iND83223 "µSesame"

indie's highly integrated, microcontroller with integrated 400MHz Transmitter and high power I/Os

9/14/15 **Preliminary Data sheet**

1.0 TABLE OF CONTENT

2.0 LIST OF TABLES

3.0 LIST OF FIGURES

4.0 REGISTER CONVENTION

Several registers will be defined and explained throughout this document. The general format of the description of the registers is as follows:

Name of the Register		Starting Address (Hex)			Reset or Default Value (Hex)		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit_Name	Bit_Name	Bit_Name	Bit_Name	Bit_Name	Bit_Name	Bit_Name	Bit_Name
MSB							LSB

Where R/W is the read and write permissions of the specific bit. An example:

RF_DCDTIME		0x50011004			0x3614		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
MAX_TE1	MAX_TE0	MIN_GB5	MIN_GB4	MIN_GB3	MIN_GB2	MIN_GB1	MIN_GB0
MIN_TE3	MIN_TE2	MIN_TE1	MIN_TE0	MAX_TE5	MAX_TE4	MAX_TE3	MAX_TE2
MSB							LSB

The name of this register is RF_DCDTIME (RF Decoder Time). It is a 16-bit register, located at address 0x50011004 and 0x50011005. The first row of data (MAX_TE[1:0], MIN_GB[5:0]) corresponds to address 0x50011004 with default value of 0x14 and the second row of data (MIN_TE[3:0], MAX_TE[5:2]) corresponds to address 0x50011005 with default value of 0x36.

5.0 General Description

This ASIC integrates all of the functions necessary to implement a car alarm module.

It has the following features:

- A super-heterodyne ISM band, 433.92MHz, ASK receiver, with up to -110dBm of sensitivity
- An ARM Cortex M0 32-bit microcontroller with 160kBytes of flash memory and 8kBytes of SRAM, with the following additional architectural features
    - System Tick Timer (SysTick – 24 bits, interruptible)
    - 3 additional 32-bit timers
    - Programmable Watch-Dog Timer
    - Built-in Nested Vectored Interrupt Controller (NVIC)
    - Built-in Wake-up Interrupt Controller (WIC)
    - Serial Wire Debugger
- A 40kHz ultra-sound Doppler-effect based intrusion detector transceiver
- 15 high voltage (9-45V) general purpose I/O ports which can source 5mA or sink 25mA
- 8 high voltage (9-45V) general purpose I/O ports, which can sink 200mA in order to directly drive a relay coil,
- 1 high voltage (9-45V) general purpose I/O port which can source 200mA or sink 25mA
- 12 low voltage (3.3V nominal) general purpose I/O ports.
- All associated power management circuits are included
- 2 x ADC (8-bit), total of 28 channels, and selectable input references.
- 2 x PWM (12-bit)
- LIN Interface (2.0)
- UART Interface
- SPI Interface
- I2C Interface
- μSesame contains three oscillators which may be used to generate a timebase
    - a 3.58MHz high accuracy crystal oscillator, which is used as a reference for the RF and for the ultrasound blocks
    - a 10MHz, 1% accurate R-C oscillator
    - a 10kHz, low power oscillator, for current saving operation
- μSesame is designed to withstand load dump events of up to 45V on its supply pin and on every high voltage I/O. It is also designed to withstand electrical discharges to its 12V I/Os according to ISO10605 standard at 8KV.

6.0 PINOUT AND PACKAGE

6.1 PACKAGE OVERVIEW

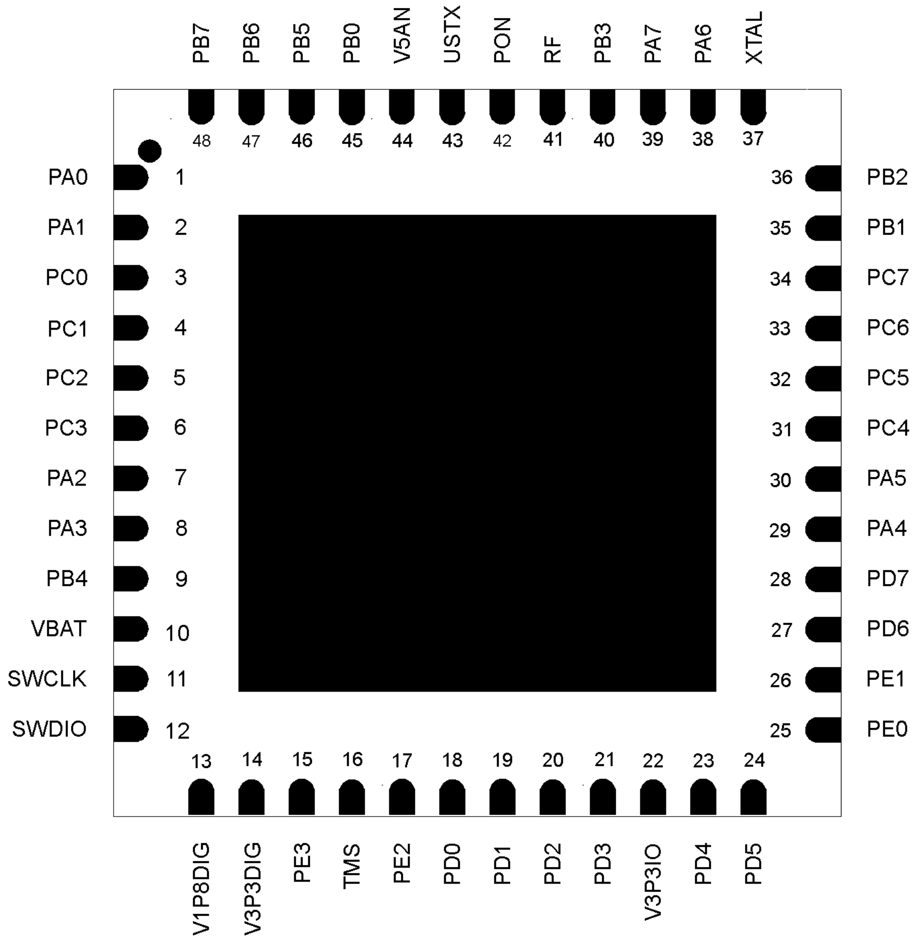
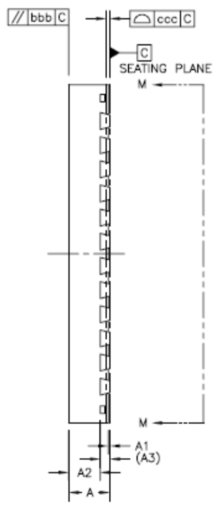


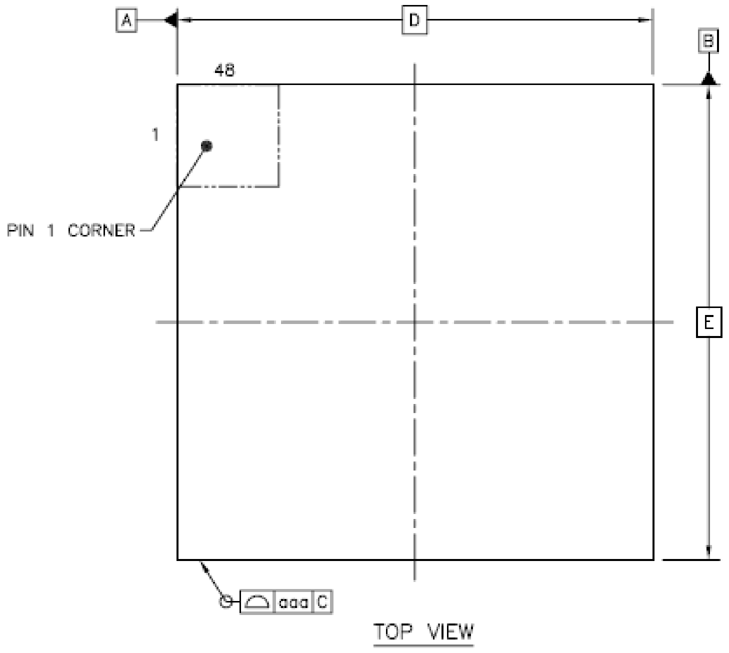
Figure 1: µSesame Pinout Diagram (Top View)

6.2 PACKAGE DIMENSIONS

The dimensions of the package are defined in the following table and drawings:

DESCRIPTION		SYMBOL	MILLIMETER		
			MIN	NOM	MAX
TOTAL THICKNESS		A	0.8	0.85	0.9
STAND OFF		A1	0	0.035	0.05
MOLD THICKNESS		A2	---	0.65	0.67
L/F THICKNESS		A3	0.203 REF		
LEAD WIDTH		b	0.2	0.25	0.3
BODY SIZE	X	D	7 BSC		
	Y	E	7 BSC		
LEAD PITCH		e	0.5 BSC		
EP SIZE	X	J	5.2	5.3	5.4
	Y	K	5.2	5.3	5.4
LEAD LENGTH		L	0.35	0.4	0.45
PACKAGE EDGE TOLERANCE		aaa	0.1		
MOLD FLATNESS		bbb	0.1		
COPLANARITY		ccc	0.08		
LEAD OFFSET		ddd	0.1		
EXPOSED PAD OFFSET		eee	0.1		





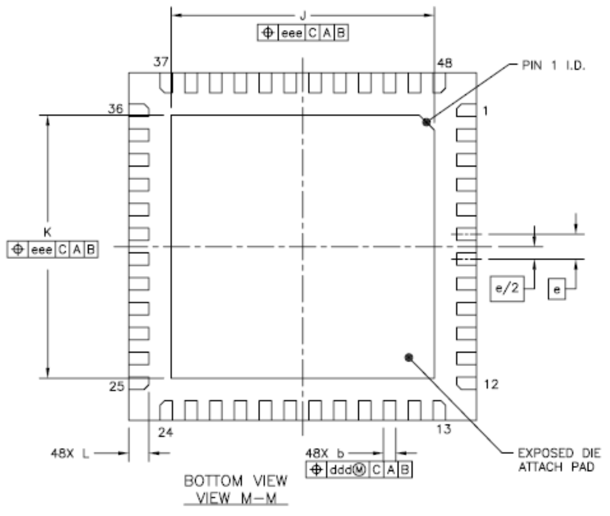


Figure 2: QFN (7mm x 7mm) 48-pin Package Dimensions

6.3 PIN DESCRIPTION

Table 1 : PIn List

Pin#	Name	Type	Description
1	PA0/CAPO	GIO	General purpose I/O operating over full Vbat range
2	PA1/P	GIO	General purpose I/O operating over full Vbat range
3	PC0/AUX3	SIO	High current, general purpose I/O operating over full Vbat range
4	PC1/AUX2	SIO	High current, general purpose I/O operating over full Vbat range
5	PC2/DTV	SIO	High current, general purpose I/O operating over full Vbat range
6	PC3/TRV/PWM2	SIO	High current, general purpose I/O operating over full Vbat range, with PWM
7	PA2/ LANTERNA	GIO	General purpose I/O operating over full Vbat range
8	PA3/MASTER	GIO	General purpose I/O operating over full Vbat range
9	PB4/TAPE	PSIO	General purpose I/O operating over full Vbat range, with high current sourcing capability
10	VBAT	Supply	9V to 45V battery voltage
11	SWCLK	Digital Input	Serial Clock Input (Debugger)
12	SWDIO	DigIO	Serial Data (Debugger)
13	V1p8DIG	Analog output	1.8V digital voltage regulator output for external circuit and/or bypass capacitor. Used internally to supply MCU and SRAM.
14	V3p3DIG (Vdd)	Analog output	Vdd, 3.3V digital voltage regulator output for external circuit and/or bypass capacitor. Used internally to supply digital circuits
15	PE3/TCK	3V3IO	3.3V I/O, or JTAG test mode clock
16	TMS	3V3IN	JTAG test mode select
17	PE2/LIN_TR_EN TDI	3V3IO	3.3V I/O, LIN_TR_EN or JTAG TDI
18	PD0/MISO	3V3IO	3.3V I/O, SPI-MISO
19	PD1/MOSI	3V3IO	3.3V I/O, SPI-MOSI
20	PD2/SCK	3V3IO	3.3V I/O or SPI-SCK
21	PD3/SSEL	3V3IO	3.3V I/O, SPI-SSEL
22	V3p3IO	Analog output	3.3V voltage regulator output for external circuit and/or bypass capacitor
23	PD4/UTXD/URXD	3V3IO	3.3V I/O, UART-TXD or UART-RXD
24	PD5/UTXD/URXD	3V3IO	3.3V I/O, UART-TXD or UART-RXD
25	PE0/SCL	3V3IO	3.3V I/O or open drain I2C SCL
26	PE1/SDA/TDO	3V3IO	3.3V I/O or open drain I2C-SDA or JTAG TDO

Table 1 : PIn List

Pin#	Name	Type	Description
27	PD6/LTXD/LRXD	3V3IO	3.3V I/O, LIN-TXD or LIN-RXD
28	PD7/LTXD/LRXD	3V3IO	3.3V I/O, LIN-TXD or LIN-RXD
29	PA4/SIREN_OUT	GIO	General purpose I/O operating over full Vbat range or sensing input for short circuit protection of siren
30	PA5/SETAE	GIO	General purpose I/O operating over full Vbat range or sensing input for short circuit protection of blinker
31	PC4/VIDRO	SIO	High current, general purpose I/O operating over full Vbat range
32	PC5/BLOQ/PWM1	SIO	High current, general purpose I/O operating over full Vbat range with PWM
33	PC6/AUX	SIO	High current, general purpose I/O operating over full Vbat range
34	PC7/SETAS/PWM1	SIO	High current, general purpose I/O operating over full Vbat range with PWM
35	PB1/SIR/PWM2	GIO	General purpose I/O operating over full Vbat range with PWM
36	PB2/PAN	GIO	General purpose I/O operating over full Vbat range)
37	XTAL	Analog In	crystal oscillator pin or internal clock input pin, requires 100nF DC block capacitor in series between the pin and crystal
38	PA6/SETAD	GIO	General purpose I/O operating over full Vbat range or sensing input for short circuit protection of blinker
39	PA7/IGN/PWM2	GIO	General purpose I/O operating over full Vbat range with PWM
40	PB3/AUX5	GIO	General purpose I/O operating over full Vbat range
41	RF	Analog In	Single ended input for ISM band ASK receiver
42	PON	Analog In/Out	40KHz, 5V, ultrasound receiver input and bias output for external buffer
43	USTX/SHKIN	Analog In/Out	40KHz, 5V, CW ultrasound transmitter, Shock sensor input
44	V5AN	Analog Out	5V voltage regulator output for external circuit and/or bypass capacitor
45	PB0/LED/PWM1	GIO	General purpose I/O operating over full Vbat range, with PWM
46	PB5/AUX4	GIO	General purpose I/O operating over full Vbat range
47	PB6/BLOQ1	GIO	General purpose I/O operating over full Vbat range
48	PB7/BLOQ2	GIO	General purpose I/O operating over full Vbat range
TAB	GND	Ground	Ground

7.0 ELECTRICAL CHARACTERISTICS

7.1 ABSOLUTE MAXIMUM RATING

Absolute maximum ratings are defined in the following table. The operation of the device above these conditions may cause lasting damage and is not recommended.

Table 2 : Absolute Maximum Ratings

Parameter	Conditions	Min.	Typ.	Max.	Unit
Vbat voltage		-0.3		+50	V
High voltage digital I/O input voltage	All GIO, SIO and SPIO pins configured as input	-0.3		Vbat+0.3	V
Low voltage digital I/O input voltage	configured as input (3V3IO), no damage	-0.3		V3p3DIG+0.3	V
3V Analog input voltage	Pins, XTAL and RF	-0.3		V3p3AN+0.3	V
5V Analog input voltage	Pins, PON and USTX	-0.3		V5AN+0.3	V
Operating Temp.	de-rated performance, full functionality	-40		+85	°C
HBM (all pins)		-8		8	kV
CDM (all pins)		-800		800	V
MM (all pins)		-400		400	V

7.2 RECOMMENDED OPERATING CONDITIONS

Table 3 : Recommended Operating Conditions

Parameter	Conditions	min	typ	max	unit
Vbat voltage		9	12	45	V
Operating Temp.		-40	25	85	°C

7.3 Current Consumption

Table 4 : Current Consumption

Name	Conditions	Min.	Typ.	Max.	Unit
Sleep Mode	All circuits disabled, Xtal enable			300	μA
Radio RX	Radio Active		6	TBD	mA
CPU	fCPU=1MHz		100		μA
Ultrasound	Both TX and RX enabled			2	mA

8.0 DEVICE OVERVIEW

The μSesame device contains the necessary hardware to realize a car alarm module. Figure 3 depicts a high-level block diagram of the device. The device subsystems are described in the following chapters.

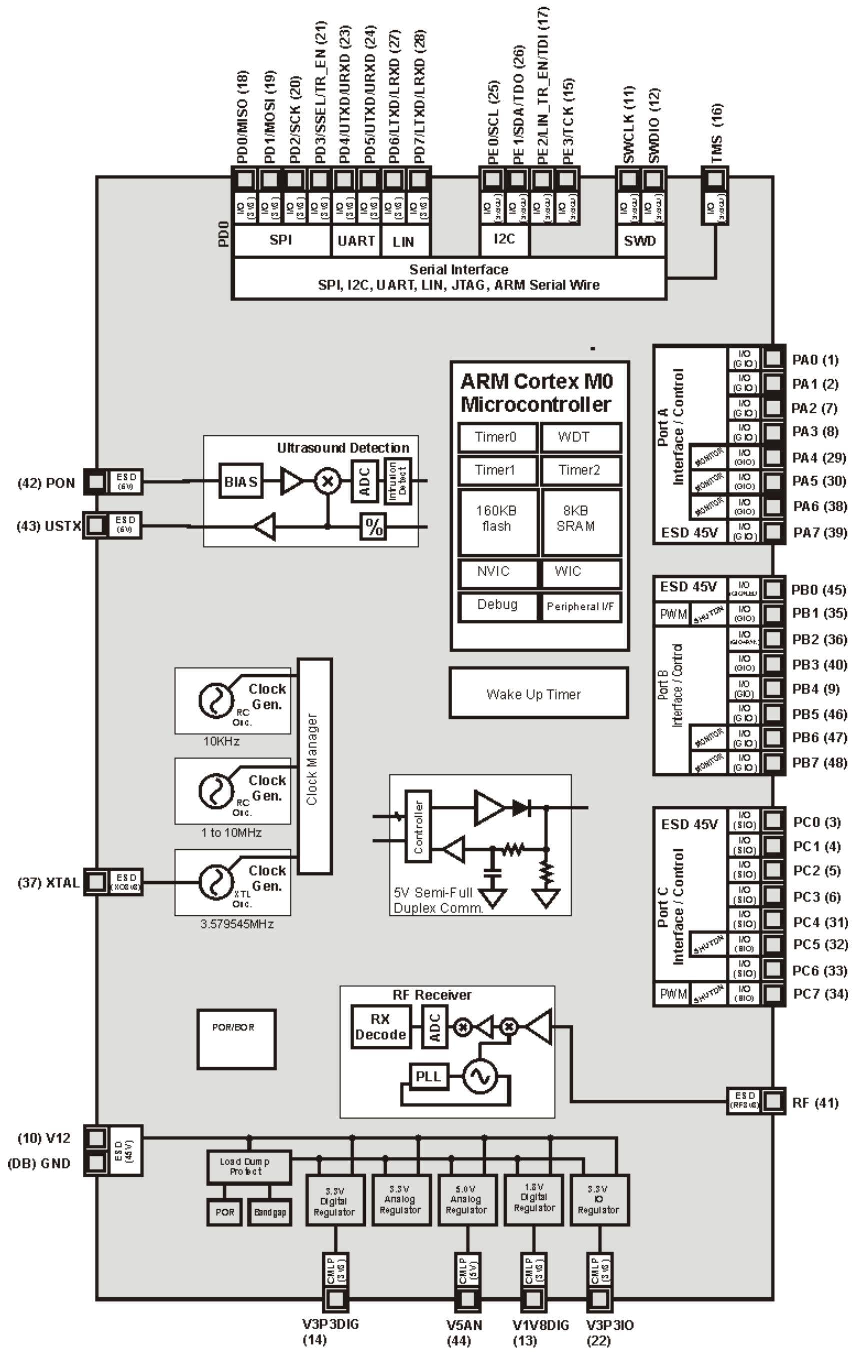


Figure 3: μSesame Block Diagram

8.1 MICROCONTROLLER SUBSYSTEM

The µSesame device includes an embedded microcontroller subsystem, which is based on the ARM Cortex M0 core. It includes a program flash memory of 160kBytes, and an SRAM of 8kBytes. It includes three 32-bit timers, plus a dedicated watchdog timer. Additionally, it includes a **N**ested **V**ector **I**nterrupt **C**ontroller (NVIC) to scheduled hardware interrupts, and a **W**akeup **I**nterrupt **C**ontroller (WIC), which enable the control of the various power modes.

Further information can be obtained in the AyDeeKay document <<AyDeeKay_Core_160_8.pdf>>.

8.1.1 Timers (0,1, and 2)

µSesame implements three identical timers: Timer0, Timer1 and Timer2. These timers use the system clock as clock source and once activated count up continuously. They start from the value initially loaded into the counting register (32-bit) and, if enabled, generate an interrupt upon rolling over (0xFFFFFFFF → 0x00000000).

8.1.1.1 Timers Registers

There are two basic registers associated with each of three timers:

TMR0REG: 32-bit Timer initial value register

TMR0REG		0x50020000			0x00000000		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
T7	T6	T5	T4	T3	T2	T1	T0
T15	T14	T13	T12	T11	T10	T9	T8
T23	T22	T21	T20	T19	T18	T17	T16
T31	T30	T29	T28	T27	T26	T25	T24
MSB							LSB
Bit31-0 **T[31:0]**: Timer Register initial value register.							

TMR0CTRL: Timer Control

TMR0CTRL		0x50020004			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	TSTART
MSB							LSB
Bit0	**TSTART**: Timer enable bit. 0 = Timer not running 1 = Timer running						

TMR1REG: 32-bit Timer initial value register

TMR1REG		0x50020008			0x00000000		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
T7	T6	T5	T4	T3	T2	T1	T0
T15	T14	T13	T12	T11	T10	T9	T8
T23	T22	T21	T20	T19	T18	T17	T16
T31	T30	T29	T28	T27	T26	T25	T24
MSB							LSB
Bit31-0	**T[31:0]**: Timer Register initial value register.						

TMR1CTRL: Timer Control

TMR1CTRL		0x5002000C			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	TSTART
MSB							LSB
Bit0	**TSTART**: Timer enable bit. 0 = Timer not running 1 = Timer running						

TMR2REG: 32-bit Timer initial value register

TMR2REG		0x50020010			0x00000000		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
T7	T6	T5	T4	T3	T2	T1	T0
T15	T14	T13	T12	T11	T10	T9	T8
T23	T22	T21	T20	T19	T18	T17	T16
T31	T30	T29	T28	T27	T26	T25	T24
MSB							LSB
Bit31-0 **T[31:0]**: Timer Register initial value register.							

TMR2CTRL: Timer Control

TMR2CTRL		0x50020014			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	TSTART
MSB							LSB
Bit0 **TSTART**: Timer enable bit. 0 = Timer not running 1 = Timer running							

8.1.1.2 Timer Operation

The operation of the timers is quite straightforward. Load the initial counter register, enable the timer and either check (polling mode) the current value of the counter register or enable the interrupt and process it inside the interrupt service routine.

Note: Inside the interrupt the application code must reload the timer counting register.

Code Example1: Enable Timer1 to count from 0xFFFF0000 and to generate interrupt:

```
TMR_Config(  1,  TIMERON,  0xFFFF0000);    //Enable  timer1  to  count  up  from
0xFFFF0000
NVIC_EnableIRQ( TIMER1_IRQn );        //Enable Timer1 interrupt

void Timer1_Handler( void )
{
  *TMR1REG = 0xFFFF0000;             //Reload Register
  //**** From this point application code inside ISR****
}
```

8.1.2 Watch Dog Timer

µSesame implements a WDT (**W**atch **D**og **T**imer) that can operate in one of two basic ways:

Interrupt Mode: In the event of a WDT rollover an interrupt will be generated.

Reset Mode: In the event of a WDT rollover the microcontroller will reset.

8.1.2.1 WDT Registers

The Watch Dog Timer implements two 32-bit registers:

WDTCTRL: WDT (**W**atch **D**og **T**imer) Control Register. (32-bit)

WDTCTRL		0x50020018			0x0000000x		
Reserved	Reserved	Reserved	R/W	R/W	R/W	R/W	R/W
-	-	-	WDTPRES1	WDTPRES0	RSTFLAG	RESETEN	WDTEN
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
MSB							LSB

Bit4-3 **WDTPRES1: WDTPRES0**: WDT Prescaler:

00 = 2^{13}/SystemClock

01 = 2^{19}/SystemClock

10 = 2^{22}/SystemClock

11 = 2^{32}/SystemClock

Bit2 **RSTFLAG**: Reset Flag. This flag is set by the system at the initialization if the initialization was caused by a reset triggered by the WDT. The bit can be de-asserted by the application.

Bit1 **RESETEN**: Reset enable. If enabled a WDT time-out will force the microcontroller to reset. This bit can be asserted but it cannot be de-asserted.

Bit0 **WDTEN**: WDT enable. This bit can be asserted but it cannot be de-asserted. It means that once the WDT is enabled it cannot be turned off until a Reset or Power-On Reset occurs.

For instance, a system running from a 30MHz Crystal with WDTPRES[1…0] = 10 will trigger the WDT after approximately 0.14seconds if not cleared properly and in time by the application.

WDTCLR: WDT Clear Register. (32-bit)

WDTCLR		0x5002001C			0x0000000x		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
WCLR7	WCLR6	WCLR5	WCLR4	WCLR3	WCLR2	WCLR1	WCLR0
WCLR15	WCLR14	WCLR13	WCLR12	WCLR11	WCLR10	WCLR9	WCLR8
WCLR23	WCLR22	WCLR21	WCLR20	WCLR19	WCLR18	WCLR17	WCLR16
WCLR31	WCLR30	WCLR29	WCLR28	WCLR27	WCLR26	WCLR25	WCLR24
MSB							LSB

Bit31-0 **WCLR[31:0]**: Clear Register. To clear the WDT counting the following words must be written in this order and without any other instruction between then:

0x3C570001

0x007F4AD6

Warning: Programming WDTCLR with other values or in the wrong order will cause the watchdog to throw an interrupt or reset the system.

Example Code: Setting and clearing the WDT. (Interrupt mode with a time of 2^22)

```
WDT_Config(WDT_INT, WDT22);   //Enable WDT in interrupt mode (2^22 system clock
cycles)
WDT_Clear();                         //Clear WDT
```

8.1.3 Interrupt Vectors

µSesame implements an interrupt vector defined in the following table:

Table 5 : Interrupt Vector Table

Cortex M0 Specific Exceptions			
Name	**Number**	**Comments**	**Required Interrupt Handler (Function)**
HardFault_IRQn	-13	HardFault handler*	HardFault_Handler (void)
SVCall_IRQn	-5	Supervisory call*	
PendSV_IRQn	-2	Interrupt-driven request for system level service*	
SysTick_IRQn	-1	SysTick Timer interrupt	void SysTick_Handler(void)
Cortex M0 Specific Exceptions			
Name	**Number**	**Comments**	**Required Interrupt Handler (Function)**
BrownOut_IRQn	0	Brownout detection interrupt	void BrownOut_Handler (void)
ClkMon_IRQn	1	Clock monitor interrupt	void ClkMon_Handler (void)
-	2	*Reserved*	
PIN_IRQn	3	Pin change interrupt	void PIN_Handler (void)
RFRE_IRQn	4	RF: Rising Edge base band signal reception interrupt	void RFRE_Handler (void)
RFFE_IRQn	5	RF: Falling Edge base band signal reception interrupt	void RFFE_Handler (void)
I2C_Collision_IRQn	6	I^{2}C Collision detection interrupt	void I2C_Collision_Handler (void)

I2C_IRQn	7	I²C event interrupt	void I2C_Handler (void)
UART_IRQn	8	UART event interrupt	void UART_Handler (void)
LIN_IRQn	9	LIN event interrupt	void LIN_Handler (void)
SPI_IRQn	10	SPI event interrupt	void SPI_Handler (void)
-	11	*Reserved*	void Default_IRQ_Handler (void)
RFMSG_IRQn	12	RF: Message received interrupt	void RFMSG_Handler (void)
IRQ13_IRQn to IRQ15_IRQn	13-15	*Reserved*	void Default_IRQ_Handler(void)
TIMER0_IRQn	16	Timer0 interrupt	void Timer0_Handler (void)
TIMER1_IRQn	17	Timer1 interrupt	void Timer1_Handler (void)
TIMER2_IRQn	18	Timer2 interrupt	void Timer2_Handler (void)
WATCHDOG_IRQn	19	Watchdog timer interrupt	void Watchdog_Handler (void)

*Note: For more information see *Cortex-M0 Devices – Generic Users Guide (ARM DUI 0497A (ID112109))* at: http://infocenter.arm.com/help/topic/com.arm.doc.dui0497a/DUI0497A_cortex_m0_r0p0_generic_ug.pdf

8.2 RF RECEIVER

µSesame implements a programmable ISM (Industrial, scientific and medical band, 300-450MHz) OOK (on-off keying) low-IF receiver. The local oscillator is generated using a fully integrated fractional-N PLL referenced to an external crystal reference. The received data is digitized using analog to digital converters before being processed by an autonomous digital section.

