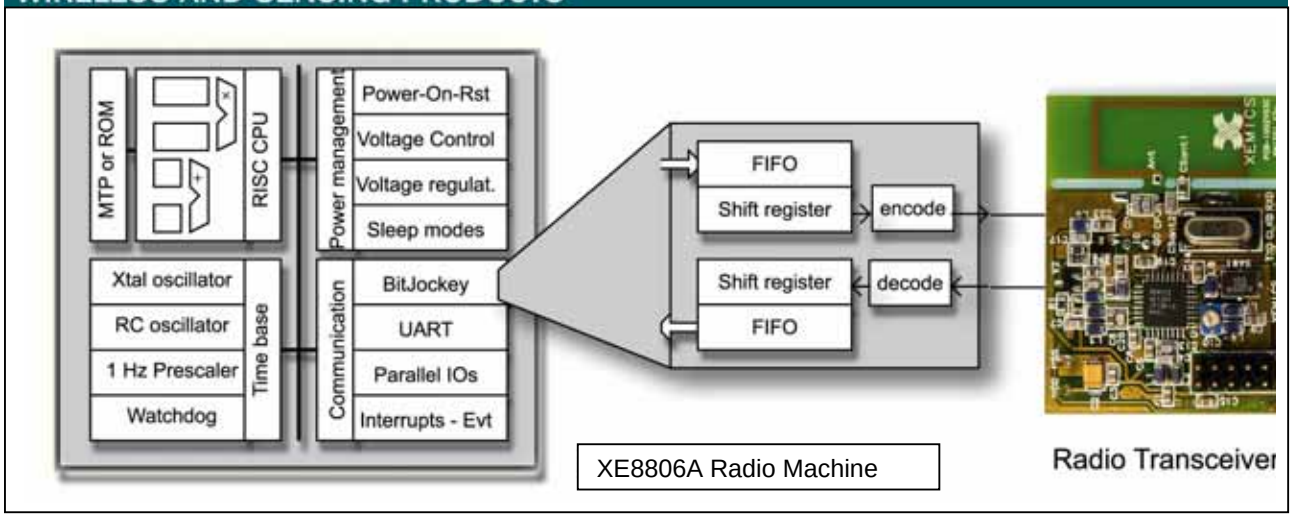


XE8806A and XE8807A

Ultra Low-Power Low-Voltage Radio Machines

General Description

The XE8806A and XE8807A are ultra low-power low-voltage microcontroller based Radio Machines. They include the revolutionary BitJockey™, UART type of peripheral specialized for radio communication.

The XE8806A and XE8807A are available with on chip ROM or Multiple-Time-Programmable (MTP) program memory.

Key product Features

- Ultra low-power MCU, up to 7 MIPS
- 300 uA at 1 MIPS operation
- 6 uA at 32 kHz operation
- 1 uA time keeping
- Low-voltage operation (1.2 - 5.5 V supply voltage)
- 22 kB (8 kW) ROM/MTP (XE8806A)
  11 kB (4kW) MTP (XE8807A)
- 520 B RAM
- 4 counters
- PWM, UART, BitJockey™
- Analog matrix switching
- 4 low-power analog comparators
- independant RC and crystal oscillators
- 5 reset, 15 interrupt, 8 event sources
- 100 years MTP Flash retention at 55°C

Applications

- RF companion chip
- RF system supervisor
- Portable, battery operated instruments
- Metering
- Remote control
- HVAC control

Ordering Information

Product	Temperature range	Memory type	Package
XE8806AMI000	-40°C to 85 °C	MTP	die
XE8806AMI026LF	-40°C to 85 °C	MTP	TQFP32
XE8806ARI000	-40°C to 125°C	ROM	die
XE8806ARI026LF	-40°C to 125°C	ROM	TQFP32
XE8807AMI000	-40°C to 85 °C	MTP	die
XE8807AMI026LF	-40°C to 85 °C	MTP	TQFP32

TABLE OF CONTENTS

1. General overview

1.1 Top schematic

The top level block schematic of the circuit is shown in Figure 1-1. The heart of the circuit consists of the Coolrisc816 CPU (central processing unit) core. This core includes an 8x8 multiplier and 16 internal registers.

The bus controller generates all control signals for access to all data registers other than the CPU internal registers.

The reset block generates the adequate reset signals for the rest of the circuit as a function of the set-up contained in its control registers. Possible reset sources are the power-on-reset (POR), the external pin NRESET, the watchdog (WD), a bus error detected by the bus controller or a programmable pattern on Port A.

The clock generation and power management block sets up the clock signals and generates internal supplies for different blocks. The clock can be generated from the RC oscillator (this is the start-up condition), the crystal oscillator (XTAL) or an external clock source (given on the XIN pin).

The test controller generates all set-up signals for different test modes. In normal operation, it is used as a set of 8 low power RAM. If power consumption is important for the application, the variables that need to be accessed frequently should be stored in these registers rather than in the RAM.

The IRQ handler routes the interrupt signals of the different peripherals to the IRQ inputs of the CPU core. It allows masking of the interrupt sources and it flags which interrupt source is active.

Events are generally used to restart the processor after a HALT period without jumping to a specified address, i.e. the program execution resumes with the instruction following the HALT instruction. The EVN handler routes the event signals of the different peripherals to the EVN inputs of the CPU core. It allows masking of the event sources and it flags which event source is active.

The Port B is an 8 bit parallel IO port with analog capabilities. The USRT, UART, PWM and CMPD blocks also make use of this port.

The instruction memory is a 22-bit wide flash or ROM memory depending on the circuit version. In case of the ROM version, the VPP pin is not used. The maximal number of instructions in the XE8806A is 8192. The maximal number of instructions in the XE8807A is 4096.

The data memory on this product is a 512 byte SRAM.

The port A is an 8 bit parallel input port. It can also generate interrupts, events or a reset. It can be used to input external clocks for the timer/counter/PWM block.

The Port D is a general purpose 8 bit parallel IO port.

The USRT (universal synchronous receiver/transmitter) contains some simple hardware functions in order to simplify the software implementation of a synchronous serial link.

The UART (universal asynchronous receiver/transmitter) contains a full hardware implementation of the asynchronous serial link.

The RFIF interface is a serial interface dedicated to communication with RF circuits. From the CPU side, it very much looks like an ordinary UART but it also implements low level coding/decoding and frame synchronisation. The input/output pins are multiplexed on port D.

The counters/timers/PWM can take its clocks from internal or external sources (on Port A) and can generate interrupts or events. The PWM is output on Port B.

The VLD (voltage level detector) detects the battery end of life with respect to a programmable threshold.

The CMPD contains a 4 channel comparator. It is intended to monitor analog or digital signals whilst having a very low power consumption.

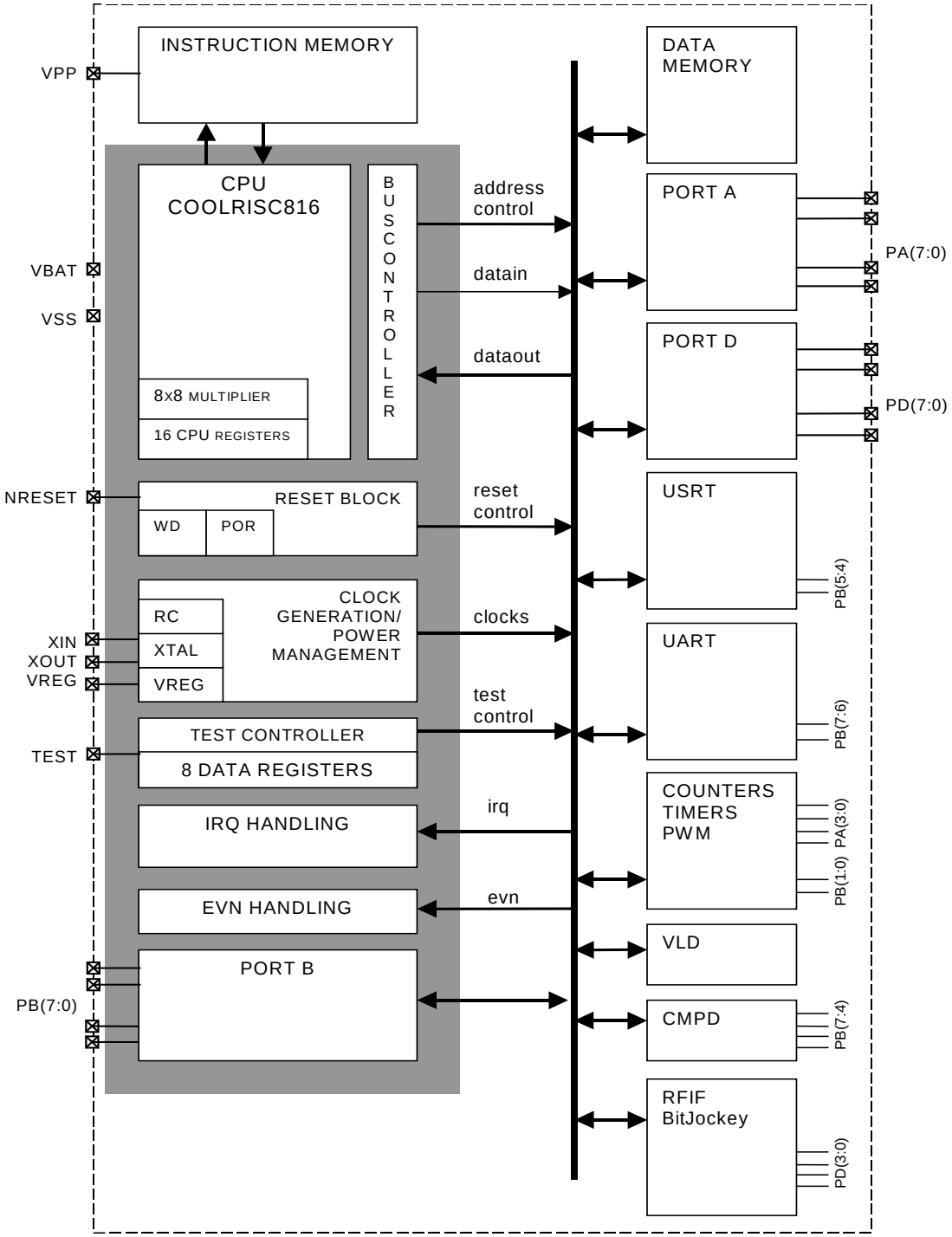


Figure 1-1. Block schematic of the XE8806A and XE8807A circuits.

1.2 Pin map

The XE8806A and XE8807A can be delivered in different packages. The pin maps for the different packages are given below.

1.2.1 TQFP-32

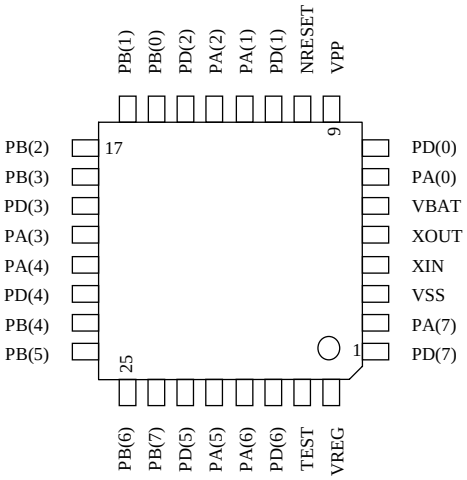


Figure 1-2. TQFP-32 pin map

1.2.2 SO-28

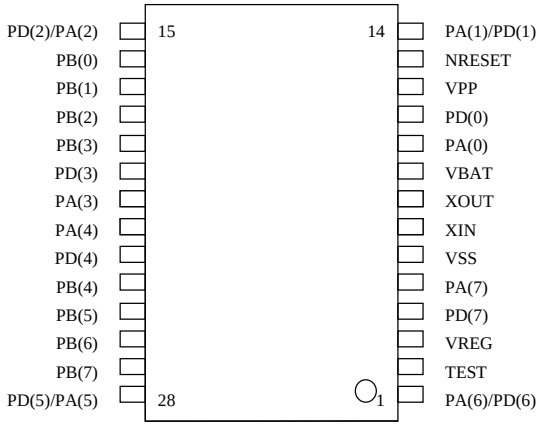


Figure 1-3. SO28 pin map

In the SO-28 package, 4 pins of Port A and Port D are connected together. It is up to the user to choose between the functionality of Port A or Port D for these pins.

Note: if one of the pins PD(1), PD(2), PD(5), PD(6) is used as output, the pull up of the corresponding pin of Port A should be disabled in order to have low power consumption.

1.2.3 SO-24

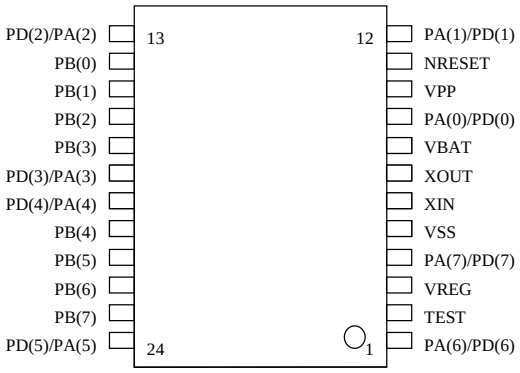


Figure 1-4. SO24 pin map

In the SO-24 package, all pins of Port A and Port D are connected together. It is up to the user to choose between the functionality of Port A or Port D.

Note: if one of the pins of Port D is used as output, the pull up of the corresponding pin of Port A should be disabled in order to have low power consumption.

1.2.4 Bare die XE8806A

The circuit is also available in bare die for chip on board assembly. All VBAT pins and all VSS pins should be connected together. The substrate of the circuit is connected to VSS.

