Mode.** This bit corresponds to **RAM Clear** checkbox on the NVM programmer GUI.
4	BOOT_XO_ENA	**Enable the Crystal Oscillator (XO) at the Beginning of the Boot Process.** This is valid in any device programming level, including **Factory**. Since it can take up to 10ms for the XO to stabilize and about 3.6 ms to load 1 kB of data from NVM to RAM, the user may decide to enable the XO at the beginning of the boot process so the XO will be stabilizing while the device is going through the boot process to save time in the main application. This bit corresponds to **XO Early Enable** checkbox on the NVM programmer GUI.
2:3	Reserved	**Reserved.**
1	USER_CONT	**Run the User Code in User Part after Boot Automatically.** For **User** programming level only, has no effect in other programming levels. Normally when the part is programmed as User the user code is loaded from NVM to RAM, but is not executed automatically. If this bit is set, then the user load is executed automatically after boot. This bit corresponds to **Exe User Boot** checkbox on the NVM programmer GUI.
0	Reserved	**Reserved.**

23.13. Chip Protection Control Register

The boot process sets the values of the device protection and configuration SFR register, **PROT0_CTRL**. The user can read the register and check the programming level of the device as well as protections set to control access to the NVM and MTP memories and C2 interface. The register is user writable, but once a value of 1 is written to any of the bits in the register it cannot be undone. Protections can only be made stronger, not weaker.

SFR Definition 23.4. PROT0_CTRL

Bit	7	6	5	4	3	2	1	0
Name	NVM_ PROT	C2_OFF	Reserved	MTP_ PROT	NVM_ WR_ PROT	NVM_BLOWN[2:0]		
Type	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Reset	0	0	1	0	0	0	0	0

SFR Address = 0xDA

Bit	Name	Function
7	NVM_ PROT	**NVM Protection.** Disable NVM access completely. Neither read nor write to NVM is possible. Write 1 sets the bit, write 0 has no effect.
6	C2_OFF	**C2 Interface Disable.** Write 1 sets the bit, write 0 has no effect. This bit is reset by the main digital power on reset. Power has to be cycled to reset this bit or chip has to wake up from shutdown. If C2 is disabled then the chip is not accessible by a debug chain and not available for retest.
5	Reserved	**Reserved.**
4	MTP_ PROT	**MTP Protection.** Disable MTP access. If set then MTP will be completely disabled. All reads from MTP will be 0x00. Write 1 sets the bit, write 0 has no effect.
3	NVM_ WR_ PROT	**NVM Write Protection.** If this bit is set the NVM is write protected. However, the value is used only if the chip program level is **Run,** NVM_BLOWN=3'b11x. In all other cases the value of this bit is ignored.
2:0	NVM_ BLOWN [2:0]	**Displays Chip Program Level.** The bits can only be set to 1, write 0 has no effect: 001 .. **Factory** 011 .. **User** 111 .. **Run**

24. On-Chip Registers

There are two register regions on chip:

- Special Function Registers region
- XREG region

24.1. Special Function Registers

The direct-access data memory locations from 0x80 to 0xFF constitute the special function registers (SFRs). The SFRs provide control and data exchange with the Si4010's resources and peripherals. The CIP-51 controller core duplicates the SFRs found in a typical 8051 implementation as well as implementing additional SFRs used to configure and access the sub-systems unique to the Si4010. This allows the addition of new functionality while retaining compatibility with the MCS-51™ instruction set. Table 24.2 lists the SFRs implemented in the Si4010 device family.

The SFR registers are accessed anytime the direct addressing mode is used to access memory locations from 0x80 to 0xFF. SFRs with addresses ending in 0x0 or 0x8 (e.g. P0, P1, ACC, IE, etc.) are bit-addressable as well as byte-addressable. All other SFRs are byte-addressable only. Unoccupied addresses in the SFR space are reserved for future use. Accessing these areas will have an indeterminate effect and should be avoided. Refer to the corresponding pages of the data sheet, as indicated in Table 24.2, for a detailed description of each register.

Table 24.1. Special Function Register (SFR) Memory Map

	0X80	0x90	0xA0	0xB0	0xC0	0xD0	0xE0	0xF0
0*	P0*	P1*	P2*			PSW*	ACC*	B*
1	SP			GPR_CTRL				
2	DPL			GPR_DATA				
3	DPH							
4	GFM_DATA		P0CON				LC_FSK	
5	GFM_CONST		P1CON	PORT_CTRL				
6	SBOX_DATA			PORT_SET			EIE1	EIP1
7	PCON			PORT_INTCFG				
8*			IE*	IP*	TMR2CTRL*			
9		RBIT_DATA	ODS_CTRL	TMR3CTRL	TMR_CLKSEL			
A			ODS_TIMING	TMR3RL	TMR2RL	PROT0_CTRL		
B		FC_CTRL	ODS_DATA	TMR3RH	TMR2RH			
C		RTC_CTRL	ODS_RATEL	TMR3L	TMR2L			
D		FC_INTERVAL	ODS_RATEH	TMR3H	TMR2H	BOOT_FLAGS		
E			ODS_WARM1	SYSGEN	PA_LVL		SYS_SET	
F	CLKOUT_SET		ODS_WARMS2	INT_FLAGS				

***Notes:** Bit addressable registers.

Table 24.2. Special Function Registers

SFRs are listed in alphabetical order. All undefined SFR locations are reserved

Register	Address	Description	Page
ACC	0xE0	Accumulator	68
B	0xF0	B Register	69
BOOT_FLAGS	0xDD	Boot Flags	83
CLKOUT_SET	0x8F	Clock Output Settings	128
DPH	0x83	Data Pointer High	67
DPL	0x82	Data Pointer Low	67
EIE1	0xE6	Extended Interrupt Enable 1	101
EIP1	0xF6	Extended Interrupt Priority 1	102
FC_CTRL	0x9B	Frequency Counter Control	58
FC_INTERVAL	0x9D	Frequency Counter Interval	59
GFM_CONST	0x85	AES GFM Multiplier Constant	109
GFM_DATA	0x84	AES GFM Data	109
GPR_CTRL	0xB1	General Purpose Control Register	130
GPR_DATA	0xB2	General Purpose Data Register	130
IE	0xA8	Interrupt Enable	99
IP	0xB8	Interrupt Priority	100
INT_FLAGS	0xBF	Interrupt Flags	103
LC_FSK	0xE4	LC FSK Deviation	52
ODS_CTRL	0xA9	ODS Control	47
ODS_DATA	0xAB	ODS Data	49
ODS_RATEH	0xAD	ODS Rate High Byte	50
ODS_RATEL	0xAC	ODS Rate Low Byte	49
ODS_TIMING	0xAA	ODS Timing Register	48
ODS_WARM1	0xAE	ODS Warm up times for PA and Divider	50
ODS_WARM2	0xAF	ODS Warm up time for LCOSC	51
P0	0x80	Port 0 Latch	122
P0CON	0xA4	Port 0 Configuration	123
P1	0x90	Port 1 Latch	123
P1CON	0xA5	Port 1 Configuration	124
P2	0xA0	Port 2 Latch	124
PA_LVL	0xCE	Power Amplifier Level	44
PCON	0x87	Power Control	107
PORT_CTRL	0xB5	Port Control	125

Table 24.2. Special Function Registers (Continued)

SFRs are listed in alphabetical order. All undefined SFR locations are reserved

Register	Address	Description	Page
PORT_INTCFG	0xB7	Port Interrupt Configuration	105
PORT_SET	0xB6	Port Set	126
PROT0_CTRL	0xDA	Protection 0 Control	89
PSW	0xD0	Program Status Word	70
RBIT_DATA	0x99	Read Bit Data	131
RTC_CTRL	0x9C	Real Time Clock Control	134
SBOX_DATA	0x86	AES SBOX Data	110
SP	0x81	Stack Pointer	68
SYSGEN	0xBE	System Generator Register	54
TMR2CTRL	0xC8	Timer/Counter 2 Control	144
TMR2H	0xCD	Timer/Counter 2 High	147
TMR2L	0xCC	Timer/Counter 2 Low	147
TMR2RH	0xCB	Timer/Counter 2 Reload High	146
TMR2RL	0xCA	Timer/Counter 2 Reload Low	146
TMR3CTRL	0xB9	Timer/Counter 3Control	148
TMR3H	0xBD	Timer/Counter 3 High	151
TMR3L	0xBC	Timer/Counter 3Low	151
TMR3RH	0xBB	Timer/Counter 3 Reload High	151
TMR3RL	0xBA	Timer/Counter 3 Reload Low	150
TMR_CLKSEL	0xC9	Timer Source Clock Selection	143

24.2. XREG Registers

The chip contains another set of registers implemented in the **XREG** memory area. These registers are located in the XDATA address space, addressable by MOVX instructions only. From CPU perspective it is a regular external memory.

The advantage of the XREG registers is that they are viewed by the CPU as a regular memory. Therefore, they can be declared as different data types, structures, array of bytes, and so on. With SFR we only have special registers and it is not possible to declare them as long integers, for example. On the other hand the SFR register access is faster and one can use arithmetic and logical operations on them.

Note registers in the XREG regions are aligned at 8, 16, and 32 bit boundaries and they are stored in **big** endian fashion. This is to support Keil C compiler, which uses **big endian**. Note that if the register is, say 23 bits wide, the 32 bits (4 bytes) are allocated for the register and the register is aligned in big endian fashion.

Therefore, the LSB byte of the register will be at the address <*reg_addr*> + 3, while the byte directly at the <*reg_addr*> is the MSB byte and is empty (read as 0x0), since the register itself is only 23 bits wide. Table 24.3 shows a memory map of the XREG registers in the external memory space.

Table 24.3. XREG Register Memory Map in External Memory

XDATA Address	Type	Name	
0x4002	BYTE	bLPOSC_TRIM	
0x4003 ... 0x4007		<reserved>	
0x4008 0x4009 0x400a 0x400b	LWORD	IFC_COUNT	MSB Byte LSB Byte
0x400c 0x400d	WORD	wPA_CAP	MSB Byte LSB Byte
0x400e ... 0x4011		<reserved>	
0x4012	BYTE	bPA_TRIM	
0x4013 ... 0x4015		<reserved>	
0x4016	BYTE	bXO_CTRL	
0x4017	BYTE	bPORT_TST	
0x4018 ... 0x4026		<reserved>	
0x4040 ... 0x404f	BYTE	abMTP_RDATA[16]	Byte [0] Byte [15]

Note: Multiple byte variables, if they are not arrays, are stored in **big endian ..** MSB byte stored on lower address. Arrays are stored with byte index [0] at lower address.

Table 24.4. XREG Registers

XREGs are listed in alphabetical order.

Register	Address	Description	Page
wPA_CAP	0x400C	PA Variable Capacitor	44
bPA_TRIM	0x4012	PA MAX Drive bit	45
bLPOSC_TRIM	0x4002	Low Power Oscillator Trim	53
abMTP_RDATA[16]	0X4040	MTP_Read Data Bytes	74
bXO_CTRL	0x4016	XO Control	55
IFC_COUNT	0x4008	Frequency Counter Output	59

Description of the XREG register fields on the previous pages includes only the used register bits. The fields are aligned towards the LSB byte of the XREG register. If the actual XREG register is wider then the field described the missing bits towards MSB byte are all read as 0's and writing to them has no effect. For example, the register wPA_CAP contains a single 9 bit field. Since it is more than 8 bits and less then 16 it occupies two bytes. That's why the prefix letter 'w' denoting a two byte WORD. The bits [15:9] are read as all zeros and write has no effect. They are aligned towards MSB byte of the wPA_CAP, the one at lower address since the byte ordering is in big endian fashion.

25. Interrupts

The Si4010 device includes an extended interrupt system supporting a total of 12 interrupt sources with two priority levels. Each interrupt source has one or more associated interrupt-pending flag(s) located in an SFR. When a peripheral or external source meets a valid interrupt condition, the associated interrupt-pending flag is set to logic ‘1’.

If interrupts are enabled for the source, an interrupt request is generated when the interrupt-pending flag is set. As soon as execution of the current instruction is complete, the CPU generates an LCALL to a predetermined address to begin execution of an interrupt service routine (ISR). Each ISR must end with an RETI instruction, which returns program execution to the next instruction that would have been executed if the interrupt request had not occurred. If interrupts are not enabled, the interrupt-pending flag is ignored by the hardware and program execution continues as normal. The interrupt-pending flag is set to logic ‘1’ regardless of the interrupt's enable/disable state.

Each interrupt source can be individually enabled or disabled through the use of an associated interrupt enable bit in the Interrupt Enable and Extended Interrupt Enable SFRs. However, interrupts must first be globally enabled by setting the EA bit (IE.7) to logic ‘1’ before the individual interrupt enables are recognized.

Setting the EA bit to logic ‘0’ disables all interrupt sources regardless of the individual interrupt-enable settings. Note that interrupts which occur when the EA bit is set to logic ‘0’ will be held in a pending state, and will not be serviced until the EA bit is set back to logic ‘1’.

Note: Any instruction that clears a bit to disable an interrupt should be immediately followed by an instruction that has two or more opcode bytes. Using EA (global interrupt enable) as an example:

```
// in 'C':
   EA = 0; // clear EA bit.
   EA = 0; // this is a dummy instruction with two-byte opcode.

; in assembly:
   CLR EA ; clear EA bit.
   CLR EA ; this is a dummy instruction with two-byte opcode.
```

For example, if an interrupt is posted during the execution phase of a "CLR EA" opcode (or any instruction which clears a bit to disable an interrupt source), and the instruction is followed by a single-cycle instruction, the interrupt may be taken. However, a read of the enable bit will return a '0' inside the interrupt service routine. When the bit-clearing opcode is followed by a multi-cycle instruction, the interrupt will not be taken.

On this device no interrupt-pending flags are automatically cleared by the hardware when the CPU vectors to the ISR. The flags must be cleared by software before returning from the ISR. If an interrupt-pending flag remains set after the CPU completes the return-from-interrupt (RETI) instruction, a new interrupt request will be generated immediately and the CPU will re-enter the ISR after the completion of the next instruction.

25.1. MCU Interrupt Sources and Vectors

The device supports 12 interrupt sources. Software can simulate an interrupt by setting any interrupt-pending flag to logic ‘1’. If interrupts are enabled for the flag, an interrupt request will be generated and the CPU will vector to the ISR address associated with the interrupt-pending flag. MCU interrupt sources, associated vector addresses, priority order, and control bits are summarized in Table 25.1. Refer to the data sheet section associated with a particular on-chip peripheral for information regarding valid interrupt conditions for the peripheral and the behavior of its interrupt-pending flag(s).

25.2. Interrupt Priorities

Each interrupt source can be individually programmed to one of two priority levels: low or high. A low priority interrupt service routine can be preempted by a high priority interrupt. A high priority interrupt cannot be preempted. Each interrupt has an associated interrupt priority bit in an SFR (IP or EIP1) used to configure its priority level. Low priority is the default. If two interrupts are recognized simultaneously, the interrupt with the higher priority is serviced first. If both interrupts have the same priority level, a fixed priority order is used to arbitrate, given in Table 25.1.

25.3. Interrupt Latency

Interrupt response time depends on the state of the CPU when the interrupt occurs. Pending interrupts are sampled and priority decoded each system clock cycle. Therefore, the fastest possible response time is 5 system clock cycles: 1 clock cycle to detect the interrupt and 4 clock cycles to complete the LCALL to the ISR. Additional clock cycles will be required if a chace miss occurs. If an interrupt is pending when a RETI is executed, a single instruction is executed before an LCALL is made to service the pending interrupt. Therefore, the maximum response time for an interrupt (when no other interrupt is currently being serviced or the new interrupt is of greater priority) is when the CPU is performing an RETI instruction followed by a DIV as the next instruction, and a cache miss event also occurs. If the CPU is executing an ISR for an interrupt with equal or higher priority, the new interrupt will not be serviced until the current ISR completes, including the RETI and following instruction.

Table 25.1. Interrupt Summary

Interrupt Source	Interrupt Vector	Priority Order	Pending Flag	Bit addressable?	Enable Flag	Priority Control
Reset	0x0000	Top	None	N/A	Always Enabled	Always Highest
External INT 0 (INT0)	0x0003	0	INT0_FLAG (INT_FLAGS.0)	N	EINT0 (IE.0)	PINT0 (IP.0)
Timer 2 Overflow	0x000B	1	TMR2INTL (TMR2CTRL.6) TMR2INTH (TMR2CTRL.7)	Y	ETMR2 (IE.1)	PTMR2 (IP.1)
RESERVED	0x0013	2	N/A	N	N/A	N/A
Real Time Clock Tick	0x001B	3	RTC_INT (RTC_CTRL.7)	N	ERTC (IE.3)	PRTC (IP.3)
ODS Ready for Data	0x0023	4	ODS_FLAG (INT_FLAGS.2)	N	EODS (IE.4)	PODS (IP.4)
Timer 3 Overflow	0x002B	5	TMR3INTL (TMR3CTRL.6) TMR3INTH (TMR3CTRL.7)	N	ETMR3 (IE.5)	PTMR3 (IP.5)
External INT1	0x0033	6	INT1_FLAG (INT_FLAGS.1)	N	EINT1 (IE.6)	PINT1 (IP.6)
Reserved	0x003B	7	N/A	N/A	N/A	N/A
Reserved	0x0043	8	N/A	N/A	N/A	N/A
Frequency Counter Count Done	0x004B	9	FC_DONE (FC_CTRL.7)	N	EFC (EIE1.2)	PFC (EIP1.2)
Software Source 0 (RESERVED)	0x0053	10	VOID0_FLAG (INT_FLAGS.3)	N	EVOID0 (EIE1.3)	PVOID0 (EIP1.3)
Software Source 1 (RESERVED)	0x005B	11	VOID1_FLAG (INT_FLAGS.4)	N	EVOID1 (EIE1.4)	PVOID1 (EIP1.4)

25.4. Interrupt Register Descriptions

The SFRs used to enable the interrupt sources and set their priority level are described in this section. Refer to the data sheet section associated with a particular on-chip peripheral for information regarding valid interrupt conditions for the peripheral and the behavior of its interrupt-pending flag(s).