The receiver uses Weaver architecture for image rejection, primarily to avoid noise imaging. After amplification through an LNA, a rf mixer is used to generate I/Q signals at the IF frequency of 795KHz, where it is filtered to ~500KHz bandwidth. After the second frequency conversion, the I and Q signals are filtered to ~150kHz bandwidth. The wide bandwidth relative to the data symbol rate is necessary to accommodate manufacturing variation in the transmit and receive frequency references.

The frequency generation for the local oscillators is accomplished using a PLL locked to the crystal frequency. The VCO is a low current quadrature ring oscillator. It is expected that 3.579545MHz crystal will be utilized for frequency reference, but care has been taken to allow other potential choices of crystal, such as 4.096MHz. In the default frequency plan, the first LO is generated from 121*fXO = 433.125MHz using the PLL in integer-N mode. The second LO is generated by dividing the first LO, first by a high speed divide-by-eight prescalar, followed by a programmable divider and a quadrature divide-by-four. The default frequency plan uses divide by 17 for the programmable part, for a second LO of 796.2kHz. Due to the slight difference between the LO and IF frequencies, there is a 1.2kHz frequency offset in the baseband data, which appears as a slight additional transmit frequency error to the decoder. For other crystal frequencies, different settings will need to be programmed for the PLL and divider as described below.

After analog filtering, the baseband signal is then digitized at 298kS/s using a 12bit ADC. The digitized signal is dc-offset corrected and AM detected using a CORDIC to produce an AM baseband signal, which is filtered and decimated to an approximately 5kHz bandwidth with 18.6kS/s data rate.

Data is digitally extracted from filtered baseband signal using a digital bit slicer. An integrated decoder may be utilized to decode 1/3-2/3 duty-cycle encoded data. Decoded bits are stored in a bit buffer with capability to store messages as long as 80 bits. Once an entire valid message is stored in the RF bit buffer, an interrupt is generated. The receiver then enters an armed state, but with decoder inactive until the microcontroller re-enables the receiver to receive subsequent messages. The microcontroller should read any data from the bit buffer before re-enabling or else it will be lost.

Alternatively, if an application requires a coding scheme other than 1/3-2/3 coding, the slicer digital output may be made available in real time for the micro to decode the signal by software. The raw signal is guaranteed glitch-free, allowing a simple decode.

The receiver can be run in an autonomous “sniffing” mode with, for example, a 5% on-time duty cycle, in order to save power. Whether in sleep mode or not, intervention by the microcontroller is only required upon reception of an entire message with valid number of bits. The micro can therefore be asleep during normal RF reception, and only needs be awoken by interrupt after an entire message arrives, allowing significant power savings.

Many parameters in the RF are controllable by software. Additionally, functional control such as enabling and disabling the receiver, and the received bits are controllable through registers. The register section describes the functions of various registers related to the receiver. All control registers preset when power-on reset or software reset is asserted.

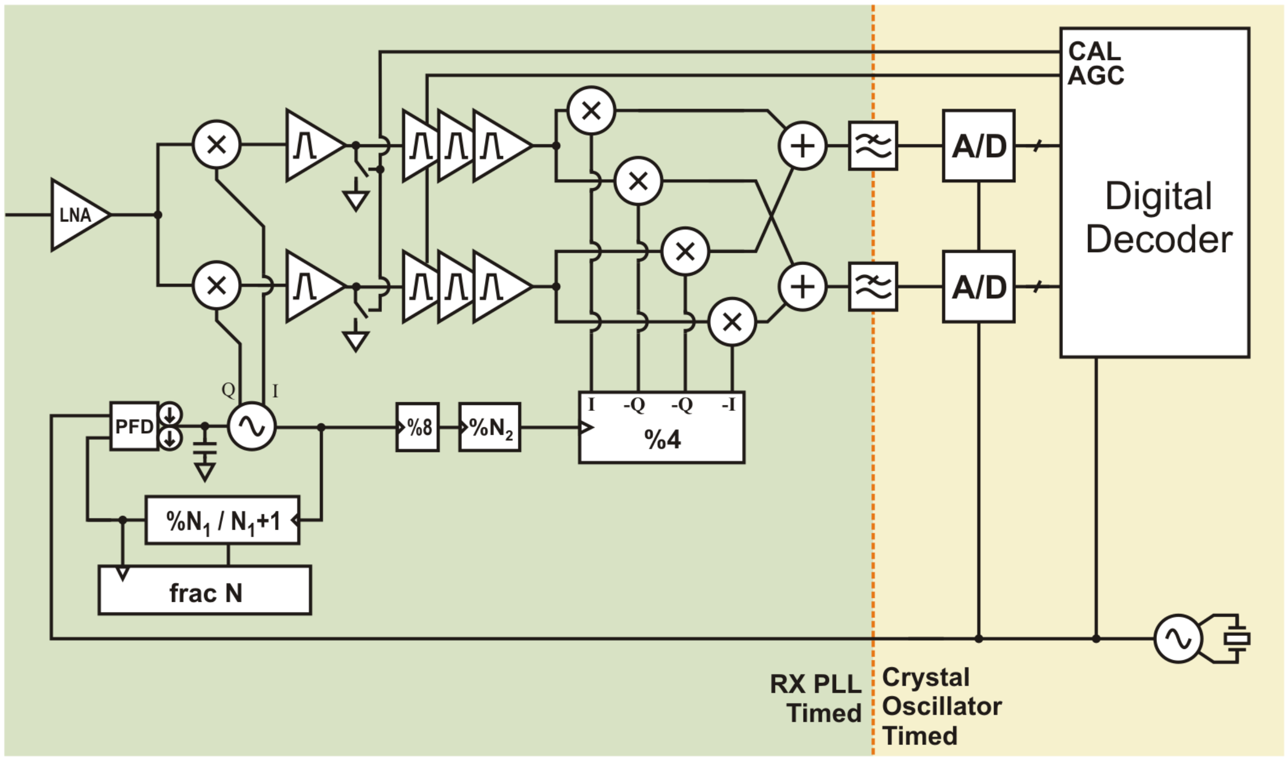


Figure 4: ASK RF receiver

Table 6 - RF specification, recommended operating conditions unless otherwise specified					
Parameter	**Conditions**	**min**	**typ**	**max**	**unit**
LNA input impedance	@433.92 MHz		10-129j		Ω
Sensitivity			-110		dBm
Frequency Range		300		470	MHz
Data Rate				5	kbps
Maximum input signal				-10	dBm
Total Gain	Voltage gain from rf input to I or Q IF outputs at default gain programming	59	71	74	dB
Spurious Emission				-60	dBm

8.2.1 RF receiver usage description

The following code fragment, shows how to configure the radio receiver to a wanted receive frequency of 433.92MHz with a minimum message length of 40 bits. Additionally, it enables message reception interrupts and employs the interrupt handler via the NVIC block in the microcontroller, and to perform a basic response to the interrupt:

```
void RF_Init ( void )
{
  // RF Setup for 433.92MHz reception

  RF_SetState( RFEN );                  //RF Enabled
  RF_SnifferMode( SNIFEN );             //Sniffer enabled
  RF_AgcControl( AGCEN, 0x1A );         //AGC Enabled
  RF_SetPllBiasTime( PLL48 );           //PLL waits 48 clock cycles
  RF_SetSleepTime(  RFSLEEP10  );           //Sleep  time  for  sniffer  is  10*1024
cycles
  RF_SetWakeTime( WAKET18 );            //Wake time from sniff is 18*1024 cycles
  *RFNX = 0x79;                         //PLL Integer divider
  *RFNF = 0x00;                         //PLL Fractional Divider
  // Hi side rejection enabled, fractional disabled, charge pump and loop filter
  RF_Setup0( HISDEN, FDIS, CPT01, 0x0A );
  //16  +DIV_LO2[1:0],  Base  band  gain  00  =  0dB,  LNA  drain  res.  1Kohm,  LNA  bias
1mA
  RF_Setup1( DIV_LO2_18, BBGAIN0dB, LNADR1K, LNABIAS1mA      );
  //Delay timer of The slicer, Attack time of The slicer, Time to allow slicer
  //to fast bias, Symbol Decimation rate
  RF_SlicerControl( ALPHA02, BETA02 , FT860uS , SDR03 );
  RF_SetMinTe( 3 );                 //(1+Min_Te)*(12*16)/3.58e6
  RF_SetMaxTe( 0x1B );              //(1 + Min_Te + Max_Te)*(12*16)/3.58e6
  RF_SetMinGb( 0x07 );              //(1     +     Min_Te     +     Max_Te     +
Min_Gb)*(12*16)/3.58e6
  RF_SetSnifRt( RTPLL, RFONDIS, 0x0B );
  RF_SetMinBitNumber( 0x28 );       //Define minimum message size in bits
}

  NVIC_EnableIRQ( RFMSG_IRQn );           //Enable RF reception interrupt
```

```
void RFMSG_Handler( void )                   // IRQ C RF Message Arrived
{
  if ( *RXNUMBER == CORRECT_FRAME_SIZE )  // If received correct number of bits
  {
    Decode_Message();                        // Decode Message
  }
  RF_SetState(RFEN);                         // Set The RF on
  RF_ReinitDecoder();                         //  Clear  The  MSGRDY  bit,  restarting
reception
}
```

8.2.2 RF Registers

The following registers define the behavior of the RF:

RF_BUFF0-9: RF Buffer Registers containing received bits.

RF_BUFF0-9		0x50000020-29			0xXX		
R	R	R	R	R	R	R	R
RXDATA7	RXDATA6	RXDATA5	RXDATA4	RXDATA3	RXDATA2	RXDATA1	RXDATA0
RXDATA15	RXDATA14	RXDATA13	RXDATA12	RXDATA11	RXDATA10	RXDATA9	RXDATA8
:	:	:	:	:	:	:	:
RXDATA71	RXDATA70	RXDATA69	RXDATA68	RXDATA67	RXDATA66	RXDATA65	RXDATA64
RXDATA79	RXDATA78	RXDATA77	RXDATA76	RXDATA75	RXDATA74	RXDATA73	RXDATA72
MSB							LSB
Bit79-0 **RXDATA[7:0]**: Received data bits. Most recently received bit is stored in RXDATA0. This register should be read when a complete message is ready (determined by reading MSG_RDY bit or having received an interrupt from the RF system) for repeatable results							

RF_NUMB: Returns the number of bits contained in the bit buffer.

RF_NUMB		0x5000002A			0xXX		
Reserved	R	R	R	R	R	R	R
-	RFNUMB6	RFNUMB5	RFNUMB4	RFNUMB3	RFNUMB2	RFNUMB1	RFNUMB0
MSB							LSB
Bit6-0	**RFNUMB [6:0]:** Returns the number of received data bits contained in the bit buffer. For repeatable results, it is recommended to only read this register when a complete message is ready						

RF_STATUS: RF Status Register.

RF_STATUS		0x5000002B			0xXX		
R/W	R/W	R	R	Reserved	Reserved	R	R
RF_EN	SNF_EN	MSG_RDY	AGC_FLG	DCDMD1	DCDMD0	SLC_OUT	RF_SLEEP
MSB							LSB

Bit7	**RF_EN**: Enables RF block. 0 = Disable the receiver / 1 = Enable the receiver
Bit6	**SNF_EN**: Enables sniff/sleep mode in the receiver 0 = Continuous mode / 1 = Sniff/Sleep mode
Bit5	**MSG_RDY**: Message ready indicator. Reading returns one if a complete message is available and the receiver is in armed mode, otherwise returns zero. Write is ignored unless in armed mode, where writing a zero causes the receiver to leave the armed state and start decoding message again
Bit4	**AGC_FLG**: AGC overflow indicator. Returns one if an overflow(signal too large) has occurred in receiver. Intended for use in continuous mode, where the micro may want to reduce the gain setting
Bit3	**DCDMD1**: Determines what to do if there is an overly-long non-guard band element 0 = reset the decoder and start looking for new message 1 = wait until a guardband is received before resetting state machine
Bit2	**DCDMD0**: Determines whether state machine transitions on edge or level 0 = transition on edge / 1 = transition on level
Bit1	**SLC_OUT**: Slicer Output
Bit0	**RF_SLEEP**: RF sleep mode indicator 0 = RF RX is active / 1 = RF RX is sleeping

RF_NBMIN: RF NBMIN Register.

RF_NBMIN		0x50011000			0x28		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
LOW_BPS	NBMIN6	NBMIN5	NBMIN4	NBMIN3	NBMIN2	NBMIN1	NBMIN0
MSB							LSB

Bit7	**LOW_BPS**: Indication to interpret all bit timings as 2X (for slow transmitters)
	0 = normal rate
	1 = count bit timings at half rate
Bit6-0	**NBMIN[6:0]**: Minimum number of bits for a valid message

RF_AGCCTRL: RF Automatic Gain Control Register.

RF_AGCCTRL		0x50011001			0x98		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
AGC_EN	AGCTRM6	AGCTRM5	AGCTRM4	AGCTRM3	AGCTRM2	AGCTRM1	AGCTRM0
MSB							LSB

Bit7 **AGC_EN**: Automatic Gain Control Enable

0 = Fixed gain mode

1 = AGC enabled

Bit6-0 **AGCTRM[6:0]**: Controls gains of analog blocks

If AGC_EN is zero, then the bits in the control registers are used to directly control the AGC trim of the analog blocks.

If AGE_EN is one, then the bits sets the default gain and reduction step

Bit4 (0 = gain steps down the table in increment of 1; 1= gain steps down the table in increment of 2)

Bit3-0	st3	st2	LNA	gain	delta
0000	00	00	000	-4.7dB	
0001	00	00	010	-1.0dB	3.7dB
0010	00	00	011	4.5dB	5.5dB
0011	00	00	100	9.7dB	5.2dB
0100	00	00	101	15.4dB	5.7dB
0101	00	00	110	21.0dB	5.6dB
0110	00	00	111	26.8dB	5.8dB
0111	00	10	111	32.4dB	5.6dB
1000	10	11	111	38.6dB	6.2dB (default)
1001	11	11	111	45.5dB	6.9dB
1010	11	11	111	52.6dB	7.1dB
1011	not allowed				
11xx	not allowed				

RF_SLCCTRL: RF Slicer Control Register

RF_SLCCTRL		0x50011002			0xAB		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ALPHA1	ALPHA0	BETA1	BETA0	FTIME1	FTIME0	DR_SYM1	DR_SYM0
MSB							LSB

Bit7-6	**ALPHA[1:0]**: Controls decay time of slicer level. When input is inside slicer levels, slicer decays according to equation (clocked at decimated data rate):
	y[n] = (1-ALPHA)*y[n-1] + ALPHA*x[n]
	00 = 1/256 (fastest decay rate)
	01 = 1/512
	10 = 1/1024
	11 = 1/2048 (slowest decay rate)
Bit5-4	**BETA[1:0]**: Controls attack time of slicer level. When input is outside slicer levels, slicer grows according to equation (clocked at decimated data rate):
	y(n) = (1-BETA)*y[n-1] + BETA*x[n]
	00 = 1/2 (fastest attack rate)
	01 = 1/4
	10 = 1/8
	11 = 1/16 (slowest attack rate)
Bit3-2 fxo/64	**FTIME[1:0]**: Controls the time to allow the slicer to fast bias, measured in clock cycles of
	00 = 1 cycle (18us for 3.58MHz crystal)
	01 = 32 cycles (570us for 3.58MHz crystal)
	10 = 48 cycles (860us for 3.58MHz crystal)
	11 = 64 cycles (1.14ms for 3.58MHz crystal)
Bit1-0	**DR_SYM[1:0]**: Sets post-CORDIC decimation rate.
	00 = 13X
	01 = 14X
	10 = 15X
	11 = 16X

RF_SYSTIME: RF System Time Register.

RF_SYSTIME		0x50011003			0x5A		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PLLTIME1	PLLTIME0	SLPTIME2	SLPTIME1	SLPTIME0	WAKETIME2	WAKETIME1	WAKETIME0
MSB							LSB

Bit7-6	**PLLTIME[1:0]**: Controls the time to wait for PLL to bias.
	Measured in clock cycles of the fxo/64
	00 = 32 cycles (570us for 3.58MHz crystal)
	01 = 48 cycles (860us for 3.58MHz crystal)
	10 = 64 cycles (1.15ms for 3.58MHz crystal)
	11 = 128 cycles (2.29ms for 3.58MHz crystal)
Bit5-3	**SLPTIME[2:0]**: Controls the sleep time between sniff cycles.
	Measured in clock cycles of the fxo/64.
	000 = 4*1024 cycles (73ms for 3.58MHz crystal)
	001 = 6*1024 cycles (110ms for 3.58MHz crystal)
	010 = 8*1024 cycles (146ms for 3.58MHz crystal)
	011 = 10*1024 cycles (183ms for 3.58MHz crystal)
	100 = 12*1024 cycles (220ms for 3.58MHz crystal)
	101 = 14*1024 cycles (256ms for 3.58MHz crystal)
	110 = 32*1024 cycles (586ms for 3.58MHz crystal)
	111 = 128*1024 cycles (2.34s for 3.58MHz crystal)
Bit2-0	**WAKETIME[2:0]**: Controls the time to stay awake after seeing a valid guard band.
	Measured in clock cycles of the fxo/64.
	000 = 8*1024 cycles (146ms for 3.58MHz crystal)
	001 = 10*1024 cycles (183ms for 3.58MHz crystal)
	010 = 12*1024 cycles (220ms for 3.58MHz crystal)
	011 = 14*1024 cycles (256ms for 3.58 MHz crystal)
	100 = 16*1024 cycles (293ms for 3.58MHz crystal)
	101 = 18*1024 cycles (330ms for 3.58MHz crystal)
	110 = 24*1024 cycles (440ms for 3.58MHz crystal)
	111 = 32*1024 cycles (586ms for 3.58 MHz crystal)

RF_DCDTIME: RF Decode Time Control Register.

RF_DCDTIME		0x50011004/5			0x3614		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
MAX_TE1	MAX_TE0	MIN_GB5	MIN_GB4	MIN_GB3	MIN_GB2	MIN_GB1	MIN_GB0
MIN_TE3	MIN_TE2	MIN_TE1	MIN_TE0	MAX_TE5	MAX_TE4	MAX_TE3	MAX_TE2
MSB							LSB

Bit5-0 **MIN_GB[5:0]**: Minimum number of ADDITIONAL samples after passing MIN_TE and MAX_TE for a low element to be considered a valid guard band length. Default setting with 3.58MHz clock corresponds to minimum guard band time of 3.6ms

Bit11-6 **MAX_TE[5:0]**: Maximum number of ADDITIONAL samples after passing MIN_TE for an edge to be considered short enough to be valid. Default setting with 3.58MHz clock Corresponds to maximum element time of 1.45ms

Bit15-12**MIN_TE[3:0]**: Minimum number of samples for a valid element time. Counted in decimated data rate (fxo/(12*DR_SYM)). Default setting with 3.58MHz clock corresponds to minimum element time of 160us

RF_SNIFMODE: RF Sniff Mode Regiseter.

RF_SNIFMODE		0x50011006			0x0B		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
RT_SEL1	RT_SEL0	RF_ON	Reserved	SNIFF_NE3	SNIFF_NE2	SNIFF_NE1	SNIFF_NE0
MSB							LSB

Bit7-6 **RT_SEL[1:0]**: Selects a source for real-time output.
00 = supervisor clock
01 = decimator output (serialized stream)
10 = slicer output
11 = PLL_EN (high if analog blocks enabled)

Bit5 **RF_ON**: Force analog RF to stay on (for test).
0 = Normal
1 = RF stays on for test

Bit3-0 **SNIFF_NE[3:0]**: Number of edges to check in each sniff cycle before committing to a long Wake cycle. When in sniff mode, the first SNIFF_NE edges are tested for valid timing. If any one of these first edges is badly timed, then the receiver will go to sleep

RF_AGCMON: AGC Monitor Register

RF_AGCMON		0x50011007			0xXX		
R	R	R	R	R	R	R	R
Reserved	AGCDATA6	AGCDATA5	AGCDATA4	AGCDATA3	AGCDATA2	AGCDATA1	AGCDATA0
MSB							LSB
Bit6-0 **AGCDATA[6:0]**: RF gain control value							

RF_SIGI: I Channel Calibration Monitor Register

RF_SIGI		0x50011008/9			0xXXXX		
R	R	R	R	R	R	R	R
CAL_I7	CAL_I6	CAL_I5	CAL_I4	CAL_I3	CAL_I2	CAL_I1	CAL_I0
Reserved	Reserved	Reserved	Reserved	CAL_I11	CAL_I10	CAL_I9	CAL_I8
MSB							LSB
Bit11-0 **CAL_I[11:0]**: Channel I DC calibration value							

RF_SIGQ: Q Channel Calibration Monitor Register

RF_SIGQ		0x5001100A/B			0xXXXX		
R	R	R	R	R	R	R	R
CAL_Q7	CAL_Q6	CAL_Q5	CAL_Q4	CAL_Q3	CAL_Q2	CAL_Q1	CAL_Q0
Reserved	Reserved	Reserved	Reserved	CAL_Q11	CAL_Q10	CAL_Q9	CAL_Q8
MSB							LSB
Bit11-0 **CAL_Q[11:0]**: Channel Q DC calibration value							

RF_SLCHI: Peak Detector High Value Monitor Register

RF_SLCHI		0x5001100C/D			0xXXXX		
R	R	R	R	R	R	R	R
PDHI7	PDHI6	PDHI5	PDHI4	PDHI3	PDHI2	PDHI1	PDHI0
PDHI15	PDHI14	PDHI13	PDHI12	PDHI11	PDHI10	PDHI9	PDHI8
MSB							LSB
Bit15-0 **PDHI[15:0]**: Peak detector high value							

RF_SLCLO: Peak Detector Low Value Monitor Register

RF_SLCLO		0x5001100E/F			0xXXXX		
R	R	R	R	R	R	R	R
PDLO7	PDLO6	PDLO5	PDLO4	PDLO3	PDLO2	PDLO1	PDLO0
PDLO15	PDLO14	PDLO13	PDLO12	PDLO11	PDLO10	PDLO9	PDLO8
MSB							LSB
Bit15-0 **PDLO[15:0]**: Peak detector low value							

RF_NX: Controls the integer portion of the PLL feedback divider.

RF_NX		0x50018004			0x79		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0
MSB							LSB
Bit7-0 **NX[7:0]**: Integer part of the divider value for PLL divider. If (F_EN = 0) $f_{LO} = f_{xo} * NX$ If (F_EN = 1) $f_{LO} = f_{xo} * (NX + NF/256)$							

RF_NF: Controls the fractional portion of the PLL feedback divider.

RF_NF		0x50018005			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
NF7	NF6	NF5	NF4	NF3	NF2	NF1	NF0
MSB							LSB

Bit7-0 **NF[7:0]**: Fractional part of the divider value for PLL divider.

If (F_EN = 0) $f_{LO} = f_{xo} * NX$

If (F_EN = 1) $f_{LO} = f_{xo} * (NX + NF/256)$

RF_FETRIM0: RF Front-end Trim0 Register.

RF_FETRIM0		0x50018006			0x9A		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
HI_SD	F_EN	CP_TRM1	CP_TRM0	LF_TRM3	LF_TRM2	LF_TRM1	LF_TRM0
MSB							LSB

Bit7 **HI_SD**: Controls image reject mixer to control whether to use high-side or low-side mixing

0 = low-side

1 = high-side

Bit6 **F_EN**: Controls fractional mode

0 = integer-N mode

1 = fractional-N mode

Bit5-4 **CP_TRM[1:0]**: Adjust charge pump current in PLL for optimum settling time

Bit3-0 **LF_TRM[3:0]**: Adjust loop filter stabilization resistor in PLL to control overshoot

RF_FETRIM1: RF Front-end Trim1 Register.

RF_FETRIM1		0x50018007			0x4A		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
DIV_LO21	DIV_LO20	BBGAIN1	BBGAIN0	LNA_DRES	LNAB_PG2	LNAB_PG1	LNAB_PG0
MSB							LSB

Bit7-6	**DIV_LO2[1:0]**: Divide value from PLL frequency to LO2 frequency ($f_{LO2} = f_{LO1}/(32*DIV\_LO2)$)
	00 = 8*15*4
	01 = 8*16*4
	10 = 8*17*4
	11 = 8*18*4
Bit5-4	**BBGAIN[1:0]**: Sets baseband amplifier gain
	00 = 0.6 dB
	01 = 6.0 dB
	10 = 11.3 dB
	11 = 15.8 dB
Bit3	**LNA_DRES**: LNA drain resistor
	0 = 2kΩ
	1 = 1kΩ
Bit2-0	**LNAB_PG[2:0]**: Adjust loop filter stabilization resistor in PLL
	I = 200uA*Bit0 + 500uA*Bit1 + 1mA*Bit2

8.3 ULTRASOUND TRANSCEIVER

An ultrasound intrusion sensor is a transceiver transmitting 40kHz ultrasound to the environment and receiving the reflected audio signal back through two piezo transducers. If sound is reflected by any moving object with orthogonal portion of the speed relative to the sensors is above ~6km/h, with enough surface area, the sensor will trigger the alarm. Due to the Doppler effect the receiving audio from approaching object will be at a higher frequency than the transmitted signal and from a distancing object will be at lower frequency.

The transmitter consists of a continuous wave source at 40kHz, which is generated by dividing down from the 3.58MHz crystal oscillator. The receiver is an AM demodulator in digital domain, which detects any AM modulation in the received signal with frequencies between 20Hz and 200Hz with more than 0.9% modulation depth in the highest sensitivity setting.

Furthermore, the receiver provides amplitudes of demodulated I and Q signals, which can be used by a more complex detection algorithm.

8.3.1 Receiver

First stage of the receiver is an external bipolar transistor (2SC4081 or equivalent) common emitter amplifier mounted with the transducer. This way the signal is amplified before being sent to main board through a 1-2m cable, in which unwanted interference signals may be picked up electromagnetically from sources outside the vehicle causing false triggering of the alarm. This bipolar transistor is biased with a constant dc current by from μSesame.

The integrated part of the receiver comprises an external buffer, DC bias current circuit, three stage DC coupled low pass amplifiers with one continuous time balun/amplifier with single ended input and differential output followed by two differential switched capacitor amplifiers/filters. The gain of the last stage amplifier can be changed digitally.