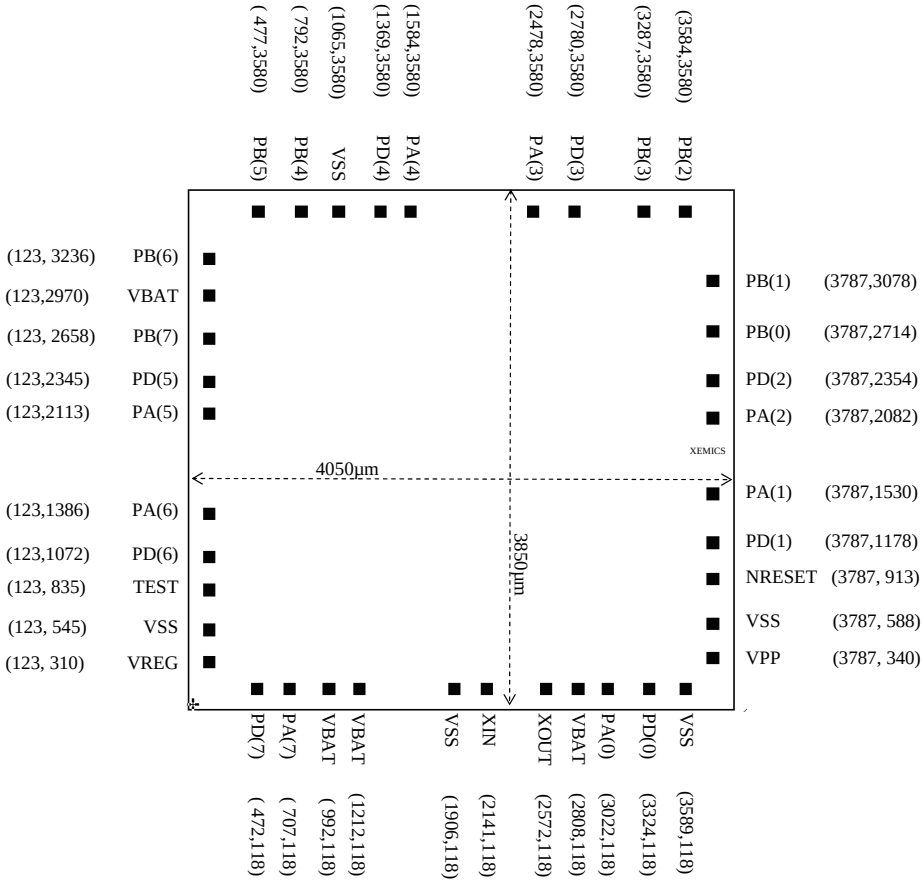


Figure 1-5. Die dimension and pin location of the XE8806A

1.2.5 Bare die XE8807A

The circuit is also available in bare die for chip on board assembly. All VBAT pins and all VSS pins should be connected together. The substrate of the circuit is connected to VSS.

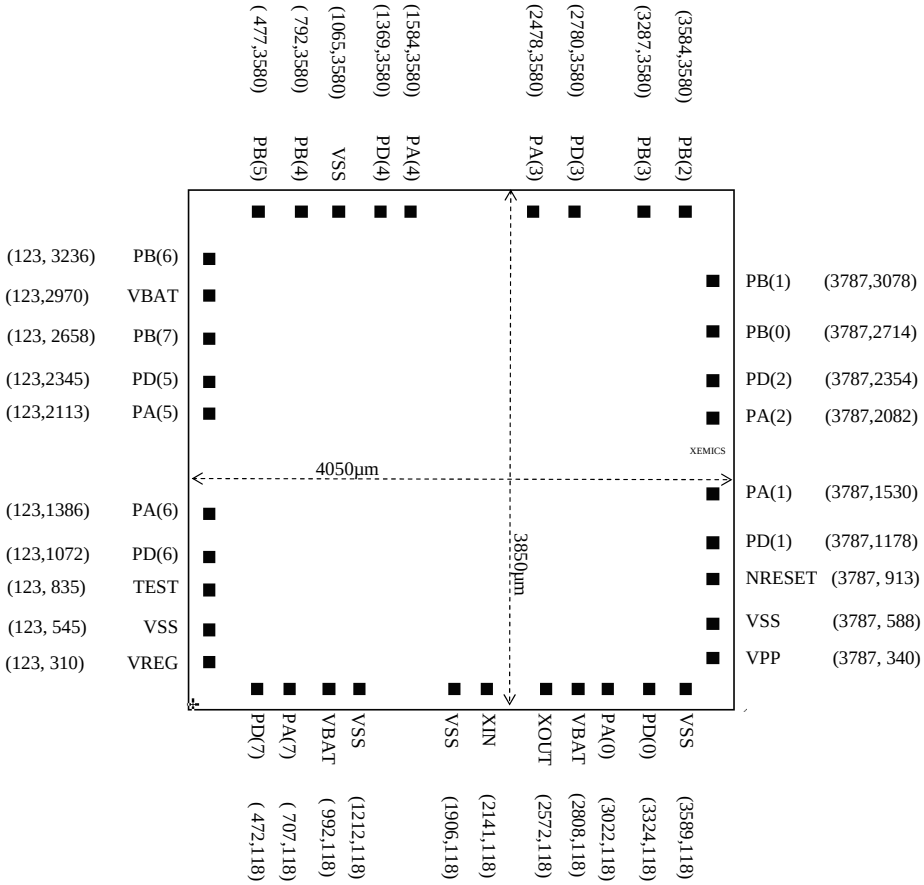


Figure 1-6. Die dimension and pin location of the XE8807A

1.3 Pin assignment

The table below gives a short description of the different pin assignments.

Pin	Assignment
VBAT	Positive power supply
VSS	Negative power supply
VREG	Connection for the mandatory external capacitor of the voltage regulator
VPP	High voltage supply for flash memory programming (NC in ROM versions)
NRESET	Resets the circuit when the voltage is low
TEST	Sets the pin to flash programming mode
XIN/XOUT	Quartz crystal connections, also used for flash memory programming
PA(7:0)	Parallel input port A pins
PB(7:0)	Parallel I/O port B pins
PD(7:0)	Parallel I/O port D pins

Table 1-1. Pin assignment

Table 1-2 gives a more detailed pin map for the different pins in the different packages. It also indicates the possible I/O configuration of these pins. The indications in blue bold are the configuration at start-up. Please note that in the SO-28 and SO-24 package several functions are routed to the same package pins. These pins are indicated in red italic. The pins RFIF(3:0) are the I/O pins of the RF interface, the CNTx pins are possible counter inputs, PWMx are possible PWM outputs, the CMPD pins are comparator inputs.

pin number			function			I/O configuration								
tqfp-32	SO-28	SO-24	first	second	third	AI	AO	DI	DO	OD	PU	PD	SNAP	POWER
1	4	*4*	**PD(7)**					**X**	X		**X**		X	
2	5	*4*	**PA(7)**					**X**			**X**		X	
3	6	5	**VSS**											**X**
4	7	6	**XIN**			**X**								
5	8	7	**XOUT**				**X**							
6	9	8	**VBAT**											**X**
7	10	*9*	**PA(0)**	CNTA				**X**			**X**		X	
8	11	*9*	**PD(0)**	RFIF(0)				**X**	X		**X**		X	
9	12	10	**VPP**											**X**
10	13	11	**NRESET**					**X**			**X**			
11	*14*	*12*	**PD(1)**	RFIF(1)				**X**	X		**X**		X	
12	*14*	*12*	**PA(1)**	CNTB				**X**			**X**		X	
13	*15*	*13*	**PA(2)**	CNTC				**X**			**X**		X	
14	*15*	*13*	**PD(2)**	RFIF(2)				**X**	X		**X**		X	
15	16	14	**PB(0)**	PWM0		X	X	**X**	X	X	**X**			
16	17	15	**PB(1)**	PWM1		X	X	**X**	X	X	**X**			
17	18	16	**PB(2)**			X	X	**X**	X	X	**X**			
18	19	17	**PB(3)**			X	X	**X**	X	X	**X**			
19	20	*18*	**PD(3)**	RFIF(3)				**X**	X		**X**		X	
20	21	*18*	**PA(3)**	CNTD				**X**			**X**		X	
21	22	*19*	**PA(4)**					**X**			**X**		X	
22	23	*19*	**PD(4)**					**X**	X		**X**		X	
23	24	20	**PB(4)**	USRT_S0	CMPD(0)	X	X	**X**	X	X	**X**			
24	25	21	**PB(5)**	USRT_S1	CMPD(1)	X	X	**X**	X	X	**X**			
25	26	22	**PB(6)**	UART_Tx	CMPD(2)	X	X	**X**	X	X	**X**			
26	27	23	**PB(7)**	UART_Rx	CMPD(3)	X	X	**X**	X	X	**X**			
27	*28*	*24*	**PD(5)**					**X**	X		**X**		X	
28	*28*	*24*	**PA(5)**					**X**			**X**		X	
29	*1*	*1*	**PA(6)**					**X**			**X**		X	
30	*1*	*1*	**PD(6)**					**X**	X		**X**		X	
31	2	2	**TEST**					**X**				**X**		
32	3	3	**VREG**				**X**							

Table 1-2. Pin description table

Pin map table legend:

red italic: pin shared with another peripheral in a specific package
blue bold: configuration at start up
AI: analog input
AO: analog output
DI: digital input
DO: digital output
OD: nMOS open drain output
PU: pull-up resistor
PD: pull-down resistor
SNAP: snap-to-rail function (see peripheral description for detailed description)
POWER: power supply

2 XE8806A and XE8807A Performance

2.1 Absolute maximum ratings

	Min.	Max.		Note
Voltage applied to VBAT with respect to VSS	-0.3	6.0	V	
Voltage applied to VPP with respect to VSS	VBAT-0.3	12	V	
Voltage applied to all pins except VPP and VBAT	VSS-0.3	VBAT+0.3	V	
Storage temperature (ROM device or unprogrammed flash device)	-55	150	°C	
Storage temperature (programmed flash device)	-40	85	°C	

Table 2-1. Absolute maximal ratings

Stresses beyond the absolute maximal ratings may cause permanent damage to the device. Functional operation at the absolute maximal ratings is not implied. Exposure to conditions beyond the absolute maximal ratings may affect the reliability of the device.

2.2 Operating range

	Min.	Max.		Note
Voltage applied to VBAT with respect to VSS	2.4	5.5	V	
Voltage applied to VBAT with respect to VSS during the flash programming	4.5	5.5	V	1
Voltage applied to VPP with respect to VSS	VBAT	11.5	V	
Voltage applied to all pins except VPP and VBAT	VSS	VBAT	V	
Operating temperature range	-40	85	°C	
Capacitor on VREG	0.8	1.2	μF	

Table 2-2. Operating range for the flash device

Note 1. During the programming of the device, the supply voltage should at least be equal to the supply voltage used during normal operation, and temperature between 10°C and 40°C.

		Min.	Max.		Note
Voltage applied to VBAT with respect to VSS	VREG by-passed	1.2	5.5	V	
	VREG on	1.5	3.6	V	
Voltage applied to all pins except VPP and VBAT		VSS	VBAT	V	
Operating temperature range		-40	125	°C	
Capacitor on VREG		0.1	1.2	μF	1

Table 2-3. Operating range for the ROM device

Note 1. The capacitor may be omitted when VREG is connected to VBAT.

All specifications in this document are valid for the complete operating range unless otherwise specified.

	Min.	Max.		Note
Retention time at 85°C	10		years	1
Retention time at 55°C	100		years	1
Number of programming cycles	10			2

Table 2-4. Operating range of the Flash memory

Note 1. Valid only if programmed using a programming tool that is qualified
Note 2. Circuits can be programmed more than 10 times but in that case, the retention time is no longer guaranteed.

2.3 Current consumption

The tables below give the current consumption for the circuit in different configurations. The figures are indicative only and may change as a function of the actual software implemented in the circuit.

Table 2-5 gives the current consumption for the flash version of the circuit. The peripherals (USRT, UART, CNT, VLD, CMPD) are disabled. The parallel ports are configured in input with pull up. Their pins are not connected externally.

Operation mode	CPU	RC	Xtal	Consumption	comments	Note
High speed CPU	1 MIPS	1 MHz	Off	200 µA	2.4V<>5.5V, 27°C	1
				320 µA		2
				410 µA		3
				310 µA		4
Low speed CPU	.1 MIPS	100 kHz	Off	21 µA	2.4V <>5.5V, 27°C	1
				33 µA		2
				42 µA		3
Low power CPU	32 kIPS	Off	32 kHz	7.5 µA	2.4V <>5.5V, 27°C	1
				11.0 µA		2
				14.5 µA		3
Low power time keeping	HALT	Off	32 kHz	1.9 µA	2.4V <>5.5V, 27°C	
Fast wake-up time keeping	HALT	Ready	32kHz	2.3 µA	2.4V <>5.5V, 27°C	
Immediate wake-up time keeping	HALT	1 MHz	Off	35 µA	2.4V <>5.5V, 27°C	
VLD static current				15 µA	2.4V <>5.5V, 27°C	
CMPD static current				2 µA	2.4V <>5.5V, 27°C	

Table 2-5. Typical current consumption of the XE8806AM version (8k instructions flash memory) and XE8807AM version (4k instructions flash memory)

1. Software without data access
2. 100% low power RAM access
3. 100% RAM access
4. typical software

Table 2-6 shows the typical current consumption for the ROM version with 8k instructions. Two possible modes are possible: a 2.4V-5.5V operating range using the internal regulator and a 1.2V-3.3V operating range short circuiting the voltage regulator (i.e. connect VREG to VBAT).

Operation mode	CPU	RC	Xtal	Consumption	comments	Note
High speed CPU	1 MIPS	1 MHz	Off	200	2.4V<>5.5V, 27°C	1,2
Max. Speed CPU	4 MIPS	4 MHz	Off	800	2.4V<>5.5V, 27°C	1,2
Low speed CPU	.1 MIPS	100 kHz	Off	21	2.4V <>5.5V, 27°C	1,2
Low power CPU	32 kIPS	Off	32 kHz	7	2.4V <>5.5V, 27°C	1,2
Low voltage CPU	32 kIPS	Off	32 kHz	1	1.2V, 27°C	1,3
Low power time keeping	HALT	Off	32 kHz	1.3	2.4V <>5.5V, 27°C	2

Table 2-6. Current consumption of the XE8806AR version (8k instructions ROM memory)

1. Software using MOVE instruction using internal CPU registers and peripheral registers.
2. Using the internal voltage regulator (see Figure 2-5).
3. With the internal regulator short circuited (i.e. by connecting VREG to VBAT, see Figure 2-7). In this case, the current consumption will increase with VBAT.

Hints for low power operation:

1. Use the low power RAM instead of the RAM for all parameters that are accessed frequently. The average current consumption for the low power RAM is about 40 times lower than for the RAM.
2. Rather than using the circuit at low speed, it is better to use the circuit at higher speed and switch off the blocks when not needed.
3. The power consumption of the program memory is an important part of the overall power consumption. In case you intend to use a ROM version and power consumption is too high, please ask us to provide you with a circuit version with smaller ROM size.

2.4 Operating speed

2.4.1 Flash circuit version XE8806AM

The speed of the flash devices is not highly dependent upon the supply voltage. However, by limiting the temperature range, the speed can be increased. The minimal guaranteed speed as a function of the supply voltage and maximal temperature operating temperature is given in Figure 2-2.