SFR Definition 25.1. IE

Bit	7	6	5	4	3	2	1	0
Name	EA	EINT1	ETMR3	EODS	ERTC	Reserved	ETMR2	EINT0
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xA8; Bit-Addressable

Bit	Name	Function
7	EA	**Enable All Interrupts.** Globally enables/disables all interrupts. It overrides individual interrupt mask settings. 0: Disable all interrupt sources. 1: Enable each interrupt according to its individual mask setting.
6	EINT1	**Enable External Edge Interrupt 1.** This bit sets the masking of External Interrupt 1. 0: Disable external interrupt 1. 1: Enable interrupt requests generated by the INT1 input.
5	ETMR3	**Enable Timer 3 Interrupt.** This bit sets the masking of the Timer 3 interrupt. 0: Disable Timer 3 interrupt. 1: Enable interrupt requests generated by the TF3L or TF3H flags.
4	EODS	**Enable Output Data Serializer Interrupt.** This bit sets the masking of the ODS interrupt. 0: Disable ODS interrupt. 1: Enable ODS interrupt.
3	ERTC	**Enable Real Time Clock Interrupt.** This bit sets the masking of the RTC interrupt. 0: Disable all RTC interrupt. 1: Enable RTC interrupt.
2	Reserved	Do not write 1 to this bit.
1	ETMR2	**Enable Timer 2 Interrupt.** This bit sets the masking of the Timer 2 interrupt. 0: Disable all Timer 2 interrupt. 1: Enable interrupt requests generated by the TF2 flag.
0	EINT0	**Enable External Edge Interrupt 0.** This bit sets the masking of External Interrupt 0. 0: Disable external interrupt 0. 1: Enable interrupt requests generated by the INT0 input.

SFR Definition 25.2. IP

Bit	7	6	5	4	3	2	1	0
Name	Reserved	PINT1	PTMR3	PODS	PRTC	Reserved	PTMR2	PINT0
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	0	0	0	0	0	0	0

SFR Address = 0xB8; Bit-Addressable

Bit	Name	Function
7	Reserved	Read = 1, Write = Don't Care.
6	PINT1	**External Edge Interrupt 1 Priority Control.** This bit sets the priority of the External Interrupt 1 interrupt. 0: External Interrupt 1 set to low priority level. 1: External Interrupt 1 set to high priority level.
5	PTMR3	**Timer 3 Interrupt Priority Control.** This bit sets the priority of the Timer 3 interrupt. 0: Timer 3 interrupt set to low priority level. 1: Timer 3 interrupt set to high priority level.
4	PODS	**Output Data Serializer Interrupt Priority Control.** This bit sets the priority of the ODS interrupt. 0: ODS interrupt set to low priority level. 1: ODS interrupt set to high priority level.
3	PRTC	**Real Time Clock Interrupt Priority Control.** This bit sets the priority of the RTC interrupt. 0: RTC interrupt set to low priority level. 1: RTC interrupt set to high priority level.
2	Reserved	Do not write 1 to this bit.
1	PTMR2	**Timer 2 Interrupt Priority Control.** This bit sets the priority of the Timer 2 interrupt. 0: Timer 2 interrupt set to low priority level. 1: Timer 2 interrupt set to high priority level.
0	PINT0	**External Edge Interrupt 0 Priority Control.** This bit sets the priority of the External Interrupt 0 interrupt. 0: External Interrupt 0 set to low priority level. 1: External Interrupt 0 set to high priority level.

SFR Definition 25.3. EIE1

Bit	7	6	5	4	3	2	1	0
Name	Reserved	Reserved	Reserved	EVOID1	EVOID0	EFC	Reserved	Reserved
Type	R	R	R	R/W	R/W	R/W	R	R
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xE6

Bit	Name	Function
7:5	Reserved	Read as 0x0. Write has no effect.
4	EVOID1	**Enable VOID1 Interrupt (Reserved).** This bit sets the VOID1 interrupt.(Reserved) 0: Disable VOID1 interrupts. 1: Enable interrupt requests generated by VOID1 flags (Reserved).
3	EVOID2	**Enable VOID2 Interrupt (Reserved).** This bit sets the VOID2 interrupt.(Reserved) 0: Disable VOID2 interrupts. 1: Enable interrupt requests generated by VOID2 flags (Reserved).
2	EFC	**Enable Frequency Counter Interrupt.** This bit sets the Frequency Counter interrupt. 0: Disable Frequency Counter interrupt. 1: Enable interrupt requests generated by Frequency Counter.
1:0	Reserved	Reset value 0x0 must not be changed.

SFR Definition 25.4. EIP1

Bit	7	6	5	4	3	2	1	0
Name	Reserved			PVOID1	PVOID0	PFC	Reserved	
Type	R			R/W	R/W	R/W	R/W	
Reset	0			0	0	0	0	

SFR Address = 0xF6

Bit	Name	Function
7:5	Reserved	Read as 0x0. Write has no effect.
4	PVOID1	**VOID1 Interrupt Priority Control.** This bit sets the priority of the VOID1 interrupt. 0: VOID1 interrupt set to low priority level. 1: VOID1 interrupt set to high priority level.
3	PVOID0	**VOID0 Interrupt Priority Control.** This bit sets the priority of the VOID0 interrupt. 0: VOID0 interrupt set to low priority level. 1: VOID0 interrupt set to high priority level.
2	PFC	**Frequency Counter Interrupt Priority Control.** This bit sets the priority of the Frequency Counter interrupt. 0: Frequency Counter interrupt set to low priority level. 1: Frequency Counter interrupt set to high priority level.
1:0	Reserved	Reset value 0x0 must not be changed.

SFR Definition 25.5. INT_FLAGS

Bit	7	6	5	4	3	2	1	0
Name	Reserved	Reserved	Reserved	VOID1_ FLAG	VOID0_ FLAG	ODS_ FLAG	INT1_ FLAG	INT0_ FLAG
Type	R	R	R	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xBF

Bit	Name	Function
7:5	Reserved	Read as 0x0. Write has no effect.
4	VOID1_ FLAG	**Spare Interrupt Flag for Future Hardware Expansion.** Interrupt can be invoked by software only by writing 1 here. Test use only.
3	VOID0_ FLAG	**Spare Interrupt Flag for Future Hardware Expansion.** Interrupt can be invoked by software only by writing 1 here. Test use only.
2	ODS_ FLAG	**Set when TX Data Holding Register becomes Empty.** It must be cleared by software BEFORE writing a new byte into the ODS Tx data register. Hardware will not clear this bit.
1	INT1_ FLAG	**Set by Selected GPIO Input by a Selected Edge.** It gets set irrespective of the EINT0 setting. It must be cleared by software. Hardware will not clear this bit.
0	INT0_ FLAG	**Set by Selected GPIO Input by a Selected Edge.** It gets set irrespective of the EINT0 setting. It must be cleared by software. Hardware will not clear this bit.

25.5. External Interrupts

The INT0 and INT1 external interrupt sources are configurable as active high or low. They are edge sensitive only, not level sensitive. These are not the same INT0 and INT1 as found on original 8051 architecture.

Each of the INT0 and INT1 can invoke interrupt on the rising edge, falling edge, or both edges of the selected GPIO pins associated with the INT0 and INT1, respectively.

The single edge or double edge feature is controlled by the EDGE_INT0 and EDGE_INT1 bits in the PORT_SET register. The edge polarity is defined in the PORT_INTCFG register.

INT0 and INT1 are assigned to Port pins as defined in the PORT_INTCFG register. Note that the corresponding pending flag for INT0 or INT1 is **not** automatically cleared by the hardware when the CPU vectors to the ISR. This is a departure from the original 8051 architecture where if external interrupts were configured to be edge sensitive the corresponding interrupt flag was cleared by hardware upon the exit from the ISR routine.

The detection of the edges of INT0 and INT1 sources is done by sampling the associated port inputs by the internal system clock. Therefore, the edge detector will miss pulses shorter than 2 periods of the internal system clock periods. Note that the internal system clock frequency is programmable and can be as low as 24 MHz/128. It is up to the user to recognize possible external interrupt delays associated with sampling of the INT0 and INT1 by the system clock at the current, user selected, clock frequency.

The INT1 and INT0 internal signals are also used as capture event signals for timer 3 and 2, respectively, if they are running in capture mode.

SFR Definition 25.6. PORT_INTCFG

Bit	7	6	5	4	3	2	1	0
Name	NEG_ INT1	SEL_INT1[2:0]			NEG_ INT0	SEL_INT0[2:0]		
Type	R/W	R/W			R/W	R/W		
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xB7

Bit	Name	Function
7	NEG_ INT1	**Negative INT1 Polarity.** This bit selects whether the selected INT1 GPIO input will get inverted or pass as is before going to the edge detector. Note the edge detector detects either the rising edge or both. The mode is selectable by EDGE_INT1 bit is separate register. 0: Inverts the selected GPIO. 1: Pass the selected GPIO unchanged.
6:4	SEL_ INT1[2:0]	**INT1 Port Pin Selection Bits.** These bits select which Port pin is assigned to INT1. 000: Select GPIO0 001: Select GPIO1 010: Select GPIO2 011: Select GPIO3 100: Select GPIO4 101: Select **GPIO9** 110: Select GPIO6 111: Select GPIO7
3	NEG_ INT0	**Negative INT0 Polarity.** This bit selects whether the selected INT0 GPIO input will get inverted or pass as is before going to the edge detector. Note the edge detector detects either the rising edge or both. The mode is selectable by EDGE_INT0 bit is separate register. 0: Inverts the selected GPIO. 1: Pass the selected GPIO unchanged.
2:0	SEL_ INT0[2:0]	**INT0 Port Pin Selection Bits.** These bits select which Port pin is assigned to INT0. 000: Select GPIO0 001: Select GPIO1 010: Select GPIO2 011: Select GPIO3 100: Select GPIO4 101: Select **GPIO8** 110: Select GPIO6 111: Select GPIO7

26. Power Management Modes

The CIP-51 core has two software programmable power management modes: **Idle** and **Stop**. **Idle** mode halts the CPU while leaving the external peripherals and internal clocks active. In **Stop** mode, the CPU is halted, all interrupts and timers are inactive. The system clock is still running when the CPU is in **Stop** mode. Since clocks are running, power consumption is dependent upon the system clock frequency and the number of peripherals left in active mode before entering **Idle** or **Stop**. See the SFR definition of the Power Control Register (PCON) used to control the CPU power management modes.

Although the CIP-51 has **Idle** and **Stop** modes built in (as with any standard 8051 architecture), power management of the entire MCU is better accomplished by enabling/disabling individual peripherals as needed. Each analog peripheral can be disabled when not in use and put into low power mode. Digital peripherals, such as timers, draw little power whenever they are not in use.

The devices feature an additional shutdown mode, which shuts the device down. The device then can be woken up by pulling GPIO input to ground. See other sections for details.

26.1. Idle Mode

Setting the Idle Mode Select bit (PCON.0) causes the CIP-51 to halt the CPU and enter **Idle** mode as soon as the instruction that sets the bit completes. All internal registers and memory maintain their original data. All analog and digital peripherals can remain active during **Idle** mode.

Idle mode is terminated when an enabled interrupt or reset is asserted. The assertion of an enabled interrupt will cause the Idle Mode Selection bit (PCON.0) to be cleared and the CPU to resume operation. The pending interrupt will be serviced and the next instruction to be executed after the return from interrupt (RETI) will be the instruction immediately following the one that set the Idle Mode Select bit. If **Idle** mode is terminated by an external reset, the CIP-51 performs a normal reset sequence.

Note: Any instruction which sets the IDLE bit should be immediately followed by an instruction which has two or more opcode bytes. For example:

In C:

```
PCON |= 0x01; // Set IDLE bit

PCON = PCON; // ... Followed by a 3-cycle Dummy Instruction;
```

In assembly:

```
ORL PCON, #01h ; Set IDLE bit

MOV PCON, PCON ; ... Followed by a 3-cycle Dummy Instruction
```

If the instruction following the write to the IDLE bit is a single-byte instruction and an interrupt occurs during the execution of the instruction of the instruction which sets the IDLE bit, the CPU may not wake from IDLE mode when a future interrupt occurs.

26.2. Stop Mode

Setting the Stop Mode Select bit (PCON.1) causes the CIP-51 to enter **Stop** mode as soon as the instruction that sets the bit completes. In **Stop** mode, the CPU is stopped, effectively shutting down all digital peripherals. Each analog peripheral must be shut down individually prior to entering **Stop** mode. **Stop** mode can only be terminated by an external reset. On reset, the CIP-51 performs the normal reset sequence and begins program execution based on the program level of the chip.

The system clock is not stopped when in **Stop** mode.

SFR Definition 26.1. PCON

Bit	7	6	5	4	3	2	1	0
Name	GF[5:0]						STOP	IDLE
Type	R/W						R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0x87

Bit	Name	Function
7:2	GF[5:0]	**General Purpose Flags 5–0.** These are general purpose flags for use under software control.
1	STOP	**Stop Mode Select.** Setting this bit will place the CIP-51 in Stop mode. This bit will always be read as 0. 1: CPU goes into Stop mode (internal oscillator stopped).
0	IDLE	**Idle Mode Select.** Setting this bit will place the CIP-51 in Idle mode. This bit will always be read as 0. 1: CPU goes into Idle mode. (Shuts off clock to CPU, but clock to Timers, Interrupts, Serial Ports, and Analog Peripherals are still active.)

27. AES Hardware Accelerator

The device implements the AES (Advanced Encryption Standard) hardware accelerator. It is not a full hardware solution. The hardware accelerator is used by the Silicon Labs API firmware to implement AES 128 bit encrypt/decrypt functions. If the user wants to implement proprietary AES implementation in firmware it is possible to use the AES hardware accelerator.

The accelerator has two parts:

1. AES Galois field (GF) hardware multiplier
2. AES SBox/Inverse SBox hardware module

The Galois field multiplier is designed to multiply two AES Galois field 8-bit elements, even though the AES just multiplies values by a constant. It is up to the firmware to setup the constant and data to multiply.

The hardware implements efficient SBox/Inverse SBox data processing.

Consult the AES standard for details.

27.1. AES SFR Registers

There are three SFR registers associated with the AES accelerator.

To use the GF multiplier the user must first write the GFM_CONST register. The write is needed only if the user desires to change the previous value in that register. It holds its value until overwritten. To perform the multiply operation the data has to be written to GFM_DATA register. Writing data to GFM_DATA register invokes the actual multiply operation. It takes 2 system clock cycles to perform the multiplication and the calculated result appears in the GFM_DATA register, overwriting the user input data. Therefore, at least a single cycle dummy instruction must be added in between writing the data to be multiplied to the GFM_DATA register and reading the result from there:

```
mov GFM_DATA, #data    ; Invoke a GF multiply
nop                    ; At least single cycle wait instruction
mov A, GFM_DATA        ; Read the result
```

Usage of the SBox/Inverse SBox hardware is controlled by the AES_DECRYPT bit in the SYS_SET register (SYS_SET.3). For encryption, the SBox operation is selected, for decryption the Inverse SBox operation is selected.

To pass data through the SBox the user has to write the data to the SBOX_DATA register. Writing data there invokes the conversion operation. The result appears in the SBOX_DATA register, overwriting the original data. It takes 2 system clock cycles to perform the conversion. Therefore, at least a single cycle dummy instruction must be added in between writing the data to be converted to the SBOX_DATA register and reading the result from there:

```
mov SBOX_DATA, #data   ; Invoke a SBox conversion
nop                    ; At least single cycle wait instruction
mov A, SBOX_DATA       ; Read the result
```

If the Silicon Labs device API AES implementation is used by the user application, all the AES accelerator communication is handled by the API functions and is hidden from the user.

SFR Definition 27.1. GFM_DATA

Bit	7	6	5	4	3	2	1	0
Name	GFM_DATA[7:0]							
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0x84

Bit	Name	Function
7:0	GFM_DATA [7:0]	**GFM Multiplier Data Processing.** Writing of a value here registers the data for processing. Processed data is registered into the same register with single CLK_SYS cycle delay. Read from this register reads the processed multiplied data. The register GFM_CONST must be written before GFM_DATA is written.