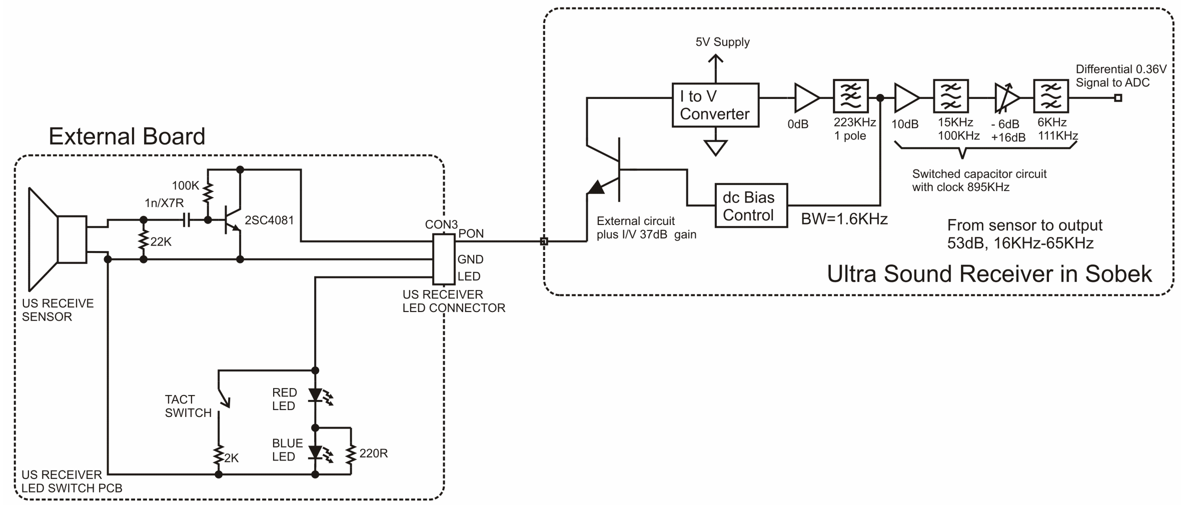


Figure 5– Ultra Sound Receiver Block Diagram

The received signal is converted from analog to digital via an ADC. The receiver operates at a data rate of 160kHz, digitally generated from the crystal oscillator clock using a fractional-N clock divider. The integer and fractional divide values are programmable using the US_NX and US_NF register settings. This divider generates a strobe signal at 160kHz, which is further divided into four phases of 40kHz for signal sampling. Since the transmit signal is also derived from this 160kHz frequency, the demodulation is homodyne.

The ADC is set to sample at 160kS/s. The first sample is subtracted from the third sample to generate an I phase data. Similarly, the second sample is subtracted from the fourth sample to generate Q phase data. After four samples are received to generate the I/Q data, a CORDIC produces sqrt(I^2+Q^2), which is used as the amplitude of the waveform.

This amplitude data is low pass filtered with a single pole filter with a bandwidth set to either 100Hz or 200Hz selectable through the US_LPF register bit. This signal is then decimated by a factor of 18 using an accumulate and dump decimator to produce a 2.22kS/s data stream.

The signal is then filtered with either one or two high pass filter poles. The number of filters is selectable by US_HPF[4], and the bandwidth of each is selectable to be one of four possible values using the US_HPF[3:2] and US_HPF[1:0] bits for the first and second filter respectively. After filtering, a detection threshold programmable using the US_THRES register word is subtracted. The detection threshold can be programmed from 0.1% to 9.4% of the full scale ADC signal level in 16 approximately log spaced steps. The sign of this signal is passed to a state machine, which detects the presence of an intruder according to a proprietary detection algorithm.

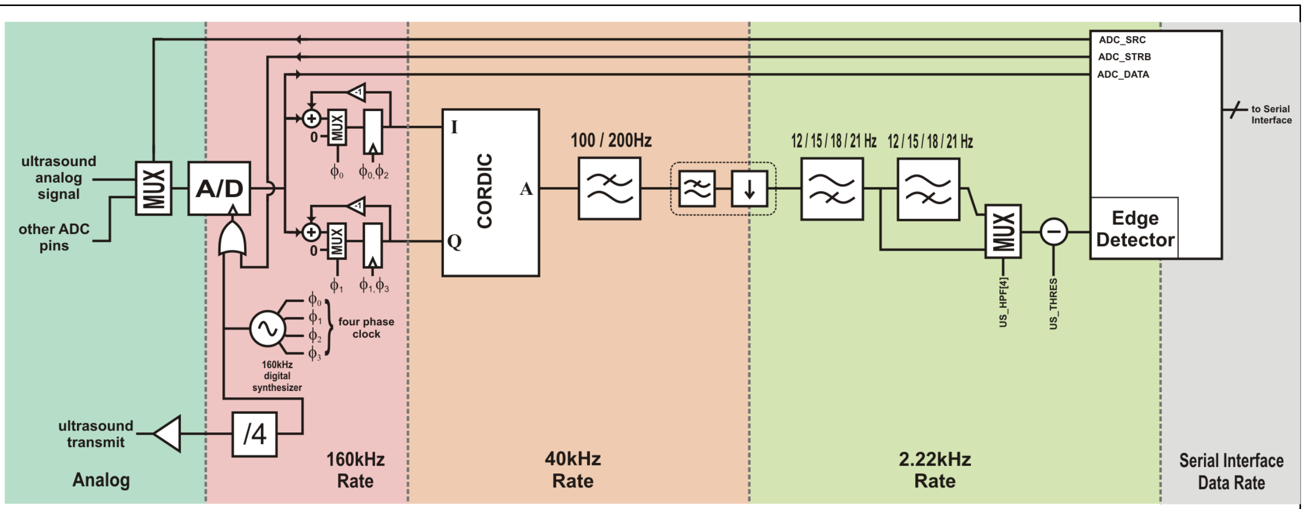


Figure 6– Ultra Sound Receiver Digital System

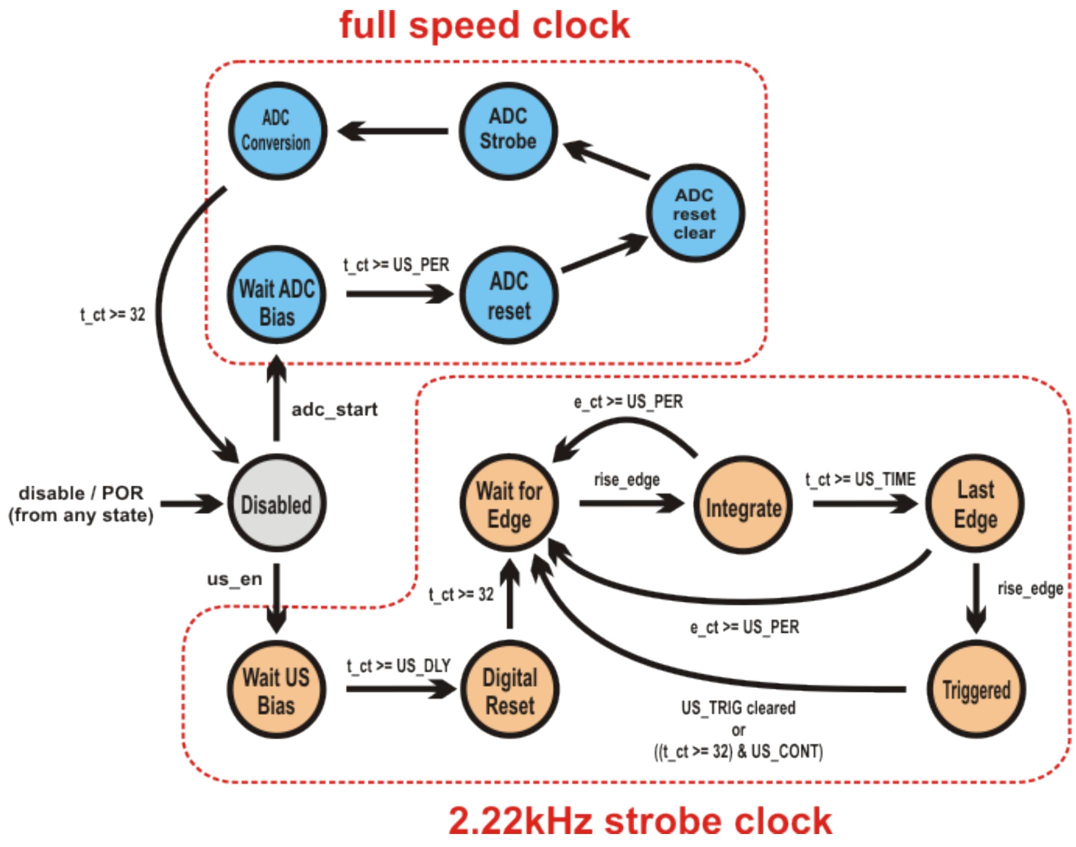


Figure 7– Ultra Sound State Machine

The detection state machine is depicted in Figure 7. The state machine utilizes two counters: t_ct counts the number of cycles elapsed in the present state, and e_ct counts the number of cycles since the last rising edge of the ultrasound signal. The number of cycles is counted at the clock rate of the system, according to the state it is in, as explained below.

The state machine is divided into two separate parts. The main function is enabled by setting the US_EN register bit, which activates the ultrasound transmit and receive subsystems. This function has the state machine timed at 2.22kHz (40kHz divided by 18). The system starts by enabling all the analog components and waiting for a programmable time until they are fully biased. Then the digital system is reset, which also has the effect of setting all filter outputs and state to zero. The system then waits until it sees the first rising edge of the ultrasound signal. It then goes to a state which counts the total time elapsed while having rise-to-rise time not longer than a programmable value set by US_PER. If any rise-to-rise transition, as measured by the e_ct counter, is longer than the value programmed by US_PER then the system returns to the “Wait for Edge” state. As long as edges come with small enough period, the e_ct counter never achieves the value set by US_PER so that the t_ct counter can reach the alarm trigger time set by US_TIME. In this case, the system moves to the “Last Edge” state. Here it will wait to see one additional correctly timed edge before alarming. If the next edge occurs too late, the system resets back to “Wait for Edge”. Since the t_ct counter resets every time the system changes state, this forces there to be at least US_TIME duration with fast enough edges and guarantees alarm with at least US_TIME+US_PER such time.

If the “Triggered” state is entered, an interrupt is generated. The US_TRIG bit is set, as well as the appropriate bit in the interrupt vector. The analog bias, 40kHz transmit signal, and digital DSP blocks remain on, but the edge decoder is deactivated. The edge decoder can be activated by clearing the US_TRIG bit. The ultrasound function can be disabled from any state by clearing the US_EN bit.

µSesame detects the receiver input open and short conditions. This information is stored in US_RXFAULT register.

Table 7 : Ultrasound Receiver Performance Specification, Recommended Operating Conditions unless otherwise specified

Parameter	Conditions	min	typ	max	unit
Carrier amplitude from US transducer		0.5	1	1.5	mV
Carrier frequency		20	40	80	kHz
Trigger threshold of envelope amplitude at transducer	Level 1- largest environment	6	9	12	µV
	Level 2	8	12	16	µV
	Level 3	14	20	28	µV
	Level 4- smallest environment	35	50	70	µV
Total equivalent input noise at transducer				0.6	µV
Programmable external Bipolar transistor bias current	USIBIAS=001 – 111	100		900	µA
Min Voltage Gain Input Output	USGAIN=0000, Voltage gain from transducer to output rBase=80k∧, 120k∧,		41		dB
Max Voltage Gain Input Output	USGAIN=1111, Voltage gain from transducer to output rBase=80k∧, 120k∧,		64		dB
Receiver 3dB bandwidth		16		65	kHz
Supply Rejection	from 5V supply to output at 40KHz		17		dB

8.3.2 Transmitter

The ultrasound transmitter is a 5V inverter with average 1.2mA and peak 26mA capability to drive the ultrasound piezo transducer at 40kHz. The 40kHz signal is generated by programmable fractional division of the crystal oscillator output signal. The shape of output waveform can be controlled by TX_WF bit in US_THRES register. Furthermore, the ultrasound transmitter may be disabled separately from the RX with US_TX_ON bit in US_STATUS register, in order to alleviate any potential interference issues with the RF receiver.

The ultrasound TX open/short is detected through auxiliary ADC. SMP_CYC bits in US_THRES register controls when to generate ADC sampling strobe for open/short detection.

8.3.3 Ultrasound Related Registers

US_STATUS: Ultrasound System Status Register.

US_STATUS		0x50000040			0x02		
R/W	R/W	Reserved	R/W	Reserved	R/W	R/W	Reserved
US_EN	US_TRIG	-	US_INTTYPE	US_ACTIVE	US_CONT	US_TX_ON	-
MSB							LSB

Bit7	**US_EN**: Writing one turns on the US sensor. Read returns one if US sensor is enabled.
Bit6	**US_TRIG**: Read returns one if triggered. Writing zero clears the triggered condition of the US sensor.
Bit4	**US_INTTYPE:** Control when to throw an interrupt for ultrasound
	0 = when integrator timeout happens
	1 = when DSP rising edge is detected
Bit3	**US_ACTIVE:** Ultrasound Sensor Ready
	0 = Ultrasound Sensor not ready
	1 = Ultrasound Sensor ready
Bit2	**US_CONT:** Ultrasound Sensor Continuous Mode
	0 = Require software rearm after a trigger
	1 = Auto-rearm the ultrasound after a trigger
Bit1	**US_TX_ON**: Ultrasound Transmitter Enable Signal
	0 = Transmitter is off
	1 = Transmitter is on

US_TIME: Ultrasound System Time Register.

US_TIME		0x50000041			0x67		
R/W	R/W	Reserved	R/W	Reserved	R/W	R/W	Reserved
US_DLY1	US_DLY0	US_PER1	US_PER0	US_TIME3	US_TIME2	US_TIME1	US_TIME0
MSB							LSB

Bit7-6	**US_DLY[1:0]:** Ultrasound Bias Time (Time to allow analog to settle)
	00 = 24 cycles (11ms)
	01 = 32 cycles (14ms)
	10 = 64 cycles (29ms)
	11 = 128 cycles (58ms)
Bit5-4	**US_PER[1:0]:** period/frequency threshold for detection
	00 = 144 cycles (65ms) : 15.4Hz
	01 = 128 cycles (58ms) : 17.2Hz
	10 = 112 cycles (50ms) : 20.0Hz
	11 = 88 cycles (40ms) : 25.0Hz
Bit3-0	**US_TIME[3:0]:** Minimum length of time required for integrator to trigger
	0000 = 288 cycles (130ms)
	0001 = 320 cycles (144ms)
	0010 = 352 cycles (158ms)
	0011 = 384 cycles (173ms)
	0100 = 416 cycles (187ms)
	0101 = 448 cycles (202ms)
	0110 = 480 cycles (216ms)
	0111 = 512 cycles (230ms)
	1000 = 544 cycles (245ms)
	1001 = 576 cycles (259ms)
	1010 = 640 cycles (288ms)
	1011 = 704 cycles (317ms)
	1100 = 768 cycles (346ms)
	1101 = 1024 cycles (461ms)
	1110 = 1536 cycles (691ms)
	1111 = 1920 cycles (864ms)

US_THRES: Ultrasound Threshold Register.

US_THRES		0x50000042			0x08		
R/W	R/W	Reserved	R/W	Reserved	R/W	R/W	Reserved
TX_WF	SMP_CYC2	SMP_CYC1	SMP_CYC0	US_THRES3	US_THRES2	US_THRES1	US_THRES0
MSB							LSB

Bit7 **TX_WF**: Control shape of ultrasound TX waveform

0 = stepped

1 = square

Bit6-4 **SMP_CYC[2:0]**: Controls which of 1/8 period increments to generate the sampling pulse for open/short

detection

Bit3-0 **US_THRES[3:0]**: Adjust comparator threshold

Ultrasound Threshold Programming	
US_THRES	**Threshold (Pct Full Scale)**
0000	0.10 %
0001	0.15 %
0010	0.20 %
0011	0.24 %
0100	0.30 %
0101	0.34 % (default High, 0.9% AM mod)
0110	0.39 % (default Med High, 1.1% AM mod)
0111	0.59 %
1000	0.78 % (default Med Low, 2.2% AM mod)
1001	1.2 %
1010	1.6 % (default Low, 4.4% AM mod)
1011	2.3 %
1100	3.1 %
1101	4.7 %
1110	6.3 %
1111	9.4 %

US_FILT: Ultrasound Filter Register.

US_FILT		0x50000043			0x35		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
RTSEL1	RTSEL0	US_LPF	US_HPF4	US_HPF3	US_HPF2	US_HPF1	US_HPF0
MSB							LSB

Bit7-6 **RTSEL[1:0]**: Real Time Outputs

00 = STRBI

01 = I/Q Data

10 = open/short detect signal (or-ed together)

11 = US_INT

Bit5 **US_LPF**: Control low-pass filter pole frequency

0 = 100 Hz pole frequency

1 = 200 Hz pole frequency

Bit4-0 **US_HPF[3:0]**: Control high-pass pole frequencies

Ultrasound High Pass Filter Programming

US_HPF[4:0]	**Filter 1 Pole (Hz)**	**Filter 2 Pole (Hz)**
X00XX	11.5	see below
X01XX	14.8	
X10XX	17.8	
X11XX	21.3	
0XXXX	see above	<Disabled>
1XX00		11.5
1XX01		14.8
1XX10		17.8
1XX11		21.3

US_PLL: Ultrasound Divider Register.

US_PLL		0x50000044			0x4C		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
US_NX2	US_NX1	US_NX0	US_NF4	US_NF3	US_NF2	US_NF1	US_NF0
MSB							LSB

Bit7-5 **US_NX[2:0]**: Integer portion of feedback divider

Bit4-0 **US_NF[4:0]**: Fractional portion of feedback divider

Ultrasound Frequency Programming

US_NX[2:0]	f_{XO} / (4*f_{US})	Crystal Frequency Range (MHz)
000	20 + (US_NF / 32)	3.20 – 3.36
001	21 + (US_NF / 32)	3.36 – 3.52
010	22 + (US_NF / 32)	3.52 – 3.68
011	23 + (US_NF / 32)	3.68 – 3.84
100	24 + (US_NF / 32)	3.84 – 4.00
101	25 + (US_NF / 32)	4.00 – 4.16
110	26 + (US_NF / 32)	4.16 – 4.32
111		4.32 – 4.48

US_SIG: Ultrasound Signal Register.

US_SIG		0x50000046/7			0xXXXX		
R	R	R	R	R	R	R	R
US_SIG5	US_SIG4	US_SIG3	US_SIG2	US_SIG1	US_SIG0	0	0
US_SIG13	US_SIG12	US_SIG11	US_SIG10	US_SIG9	US_SIG8	US_SIG7	US_SIG6
MSB							LSB

Bit15-2 **US_SIG[13:0]**: Snapshot of RSSI

US_RXFAULT: Ultrasound RX Fault Register.

US_RXFAULT		0x5001800E			0xXX		
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	R	R
						USRxOpen	USRxShort
MSB							LSB

Bit1 **USRxOpen**: Ultrasound input open detect status
0 = No open detected
1 = Open detected

Bit0 **USRxShort**: Ultrasound input short detect status
0 = No short detected
1 = Short detected

US_ANATRIM: Ultrasound Analog Trim Register.

US_ANATRIM		0x50018013			0x24		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
USADC_T	USGAIN3	USGAIN2	USGAIN1	USGAIN0	USIBIAS2	USIBIAS1	USIBIAS0
MSB							LSB

Bit7 **USADC_T**: Ultrasound ADC Test Mode
0 = normal mode
1 = test mode

Bit6-3 **USGAIN[3:0]**: Control the third stage gain of ultrasound analog blocks
Gain = USGAIN*0.435 (example : USGAIN[3:0] = 1000, Gain = 8*0.435 = 10.8 dB

Bit2-0 **USIBIAS[2:0]**: Control ultrasound bias current

8.4 SHOCK SENSOR

The µSesame may be used with a simple shock sensor for low cost alarm applications. Pin 43, SHKIN may be used as an input to measure the response of an external shock sensor.

8.4.1 Shock Sensor Usage Description

The µSesame provides sourcing current to the load(shock sensor) and monitors the shock sensor output voltage.

8.4.2 Shock Sensor Related Registers

SHKSNS: Shock Sensor Register.

SHKSNS		0x50000045			0x04		
R/W	R/W	R	R/W	Reserved	R/W	R/W	R/W
SHKEN	SHKTRIG	SHKRX	SHKRDY		RSSISEL2	RSSISEL1	RSSISEL0
MSB							LSB

Bit7 **SHKEN**: Writing one turns on the shock sensor. Read returns one if shock sensor is enabled.

Bit6 **SHKTRIG**: Read returns one if triggered. Writing zero clears the triggered condition of the shock sensor.

Bit5 **SHKRX**: Reading returns sampled shock sensor receive level

Bit4 **SHKRDY:** Shock Sensor Ready

0 = Shock Sensor not ready

1 = Shock Sensor ready

Bit2-0 **RSSISEL[2:0]**: Select source for RSSI data

000 = I phase ADC

001 = Q phase ADC

010 = I_N phase ADC

011 = Q_N phase ADC

1xx = low pass filter

8.5 LIN

The µSesame device contains digital hardware, which implements a LIN 2.0 serial communications interface.

8.5.1 LIN Interface

µSesame implements a LIN (Specification 2.0) interface. Its main characteristics are:

- Configurable for support of both master or slave functionality
- Programmable data rate between 1 Kbit/s and 20 Kbit/s (for master)
- Automatic bit rate detection (for slave)

8.5.1.1 LIN Usage Description

µSesame implements a LIN (**L**ocal **I**nterconnect **N**etwork) peripheral. This implementation is compatible with the specification 2.0 and allows for the selection of both Master and Slave modes.

The definition of the protocol is beyond the scope of this datasheet and can be found in the following reference: (LIN Consortium)

http://www.lin-subbus.de/index.php?pid=7&lang=en&sid=e10bc3f7d4a021f4e8c083aa02e6e881

8.5.1.1.1 Data Length Control

The host controller has to define the length of the data field of the current LIN frame by adjusting the LINLENGTH register. If the data length bits[3:0] are loaded with the value “1111b” the length of the data field is decoded from Bit 5 and 4 of the identifier register (LINID) according to table below (e.g. compatibility to LIN specification 1.1). Otherwise the amount of data bytes can be written directly to the DATA_LENGTH[3:0] register (supported values are 0...8).

Table 8 - ID bits and number of bits

ID (Bit 5)	ID (Bit 4)	Number of Bytes in the data field
0	0	2
0	1	2
1	0	4
1	1	8

8.5.1.1.2 Timing Settings for "Wake Up Repeat Time" and "Bus Inactivity Time"

The time for repeating of wake up because of no reaction on the bus and to go to sleep because of inactivity on the bus can be optionally written by the application in registers LINTIMING:

Table 9 - LIN Inactivity Time

LINIT[1:0]	LIN Inactivity Time (sec.)
00	4
01	6
10	8
11	10

Table 10 - LIN Wake-Up Repeat Time

LINWPR1 [1:0]	LIN Wake-Up Repeat Time (msec.)
00	180
01	200
10	220
11	240

8.5.1.1.3 Bit Time Settings

The Bit rate of the LIN system has to be defined in the bit timing registers (LINBITDIV and LINBITMUL). The table below shows an overview of the registers.

Table 11 - LIN Timing Related Registers

Name	Description	Width(bits)
LINDIV[8:0]	Bit time divider integer value	9
LINMUL[4:0]	Bit time multiplier (master only)	5
LINDFRAC[2:0]	Bit time divider fraction value (master only)	3

The LIN bit rate *f*bit can be calculated from system clock *f*clk and bit timing registers according to the following equation.

$$Fbit = \frac{Fclk}{2 * (LINDIV + LINDFRAC/8) * (LINMUL + 1)}$$

The procedure of adjusting the bit timing registers is different between master and slave.

8.5.1.1.4 Bit Timing Register Adjustment of Master

The steps for adjusting the bit timing registers of the master are explained in the following.

1. Setting up the bit time multiplier depending on used LIN data rate *f*bit according to the following equation:

$$LINMUL = \frac{20KBits/sec}{Fbit} - 1$$

The value has to be rounded down to the next integer value.

1) Adjusting the bit time divider integer value depending on system clock, data rate and bit time multiplier according to the following equation:

$$LINDIV = \frac{Fclk}{2*(LINMUL+1)*(Fbit)}$$

The value has to be rounded down to the next integer value.

1.0 Adjusting the bit time divider fraction value depending on system clock, data rate, bit time multiplier and bit time divider integer according to the following equation:

$$LINDFRAC = \left(\frac{Fclk}{2*(LINMUL+1)*\ Fbit} - LINDIV\right)*\ 8$$

The value has to be rounded down to the next integer value.

The table below shows sample values of the bit timing registers for different LIN data rates.

Table 12 - LIN Timing Related Registers

System Clock	LIN data rate	LINMUL	LINDIV	LINDFRAC
3.58 MHz	19.2 Kbit/s	0	93	1
	9.6 Kbit/s	1	93	1
	1 Kbit/s	19	89	4

8.5.1.1.5 *Bit Timing Register Adjustment of Slave*

The steps for adjusting the bit timing registers of the LIN slave are explained in the following paragraphs.

Note: Register fields **LINMUL** and **LINDFRAC** do not exist in the slave. The LIN core slave synchronizes to any bit rate between 1 Kbit/s and 20 Kbit/s. Nevertheless, the bit timing registers have to be adjusted to adapt the LIN core to the used system clock frequency. Adjusting the bit time divider integer value depending on system clock according to the following equation:

$$LINDIV = \frac{Fclk}{40\mathrm{K}}$$

For a system clock of 3.58MHz LINDIV = 89.5 = 89. (Always rounded down)

Code Example: To set the LIN interface to operate in Master Mode with a baud rate of 9600baud from a system clock of 3.58MHz: (From Table above)

```
      LIN_SetMultDiv ( 1, 93);      //Multiplier = 1, divider = 93

      LIN_TimingControl ( 4, LIN_INACT4SEC, LIN_WPRPT200MS); // Fract. divider =
4

      LIN_MasterSlave (LIN_MASTER); //Lin Master
```

8.5.1.2 Control of the LIN Module

The first step before transmitting messages with the LIN core is setting up the bit rate of the LIN system. For that, the host controller has to load the bit time registers, which has been explained in the previous sections. After that, the message transfer can be started. Controlling LIN core master and LIN core slave by the application is explained in the following.

8.5.1.2.1 *Controlling the LIN Master*

The master is responsible for the schedule of the messages. It sends the header of each frame that contains SYNC BREAK FIELD, SYNC FIELD and IDENTIFIER FIELD. The steps for scheduling a message frame are explained in the following.

Code Example: Code segment to load and send a message as master. (8 bytes, enhanced checksum, transceiver active low, message = 0x123456789ABCDEF)

```
   LIN_SetID ( 0x10 );     //Set ID of message as 0x10
   LIN_Set_Chk_Tran_length (ENHANCED_CHECKSUM, LIN_TRANSCEIVER_ACTIVE_LOW, 
8);
   LIN_SetMode(LIN_TRANSMIT_ENABLE);
   LINDATA->BYTE[0] = 0x12;
   LINDATA->BYTE[1] = 0x34;
   LINDATA->BYTE[2] = 0x56;
   LINDATA->BYTE[3] = 0x78;
   LINDATA->BYTE[4] = 0x9A;
   LINDATA->BYTE[5] = 0xAB;
   LINDATA->BYTE[6] = 0xCD;
   LINDATA->BYTE[7] = 0xEF;
   LIN_MasterStartTransmission ();
```

1. The following steps have to be done by the application when an interrupt is requested.

- Check the **LIN_ERR bit (LINSTATUS**). Perform error handling and proceed to step d if bit ERROR is set, otherwise proceed to step b.

- Check the **LIN_WAKEUP** bit (**LINSTATUS)** - it is set if the master has received or transmitted a wakeup signal. Proceed with the step d if **LIN_WAKEUP** is set else proceed with step c.

- Check the **LIN_CMPLT** (**LINSTATUS**) - it is set if the transfer was successful. If **LIN_CMPLT** is set and the current frame was a receive operation load the received data from the data buffer.

- Set the **LIN_RST_INT** and **LIN_RST_ERROR** bits (**LINCONTROL**) register to reset the interrupt request and the error flags.

Code Example: Interrupt handler that implements above steps:

```
void LIN_Handler( void )         // IRQ 9 LIN
{
   if (LIN_ERROR & LIN_ReadStatus ())  //If an error was detected
   {
         if (LIN_ReadErrors () & LIN_PARITY_ERROR) //
         {
         // Process parity error
         }
         if (LIN_ReadErrors () & LIN_TIMEOUT_ERROR)      //
         {
         // Process timeout error
         }
         if (LIN_ReadErrors () & LIN_CHECKSUM_ERROR)     //
         {
         // Process checksum error
         }
         if (LIN_ReadErrors () & LIN_BIT_ERROR)    //
         {
         // Process bit error
         }
   }
   else //No error
   {
         if ( (LIN_WAKEUP & *LINSTATUS) == 0)//If didn't receive nor
                                             //transmit sleep wake-up
signal
         {
               if  ( *LINSTATUS & LIN_COMPLETE)    //If finished transmitting
               {
               //Process transmission of message completed
               }
         }
   }
   LIN_ResetInterrupt();   //Reset interrupt
   LIN_ResetError();       //Reset error
}
```

8.5.1.2.2 Controlling the LIN Slave

The LIN core slave detects the header of the message frame sent by the LIN master and synchronizes its internal bit time to the master bit time. An interrupt is requested after the reception of the IDENTIFIER FIELD, after the reception of a wakeup signal (if the slave is in sleep mode), when an error is detected or when the message transfer is completed.