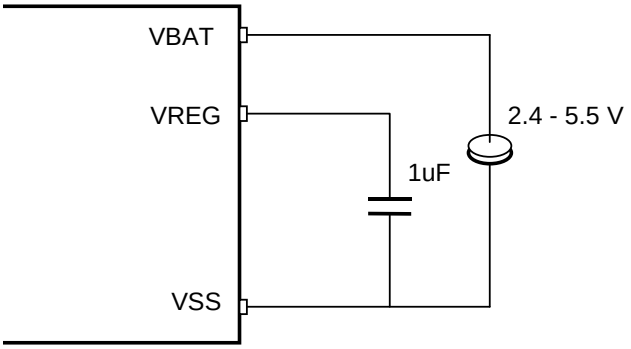


Figure 2-1. Supply configuration for flash circuit operation.

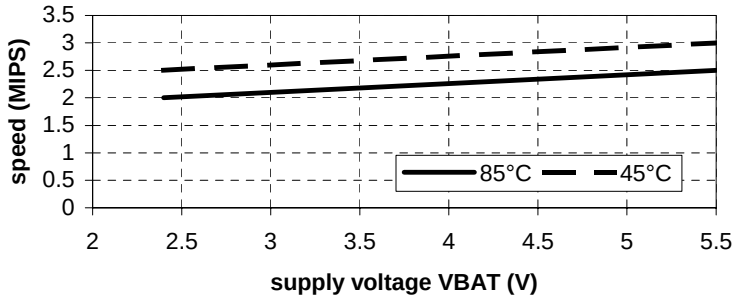


Figure 2-2. Guaranteed speed as a function of the supply voltage and maximal temperature.

Note that the speed of the flash circuit version is limited by the flash memory. All other peripherals of the device can run at the same speed as the ROM version (see Figure 2-6). The maximal speed of the peripherals can be exploited by reducing the CPU frequency by a factor of 2 with respect to the clock source by executing the instruction “FREQ div2”. Take care to execute this instruction before increasing the clock speed above the figures given in Figure 2-2.

2.4.2 Flash circuit version XE8807AM

The speed of the flash devices is not highly dependent upon the supply voltage. However, by limiting the temperature range, the speed can be increased. The minimal guaranteed speed as a function of the supply voltage and maximal temperature operating temperature is given in Figure 2-4.

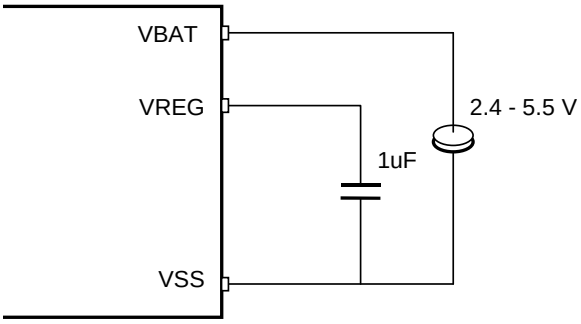


Figure 2-3. Supply configuration for flash circuit operation.

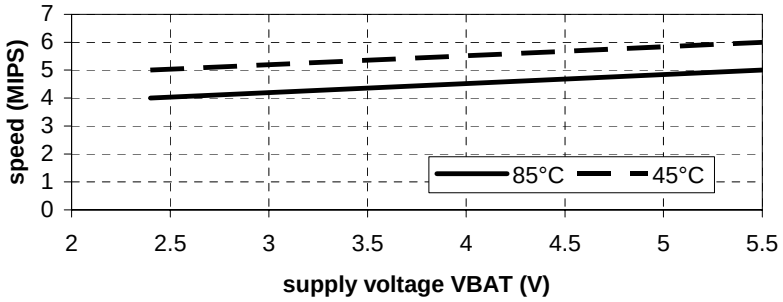


Figure 2-4. Guaranteed speed as a function of the supply voltage and maximal temperature.

2.4.3 ROM circuit version, regulator on

For the ROM version, two possible operating modes exist: with and without voltage regulator. Using the voltage regulator, a low power consumption will be obtained even with supply voltages above 2.4V. Without the voltage regulator (i.e. VREG short-circuited to VBAT) , a higher speed can be obtained.

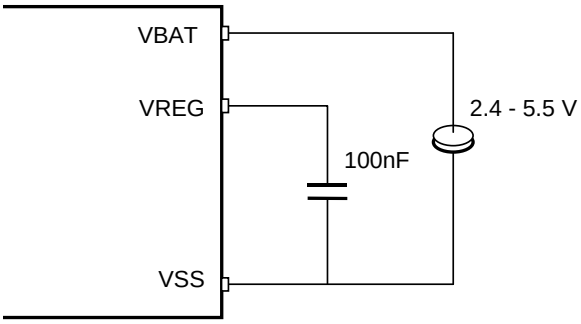


Figure 2-5. Supply configuration for ROM circuit operation using the internal regulator.

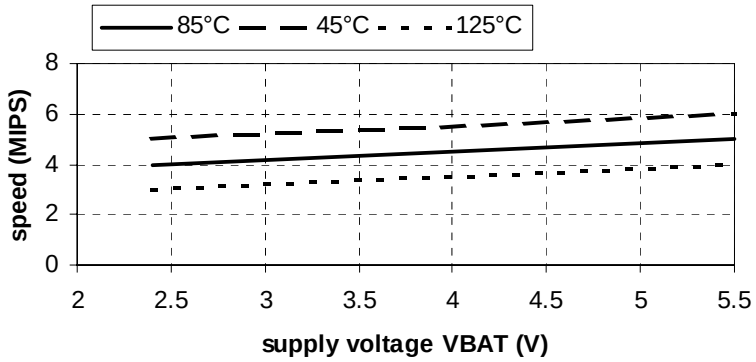


Figure 2-6. Guaranteed speed as a function of supply voltage and for different maximal temperatures using the voltage regulator.

2.4.4 ROM circuit version, regulator by-passed

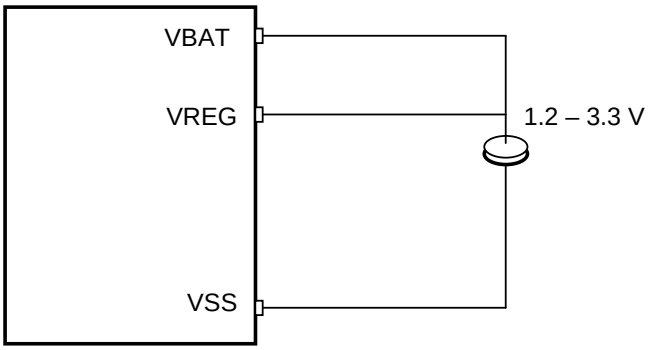


Figure 2-7. Supply configuration for ROM circuit operation by-passing the internal regulator.

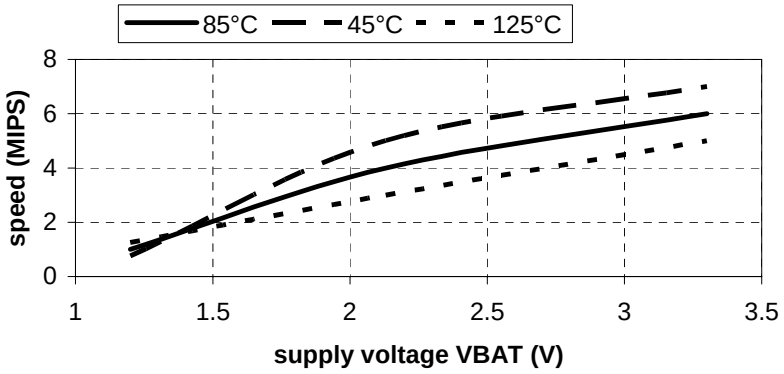


Figure 2-8. Guaranteed speed as a function of supply voltage and for two temperature ranges when VREG=VBAT.

3. CPU

CONTENTS

3.1 CPU description

The CPU of the XE8000 series is a low power RISC core. It has 16 internal registers for efficient implementation of the C compiler. Its instruction set is made up of 35 generic instructions, all coded on 22 bits, with 8 addressing modes. All instructions are executed in one clock cycle, including conditional jumps and 8x8 multiplication. The circuit therefore runs on 1 MIPS on a 1MHz clock.

The CPU hardware and software description is given in the document “Coolrisc816 Hardware and Software Reference Manual”. A short summary is given in the following paragraphs.

The good code efficiency of the CPU core makes it possible to compute a polynomial like $Z = (A_0 + A_1 \cdot Y) \cdot X + B_0 + B_1 \cdot Y$ in less than 300 clock cycles (software code generated by the XEMICS C-compiler, all numbers are signed integers on 16 bits).

3.2 CPU internal registers

As shown in Figure 3-1, the CPU has 16 internal 8-bit registers. Some of these registers can be concatenated to a 16-bit word for use in some instructions. The function of these registers is defined in Table 3-1. The status register stat (Table 3-2) is used to manage the different interrupt and event levels. An interrupt or an event can both be used to wake up after a HALT instruction. The difference is that an interrupt jumps to a special interrupt function whereas an event continues the software execution with the instruction following the HALT instruction.

The program counter (PC) is a 16 bit register that indicates the address of the instruction that has to be executed. The stack (ST_n) is used to memorise the return address when executing subroutines or interrupt routines.

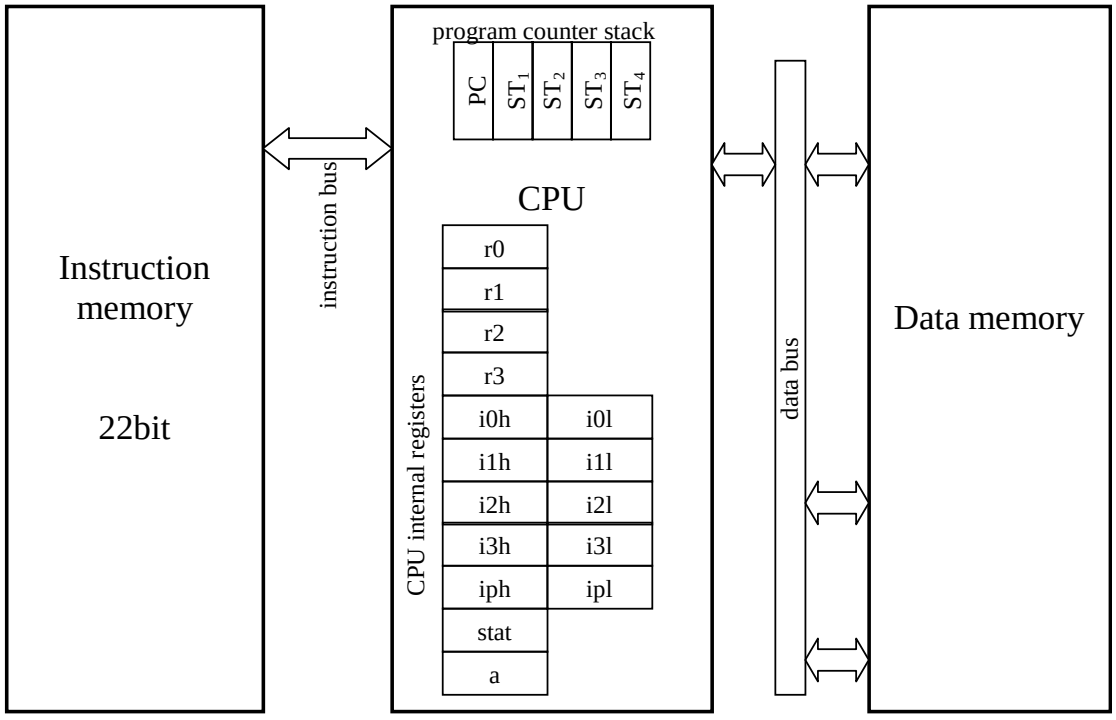


Figure 3-1. CPU internal registers

Register name	Register function
r0	general purpose
r1	general purpose
r2	general purpose
r3	data memory offset
i0h	MSB of the data memory index i0
i0l	LBS of the data memory index i0
i1h	MSB of the data memory index i1
i1l	LBS of the data memory index i1
i2h	MSB of the data memory index i2
i2l	LBS of the data memory index i2
i3h	MSB of the data memory index i3
i3l	LBS of the data memory index i3
iph	MSB of the program memory index ip
ipl	LBS of the program memory index ip
stat	status register
a	accumulator

Table 3-1. CPU internal register definition

bit	name	function
7	IE2	enables (when 1) the interrupt request of level 2
6	IE1	enables (when 1) the interrupt request of level 1
5	GIE	enables (when 1) all interrupt request levels
4	IN2	interrupt request of level 2. The interrupts labelled “low” in the interrupt handler are routed to this interrupt level. This bit has to be cleared when the interrupt is served.
3	IN1	interrupt request of level 1. The interrupts labelled “mid” in the interrupt handler are routed to this interrupt level. This bit has to be cleared when the interrupt is served.
2	IN0	interrupt request of level 0. The interrupts labelled “hig” in the interrupt handler are routed to this interrupt level. This bit has to be cleared when the interrupt is served.
1	EV1	event request of level 1. The events labelled “low” in the event handler are routed to this event level. This bit has to be cleared when the event is served.
0	EV0	event request of level 1. The events labelled “hig” in the event handler are routed to this event level. This bit has to be cleared when the event is served.

Table 3-2. Status register description

The CPU also has a number of flags that can be used for conditional jumps. These flags are defined in Table 3-3.

symbol	name	function
Z	zero	Z=1 when the accumulator a content is zero
C	carry	This flag is used in shift or arithmetic operations. For a shift operation, it has the value of the bit that was shifted out (LSB for shift right, MSB for shift left). For an arithmetic operation with unsigned numbers: it is 1 at occurrence of an overflow during an addition (or equivalent). it is 0 at occurrence of an underflow during a subtraction (or equivalent).
V	overflow	This flag is used in shift or arithmetic operations. For arithmetic or shift operations with signed numbers, it is 1 if an overflow or underflow occurs.

Table 3-3. Flag description

3.3 CPU instruction short reference

Table 3-4 shows a short description of the different instructions available on the Coolrisc816. The notation cc in the conditional jump instruction refers to the condition description as given in Table 3-6. The notation reg, reg1, reg2, reg3 refers to one of the CPU internal registers of Table 3-1. The notation eaddr and DM(eaddr) refer to one of the extended address modes as defined in Table 3-5. The notation DM(xxx) refers to the data memory location with address xxx.