SFR Definition 27.2. GFM_CONST

Bit	7	6	5	4	3	2	1	0
Name	GFM_CONST[7:0]							
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0x85

Bit	Name	Function
7:0	GFM_CONST [7:0]	**GFM Multiplier Constant Register.** This is the constant by which the GFM_DATA is multiplied by. It has to be written prior to GFM_DATA.

SFR Definition 27.3. SBOX_DATA

Bit	7	6	5	4	3	2	1	0
Name	SBOX_DATA[7:0]							
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0x86

Bit	Name	Function
7:0	SBOX_DATA [7:0]	**AES SBox Processing.** Writing of a value here registers the data for processing. Processed data is registered into the same register with single CLK_SYS cycle delay. Read from this register reads the processed data. The type of SBox processing is controlled by AES_DECRYPT bit

28. Reset Sources

Reset circuitry allows the controller to be easily placed in a predefined default condition. There is only one external reset source for the device, which is power on reset. It gets invoked at two occasions:

1. Power is supplied to the device. This means connecting the power supply to disconnected device or cycling the external power to the device.
2. The device is waking up from a shutdown/standby mode. The power supply was connected before, but the device was put into the shutdown/standby mode. When it is awaken the power is supplied internally to all the device systems.

On entry to the reset state, the following events occur:

- CIP-51 halts program execution
- Special Function Registers (**SFR**) are initialized to their defined reset values
- XDATA registers (**XREG**) are initialized to their defined reset values
- External Port pins are forced to a known state
- Interrupts and timers are disabled

All SFRs are reset to the predefined values noted in the SFR detailed descriptions. The contents of internal data memory is lost since the power got cycled.

The Port I/O latches are reset to 0xFF (all logic ones) in open-drain mode.

On exit from the reset state, the program counter (PC) is reset, and the system clock defaults to the internal oscillator frequency of 24 MHz. Device starts its startup boot procedure. See other sections for description of the boot procedure. The user code starts being executed only after the boot procedure finishes. See section 23. System Boot and NVM Programming for details.

28.1. Device Boot Outline

Since the device does not have flash memory to permanently hold user code, the device has to go through a boot sequence in which the user code is copied from the one time programmable NVM memory to the CODE/XDATA RAM. After that is done the user program execution starts at address 0x0000.

It takes about fixed 2 ms plus about 3.6 ms per 1 kB of user data to be copied from NVM to RAM. When the user puts the device into shutdown mode this will be the estimated time for waking up the chip from shutdown mode by applying any GPIO to ground and the execution of the first instruction of the user code in CODE/XDATA RAM.

For debugging purposes the user will not program the NVM, but will use the RAM for code development. In that case the device will go through much shorter startup routine, which would take less than 2 ms to conclude.

See “23. System Boot and NVM Programming” on page 75 for details.

28.2. External Reset

There is no external reset. There is no pin dedicated to the device reset. The Silicon Labs debug chain using USB debug adapter or ToolStick has access to the proprietary reset control on chip to facilitate user code debug and development. During the debugging sessions on unprogrammed part the content of the CODE/XDATA RAM is preserved in between IDE environment invoked resets (**Reset** button inside IDE).

28.3. Software Reset

There is no software reset other than what a running program can invoke by software means. The only thing the running program can do is to put the device into the shutdown mode, effectively disconnecting power to internal systems of the device. User can wake the device up by connecting any of the GPIO ports to ground.

29. Port Input/Output

Digital resources are available through up to 10 I/O pins. The number of I/O depends on the package:

- 10 pin package .. 6 port pins organized as 6 bottom bits of Port 0.
- 14 pin package .. 10 port pins organized as a full 8-bit Port 0 and 2 bottom bits of Port 1.

The package pin assignment is in Figure 29.1.

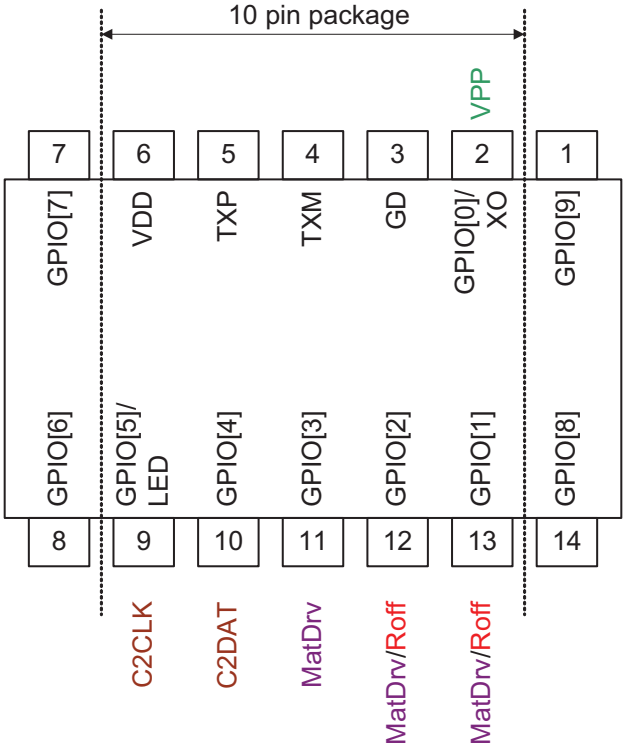


Figure 29.1. Device Package and Port Assignments

Pin assignments for 10– and 14–pin packages are shown in Table 29.1 and Table 29.2.

Table 29.1. 10–Pin Mode

Package Pin Number	Package Pin Name
1	GPIO0/XO
10	GPIO1
9	GPIO2
8	GPIO3
7	GPIO4
6	GPIO5/LED

Table 29.2. 14–Pin Mode

Package Pin Number	Package Pin Name
2	GPIO0/XO
13	GPIO1
12	GPIO2
11	GPIO3
10	GPIO4
9	GPIO5/LED
8	GPIO6
7	GPIO7
14	GPIO8
1	GPIO9

The GPIO Port I/O can be configured as either open-drain or push-pull in SFR registers P0CON and P1CON.

The GPIO functional diagrams and related digital control are in Figure 29.2 and Figure 29.3.

The option for Matrix and Roff modes is available only on GPIO[3:1].

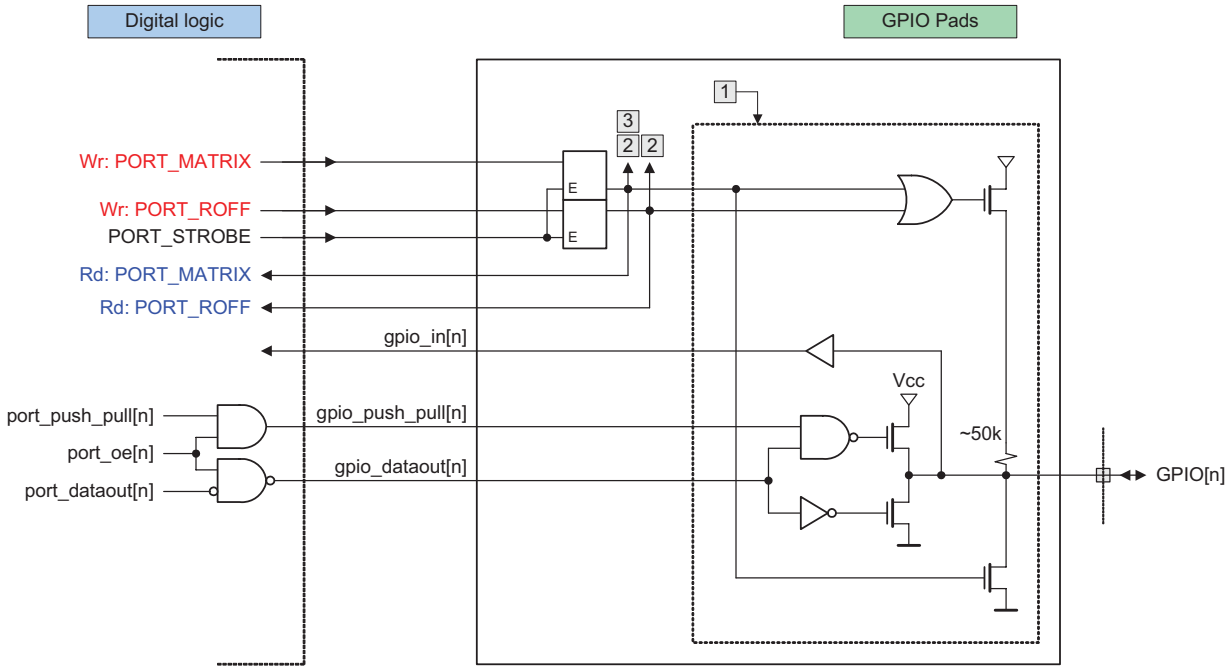


Figure 29.2. GPIO[3:1] Functional Diagram

Functional diagram of the other GPIO ports is in Figure 29.3. It is the general GPIO circuit that can be forced by digital control to have limited functionality (e.g., as input only, etc.).

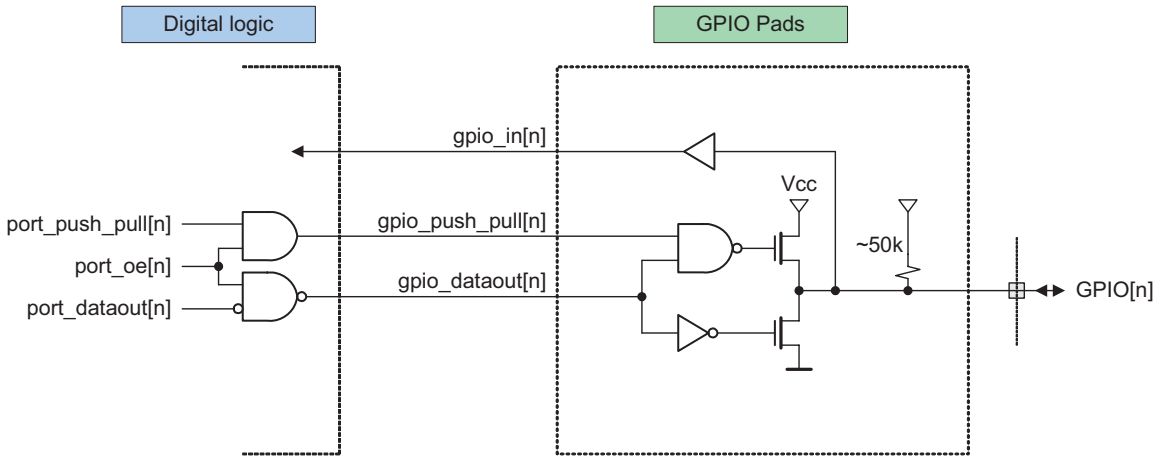


Figure 29.3. Other GPIO Functional Diagram

29.1. GPIO Pin Special Roles

Not all GPIO ports can be configured as both input and outputs. Given the limited number of GPIO each pin can assume different functionality based on the software configuration of the ports. The functionality of each GPIO is described in Table 29.3.

Table 29.3. GPIO Special Roles

GPIO Number	Other Special Roles	C2	FOB	Can Drive Low During Sleep	Pullup Roff Option
0	XO/VPP1[1]		button		
1			button	Y	Y
2			button	Y	Y
3	clk_ref[2]		button	Y	
4	clk_out out	C2DAT	button		
5		C2CLK	LED[3]		
6 (14 pin only)	clk_out out[4]		button		
7 (14 pin only)			button		
8 (14 pin only)			button		
9 (14 pin only)			button		

Notes:
1. Can be set as GPIO input only. Special roles are crystal oscillator (XO) and VPP=6.5V NVM programming voltage supply during NVM programming.
2. Reference clock source for frequency counter.
3. Current mode driver for LED connected directly to VCC supply. GPIO[5] cannot be used for any other purpose in user application.
4. Optional customer clock **clk_out** output can be set independently on GPIO[4] and GPIO[6], or on both at the same time.

It is important to emphasize the following:

- GPIO[0] can be used only as input for user application. It can also serve as a crystal oscillator input. During device NVM programming the programming VPP=6.5V voltage is applied to this pin.
- GPIO[5] can be used only as a up to 1mA LED current driver. The LED should be connected directly in between the GPIO[5] and VCC. In a development system this pin is used as a C2 interface C2CLK. In the development system the LED has to be isolated from the pin as shown in Figure 34.1 and Figure 34.2. The LED is disabled during debugging.

29.2. Pullup Roff Option

There is an option to disable the weak pullup pad resistors. This feature is called **Roff** option. The **Roff** option is controlled directly by the GPIO pads and persist when the chip is in the shutdown mode. Control of the **Roff** control bit in the GPIO is described in section 29.4. Pullup Roff and Matrix Mode Option Control.

29.3. Matrix Mode Option

The target application of the device is the button intensive application, which samples button pushes at the device inputs and acts accordingly.

Given the pin limited package, the target user application could use at most 5 buttons on a 10-pin package and 9 buttons on 14-pin package. If the chip is in a shutdown mode, any button push (connection to any GPIO to ground) wakes the chip up.

For the applications requiring more push button inputs than the available GPIO inputs, **Matrix** button mode should be implemented on the device. This allows the buttons to be organized in 3x2 matrix for 10 pin package or 3x6 matrix for the 14-pin package, allowing for up to 6 push buttons for 10 pin package and up to 18 buttons on the 14-pin package. It is up to the firmware to scan the matrix sequentially to determine the status of the buttons.

When the buttons are organized in **Matrix** mode any button push must wake the chip up from a shutdown mode. Since the buttons are not connecting GPIO to ground, but connecting an input GPIO to some output GPIO, the output GPIO must be connected to ground during the chip shutdown. That is achieved by setting the Matrix option control bit in the GPIO latch. When that bit is set then the GPIO[3:1] are actively pulled to ground when the chip is in the shutdown mode and digital logic has no power internally.

Note that to use the **Matrix** mode the **Roff** option must not be used. In other words, all the pullup resistors must stay in place for all the GPIO. There should be values PORT_MATRIX=1 and PORT_ROFF=0 latched into GPIO options control latch.

When the **Matrix** mode is latched into the GPIO control latch the pullup resistors of the GPIO[3:1] are disconnected and the pull down transistor on those GPIOs is activated.

Important: Before invoking a **Matrix** mode the user is responsible for programming all GPIO[3:1] as inputs. This is achieved by writing 1 to P0[3:1] and writing 0 to P0CON[3:1]. Only after that the **Matrix** option can be invoked.

If the chip went to shutdown with **Matrix** option set, then it will be woken up by any button press of the button matrix. It is a responsibility of the user application which *must turn the **Matrix** mode off* before the software can scan the button matrix for current button status. The button scanning is usually done scanning the matrix driver pins GPIO[3:1] with one-cold pattern, applying sequential binary patterns GPIO[3:1]=110, 101, 011, and 111 using open drain configuration of the GPIO[3:1]. By collecting corresponding responses on the GPIO[4,0] or GPIO[4,0,9:6], input GPIOs to the driving one-cold patterns firmware can determine what buttons are currently pushed.

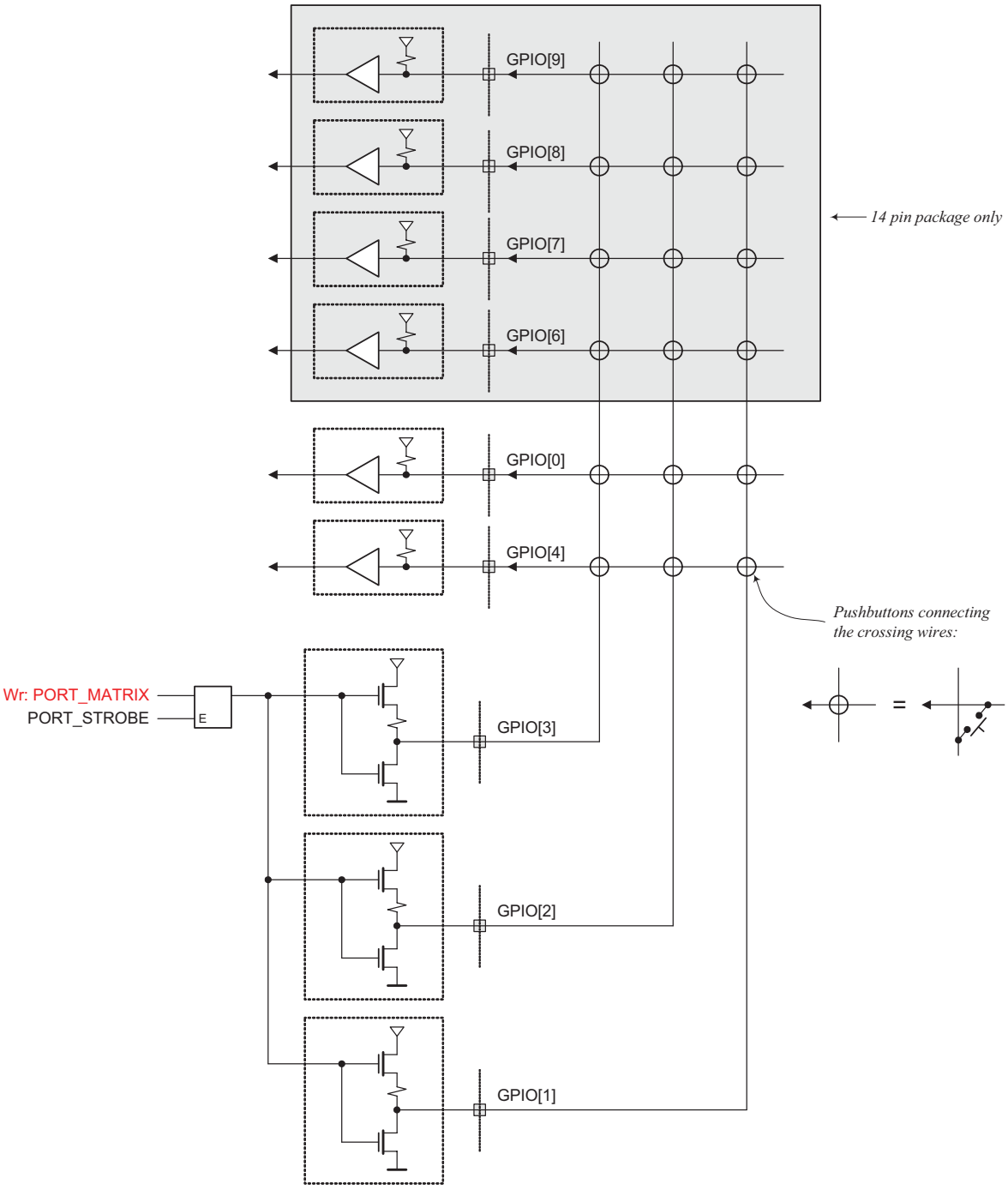


Figure 29.4. Push Button Organization in Matrix Mode

29.4. Pullup Roff and Matrix Mode Option Control

Both **Roff** and **Matrix** mode options are controlled by the GPIO pad itself. The control is implemented as 2 bit latch inside of the GPIO pads. Both options stay in their used defined states during chip shutdown. In other words, if the chip is in shutdown mode, the digital logic does not have power, but the two GPIO latches keep the user set values of those options.