The following steps have to be done by the application when an interrupt is requested.

- Check the **LIN_DATA_REQ** bit (**LINSTATUS)** (it is 1 when the IDENTIFIER FIELD has been received). Proceed with the following if **LIN_DATA_REQ** is set else proceed with step 2.

  - Load the identifier from the **LINID** register and process it.

  - Adjust the **LINTX** bit (1 - if the current frame is a transmit operation for the slave, 0 – if the current frame is a receive operation for the slave).

  - Load the data length in the **LINLENGTH** register (number of data bytes or value “1111b” if the data length should be decoded from the identifier) and set the checksum type (enhanced or classic).

  - Load the data to transmit into the data buffer (for transmit operation only).

  - Set the **LINACK** bit (**LINCONTROL**) register.

    Note 1: Steps a thru e have to be done during the IN-FRAME RESPONSE SPACE, if the current frame is a transmit operation for the slave; otherwise a timeout will be detected by the master. If the current frame is a receive operation for the slave, steps a thru e have to be finished until the reception of the first byte after the IDENTIFIER FIELD. Otherwise, the internal receive buffer of the slave core will be overwritten and a timeout error will be detected in the slave core.

    Note 2: If the application of the slave detects an unknown identifier (e.g. extended identifier = 0x3E) it has to write a 1 to bit **LIN_SLAVE_STOP** (**LINCONTROL**) instead of setting the **LINACK** bit (steps b thru e can be skipped). In that case the LIN core slave stops the processing of the LIN communication until the next SYNC BREAK is received.

1. Check the **LIN_ERR** (**LINSTATUS**). Perform error handling and proceed with step 6 if bit **LIN_ERR** is set else proceed with step 3.

   Note 3: Bit **LIN_TOUT_ERR** and bit **LIN_WAKEUP** are set if the slave has sent a wakeup signal but the master did not respond within 150 ms.

2. Check bit **LIN_IDL_TOUT** (**LINSTATUS**) is set and activate the sleep mode by setting bit **LINSLEEP** if it is.

3. Check bit **LIN_WAKEUP** - it is set if the slave has received a wakeup signal. If **LIN_WAKEUP** is set proceed with step 6 else proceed with step 5.

   Note 4: Bit **LIN_CMPLT** is not changed when a wake-up signal is transmitted or received. Therefore, bit **LIN_WAKEUP** has to be checked before bit **LIN_CMPLT**.

3. Check **LIN_CMPLT bit** in the **LINSTATUS** register (it is set if the transmission was successful). If **LIN_CMPLT** is set and the current frame was a receive operation for the slave, load the received data bytes from the data buffer.

4. Set the bits **LIN_RST_INT** and **LIN_RST_ERR** in the control register to reset the interrupt request and the error flags.

Code Example: Processing of an interrupt (slave mode) following above rules, slave transmitting data back to master :(rule between [])

```
void LIN_Handler( void )          // IRQ 9 LIN
{
   if (LIN_ReadStatus() == LIN_DATA_REQ )    //[1]
   {
         if (*LINID == 0x3E)     //Check for Extended ID and process if so 
[1.a]
         {
               //Process extended ID
               LIN_SlaveStop();  //[1.e]
         }
         else
         {
               if (*LINID == ID_SEND_DATA )  //If some ID of interest
               {
                     LIN_SetMode(LIN_TRANSMIT_ENABLE);   //[1.b]
                           LIN_Set_Chk_Tran_length (ENHANCED_CHECKSUM,
                                 LIN_TRANSCEIVER_ACTIVE_LOW, 3);//[1.c]
                                       LINDATA->BYTE[0] = 0x12;//[1.d]
                                             LINDATA->BYTE[1] = 0x34;
                                                   LINDATA->BYTE[2] = 0x56;
                     LIN_SlaveDataAck( LIN_SLAVE_DATA_ACK );//[1.e]

               }
```

```
            }
      }
      else
      {
            if (LIN_ERROR & LIN_ReadStatus ())  //If an error was detected
                                                      //[2/note3]
            {
                  if (LIN_ReadErrors () & LIN_PARITY_ERROR) //
                  {
                  // Process parity error
                  }
                  if (LIN_ReadErrors () & LIN_TIMEOUT_ERROR)      //
                  {
                  // Process timeout error
                  }
                  if (LIN_ReadErrors () & LIN_CHECKSUM_ERROR)     //
                  {
                  // Process chekcsum error
                  }
                  if (LIN_ReadErrors () & LIN_BIT_ERROR)    //
                  {
                  // Process bit error
                  }

                  }
            }
            if (LIN_ReadStatus() == LIN_IDL_TOUT )    //[1]
            {
                  LIN_SetSleep(LIN_SLEEP_ENABLE);//[3]

            }
            if (LIN_ReadStatus() == LIN_WAKEUP) //[4]
            {
            //Process wake-up
            }
            else if (LIN_ReadStatus() == LIN_CMPLT)   //[5]
            {
                  //If transmission completed process it
            }
      LIN_ResetInterrupt();   //Reset interrupt [6]
      LIN_ResetError();       //Reset error
   }
```

8.5.1.2.3 Sleep Mode and Wakeup

To reduce the systems power consumption the LIN Protocol Specification defines a Sleep Mode. The message used to broadcast a Sleep Mode request has to be started by the host controller of the LIN core master in the same way as a normal transmit message. The host controller of the LIN core slave has to decode the Sleep Mode Frame from Identifier and data bytes. After that, it has to put the LIN slave node into the Sleep Mode by setting bit **LINSLEEP** in the control register. If bit **LINSLEEP** in the control register of the LIN core slave is not set and there is no bus activity for 4 s to 10 s (specified bus idle timeout) bit **LIN_IDL_TOUT** is set and an interrupt request is generated. After that application may assume that the LIN bus is in Sleep Mode and set bit **LINSLEEP** in the **LINCONTROL** register of the LIN core slave. The bus inactivity time which should be defined as bus idle timeout for the slave can optionally set to values 4s, 6s, 8s or 10s as possible accordingly with specification 2.0.

Sending a Wakeup signal with the master or any slave node terminates the Sleep Mode of the LIN bus. To send a Wakeup signal, the host controller of the LIN core has to set the bit **LIN_WAKEUP** in the **LINSTATUS** register. After successful transmission of the wakeup signal with the LIN core master the **LIN_WAKEUP** bit in the **LINSTATUS** register of the sending LIN core master is set and an interrupt request is generated. The LIN core slave does not generate an interrupt request after successful transmission of the Wakeup signal but it generates an interrupt request if the master does not respond to the Wakeup signal within 150 msec. to 250 msec. This value can be set optionally to 180ms, 200ms, 220ms or 240ms as it is possible accordingly with specification 2.0. In that case, bit **LIN_ERR** and bit **LIN_TOUT_ERR** are set. The host controller has to decide whether to transmit another Wakeup signal or not.

All LIN cores that detect a wakeup signal will set the bit **LIN_WAKEUP** and generate an interrupt request to their host controller. The inverted bit **LINSLEEP** is connected to the output **LIN_TR_EN**. Bit **LINSLEEP** is automatically reset and **LIN_TR_EN** (whose polarity can be flipped by setting/clearing **LINTRAN**) is set to high when the LIN core detects a wakeup signal. Output **LIN_TR_EN** may be used for connecting the enable signal of the LIN transceiver. It depends on the transceiver type whether this is possible or not.

8.5.1.2.4 Error Detection and Handling

The LIN core generates an interrupt request and stops the processing of the current frame if it detects an error. The application has to check the type of error by processing the **LINERROR** register. After that, it has to reset the **LINERROR** register and the **LIN_ERR** bit in status register by writing a 1 to bit **LIN_RST_ERR** in control register. Starting a new message with the LIN core master or sending a Wakeup signal with master or slave is possible only if bit LIN_ERR in **LINSTATUS** register is 0.

8.5.1.3 LIN Registers

The following registers are available:

LIN_DATAn: LIN data registers. (n = 0, 1, …, 7)

LIN_DATAn		0x50000030/1/2/3/4/5/6/7			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
LDT7	LDT6	LDT5	LDT4	LDT3	LDT2	LDT1	LDT0
MSB							LSB
Bit7-0 **LDT7- LDT0:** LIN data bits							

LINCTRL: LIN control register.

LINCTRL		0x50000038			0x00		
W	R/W	R/W	R/W	W	W	R/W	R/W
LINSTOP	LINSLEEP	LINTX	LINACK	LIN_RST_INT	LIN_RST_ERR	LIN_WKUP_REQ	LIN_START_REQ
MSB							LSB

Bit7 **LINSTOP:** LIN Stop command (slave only): The host has to write a '1' to this bit if it handles a data request interrupt and cannot make use of the frame content with the received identifier (e.g. extended identifiers). For that case the LIN slave stops the processing of the LIN communication until the next SYNC BREAK is detected.

0 = No action

1 = STOP

Bit6 **LINSLEEP:** LIN Sleep command: The bit is used by the LIN core to determine whether the LIN bus is in Sleep Mode or not. The application has to set the bit after sending or receiving a Sleep Mode frame or if a bus idle timeout interrupt is requested. The bit will be reset by the LIN core, when a wakeup signal is detected.

0 = LIN interface is not in sleep mode

1 = LIN interface is in sleep mode

Bit5 **LINTX:** LIN transmit command: The bit determines whether the current frame is a transmit frame or a receive frame for the LIN node. It has to be set by the application.

0 = LIN interface is receiving

1 = LIN interface is transmitting

Bit4 **LINACK:** LIN data acknowledge (slave only): The bit has to be set by the application after handling a data request interrupt (compare bit LIN_DT_REQ in LINSTATUS register). The bit will be reset by the LIN core.

0 = Acknowledge not requested or already reset by core

1 = LIN interface acknowledge request

Bit3 **LIN_RST_INT:** LIN reset interrupt: The application has to write a '1' to this bit to reset the LIN_INT_REQ bit in the LINSTATUS register.

0 = No Interrupt reset request

1 = Reset of interrupt request

Bit2 **LIN_RST_ERR:** LIN reset error: The application has to write a '1' to this bit

to reset the error bits in status register and error register.

0 = No errors reset request

1 = Errors reset request

Bit1 **LIN_WKUP_REQ:** LIN Wake-Up request: The bit has to be set by the application to

terminate the Sleep Mode of the LIN bus by sending a Wakeup signal. The bit will be reset by the LIN core.

0 = No wake-up request

1 = Wake-up request

Bit0 **LIN_START_REQ:** LIN start request (master only):

The bit has to be set by the application to start the LIN transmission after loading Identifier, data length and data buffer. The LIN core will reset the bit after the transmission is finished or an error is occurred.

0 = No action

1 = Start Transmission

LINSTATUS: LIN status register.

LINSTATUS		0x50000039			0x00		
W	R/W	R/W	R/W	W	W	R/W	R/W
LINACTIVE	LIN_IDL_TOUT	LINABRT	LIN_DT_ REQ	LIN_INT_ REQ	LIN_ERR	LIN_WAKEUP	LIN_CMPLT
MSB							LSB

Bit7 **LINACTIVE:** LIN active: The bit indicates whether the LIN bus is active or not.

1 = Transmission on the LIN bus is active

0 = No LIN bus activity

Note: For the LIN slave, this bit is set after the detection of a correct SYNC BREAK / SYNC FIELD sequence and it is reset at the end of the transmission or if the processing of the current frame is stopped by the host controller.

Bit6 **LIN_IDL_TOUT:** LIN idle timeout (slave only):

This bit is set by the LIN core if bit LINSLEEP in control register is not set and no bus activity is detected for 4 s. In addition, an interrupt request is generated in that case. After that, the application may assume that the LIN bus is in sleep mode and it has to set bit LINSLEEP in the LINCTRL register.

0 = NO sleep mode condition detected

1 = Sleep mode condition detected

Bit5 **LINABRT:** LIN aborted (slave only):

This bit is set by the LIN core slave if a transmission is aborted after the beginning of the data field due to a timeout or bit error (caused e. g. by a new sync break after missing data bytes). The bit is also set if the processing of the current frame has been stopped by writing a ‘1’ to bit STOP in control register. The bit is cleared by the LIN core after receiving a correct SYNC BREAK / SYNC FIELD sequence.

0 = LIN transmission NOT aborted

1 = LIN transmission aborted

Bit4 **LIN_DT_REQ:** LIN data request (slave only):

The LIN core slave sets the bit after receiving the Identifier and sends an interrupt to the host controller. The application has to decode the Identifier to decide whether the current frame is a transmit or a receive operation. It has to adjust the LINTX bit in the control register and to load the data length. For transmit operations it has to load the data buffer too. After that the host controller has to set the bit LINACK in the control register.

0 = No data requested

1 = Data requested

Bit3	**LIN_INT_REQ:** LIN interrupt request: The LIN core sets the bit when it sends an interrupt. The bit has to be reset by the application by setting the bit LIN_RST_INT in the control register. 0 = No Interrupt request 1 = Interrupt requested
Bit2	**LIN_ERR:** LIN error: The LIN core sets the bit if an error has been detected (compare error register). The bit has to be reset by the host controller by setting the bit LIN_RST_ERR in the control register. 0 = No errors 1 = Errors detected
Bit1	**LIN_WAKEUP:** LIN Wake-up: The bit is set when the LIN core is transmitting a Wake-up signal or when the LIN core has received a Wakeup signal. 0 = No wake-up 1 = Wake-up signal
Bit0	**LIN_CMPLT:** LIN complete: The LIN core will set the bit after a transmission has been successfully finished and it will reset it at the start of a transmission. 0 = Transmission started 1 = Last transmission succeeded

LINERROR: LIN error register.

LINERROR		0x5000003A			0x00		
Reserved	Reserved	Reserved	Reserved	R	R	R	R
-	-	-	-	LIN_PARITY_ERR	LIN_TOUT_ERR	LIN_CHK_ERR	LIN_BIT_ERR
MSB							LSB

Bit3	**LIN_PARITY_ERR:** LIN parity error: Identifier parity error. (Slave only) 0 = No parity error identified 1 = Parity error identified
Bit2	**LIN_TOUT_ERR:** LIN timeout error: There are several reasons that can cause a timeout error: The master detects a timeout error if it is expecting data from the bus but no slave does respond. If the slave responds to late and the frame is not finished within the maximum frame length TFRAME_MAX a timeout error will be detected too. The slave detects a timeout error if it is requesting a data acknowledge to the host controller (for selecting receive or transmit, data length and loading data) and the host controller does not set the bit DATA_ACK or bit STOP in control register until the end of the reception of the first byte after the identifier. The slave detects a timeout error if it has transmitted a wakeup signal and it detects no sync field (from the master) within 150 ms. 0 = No timeout error

1 = Timeout error detected

Note: The slave does not perform an exact check of the frame length TFRAME_MAX but a timeout is detected after 200 bit times if the slave is in receive mode and there are missing data fields or a missing ID field from the master.

Bit1 **LIN_CHK_ERR:** LIN checksum error:

0 = No checksum error

1 = Checksum error

Bit0 **LIN_BIT_ERR:** LIN bit error: The bit transmitted does not match the one read.

0 = No bit error

1 = Bit error

LINLENGTH: LIN data length, checksum mode and transceiver polarity.

LINSTATUS		0x5000003B			0x00		
R/W	R/W	Reserved	Reserved	R/W	R/W	R/W	R/W
LINCHK	LINTRAN	-	-	LINDLEN3	LINDLEN2	LINDLEN1	LINDLEN0
MSB							LSB

Bit7 **LINCHK:** LIN checksum:

0 = Classic Checksum

1 = Enhanced Checksum

Bit6 LINTRAN: LIN transceiver enable polarity:

0 = Transceiver enable signal active high

1 = Transceiver enable signal active low

Bit3-0 **LINDLEN:** LIN data length:
The application has to define the length of the data field of the current LIN frame by adjusting the LINDLEN[3:0] bits. If the bits are loaded with the value "1111b" the length of the data field is decoded from Bit 5 and 4 of the identifier register "id" according to the table below (e.g. compatibility to LIN specification 1.1). Otherwise the amount of data bytes can be written directly to the register (supported values are 0...8).

Table 13- LIN data length (when the length bits have the value "1111b")

ID Bit 5	ID Bit 4	Number of Bytes
0	0	2
0	1	2
1	0	4
1	1	8

LINBITDIV: LIN Bit Divider

LINBITDIV		0x5000003C			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
LINDIV7	LINDIV6	LINDIV5	LINDIV4	LINDIV3	LINDIV2	LINDIV1	LINDIV0
MSB							LSB
Bit7-0 **LINDIV[7:0]:** LIN bit divider.							

LINBITMUL: LIN Bit Divider

LINBITMUL		0x5000003D			0x7F		
Reserved	Reserved	R/W	R/W	R/W	R/W	R/W	R/W
-	-	LINMUL4	LINMUL3	LINMUL2	LINMUL1	LINMUL0	LINDIV8
MSB							LSB
Bit5-1 **LINMUL[4:0]:** LIN bit multiplier: Bit0 **LINDIV[8]:** LIN divider bit8							

LINID: LIN ID.

LINID		0x5000003E			0x00		
Reserved	Reserved	R/W	R/W	R/W	R/W	R/W	R/W
-	-	LINID5	LINID4	LINID3	LINID2	LINID1	LINID0
MSB							LSB
Bit5-0 **LINID[5:0]:** LIN id.							

LINTIMING: LIN timing.

LINTIMING		0x5000003F			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
LINMS	LINDFRAC2	LINDFRAC1	LINDFRAC0	LINIT1	LINIT0	LINWPR1	LINWPR0
MSB							LSB

Bit7	**LINMS:** LIN Master/Slave selection: 0 = Slave 1 = Master
Bit6-4	**LINDFRAC[2:0]:** LIN fractional divider.
Bit3-2	**LINIT[1:0]:** LIN inactive time.
Bit1-0	**LINWPR[1:0]:** LIN wake-up repeat time.

8.6 UART

µSesame implements a UART (**U**niversal **A**synchronous **R**eceiver **T**ransmitter) module. The main characteristics are defined below:

- Four bytes deep reception FIFO (**F**irst **I**n - **F**irst **O**ut) with "watermark" selectable to one and three bytes
- Four bytes deep transmission FIFO (**F**irst **I**n - **F**irst **O**ut)
- Interrupt available for transmission, reception and error events
- Reception timeout timer
- Programmable break reception and transmission
- Programmable parity with "sticky" parity option
- Selectable number of bits from 5 to 8
- Selectable number of stop-bits: 1, 1 ½, 2
- Programmable loop-back
- Swappable TXD and RXD (PD[4] and PD[5])
- Transmitter Polarity selection

8.6.1 UART Operation

The UART protocol requires two wires (UTXD and URXD). Port D[5:4] are configured as UTXD and URXD when the MDUART bit is set in PORTEOE register. In order to use it the following steps must be followed:

1. Select the pins position (normal or swapped) of the interface and also its polarity. The normal position (not swapped) is TX=PD[5] and RX = PD[4].

Code Example: Selecting UART with normal polarity and swapped: (UART pins swapped: TX=PD[4] and RX = PD[5])

```
UART_Setup(UARTSWAP_EN,  UARTPOL_NORMAL);
```

2. Define the following parameters:

   a. Loop back: Used mainly in tests, internally connects the output to the input.
   b. Break enable: Puts the output down while asserted, rising the output once de-asserted.
   c. Sticky parity: Forces the parity to stay stable in one direction.
   d. Even/Odd parity selection and enable: Selection and enable of Even or Odd parity bits.
   e. Number of stop bits: Selection of 1 (default), 1½ (5-bit communications only) or 2 stop bits.
   f. Data size in bits (5,6,7,8): Selection of the number of bits used in the communication

Code example: Setting the above parameters: (no loop-back, no break signal, no sticky parity, no parity, one stop-bit, 8-bit communications)

```
UART_Ctrl(UART_LBDIS, UART_BREAKDIS, UART_STPDIS,UART_ODDEN, UART_PARDIS,
  UART_10STOP, UART_8BITS);
```

3. Define the baud rate. The baud rate is calculated as follows: (UARTDIV is a 16-bit register)

$$Baud = \frac{Fclk}{16 * (UARTDIV + 1)}$$

Assuming a 3.579545MHz system clock the following table provides some register values, baud rates and related errors:

Table 14 - UART baud rates, divider values and errors

Baud	UARTDIV	Real Baud	Error (%)
300	745	299.9	0.04
600	372	599.8	0.04
1200	185	1203	0.23
2400	92	2406	0.23
4800	46	4760	0.83
9600	22	9727	1.3
19200	11	18644	2.9
38400	5	37287	2.9
57600	3	55930	2.9
115200	1	111861	2.9

Code Example: Setting the UART to operate at 9600 baud:

```
      UART_BaudRateDivider(22);
```

4. Enable the UART and its interrupt: The UART may generate an interrupt for events related to:

1. Transmission completed.
2. Reception: Timeout of ~40 bit-times without reception, and data received (one or three bytes received, programmable).
3. Errors detected: Framing error, parity error, and overrun error.
4. Break signal detected (received).

Code example: Enabling the UART with no timeout interrupts, errors, transmission and reception interrupts with interrupts once 3 bytes are received in the FIFO.

```
//Timeout interrupt disabled, Error enabled, TX enabled, RX enabled

      UART_Interrupt_Control(UART_TOUTDIS,UART_ERREN,UART_TXEN,UART_RXEN);

//UART enabled, receive interrupt after 3 bytes received, reset RX FIFO and TX
//FIFO.

      UART_Ctrl1( UARTEN, UART_INT3, uint8_t tfr, uint8_t rfr );

//Enabling interrupt from UART at the microcontroller

      NVIC_EnableIRQ(UART_IRQn);         //Enable UART interrupt
```

5. Processing of the UART interrupt:

```
void UART_Handler( void )           // IRQ 8 UART
{
      switch(UART_Interrupt_Status())
      {
            case  UART_ERROR:
                  //Process reception error here
                  SWITCH(UART_CheckError())
                  {
                        case UART_FRAMING_ERROR: // Process this error here
                              break;
                        case UART_PARITY_ERROR: // Process this error here
                              break;
                        case UART_OVERRUN_ERROR: // Process this error here
                              break;
                        case 0: // No error, exit
                        default:
                              break;
                  }
```

```
                        break;
            case    UART_RXRDY:
                        //Process data received
                        mydata = *UARTDATA;     //Read data received by uart into
mydata
                        break;
            case    UART_TIMEOUT:
                        //Process reception timeout
                        break;
            case    UART_TXDONE:
                        //Process transmission complete
                        break;
            case    UART_NOINT: //No int.
            default:
            //If no interrupt asserted or something else happened nothing to do
                              break;
      }
}
```

8.6.2 UART Registers

The following registers are defined in µSesame:

UARTDATA: UART data.

UARTDATA		0x50000010			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
UARTD7	UARTD6	UARTD5	UARTD4	UARTD3	UARTD2	UARTD1	UARTD0
MSB							LSB
Bit7-0 **UARTD [7:0]:** UART data, both received and to be transmitted.							

UARTICR: UART Interrupt Control Register.

UARTICR		0x50000011			0x00		
R	R	R	R	R/W	R/W	R/W	R/W
UISTTS3	UISTTS2	UISTTS1	UISTTS0	UTOUTIEN	URXERREN	UTXIEN	URXIEN
MSB							LSB

Bit7-4 **UISTTS [3:0]:** UART Interrupt status:

0001 = No Interrupt asserted

0010 = Transmission completed

0100 = Data received

0110 = Reception error

1100 = Reception timeout (~40 bit-time)

Bit3 **UTOUTIEN**: UART time-out interrupt enable bit:

0 = Time-out interrupt disabled

1 = Time-out interrupt enabled

Bit2 **URXERREN**: UART reception error interrupt enable bit:

0 = Reception error interrupt disabled

1 = Reception error interrupt enabled

Bit1 **UTXIEN**: UART transmission completed interrupt enable bit:

0 = Transmission completed interrupt disabled

1 = Transmission completed interrupt enabled

Bit0 **URXIEN**: UART reception interrupt enable bit:

0 = Reception interrupt disabled

1 = Reception interrupt enabled

UARTCTRL: UART Control Register.

UARTCTRL		0x50000012			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ULOOPEN	UBREAKEN	USTICKEN	UPARITY	UPAREN	USTOP	USTOP	USIZE
MSB							LSB

Bit7 **ULOOPEN**: UART loop back enable:
0 = UART loop back disabled
1 = UART loop back enabled

Bit6 **UBREAKEN:** UART break enable:
0 = UART break disabled
1 = UART break enabled

Bit5 **USTICKEN**: UART sticky parity enable bit:
0 = Sticky parity disabled
1 = Sticky parity enabled

Bit4 **UPARITY**: UART parity bit:
0 = Odd parity
1 = Even parity

Bit3 **UPAREN**: UART parity enable bit:
0 = Parity disabled
1 = Parity enabled

Bit2 **USTOP**: UART stop bit:
0 = One stop bit
1 = If a 5-bit transmission it selects1.5 stop bits, otherwise 2 stop bits (6, 7 and 8 bits)

Bit1-0 **USIZE**: UART transmission size:
00 = 5-bit data
01 = 6-bit data
10 = 7-bit data
11 = 8-bit data

UARTCTRL1: UART Control Register1.

UARTCTRL1		0x50000013			0x00		
Reserved	Reserved	Reserved	Reserved	R/W	R/W	R/W	R/W
-	-	-	-	UARTEN	URXFS	UTXFRST	URXFRST
MSB							LSB

Bit3 **UARTEN**: UART enable:
0 = UART disabled
1 = UART enabled

Bit2 **URXFS:** UART RX FIFO interrupt level:
0 = UART interrupts after one byte received
1 = UART interrupts after three bytes received

Bit1 **UTXFRST**: UART transmission FIFO reset bit:
0 = TX FIFO not reset
1 = TX FIFO reset

Bit0 **URXFRST**: UART reception FIFO reset bit:
0 = RX FIFO not reset
1 = RX FIFO reset

UARTDIV: UART Baud rate divider. (16-bit)

UARTDIV		0x50000016			0x0000		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
UDIV7	UDIV6	UDIV5	UDIV4	UDIV3	UDIV2	UDIV1	UDIV0
UDIV15	UDIV14	UDIV13	UDIV12	UDIV11	UDIV10	UDIV9	UDIV8
MSB							LSB

Bit15-0 **UDIV [15:0]**: UART clock divider

UARTSTATUS: UART Control Register1.

UARTSTATUS		0x50000014			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
UERR	UTXEMPTY	UTXFFEMPTY	UBREAKINT	UFRMERR	UPRTYERR	UOVRUNERR	UDTRDY
MSB							LSB

Bit7	**UERR**: UART error:
	0 = No error
	1 = Error in UART
Bit6	**UTXEMPTY:** UART transmission empty:
	0 = UART transmitter not empty
	1 = UART transmitter empty
Bit5	**UTXFFEMPTY**: UART transmission FIFO empty:
	0 = TX FIFO not empty
	1 = TX FIFO empty
Bit4	**UBREAKINT**: UART break interrupt:
	0 = No break interrupt
	1 = Break interrupt
Bit3	**UFRMERR**: UART framing error:
	0 = UART no framing error
	1 = UART framing error
Bit2	**UPRTYERR**: UART parity error:
	0 = No parity error
	1 = Parity error
Bit1	**UOVRUNERR**: UART overrun error:
	0 = No overrun error
	1 = Overrun error
Bit0	**UDTRDY**: UART data ready:
	0 = No data ready (reception)
	1 = Data ready (reception)

8.7 SPI INTERFACE

The Serial Peripheral Interface (SPI) is a synchronous full-duplex serial interface. It communicates in master/slave mode where the master initiates the data transfer. In µSesame, the SPI is implemented as a master. The module is compatible with Motorola SPI interface. There are many references available, and one of them is:

http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

µSesame SPI module's main features are defined below:

- Compatible with Motorola SPI interface
- Four bytes deep reception FIFO
- Four bytes deep transmission FIFO
- Interrupt upon events related to transmission, reception and error:
  - Write Collision
  - Transmission FIFO full and empty
  - Reception FIFO full and empty

The SPI protocol requires four wires (SCK, MISO, MOSI, and SS). Port D[3:0] are configured as the SPI bus when the MDSPI bit is set in PORTEOE register. The following table describes how each pin is connected:

Table 15 : SPI interface signals

Name	Pin Number	Pin Name	Comments
MISO	18	PD0	3.3V
MOSI	19	PD1	
SCLK	20	PD2	
SS (SPI-SSEL)	21	PD3	

8.7.1 SPI Functionality

Only the master mode is implemented in µSesame. µSesame configures the clock frequency and generates the serial clock (SCK) for the interface. The data transfer is synchronous through SCK. The SPI is a full-duplex system; data is transmitted and received simultaneously. µSesame sends the information to the slave device through MOSI line and receives the data through MISO line. CPOL and CPHA bits determine when to sample the data.