Instruction	Modification	Operation
Jump addr[15:0]	-,-,-, -	PC := addr[15:0]
Jump ip	-,-,-, -	PC := ip
Jcc addr[15:0]	-,-,-, -	if cc is true then PC := addr[15:0]
Jcc ip	-,-,-, -	if cc is true then PC := ip
Call addr[15:0]	-,-,-, -	ST_{n+1} := ST_n (n>1); ST_1 := PC+1; PC := addr[15:0]
Call ip	-,-,-, -	ST_{n+1} := ST_n (n>1); ST_1 := PC+1; PC := ip
Calls addr[15:0]	-,-,-, -	ip := PC+1; PC := addr[15:0]
Calls ip	-,-,-, -	ip := PC+1; PC := ip
Ret	-,-,-, -	PC := ST_1; ST_n := ST_{n+1} (n>1)
Rets	-,-,-, -	PC := ip
Reti	-,-,-, -	PC := ST_1; ST_n := ST_{n+1} (n>1); GIE :=1
Push	-,-,-, -	PC := PC+1; ST_{n+1} := ST_n (n>1); ST_1 := ip
Pop	-,-,-, -	PC := PC+1; ip := ST_1; ST_n := ST_{n+1} (n>1)
Move reg,#data[7:0]	-,-, Z, a	a := data[7:0]; reg := data[7:0]
Move reg1, reg2	-,-, Z, a	a := reg2; reg1 := reg2
Move reg, eaddr	-,-, Z, a	a := DM(eaddr); reg := DM(eaddr)
Move eaddr, reg	-,-,-, -	DM(eaddr) := reg
Move addr[7:0],#data[7:0]	-,-,-, -	DM(addr[7:0]) := data[7:0]
Cmvd reg1, reg2	-,-, Z, a	a := reg2; if C=0 then reg1 := a;
Cmvd reg, eaddr	-,-, Z, a	a := DM(eaddr); if C=0 then reg := a
Cmvs reg1, reg2	-,-, Z, a	a := reg2; if C=1 then reg1 := a;
Cmvs reg, eaddr	-,-, Z, a	a := DM(eaddr); if C=1 then reg := a
Shl reg1, reg2	C, V, Z, a	a := reg2<<1; a[0] := 0; C := reg2[7]; reg1 := a
Shl reg	C, V, Z, a	a := reg<<1; a[0] := 0; C := reg[7]; reg := a
Shl reg, eaddr	C, V, Z, a	a := DM(eaddr)<<1; a[0] :=0; C := DM(eaddr)[7]; reg := a
Shlc reg1, reg2	C, V, Z, a	a := reg2<<1; a[0] := C; C := reg2[7]; reg1 := a
Shlc reg	C, V, Z, a	a := reg<<1; a[0] := C; C := reg[7]; reg := a
Shlc reg, eaddr	C, V, Z, a	a := DM(eaddr)<<1; a[0] := C; C := DM(eaddr)[7]; reg := a
Shr reg1, reg2	C, V, Z, a	a := reg2>>1; a[7] := 0; C := reg2[0]; reg1 :=a
Shr reg	C, V, Z, a	a := reg>>1; a[7] := 0; C := reg[0]; reg := a
Shr reg, eaddr	C, V, Z, a	a := DM(eaddr)>>1; a[7] := 0; C := DM(eaddr)[0]; reg := a
Shrc reg1, reg2	C, V, Z, a	a := reg2>>1; a[7] := C; C := reg2[0]; reg1 := a
Shrc reg	C, V, Z, a	a := reg>>1; a[7] := C; C := reg[0]; reg := a
Shrc reg, eaddr	C, V, Z, a	a := DM(eaddr)>>1; a[7] := C; C := DM(eaddr)[0]; reg := a
Shra reg1, reg2	C, V, Z, a	a := reg2>>1; a[7] := reg2[7]; C := reg2[0]; reg1 := a
Shra reg	C, V, Z, a	a := reg>>1; a[7] := reg[7]; C := reg[0]; reg := a
Shra reg, eaddr	C, V, Z, a	a := DM(eaddr)>>1; a[7] := DM(eaddr)[7]; C := DM(eaddr)[0]; reg := a
Cpl1 reg1, reg2	-,-, Z, a	a := NOT(reg2); reg1 := a
Cpl1 reg	-,-, Z, a	a := NOT(reg); reg := a
Cpl1 reg, eaddr	-,-, Z, a	a := NOT(DM(eaddr)); reg := a
Cpl2 reg1, reg2	C, V, Z, a	a := NOT(reg2)+1; if a=0 then C:=1 else C := 0; reg1 := a
Cpl2 reg	C, V, Z, a	a := NOT(reg)+1; if a=0 then C:=1 else C := 0; reg := a
Cpl2 reg, eaddr	C, V, Z, a	a := NOT(DM(eaddr))+1; if a=0 then C:=1 else C := 0; reg := a
Cpl2c reg1, reg2	C, V, Z, a	a := NOT(reg2)+C; if a=0 and C=1 then C:=1 else C := 0; reg1 := a
Cpl2c reg	C, V, Z, a	a := NOT(reg)+C; if a=0 and C=1 then C:=1 else C := 0; reg := a
Cpl2c reg, eaddr	C, V, Z, a	a := NOT(DM(eaddr))+C; if a=0 and C=1 then C:=1 else C := 0; reg := a
Inc reg1, reg2	C, V, Z, a	a := reg2+1; if a=0 then C := 1 else C := 0; reg1 := a
Inc reg	C, V, Z, a	a := reg+1; if a=0 then C := 1 else C := 0; reg := a
Inc reg, eaddr	C, V, Z, a	a := DM(eaadr)+1; if a=0 then C := 1 else C := 0; reg := a
Incc reg1, reg2	C, V, Z, a	a := reg2+C; if a=0 and C=1 then C := 1 else C := 0; reg1 := a
Incc reg	C, V, Z, a	a := reg+C; if a=0 and C=1 then C := 1 else C := 0; reg := a
Incc reg, eaddr	C, V, Z, a	a := DM(eaadr)+C; if a=0 and C=1 then C := 1 else C := 0; reg := a
Dec reg1, reg2	C, V, Z, a	a := reg2-1; if a=hFF then C := 0 else C := 1; reg1 := a

Dec reg	C, V, Z, a	a := reg-1; if a=hFF then C := 0 else C := 1; reg := a
Dec reg, eaddr	C, V, Z, a	a := DM(eaddr)-1; if a=hFF then C := 0 else C := 1; reg := a
Decc reg1, reg2	C, V, Z, a	a := reg2-(1-C); if a=hFF and C=0 then C := 0 else C := 1; reg1 := a
Decc reg	C, V, Z, a	a := reg-(1-C); if a=hFF and C=0 then C := 0 else C := 1; reg := a
Decc reg, eaddr	C, V, Z, a	a := DM(eaddr)-(1-C); if a=hFF and C=0 then C := 0 else C := 1; reg := a
And reg,#data[7:0]	-,-, Z, a	a := reg and data[7:0]; reg := a
And reg1, reg2, reg3	-,-, Z, a	a := reg2 and reg3; reg1 := a
And reg1, reg2	-,-, Z, a	a := reg1 and reg2; reg1 := a
And reg, eaddr	-,-, Z, a	a := reg and DM(eaddr); reg := a
Or reg,#data[7:0]	-,-, Z, a	a := reg or data[7:0]; reg := a
Or reg1, reg2, reg3	-,-, Z, a	a := reg2 or reg3; reg1 := a
Or reg1, reg2	-,-, Z, a	a := reg1 or reg2; reg1 := a
Or reg, eaddr	-,-, Z, a	a := reg or DM(eaddr); reg := a
Xor reg,#data[7:0]	-,-, Z, a	a := reg xor data[7:0]; reg := a
Xor reg1, reg2, reg3	-,-, Z, a	a := reg2 xor reg3; reg1 := a
Xor reg1, reg2	-,-, Z, a	a := reg1 xor reg2; reg1 := a
Xor reg, eaddr	-,-, Z, a	a := reg or DM(eaddr); reg := a
Add reg,#data[7:0]	C, V, Z, a	a := reg+data[7:0]; if overflow then C:=1 else C := 0; reg := a
Add reg1, reg2, reg3	C, V, Z, a	a := reg2+reg3; if overflow then C:=1 else C := 0; reg1 := a
Add reg1, reg2	C, V, Z, a	a := reg1+reg2; if overflow then C:=1 else C := 0; reg1 := a
Add reg, eaddr	C, V, Z, a	a := reg+DM(eaddr); if overflow then C:=1 else C := 0; reg := a
Addc reg,#data[7:0]	C, V, Z, a	a := reg+data[7:0]+C; if overflow then C:=1 else C := 0; reg := a
Addc reg1, reg2, reg3	C, V, Z, a	a := reg2+reg3+C; if overflow then C:=1 else C := 0; reg1 := a
Addc reg1, reg2	C, V, Z, a	a := reg1+reg2+C; if overflow then C:=1 else C := 0; reg1 := a
Addc reg, eaddr	C, V, Z, a	a := reg+DM(eaddr)+C; if overflow then C:=1 else C := 0; reg := a
Subd reg,#data[7:0]	C, V, Z, a	a := data[7:0]-reg; if underflow then C := 0 else C := 1; reg := a
Subd reg1, reg2, reg3	C, V, Z, a	a := reg2-reg3; if underflow then C := 0 else C := 1; reg1 := a
Subd reg1, reg2	C, V, Z, a	a := reg2-reg1; if underflow then C := 0 else C := 1; reg1 := a
Subd reg, eaddr	C, V, Z, a	a := DM(eaddr)-reg; if underflow then C := 0 else C := 1; reg := a
Subdc reg,#data[7:0]	C, V, Z, a	a := data[7:0]-reg-(1-C); if underflow then C := 0 else C := 1; reg := a
Subdc reg1, reg2, reg3	C, V, Z, a	a := reg2-reg3-(1-C); if underflow then C := 0 else C := 1; reg1 := a
Subdc reg1, reg2	C, V, Z, a	a := reg2-reg1-(1-C); if underflow then C := 0 else C := 1; reg1 := a
Subdc reg, eaddr	C, V, Z, a	a := DM(eaddr)-reg-(1-C); if underflow then C := 0 else C := 1; reg := a
Subs reg,#data[7:0]	C, V, Z, a	a := reg-data[7:0]; if underflow then C := 0 else C := 1; reg := a
Subs reg1, reg2, reg3	C, V, Z, a	a := reg3-reg2; if underflow then C := 0 else C := 1; reg1 := a
Subs reg1, reg2	C, V, Z, a	a := reg1-reg2; if underflow then C := 0 else C := 1; reg1 := a
Subs reg, eaddr	C, V, Z, a	a := reg-DM(eaddr); if underflow then C := 0 else C := 1; reg := a
Subsc reg,#data[7:0]	C, V, Z, a	a := reg-data[7:0]-(1-C); if underflow then C := 0 else C := 1; reg := a
Subsc reg1, reg2, reg3	C, V, Z, a	a := reg3-reg2-(1-C); if underflow then C := 0 else C := 1; reg1 := a
Subsc reg1, reg2	C, V, Z, a	a := reg1-reg2-(1-C); if underflow then C := 0 else C := 1; reg1 := a
Subsc reg, eaddr	C, V, Z, a	a := reg-DM(eaddr)-(1-C); if underflow then C := 0 else C := 1; reg := a
Mul reg,#data[7:0]	u, u, u, a	a := (data[7:0]*reg)[7:0]; reg := (data[7:0]*reg)[15:8]
Mul reg1, reg2, reg3	u, u, u, a	a := (reg2*reg3)[7:0]; reg1 := (reg2*reg3)[15:8]
Mul reg1, reg2	u, u, u, a	a := (reg2*reg1)[7:0]; reg1 := (reg2*reg1)[15:8]
Mul reg, eaddr	u, u, u, a	a := (DM(eaddr)*reg)[7:0]; reg := (DM(eaddr)*reg)[15:8]
Mula reg,#data[7:0]	u, u, u, a	a := (data[7:0]*reg)[7:0]; reg := (data[7:0]*reg)[15:8]
Mula reg1, reg2, reg3	u, u, u, a	a := (reg2*reg3)[7:0]; reg1 := (reg2*reg3)[15:8]
Mula reg1, reg2	u, u, u, a	a := (reg2*reg1)[7:0]; reg1 := (reg2*reg1)[15:8]
Mula reg, eaddr	u, u, u, a	a := (DM(eaddr)*reg)[7:0]; reg := (DM(eaddr)*reg)[15:8]
Mshl reg,#shift[2:0]	u, u, u, a	a := (reg*2^{shift})[7:0]; reg := (reg*2^{shift})[15:8]
Mshr reg,#shift[2:0]	u, u, u, a	a := (reg*$2^{(8-shift)}$)[7:0]; reg := (reg*$2^{(8-shift)}$)[15:8]
Mshra reg,#shift[2:0]	u, u, u, a*	a := (reg*$2^{(8-shift)}$)[7:0]; reg := (reg*$2^{(8-shift)}$)[15:8]
Cmp reg,#data[7:0]	C, V, Z, a	a := data[7:0]-reg; if underflow then C :=0 else C:=1; V := C and (not Z)
Cmp reg1, reg2	C, V, Z, a	a := reg2-reg1; if underflow then C :=0 else C:=1; V := C and (not Z)
Cmp reg, eaddr	C, V, Z, a	a := DM(eaddr)-reg; if underflow then C :=0 else C:=1; V := C and (not Z)
Cmpa reg,#data[7:0]	C, V, Z, a	a := data[7:0]-reg; if underflow then C :=0 else C:=1; V := C and (not Z)
Cmpa reg1, reg2	C, V, Z, a	a := reg2-reg1; if underflow then C :=0 else C:=1; V := C and (not Z)
Cmpa reg, eaddr	C, V, Z, a	a := DM(eaddr)-reg; if underflow then C :=0 else C:=1; V := C and (not Z)
Tstb reg,#bit[2:0]	-, -, Z, a	a[bit] := reg[bit]; other bits in a are 0
Setb reg,#bit[2:0]	-, -, Z, a	reg[bit] := 1; other bits unchanged; a := reg
Clrb reg,#bit[2:0]	-, -, Z, a	reg[bit] := 0; other bits unchanged; a := reg
Invb reg,#bit[2:0]	-, -, Z, a	reg[bit] := not reg[bit]; other bits unchanged; a := reg

Sflag	-,-,-, a	a[7] := C; a[6] := C xor V; a[5] := ST full; a[4] := ST empty
Rflag reg	C, V, Z, a	a := reg << 1; ; a[0] := 0; C := reg[7]
Rflag eaddr	C, V, Z, a	a := DM(eaddr)<<1; a[0] :=0; C := DM(eaddr)[7]
Freq divn	-,-,-, -	reduces the CPU frequency (divn=nodiv, div2, div4, div8, div16)
Halt	-,-,-, -	halts the CPU
Nop	-,-,-, -	no operation

- = unchanged, u = undefined, *MSHR reg,# 1 doesn't shift by 1

Table 3-4. Instruction short reference

The Coolrisc816 has 8 different addressing modes. These modes are described in Table 3-5. In this table, the notation ix refers to one of the data memory index registers i0, i1, i2 or i3. Using eaddr in an instruction of Table 3-4 will access the data memory at the address DM(eaddr) and will simultaneously execute the index operation.

extended address eaddr	accessed data memory location DM(eaddr)	index operation	
addr[7:0]	DM(h00&addr[7:0])	-	direct addressing
(ix)	DM(ix)	-	indexed addressing
(ix, offset[7:0])	DM(ix+offset)	-	indexed addressing with immediate offset
(ix,r3)	DM(ix+r3)	-	indexed addressing with register offset
(ix)+	DM(ix)	ix := ix+1	indexed addressing with index post-increment
(ix,offset[7:0])+	DM(ix+offset)	ix := ix+offset	indexed addressing with index post-increment by the offset
-(ix)	DM(ix-1)	ix := ix-1	indexed addressing with index pre-decrement
-(ix,offset[7:0])	DM(ix-offset)	ix := ix -offset	indexed addressing with index pre-decrement by the offset

Table 3-5. Extended address mode description

Eleven different jump conditions are implemented as shown in Table 3-6. The contents of the column CC in this table should replace the CC notation in the instruction description of Table 3-4.