The options are controlled by the PORT_CTRL SFR register. The user has to strobe the desired values to the GPIO latches by software sequence. The latch enable is a PORT_STROBE bit in the PORT_CTRL register.

For example, to disable the both **Matrix** and **Roff** options at the beginning of use application, the user code should look like this in assembly:

```
anl PORT_CTRL, #10011111B   ; Clear PORT_MATRIX and PORT_ROFF
orl PORT_CTRL, #10000000B   ; Set PORT_STROBE=1
anl PORT_CTRL, #01111111B   ; Clear PORT_STROBE=0
```

Using Silicon Labs provided masks in the header:

```
anl PORT_CTRL, #NOT(M_PORT_MATRIX OR M_PORT_ROFF)
orl PORT_CTRL, #M_PORT_STROBE
anl PORT_CTRL, #NOT(M_PORT_STROBE)
```

The toggle of the PORT_STROBE from 0 to 1 back to 0 latches the current register values of PORT_MATRIX and PORT_ROFF.

To summarize: To change the values of the **Matrix** an **Roff** options, the following software sequence is required:

1. Set the desired values of PORT_MATRIX and PORT_ROFF bits in the PORT_CTRL register.
2. Toggle the PORT_STROBE bit in the PORT_CTRL register from 0 to 1 back to 0 while not changing any other bit in the PORT_CTRL register. The new **Matrix** and **Roff** control values are latched into the GPIO.
3. Note that while reading the PORT_CTRL the current value of the **Matrix** and **Roff** options is read from the GPIO, **not** the value of the write register for the new **Matrix** and **Roff** setting.

Invoking a **Matrix** mode requires the following sequence:

1. Set the GPIO[3:1] as inputs, which means writing 1 to the port value and making the driver open drain.
2. Latch PORT_MATRIX=1 and PROT_ROFF=0 values to the GPIO option control latch.

In assembly:

```
orl P0,         #00001110B        ; Turn GPIO[3:1] as inputs
anl P0CON,      #NOT 00001110B
anl PORT_CTRL, #NOT(M_PORT_MATRIX OR M_PORT_ROFF)
orl PORT_CTRL, #M_PORT_MATRIX     ; Set Matrix mode and keep resistors
orl PORT_CTRL, #M_PORT_STROBE     ; Strobe new Matrix/Roff modes to GPIO
anl PORT_CTRL, #NOT(M_PORT_STROBE)
```

29.5. Special GPIO Modes Control

Some of the GPIO serves multiple purposes. Special configuration registers PORT_CTRL and PORT_SET are used to configure GPIO for other purpose then regular GPIO. Some GPIO can server multiple special purposes.

Table 29.4 shows all the functionality the GPIO can assume along with control signals and priority of the functionality. The lower the priority number, the higher the functional priority. For example, if the functionality with priority 1 is programmed, then controls selecting functionality of priority 2 and above will be ignored no matter what the control settings are.

Table 29.4. GPIO Special Roles Control and Order

GPIO	Roles	Order	Control	Comment
0	VPP	1		NVM programming voltage VPP = 6.5 V
	XO	2	**XO_CTRL**.XP_ENA	
	GPIO	3	**P0**.0 fixed as input only	
1	GPIO	1	**P0**.1 **P0CON**.1	
	Matrix	Ind*	PORT_CTRL	
2	GPIO	1	**P0**.2 **P0CON**.2	
	Matrix, Roff	Ind*	PORT_CTRL	
3	Reference **clk_ref**	1	**PORT_SET**.PORT_REFEN	Reference interval clock for frequency counter
	GPIO	2	**P0**.3 **P0CON**.3	
	Matrix, Roff	Ind*	PORT_CTRL	
4	C2DAT	1	Automatically “stolen” from application during C2 transaction.	
	Output clk_out	2	**PORT_SET**.PORT_CLKEN **PORT_SET**.PORT_CLKOUT[0]	Cannot be used in the development system, since C2 transaction disrupts the output.
	GPIO	3	**P0**.4 **P0CON**.4	
5	C2CLK	1	Acts as if a C2 debug clock input of the LED driver is not turned on.	
	LED driver	2	**P0**.5 **PORT_CTRL**.PORT_LED[1:0]	Port forced as output. To read the actual LED driver status (on/off) the user should read **RBIT_DATA**.GPIO_LED_DRIVE
6	Output clk_out	1	**PORT_SET**.PORT_CLKEN **PORT_SET**.PORT_CLKOUT[1]	14 pin only
	GPIO	2	**P0**.6 **P0CON**.6	
7	GPIO	1	**P0**.7 **P0CON**.7	14 pin only
8	GPIO	1	**P1**.0 **P1CON**.0	14 pin only
9	GPIO	1	**P1**.1 **P1CON**.1	14 pin only

***Note: Ind** stands for “Independent” setting. The **Matrix** and **Roff** modes are controlled in analog pad circuitry.

29.6. LED Driver on GPIO[5]

For application mode the GPIO[5] is shared with LED current driver. The LED current driver provides three levels of LED current, 1mA maximum. The current levels are described in SFR Definition 29.6. User can set the current intensity and then control the LED on and off by P0.5, port P0 bit 5, as a regular output. There is no need to modify the P0CON.5 bit, since the GPIO[5] output driver is set to be open drain. When the LED driver is on by setting the P0.5=1 then the pulldown output transistor is disabled. The GPIO[5] is used as a regular open drain output during the C2 debugging sessions only.

During the C2 debug sessions the IDE will forcibly disable the LED driver so the LED drive will not interfere with the debugging session. There will be an option on IDE to disable the “LED disable”, but it will have to be used with caution.

When the user hits **Disconnect** button on the IDE then the IDE clears all breakpoint, removes the LED disable, and runs the application from the point where it was halted. Then the application will control the LED. The user then can hit the **`Connect`** button on the IDE to connect to the chip again. For the IDE to be able to connect to the chip the LED must not be driven (not lit).

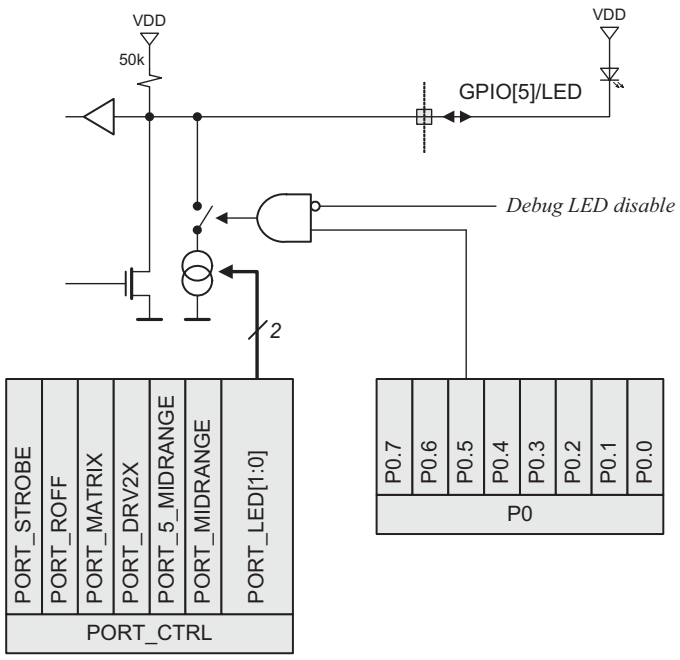


Figure 29.5. GPIO[5] LED Driver Block Diagram

SFR Definition 29.1. P0

Bit	7	6	5	4	3	2	1	0
Name	P0[7:0]							
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	0	1	1	1	1	1

SFR Address = 0x80

Bit	Name	Function
7:0	P0[7:0]	**Port 0 Register, GPIO[7:0], Bit Addressable.** Write appears at the GPIO[7:0] outputs, read reads directly the GPIO input values. Write: 0 .. output low value 1 .. output open-drain or high drive value in push-pull mode Read: 0 .. GPIO pin is at logic low value 1 .. GPIO pin is at logic high value Special pins: The GPIO[0] is input only. Write to GPIO[0] has no effect. The GPIO[5] is output LED driver only and requires setting of the proper LED drive current. Then GPIO[5] just turns the LED current on (1) or off (0). Reading from GPIO[5] returns the user intended driver of LED (1 .. driving, 0 .. off). The read value will be read as 0 if, for example, the user writes GPIO[5] as 1, but the LED current value PORT_CTRL.PORT_LED will be 0. The read GPIO[5] value does not represent the actual driving status of the LED drive, since the debug logic and C2 can disable the LED. The actual LED driving status can be read as RBIT_DATA.GPIO_LED_DRIVE bit.

SFR Definition 29.2. P0CON

Bit	7	6	5	4	3	2	1	0
Name	P0CON[7:0]							
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xA4

Bit	Name	Function
7:0	P0CON[7:0]	**Port 0 Configuration Register, for GPIO[7:0].** This bit controls configuration of each corresponding output bit in P0. 0 .. open-drain 1 .. push-pull If the pin to be input, it must be configured as open-drain and 1 has to be written as output value to it.

SFR Definition 29.3. P1

Bit	7	6	5	4	3	2	1	0
Name	P1[7:0]							
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	1	1

SFR Address = 0x90

Bit	Name	Function
7:0	P1[7:0]	**Port 1 Register GPIO[15:8], Bit Addressable.** Write appears at the GPIO[15:8] outputs, read reads directly the GPIO input values. Same as for P0. Only GPIO[9:8] are used, write to the rest of the register has no effect, read returns 0 at those bits.

SFR Definition 29.4. P1CON

Bit	7	6	5	4	3	2	1	0
Name	P1CON[7:0]							
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xA5

Bit	Name	Function
7:0	P1CON[7:0]	**POrt 1 Register GPIO[15:8], Bit Addressable.** This bit controls configuration of each corresponding output bit in P1. 0 .. open-drain, pull up resistor connected (see PORT_ROFF) 1 .. push-pull, pull up resistor disabled If the pin to be input, it must be configured as open-drain and 1 has to be written as output value to it. Only bits [1:0] corresponding to GPIO[9:8] are used, write to the rest of the register has no effect, read returns 0 for those bits.

SFR Definition 29.5. P2

Bit	7	6	5	4	3	2	1	0
Name	P2[7:0]							
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xA0

Bit	Name	Function
7:0	P2[7:0]	**Port 2 Register, Bit Addressable.** It is not a port, but a regular register. This register is used as a page MSB address byte for XDATA addressing in mode, using the PDATA memory accesses. The sole purpose for it is to support the PDATA model.

SFR Definition 29.6. PORT_CTRL

Bit	7	6	5	4	3	2	1	0
Name	PORT_ STROBE	PORT_ ROFF	PORT_ MATRIX	PORT_ DRV2X	PORT_5_ MID-RANGE	PORT_ MID-RANGE	PORT_LED[1:0]	
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	—	—	0	1	1	0	0

SFR Address = 0xB5

Bit	Name	Function
7	PORT_ STROBE	**Port Strobe.** Strobe the port_matrix and port_roff bits values from this register to the GPIO pads. The operation requires additional 2 CPU clock to finish after writing 0->1->0 to this bit. When 1 is written to this bit the GPIO latches open and the values of port_matrix and port_off are propagated to GPIO pads. Software must clear this bit to capture those two bits in the GPIO pads internal HV permanent latches.
6	PORT_ ROFF	**Port Roff Mode.** Roff mode, read from this bit returns the actual Roff mode value as reported from GPIO pad. When a 1 is latched into the GPIO pad internal Roff mode HV latch then the GPIO Roff mode gets invoked. The GPIO[1:2] will have their pull-up resistors turned off.
5	PORT_ MATRIX	**Port Matrix Mode.** Matrix mode, read from this bit returns the actual value matrix mode value as reported from GPIO pad. When a 1 is latched into the GPIO pad internal matrix mode HV latch then the GPIO matrix mode gets invoked. The GPIO[1:3] are driven low with resistor pull-ups disabled. This is intended for matrix button mode to wake up from sleep mode.
4	PORT_ DRV2X	**Increase Drive Strength by 2x on All Outputs.**
3	PORT_5_ MIDRANGE	**Input GPIO[5] pin trip point set to 45% VDD.**
2	PORT_ MIDRANGE	**Input GPIO Pin Trip Point Set to 45% VDD (except GPIO[5])**
1:0	PORT_LED [1:0]	**LED Current Drive Strength.** It must be set to non-zero value for LED to have any current. This is just a current source setting. The actual turning of the LED on and off is controlled by the GPIO[5] output bit in P0. 00: LED off 01: LED current = 0.62*600uA 10: LED current = 1.00*600uA 11: LED current = 1.62*600uA

SFR Definition 29.7. PORT_SET

Bit	7	6	5	4	3	2	1	0
Name	EDGE_ INT1	EDGE_ INT0	PORT_CLKOUT[1:0]		PORT_ CLKEN	PORT_ REFEN	Reserved	Reserved
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xB6

Bit	Name	Function
7	EDGE_ INT1	**Edge Control for INT1.** This bit controls whether single edge or both edges invoke the interrupt. 0 .. single edge, polarity specified by NEG_INT1 in PORT_INTCFG 1 .. both edges, which means any edge, invoke INT1 interrupt
6	EDGE_ INT0	**Edge Control for INT0.** This bit controls whether single edge or both edges invoke the interrupt. 0 .. single edge, polarity specified by NEG_INT0 in PORT_INTCFG 1 .. both edges, which means any edge, invoke INT0 interrupt
5:4	PORT_ CLKOUT [1:0]	**Select which GPIO Pin is used as Clock Output Pin:** port_clkout[0] .. GPIO[4]: 1 .. clk output, 0 .. normal operation port_clkout[1] .. GPIO[6]: 1 .. clk output, 0 .. normal operation Both outputs can be used simultaneously. The actual clock waveform can be enabled/disabled by port_clken bit, but the GPIO configuration is purely controlled by port_clkout.
3	PORT_ CLKEN	**Enable Output Clock, Which is Possibly Coming out on GPIO[4] and/or GPIO[6].** This bit is just a clock enable/disable, it does not configure the GPIO for clock outputs. The port configuration must be done by port_clkout below. The generated clock division is controlled by CLKOUT_SET register. If the clock is disabled by PORT_CLKEN=0 the current period in progress will be finished and the output clock will stop as logic 0.
2	PORT_ REFEN	**Enable CLK_REF Reference Clock to come from GPIO[3].** The GPIO[3] pad is forced to be an input. There is not need to change p0 or p0con register values, since port_refen has higher priority.
1:0	Reserved	These bits must be left at 0.

30. Clock Output Generation

The device includes an option to be used as a clock generator for other chips connected to the device. The generated clock frequency, **clk_out**, is derived from the internal 24MHz oscillator. System clock division set in SYSGEN register has no effect on the **clk_out** frequency.

The **clk_out** is an output of a divider with programmable division from 1 to 31 in an increment of 1. Therefore, the output frequency of the output clock can range from 24MHz to 24MHz/31 = 774kHz.

The divider has an option to keep the **clk_out** duty cycle to 1:1 even for odd division ratios. There is an option of at which logic level the **clk_out** stops when the clock generator is disabled.

The clock divider/generator always finishes the period it started before it accepts a new division factor CLKOUT_DIV. It is recommended to fix all the settings before enabling the output clock generator. The master enable is PORT_CLKEN bit in the PORT_SET register.