When CPOL=0, the base value of clock is logic ‘0’. In this case, if CPHA=0, data is captured on the rising edge of SCK and data is propagated on the falling edge of SCK. For CPHA=1, data is captured on the falling edge of SCK and data is propagated on the rising edge of SCK.

If CPOL=1, the base value of clock is logic ‘1’. In this case, if CPHA=0, data is captured on the falling edge of SCK and data is propagated on the rising edge of SCK. For CPHA=1, data is captured on the rising edge of SCK and data is propagated on the falling edge of SCK.

The timing diagram is shown below.

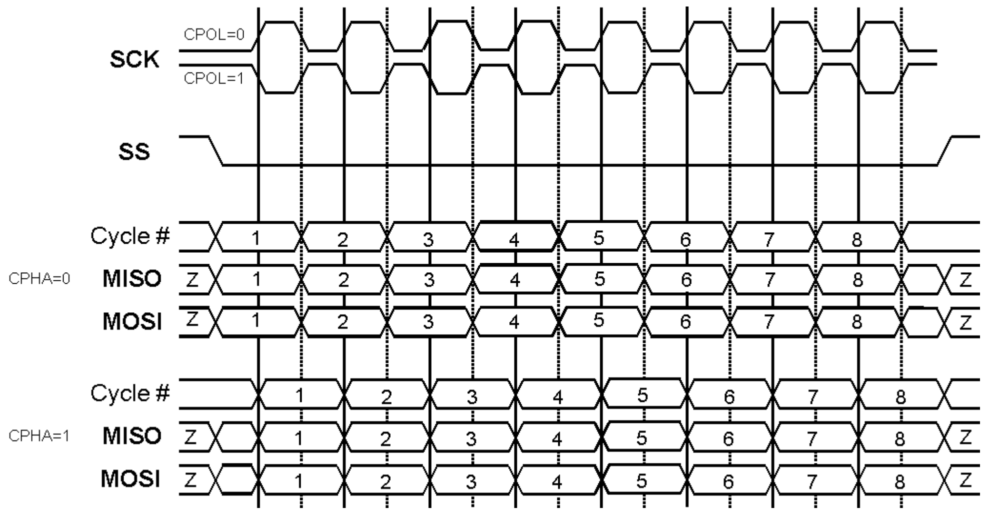


Figure 8– SPI Timing Diagram

After a desired configuration is set through configuration registers, a transfer is initiated by writing to the Serial Peripheral Data Register (SPDR). The data is entered to 4-deep FIFO before it is actually transmitted. When the data is transmitted, the slave also transmits the data simultaneously for µSesame to receive. The received data is stored in a separate 4-deep FIFO. The data is accessed by reading SPDR register.

To operate it properly the following steps must be performed:

1. Configure and enable the SPI: Select if the interrupt is enabled, polarity, phase and the clock divider:

   Code Example: Enabling the SPI interface with interrupt enabled, clock polarity negative (base value = 0), phase1 (data captured on the clock's falling edge and propagated on the rising edge), and divider = 2.

   ```
   SPI_Config(SPI_INT_EN,SPI_EN,SPI_CPOL_NEG,SPI_PHASE1,2);
   ```

2. Enable the interrupt (microcontroller) if required:

   Code Example:

```
//Enabling interrupt from SPI at the microcontroller

NVIC_EnableIRQ(SPI_IRQn);         //Enable SPI interrupt
```

3. Process Interrupt if required: Detect reason for the interrupt (error, transmission or reception related and act accordingly):

   Code Example: Processing SPI interrupt:

```
void SPI_Handler( void )            // IRQ A SPI
{
      uint8_t status;
      if ( (status = SPI_ReadStatus() ) & SPI_INT_FLAG )
      {
            if (status == SPI_WCOL)
            {
            //Process write collision (data is written to the SPI data
            //register while a SPI data transfer is in progress)

            }
            if (status == SPI_TX_FIFO_FULL)
            {
            //Process Transmission FIFO full
            }
            if (status == SPI_TX_FIFO_EMPTY)
            {
            //Process end of transmission of data previously
            //in transmission FIFO
                  *SPIDATA = outdata[j++];
                  *SPIDATA = outdata[j++];
                  *SPIDATA = outdata[j++];
            }
            if (status == SPI_RX_FIFO_FULL)
            {
            //Process reception FIFO full, normally by reading
            //all bytes of data
                  for ( i = SPI_ReadRxFifoSize(); i > 0; i--)
                        mydata[i++] = *SPIDATA;
            //OTher processing here
            }
            if (status == SPI_RX_FIFO_EMPTY)
            {
            //Process when no more information is available (received)
            }
      }
}
```

8.7.2 SPI Registers

The following registers are defined in the SPI interface:

SPCR: SPI Control Register.

SPCR		0x5000001C			0x10		
R/W	Reserved	Reserved	R	R/W	R/W	R/W	R/W
SINTE	-	-	MSTR	CPOL	SPH	SCKSTD1	SCKSTD0
MSB							LSB

Bit7 **SINTE**: SPI Interrupt enable

0 = Interrupt is disabled

1 = Interrupt is enabled

Bit4 **MSTR:** Master Mode Select Bit

SPI is always in master mode in μSesame, and therefore, it is always set to logic ‘1’.

Bit3 **CPOL** SPI clock polarity

0 = The base value of the clock is zero

1 = The base value of the clock is one

Bit2 **CPHA**: SPI clock phase

0 = data is captured on clock transition from base and data is propagated on the clock transition to base

1 = data is captured on clock transition to base and data is propagated on the clock transition from base

Bit1-0 **SCKSTD[1:0]**: SPI standard clock divider selection

Please refer to SPER register for system clock

SPSR: SPI Status Register.

SPSR		0x5000001D			0x00		
R/W	R/W	Reserved	Reserved	R/W	R/W	R/W	R/W
SINTF	SWCOL	-	-	STXFF	STXFE	SRXFF	SRXFE
MSB							LSB

Bit7	**SINTF**: SPI interrupt flag 0 = Interrupt not asserted 1 = Interrupt asserted
Bit6	**SWCOL:** SPI write collision is set when the SPDR register is written to while the transmit FIFO is full 0 = No collision 1 = collision
Bit3	**STXFF**: SPI transmit FIFO full 0 = transmit FIFO not full 1 = transmit FIFO full
Bit2	**STXFE**: SPI transmit FIFO empty 0 = transmit FIFO not empty 1 = transmit FIFO empty
Bit1	**SRXFF**: SPI reception FIFO full 0 = reception FIFO not full 1 = reception FIFO full
Bit0	**SRXFE**: SPI reception FIFO empty 0 = reception FIFO not empty 1 = reception FIFO empty

SPDR: SPI Data Register.

SPDR		0x5000001E			0xXX		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
SPID7	SPID6	SPID5	SPID4	SPID3	SPID2	SPID1	SPID0
MSB							LSB
Bit7-0 **SPID[7:0]**: SPI data, used in both transmission and reception							

SPER: SPI Extended Register

SPER		0x5000001F			0x00		
R/W	R/W	Reserved	Reserved	Reserved	R/W	Reserved	Reserved
SICNT1	SINCT0	-	-	-	SPE	SCKEXT1	SCKEXT0
MSB							LSB

Bit7-6 **SICNT[1:0]**: SPI Interrupt Counter Bits

00 = SINTF is set after every completed transfer

01 = SINTF is set after every two completed transfers

10 = SINTF is set after every three completed transfers

11 = SINTF is set after every four completed transfers

Bit2 **SPE**: SPI Enable

0 = SPI module is disabled

1 = SPI module is enabled

Bit1-0 **SCKEXT[1:0]**: SPI extended clock divider

SCKSTD	**SCKEXT**	**Result Clock Divider**
00	00	= System Clock/2
01	00	= System Clock/4
10	00	= System Clock/16
11	00	= System Clock/32
00	01	= System Clock/8
01	01	= System Clock/64
10	01	= System Clock/128
11	01	= System Clock/256
00	10	= System Clock/512
01	10	= System Clock/1024
10	10	= System Clock/2048
11	10	= System Clock/4096
xx	11	= Reserved

8.8 I^{2}C INTERFACE

µSesame implements an I^{2}C interface. Its main characteristics are:

- Support for multi-master mode
- General call support
- 10-bit address
- Address masking

The I^{2}C interface is a well-known interface and many references that describe its behavior are available. As an example:

http://en.wikipedia.org/wiki/I%C2%B2C

8.8.1 I2C Functionality

µSesame's I^{2}C must be configured for proper use.

8.8.1.1 Slave Mode

Slave Mode:

1. Select the peripheral as slave. Code Example:
   ```
   I2C_Mode( I2C_SLAVE );
   ```
2. Select the address size. Code Example selecting 7-bit address size:
   ```
   I2C_Slave_Sets_Address_Size ( 7_BIT_ADDRESS );
   ```
3. Load the address. Code example loading 0x7A as address:
   ```
   I2C_WriteAddress(0x7A);
   ```
4. Select if general call is to be supported. Code example disabling general call:
   ```
   I2C_Slave_Enable_General_Call ( GC_OFF );
   ```
5. Select address masking if required. If required the peripheral provides a 5-bit address mask for the lower 5 address bits. Each bit masks the corresponding bit in address comparison when set. For example, as an I^{2}C Slave in 7-bit address mode is using 0x03 as mask and 0x36 as address Then the I^{2}C will answer to all messages with addresses 0x34, 0x35, 0x36 or 0x37. Code example selecting 0x03 as mask:
   ```
   I2C_Slave_Address_Mask ( 0x03 );
   ```
6. Enable the interface and its interrupts if used. Code example:
   ```
   I2C_Enable( I2C_ON );//Enabling the interface
   NVIC_EnableIRQ( I2C_IRQn );                //Enable I2C communications
interrupt
   NVIC_EnableIRQ( I2C_Collision_IRQn ); //Enable I2C collision interrupt
   ```
7. Define the Interrupt handlers if required. Code example:

```
void I2C_Collision_Handler( void )//
{
      //Collision handling code
}

void I2C_Handler( void )//
{
      //Communications code
}
```

In the following paragraphs the several phases of the slave side of the communications will be described.

8.8.1.1.1 I^{2}C Slave Access Sequence

Note: I^{2}C address phases are always prefixed by Start or Repeated Start condition.

7-bit Address mode

The slave, once enabled, waits for an I^{2}C Start condition to happen. Once a Start condition is detected, the slave shifts in the next 8 bits into an internal shift register and the following actions take place:

The **IBUFF** bit is set.

If the address contained in the internal register matches the one from **I2CADDR0** the interface sends back an ACK and an interrupt is asserted.

At this moment the application must read the **I2CSTATUS** register and check the **IADDRR** and **IRWBUSY** bits.

The **IADDRR** should be ‘1’ (address received).

The **IRWBUSY** will inform if the operation is a write (‘1’) or a read (‘0’).

After reading these bits the application must read the **I2CDATA** register to clear the buffer. The figures 1, 2 and 3 show how the process occurs.

If **IBUFF** is set before receiving the address or **IRBUFOVL** is set when receiving the address, then the slave will send NACK and issue an error interrupt to notify the application about these errors.

10-bit Address mode

Two address-byte receptions are required in this mode. The first byte shifted consists of '1 1 1 1 0 A[9] A[8] 0', where A[9:8] is the upper two bits of I^{2}C address.

The last bit, R1W0, must be 0 so the slave can receive another address byte. If the upper two address bits match then the Slave sends the ACK and asserts an interrupt.

The application must at this point read **I2CSTATUS** to check the **IADDRR** and **IRWBUSY**, which are 1 (address byte) and 0 (write operation). The user then needs to read **I2CDATA** to clear the buffer.

The second byte shifted into contains the address bits A[7:0]. In the same fashion if the lower 8 address bits match then the slave sends the ACK and asserts an interrupt.

The application reads **I2CSTATUS** to check the **IADDRR**, which is 1 (address byte). The user then needs to read **I2CDATA** to clear the buffer. Figure 12 shows an example of 10-bit address mode receiving timing waveform.

If it is an I^{2}C read access, then after the two address-byte receptions the slave shall receive a Repeated Start condition and then the first address byte again with last bit R1W0 being 1. The slave sends The ACK bit and asserts an interrupt

The application reads **I2CSTATUS** to check **IBUFF** and **IRWBUSY**, which are 1 (address byte) and 1 (read operation). The user then needs to read **I2CDATA** to clear the buffer.

(Figure 13 shows an example of 10-bit address mode transmitting timing waveform.)

8.8.1.1.2 I^{2}C Access Sequence – Write Data Phase

If the received R1W0 field is 0, it is an I^{2}C write access and the slave remains in receiving mode. Every time the slave shifts in a byte, it sends the ACK bit as long as **IBUFF** is cleared before receiving the data and **IRBUFOVL** is cleared when receiving the data.

The slave also asserts an interrupt after receiving each byte from I^{2}C bus. The user needs to read **I2CSTATUS** for status checking and Then **I2CDATA** for data fetching. The write data phase is concluded when detecting a Stop or Repeated Start condition. (Figure 9, Figure 10, and Figure 12 show examples of I^{2}C write accesses)

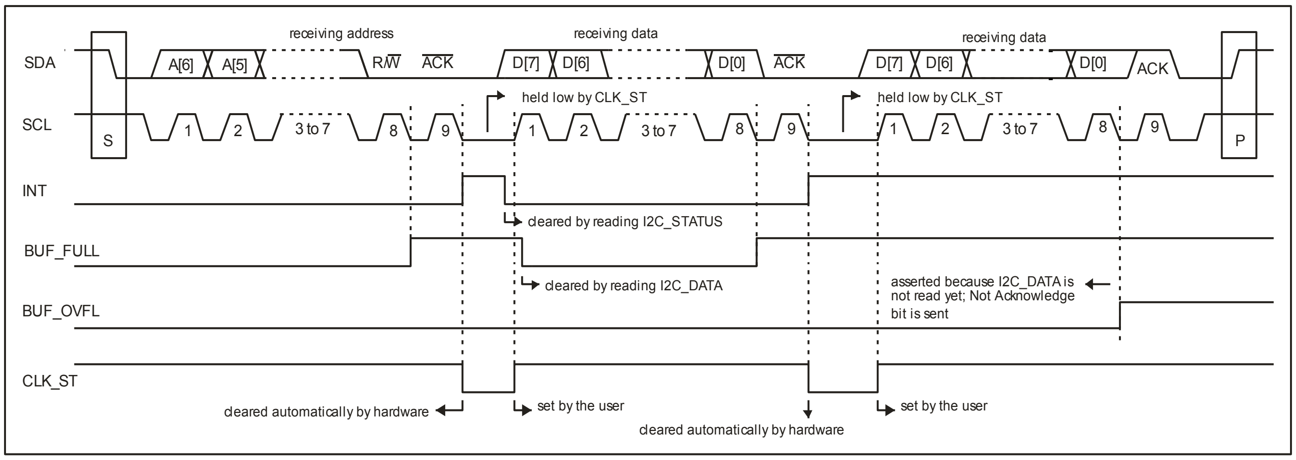


Figure 9 – Slave Mode Timing Waveform with CLK_ST_ENB = 1 (Reception, 7-bit Address Mode)

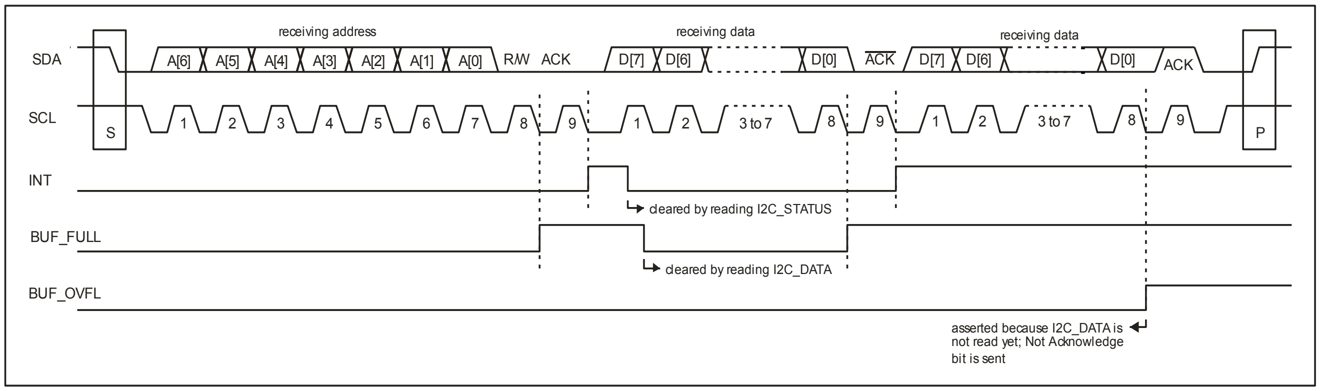


Figure 10 – Slave Mode Timing Waveform with CLK_ST_ENB = 0 (Reception, 7-bit Address Mode)

8.8.1.1.3 I^{2}C Access Sequence – Read Data Phase

If the received R1W0 field is 1, it is an I^{2}C read access and the slave switches to transmitting mode.

Before each byte shift out process, **ICLKSTR** is cleared automatically to hold SCL low (clock stretching). The user needs to program **I2CDATA** with the byte to be transmitted and then set **ICLKSTR** to release SCL. Every time the slave shifts out a byte, it receives the ACK/NACK bit. If receiving the ACK bit, the slave clears **ICLKSTR** automatically, and the user needs to program **I2CDATA** and set **ICLKSTR** to resume transmission. If receiving the NACK bit, which means the Master device is done reading data, the Slave releases both SCL and SDA. The Slave asserts an interrupt after receiving the ACK/NACK bit. The read data phase is concluded when receiving the NACK bit or detecting Repeated Start or Stop condition. Figure 11 and show examples of I^{2}C read accesses.

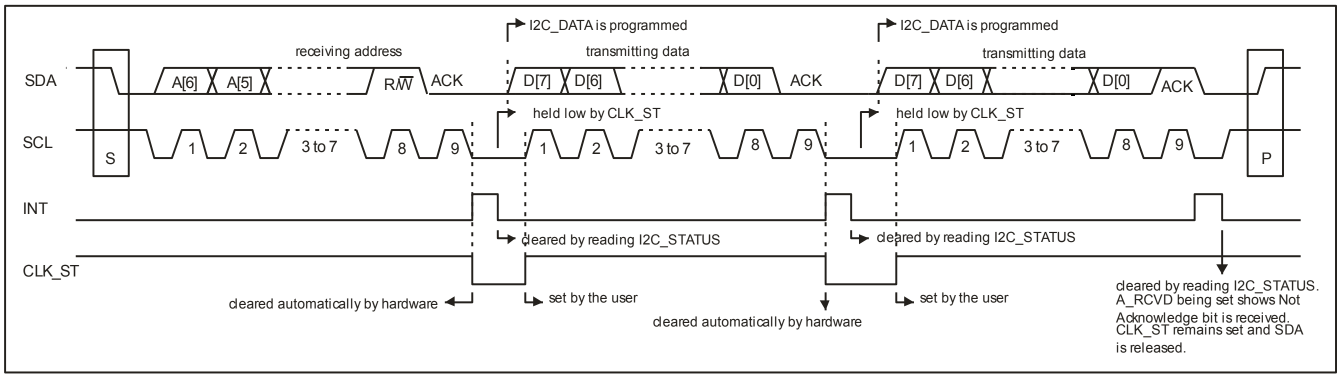

SDA
SCL
INT
CLK_ST
receiving address
A[6]
A[5]
R/W
ACK
I2C_DATA is programmed
transmitting data
D[7]
D[6]
D[0]
ACK
S
1
2
3 to 7
8
9
P
held low by CLK_ST
cleared by reading I2C_STATUS
cleared automatically by hardware
set by the user
cleared by reading I2C_STATUS. A_RCVD being set shows Not Acknowledge bit is received. CLK_ST remains set and SDA is released.

Figure 11 – Slave Mode Timing Waveform (Transmission, 7-bit Address Mode)

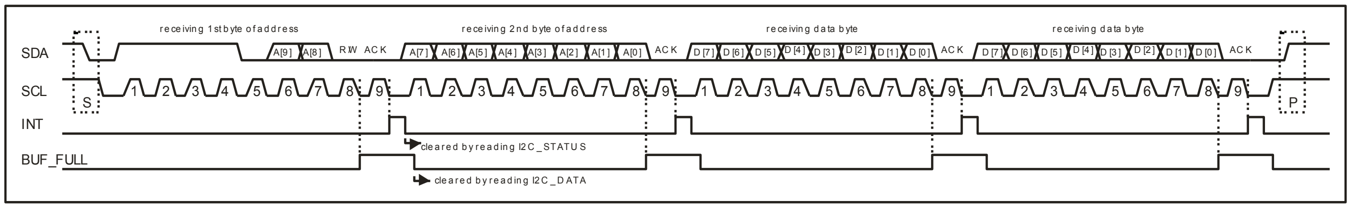


Figure 12 – Slave Mode Timing Waveform (Reception, 10-bit Address Mode)

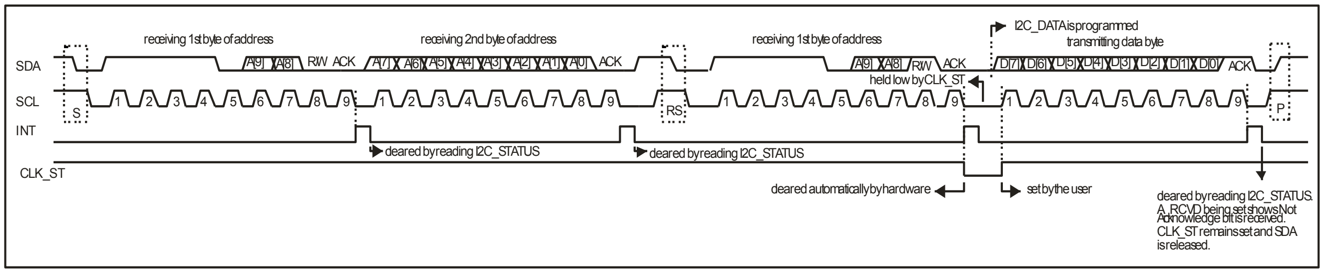

SDA
SCL
INT
CLK_ST
receiving 1st byte of address
receiving 2nd byte of address
I2C_DATA is programmed
transmitting data byte
A[9]
A[8]
R/W
ACK
RS
S
P
held low by CLK_ST
cleared by reading I2C_STATUS
cleared automatically by hardware
set by the user
cleared by reading I2C_STATUS. A_RCVD being set shows Not Acknowledge bit is received. CLK_ST remains set and SDA is released.

Figure 13 – Slave Mode Timing Waveform (Transmission, 10-bit Address Mode)

8.8.1.2 Master Mode

Master Mode:

1. Select the master mode. Code example:
   ```
   I2C_Mode( I2C_MASTER );
   ```
2. Define the baud rate. The equation defining the baud rate as a function of the system clock is:

   $$Fi2c = \frac{Fclk}{2 * (divider + 1)}$$

   Where Fi2c is the frequency of the I^2C interface, divider is the i2c divider and Fclk is the system clock. Code example to select an I^2C clock of 100kbits/sec. (Assuming a system clock of 4MHz)

   ```
   I2C_WriteClockDivider( 19 );  //I2C baud = 100kbits/sec
   ```
3. Enable the interface. Code example:
   ```
   I2C_Enable( I2C_ON );//Enabling the interface
   ```
4. Enable the interrupts and define the corresponding handlers, exactly like in items 6 and 7 from the slave mode.

From this point on the application must handle the communication. The following paragraphs will describe the general steps:

As an I^2C Master, the Master controls SCL and SDA when issuing Start, Repeated Start and Stop conditions. It also drives (release/drain) SCL and SDA when transmitting address/data bytes as well as ACK/NACK bits after receiving data bytes.

8.8.1.2.1 Configuration Settings

As an I^2C Master, the Master controls SCL and SDA when issuing Start, Repeated Start and Stop conditions. It also drives (release/drain) SCL and SDA when transmitting address/data bytes as well as ACK/NACK bits after receiving data bytes.

When **IEN** and **IMS** fields are both set the interface is configured as an I^2C Master.

8.8.1.2.2 Baud Rate Generators Configuration

A baud rate generator (BRG) inside the peripheral serves as an engine to time SCL transition during data transfer as well as the transitions of both SCL/SDA during Start, Repeated Start and Stop conditions.

BRG consists of an 8-bit counter that when enabled, loads the value from **I2CADDR0** and counts down to 0 and Then goes back to **I2CADDR0** value and repeats counting down process. When BRG counter counts down to 0, it triggers the SCL transitions during data transfer and SCL/SDA transitions during Start, Repeated Start and Stop conditions.

The peripheral's baud rate is determined by the system clock frequency F_{clk} and divider. The equation is:

$$Fi2c = \frac{Fclk}{2 * (divider + 1)}$$

8.8.1.2.3 Start Condition

The Master issues a Start condition when **IRSTRB** is set by the user. When detecting the issued Start condition the Master asserts an interrupt and clears **ISTRSTRETCH**. The user reads **I2CSTATUS** to clear the interrupt condition. At this point the **ISTRR** is set.

Once **ISTRSTRETCH** is set, if SCL is sampled low first before SDA goes low or if SCL or SDA is already sampled low when **ISTRSTRETCH** is set, Then There is a bus collision due to another I^2C Master on the bus, and the bus collision interrupt is asserted. The application must read **I2CSTATUS** to clear this interruption condition.

8.8.1.2.4 Repeated Start Condition

The Master issues a Repeated Start condition when **ISTRSTRETCH** is set by the user. When detecting The issued Repeated Start condition, the Master asserts an interrupt and clears **IRSTR**. The user reads **I2CSTATUS** to clear the interruption condition. At this point the **ISTRR** is set.

Once **IRSTR** is set, if SCL is sampled low first before SDA goes low, or if SDA is sampled low when SCL goes from low to high, Then There is an I^2C bus collision and a collision interrupt is asserted. The user reads **I2CSTATUS** to the interrupt condition.

8.8.1.2.5 Stop Condition

The Master issues a Stop condition when **ISTPSIZE** is set by the user. When detecting the issued Stop condition the Master asserts an interrupt and clears **ISTPSIZE**. The user reads **I2CSTATUS** to clear the interrupt condition. **ISTPR** is set.

Once **ISTPSIZE** is set, if SDA is sampled low one baud period (T_{br}) after it is released by the peripheral, or after SCL is released, SCL is sampled low before SDA goes high, Then There is an I^{2}C bus collision and a collision interrupt is asserted. The user reads **I2CSTATUS** to clear the collision interrupt condition.

8.8.1.2.6 Acknowledge Bit

The Master transmits ACK/NACK bit when **ISACK** is set by the user. If the value of **ISNACK** is 1, a NACK bit is transmitted otherwise an ACK bit is transmitted. The Master asserts an interrupt and clears **ISNACK**. The user reads **I2CSTATUS** to clear the interrupt condition.

If the Master transmits a NACK bit but detects an ACK bit, Then There is an I^{2}C bus collision and a collision interrupt is asserted. The user reads **I2CSTATUS** to clear the collision interrupt.