CC	condition
CS	C=1
CC	C=0
ZS	Z=1
ZC	Z=0
VS	V=1
VC	V=0
EV	(EV1 or EV0)=1
After CMP op1,op2	
EQ	op1=op2
NE	op1≠op2
GT	op1>op2
GE	op1≥op2
LT	op1<op2
LE	op1≤op2

Table 3-6. Jump condition description

4. Memory Mapping

4.1 Memory organisation

The XE8806A and XE8807A CPU are built with Harvard architecture. The Harvard architecture uses separate instruction and data memories. The instruction bus and data bus are also separated. The advantage of such a structure is that the CPU can fetch a new instruction and read/write data simultaneously. The circuit configuration is shown in Figure 4-1. The CPU has its 16 internal registers. The instruction memory has a capacity of 8192 22-bit instructions in the XE8806A and 4096 22-bit instructions in the XE8807A. The data memory space has 8 low power registers, the peripheral register space and 512 bytes of RAM.

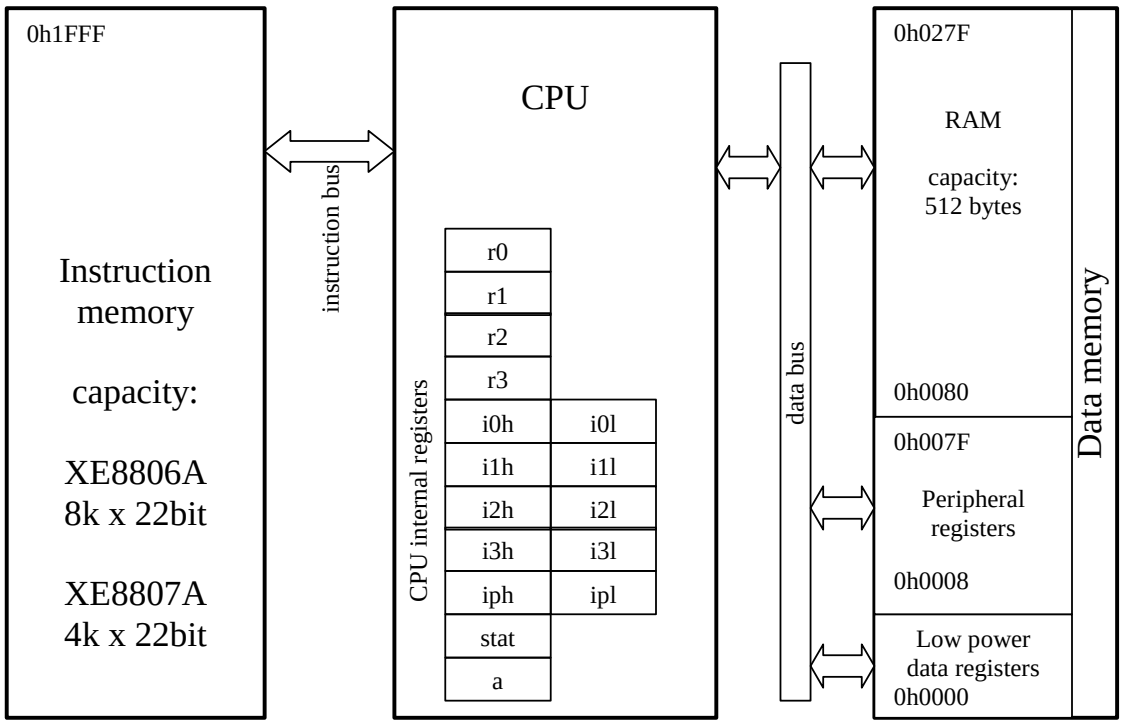


Figure 4-1. Memory mapping

The CPU internal registers are described in the CPU chapter. A short reference of the low power registers and peripheral registers is given in 4.2.

4.2 Quick reference data memory register map

The data register map is given in the tables below. A more detailed description of the different registers is given in the detailed description of the different peripherals.

The tables give the following information:

1. The register name and register address
2. The different bits in the register
3. The access mode of the different bits (see Table 4-1 for code description)
4. The reset source and reset value of the different bits

The reset source coding is given in Table 4-2. To get a full description of the reset sources, please refer to the reset block chapter.

code	access mode
r	bit can be read
w	bit can be written
r0	bit always reads 0
r1	bit always reads 1
c	bit is cleared by writing any value
c1	bit is cleared by writing a 1
ca	bit is cleared after reading
s	special function, verify the detailed description in the respective peripherals

Table 4-1. Access mode codes used in the register definitions

code	reset source
glob	nresetglobal
cold	nresetcold
pconf	nresetpconf
sleep	nresetsleep

Table 4-2. Reset source coding used in the register definitions

4.2.1 Low power data registers (h0000-h0007)

Name Address	7	6	5	4	3	2	1	0
Reg00 h0000	Reg00[7:0] rw,00000000,glob							
Reg01 h0001	Reg01[7:0] rw,00000000,glob							
Reg02 h0002	Reg02[7:0] rw,00000000,glob							
Reg03 h0003	Reg03[7:0] rw,00000000,glob							
Reg04 h0004	Reg04[7:0] rw,00000000,glob							
Reg05 h0005	Reg05[7:0] rw,00000000,glob							
Reg06 h0006	Reg06[7:0] rw,00000000,glob							
Reg07 h0007	Reg07[7:0] rw,0000000,glob							

Table 4-3. Low power data registers

4.2.2 System, clock configuration and reset configuration (h0010-h001F)

Name Address	7	6	5	4	3	2	1	0
RegSysCtrl h0010	SleepEn rw,0,cold	EnResetPConf rw,0,cold	EnBusError rw,0,cold	EnResetWD rw,0,cold	r0	r0	r0	r0
RegSysReset h0011	Sleep rw,0,glob	SleepFlag rc,0,cold	ResetBusError rc, 0, cold	ResetWD rc, 0, cold	ResetfromportA rc, 0, cold	r0	r0	r0
RegSysClock h0012	CpuSel rw,0,sleep	r0	EnExtClock rw,0,cold	BiasRC rw,1,cold	ColdXtal r,1,sleep	r0	EnableXtal rw,0,sleep	EnableRC rw,1,sleep
RegSysMisc h0013	r0	r0	r0	r0	r0	r0	Output16k rw,0,sleep	OutputCpuCk rw,0,sleep
RegSysWd h0014	r0	r0	r0	r0	WatchDog[3:0] s,0000,glob			
RegSysPre0 h0015	r0	r0	r0	r0	r0	r0	r0	ClearLowPresca l c1r0,0,-
RegSysRcTrim1 h001B	r0	r0	r0	RcFreqRange rw,0,cold	RcFreqCoarse[3:0] rw,0001,cold			
RegSysRcTrim2 h001C	r0	r0	RcFreqFine[5:0] rw,00000,cold					

Table 4-4. Reset block and clock block registers

4.2.3 Port A (h0020-h0027)

Name / Address	7	6	5	4	3	2	1	0
RegPAIn h0020	PAIn[7:0] r							
RegPADebounce h0021	PADebounce[7:0] rw,00000000,pconf							
RegPAEdge h0022	PAEdge[7:0] rw,00000000,glob							
RegPAPullup h0023	PAPullup[7:0] rw,11111111,pconf							
RegPARes0 h0024	PARes0[7:0] rw, 00000000, glob							
RegPARes1 h0025	PARes1[7:0] rw,00000000,glob							
RegPACtrl h0026	r0	r0	r0	r0	r0	r0	r0	DebFast rw,0,pconf
RegPASnaptorail h0027	PASnaptorail[7:0] rw,00000000,pconf							

Table 4-5. Port A registers

4.2.4 Port B (h0028-h002F)

Name / Address	7	6	5	4	3	2	1	0
RegPBOut h0028	PBOut[7:0] rw,00000000,pconf							
RegPBIn h0029	PBIn[7:0] r							
RegPBDir h002A	PBDir[7:0] rw,00000000,pconf							
RegPBOpen h002B	PBOpen[7:0] rw,00000000,pconf							
RegPBPullup h002C	PBPullup[7:0] rw,11111111,pconf							
RegPBAna h002D	PBAna[7:0] rw,00000000,pconf							

Table 4-6. Port B registers

4.2.5 Port D (h0030-h0033)

Name / Address	7	6	5	4	3	2	1	0
RegPDOut h0030	PDOut[7:0] rw,00000000,pconf							
RegPDIn h0031	PDIn[7:0] r							
RegPDDir h0032	PDDir[7:0] rw,00000000,pconf							
RegPDPullup h0033	PDSnapToRail[3:0] rw,0000,pconf				PDPullup[3:0] rw,1111,pconf			

Table 4-7. Port D registers

4.2.6 Flash programming (h0038-003B)

These four registers are used during flash programming only. Refer to the flash programming algorithm documentation for more details.

4.2.7 Event handler (h003C-h003F)

Name Address	7	6	5	4	3	2	1	0
RegEvn h003C	CntIrqA rc1,0,glob	CntIrqC rc1,0,glob	128Hz rc1,0,glob	PAEvn[1] rc1,0,glob	CntIrqB rc1,0,glob	CntIrqD rc1,0,glob	1Hz rc1,0,glob	PAEvn[0] rc1,0,glob
RegEvnEn h003D	EvnEn[7:0] rw,00000000,glob							
RegEvnPriority h003E	EvnPriority[7:0] r,11111111,glob							
RegEvnEvn h003F	r0	r0	r0	r0	r0	r0	EvnHigh r,0,glob	EvnLow r,0,glob

Table 4-8. Event handler registers

The origin of the different events is summarised in the table below.

Event	Event source
CntIrqA	Counter/Timer A (counter block)
CntIrqB	Counter/Timer B (counter block)
CntIrqC	Counter/Timer C (counter block)
CntIrqD	Counter/Timer D (counter block)
128Hz	Low prescaler (clock block)
1Hz	Low prescaler (clock block)
PAEvn[1:0]	Port A

Table 4-9. Event source description

4.2.8 Interrupt handler (h0040-h0047)

Name Address	7	6	5	4	3	2	1	0
RegIrqHig h0040	RfifRx rc1,0,glob	128Hz rc1,0,glob	RfifTx rc1,0,glob	CntIrqA rc1,0,glob	CntIrqC rc1,0,glob	CmpdIrq rc1,0,glob	UartIrqTx rc1,0,glob	UartIrqRx rc1,0,glob
RegIrqMid h0041	UsrtCond2 rc1,0,glob	UrstCond1 rc1,0,glob	PAIrq[5] rc1,0,glob	PAIrq[4] rc1,0,glob	1Hz rc1,0,glob	VldIrq rc1,0,glob	PAIrq[1] rc1,0,glob	PAIrq[0] rc1,0,glob
RegIrqLow h0042	PAIrq[7] rc1,0,glob	PAIrq[6] rc1,0,glob	CntIrqB rc1,0,glob	CntIrqD rc1,0,glob	PAIrq[3] rc1,0,glob	PAIrq[2] rc1,0,glob	r0	r0
RegIrqEnHig h0043	IrqEnHig[7:0] rw,0000000,glob							
RegIrqEnMid h0044	IrqEnMid[7:0] rw,0000000,glob							
RegIrqEnLow h0045	IrqEnLow[7:0] rw,0000000,glob							
RegIrqPriority h0046	IrqPriority[7:0] r,11111111,glob							
RegIrqIrq h0047	r0	r0	r0	r0	r0	IrqHig r,0,glob	IrqMid r,0,glob	IrqLow r,0,glob

Table 4-10. Interrupt handler registers

The origin of the different interrupts is summarised in the table below.