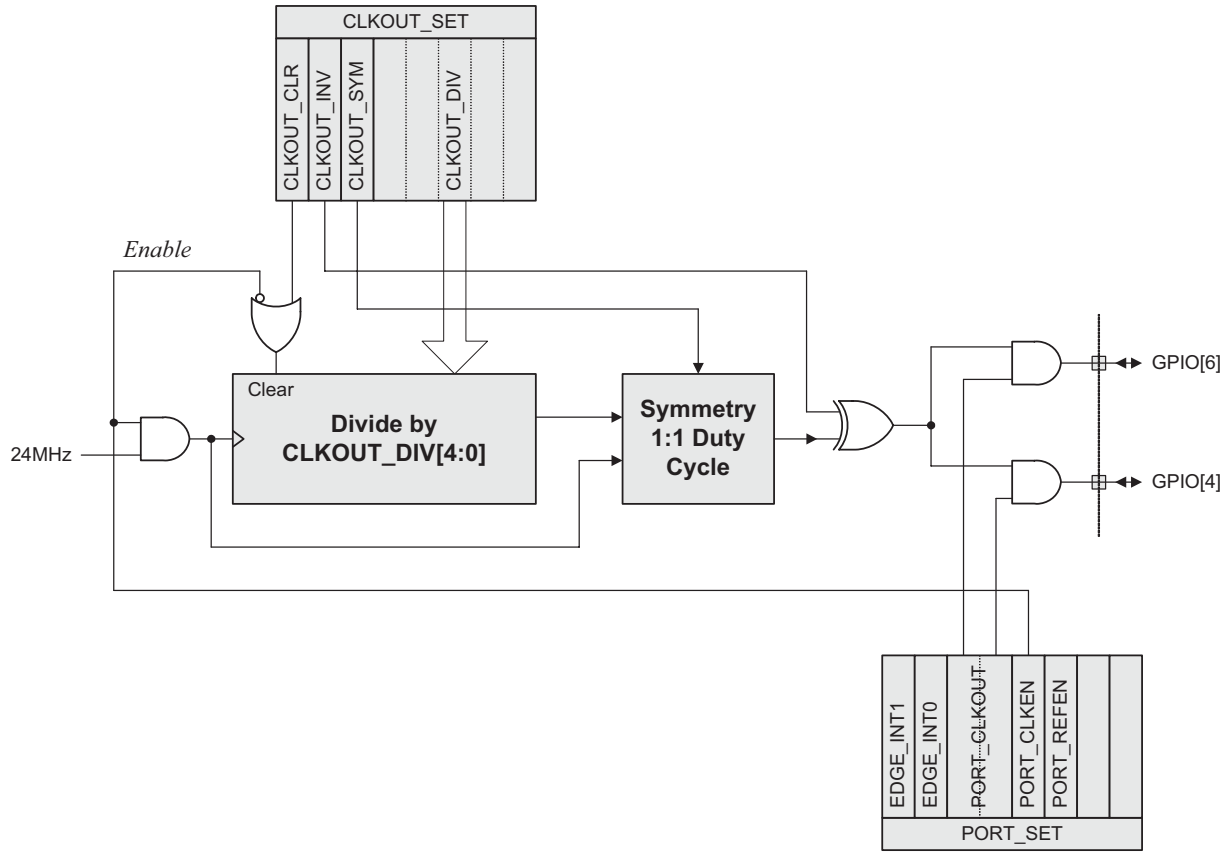


Figure 30.1. Output Clock Generator Block Diagram

30.1. Register Description

SFR Definition 30.1. CLKOUT_SET

Bit	7	6	5	4	3	2	1	0
Name	CLKOUT_CLR	CLKOUT_INV	CLKOUT_SYM	CLKOUT_DIV[4:0]				
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0x8F

Bit	Name	Function
7	CLKOUT_CLR	**CLKOUT Clear.** Write 1 to this bit clears the generated clock divider. The generated clock output is forced to 0. The pulse must be aligned with the registered write enable for this register, therefore the generated clear pulse must be registered. Reading this bit has CLKOUT_IDLE meaning. If read as 1 then it indicates that the clock divider generator is idle. It can be used to wait for the clock to get idle after the user clock output was disabled by PORT_SET.PORT_CLKEN=0. If this bit is read as 1 the clock division generator by factor 2 and above is running and the current user clock period is still in progress. The user could use this bit to synchronously switch the CLKOUT_DIV division factor, but it is not necessary. The synchronous clock period switching is built in the hardware. See the CLKOUT_DIV section above. To switch the clocks immediately without waiting for the current period to end, write 1 to this bit. The write 1 to this bit can be combined with setting the new CLKOUT_DIV value in this register at the same time.
6	CLKOUT_INV	**CLKOUT Inversion.** Invert the generated clock. The inverter is at the very end of the clock generation chain. Normally, if this bit is 0, if the generated clock is disabled the output is at 0. With this bit set to 1 the output is inverted, therefore the generated clock stops at 1. This bit must be set before customer clock is enabled to the port output by setting PORT_SET.PORT_CLKEN=1. If changed later the clock inversion takes effect immediately with possibility of short clock pulse being generated at the clock output.

Bit	Name	Function
5	CLKOUT_ SYM	**CLKOUT Symmetry.** If this bit set to 1 then the output clock duty cycle is very close to 1:1 irrespective of the division factor. However, the generated clock waveform is a combination of outputs of two flops and therefore might jitter more. If this bit is 0 then for odd division factor there is a single 24 MHz period difference in between halves of the generation output clock. This bit must be set before customer clock is enabled to the port output by setting PORT_SET.PORT_CLKEN=1.
4:0	CLKOUT_ DIV[4:0]	**CLKOUT Division Factor.** Division factor of the 24 MHz oscillator clock for generation of the output customer clock. The enable of the clock is controlled by the PORT_CLKEN and PORT_CLKOUT bits in PORT_SET register. The division factors 0 and 1 pass the 24 MHz internal cheap oscillator output as output clocks. Value bigger than 1 is the actual division factor of the 24 MHz. If CLKOUT_SYM=0 (recommended), the generated clock is an output of a flop. For odd division ratios the first part of the period in logic 0 is one 24 MHz clock cycle shorter than the second high half part of the period of generated clock, assuming CLKOUT_INV=0. If the clock is disabled by PORT_CLKEN=0 the current period in progress will be finished. To monitor when the output gets idle monitor the CLKOUT_CLR bit below. The CLKOUT_DIV bit can be changed any time. The new setting will take effect only after the current period finishes. For the new setting to take effect immediately see CLKOUT_CLR.

31. Control and System Setting Registers

The following are general system setting control registers as well as general purpose scratch pad registers. GPR_CTRL and GPR_DATA can be used as a general purpose 2 byte SFR register. They do not control any hardware on the device.

SFR Definition 31.1. GPR_CTRL

Bit	7	6	5	4	3	2	1	0
Name	GPR_CTRL[7:0]							
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xB1

Bit	Name	Function
7:0	GPR_CTRL[7:0]	**General Purpose Register.**

SFR Definition 31.2. GPR_DATA

Bit	7	6	5	4	3	2	1	0
Name	GPR_DATA[7:0]							
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xB2

Bit	Name	Function
7:0	GPR_DATA[7:0]	**General Purpose Register.**

SFR Definition 31.3. RBIT_DATA

Bit	7	6	5	4	3	2	1	0
Name	Reserved	Reserved	GPIO_ LED_ DRIVE	XO_ CKGOOD	ODS_ EMPTY	ODS_NOD ATA	TRNG_ OUT	PA_ COMPOUT
Type	R	R	R	R	R	R	R	R
Reset	0	0	0	0	1	1	0	0

SFR Address = 0xEE

Bit	Name	Function
7:6	Reserved	Read as 0x0. Write has no effect.
5	GPIO_LED_DRIVE	**GPIO LED Drive.** Actual status of the LED drive. If this bit is at 1, then the LED driver is actually on. The LED driver is controlled by P0.5 bit and the intensity value in PORT_CTRL register. If the P0.5 bit is read, then it returns user LED drive request, which does not reflect the actual LED driver status.
4	XO_CKGOOD	**Crystal Oscillator Clock Good.** Crystal oscillator XO output is stable. It takes about 5 ms before the XO_ENA is set to 1 and the XO output becomes stable. When this signal becomes 1, software must wait additional 3 ms before it can use the XO output for frequency counting. See the XO_CTRL test register.
3	ODS_EMPTY	**ODS Empty.** Supplementary flag indicating that the ODS Tx holding register is empty. It can be used as an indication for software to write a new data byte to ODS_DATA register to transmit. This applies to the Tx holding register only. See ODS_NODATA for the flag related to the actual Tx shift register.
2	ODS_NODATA	**ODS No Data.** Supplementary flag that the output digital serializer (ODS) Tx shift register ran out of data and there is nothing else to transmit.
1	TMG_OUT	**Thermal Random Number Generator (Rng) Output Bit.** Note that the temperature sensor must be enabled in order to get an interesting result here.
0	PA_COMPOUT	**Output of the Phase Detection Comparator at the PA Output.** Output of phase detection PA comparator. When PA is enabled, runs at full clock rate, and is asynchronous to clk_sys. Should be demetastabilized sufficiently by transferring to another register for purposes of getting to CPU.

32. Real Time Clock Timer

The Si4010 device contains a real time clock (RTC) timer. This dedicated timer provides accurate interrupt request pulses in precise time intervals. The device does not contain any hardware nor any battery backed up real time clock. The purpose of RTC timer is to provide accurate time intervals for user application at run time, not an absolute real calendar time.

The RTC timer clock source is the internal calibrated system clock generator. The RTC constant tick generator runs from the selected divided internal system clock, which is a power of two division of the 24 MHz internal oscillator. The frequency ranges from 24 MHz down to 24 MHz/128. The RTC tick generated is a constant frequency of 24 MHz/128 with tick period 5.33 µs and is independent of the system clock division setting SYSGEN_DIV in the SYSGEN SFR register.

The user can select what exact time intervals the RTC timer will set its interrupt flag. The time interval is programmable to be one of the following: 100 µs, 200 µs, 400 µs, 800 µs, 1 ms, 2 ms, and 5 ms. This time is independent of the selected system clock divider in the SYSGEN SFR register.

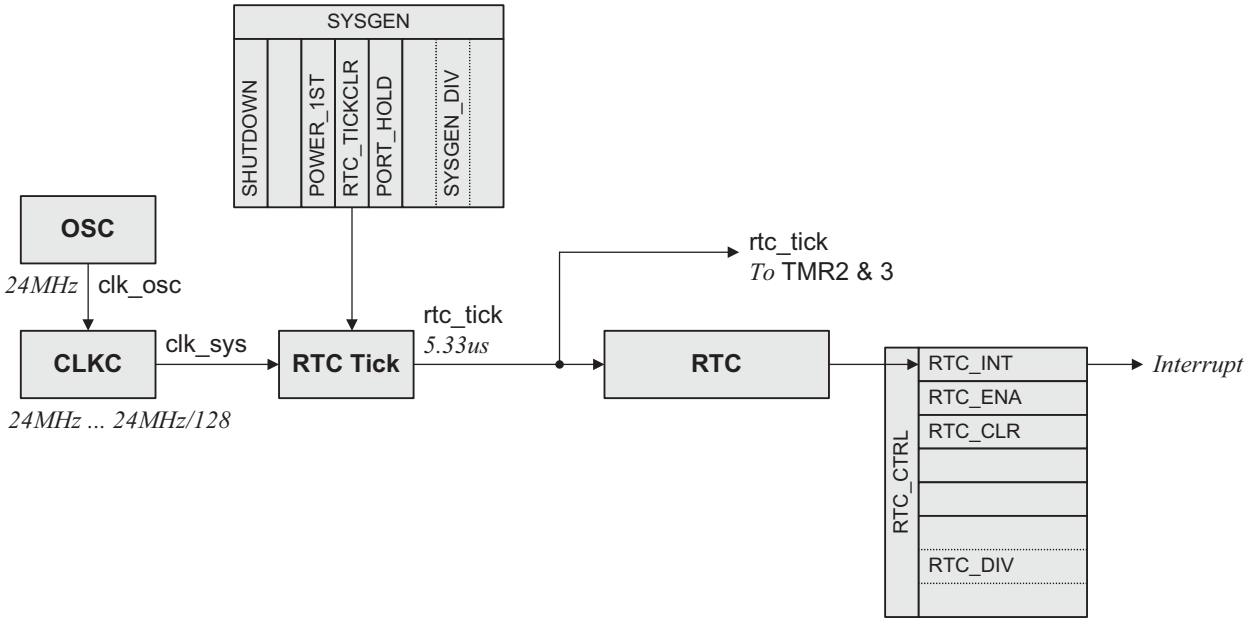


Figure 32.1. RTC Timer Block Diagram

32.1. RTC Interrupt Flag Time Uniformity

Since 100 µs and 200 µs pulse duration is not exactly an integer multiple of the 24 MHz/128 frequency, the fractional division was used. The 100 µs and 200 µs pulse durations are uniform *on average*, when observed over a sufficiently long timer period. Instantaneous time difference in between subsequent 100 µs and 200 µs pulses is not 100 µs or 200 µs, respectively, but fluctuates around those two values.

- 100 µs pulse train .. the 100 µs pulse train consists of **rtc_tick** time duration of 19, 19, 19, 18 ticks. That means that 3 subsequent 100 µs pulses has time difference of 19 x **rtc_tick** periods, which is 19 x 5.33 µs = 101.33 µs. That is followed by a singe duration or 18 x **rtc_tick** period duration, which is 18 x 5.33 µs = 96 µs. On average, the 100 µs pulse time period is (3 x 19 + 18)/4 x **rtc_tick** period, which is 18.75 x 5.33 µs = 100 µs exactly.
- 200 µs pulse train .. for 200 µs the pulse train consists of **rtc_tick** time duration of 38, 37 ticks. That means that the pulse train is an alternation train of 38 x 5.33 µs = 202.66 µs and 37 x 5.33 µs = 197.33 µs, when on average the duration is (38 + 37)/2 x 5.33 µs = 200 µs exactly.

The pulse trains for 400 µs pulses and longer have a uniform, exact, time periods.

32.2. Register Description

The RTC timer is controlled by the RTC_CTRL SFR register. If there is a need for precise beginning of the RTC timer period, the internal tick generator can be cleared by writing a bit RTC_TICKCLR in the SYSGEN register.

The **rtc_tick** generator runs freely whenever the RTC timer is enabled by RTC_ENA=1. If the user needs to clear the RTC timer to synchronize it with some event, writing 1 to RTC_CLR will clear the timer, which keeps running. The RTC **rtc_tick** generator is not cleared by that event. Therefore, there will be up to 5.33 µs time uncertainty in the calculated time period. Clearing of the RTC **rtc_tick** generator is achieved by writing 1 into the RTC_TICKCLR bit in SYSGEN register.

To achieve exact synchronization it is recommended to write 1 into the RTC_TICKCLR, then 1 to RTC_CLR, followed by another 1 into the RTC_TICKCLR. In assembly using the M_<field> masks 8-bit mask notation from the supplied assembly include file:

```
orl SYSGEN,   #M_RTC_TICKCLR
orl RTC_CTLR, #M_RTC_CLR
orl SYSGEN,   #M_RTC_TICKCLR
```

The reason for splitting the clear is that the RTC tick output, **rtc_tick** can also be selected as a time source for TMR2 and TMR3, so there is a need to have separate control over the **rtc_tick** generator clearing.

To get the RTC tick generator running the RTC_ENA=1 must be set. Therefore, even if the RTC interrupt is not used, the RTC timer must be enabled if the user wants to use the **rtc_tick** as a clock source for TMR2 or TMR3.

SFR Definition 32.1. RTC_CTRL

Bit	7	6	5	4	3	2	1	0
Name	RTC_INT	RTC_ENA	RTC_CLR	Reserved	Reserved	RTC_DIV[2:0]		
Type	R/W	R/W	W	R	R	R/W		
Reset	0	0	0	0	0	0	0	0

SFR Address = 0x9C

Bit	Name	Function
7	RTC_INT	**Real Time Clock Interrupt Flag.** Set after the time interval set by RTC_DIV field elapses. Software must clear the flag. Hardware will not clear the flag
6	RTC_ENA	**Real Time Clock Enable.** If set to 1 then the RTC_TICK and bottom part of the pulse generator starts running where it left off. If RTC_DIV >=3 then top half also starts. 0: RTC disabled 1: RTC enabled.
5	RTC_CLR	**Real Time Clock Clear.** Writing 1 will clear the pulse generator but will leave the RTC_TICK generator intact. See the RTC_TICKCLR in the SYSGEN register for clearing the RTC_TICK counter. 0: Normal operation 1: RTC cleared
4:3	Reserved	Read as 0x00. Write has no effect.
2:0	RTC_DIV [2:0]	**Real Time Clock Divider.** Select the divider of the RTC_TICK to determine the interval for the RTC interrupt generation. 000: No interrupt generation 001: 100 µs .. it is a 19/19/19/18 divider 010: 200 µs .. it is a 38/37 divider 011: 400 µs 100: 800 µs 101: 1 ms 110: 2 ms 111: 5 ms

33. Timers 2 and 3

The Si4010 device includes two identical timers, Timer 2 (TMR2) and Timer 3 (TMR3). Since the timers are identical, the description will refer to Timer 2 (TMR2). The reader can replace the TMR2 with TMR3 in the text to get the description of Timer 3 (TMR3). The description refers to a “Timer” as an alias for either TMR2 or TMR3.