8.8.1.2.7 I2C Write Access Sequence

The typical I^{2}C write access sequence consists of the following steps:

- The user sets **ISTRSTRETCH** to issue a Start condition.
- The Master detects the Start condition and asserts an interrupt. The user reads **I2CSTATUS** to clear the interrupt and check **ISTRR**.
- The user programs **I2CDATA** with the destined I^{2}C Slave's address, Then the Master starts transmitting the address byte.
- After sampling ACK/NACK bit sent by the Slave, the Master asserts an interrupt. The user reads **I2CSTATUS** to clear the interrupt condition and check **IACKR**.
- The user programs **I2CDATA** with the data byte to be transmitted, Then the Master starts transmitting the data byte.
- After sampling ACK/NACK bit sent by the Slave, the Master asserts an interrupt. The user reads **I2CSTATUS** to clear the interrupt condition and check **IACKR**.
- Repeat steps 5 and 6 to transmit more bytes.
- The user can access a different I^{2}C Slave or read from the same one by setting **IRSTR**, Then Master issues a Repeated Start condition. When the condition is sampled the Master asserts an interrupt. The user reads **I2CSTATUS** to clear the interrupt condition and check **ISTRR**.
- The user concludes current transfer by setting **ISTPSIZE**, and the Master issues a Stop condition. When the condition is sampled the Master asserts an interrupt. The user reads **I2CSTATUS** to clear the interrupt and check **ISTPR**.

Figure 14 shows a timing waveform of Master write access. Note that the only difference between 7-bit and 10-bit address mode is that the user needs to program two address bytes in 10-bit mode.

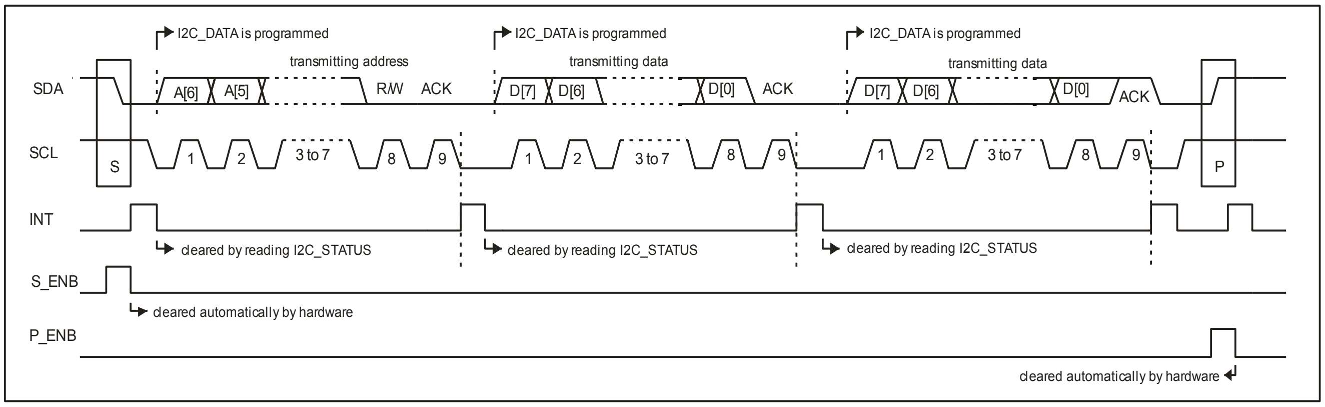


Figure 14 – Master Timing Waveform (Transmission)

8.8.1.2.8 I²C Read Access Sequence

The typical I²C read access sequence consists of the following steps:

- The user sets **ISTRSTRETCH** to issue a Start condition.
- The Master detects the Start condition and asserts an interrupt. The user reads **I2CSTATUS** to clear the interrupt and check **ISTRR**.
- The user programs **I2CDATA** with the destined I²C Slave's address, Then the Master starts transmitting the address byte.
- After sampling ACK/NACK bit sent by The Slave, the Master asserts an interrupt. The user reads **I2CSTATUS** to clear the interrupt and check **IACKR**.
- The user sets **IRCSTRT**, which enables the Master to pulse SCL and shift in data byte. After shifting in the whole data byte the Master asserts an interrupt. The user reads **I2CSTATUS** to clear the interrupt. The user then reads **I2CDATA** to fetch the received data byte.
- The user clears **ISNACK** (Acknowledge bit to be sent) and sets **ISACK**. The Master transmits ACK bit. After the transmission the Master asserts an interrupt. The user reads **I2CSTATUS** to clear the interrupt.
- Repeat steps 5 and 6 to receive more bytes.

- The user can access a different I^2C Slave or write to the same one by setting **IRSTR**, then the Master issues a Repeated Start condition. When the condition is sampled the Master asserts an interrupt. The user reads **I2CSTATUS** to clear the interrupt and check **ISTRR**.
- If the data byte being received is the last one, after clearing the interrupt, the user sets **ISNACK** (Not Acknowledge bit to be sent) and **IRSTS**. Master transmits NACK bit. After the transmission the Master asserts an interrupt. The user reads **I2CSTATUS** to clear the interrupt.
- The user concludes current transfer by setting **ISTPSIZE**, and the Master issues a Stop condition. When the condition is sampled the Master asserts an interrupt. The user reads **I2CSTATUS** to clear the interrupt and check **ISTPR**.

Figure 15 shows an example of Master read access.

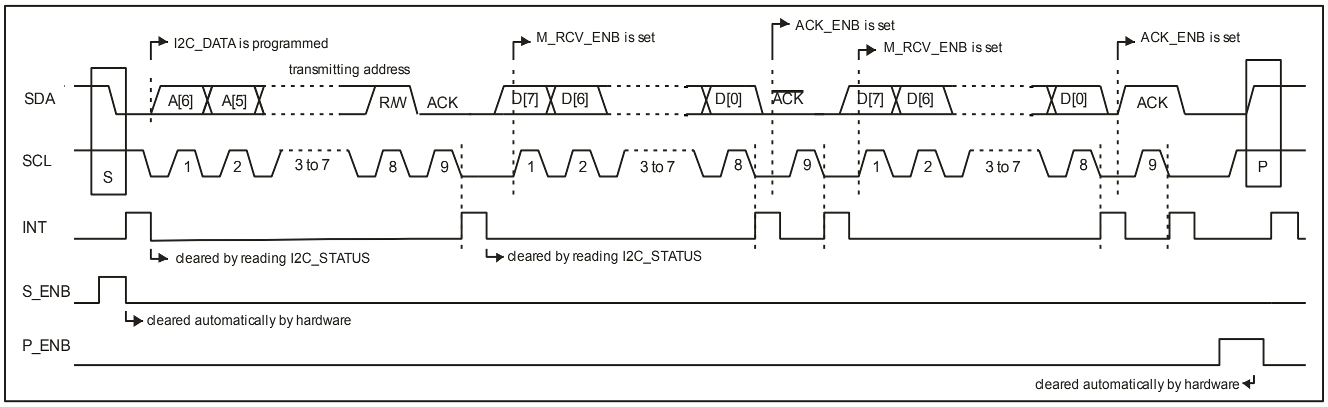


Figure 15 – Master Timing Waveform (Reception)

8.8.1.2.9 RW_BUSY Indicator

Whenever detecting a Start/Stop condition it does not issue, the Master asserts/de-asserts RW_BUSY to indicate the busy/idle status of I^2C bus. The user can check **IRWBUSY** before issuing a Start condition so bus collision can be avoided.

8.8.2 I2C Registers

The following registers are made available by the I^{2}C interface:

I2CSTATUS: I^{2}C status register.

I2CSTATUS		0x50000008			0x00		
R	R	R	R	R	R	R	R
IACKR	IADDRR	ISTRR	ISTPR	IRWBUSY	IBUFF	IWBUFOVL	IRBUFOVL
MSB							LSB

Bit7 **IACKR**: Acknowledge received

0 = ACK received / 1 = ACK not received

Bit6 **IADDRR**: Data/Address received (slave mode)

0 = DATA received / 1 = ADDRESS received

Bit5 **ISTRR**: Start bit received

0 = Start bit not received

1 = Start bit received

Bit4 **ISTPR**: Stop bit received (slave mode)

0 = No stop bit received

1 = Stop bit received

Bit3 **IRWBUSY**: Read/Write Busy:

Master Mode:

0 = Bus not being accessed

1 = Bus being accessed

Slave Mode:

0 = I2C write operation (Slave receives data)

1 = I2C read operation (Slave transmits data)

Bit2 **IBUFF**: Buffer Full

0 = Buffer is empty

1 = Buffer full, either because it received data or There is one byte to be transmitted

Bit1 **IWBUFOVL**: Write buffer overflow

0 = Buffer is empty

1 = Internal shift register is full and I2CDATA was written to

Bit0 **IRBUFOVL**: Data/Address received (slave mode)

0 = Register was read

1 = Internal shift register is full and another byte is received from I2C bus.

NOTE: While **IRBUFOVL** is set the shift-in of bits from bus is stopped.

I2CCTRL1: I^{2}C control register 1.

I2CCTRL1		0x50000009			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
IRSTR	ICLKSTR	IGC	IRCSTRT	ISNACK	ISACK	ISTPSIZE	ISTRSTRETCH
MSB							LSB

Bit7 **IRSTR**: Repeated start bit (Master Only)

0 = Repeated start bit disabled

1 = Issue start-bit transmit enable (when set the I2C transmits Repeated Start bit), cleared by HW.

Bit6 **ICLKSTR**: Clock stretch (Slave Only)

0 = SCL held low / 1 = SCL released

Bit5 **IGC**: General call address (Slave Only)

0 = General call address disabled / 1 = General call address enabled

Bit4 **IRCSTRT**: Start bit reception (Master Only and cleared by HW)

0 = Receive operation not allowed

1 = Receive operation starts (the receive operation starts when this bit is set)

Bit3 **ISNACK**: ACK bit to be transmitted: (Master Only)

0 = ACK is transmitted upon reception of byte / 1 = NACK is transmitted upon reception of byte

Bit2 **ISACK**: ACK bit (Master Only)

0 = No ACK/NACK bit transmitted

1 = Acknowledge (ACK/NACK, defined by ISNACK) is transmitted

Bit1 **ISTPSIZE**: Stop bit or selection of address size

Master Mode

- 0 = No stop bit sent
- 1 = Stop bit sent

Slave Mode

- 0 = 7-bit address
- 1 = 10-bit address

Bit0 **ISTRSTRETCH**: Start and stretch

Master Mode

- 0 = No start bit sent
- 1 = Send start bit

Slave Mode

- 0 = no clock stretch
- 1 = Clock stretched

I2CCTRL2: I^{2}C control register 2.

I2CCTRL2		0x5000000A			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
IMSK4	IMSK3	IMSK2	IMSK1	IMSK0	IEN	IFILTER	IMS
MSB							LSB

Bit7-3	**IMSK[4:0]**: I^{2}C Address mask (Slave Only)
Bit2	**IEN**: I^{2}C Enable bit
	0 = I^{2}C disabled
	1 = I^{2}C enabled
Bit1	**IFILTER**: I^{2}C filter
	0 = Filter disabled
	1 = Filter enabled
Bit0	**IMS**: I^{2}C Master/Slave
	0 = I^{2}C slave
	1 = I^{2}C master

NOTE: In order to ignore the glitches on I^{2}C bus, a 3-tab median filter operating at system clock rate is implemented on the incoming SCL and SDA data paths. This filter can be enabled/disabled by setting/clearing IFILTER. The truth table of the median filter is shown below.

Table 16 - Filter Tabs and output

Filter tab 0	Filter tab 1	Filter tab 2	Filter output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

I2CDATA: I^{2}C data.

I2CDATA		0x5000000B			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
IDT7	IDT6	IDT5	IDT4	IDT3	IDT2	IDT1	IDT0
MSB							LSB
Bit7-0 **IDT[7:0]**: I^{2}C Data							

I2CADDR0: I^{2}C address 0.

I2CADDR0		0x5000000C			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
IADDR7	IADDR6	IADDR5	IADDR4	IADDR3	IADDR2	IADDR1	IADDR0
MSB							LSB
Bit7-0 **IADDR[7:0]**: I^{2}C Data register low.							

I2CADDR1: I^{2}C address 1.

I2CADDR1		0x5000000D			0x00		
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	R/W	R/W
-	-	-	-	-	-	IADDR9	IADDR8
MSB							LSB
bit7-0 **IADDR[9:8]**: I^{2}C Data register high.							

8.9 ADC

µSesame features two analog to digital converters (ADC1 and ADC2). These ADCs are used for short circuit detection algorithm. However, when they are not used for the short circuit detection, they are available for the general purpose. Each ADC is an 8-bit analog to digital converter with single ended input. The main features are described below:

- 8-bit resolution
- Single ended input
- Up to 80 kSPS
- Configurable reference (VREF = VREFHI-VREFLO)
  - Either based on the bandgap voltage(VBG) or regulated supply voltage(VDD)
  - Scalable
- ADC input range is from 0V to VDD
- The 8 bit resolution may be targeted over a reduced input voltage range via a programmable gain block
- Total of 28 channels (15 in ADC1 and 13 in ADC2)

8.9.1 ADC Description

µSesame ADC uses the standard charge redistribution technique, with a single-ended input and internally generated positive and negative reference voltages. There are two 8-bit ADC converters µSesame. ADC1 accommodates 15 analog input channels while ADC2 accommodates 13 analog input channels. The user can select which input channels to be sampled by setting ADCCHANNEL register. Each ADC has its own internally generated reference voltages (VREFHI and VREFLO). The performance table is shown below.

Table 17 : ADC Performance Specification, Recommended Operating Conditions, unless otherwise specified

Parameter	Conditions	min	typ	max	unit
Conversion speed				80	ksps
Clock Frequency				1	MHz
Input voltage range		0		VDD	V
Resolution				8	bits
INL				1	LSB
DNL				1	LSB

There are several steps required for the user to use the ADC. The general sequence is described below:

1. Select the input channel to be selected
   a. ADCCH1 and ADCCH2 bits control the input multiplexor
   b. Configure the input range with LVL bit
2. Configure ADC settings.
   a. Set ADC clock frequency.
   b. Configure references by setting ADCREFHI, ADCREFLO, ADCPGN, and ADCREFS bits.
3. Enable ADC.
4. Start the ADC conversion.
5. Check the ADC status bit and read the data.

The following section will describe each configuration steps in detail.

8.9.1.1 Input Channel Selection

All high voltage GPIOs (PA[7:0], PB[7:0], and PC[7:0]), the bandgap voltage (VBG), and the battery voltage (VBAT) are available as an input to the ADCs. The user can control which inputs are connected for the conversion by programming the control bits, ADDCH1 and ADDCH2 in ADDCHANNEL register.

Please note that since the high voltage GPIOs can have a signal that ranges from 0 to VBAT and the input range of the ADC is from 0 to VDD, it is necessary to have an option to attenuate the signal if the user wants to convert the full signal range for the GPIO. Each GPIO has a programmable control bit (LVL) to attenuate its signal by a factor of 8.

Code Example: Selecting PA0 for ADC1 input channel and VBG for ADC2 input channel

```
ADC_Select_Channel( ADC1_PA0, ADC2_BG );
```

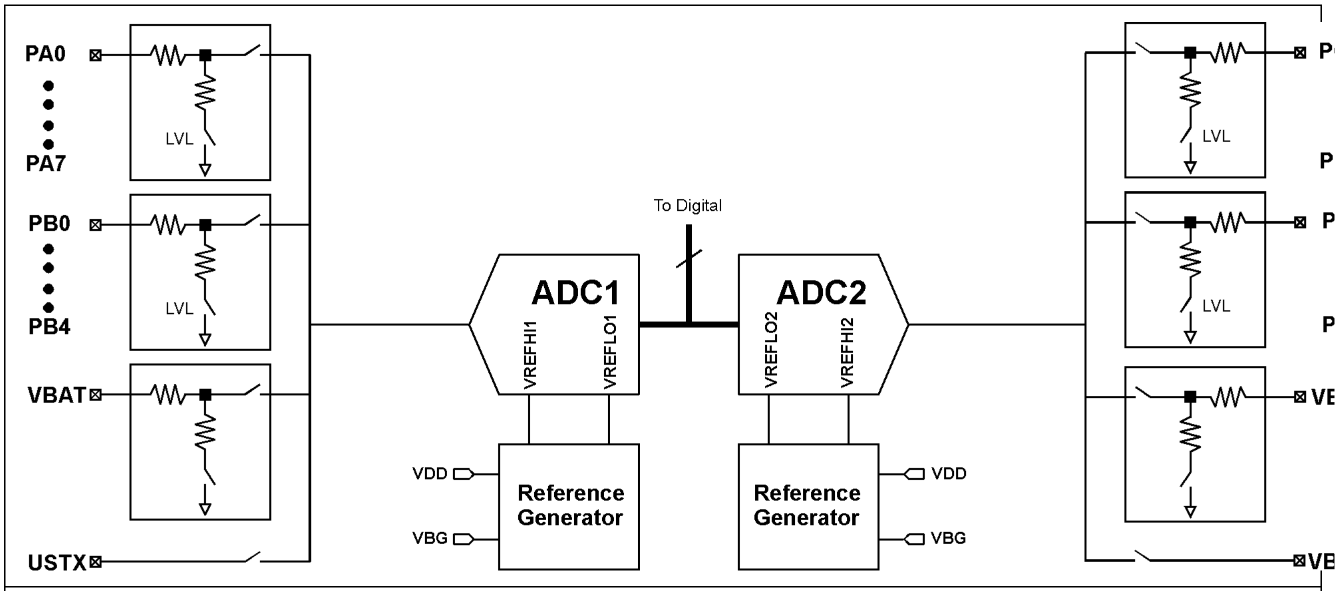


Figure 16– ADC Block Diagram

8.9.1.2 ADC Clock and Sampling Period

The conversion algorithm has a basic period of 9 cycles (one for sampling, one for each bit). There is a two-cycle latency from the last bit measurement and digital data availability. Additionally there is single idle cycle to allow biasing before any conversion is initiated. Thus a single conversion will take 13 cycles.

The converter will use a single clock cycle to sample the input into an input capacitor. When the channel is selected, the source must drive the S/H capacitor through the series resistance. The sampling time will vary with this resistance. The input to ADC must have sufficiently low driving impedance and settling time to settle the input to within 1 LSB of the data conversion during the input sampling stage. An equivalent circuit and related equations are depicted in Figure 17.

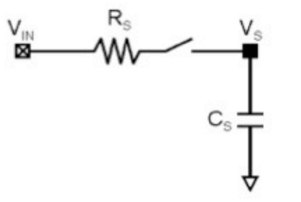
The ADC clock frequency can be programmed through ADCCLKDIV register. As an example, if ADC clock is derived from 3.58MHz crystal divided by 4, the input has 1.12µs to settle. Since the C_s = 10pF in µSesame, the maximum source resistance to guarantee 8-bit performance can be calculated as below:

$$R_s = \frac{1.12\mu s}{10\,pF \times 6} = 18.6k\Omega$$

If the source impedance is larger, the user can reduce the ADC clock frequency.

Code Example: Configure ADC clock to crytal frequency divided by 16.

```
ADC_ClkDiv (16);
```



$$V_S = V_{IN}(1 - e^{-\frac{t}{\tau}}) \qquad , where\ \tau = R_S C_S$$

$$V_S \textit{ is within 1 LSB of } V_{IN} \textit{ after } t = 5.5\tau$$

Figure 17– ADC Input Settling Time

8.9.1.3 Configuration of Reference Voltages for the ADC

The ADC can generate its reference voltages (VREFHI and VREFLO) from two different sources, the regulated supply voltage (VDD) or the bandgap voltage (VBG). The ADCREFS bit in ADCREG3 register selects the source. Once the reference source is selected, the reference voltages can be programmed through ADCREFHI, ADCREFLO, and ADCPGN bits according to Figure 18. It should be noted that when VBG is used as the reference source, care must be taken so that the internal voltages do not saturate.

ADCREFS=0*	ADCREFS=1
$VREFHI = \frac{ADCREFHI}{ADCPGN} \times VBG$	$VREFHI = \frac{ADCREFHI}{15} \times VDD$
$VREFLO = \frac{ADCREFLO}{ADCPGN} \times VBG$	$VREFLO = \frac{ADCREFLO}{15} \times VDD$

* If $\left(\frac{15}{ADCPGN} \times VBG\right) < (VDD - 0.1V)$

Figure 18– ADC Reference Voltage

Once the reference voltages are established, the ADC conversion equation for input voltage (VIN) can be defined as:

$$ADCDT = floor\left(255 \times \frac{(VIN - VREFLO)}{(VREFHI - VREFLO)}\right)$$

Here are few examples:

- Example 1: In a system operating with VDD=3V, there is a signal that moves between 0V and 2.94V. In this case it would be recommended to select VDD as the reference source and ADCREFH=15, ADCREFL=0, and ADCPGN=15. This selection would allow for the maximum range of measurements (0V to VDD).
  - o `ADC_Reference_Config( ADC1, 15, 0, 15, ADCREFVDD);`
- Example 2: In a system operating with VDD=3V and VBG= 1.21V, there is a signal that moves between 0V and 2.5V. In this case VBG can be selected as the reference source with ADCREFH=13, ADCREFL=0 and ADCPGN=7. This configuration would allow the signal range from 0V to 2.6V:
  - o `ADC_Reference_Config( 13, 0, 7, ADCREFBG);`

- Example 3: In a system operating with VDD=3V and VBG=1.21V, there is a signal that moves between 1.71V and 2.2V. In this case selecting VBG as the reference source and select ADCREFH=15, ADCREFL=11, and ADCPGN=8 we can achieve higher resolution:
  - `ADC_Reference_Config( 15, 11, 8, ADCREFBG);`

The resolution in Example 3 can be calculated as follow:

$$RESOLUTION = \left(\frac{VREFHI - VREFLO}{255}\right) = \left(\frac{\left(\frac{15}{8} \times 1.21\right) - \left(\frac{11}{8} \times 1.21\right)}{255}\right) = \frac{2.27 - 1.66}{255} = 2.38mV$$

Note that in this particular case we have the 8-bit ADC effectively generating a digital value with the precision of a 10-bit ADC operating from 0V to 3V.

It is clear from the examples how flexible the ADC can be in a range of applications. The user can devise several schemes to cleverly measure the range of signal of interest and then narrow the reference values to get the optimum resolution if the conversion time is within range.

8.9.1.4 ADC Start and Status

Before starting the conversion, the ADC must be enabled and biased. The ADC is enabled by the ADCEN bit (**ADCREG3**). The START bit (**ADCSTART**) starts the conversion process. Once completed the value of the conversion is loaded into the ADCDATA registers.

Code Example: Enable the converter and start conversion

```
ADC_Enable(ADCEN);
ADC_Start();                                //ADC enabled and start
while ( ADC_ConversionComplete() == 1 );   //Wait until completed
```

8.1.1 ADC Registers

The following registers define the behavior of the ADC:

ADCCHANNELS: Channel selection for both ADCs.

ADCCHANNELS		0x50000054			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADC2CH3	ADC2CH2	ADC2CH1	ADC2CH0	ADC1CH3	ADC1CH2	ADC1CH1	ADC1CH0
MSB							LSB

Bit7-4 **ADC2CH[3:0]**: Channel Selection for ADC2:

Bit3-0 **ADC1CH[3:0]**: Channel Selection for ADC1:

ADC2CH[3:0]	ADC1CH[3:0]
0000 = PC0	0000 = PA0
0001 = PC1	0001 = PA1
0010 = PC2	0010 = PA2
0011 = PC3	0011 = Not used
0100 = PC4	0100 = PA4
0101 = PC5	0101 = PA5
0110 = PC6	0110 = PA6
0111 = PC7	0111 = PA7
1000 = PA3	1000 = PB0
1001 = PB5	1001 = PB1
1010 = PB6	1010 = PB2
1011 = PB7	1011 = PB3
1100 = NC	1100 = PB4
1101 = NC	1101 = USTX Signal (Ultrasound transmit)
1110 = Reserved	1110 = Reserved
1111 = NC	1111 = NC

ADCSTART: ADC start of conversion control

ADCSTART		0x50000055			0x00		
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	R/W
–	–	–	–	–	–	–	START
MSB							LSB
Bit0 **START**: Writing one starts the conversion. Reading returns the status of conversion; '0' means conversion Is finished and '1' means the conversion is ether pending or in progress							

ADC1DATA: ADC1 result.

ADC1DATA		0x50000056			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
AD1DT7	AD1DT6	AD1DT5	AD1DT4	AD1DT3	AD1DT2	AD1DT1	AD1DT0
MSB							LSB
Bit7-0 **AD1DT[7:0]**: ADC1 Result							

ADC2DATA: ADC2 result.

ADC2DATA		0x50000057			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
AD2DT7	AD2DT6	AD2DT5	AD2DT4	AD2DT3	AD2DT2	AD2DT1	AD2DT0
MSB							LSB
Bit7-0 **AD2DT[7:0]**: ADC1 Result							

ADCCLKDIV: ADCs Clock Divider Control.

ADCCLKDIV		0x5000005A			0x6F		
Reserved	R/W	Reserved	Reserved	R/W	R/W	R/W	R/W
-	-	-	-	-	ADCDIV1	-	ADCDIV0
MSB							LSB

Bit2-0 **ADCDIV[1:0]:** ADC Clock divider
00 = System Clock/2
01 = System Clock/4
1x = System Clock/1

ADCREG0: ADC1 Reference Settings

ADCREG0		0x50018008			0xF0		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADC1REFH3	ADC1REFH2	ADC1REFH1	ADC1REFH0	ADC1REFL3	ADC1REF L 2	ADC1REF L 1	ADC1REF L 0
MSB							LSB

Bit7-4 **ADC1REFH[3:0]:** ADC1 Reference High Setting
Bit3-0 **ADC1REFL[3:0]:** ADC1 Reference Low Setting

ADCREG1: ADC2 Reference Settings

ADCREG1		0x50018009			0xF0		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADC2REFH3	ADC2REFH2	ADC2REFH1	ADC2REFH0	ADC21REFL3	ADC2REF L2	ADC2REF L1	ADC2REF L0
MSB							LSB

Bit7-4 **ADC2REFH[3:0]:** ADC2 Reference High Setting
Bit3-0 **ADC2REFL[3:0]:** ADC2 Reference Low Setting

ADCREG2: ADCs Reference Gain Values

ADCREG2		0x5001800A			0xFF		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADC1PGN3	ADC1PGN2	ADC1PGN1	ADC1PGN0	ADC2PGN3	ADC2PGN2	ADC2PGN1	ADC2PGN0
MSB							LSB

Bit7-4 **ADC1PGN[3:0]:** ADC1 Reference Gain

Bit3-0 **ADC2PGN[3:0]:** ADC2 Reference Gain

ADCREG3: ADCs General Control

ADCREG3		0x5001800B			0x13		
Reserved	R/W	R/W	R/W	R/W	R/W	R/W	R/W
-	ADCEN	ADCGNDOFF	ADC_SW1	ADC_SW0	ADC_CAL	ADC2REFS	ADC1REFS
MSB							LSB

Bit6 **ADCEN:** ADCs Enable Bit.

0 = ADCs Disabled / 1 = ADCs Enabled

Bit5 **ADCGNDOFF:** Ground offset correction

0 = bandgap and ADC reference have common ground

1 = difference between bandgap and ADC reference is compensated by switched capacitor circuit

Bit4-3 **ADCSW[1:0]:** ADCs Enable Bit.

00 = Correlated double sampling off

01 = Input offset calibration on

1x = Correlated doubling sampling on (default)

Bit2 **ADCCAL:** ADC Calibration

0 = Normal

1 = Calibration mode

Bit1 **ADC2REFS:** ADC2 Internal Reference Source Selection

0 = Band Gap

1 = VDD

Bit0 **ADC1REFS:** ADC1 Internal Reference Source Selection

0 = Band Gap

1 = VDD

8.10 PULSE WIDTH MODULATORS (PWM)

µSesame implements two PWMs. Their main characteristics are:

- Twelve bits resolution - Both period and width.
- Independent Prescalers
- Programmable active level
- Short Circuit detection circuit with programmable level
- Programmable outputs
  - PWM1 = [PB0, PC5 and PC7]
  - PWM2 = [PA7, PB1 and PC3]

8.10.1 PWMs Usage Description

The PWM circuit generates wide range high resolution modulated output for siren driver, led drivers and other drivers used in µSesame. Each PWM has total of 4 data and configuration registers to communicate with the microcontroller.

The waveform is controlled by 12-bit period word (PWMnPER and PWMnEXT) and 12-bit pulse width word (PWMnPW and PWMnEXT) are used to determine the output waveform.

The entire waveform can be scaled by adjusting the Prescaler value in PWMnCTRL. The prescaler divided value, PWM_DIV, can be set to one of eight different settings shown in Table 18.