Interrupt	Interrupt source
CmpdIrq	Low power comparators
CntIrqA	Counter/Timer A (counter block)
CntIrqB	Counter/Timer B (counter block)
CntIrqC	Counter/Timer C (counter block)
CntIrqD	Counter/Timer D (counter block)
128Hz	Low prescaler (clock block)
1Hz	Low prescaler (clock block)
PAIrq[7:0]	Port A
RfifRx	RF interface reception
RfifTx	RF interface transmission
UartIrqRx	UART reception
UartIrqTx	UART transmission
UsrtCond1	USRT condition 1
UsrtCond2	USRT condition 2
VldIrq	Voltage level detector

Table 4-11. Interrupt source description

4.2.9 USRT (h0048-h004F)

Name Address	7	6	5	4	3	2	1	0
RegUsrtS1 h0048	r0	r0	r0	r0	r0	r0	r0	UsrtS1 s,1,glob
RegUsrtS0 h0049	r0	r0	r0	r0	r0	r0	r0	UsrtS0 s,1,glob
RegUsrtCond1 h004A	r0	r0	r0	r0	r0	r0	r0	UsrtCond1 rc,0,glob
RegUsrtCond2 h004B	r0	r0	r0	r0	r0	r0	r0	UsrtCond2 rc,0,glob
RegUsrtCtrl h004C	r0	r0	r0	r0	UsrtWaitS0 r,0,glob	UsrtEnWaitCond1 rw,0,glob	UsrtEnWaitS0 rw,0,glob	UsrtEnable rw,0,glob
RegUsrtBufferS1 h004D	r0	r0	r0	r0	r0	r0	r0	UsrtBufferS1 r,0,glob
RegUsrtEdgeS0 h004E	r0	r0	r0	r0	r0	r0	r0	UsrtEdgeS0 r,0,glob

Table 4-12. USRT register description

4.2.10 UART (h0050-h0057)

Name Address	7	6	5	4	3	2	1	0
RegUartCtrl h0050	UartEcho rw,0,glob	UartEnRx rw,0,glob	UartEnTx rw,0,glob	UartXRx rw,0,glob	UartXTx rw,0,glob	UartBR[2:0] rw,101,glob		
RegUartCmd h0051	SelXtal rw,0,glob	r0	UartRcSel[2:0] rw,000,glob			UartPM rw,0,glob	UartPE rw,0,glob	UartWL rw,1,glob
RegUartTx h0052	UartTx[7:0] rw,0000000,glob							
RegUartTxSta h0053	r0	r0	r0	r0	r0	r0	UartTxBusy r,0,glob	UartTxFull r,0,glob
RegUartRx h0054	UartRx[7:0] r,00000000,glob							
RegUartRxSta h0055	r0	r0	UartRxSErr r,0,glob	UartRxPErr r,0,glob	UartRxFErr r,0,glob	UartRxOerr rc,0,glob	UartRxBusy r,0,glob	UartRxFull r,0,glob

Table 4-13. UART register description

4.2.11 Counter/Timer/PWM registers (h0058-h005F)

Name Address	7	6	5	4	3	2	1	0
RegCntA h0058	CounterA[7:0] s,00000000,glob							
RegCntB h0059	CounterB[7:0] s,00000000,glob							
RegCntC h005A	CounterC[7:0] s,00000000,glob							
RegCntD h005B	CounterD[7:0] s,00000000,glob							
RegCntCtrlCk h005C	CntDCkSel[1:0] rw,00,glob		CntCCkSel[1:0] rw,00,glob		CntBCkSel[1:0] rw,00,glob		CntACkSel[1:0] rw,00,glob	
RegCntConfig1 h005D	CntDDownUp rw,0,glob	CntCDownUp rw,0,glob	CntBDownUp rw,0,glob	CntADownUp rw,0,glob	CascadeCD rw,0,glob	CascadeAB rw,0,glob	CntPWM1 rw,0,glob	CntPWM0 rw,0,glob
RegCntConfig2 h005E	CapSel[1:0] rw,00,glob		CapFunc[1:0] rw,00,glob		Pwm1Size[1:0] rw,00,glob		Pwm0Size[1:0] rw,00,glob	
RegCntOn h005F	CntDExtDiv rw,0,glob	CntCExtDiv rw,0,glob	CntBExtDiv rw,0,glob	CntAExtDiv rw,0,glob	CntDEnable rw,0,glob	CntCEnable rw,0,glob	CntBEnable rw,0,glob	CntAEnable rw,0,glob

Table 4-14. Counter/timer/PWM register description.

4.2.12 RF interface BitJockey (h0060-h0067)

Name Address	7	6	5	4	3	2	1	0
RegRfifCmd1 h0060	r0	r0	RfifBRCoarse[1:0] rw,00,glob		RfifBRFine[3:0] rw,0000,glob			
RegRfifCmd2 h0061	RfifEnStart[1:0] rw,00,glob		RfifEnCod rw,0,glob	RfifRxClock rw,0,glob	RfifTxClock rw,0,glob	RfifPCM[2:0] rw,000,glob		
RegRfifCmd3 h0062	RfifRxIrqEn[2:0] rw,000,glob			RfifRxIrqMem[2:0] rc1,000,glob			RfifEnRx rw,0,conf	RfifEnTx rw,0,conf
RegRfifTx h0063	RfifTx wr0,00000000,glob							
RegRfifTxSta h0064	r0	r0	r0	r0	RfifTxFifoOverrun rc1,0,glob	RfifTxFifoFull r,0,glob	RfifTxFifoEmpty r,1,glob	RfifTxStopped r,0,glob
RegRfifRx h0065	RfifRx r,00000000,glob							
RegRfifRxSta h0066	r0	r0	r0	RfifRxFifoOverrun rc1,0,glob	RfifRxFifoFull r,0,glob	RfifRxStartDet rc1,0,glob	RfifRxBusy r,0,glob	RfifRxReady r,0,glob
RegRfifSPat h0067	RfifSPat[7:0] rw,0000000,glob							

Table 4-15. RF interface (Rfif) register description.

4.2.13 Comparator registers (h0072-h0073)

Name Address	7	6	5	4	3	2	1	0
RegCmpdStat h0072	CmpdStat[3:0] rca,0000,glob				CmpdOut[3:0] r,0000,glob			
RegCmpdCtrl h0073	IrqOnRising[2:0] rw,000,glob			EnIrqCh[3:0] rw,0000,glob				Enable rw,0,glob

Table 4-16. Low power comparator registers

4.2.14 Voltage Level Detector registers (h007E-h007F)

Name Address	7	6	5	4	3	2	1	0
RegVldCtrl h007E	r0	r0	r0	r0	VldRange rw,0,glob	VldTune[2:0] rw,000,glob		
RegVldStat h007F	r0	r0	r0	r0	r0	VldResult r,0,glob	VldValid r,0,glob	VldEn rw,0,glob

Table 4-17. Voltage level detector register description

4.2.15 RAM (h0080-h027F)

The 512 RAM bytes can be accessed for read and write operations. The RAM has no reset function. Variables stored in the RAM should be initialised before use since they can have any value at circuit start up.

5. Low Power Modes

5.1 Features

5.1.1 Overview

The XE8000 chips have three operating modes. These are the normal, low current and very low current modes (see Figure 5-1). The different modes are controlled by the reset and clock blocks (see the documentation of the respective blocks).

5.2 Operating mode

Start-up
All bits are reset in the design when a POR or padnreset is active.
RC is enabled, Xtal is disabled and CPU is reset (pmaddr = 0000).
If the port A is used to return from the sleep mode, all bits with nresetcold do not change (see sleep mode)

Start-up
All bits with nresetglobal and nresetpconf(if enabled) are reset. Clock configuration doesn't change except cpuck (freqdiv is reset, see clock block). CPU is reset

Active mode
This is the mode where the CPU and all peripherals can work and execute the embedded software.

Standby mode
Executing a HALT instruction moves the XE8000 into the Standby mode. The CPU is stopped, but the clocks remain active. Therefore, the enabled peripherals remain active e.g. for time keeping. A reset or an interrupt/event request (if enabled) cancels the standby mode.

Sleep mode
This is a very low-power mode because all circuit clocks and all peripherals are stopped. Only some service blocks remain active. No time-keeping is possible. Two instructions are necessary to move into sleep mode. First, the **SleepEn** (sleep enable) bit in **RegSysCtrl** has to be set to 1. The sleep mode can then be activated by setting the **Sleep** bit in **RegSysReset** to 1.

There are three possibe ways to wake-up from the sleep mode:

1. The POR (power-on-reset caused by a power-down followed by power-on). The RAM information is lost.
2. The padnreset
3. The Port A reset combination (if the Port A is present in the product). See Port A documentation for more details.

Note: If the Port A is used to return from the sleep mode, all bits with nresetcold do not change (**RegSysCtrl**, **RegSysReset** (except bit **sleep**), **Enextclock** and **Biasrc** in **RegSysClock**, **RegSysRcTrim1** and **RegSysRcTrim2**). The **SleepFlag** bit in **RegSysReset**, reads back a 1 if the circuit was in sleep mode since the flag was last cleared (see reset block for more details).

Note: It is recommended to insert a NOP instruction after the instruction that sets the circuit in sleep mode because this instruction can be executed when the sleep mode is left using the resetfromportA.

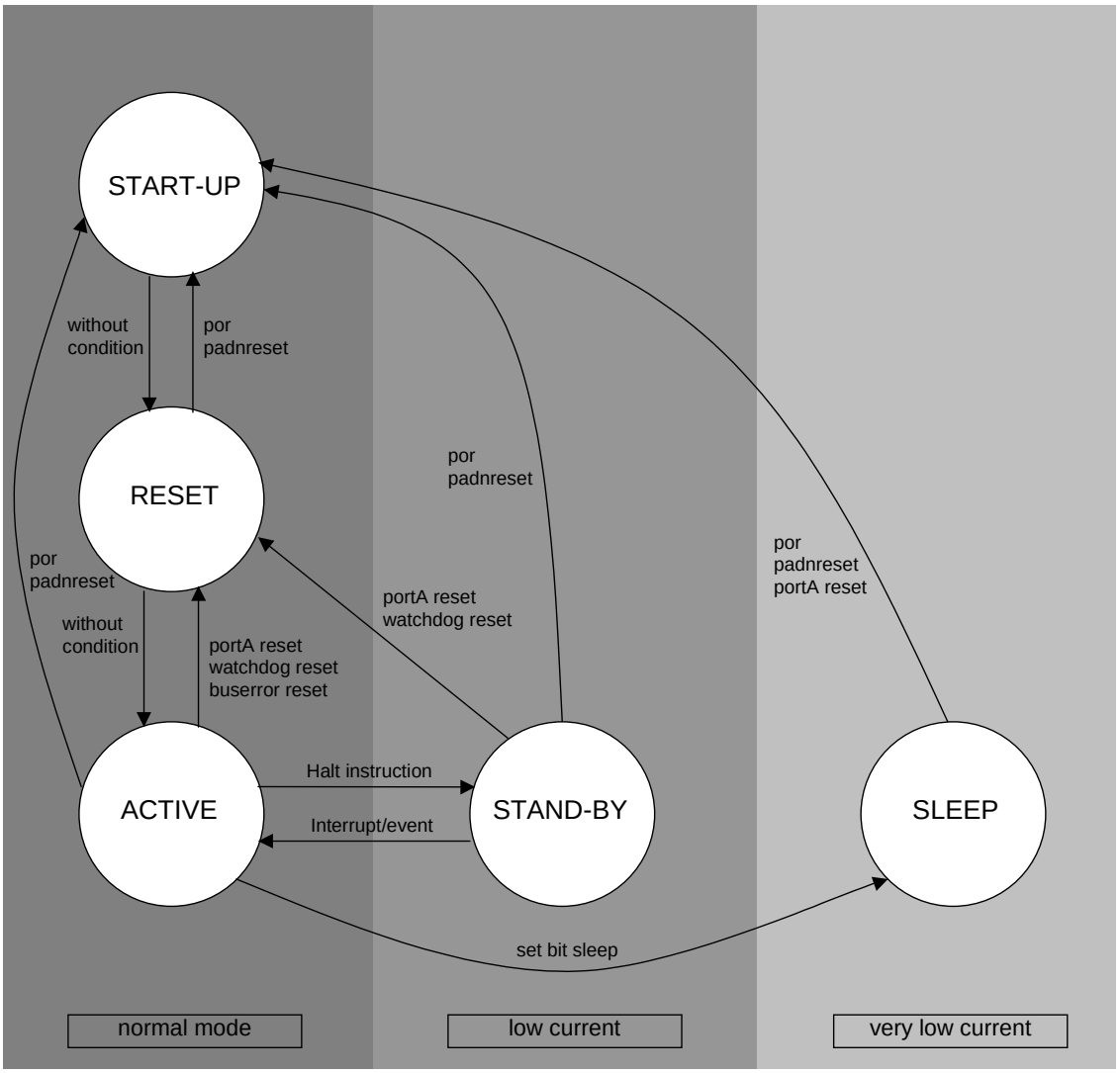


Figure 5-1. XE8000 operating modes.

6. Reset Generator

6.1 Features

- Power On Reset (POR)
- External reset from the NRESET pin
- Programmable Watchdog timer reset
- Programmable BusError reset
- Sleep mode management

Product dependant:
- Programmable Port A input combination reset

6.2 Overview

The reset block is the reset manager. It handles the different reset sources and distributes them through the system. It also controls the sleep mode of the circuit.

6.3 Register map

register name
RegSysCtrl
RegSysReset
RegSysWd

Table 6-1. Reset registers

Table 6-1 gives the different registers used by this block.

Pos.	RegSysCtrl	Rw	Reset	Function
7	SleepEn	r w	0 nresetcold	enables Sleep mode 0: sleep mode is disabled 1: sleep mode is enabled
6	EnResetPConf	r w	0 nresetcold	enables the nresetpconf signal when the nresetglobal is active 0: nresetpconf is disabled 1: nresetpconf is enabled
5	EnBusError	r w	0 nresetcold	enables reset from BusError 0: BusError reset source is disabled 1: BusError reset source is enabled
4	EnResetWD	r w	0 nresetcold	enables reset from Watchdog 0: Watchdog reset source is disabled 1: Watchdog reset source is enabled this bit can not be set to 0 by SW
3 – 0	-	r	0000	unused

Table 6-2. **RegSysCtrl** register.

Pos.	RegSysReset	Rw	Reset	Function
7	Sleep	rw	0 nresetglobal	Sleep mode control (reads always 0)
6	SleepFlag	r c	0 nresetcold	Sleep mode was active before
5	ResetBusError	r c	0 nresetcold	reset source was BusError
4	ResetWD	r c	0 nresetcold	reset source was Watchdog
3	ResetfromportA	r c	0 nresetcold	reset source was Port A combination
2 – 0		r	000	unused

Table 6-3. **RegSysReset** register

Pos.	RegSysWd	Rw	Reset	Function
7 - 4	-	r	0000	unused
3	WDKey[3]	w	0 nresetglobal	Watchdog Key bit 3
	WDCounter[3]	r		Watchdog counter bit 3
2	WDKey[2]	w	0 nresetglobal	Watchdog Key bit 2
	WDCounter[2]	r		Watchdog counter bit 2
1	WDKey[1]	w	0 nresetglobal	Watchdog Key bit 1
	WDCounter[1]	r		Watchdog counter bit 1
0	WDKey[0]	w	0 nresetglobal	Watchdog Key bit 0
	WDCounter[0]	r		Watchdog counter bit 0

Table 6-4. **RegSysWd** register

6.4 Reset handling capabilities

There are 5 reset sources:

- Power On Reset (POR)
- External reset from the NRESET pin
- Programmable port A input combination
- Programmable watchdog timer reset
- Programmable BusError reset on processor access outside the allocated memory map

Another reset source is the bit **Sleep** in the **RegSysReset** register. This source is fully controlled by software and is only used during the sleep mode.

Four internal reset signals are generated from these sources and distributed through the system:

- nresetcold: is asserted on POR or by the NRESET pin
- nresetglobal: is asserted when nresetcold or any other enabled reset source is active
- nresetsleep: is asserted when the circuit is in sleep mode
- nresetpconf: is asserted when nresetglobal is active and if the **EnResetPConf** bit in the **RegSysCtrl** register is set. This reset is generally used in the different ports. It allows to maintain the port configuration unchanged while the rest of the circuit is reset.

Table 6-5 shows a summary of the dependency of the internal reset signals on the various reset sources.

In all the tables describing the different registers, the reset source is indicated.