Timer 2 is a 16-bit timer formed by two 8-bit SFRs: TMR2L (low byte) and TMR2H (high byte). Timer may operate in on of the two width modes:

- **Wide** mode .. timer operates as a single 16 bit wide timer controlled by the control bits related to the low half of the timer, like TMR2L_MODE, etc. The timer sets the TMR2INTH bit as an interrupt flag.
- **Split** mode .. timer operates as two independent 8 bit wide times, with related control bits related to high (H) and low (L) half of the overall 16 bit timer.

In each of the width modes each timer or each half of the timer can operate in two different functional modes:

- **Timer** mode .. the timer runs as a counter counting up, when it overflows it sets corresponding interrupt flag, reloads initial value, and keeps going, counting up.
- **Capture** mode .. the timer counter is free running counting up. When it overflows it keeps counting up from 0. When an external capture event happens then the current value of the timer is captured in the capture register, the counter keeps counting and will not stop. The interrupt flag is set by the capture event.

Each timer or timer half can be independently clocked from one of 4 clock sources. Clock source can be independently set for each half of the timer in **split** mode. The clock sources available for each timer half are:

1. Current system clock **clk_sys**. This is 24MHz, possibly divided by N-th power of 2 with N=0, ..., 7. See SYSGEN SFR register for system clock setting details.
2. Current system clock **clk_sys** divided by 12 .. **clk_sys**/12
3. RTC timer tick **rtc_tick** with 5.33us period (24MHz/128)
4. RTC timer 100us pulse. See the RTC section for an important note related to the uniformity of the 100us pulse train.

All clock sources are synchronous with the system clock.

The capture event is INT0 for TMR2 and INT1 for TMR3. They are edge events coming from external GPIO and are the same as for the external interrupt generation, INT0 and INT1. To use these events as capture events they have to be programmed exactly the same way as if they were intended to be used for interrupt generation. They could generate INT0 and INT1 interrupts at the same time when the are being used as capture events for TMR2 and TMR3, respectively.

If the timer operates in **split** mode both halves are completely independent. Therefore, all 4 combinations of functionality in **split** mode, timer/timer, timer/capture, capture/timer, and capture/capture are possible. Each half has separate clock selection. The only common thing is the capture signal, which is the same for both halves in **split** mode. The only difference in between of two halves in capture/capture mode can be the counter clock, set independently for each half.

33.1. Interrupt Flag Generation

Timer 2 has a single interrupt signal going to interrupt controller. Internally, there are 2 interrupt flags, TMR2INTH for high half of the timer and TMR2INTL for low half of the timer, which are combined to generate the final interrupt signal. The low half has a local interrupt flag enable TMR2INTL_EN control bit.

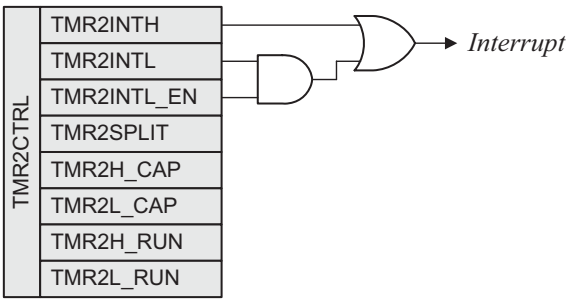


Figure 33.1. Timer Interrupt Generation

Setting of the interrupt flags depends on the width and functional modes of each timer or its half.

- **Wide** mode
  - **Timer mode**
    TMR2INTH set if TMR2H overflows
    TMR2INTL set if TMR2L overflows
  - **Capture** mode
    TMR2INTH set if capture event happens and TMR2H, TMR2L 16-bit value gets captured
    TMR2INTL set if TMR2H overflows.

    Note: This is an exception when low interrupt flag gets set based on the high half of the timer. This is a supplemental information for the interrupt handler about the capture, indicating that the 16-bit counter overflew in between captures.
- **Split** mode
  - **Timer** mode
    TMR2INTH set if TMR2H overflows
    TMR2INTL set if TMR2L overflows
  - **Capture** mode
    TMR2INTH set by capture event when TMR2H gets captured
    TMR2INTL set by capture event when TMR2L gets captured

Each of the modes is described in a separate section. There is a clock selection register TMR_CLKSEL common for both Timer 2 and Timer 3.

33.2. 16-bit Timer with Auto Reload (Wide Mode)

When TMR2SPLIT=0 and TMR2L_CAP=0, the timer operates as a 16-bit timer with auto reload.

As the 16-bit timer register increments and overflows from 0xFFFF to 0x0000, the 16-bit value in the time reload registers (TMR2RH and TMR2RL) is loaded into the timer register as shown in Figure 33.2, and the timer High Byte Overflow Flag TMR2INTH (TMR2CTRL.7) is set. If timer interrupts are enabled (see IE and EIE1 registers), an interrupt will be generated on each timer overflow. Additionally, if timer interrupts are enabled and the TMR2INTL_EN bit is set (TMR2CTRL.5), an interrupt will be generated each time the lower 8 bits (TMR2L) overflow from 0xFF to 0x00.

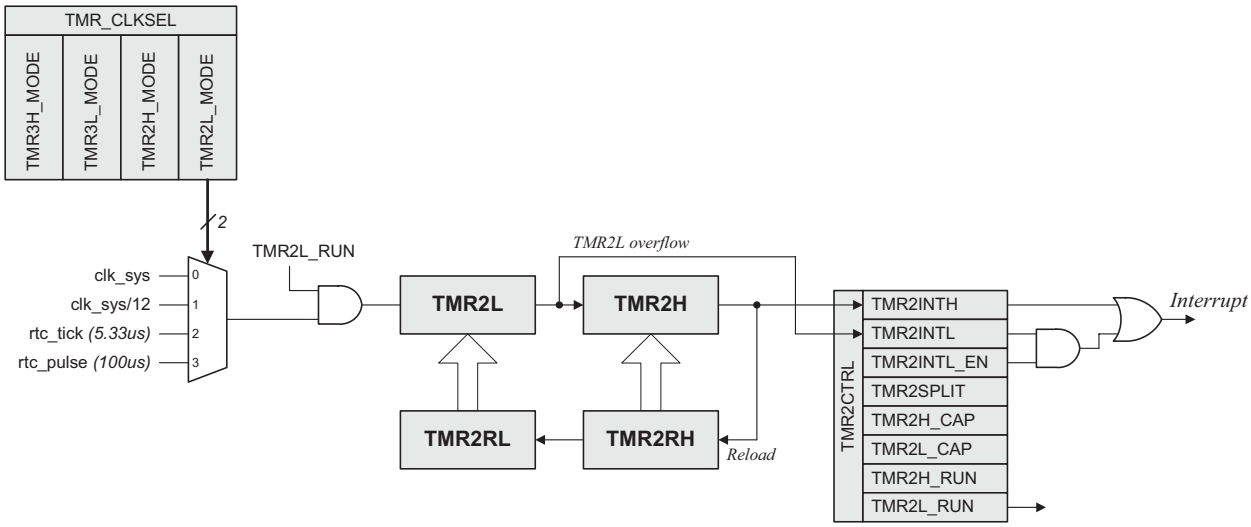


Figure 33.2. Timer 16-bit Mode Block Diagram (Wide Mode)

33.3. 16-bit Capture Mode (Wide Mode)

When TMR2SPLIT=0 and TMR2L_CAP=1, the timer operates in a 16-bit capture mode. The capture event is INT0 for Timer 2 and INT1 for Timer 3. It is the same edge event as programmed to generate external interrupt INT0 or INT1, respectively. The capture event can be positive edge, negative edge, or both edges of the GPIO associated with the INT0 and INT1. Capture mode can be used for measurement of time intervals on external signals.

Timer counts up and overflows from 0xFFFF to 0x0000. Each time a capture event is received, the contents of the timer registers (TMR2H:TMR2L) are latched into the timer reload registers (TMR2RH:TMR2RL). A timer high half interrupt TMR2INTH is generated by capture event. Additionally, the low byte interrupt flag TMR2INTL is set whenever the timer overflows from 0xFFFF to 0x0000. This additional information may be used by and application.

Note that the capture event can also generate its own external interrupt on top of the timer interrupt, if enabled by the application. Also note that if the capture timer is stopped (TMR2L_RUN=0) the capture event still captures the current counter registers (TMR2H:TMR2L) into the timer reload registers (TMR2H:TMR2RL) and sets the flag TMR2INTH.

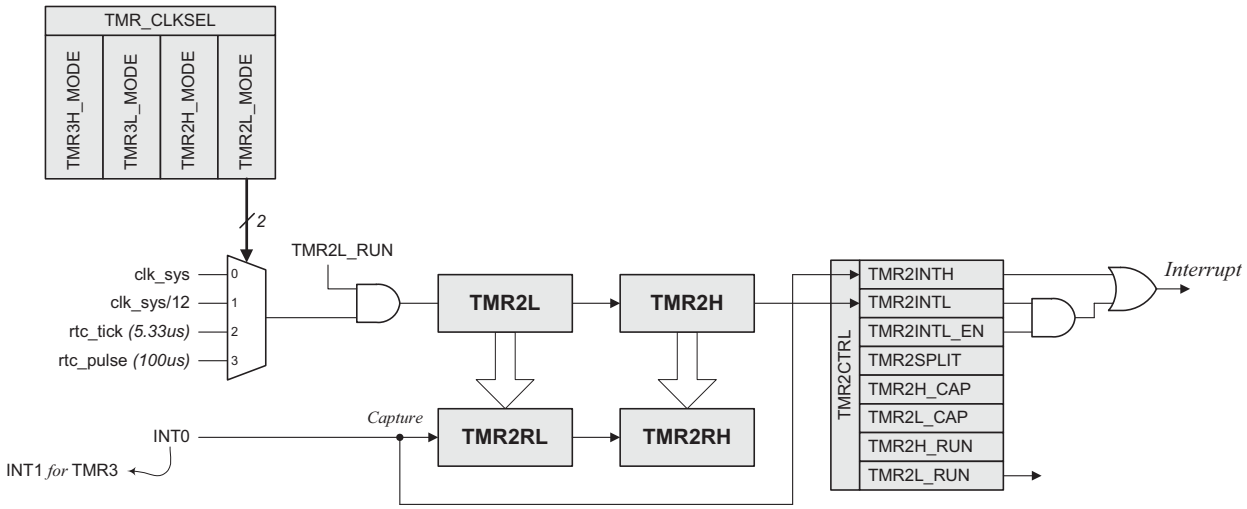


Figure 33.3. Capture 16-bit Mode Block Diagram (Wide Mode)

33.4. 8-Bit Timer/Timer Mode (Split Mode)

When TMR2SPLIT=1, the timer operates as two independent 8-bit timers. Each of the 8-bit timers can independently operate in either 8-bit timer or 8-bit capture modes. The only common signals for both 8-bit timers are capture event input signal and the interrupt output signal. Therefore, four possible configurations are possible in split mode. All of them are described in the subsequent sections.

If TMR2L_CAP=0 and TMR2H_CAP=0, both halves operate as two independent 8-bit timers with independently set clocks.

As the 8-bit timer register increments and overflows from 0xFF to 0x00, the 8-bit value in the time reload registers (TMR2RH or TMR2RL) is loaded into the corresponding timer register (TMR2H or TMR2L), and the corresponding byte overflow flag TMR2INTH or TMR2INTL are set, respectively. If timer interrupts are enabled (see IE and EIE1 registers), an interrupt will be generated on each timer overflow.

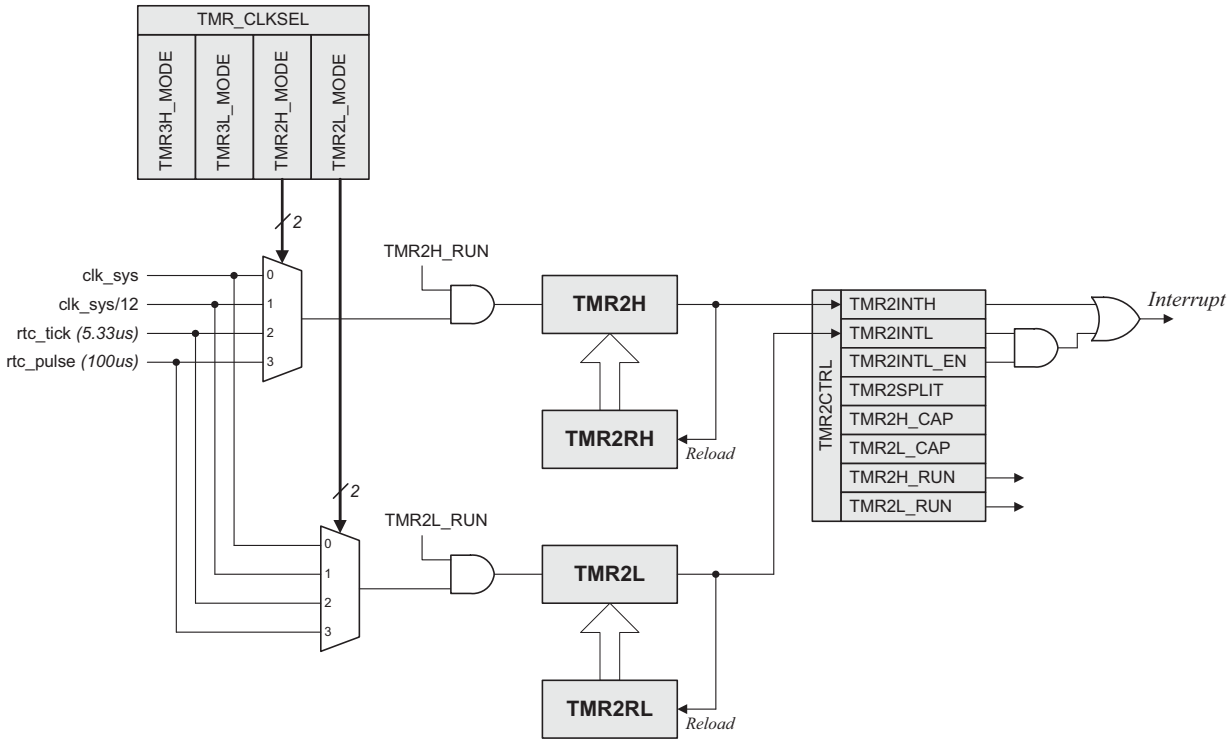


Figure 33.4. Two 8-bit Timers in Timer/Timer Configuration (Split Mode)

33.5. 8-Bit Capture/Capture Mode (Split Mode)

When TMR2SPLIT=1, TMR2L_CAP=1 and TMR2H_CAP=1, both halves operate independently in 8-bit capture modes. However, the capture event is the same for both timers. The clock sources for each timer are selected independently, so one timer can capture short pulses while the other one long pulses, for example.

Each 8-bit timer is free running, counts up and overflows from 0xFF to 0x00. Each time a capture event is received, the contents of the timer registers (TMR2H and TMR2L) are latched into the corresponding timer reload registers (TMR2RH and TMR2RL). Common capture event INT0 (INT1 for Timer 3) sets both high and low half interrupt flags TMR2INTH and TMR2INTL at the same time.

The capture event can also generate its own external interrupt on top of the timer interrupt, if enabled by the application. If the capture timer is stopped (TMR2L_RUN=0), the capture event still captures the current counter register TMR2L into the reload register TMR2RL and sets the flag TRM2INTL. Same independently applies to the upper half TMR2H with its respective registers and flags.