Table 18 PWM Prescaler Divide Values

PWM_PRESC	PWM_DIV (f_{XO}/f_{PWM})
000	1
001	2
010	4
011	8
100	32
101	256
110	8,192
111	262,144

The output period is calculated as:

$$Period = \frac{1 + (PWM\_PER \times PWM\_DIV)}{SystemClock}$$

For PWM1 and for PWM2 the PWM pulse width is calculated as:

$$PulseWidth = \frac{1 + (PWM\_PW \times PWM\_DIV)}{SystemClock}$$

To control the active level of the PWM, a control bit **PWM_INV** is used. If this bit is set to one, the PWM output is low level during the pulse and one at other times, including if the PWM is disabled either by the user or by the short circuit protection.

Alternatively if **PWM_INV** is set to zero then the PWM outputs a high level during the pulse.

Code Example: Setting the PWM1, enabled, with period and width separated, not inverted, with a prescaler of 2^5:

```
PWM_Setup( PWM1, PWMEN, PWMNINV, PRESC5);
```

Code Example: Selecting the period and pulse width of the same PWM1. (Prescaler of 2^5)

```
PWM_Period (PWM1,373);
PWM_Width (PWM1,280);
```

NOTE: From the above equations we can see that for a system clock of 3.579545MHz the period is of 3.325msec. (Frequency of ~300Hz with a width of 249.4msec, duty cycle of ~75%.)

8.10.2 PWMs Registers

The following registers are provided to control the PWMs:

PWM1CTRL: PWM1 General Control

PWM1CTRL		0x50000048			0x00		
R/W	Reserved	Reserved	R/W	Reserved	R/W	R/W	R/W
PWM1_EN	-	-	PWM1_INV	-	PRESC2	PRESC1	PRESC0
MSB							LSB

Bit7 **PWM1_EN:** PWM1 enable bit.
0 = PWM Disabled
1 = PWM Enabled

Bit4 **PWM1_INV:** PWM1 output signal direction
0 = normal logic
1 = inverted logic (active low)

Bit2-0 **PRESC[2:0]:** PWM's Prescaler
000 = System Clock/1
001 = System Clock/2
010 = System Clock/4
011 = System Clock/8
100 = System Clock/32
101 = System Clock/256
110 = System Clock/8192
111 = System Clock/262144 (2^{18})

PWM1PER: PWM1 period high byte register

PWM1PER		0x50000049			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PWM1PER11	PWM1PER10	PWM1PER9	PWM1PER8	PWM1PER7	PWM1 PER6	PWM1 PER5	PWM1 PER4
MSB							LSB

Bit7-0 **PWM1PER[11:4]:** PWM1 period high register

PWM1PW: PWM1 width high byte register

PWM1PW		0x5000004A			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PWM1PW11	PWM1PW10	PWM1PW9	PWM1PW8	PWM1PW7	PWM1 PW6	PWM1 PW5	PWM1 PW4
MSB							LSB
Bit7-0 **PWM1PW[11:4]:** PWM1 width high register							

PWM1EXT: PWM1 extension with low nibble of period and width.

PWM1EXT		0x5000004B			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PWM1PER3	PWM1PER2	PWM1PER1	PWM1PER0	PWM1PW3	PWM1 PW2	PWM1 PW1	PWM1 PW0
MSB							LSB
Bit7-4 **PWM1PER[3:0]:** PWM1 period low nibble Bit3-0 **PWM1PW[3:0]:** PWM1 width low nibble							

PWM2CTRL: PWM2 General Control

PWM2CTRL		0x5000004C			0x00		
R/W	Reserved	Reserved	R/W	Reserved	R/W	R/W	R/W
PWM2_EN	-	-	PWM2_INV	-	PRESC2	PRESC1	PRESC0
MSB							LSB
Bit7 **PWM2_EN:** PWM2 enable bit. 0 = PWM Disabled 1 = PWM Enabled Bit4 **PWM2_INV:** PWM2 output signal direction 0 = normal logic 1 = inverted logic (active low)							

Bit2-0	**PRESC[2:0]:** PWM's Prescaler 000 = System Clock/1 001 = System Clock/2 010 = System Clock/4 011 = System Clock/8 100 = System Clock/32 101 = System Clock/256 110 = System Clock/8192 111 = System Clock/262144 (2^{18})

PWM2PER: PWM2 period high byte register

PWM2PER		0x5000004D			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PWM2PER11	PWM2PER10	PWM2PER9	PWM2PER8	PWM2PER7	PWM2 PER6	PWM2 PER5	PWM2 PER4
MSB							LSB
Bit7-0 **PWM2PER[11:4]:** PWM2 period high register							

PWM2PW: PWM2 width high byte register

PWM2PW		0x5000004E			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PWM2PW11	PWM2PW10	PWM2PW9	PWM2PW8	PWM2PW7	PWM2 PW6	PWM2 PW5	PWM2 PW4
MSB							LSB
Bit7-0 **PWM2PW[11:4]:** PWM2 width high register							

PWM2EXT: PWM2 extension with low nibble of period and width.

PWM2EXT		0x5000004F			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PWM2PER3	PWM2PER2	PWM2PER1	PWM2PER0	PWM2PW3	PWM2 PW2	PWM2 PW1	PWM2 PW0
MSB							LSB
Bit7-4 **PWM2PER[3:0]:** PWM2 period low nibble Bit3-0 **PWM2PW[3:0]:** PWM2 width low nibble							

8.11 GPIOs

µSesame provides 36 general-purpose I/O pins. µSesame's I/O pins are implemented with several different capabilities divided into the following groups:

- GIO is a general purpose I/O referred to Vbat (from 9V up to 45V). When used as an output it is capable of sinking (to ground) 25mA or sourcing 5mA. When used as an input the GIO can be programmed to be high impedance, 100µA/5mA pull up or 100µA/5mA pull down. The state of the pin can be read while in output mode, therefore allowing for a software-based over current protection.
- SIO has the same functions as GIO, but with 200mA sink capability, which may be protected against over-current through software.
- PSIO (a single pin) has the same function as GIO, but with 200mA source capability, which may be protected against over-current through software.
- 3V3IO are 3.3V digital I/Os, which are referenced to an internal regulator.

The following table defines the main characteristics of the GPIO pins:

Table 19 -GPIO Characteristics, Typical Operating Conditions

I/O Type	Name	Conditions	Min.	Typ.	Max.	I/O Type	Name	Conditions	Min.	Typ.	Max.	Unit
GIO	V_{IL}	Threshold Low		1.65		SIO	V_{IL}	Threshold Low		1.65		V
		Threshold High		4				Threshold High		4		V
	V_{IH}	Threshold Low		1.65			V_{IH}	Threshold Low		1.65		V
		Threshold High		4				Threshold High		4		V
	I_{OL}			25			I_{OL}			200		mA
	I_{OH}			5			I_{OH}			5		mA
	Pull-Down	Strength Low		100			Pull-Down	Strength Low		100		µA
		Strength High		5				Strength High		5		mA
	Pull-Up	Strength Low		100			Pull-Up	Strength Low		100		µA
		Strength High		5				Strength High		5		mA
PSIO	V_{IL}	Threshold Low		1.65		3V3IO	V_{IL}			1.65		V
		Threshold High		4								
	V_{IH}	Threshold Low		1.65			V_{IH}			1.65		V
		Threshold High		4								
	I_{OL}			25			I_{OL}			25		mA
	I_{OH}			200			I_{OH}			5		mA
	Pull-Down	Strength Low		100			Pull-Down	Strength Low		-		µA
		Strength High		5				Strength High		-		mA
	Pull-Up	Strength Low		100			Pull-Up	Strength Low		-		µA
		Strength High		5				Strength High		-		mA

8.11.1.1 General-Purpose I/O (GIO)

The GIO interface pins are intended to operate as reconfigurable general-purpose inputs or outputs referenced to Vbat. Additionally, they can be used for open-drain pull-down on systems with voltage equal to or lower than their supply voltage, e.g. for 5V or 3.3V systems.

High current and low current pull-ups can be selected in receive mode, as well as a selectable threshold to receive at 3.3V, 5V or Vbat levels using signal thresholds of 1.65V and 4.0V respectively.

Additionally, outputs are protected against potentially damaging loads. When the high-level output is activated, the output current is limited by an internal circuit to 5mA, which can be sustained continuously.

When the low-level output is activated, protection against thermal damage caused by a short circuit must be done by user software by comparing the voltage level on the pad with the intended driven level shortly after activation using the 1.65V threshold receiver. If the level is different, i.e. above 1.65V threshold, then the user software must tri-state or activate a high level output within 200 milliseconds to avoid potential damage to the chip.

This protection is not intended to protect these pins against voltage overshoot from driving strongly inductive loads, and so GIO pins should not be used for inductive loads without additional protection on the PCB (**P**rinted **C**ircuit **B**oard).

Pin configuration is accomplished using special function registers, SFDICFG, PBnCFG, and PCnCFG.

The receiver is active at all times, and any read from the port will always return the data read from the pin, even if the pin is set as an output.

Table 20 - GIO and SIO Pin Functional Configuration

DD	PUP	PDN	Pin Function
0	0	0	high-Z input
0	0	1	input with pull down (current level set by STR)
0	1	0	input with pull up (current level set by STR)
0	1	1	reserved
1	X	0	push/pull (to VDD levels) output, with simple load protection
1	0	X	push/pull (to VDD levels) output, with simple load protection
1	1	1	open-drain output, with simple load protection

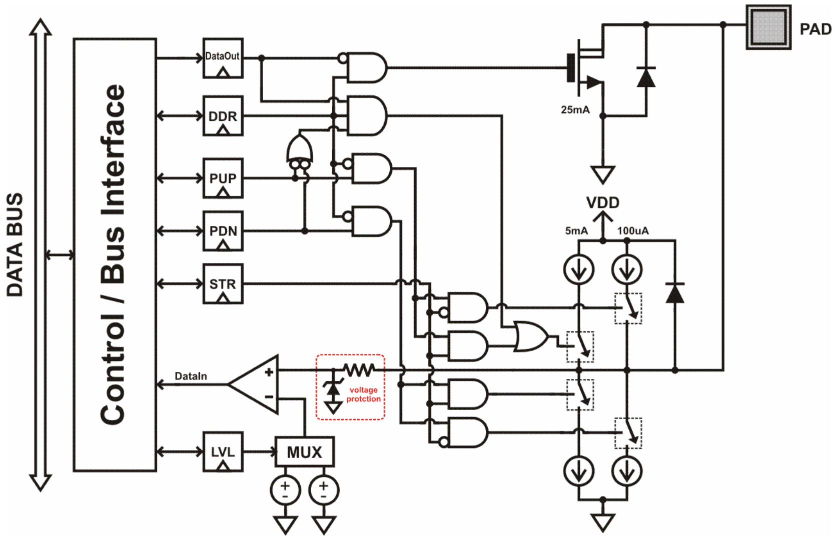


Figure 19 - Typical GIO Interface

8.11.1.2 High Current Pull down I/O (SIO)

SIO interface pins are intended to operate as reconfigurable inputs or outputs referenced to Vbat, optimized for use as high-current pull-downs. They can be used for open-drain pull-down on systems with voltage equal to or lower than their supply voltage, e.g. 3.3V or 5V systems.

High current and low current pull ups can be selected in receive mode, which uses a 4V signaling threshold level. Internal circuits are protected against sustained high voltage up to 45V applied to the pad.

A pull-down mode may be activated when using the I/O as an output. Pull-down output mode may be protected against potentially damaging loads by comparing the voltage level on the pad with a maximum level corresponding to a safe margin for thermal damage.

If the power dissipated in the transistor is too high, which happens when output voltage is above 1.65V with the pull down on, then the user software must turn the pull down off within approximately 200 milliseconds to avoid thermal damage.

In order to achieve this, the user software must read back from the pin with the threshold set to 1.65V as soon as possible after the pull down is activated in order to detect a short circuit or overload condition.

It is recommended that the pin be configured as push/pull when driving inductive loads to assist the freewheel function and reduce strain on the ESD diode which is responsible for conducting the freewheel current by allowing some current to pass through the PMOS transistor.

If the load is resistive, then the open drain mode of protection is preferred. Pull up protection is implemented in a same fashion as with the GIO type, by internally limiting the maximum current to 5mA, which can be sustained continuously at any pad output voltage within normal operating conditions.

The receiver is active at all times, and any read from the port will always return the data read from the pin, even if the pin is set as an output.

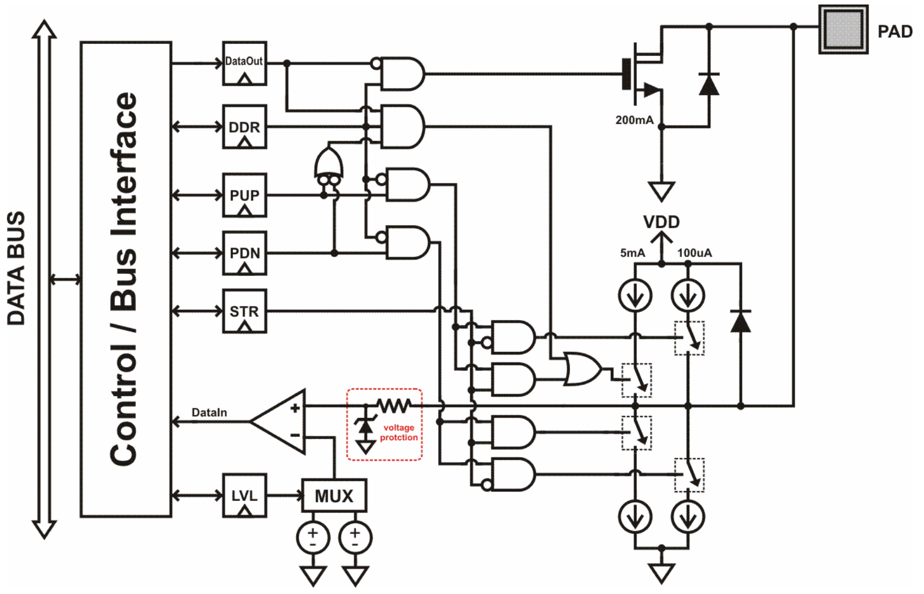


Figure 20 - Typical SIO Interface

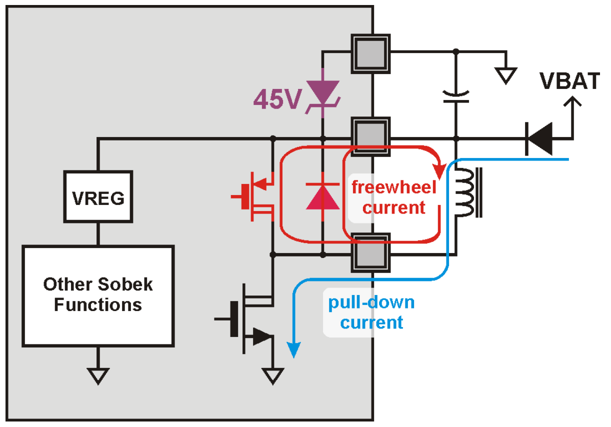


Figure 21: Freewheel Action

Figure 21 illustrates inductive freewheel operation.

Code Example: Configure Port C with the following characteristics:

- PWM2 in PC3, PWM1 in PC7 with N-MOS transistor (Active level = 1 = High)
- Interrupt mask in PC0 and PC1,
- Output enable for PC3, PC5 and PC7
- Low current in pull-up/down
- Pull-up in PC0 and PC1
- No pull downs
- Input threshold high for PC0 and PC1

```
Port_Config(PTC,(PC3_PWM2|PC7_PWM1|PC7_ACTL),0x03,0xA8,0x00,0x03,0x00,0x03);
NOTE: Function description:
/**
* @brief Port Configuration
*
* @param pt - port to be configured: PTA, PTB, PTC, PTD, PTE
* @param ss - auxiliar function enable. The designer must use defines:
* @param PORTA --> PA4_PWM2_SNS_EN, PA5_PWM1_SNS0_EN, PA6_PWM1_SNS1_EN, PA7_PWM2
* @param PORTB --> PB0_PWM1, PB1_PWM2, PB1_ACTL
* @param PORTC --> PC3_PWM2, PC5_PWM1, PC7_PWM1 ,PC7_ACTL
* @param imask - Interrupt enable mask, active High
* @param oe - output enable, active high
* @param st - Current capability (Strength) of pullup/down (0=100uA, 1=5mA)
* @param pu - pull-up enable, active high
* @param pd - pull-down enable, active high
* @param level - input threshold level. (0 = 1.65V, 1 = 4V)
*/
static __INLINE void Port_Config( uint8_t pt, uint8_t ss, uint8_t imask, uint8_t
oe, uint8_t st, uint8_t pu, uint8_t pd, uint8_t level )
```

8.11.1.3 GIO and SIO connection to ADC

All GIOs and SIOs are connected to the ADC input channel selector. The signals applied to these pins can either directly goes through the multiplexor or attenuated by factor of 8 and then goes through the multiplexor depending on the LVL bit.

8.11.1.4 LED

µSesame provides a pin specially designed to control a RED or BLUE LED simultaneously.

In the LED mode, additional high voltage current sources are activated providing up to ~45mA current. The current level is programmable. When the I/O pin is set to 1 with LEDEN bit enabled (**LEDIO**), regardless of the setting of GIO, LED current source is activated. When the I/O pin is reset to 0, it is deactivated.

The following diagram shows the recommended configuration of LEDs:

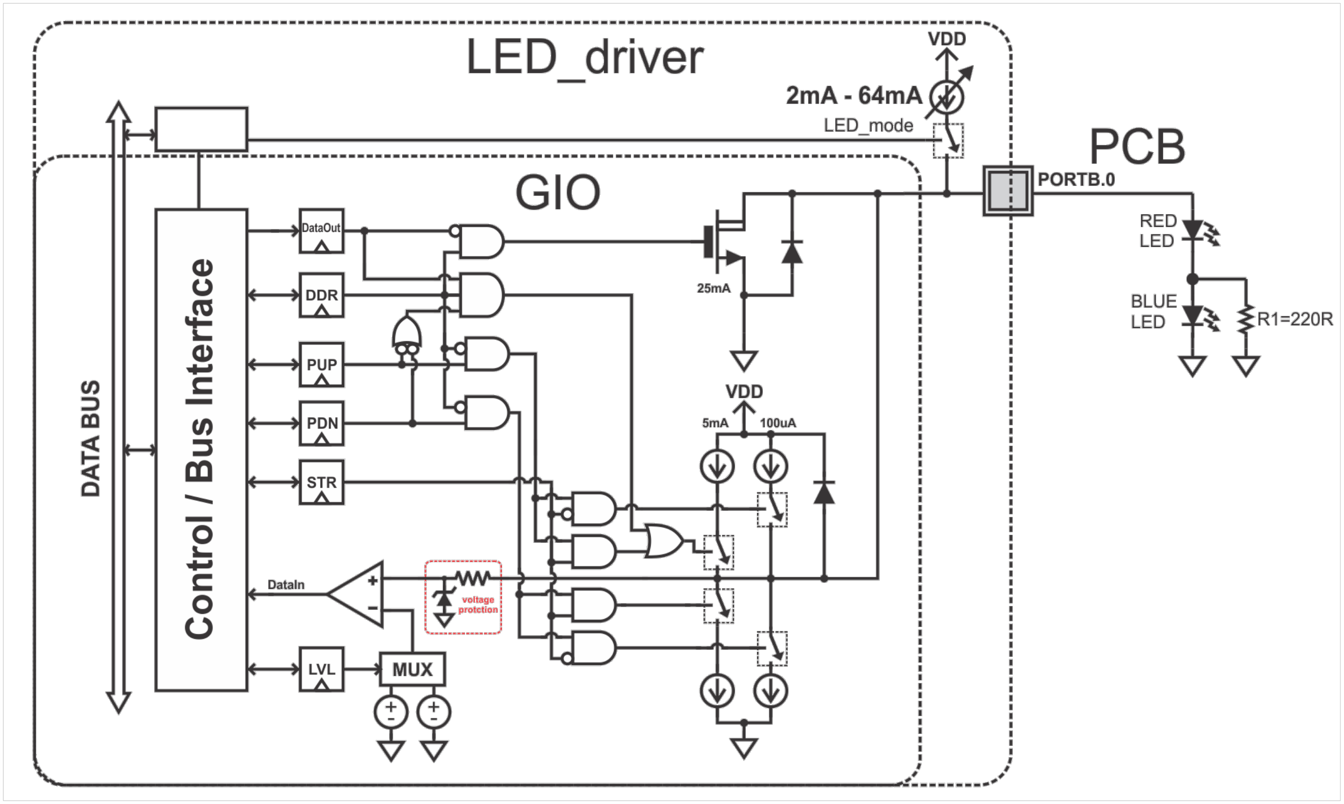


Figure 22 – LED pin Block Diagram

The nominal value of the resistor, R1 is 220 Ω. In this case, the nominal current level is around 20mA for the red LED whereas it is 6.5mA for the blue LED. This output current can be programmed in the range of 0mA up to 45mA in 3mA steps. With these programmable current settings and the resistor R1 the brightness of each LED can independently be adjusted.

Code Example: The following function calls select and enable the LED with 22mA:

```
LED_Current (22);        //Select 22mA current (from 0mA up to 45mA)
LED_Control (LEDEN);     //Enable LED
```

8.11.2 GPIO Registers

The following registers control the behavior of the GPIO pins:

PORTA: Port A input and output register. When a bit is selected as output the corresponding pin will reflect the register's bit. When a bit is selected as input the bit will reflect the condition of the pin.

PORTA		0x50000060			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
MSB							LSB
Bit7-0	**PA[7:0]**: Port A register bits. 0 = Pin state is '0' 1 = Pin state is '1'						

PORTB: Port B input and output register. When a bit is selected as output the corresponding pin will reflect the register's bit. When a bit is selected as input the bit will reflect the condition of the pin.

PORTB		0x50000061			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
MSB							LSB
Bit7-0	**PB[7:0]**: Port B register bits. 0 = Pin state is '0' 1 = Pin state is '1'						

PORTC: Port C input and output register. When a bit is selected as output the corresponding pin will reflect the register's bit. When a bit is selected as input the bit will reflect the condition of the pin.

PORTC		0x50000062			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
MSB							LSB
Bit7-0	**PC[7:0]**: Port C register bits. 0 = Pin state is '0' 1 = Pin state is '1'						

PORTD: Port D input and output register. When a bit is selected as output the corresponding pin will reflect the register's bit. When a bit is selected as input the bit will reflect the condition of the pin.

PORTD		0x50000063			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
MSB							LSB
Bit7-0	**PD[7:0]**: Port D register bits. 0 = Pin state is '0' 1 = Pin state is '1'						

PORTE: Port E input and output register. When a bit is selected as output the corresponding pin will reflect the register's bit. When a bit is selected as input the bit will reflect the condition of the pin.

PORTE		0x50000064			0x00		
Reserved	Reserved	Reserved	Reserved	R/W	R/W	R/W	R/W
-	-	-	-	PE3	PE2	PE1	PE0
MSB							LSB
Bit3-0	**PE[3:0]**: Port E register bits. 0 = Pin state is '0' 1 = Pin state is '1'						

PORTDOE: Port D output enable register.

PORTDOE		0x50000065			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PDOE7	PDOE6	PDOE5	PDOE4	PDOE3	PDOE2	PDOE1	PDOE0
MSB							LSB

Bit7-0 **PDOE[7:0]**: Port D output enable bits.
0 = Pin is input
1 = Pin is output

PORTEOE: Port E output enable and Mode configuration register

PORTEOE		0x50000066			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PEOE3	PEOE2	PEOE1	PEOE0	MDSPI	MDI2C	MDUART	MDLIN
MSB							LSB

Bit0 **MDLIN**: LIN mode enable
0 = GPIO
1 = LIN mode

Bit1 **MDUART**: UART mode enable
0 = GPIO
1 = UART mode

Bit2 **MDI2C**: I2C mode enable
0 = GPIO
1 = I2C mode

Bit3 **MDSPI**: SPI mode enable
0 = GPIO
1 = SPI mode

Bit7-4 **PEOE[3:0]**: Port E output enable bits.
0 = Pin is input
1 = Pin is output

PCONF: I2C, LIN and UART configuration register.

PCONF		0x50000067			0x00		
Reserved	Reserved	Reserved	R/W	R/W	R/W	R/W	R/W
I2C_RT[1]	I2C_RT[0]	-	LTREN	LPSWAP	UPSWAP	LTXPOL	UTXPOL
MSB							LSB

Bit7-6	**I2C_RT[1:0]**: I2C Resistor Trim
	00 = Open
	01 = 1k
	10 = 10k
	11 = 100k
Bit4	**LTREN**: Lin transmission enable bit
	0 = Transmission disabled
	1 = Transmission enabled
Bit3	**LPSWAP**: LIN Pins Swap Bit.
	0 = Pins are not swapped (TX=PD[7] and RX = PD[6])
	1 = Pins are swapped (TX=PD[6] and RX = PD[7])
Bit2	**UPSWAP**: UART Pins Swap Bit.
	0 = Pins are not swapped (TX=PD[5] and RX = PD[4])
	1 = Pins are swapped (TX=PD[4] and RX = PD[5])
Bit1	**LTXPOL**: LIN signals polarity.
	0 = normal polarity
	1 = inverted polarity
Bit0	**UTXPOL**: UART signals polarity.
	0 = normal polarity
	1 = inverted polarity

SFDICFGn: PAn configuration register. (n = 0, 1, 2, 3)

SFDICFGn		0x50000068/9/A/B			0x00		
Reserved	Reserved	R/W	R/W	R/W	R/W	R/W	R/W
-	-	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit5 **INTE**: Pin Change Interrupt enable bit
0 = Interrupt disabled
1 = Interrupt enabled

Bit4 **DD**: Data Direction
0 = Input
1 = Output

Bit3 **STR**: Pull Up/Down Strength Control
0 = Low Strength (100uA)
1 = High Strength (5mA)

Bit2 **PUP**: Pull up enable.
0 = Disabled
1 = Enabled

Bit1 **PDN**: Pull Down enable.
0 = Disabled
1 = Enabled

Bit0 **LVL**: Input Threshold level
0 = Low Threshold
1 = High Threshold

PA4CFG: PA4 configuration register.

PA4CFG		0x5000006C			0x00		
Reserved	R/W	R/W	R/W	R/W	R/W	R/W	R/W
-	PWM2_SNS	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit6 **PWM2_SNS**: PWM2 Short Circuit Sensor
0 = Sensor Disabled
1 = Sensor Enabled

Bit5 **INTE**: Interrupt not available

Bit4 **DD**: Data Direction
0 = Input
1 = Output

Bit3 **STR**: Pull Up/Down Strength Control
0 = Low Strength (100uA)
1 = High Strength (5mA)

Bit2 **PUP**: Pull up enable.
0 = Disabled
1 = Enabled

Bit1 **PDN**: Pull Down enable.
0 = Disabled
1 = Enabled

Bit0 **LVL**: Input Threshold level
0 = Low Threshold
1 = High Threshold

PA5CFG: PA5 configuration register.

PA5CFG		0x5000006D			0x00		
Reserved	R/W	R/W	R/W	R/W	R/W	R/W	R/W
-	PWM1_SNS0	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit6 **PWM1_SNS0**: PWM1 Short Circuit Sensor0

0 = Sensor Disabled

1 = Sensor Enabled

Bit5 **INTE**: Interrupt not available

Bit4 **DD**: Data Direction

0 = Input

1 = Output

Bit3 **STR**: Pull Up/Down Strength Control

0 = Low Strength (100uA)

1 = High Strength (5mA)

Bit2 **PUP**: Pull up enable.

0 = Disabled

1 = Enabled

Bit1 **PDN**: Pull Down enable.

0 = Disabled

1 = Enabled

Bit0 **LVL**: Input Threshold level

0 = Low Threshold

1 = High Threshold

PA6CFG: PA6 configuration register.

PA6CFG		0x5000006E			0x00		
Reserved	R/W	R/W	R/W	R/W	R/W	R/W	R/W
-	PWM1_SNS1	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit6 **PWM1_SNS0**: PWM1 Short Circuit Sensor1
0 = Sensor Disabled
1 = Sensor Enabled

Bit5 **INTE**: Interrupt not available

Bit4 **DD**: Data Direction
0 = Input
1 = Output

Bit3 **STR**: Pull Up/Down Strength Control
0 = Low Strength (100uA)
1 = High Strength (5mA)

Bit2 **PUP**: Pull up enable.
0 = Disabled
1 = Enabled

Bit1 **PDN**: Pull Down enable.
0 = Disabled
1 = Enabled

Bit0 **LVL**: Input Threshold level
0 = Low Threshold
1 = High Threshold

PA7CFG: PA7 configuration register.