Asserted reset source	Internal reset signals				
	nresetglobal	nresetpconf		nresetsleep	nresetcold
		when EnResetPConf is set to 0	when EnRestPConf is set to 1		
POR	Asserted	Asserted	Asserted	Asserted	Asserted
NRESET pin	Asserted	Asserted	Asserted	Asserted	Asserted
PortA input	Asserted	-	Asserted	-	-
Watchdog	Asserted	-	Asserted	-	-
BusError	Asserted	-	Asserted	-	-
Sleep	-	-	-	Asserted	-

Table 6-5. Internal reset assertion as a function of the reset source.

6.5 Reset source description

6.5.1 Power On Reset

The power on reset (POR) monitors the external supply voltage. It activates a reset on a rising edge of this supply voltage. The reset is inactivated only if the internal voltage regulator has started up. The POR block performs no precise voltage level detection.

6.5.2 NRESET pin

Applying a low input state on the NRESET pin can activate the reset.

6.5.3 Programmable Port A input combination

Port A (if present in the product) can generate a reset signal. See the description of the Port A for further information.

6.5.4 Watchdog reset

The Watchdog will generate a reset if the **EnResetWD** bit in the **RegSysCtrl** register has been set and if the watchdog is not cleared in time by the processor. See chapter 6.8 describing the watchdog for further information.

6.5.5 BusError reset

The address space is assigned as shown in the register map of the product. If the **EnBusError** bit in the **RegSysCtrl** register is set and the software accesses an unused address, a reset is generated.

6.6 Sleep mode

Entering the sleep mode will reset a part of the circuit. The reset is used to configure the circuit for correct wake-up after the sleep mode. If the **SleepEn** bit in the **RegSysCtrl** register has been set, the sleep mode can be entered by setting the bit **Sleep** in **RegSysReset**. During the sleep mode, the nresetsleep signal is active. For detailed information on the sleep mode, see the system documentation.

6.7 Control register description and operation

Two registers are dedicated for reset status and control, **RegSysReset** and **RegSysCtrl**. The bits **Sleep, SleepFlag** and **SleepEn** are also located in those registers and are described in the chapter dedicated to the different operating modes of the circuit (system block).

The **RegSysReset** register gives information on the source that generated the last reset. It can be read at the beginning of the application program to detect if the circuit is recovering from an error or exception condition, or if the circuit is starting up normally.

- when **ResetBusError** is 1, a forbidden address access generated the reset.
- when **ResetWD** is 1, the watchdog generated the reset.
- when **ResetfromPortA** is 1, a PortA combination generated the reset.

Note: If no bit is set to 1, the reset source was either the NRESET pin or the internal POR.
Note: Several bits might be set or not, if the register was not cleared in between 2 reset occurrences.

The two other bits concern the sleep mode control and information (see system documentation for the sleep mode description).

- When **SleepFlag** is 1, the sleep mode was active before the reset occurred. This bit will always appear together with the **ResetfromPortA** bit since all other possibilities to leave the sleep mode (POR and NRESET pin) will clear the **SleepFlag**.
- When **Sleep** is set to 1, and **SleepEn** is 1, the sleep mode is entered. The bit always reads back a 0.

The **RegSysCtrl** register enables the different available reset sources and the sleep mode.

- **EnBusError** enables the reset due to a bus error condition.
- **EnResetWD** enables the reset due to the watchdog (can not be disabled once enabled).
- **EnResetPConf** enables the reset of the port configurations when reset by Port A, a Bus Error or the watchdog.
- **SleepEn** unlocks the **Sleep** bit. As long as **SleepEn** is 0, the **Sleep** bit has no effect.

6.8 Watchdog

The watchdog is a timer, which has to be cleared at least every 2 seconds by the software to prevent a reset to be generated by the timeout condition.

The watchdog can be enabled by software by setting the **EnResetWD** bit in the **RegSysCtrl** register to 1. It can then only be disabled by a power on reset or by setting the NRESET pin to a low state.

The watchdog timer can be cleared by writing consecutively the values Hx0A and Hx03 to the **RegSysWd** register. The sequence must strictly be respected to clear the watchdog.

In assembler code, the sequence to clear the watchdog is:

```
move AddrRegSysWd, #0x0A
move AddrRegSysWd, #0x03
```

Only writing Hx0A followed by Hx03 resets the WD. If some other write instruction is done to the **RegSysWd** between the writing of the Hx0A and Hx03 values, the watchdog timer will not be cleared.

It is possible to read the status of the watchdog in the **RegSysWd** register. The watchdog is a 4 bit counter with a count range between 0 and 7. The system reset is generated when the counter is reaching the value 8.

6.9 Start-up and watchdog specifications

At start-up of the circuit, the POR block generates a reset signal during t_{POR}. The circuit starts software execution after this period (see system chapter). The POR is intended to force the circuit into a correct state at start-up. For precise monitoring of the supply voltage, the voltage level detector (VLD) has to be used.

Symbol	Parameter	Min	Typ	Max	Unit	Comments
T_{POR}	POR reset duration	5		20	ms	
Vbat_sl	Supply ramp up	0.5			V/ms	1
WDtime	Watchdog timeout period	2			s	2

Table 6. Electrical and timing specifications

Note: 1) The Vbat_sl defines the minimum slope required on VBAT. Correct start-up of the circuit is not guaranteed if this slope is too slow. In such a case, a delay has to be built using the NRESET pin.

Note: 2) The minimal watchdog timeout period is guaranteed when the internal oscillators are used. In case an external clock source is used, the watchdog timeout period will be correct in so far the contents of the **RegSysRCTrim1** and **RegSysRCTrim2** registers are correct (see clock block documentation for more details).

7. Clock Generation

7.1 Features

3 available clock sources (RC oscillator, quartz oscillator and external clock).
- 2 divider chains: high-prescaler (8 bits) and low-prescaler (15 bits).
- CPU clock disabling in halt mode.

7.2 Overview

The XE88LCxx chips can work on different clock sources (RC oscillator, quartz oscillator and external clock). The clock generator block is in charge of distributing the necessary clock frequencies to the circuit.

Figure 7-1 represents the functionality of the clock block.

The internal RC oscillator or an external clock source can be selected to drive the high prescaler. This prescaler generates frequency divisions down to 1/256 of its input frequency. A 32kHz clock is generated by enabling the quartz oscillator (if present in the product) or by selecting the appropriate tap on the high prescaler. The low prescaler generates clock signals from 32kHz down to 1Hz. The clock source for the CPU can be selected from the RC oscillator, the external clock or the 32kHz clock.

7.3 Register map

pos.	RegSysClock	rw	reset	function
7	CpuSel	r/w	0 nresetsleep	Select speed for cpuck
6	-	r	0	Unused
5	EnExtClock	r/w	0 nresetcold	Enable for external clock
4	BiasRc	r/w	1 nresetcold	Enable Rcbias (reduces start-up time of RC).
3	ColdXtal	r	1 nresetsleep	Xtal in start phase
2	-	r	0	Unused
1	EnableXtal	r/w	0 nresetsleep	Enable Xtal oscillator
0	EnableRc	r/w	1 nresetsleep	Enable RC oscillator

Table 7-1: **RegSysClock** register

pos.	RegSysMisc	rw	reset	function
7-2	--	r	000000	Unused
1	Output16k	r/w	0 nresetsleep	Output 16 kHz signal on PB[3]
0	OutputCpuCk	r/w	0 nresetsleep	Output CPU clock on PB[2]

Table 7-2: **RegSysMisc** register

pos.	RegSysPre0	rw	reset	function
7-1	--	r	0000000	Unused
0	ClearLowPrescal	w1 r0	0	Write 1 to reset low prescaler, but always reads 0

Table 7-3: **RegSysPre0** register

pos.	RegSysRcTrim1	rw	reset	function
7-5	--	r	000	Unused
4	RcFreqRange	r/w	0 nresetcold	Low/high freq. range (low=0)
3	RcFreqCoarse[3]	r/w	0 nresetcold	RC coarse trim bit 3
2	RcFreqCoarse[2]	r/w	0 nresetcold	RC coarse trim bit 2
1	RcFreqCoarse[1]	r/w	0 nresetcold	RC coarse trim bit 1
0	RcFreqCoarse[0]	r/w	1 nresetcold	RC coarse trim bit 0

Table 7-4: **RegSysRCTrim1** register

pos.	RegSysRcTrim2	rw	reset	function
7-6	--	r	00	Unused
5	RcFreqFine[5]	r/w	0 nresetcold	RC fine trim bit 5
4	RcFreqFine[4]	r/w	0 nresetcold	RC fine trim bit 4
3	RcFreqFine[3]	r/w	0 nresetcold	RC fine trim bit 3
2	RcFreqFine[2]	r/w	0 nresetcold	RC fine trim bit 2
1	RcFreqFine[1]	r/w	0 nresetcold	RC fine trim bit 1
0	RcFreqFine[0]	r/w	0 nresetcold	RC fine trim bit 0

Table 7-5: **RegSysRCTrim2** register

pos.	RegSysPtckmode	rw	reset	function
7-1	--	r	0000000	Unused
0	Reserved	r/w	0 nresetglobal	Reserved

Table 7-6: **RegSysPtckmode** register

7.4 Interrupts and events map

interrupt source	Default mapping in the interrupt manager	Default mapping in the event manager
ck128Hz	RegIrqHig(6)	RegEvn(5)
ck1Hz	RegIrqMid(3)	RegEvn(1)

Table 7-7: Interrupts and events map

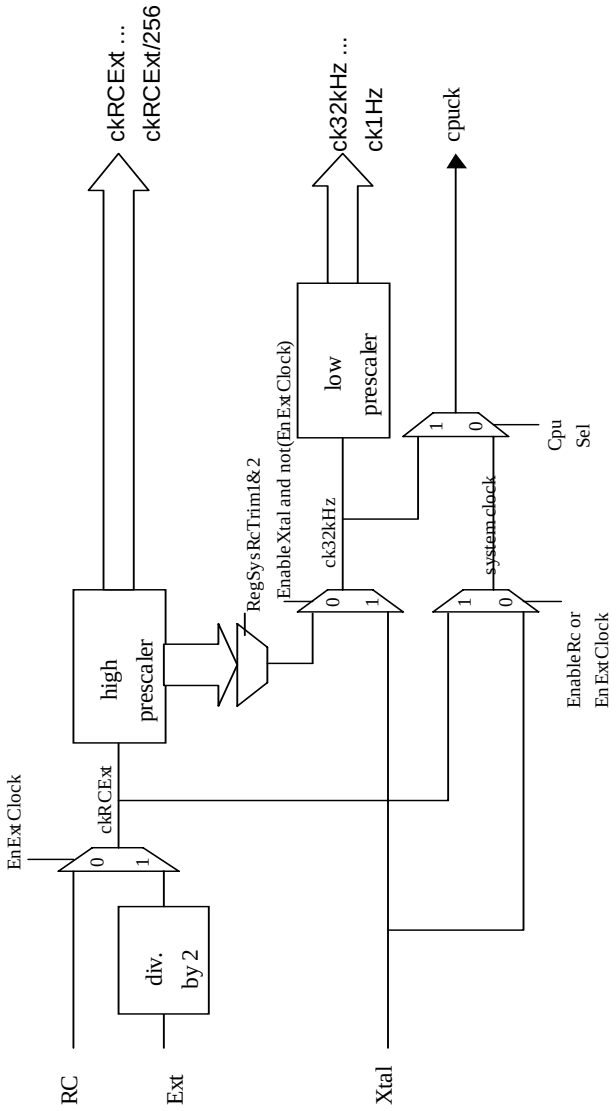


Figure 7-1. Clock block structure

7.5 Clock sources

7.6 RC oscillator

7.6.1 Configuration

The RC oscillator is always turned on and selected for CPU and system operation at power-on reset, pad NRESET, and when exiting sleep mode. It can be turned off after the Xtal (quartz oscillator) has been started, after selection of the external clock or by entering sleep mode.

The RC oscillator has two frequency ranges: sub-MHz (50 kHz to 0.5 MHz) and above-MHz (0.5 MHz to 5 MHz). Inside a range, the frequency can be tuned by software for coarse and fine adjustment. See registers **RegSysRcTrim1** and **RegSysRcTrim2**.

Bit **EnableRc** in register **RegSysClock** controls the propagation of the RC clock signal and the operation of the oscillator. The user can stop the RC oscillator by resetting the bit **EnableRc**. Entering the sleep mode disables the RC oscillator.

Note: The RC oscillator bias can be maintained while the oscillator is disabled by setting the bit **BiasRc** in **RegSysClock**. This allows a faster restart of the RC oscillator at the cost of increased power consumption (see section 7.6.3).

7.6.2 RC oscillator frequency tuning

The RC oscillator frequency can be set using the bits in the **RegSysRcTrim1** and **RegSysRcTrim2** registers. Figure 7-2 shows the nominal frequency of the RC oscillator as a function of these bits. The absolute value of the frequency for a given register content may change by ±35% from chip to chip due to the tolerances on the integrated capacitors and resistors. However, the modification of the frequency as a function of a modification of the register content is fairly precise. This means that the curves in Figure 7-2 can shift up and down but that the slope remains unchanged.

The bit **RcFreqRange** modifies the oscillator frequency by a factor of 10. The upper curve in the figure corresponds to **RcFreqRange**=1.

The **RcFreqCoarse** modifies the frequency of the oscillator by a factor (**RcFreqCoarse**+1). The figure represents the frequency for 5 different values of the bits **RcFreqCoarse**: for each value the frequency is multiplied by 2.

Incrementing the **RcFreqFine** code, increases the frequency by about 1.4%.

The frequency of the oscillator is therefor given by:

$$f_{RC}=f_{Rcmin}\cdot(1+9\cdot\mathbf{RcFreqRange})\cdot(1+\mathbf{RcFreqCoarse})\cdot(1.014)^{\mathbf{RcFreqFine}}$$

with f_{Rcmin} the RC oscillator frequency if the registers are all 0.