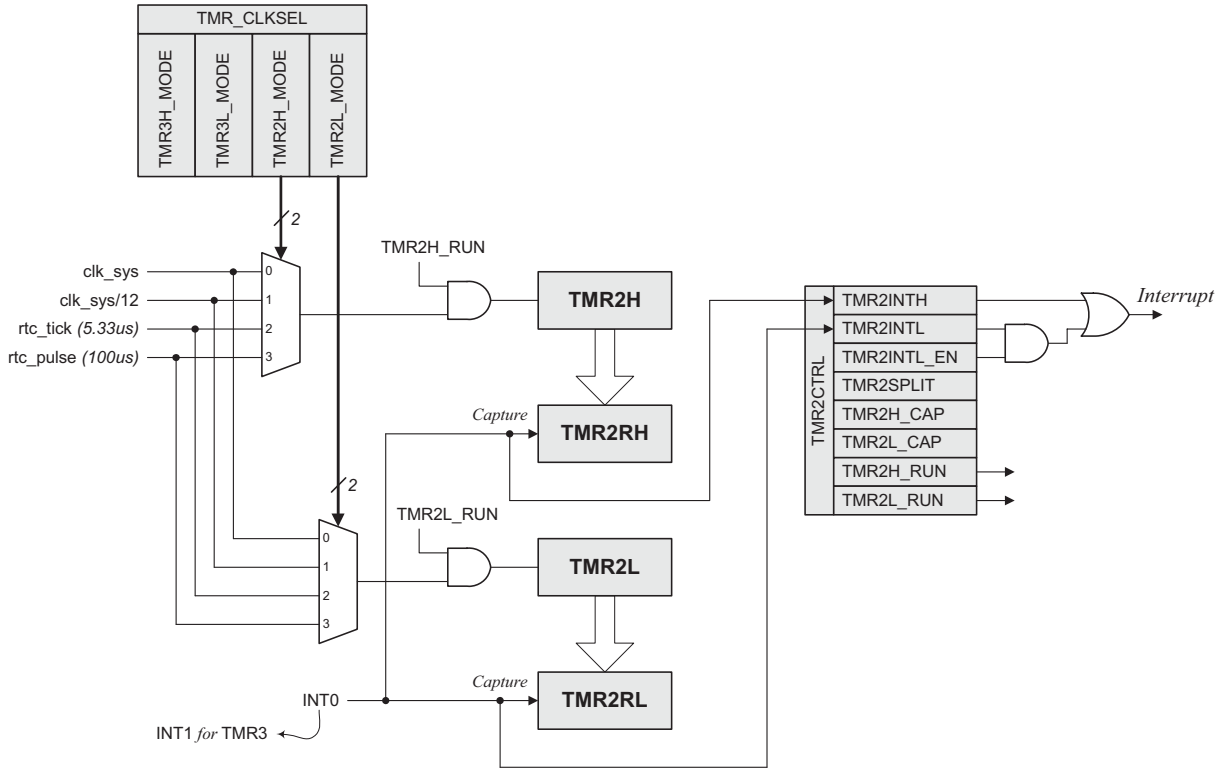


Figure 33.5. Two 8-bit Timers in Capture/Capture Configuration (Split Mode)

33.6. 8-Bit Timer/Capture Mode (Split Mode)

When TMR2SPLIT=1, TMR2L_CAP=1 and TMR2H_CAP=0, the split timers operate one in 8-bit timer mode and the other in 8-bit capture mode. Same situation happens when TMR2L_CAP=0 and TMR2H_CAP=1, only the roles of the timer 8-bit halves are reversed. The only difference in between these two scenarios are the interrupt flags settings, since TMR2INTH and TMR2INTL are not symmetrical. The TMR2INTL has a local enable TMR2INTL_EN. The functionality of the 8-bit timer and 8-bit capture modes for the respective halves is the same as described above when both halves operate in the same mode.

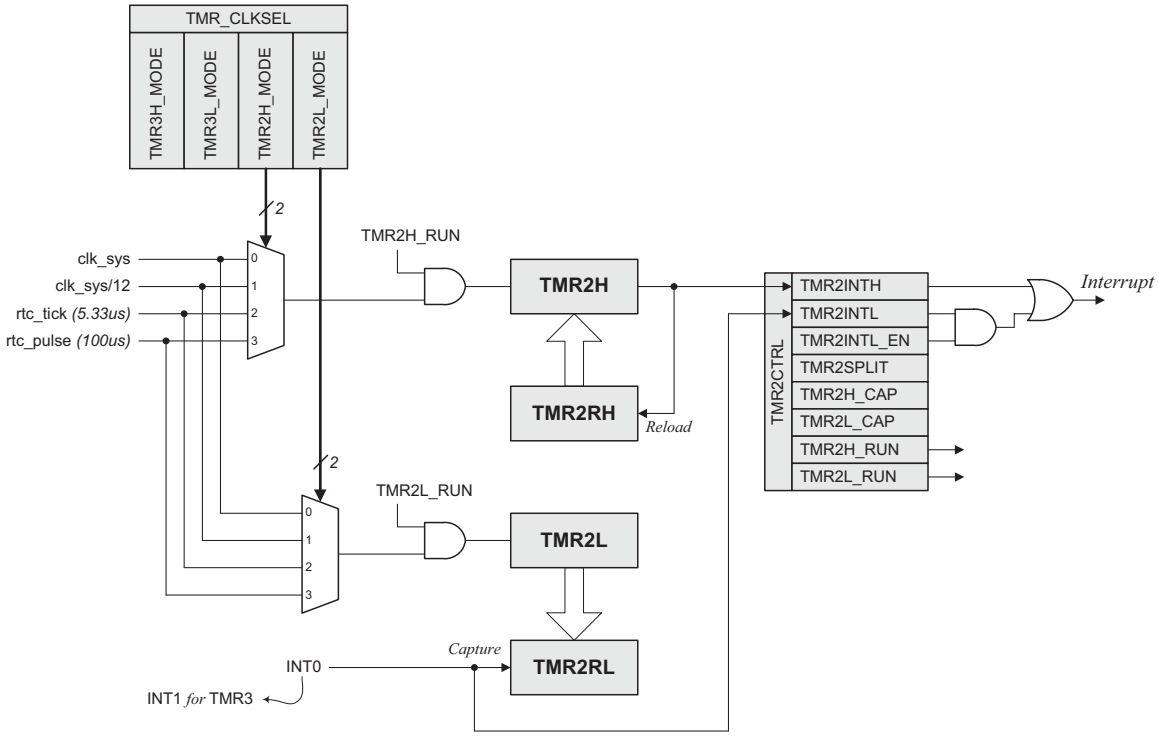


Figure 33.6. Two 8-Bit TImers in Timer/Capture Configuration (Split Mode)

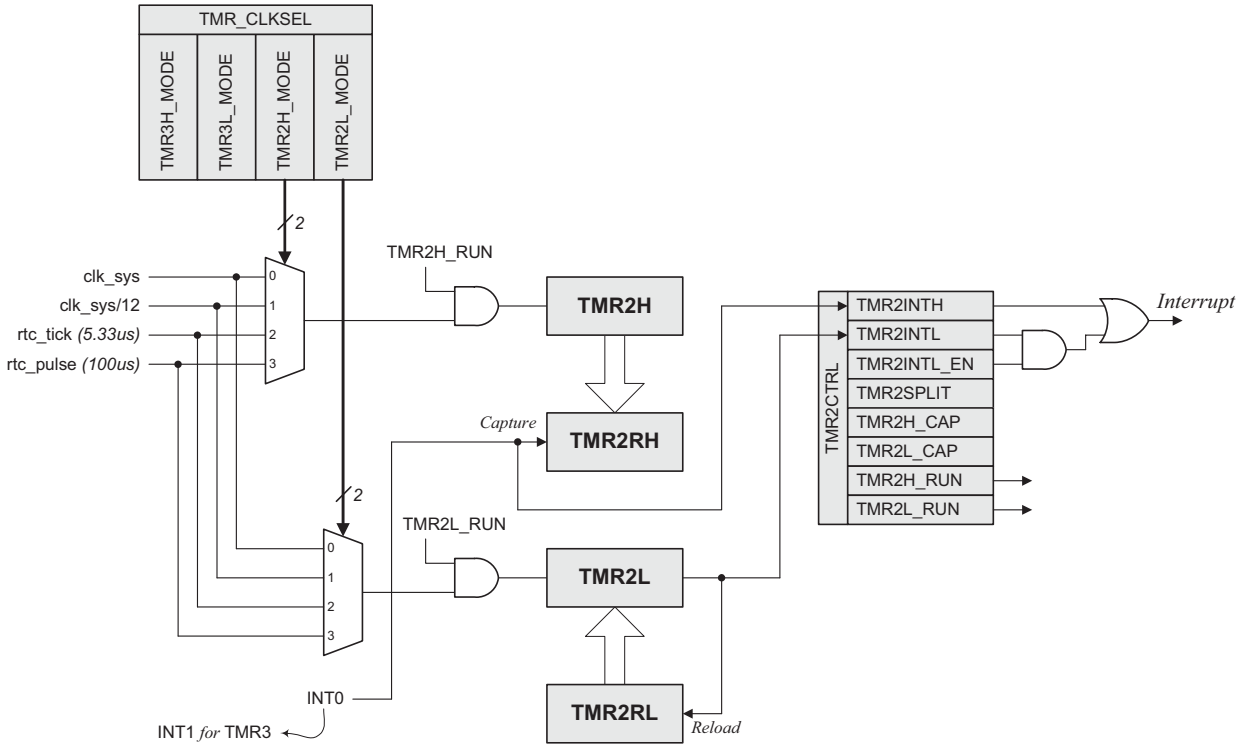


Figure 33.7. Two 8-Bit Timers In Capture/Timer Configuration (Split Mode)

SFR Definition 33.1. TMR_CLKSEL

Bit	7	6	5	4	3	2	1	0
Name	TMR3H_MODE		TMR3L_MODE		TMR2H_MODE		TMR2L_MODE	
Type	R/W		R/W		R/W		R/W	
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xC9

Bit	Name	Function
7:6	TMR3H_MODE	**Timer 3 High Byte Mode Select.** Timer 3 high half in split mode or full timer in full mode clock selection. Clock selection encoding is the same for all 4 halves. 00: CLK_SYS 01: CLK_SYS/12 10: RTC_TICK = 5.33 µs 11: RTC_PULSE = 100 µs
5:4	TMR3L_MODE	**Timer 3 Low Byte Mode Select.** Timer 3 low half in split mode or full timer in full mode clock selection. Clock selection encoding is the same for all 4 halves. 00: CLK_SYS 01: CLK_SYS/12 10: RTC_TICK = 5.33 µs 11: RTC_PULSE = 100 µs
3:2	TMR2H_MODE	**Timer 2 High Byte Mode Select.** Timer 2 high half in split mode or full timer in full mode clock selection. Clock selection encoding is the same for all 4 halves. 00: CLK_SYS 01: CLK_SYS/12 10: RTC_TICK = 5.33 µs 11: RTC_PULSE = 100 µs
1:0	TMR2L_MODE	**Timer 2 Low Byte Mode Select.** Timer 2 low half in split mode or full timer in full mode clock selection. Clock selection encoding is the same for all 4 halves. 00: CLK_SYS 01: CLK_SYS/12 10: RTC_TICK = 5.33 µs 11: RTC_PULSE = 100 µs

SFR Definition 33.2. TMR2CTRL

Bit	7	6	5	4	3	2	1	0
Name	TMR2 INTH	TMR2 INTL	TMR2 INTL_EN	TMR2 SPLIT	TMR2H_ CAP	TMR2L_ CAP	TMR2H_ RUN	TMR2L_ RUN
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xC8; Bit-Addressable

Bit	Name	Function
7	TMR2 INTH	**Timer 2 High Byte Interrupt Flag.** Interrupt flag for timer high half in split configuration or overall 16 bit timer in wide configuration. It gets set when the high half of the timer overflows or there is a capture event for the high half. This bit is not automatically cleared by hardware.
6	TMR2 INTL	**Timer 2 Low Byte Overflow Flag.** Interrupt flag for the timer low half. It gets set when the low half overflows in timer mode or by capture event of the low half in capture mode. Software must clear this bit, hardware will not clear it. This bit is set when the low half of the timer overflows even if we operate in wide configuration. When in wide configuration and in capture mode this bit is set when the high half of the timer overflows. Since in that case the capture event is the same for both halves, the capture event sets the TMR2INTH interrupt flag. Then this TMR2INTL can be used as a flag that the timer overflew, serving as an additional 17th timer bit in capture mode in wide configuration.
5	TMR2 INTL_EN	**Timer 2 Low Byte Interrupt Enable.** When set to 1, this bit enables Timer 2 Low Byte interrupts. The overall timer interrupt request signal is : TMR2 interrupt request = TMR2INTH | (TMR2INTL & TMR2INTL_EN)
4	TMR2 SPLIT	**Timer 2 Split Mode Enable.** 0: Timer operates in wide configuration as 16 bit timer. The low half controls the whole timer. 1: Timer operates in split configuration. Both halves are controlled independently.
3	TMR2H_ CAP	**Timer 2 High Byte Capture Mode Enable.** If set then TMR2H high half operates in capture mode if the timer is in split configuration mode. Ignored if the timer operates in wide configuration mode.
2	TMR2L_ CAP	**Timer 2 Low Byte Capture Mode Enable.** If set then TMR2L low half operates in capture mode if the timer is in split configuration, or the whole timer operates in capture mode if in wide configuration mode.

Bit	Name	Function
1	TMR2H_ RUN	**Timer 2 High Byte Run Model.** TMR2H high byte enable in split configuration, whole timer enable in wide configuration.
0	TMR2L_ RUN	**Timer 2 Low Byte Run Model.** TMR2L low byte enable in split configuration, whole timer enable in wide configuration.

SFR Definition 33.3. TMR2RL

Bit	7	6	5	4	3	2	1	0
Name	TMR2RL[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xCA

Bit	Name	Function
7:0	TMR2RL[7:0]	**Timer 2 Capture/Reload Register Low Byte.** TMR2RL holds the low byte of the capture/reload value for Timer 2. LSB Byte. Two halves are not double buffered. Write to each of the halves takes effect immediately. If the timer or respective half operates in capture mode this register holds the capture value. If the timer or respective half operates in timer mode this register holds the reload value.

SFR Definition 33.4. TMR2RH

Bit	7	6	5	4	3	2	1	0
Name	TMR2RH[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xCB

Bit	Name	Function
7:0	TMR2RH[7:0]	**Timer 2 Capture/Reload Register High Byte.** TMR2RH holds the high byte of the reload value for Timer 2.

SFR Definition 33.5. TMR2L

Bit	7	6	5	4	3	2	1	0
Name	TMR2L[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xCC

Bit	Name	Function
7:0	TMR2L[7:0]	**Timer 2 Low Byte Actual Timer Value.** In 16-bit mode, the TMR2L register contains the low byte of the 16-bit Timer 2. In 8-bit mode, TMR2L contains the 8-bit low byte timer value.

SFR Definition 33.6. TMR2H

Bit	7	6	5	4	3	2	1	0
Name	TMR2H[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xCD

Bit	Name	Function
7:0	TMR2H[7:0]	**Timer 2 High Byte Actual Timer Value.** In 16-bit mode, the TMR2H register contains the high byte of the 16-bit Timer 2. In 8-bit mode, TMR2H contains the 8-bit high byte timer value.

SFR Definition 33.7. TMR3CTRL

Bit	7	6	5	4	3	2	1	0
Name	TMR3 INTH	TMR3 INTL	TMR3 INTL_EN	TMR3 SPLIT	TMR3H_ CAP	TMR3L_ CAP	TMR3H_ RUN	TMR3L_ RUN
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xB9

;

Bit	Name	Function
7	TMR3 INTH	**Timer 3 High Byte Interrupt Flag.** Interrupt flag for timer high half in split configuration or overall 16 bit timer in wide configuration. It gets set when the high half of the timer overflows or there is a capture event for the high half. This bit is not automatically cleared by hardware.
6	TMR3 INTL	**Timer 3 Low Byte Overflow Flag.** Interrupt flag for the timer low half. It gets set when the low half overflows in timer mode or by capture event of the low half in capture mode. Software must clear this bit, hardware will not clear it. This bit is set when the low half of the timer overflows even if we operate in wide configuration. When in wide configuration and in capture mode this bit is set when the high half of the timer overflows. Since in that case the capture event is the same for both halves, the capture event sets the TMR3INTH interrupt flag. Then this TMR3INTL can be used as a flag that the timer overflew, serving as an additional 17th timer bit in capture mode in wide configuration.
5	TMR3 INTL_EN	**Timer 3 Low Byte Interrupt Enable.** When set to 1, this bit enables Timer 3 Low Byte interrupts. The overall timer interrupt request signal is : TMR3 interrupt request = TMR3INTH | (TMR3INTL & TMR3INTL_EN)
4	TMR3 SPLIT	**Timer 3 Split Mode Enable.** 0: Timer operates in wide configuration as 16 bit timer. The low half controls the whole timer. 1: Timer operates in split configuration. Both halves are controlled independently.
3	TMR3H_ CAP	**Timer 3 High Byte Capture Mode Enable.** If set then TMR3H high half operates in capture mode if the timer is in split configuration mode. Ignored if the timer operates in wide configuration mode.
2	TMR3L_ CAP	**Timer 3 Low Byte Capture Mode Enable.** If set then TMR3L low half operates in capture mode if the timer is in split configuration, or the whole timer operates in capture mode if in wide configuration mode.

Bit	Name	Function
1	TMR3H_ RUN	**Timer 3 High Byte Run Model.** TMR3H high byte enable in split configuration, whole timer enable in wide configuration.
0	TMR3L_ RUN	**Timer 3 Low Byte Run Model.** TMR3L low byte enable in split configuration, whole timer enable in wide configuration.

SFR Definition 33.8. TMR3RL

Bit	7	6	5	4	3	2	1	0
Name	TMR3RL[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xBA

Bit	Name	Function
7:0	TMR3RL[7:0]	**Timer 3 Capture/Reload Register Low Byte.** TMR3RL holds the low byte of the capture/reload value for Timer 3. LSB Byte. Two halves are not double buffered. Write to each of the halves takes effect immediately. If the timer or respective half operates in capture mode this register holds the capture value. If the timer or respective half operates in timer mode this register holds the reload value.

SFR Definition 33.9. TMR3RH

Bit	7	6	5	4	3	2	1	0
Name	TMR3RH[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xBB

Bit	Name	Function
7:0	TMR3RH[7:0]	**Timer 3 Capture/Reload Register High Byte.** TMR3RH holds the high byte of the reload value for Timer 3.

SFR Definition 33.10. TMR3L

Bit	7	6	5	4	3	2	1	0
Name	TMR3L[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xBC

Bit	Name	Function
7:0	TMR3L[7:0]	**Timer 3 Low Byte Actual Timer Value.** In 16-bit mode, the TMR3L register contains the low byte of the 16-bit Timer 3. In 8-bit mode, TMR3L contains the 8-bit low byte timer value.