PA7CFG		0x5000006F			0x00		
Reserved	R/W	R/W	R/W	R/W	R/W	R/W	R/W
-	PWM2_OUT	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit6 **PWM2_OUT**: PWM2 Output Enable
0 = PWM output disabled
1 = PWM output enabled

Bit5 **INTE**: Interrupt not available

Bit4 **DD**: Data Direction
0 = Input
1 = Output

Bit3 **STR**: Pull Up/Down Strength Control
0 = Low Strength (100uA)
1 = High Strength (5mA)

Bit2 **PUP**: Pull up enable.
0 = Disabled
1 = Enabled

Bit1 **PDN**: Pull Down enable.
0 = Disabled
1 = Enabled

Bit0 **LVL**: Input Threshold level
0 = Low Threshold
1 = High Threshold

PB0CFG: PB0 configuration register.

PB0CFG		0x50000070			0x00		
Reserved	R/W	R/W	R/W	R/W	R/W	R/W	R/W
-	PWM1_OUT	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit6	**PWM1_OUT**: PWM1 Output Enable
	0 = PWM output disabled
	1 = PWM output enabled
Bit5	**INTE**: Pin Change Interrupt enable bit
	0 = Interrupt disabled
	1 = Interrupt enabled
Bit4	**DD**: Data Direction
	0 = Input
	1 = Output
Bit3	**STR**: Pull Up/Down Strength Control
	0 = Low Strength (100uA)
	1 = High Strength (5mA)
Bit2	**PUP**: Pull up enable.
	0 = Disabled
	1 = Enabled
Bit1	**PDN**: Pull Down enable.
	0 = Disabled
	1 = Enabled
Bit0	**LVL**: Input Threshold level
	0 = Low Threshold
	1 = High Threshold

PB1CFG: PB1 configuration register.

PB1CFG		0x50000071			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ACTL	PWM2_OUT	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit7 **ACTL**: Active Level of Output
0 = Active Low (PMOS)
1 = Active High (NMOS)

Bit6 **PWM2_OUT**: PWM2 Output Enable
0 = PWM output disabled
1 = PWM output enabled

Bit5 **INTE**: Interrupt not available

Bit4 **DD**: Data Direction
0 = Input
1 = Output

Bit3 **STR**: Pull Up/Down Strength Control
0 = Low Strength (100uA)
1 = High Strength (5mA)

Bit2 **PUP**: Pull up enable.
0 = Disabled
1 = Enabled

Bit1 **PDN**: Pull Down enable.
0 = Disabled
1 = Enabled

Bit0 **LVL**: Input Threshold level
0 = Low Threshold
1 = High Threshold

PBnCFG: PBn configuration register. (n = 2, 3, 4)

PBnCFG		0x50000072/3/4			0x00		
Reserved	Reserved	R/W	R/W	R/W	R/W	R/W	R/W
-	-	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit5 **INTE**: Interrupt not available

Bit4 **DD**: Data Direction
0 = Input
1 = Output

Bit3 **STR**: Pull Up/Down Strength Control
0 = Low Strength (100uA)
1 = High Strength (5mA)

Bit2 **PUP**: Pull up enable.
0 = Disabled
1 = Enabled

Bit1 **PDN**: Pull Down enable.
0 = Disabled
1 = Enabled

Bit0 **LVL**: Input Threshold level
0 = Low Threshold
1 = High Threshold

PBmCFG: PBm configuration register. (m = 5, 6, 7)

PBmCFG		0x50000072/3/4			0x00		
Reserved	Reserved	W	W	W	W	W	W
-	-	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit5 **INTE**: Pin Change Interrupt enable bit
0 = Interrupt disabled
1 = Interrupt enabled

Bit4 **DD**: Data Direction
0 = Input
1 = Output

Bit3 **STR**: Pull Up/Down Strength Control
0 = Low Strength (100uA)
1 = High Strength (5mA)

Bit2 **PUP**: Pull up enable.
0 = Disabled
1 = Enabled

Bit1 **PDN**: Pull Down enable.
0 = Disabled
1 = Enabled

Bit0 **LVL**: Input Threshold level
0 = Low Threshold
1 = High Threshold

PCnCFG: PCn configuration register. (n = 0, 1, 2)

PCnCFG		0x50000078/9/A			0x00		
Reserved	Reserved	R/W	R/W	R/W	R/W	R/W	R/W
-	-	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit5 **INTE**: Pin Change Interrupt enable bit
0 = Interrupt disabled
1 = Interrupt enabled

Bit4 **DD**: Data Direction
0 = Input
1 = Output

Bit3 **STR**: Pull Up/Down Strength Control
0 = Low Strength (100uA)
1 = High Strength (5mA)

Bit2 **PUP**: Pull up enable.
0 = Disabled
1 = Enabled

Bit1 **PDN**: Pull Down enable.
0 = Disabled
1 = Enabled

Bit0 **LVL**: Input Threshold level
0 = Low Threshold
1 = High Threshold

PC3CFG: PC3 configuration register.

PC3CFG		0x5000007B			0x00		
Reserved	R/W	R/W	R/W	R/W	R/W	R/W	R/W
-	PWM2_OUT	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit6 **PWM2_OUT**: PWM2 Output Enable
0 = PWM output disabled
1 = PWM output enabled

Bit5 **INTE**: Pin Change Interrupt enable bit
0 = Interrupt disabled
1 = Interrupt enabled

Bit4 **DD**: Data Direction
0 = Input
1 = Output

Bit3 **STR**: Pull Up/Down Strength Control
0 = Low Strength (100uA)
1 = High Strength (5mA)

Bit2 **PUP**: Pull up enable.
0 = Disabled
1 = Enabled

Bit1 **PDN**: Pull Down enable.
0 = Disabled
1 = Enabled

Bit0 **LVL**: Input Threshold level
0 = Low Threshold
1 = High Threshold

PC4CFG: PC4 configuration register.

PB4CFG		0x5000007C			0x00		
Reserved	Reserved	R/W	R/W	R/W	R/W	R/W	R/W
-	-	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit5 **INTE**: Interrupt not available

Bit4 **DD**: Data Direction
0 = Input
1 = Output

Bit3 **STR**: Pull Up/Down Strength Control
0 = Low Strength (100uA)
1 = High Strength (5mA)

Bit2 **PUP**: Pull up enable.
0 = Disabled
1 = Enabled

Bit1 **PDN**: Pull Down enable.
0 = Disabled
1 = Enabled

Bit0 **LVL**: Input Threshold level
0 = Low Threshold
1 = High Threshold

PC5CFG: PC5 configuration register.

PC5CFG		0x5000007D			0x00		
Reserved	R/W	R/W	R/W	R/W	R/W	R/W	R/W
-	PWM1_OUT	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit6 **PWM1_OUT**: PWM1 Output Enable
0 = PWM output disabled
1 = PWM output enabled

Bit5 **INTE**: Interrupt not available

Bit4 **DD**: Data Direction
0 = Input
1 = Output

Bit3 **STR**: Pull Up/Down Strength Control
0 = Low Strength (100uA)
1 = High Strength (5mA)

Bit2 **PUP**: Pull up enable.
0 = Disabled
1 = Enabled

Bit1 **PDN**: Pull Down enable.
0 = Disabled
1 = Enabled

Bit0 **LVL**: Input Threshold level
0 = Low Threshold
1 = High Threshold

PC6CFG: PC6 configuration register.

PC6CFG		0x5000007E			0x00		
Reserved	Reserved	R/W	R/W	R/W	R/W	R/W	R/W
-	-	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit5 **INTE**: Interrupt not available

Bit4 **DD**: Data Direction
0 = Input
1 = Output

Bit3 **STR**: Pull Up/Down Strength Control
0 = Low Strength (100uA)
1 = High Strength (5mA)

Bit2 **PUP**: Pull up enable.
0 = Disabled
1 = Enabled

Bit1 **PDN**: Pull Down enable.
0 = Disabled
1 = Enabled

Bit0 **LVL**: Input Threshold level
0 = Low Threshold
1 = High Threshold

PC7CFG: PC7 configuration register.

PC7CFG		0x500007F			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ACTL	PWM1_OUT	INTE	DD	STR	PUP	PDN	LVL
MSB							LSB

Bit7 **ACTL**: Active Level of Output
0 = Active Low (PMOS)
1 = Active High (NMOS)

Bit6 **PWM1_OUT**: PWM1 Output Enable
0 = PWM output disabled
1 = PWM output enabled

Bit5 **INTE**: Interrupt not available

Bit4 **DD**: Data Direction
0 = Input
1 = Output

Bit3 **STR**: Pull Up/Down Strength Control
0 = Low Strength (100uA)
1 = High Strength (5mA)

Bit2 **PUP**: Pull up enable.
0 = Disabled
1 = Enabled

Bit1 **PDN**: Pull Down enable.
0 = Disabled
1 = Enabled

Bit0 **LVL**: Input Threshold level
0 = Low Threshold
1 = High Threshold

LEDIO: LED output configuration register.

LEDIO		0x50000059			0x00		
Reserved	Reserved	R/W	R/W	R/W	R/W	R/W	Reserved
-	-	LEDEN	LEDCUR3	LEDCUR2	LEDCUR1	LEDCUR0	-
MSB							LSB

Bit5 **LEDEN**: LED enable

0 = LED output disabled

1 = LED output enabled

Bit4-1 **LEDCUR[3:0]**: LED Current Control

0000 = 0 mA

0001 = 3 mA

0010 = 6 mA and so on

The equation for this current is: I= 3mA x LEDCUR

8.12 SHORT CIRCUIT PROTECTION CIRCUITS

µSesame provides support for Fuse Elimination and short circuit detection. The following sections will describe both functionalities in detail

8.12.1 Fuse Elimination Usage Description

Once enabled the fuse elimination circuit operates by automatically comparing the voltage between two input pins (PB6 and PB7). If the voltage difference between these pins exceeds a programmable voltage (Fuse Offset Voltage), a flag is set, allowing the program to detect the condition and act accordingly.

To use the fuse elimination feature the following setps must be taken:

1. Select the fuse offset value.
2. Enable the fuse detection
3. Execute a polling on the fuse detect flag at a suitable rate

8.12.2 Short Circuit Protection Usage Description

µSesame provides a short-circuit protection for the PWM1 and PWM2 when the PB1(PWM2) and PC6(PWM1) are used. This circuit operates by comparing a feedback input voltage with a programmable threshold value (using the ADC to measure this voltage) and automatically disabling the corresponding PWM (to a programmable safe state) in case of short-circuit detection.

The following sequence must be followed in order to protect against short-circuits in the PWM outputs:

1. Configure ADC settings
2. Select threshold and period
3. Enable ADC
4. Select appropriate sense level
5. Select a safe state
6. Enable Short Circuit Sense

Example: Let's enable the protection of the PWM2(PB1) using PA4 as feedback input operating the Siren.

Some comments:

a. The active level is LOW.
b. The Sense level is High.
c. The PWM output is to be inverted

With this information we can start:

```
/*Config Port B, PWM2 in PB1, with P-MOS transistor (Active level = 0 = Low),
PB1 as output, Interrupt mask disabled, low current in pull-up/down, pull-ups
disabled, pull-downs disabled*/

Port_Config( PTB, (PB1_PWM2), 0x00, 0x02, 0x00, 0x00, 0x00, 0x00);

/* Note: If a N-MOS was used it would be necessary to change the auxiliary
function parameter to (PB1_PWM2|PB1_ACTL). */

/*Config Port A, PWM2 sensor in PA4, no outputs, interrupt mask disabled, low
current in pull-up/down, pull-ups disabled, pull-downs disabled, high threshold
levels in PB4 to allow for the ADC to measure voltages up to ~28V*/

Port_Config( PTA, (PA4_PWM2_SNS_EN), 0x00, 0x00, 0x00, 0x03, 0x00, 0x10 );

ADC_Enable(ADCEN);       //ADCs enabled

ADC_ClkDiv (16);         //ADC clock divider 1:16

/*ADC 2 with a threshold of 0x80 (half scale) and 20*13 clock cycles delay
period to start conversion.*/

ADC_ProgramShortCircuit (ADC2, 0x80, 20);

*PWMFUSE |= PWM2_SNS_LVL1;    //PWM2 sense level 1
```

8.12.3 Short Circuit Protected Related Registers

ADC1THRESH: ADC1 threshold value used to detect short circuit when the ADC is used to detect it.

ADC1THRESH		0x50000050			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADC1TH_7	ADC1TH_6	ADC1TH_5	ADC1TH_4	ADC1TH_3	ADC1TH_2	ADC1TH_1	ADC1TH_0
MSB							LSB

ADC1PERIOD: ADC1 period (in conversion times) used to define the pooling of an input being tested for short circuit.

ADC1PERIOD		0x50000051			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADC1PR_7	ADC1PR_6	ADC1PR_5	ADC1PR_4	ADC1PR_3	ADC1PR_2	ADC1PR_1	ADC1PR_0
MSB							LSB

ADC2THRESH: ADC2 threshold value used to detect short circuit when the ADC is used to detect it.

ADC2THRESH		0x50000052			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADC2TH_7	ADC2TH_6	ADC2TH_5	ADC2TH_4	ADC2TH_3	ADC2TH_2	ADC2TH_1	ADC2TH_0
MSB							LSB

ADC2PERIOD: ADC2 period (in conversion times) used to define the pooling of an input being tested for short circuit.

ADC2PERIOD		0x50000053			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADC2PR_7	ADC2PR_6	ADC2PR_5	ADC2PR_4	ADC2PR_3	ADC2PR_2	ADC2PR_1	ADC2PR_0
MSB							LSB

PWMFUSE: Fuse and Short Circuit Control Register

ADC1THRESHOLD		0x50000058			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PWM1SCF	PWM2SCF	FUSEFLG	FUSEEN	PWM1SNSL	PWM2SNSL	FUSEOFF1	FUSEOFF0
MSB							LSB

Bit7	**PWM1SCF**: PWM1 Short Circuit Detection Flag
	0 = No short circuit detected
	1 = Short circuit detected
Bit6	**PWM2SCF**: PWM2 Short Circuit Detection Flag
	0 = No short circuit detected
	1 = Short circuit detected
Bit5	**FUSEFLG**: Fuse Detection Flag
	0 = Overcurrent not detected
	1 = Overcurrent detected
Bit4	**FUSEEN**: Fuse Enable Signal
	0 = Disable fuse elimination circuit
	1 = Enable fuse elimination circuit
Bit3	**PWM1SNSL**: PWM1 Sense Level
	0 = Logic low level expected for normal operation
	1 = Logic high level expected for normal operation
Bit2	**PWM2SNSL**: PWM2 Sense Level
	0 = Logic low level expected for normal operation
	1 = Logic high level expected for normal operation
Bit1-0	**FUSEOFF[1:0]**: Fuse offset level
	00 = 190 mV
	01 = 360 mV
	10 = 670 mV
	11 = 860 mV

8.13 CLOCK SOURCES

μSesame provides three clock sources:

- Internal auxiliary oscillator running at 10kHz. (This oscillator is always running, even when the device is in sleep mode, but its power consumption is negligible)
- Internal RC oscillator running at 10MHz.
- Crystal oscillator (Typically 3.579545MHz)

μSesame starts from power-on reset using the internal auxiliary oscillator. From this point on the user may select the crystal or the RC oscillators.

The crystal oscillator is required for RF reception and to operate the ultrasound interface. The 10MHz RC oscillator may be used if a higher execution speed, albeit with lower frequency accuracy, is necessary for the application. The 10kHz oscillator may be used in power saving modes.

8.13.1 Clock Sources Characteristics

The following table defines the main characteristics of the clock sources:

Table 21 - Clock Performance Specification, recommended operating conditions unless otherwise specified

name	conditions	min	typ	max	unit
Crystal Oscillator frequency			3.579545		MHz
Frequency stability	Using defined crystal			TBD	ppm
Auxiliary Oscillator	(Calibrated Frequency)		10		kHz
Auxiliary Oscillator accuracy	Post-calibration to 10KHz, T_A=27°C			5	%
RC Oscillator frequency			10		MHz
RC Oscillator accuracy	Post-calibration to 10MHz, T_A=27°C			1	%

8.13.2 Clock Related Registers

The following registers are used to control the behavior of the clock sources:

PMUCLK: Processor Control register.

PMUCLK		0x50000000			0x15		
R/W	R/W	Reserved	R	R/W	R/W	R/W	R/W
CKD1	CKD0	-	RCMON	XO_CK_ENB	RC_CK_ENB	CKSEL1	CKSEL0
MSB							LSB

Bit7-6	**CKD[1:0]**: Clock Frequency Divider
	00 = Clock Divided by 1
	01 = Clock Divided by 2
	10 = Clock Divided by 4
	11 = Clock Divided by 8
Bit4	**RCMON**: RC Oscillator Monitor
	0 = RC Oscillator Inactive
	1 = RC Oscillator Active
Bit3	**XO_CK_ENB**: Crystal Oscillator Control
	0 = Crystal Oscillator Disable
	1 = Crystal Oscillator Enable
Bit2	**RC_CK_ENB**: RC Oscillator Control
	0 = RC Oscillator Disable
	1 = RC Oscillator Enable
Bit1-0	**CKSEL[1:0]**: Clock Select
	00 = 10 KHz Auxiliary Clock
	01 = 10MHz RC Oscillator Clock*
	10 = Crystal Oscillator Clock
	11 = Not used

*Note: This is the clock selected after Power-On-Reset and for clock fault condtion

8.13.3 Clock Sources Usage Description

Upon Reset or Power-On Reset the system starts using the internal RC 10MHz oscillator.

Depending on the application requirements the designer can:

- Enable or disable the internal RC oscillator
- Enable or disable the external crystal
- Select the system clock source: RC or Crystal
- Enable the clock monitor interrupt to detect and process eventual failures in either the crystal or RC clock sources

Some peripherals require the crystal clock in order to operate properly:

Table 22 - Peripherals with specific clock source requirements

Peripheral	Clock Required	Comments
RF Receiver	Crystal (@3.579545MHz)	
Ultrasound	Crystal (@3.579545MHz)	
UART	Crystal (@3.579545MHz) or 10MHz RC Oscillator	
LIN	Crystal (@3.579545MHz)	Required in master mode only. In slave mode it can use the RC oscillator (10MHz).

Example Code: Enable the Crystal oscillator. Wait for it to be stable and then selects it. Also enables the clock monitor interrupt and create an interrupt handler routine for it.

```
CLK_CrystalControl( XTON );   //Enable Crystal Clock
for ( i = 0; i < 20000; i++); //Some delay to allow for crystal to start
CLK_SelectClockSource( XTCLOCK );     //Selects crystal as clock source
NVIC_EnableIRQ(BrownOut_IRQn);        //Enable the clock monitor interrupt
```

8.13.4 Power Management Unit (PMU)

µSesame implements a power management unit. Its main characteristics are:

- HW reset - Affects all aspects of µSesame
- SW reset - Does not affect clock nor brownout setup
- Selectable Sleep mode and Halt Mode
- Programmable brownout detector

8.13.5 PMU Registers

µSesame implements the following registers:

PMURST: Processor control register.

PMURST		0x50000001			0x01		
W	W	R/W	Reserved	Reserved	Reserved	R	R/W
HWRST	SWRST	DLEEP	-	-	-	BROUT	PORF
MSB							LSB

Bit7 **HWRST**: Hardware reset
0 = Idle
1 = Hardware reset (automatically cleared after reset process completed)

Bit6 **SWRST**: Software reset
0 = Idle
1 = Software reset (automatically cleared after reset process completed)

Bit5 **DLEEP**: Deep sleep (HALT) mode
Writing:
0 = Clear deep sleep flag / 1 = Put the system in deep sleep - Halt
Reading:
0 = Flag cleared / 1 = system in deep sleep mode

Bit1 **BROUT**: Brownout indicator
0 = No brownout / 1 = Brownout

Bit0 **PORF**: Power-On Reset flag
Writing: 0 = Clear POR / 1 = No effect
Reading: 0 = POR flag already cleared by application
1 = The system just came out of POR or HW Reset

PMUBOR: BOR Control.

PMUBOR		0x50000002			0x00		
R/W	Reserved	Reserved	Reserved	R/W	R/W	R/W	R/W
BOREN	-	-	-	BORRST	BORINT	BOUTVALUE1	BOUTVALUE0
MSB							LSB

Bit7 **BOREN**: Brownout enable
0 = Brownout disabled (for B2 parts) / enabled (for B3 parts)
1 = Brownout enabled (for B2 parts) / disabled (for B3 parts)

Bit3 **BORRST**: Brownout Reset enable
0 = Disable Brownout based reset (for B2 parts) / enable (for B3 parts)
1 = Enable Brownout based reset (for B2 parts) / disable (for B3 parts)

Bit2 **BORINT**: Brownout interrupt
0 = Brownout interrupt disabled
1 = Brownout interrupt enabled

Bit1-0 **BOUTVALUE [1:0]**: Brownout threshold value
00 = 2.0V
01 = 2.2V
10 = 2.4V
11 = 2.6V

PMUVREG1: VREG Control1.

PMUBOR		0x5001800D			0x04		
R/W	Reserved	Reserved	R/W	R/W	R/W	R/W	R/W
QPENVDD3IO	-	-	RCAL1	RCAL0	XTALBIAS2	XTALBIAS1	XTALBIAS0
MSB							LSB

Bit7 **QPENVDD3IO**: Charge Pump Enable for 3.3V IO Supply
0 = disabled
1 = enabled

PMUVREG2: VREG Control2.

PMUBOR		0x5001800F			0x00		
Reseved	R/W	R/W	R/W	R/W	R/W	R/W	R/W
-	ADCCYC2	ADCCYC1	ADCCYC0	QPENVDD3 ANA	QPENVDD FLA	QPENVDD MCU	QPENVDD3 DIG
MSB							LSB

Bit3	**QPENVDD3ANA**: Charge Pump Enable for 3.3V Analog Supply
	0 = disabled
	1 = enabled
Bit2	**QPENVDDFLA**: Charge Pump Enable for 2.6V Flash Supply
	0 = disabled
	1 = enabled
Bit1	**QPENVDDMCU**: Charge Pump Enable for 1.8V MCU Supply
	0 = disabled
	1 = enabled
Bit0	**QPENVDD3DIG**: Charge Pump Enable for 3.3V Digital Supply
	0 = disabled
	1 = enabled

8.13.6 PMU Usage Description

The PMU module allows for the control of reset, deep sleep (halt), sleep, and brownout.

8.13.6.1 PMU control of Reset:

There are two forms of reset that can be issued:

- Hardware reset: In this reset all peripherals are reset, the 10 KHz clock is selected and all other clock sources are disabled, but the brownout selection is kept.
- Software reset: In this reset all peripherals are reset but the clock setup is kept unchanged along with the brownout selection.

Code Examples: HW reset and SW reset:

```
PMU_HwReset();

PMU_SwReset();
```

8.13.6.2 PMU control of sleep and deep sleep (halt) modes:

The PMU can set the system into sleep or deep sleep (halt) modes.

In the deep sleep mode:

- the CPU is halted
- Any enabled clock source will continue to operate
- The three timers (Timer0, Timer1 and Timer2) and the SysTick Timer will stop operating
- All other peripherals will keep running (if enabled and fed by their required source clock)
- The system will leave the deep sleep (halt) mode only through a reset or POR (**P**ower-**O**n **R**eset). The sources of a reset can be the wakeup timer or any peripheral that generates an interrupt independently of the interrupt being enabled by the NVIC module.(**N**ested **V**ector **I**nterrupt **C**ontroller)

Note: For those peripherals that have in their registers a bit that locally enables the interrupt this register has to be enabled in order to reset the system. Example: For the GPIOs ports PORTA, PORTB and PORTC the INTE bits of the I/O pins selected to reset the part upon change must be set.

Code Example: Enabling the reset by enabling an interrupt from PORTA[0] upon change. (Port A, bit0)

```
//PortA.0 interrupt enabled, pull up enabled
Port_Config(PTA, 0, 0x01, 0x00, 0x00, 0x01, 0x00, 0x00);
PMU_Deep_Sleep(); //System in deep sleep (halt)
```

In the sleep mode:

1. the CPU is halted
2. Any enabled clock source will continue to operate
3. All timers (Timer0, Timer1 and Timer2) and the SysTick Timer will continue operating
4. All other peripherals will keep running if enabled and fed by their required source clock
5. Besides a POR and/or reset the system will leave the sleep mode also through an interrupt; the sources of an interrupt can be any peripheral generating an interrupt.

   Note: The interrupt must be enabled by the NVIC module.(**N**ested **V**ector **I**nterrupt **C**ontroller) and for those peripherals that have in their registers a bit that locally enables the interrupt this register also has to be enabled in order to generate an interrupt and wakeup the system from sleep.

   Example: For the GPIOs ports PORTA, PORTB and PORTC the INTE bits of the I/O pins selected to reset the part upon change must be set.

Code Example:

```
//PortA.0 interrupt enabled, pull up enabled
Port_Config(PTA, 0, 0x01, 0x00, 0x00, 0x01, 0x00, 0x00);
NVIC_EnableIRQ(PIN_IRQn);      //Pin change interrupt enabled in NVIC
PMU_Sleep();                   //Part in sleep mode
```

8.13.6.3 PMU control of Brownout:

The PMU controls the brownout. It entails:

- Enabling or disabling the brownout circuit
- Selecting the behavior when a brownout is detected:
    - Generate an interrupt
    - Reset the system
- Selecting the brownout voltage level

Code Example: Enable the BOR, with interrupt but no reset and with 2.4V level.

```
BOR_ResetControl(BORRSTDIS);  //Brownout reset disabled
BOR_IntControl(BORINTEN);     //Brownout interrupt enabled
BOR_Level(BOR24V);            //Brownout level = 2.4V
BOR_Control(BOREN);           //Brownout enabled
```

8.14 WAKE-UP TIMER

In addition to the Timer0/1/2 µSesame implements a timer capable of waking-up the microcontroller from a sleep state.

The wake up timer is a timer used to allow for recovery from deep sleep, including when the microcontroller is disconnected from its power supply.

The following register controls the wake-up timer:

WKPTIME: Wakeup timer control.

WKPTIME		0x50000004			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
MANT3	MANT2	MANT1	MANT0	EXP3	EXP2	EXP1	EXP0
MSB							LSB

Bit7-4 **MANT [3:0]**: Mantissa of the wakeup timer

Bit3-0 **EXP [3:0]**: Exponent of the wakeup timer (range: 0...12)

WakeupPeriod = Mantissa * $2^{(Exponent+1)}$ / SystemClock

For instance, a value of 0x54 would give a time of: (Assuming the application is running from the 10 kHz internal oscillator)

WakeupPeriod = 5 * $2^{(4+1)}$ / 10kHz = 16msec

Code Example: Enabling the wakeup timer according to the previous example.

```
WKP_Timing(5,4)           //Select the mantissa=5 and exponent=4
PMU_Deep_Sleep(SLEEPON);  //Put the part in deep sleep mode, it will
                          //Reset in 16msec.
```

9.0 REFERENCES

ISM Band references:

http://en.wikipedia.org/wiki/ISM_band

http://ecfr.gpoaccess.gov/cgi/t/text/text-idx?c=ecfr&sid=8a0249759d545fa2c9ea0840b08bb817&rgn=div5&view=text&node=47:1.0.1.1.14&idno=47)

10.0 REVISION HISTORY

Rev #	Date	Action	By
0.1	20 Jan 2011	Initial Draft	CG
0.5	7 June 2011	Fifth Draft	DDK
1.0	22 Jull 2014	Update PIN interrupt (latest AyDeeKay reference)	IA
2.0	14 Sep 2015	Indie version and reformat, remove PAN	CR

11.0 Contacts

United States
32 Journey
Aliso Viejo, California 92656, USA
Tel: +1 949-608-0854
sales@indiesemi.com

China
232 Room, Donghai Wanhao Plaza,
South Hi-tech 11th Road, Hi-tech Industry Park,
Nanshan District, Shenzhen, China.
Tel: +86 755-86116939

Scotland
MWB Business Exchange
9-10 St. Andrew Square
Edinburgh EH2 2AF, Scotland
Tel: +44 131 718 6378

http://www.indiesemi.com/

Important Notice