SFR Definition 33.11. TMR3H

Bit	7	6	5	4	3	2	1	0
Name	TMR3H[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xBD

Bit	Name	Function
7:0	TMR3H[7:0]	**Timer 3 High Byte Actual Timer Value.** In 16-bit mode, the TMR3H register contains the high byte of the 16-bit Timer 3. In 8-bit mode, TMR3H contains the 8-bit high byte timer value.

34. C2 Interface

The devices include an on-chip Silicon Laboratories 2-Wire (C2) debug interface in-system debugging with the production part installed in the end application. The C2 interface uses a clock signal (C2CLK) and a bi-directional C2 data signal (C2DAT) to transfer information between the device and a host system. The C2 interface is intended to be used by the Silicon Labs or third party development tools. It is not intended to be used for any other purpose. It can be completely disabled per user programming for fully programmed chips.

34.1. C2 Pin Sharing

The C2 protocol allows the C2 pins to be shared with user functions so that in-system debugging. This is possible because C2 communication is typically performed when the device is in the halt state, where all on-chip peripherals and user software are stalled. In this halted state, the C2 interface can safely borrow the C2CLK (GPIO[5]) and C2DAT (GPIO[4]) pins. In most applications, external resistors are required to isolate C2 interface traffic from the user application. A typical isolation configuration is shown in Figure 34.1 along with the connection to the standard Silicon Labs 10-pin debugging interface header.

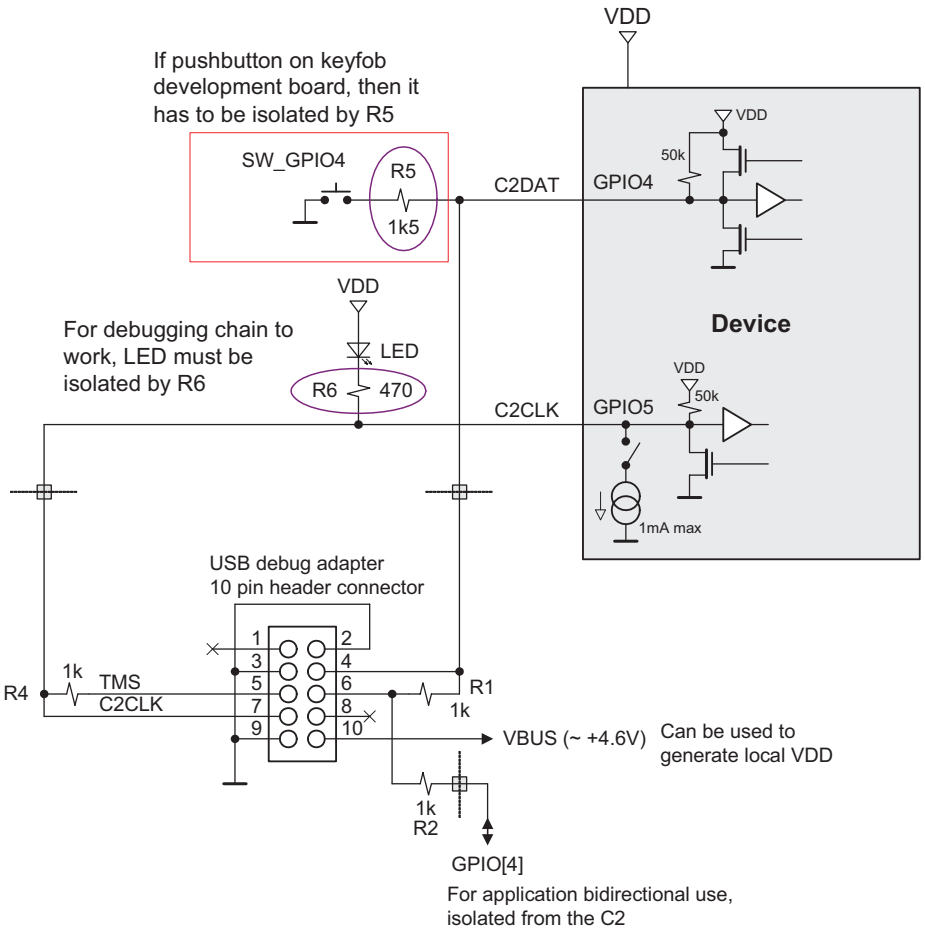


Figure 34.1. 10-pin C2 USB Debugging Adapter Connection to Device

On this device the GPIO[5] is shared with the LED current driver, which can drive up to 1mA of current to the ground. Normally the LED will be connected in between the GPIO[5] and VDD. For C2 to work the LED driver is disabled during debugging sessions, so even if the user code tries to turn the LED on, that operation will not interfere with C2 debug transactions and the actual LED current driver will not be turned on.

Whenever the user disconnects the IDE from the device by hitting the **`Disconnect`** button on the IDE, the IDE clears all the breakpoints, clears the LED driver disable (enables the LED), and runs the currently loaded user application residing in the CODE/XDATA RAM from the current position where the code was halted. If IDE is disconnected from the device the user application behaves exactly as programmed, with the LED driver driving the LED per user application. The user then can connect to the device through IDE by hitting the **`Connect`** button. The connection is only possible when the LED driver is not active. Upon connection the IDE will disable the LED driver for the duration of the debug session (until the device is **`Disconnect`**-ed).

The GPIO[4] can be used as a bidirectional input/output by a user application, but a resistive network has to be used to isolate the GPIO[4] from the C2 transactions, as shown in Figure 34.1.

Instead of the USB debug adapter the user can also use Silicon Labs ToolStick development tool. The ToolStick has a PCB edge 14 pin connector. Connection in between the device and the ToolStick for software development and debugging is in Figure 34.2.

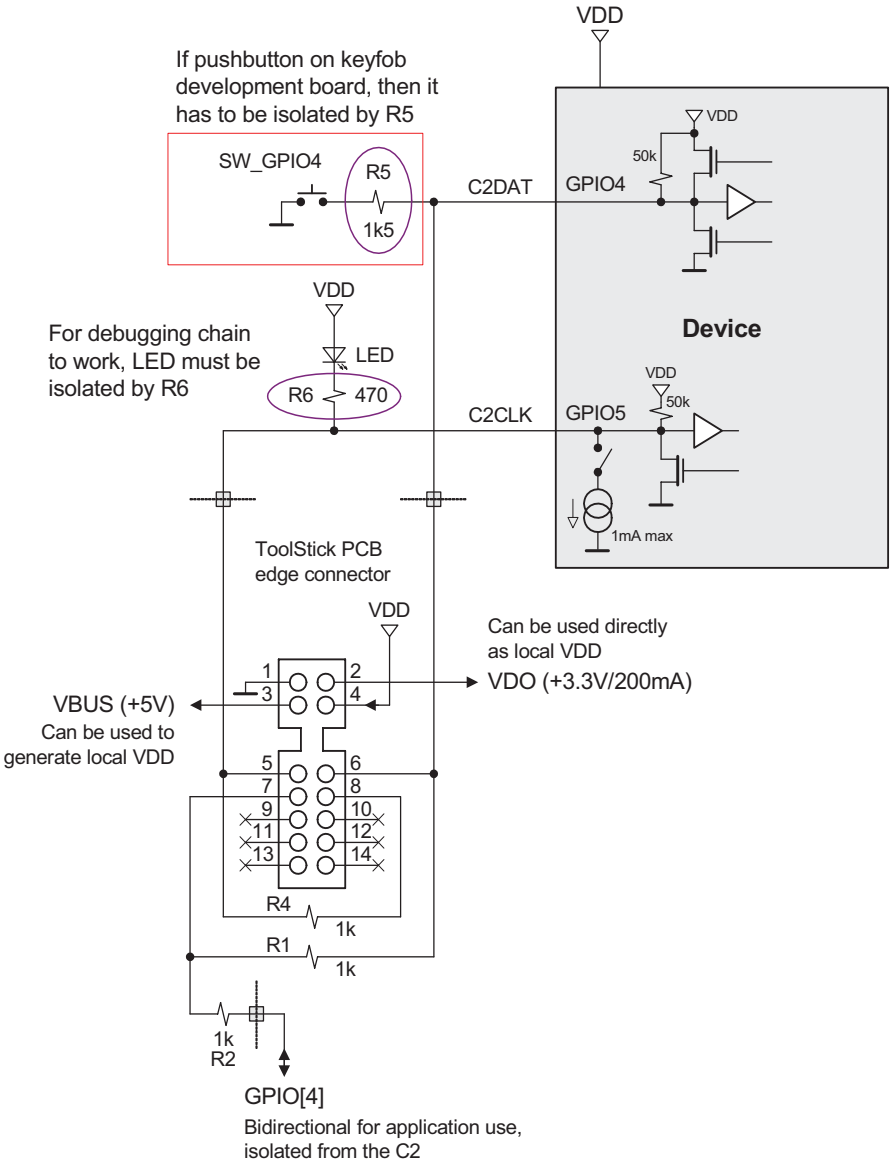


Figure 34.2. 14-Pin C2 ToolStick Connection to Device

35. IDE Development Environment and Debugging Chain

The development platform will be provided by Silicon Labs. The debugging chain consists of an evaluation board or an evaluation keyfob, USB debug adapter or a USB based ToolStick, and the Silicon Labs IDE development environment.

The debugging chain is using the C2 two wire interface to provide an on-chip debugging capability. The environment can load the standard OMF-51 object and symbol file only, not any proprietary extensions of that format as used by some tool manufacturers. For example, on Keil platform it means that the BL51 linker must be used. The IDE will not load outputs generated by the Keil LX51 linker. On Raisonance platform the output is the OMF-51 compliant and the file extension is AOF.

The IDE debugging environment has means to reset the chip without cycling the power. By pressing the **Reset** inside of the IDE the digital part of the device is reset and device startup boot sequence is invoked. All registers are reset to their initial states and all of the **Factory** values are refreshed in RAM and registers. If the part is a Factory part, the previously loaded CODE/XDATA RAM content is not disturbed. If the part is a User part then the User data region is loaded as well, overwriting the content of the CODE/XDATA RAM.

Using IDE is the only way to reset the chip without cycling the power to it or shutting it down and waking it up.

35.1. Functionality Limitations While Using IDE Development Environment

Even though using the Silicon Labs IDE development environment preserves almost all of the chip functionality, there are some limitations the user should be aware of. Given that the code is running from RAM and that the C2CLK shares the pin with LED output current driver (GPIO[5]), they are two functionality limitations for code development while using IDE:

1. The user cannot put a **Factory** or **User** chip into the shutdown mode and then wake it up by pressing a button (pulling any of the GPIO to ground). When the chip is in shutdown mode, the power to all digital is lost and therefore the RAM content with the user code will get erased.
2. The LED driver cannot be used when the device is connected to the debug adapters (USB debug adapter or a ToolStick).
3. Once the part is finalized, programmed as **Run** part, no further debugging is possible.

35.2. Chip Shutdown Limitation

While developing firmware on an unprogrammed chip the user cannot call the API function **vSys_Shutdown**() to shutdown the chip without loosing the RAM code downloaded by IDE.

Instead, the user should comment out the call to the shutdown function and replace it with a temporary code, which monitors a button press, actually monitoring P0 and P1 port inputs based on the user current port settings. If the button is pressed (input port value read as 0) then the long jump to address 0x0000 (LJMP 0x0000) should be executed. This would mimic the functionality of the chip shutdown and push button wakeup.

The limitation of this approach is that the digital logic is not reset and the current values of all the digital registers are preserved, while during the real shutdown and wakeup they are asynchronously reset during the process and the whole boot process is invoked.

Therefore, it is advisable not to rely on the reset values of any peripheral control registers and during the user application peripheral initialization the initial value should by forced to the registers by using MOV instructions rather then using ORL and ANL instructions to set or clear particular bits while relying on the SFR registers reset values.

35.3. LED Driver Usage while Using IDE Debugging Chain

To maximize utilization of the package pins the LED current driver output is shared with the debug chain clock signal C2CLK on the GPIO[5]. The debugging chain internally disables the LED driver while the device is connected to the debugging adapter. User can develop the code as if the LED were present without interfering with the debugging chain. The LED driver will not get turned on even if the user application code requests the driver to be turned on.

To share the LED and C2CLK functionality on a single pin and be able to use IDE for debugging there are some limitations and rules to follow. Figure 34.1 and Figure 34.2 show the recommended connection of the debug adapters to the device in the user application. Note that the LED must be isolated by the 470 resistor for the debugging chain to work. If the debugging in the user application is not needed then the 470 resistor is not needed either.

Facts about using the LED with IDE chain:

1. The IDE chain can connect to the device only if the LED current driver is off and the LED is not lit.
2. Once the IDE chain is connected to the device it blocks the device LED driver. Therefore, the application can be written in a normal fashion using LED as desired in the final application without worry of being disconnected from the debug chain. The only limitation is that the LED will not be lit from the application during the IDE debug session. The user will still observe LED activity, but that activity is related to the debug chain communicating with the device, not the user application driving the LED.
3. Once the IDE chain is disconnected from the device (by pressing **`Disconnect`** button in IDE, for example), the device is released from halt and at the same time the blocking of the LED driver is removed. From that point on the application behaves and runs as regular application and the LED activity reflects what the application desires to do with LED.
4. If the user wants to reconnect the IDE to the device the only requirement is that the LED must not be lit by the application at that moment. Therefore, if for whatever reason the device user software is stuck in an infinite loop and driving the LED constantly, the IDE chain will not be able to connect to the device. In such situations the device power has to be cycled to invoke internal power on reset by unplugging the keyfob from the programming or ToolStick boards and replugging again. See item 1. above.

For example, on the keyfob battery backed up development platform the user can disconnect the keyfob from the debugging platform (programming board or directly from the ToolSTick) and walk around with running application using LED as desired by the application. The only thing the user has to do is to **`Disconnect`** the keyfob from the IDE by pressing the **`Disconnect`** button. The LED gets enabled and the application runs from the point where the application is currently halted. To run the application from the very beginning, the user must press **`Reset`** on the IDE before pressing **`Disconnect`**.

35.4. LED Driver and Application Development Issues

There is a possible issue related to the LED operation and its interference with the functionality of the GPIO[4] during applicaiton development when the part is in the Factory or User state. There is no issue if the part is in the Run state.

See the Errata for possible problem description and the API documentation for possible impact of the application development and available solutions.

36. Additional Reference Resources

- AN369: Antenna Interface for the Si401x Transmitters
- AN370: Si4010 Software Programming Guide
- AN511: Si4010 NVM Burner user's guide
- AN515: Si4010 Key fob Development Kit Quick-Start Guide
- AN518: Si4010 Memory Overlay Technique
- AN526: Si4010 ROM 02.00 API Additional Library Description

DOCUMENT CHANGE LIST

Revision 0.1 to Revision 0.2

- Completely revised data sheet revision 0.1 to include MCU operation
- Reformatted data sheet to correspond with MCU data sheet format
- Removed RKE application, focus on general MCU + Tx usage
- Included 14P SOIC package and pin information
- Updated Section “4. Ordering Information” on page 15
- Updated Section “9. Electrical Characteristics” on page 26

Revision 0.2 to Revision 0.5

- Updated data sheet for revision B and C silicon
- Changed standby supply current to < 10 nA
- Increase data rate to 100 kBaud for FSK and 50 kBaud for OOK
- Corrected maximum clock frequency of the LPOSC to 24 MHz
- Updated section 2. Ordering Information to reflect the revision B and C silicon
- Updated table 7.3 DC Characteristics to reflect revision B and C silicon
- Updated table 7.4 Si4010 RF Transmitter Characteristics to reflect revision B and C silicon
- Fixed block diagram in figure 8.1. Test Block Diagram with 10-pin MSOP Package
- Updated section 10. System Description text for revision B and C silicon
- Updated section 11. Power Amplifier text for revision B and C silicon
- Updated section 23. System Boot and NVM Programming for revision B and C silicon
- Updated section 36. Additional Reference Resources to include new application notes

Revision 0.5 to Revision 0.6

- Removed revision B part numbers and replaced with revision C part numbers Si4010-C2-GT and Si4010-C2-GS

CONTACT INFORMATION

Silicon Laboratories Inc.

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
Please visit the Silicon Labs Technical Support web page:
https://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

The information in this document is believed to be accurate in all respects at the time of publication but is subject to change without notice. Silicon Laboratories assumes no responsibility for errors and omissions, and disclaims responsibility for any consequences resulting from the use of information included herein. Additionally, Silicon Laboratories assumes no responsibility for the functioning of undescribed features or parameters. Silicon Laboratories reserves the right to make changes without further notice. Silicon Laboratories makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Silicon Laboratories assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. Silicon Laboratories products are not designed, intended, or authorized for use in applications intended to support or sustain life, or for any other application in which the failure of the Silicon Laboratories product could create a situation where personal injury or death may occur. Should Buyer purchase or use Silicon Laboratories products for any such unintended or unauthorized application, Buyer shall indemnify and hold Silicon Laboratories harmless against all claims and damages.

Silicon Laboratories and Silicon Labs are trademarks of Silicon Laboratories Inc.
Other products or brandnames mentioned herein are trademarks or registered trademarks of their respective holders