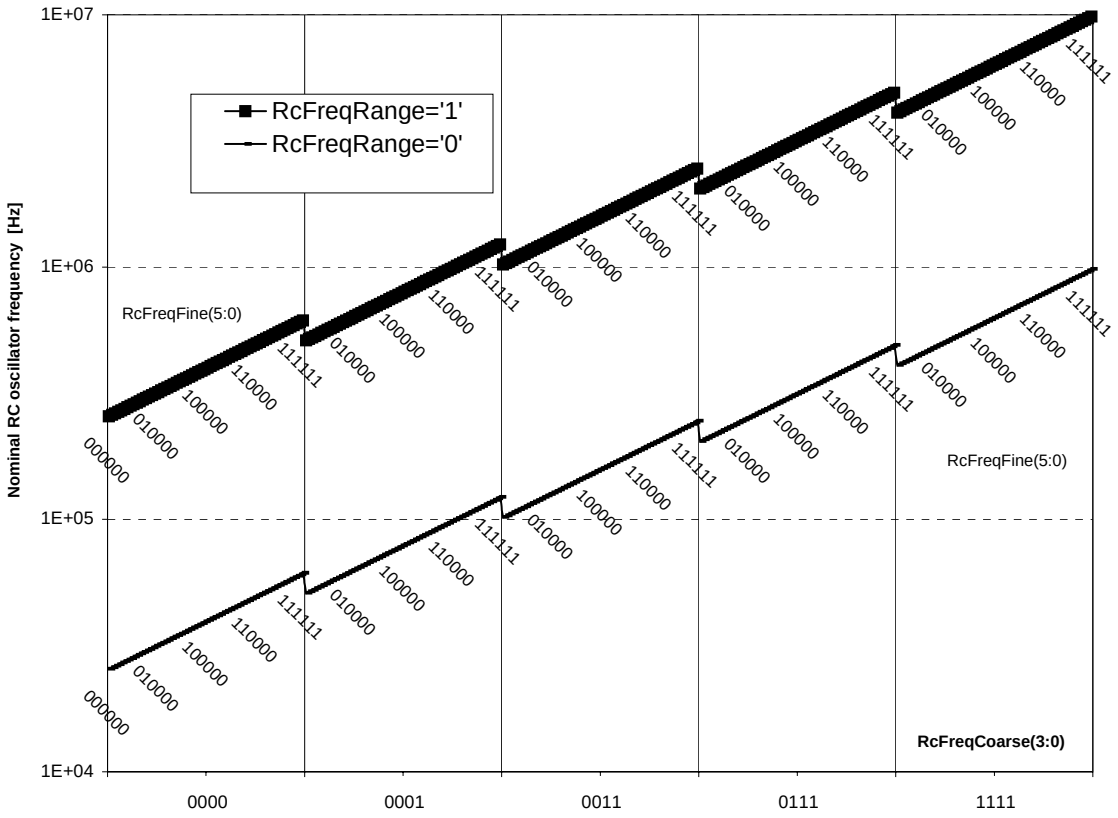


Figure 7-2. RC oscillator nominal frequency tuning.

7.6.3 RC oscillator specifications

sym	description	min	typ	max	unit	Comments
f_{RCmin}	Lowest RC frequency	25	40	55	kHz	Note 1
RcFreqFine	fine tuning step		1.4	2.0	%	
RC_su	startup time		30	50	us	**BiasRc**=0
			3	5	us	**BiasRc**=1
PSRR @ DC	Supply voltage dependence		TBD		%/V	Note 2
			TBD		%/V	Note 3
$\Delta f/\Delta T$	Temperature dependence		0.1		%/°C	

Table 7-8. RC oscillator specifications

Note 1: this is the frequency tolerance when all trimming codes are 0. The frequency at start-up is about twice as high.

Note 2: frequency shift as a function of VBAT with normal regulator function.

Note 3: frequency shift as a function of VBAT while the regulator is short-circuited to VBAT.

The tolerances on the minimal frequency and the drift with supply or temperature can be cancelled using the software or hardware DFLL (digital frequency locked loop) which uses the crystal oscillator as a reference frequency.

7.7 Xtal oscillator

7.7.1 Xtal configuration

The Xtal operates with an external crystal of 32'768 Hz. During Xtal oscillator start-up, the first 32768 cycles are masked. The two bits **EnableXtal** and **ColdXtal** in register **RegSysClock** control the oscillator.

At power-on reset, a pad NRESET pulse or during sleep mode, **EnableXtal** is reset and **ColdXtal** is set (Xtal oscillator is not selected at start-up). The user can start Xtal oscillator by setting **EnableXtal**. When the Xtal oscillator starts, bit **ColdXtal** is reset after 32768 cycles. Before **ColdXtal** is reset by the system, the Xtal frequency precision is not guaranteed. The Xtal oscillator can be stopped by the user by resetting bit **EnableXtal**.

When the user enters into sleep mode, the Xtal is stopped.

7.7.2 Xtal oscillator specifications

The crystal oscillator has been designed for a crystal with the specifications given in Table 7-9. The oscillator precision can only be guaranteed for this crystal.

Symbol	Description	Min	Typ	Max	Unit	Comments
Fs	Resonance frequency		32768		Hz	
CL	CL for nominal frequency		8.2	15	pF	
Rm	Motional resistance		40	100	kΩ	
Cm	Motional capacitance	1.8	2.5	3.2	fF	
C0	Shunt capacitance	0.7	1.1	2.0	pF	
Rmp	Motional resistance of 6^{th} overtone (parasitic)	4	8		kΩ	
Q	Quality factor	30k	50k	400k	-	

Table 7-9. Crystal specifications.

For safe operation, low power consumption and to meet the specified precision, careful board layout is required:

- Keep lines XIN and XOUT short and insert a VSS line in between them.
- Connect the crystal package to VSS.
- No noisy or digital lines near XIN or XOUT.
- Insert guards where needed.
- Respect the board specifications of Table 7-10.

Symbol	Description	Min	Typ	Max	Unit	Comments
Rh_xin	Resistance XIN-VSS	10			MΩ	
Rh_xout	Resistance XOUT-VSS	10			MΩ	
Rh_xin_xout	Resistance XIN-XOUT	50			MΩ	
Cp_xin	Capacitance XIN-VSS	0.5		3.0	pF	
Cp_xout	Capacitance XOUT-VSS	0.5		3.0	pF	
Cp_xin_xout	Capacitance XIN-XOUT	0.2		1.0	pF	

Table 7-10. Board layout specifications.

The oscillator characteristics are given in Table 7-11. The characteristics are valid only if the crystal and board layout meet the specifications above.

Symbol	Description	Min	Typ	Max	Unit	Comments
f_{Xtal}	Nominal frequency		32768		Hz	
St_xtal	Start-up time		1	2	s	
Fstab	Frequency deviation	-100		300	ppm	Note 1

Table 7-11. Crystal oscillator characteristics.

Note 1. This gives the relative frequency deviation from nominal for a crystal with CL=8.2pF and within the temperature range -40°C to 85°C. The crystal tolerance, crystal aging and crystal temperature drift are not included in this figure.

7.8 External clock

7.8.1 External clock configuration

The user can provide an external clock instead of the internal oscillators. The external provided frequency is internally divided by two. The external clock input pin is XIN.

The system is configured for external clock by bit **EnExtClock** in register **RegSysClock**. Using the bits in the registers **RegSysRcTrim1** and **RegSysRcTrim2**, the ck32kHz clock frequency can be controlled (see section 7.11).

Note: when using the external clock, the Xtal is not available.

7.8.2 External clock specification

The external clock has to satisfy the specifications in the table below. Correct behavior of the circuit can not be guaranteed if the external clock signal does not respect the specifications below.

Symbol	Description	Min	Typ	Max	Unit	Comments
F_{EXT}	External clock frequency			8	MHz	Note 1
PW_1	Pulse 1 width	0.06			µs	Note 1
PW_0	Pulse 0 width	0.03		20	µs	Note 1
F_{EXT_LV}	External clock frequency			TBD	kHz	Note 2
PW_1_LV	Pulse 1 width	TBD			µs	Note 2
PW_0_LV	Pulse 0 width	TBD		20	µs	Note 2

Table 7-12. External clock specifications.

Note 1. For VBAT≥2.4V

Note 2. For VBAT=VREG=1.2V

7.9 Clock source selection

There are three possible clock sources available for the CPU clock. The RC clock is always selected after power-up, a negative pulse on NRESET or after Sleep mode. The CPU clock selection is done with the bit **CpuSel** in **RegSysClock** (0= fastest clock, 1= 32 kHz from Xtal if **EnableXtal** =1 and **EnExtClock** = 0 else from high prescaler 32 kHz output).

Switching from one clock source to another is glitch free.

The next table summarizes the different clock configurations of the circuit:

Clock Sources				Clock targets			
Mode name	**EnExtClock**	**EnableRc**	**EnableXtal**	Cpuck		High Prescaler Clock input	Low Prescaler Clock input
				CpuSel=0	**CpuSel**=1		
Sleep	0	0	0	off	off	off	off
Xtal	0	0	1	Xtal	Xtal	off	Xtal
RC	0	1	0	RC NOTE 2	High presc.	RC	High presc.
RC + Xtal	0	1	1	RC NOTE 1 and 2	Xtal	RC NOTE 1	Xtal
External	1	X	X	External NOTE 2	High presc.	External	High presc.

Table 7-13: Table of clocking modes.

Note 1: The frequency of the RC must be higher than 100 kHz when Xtal is enabled in order to ensure a proper 32 kHz operation.

Note 2: The clock RC can be divided by the value of freq instruction (see coolrisc instruction information)

freq instruction	cpuck
nodiv	RC or external
div2	RC/2 or external/2
div4	RC/4 or external/4
div8	RC/8 or external/8
div16	RC/16 or external/16

Note 3: Switching from one clock source to another and stopping the unused clock source must be performed using 2 MOVE instructions to **RegSysClock.** First select the new clock source and then stop the unused one.

7.10 Prescalers

The clock generator block embeds two divider chains: the high prescaler and the low one.

The high prescaler is made of an 8 stage dividing chain and the low prescaler of a 15 stage dividing chain.

Features:

- High prescaler can only be driven with RC clock or external clock (bits **EnableRc** or **EnExtClock** have to be set, see Table 7-13).
- Low prescaler can be driven from the high prescaler or directly with the Xtal clock when bit **EnableXtal** is set to 1 and bit **EnExtClock** is set to 0.
- Bit **ClearLowPrescal** in the **RegSysPre0** register allows to reset synchronously the low prescaler, the low prescaler is also automatically cleared when bit **EnableXtal** is set. Both dividing chains are reset asynchronously by the *nresetglobal* signal.
- Bit **ColdXtal**=1 indicates the Xtal is in its start phase. It is active for 32768 Xtal cycles after setting **EnableXtal**.

7.11 32 kHz frequency selector

A decoder is used to select from the high prescaler, the frequency tap that is the closest to 32 kHz to operate the low prescaler when the Xtal is not running. In this case, the RC oscillator frequency of ±35% will also be valid for the low prescaler frequency outputs.

The next table shows how the RC trimming values in the **RegSysRcTrim1** and **RegSysRcTrim2** registers select the 32 kHz frequency. The least significant bits of the **RcFreqFine** word are not used.

In order to ensure the correct frequency selection for the low prescaler when having an external clock, a proper value must be set in the RC trim registers. The code can be selected from the table below as a function of the frequency ratio between half the frequency of the external clock and 32kHz. If the frequency is not set correctly, all timings derived from the low prescaler will be shifted accordingly (e.g. watchdog frequencies) and some peripherals may no longer function correctly if the deviation from 32kHz is too large (e.g. the voltage level detector).

RcFreqRange&RcFreqCoarse(3:0)&RcFreqFine(5:3)	Selected high prescaler tap
Default case (0'0001'000)	Ckrcext/2
From 0'0000'000 to 0'0000'100	Ckrcext
From 0'0000'101 to 0'0001'100	Ckrcext/2
From 0'0001'101 to 0'0001'111	Ckrcext/4
0'0010'000	Ckrcext/2
From 0'0010'001 to 0'0010'110	Ckrcext/4
0'0010'111	Ckrcext/8
From 0'0011'000 to 0'0011'100	Ckrcext/4
From 0'0011'101 to 0'0011'111	Ckrcext/8
From 0'0100'000 to 0'1000'010	Ckrcext/4
From 0'0100'011 to 0'0100'111	Ckrcext/8
0'0101'000	Ckrcext/4
From 0'0101'001 to 0'0101'110	Ckrcext/8
0'0101'111	Ckrcext/16
From 0'0110'000 to 0'0110'101	Ckrcext/8
From 0'0110'110 to 0'0110'111	Ckrcext/16
From 0'0111'000 to 0'0111'100	Ckrcext/8
From 0'0111'101 to 0'0111'111	Ckrcext/16
From 0'1000'000 to 0'1000'011	Ckrcext/8
From 0'1000'100 to 0'1000'111	Ckrcext/16
From 0'1001'000 to 0'1001'010	Ckrcext/8
From 0'1001'011 to 0'1001'111	Ckrcext/16
From 0'1010'000 to 0'1010'001	Ckrcext/8
From 0'1010'010 to 0'1010'111	Ckrcext/16
0'1011'000	Ckrcext/8
From 0'1011'001 to 0'1011'110	Ckrcext/16
0'1011'111	Ckrcext/32
From 0'1100'000 to 0'1100'110	Ckrcext/16
0'1100'111	Ckrcext/32
From 0'1101'000 to 0'1101'101	Ckrcext/16
From 0'1101'110 to 0'1101'111	Ckrcext/32
From 0'1110'000 to 0'1110'100	Ckrcext/16
From 0'1110'101 to 0'1110'111	Ckrcext/32
From 0'1111'000 to 0'1111'100	Ckrcext/16
From 0'1111'101 to 0'1111'111	Ckrcext/32
From 1'0000'000 to 1'0000'010	Ckrcext/8
From 1'0000'011 to 1'0001'010	Ckrcext/16
From 1'0001'011 to 1'0010'100	Ckrcext/32
From 1'0010'101 to 1'0010'111	Ckrcext/64
From 1'0011'000 to 1'0011'010	Ckrcext/32
From 1'0011'011 to 1'0011'111	Ckrcext/64
1'0100'000	Ckrcext/32
From 1'0100'001 to 1'0100'110	Ckrcext/64
1'0100'111	Ckrcext/128
From 1'0101'000 to 1'0101'100	Ckrcext/64
From 1'0101'101 to 1'0101'111	Ckrcext/128
From 10110'000 to 1'0110'011	Ckrcext/64
From 1'0110'100 to 1'0110'111	Ckrcext/128
From 1'0111'000 to 1'0111'010	Ckrcext/64
From 1'0111'011 to 1'0111'111	Ckrcext/128
From 1'1000'000 to 1'1000'001	Ckrcext/64
From 1'1000'010 to 1'1000'111	Ckrcext/128
1'1001'000	Ckrcext/64
From 1'1001'001 to 1'1111'111	Ckrcext/128

Table 7-14: Table of 32kHz high prescaler tap decoder.

8. Interrupt Handler

8.1 Features

The XE8000 chips support 24 interrupt sources, divided into 3 levels of priority.

8.2 Overview

The interrupt handler allows 24 interrupt sources to be managed individually.

The 24 interrupt sources are divided into 3 levels of priority: High (8 interrupt sources), Mid (8 interrupt sources), and Low (8 interrupt sources). Those 3 levels of priority are directly mapped to those supported by the CoolRisc® (IN0, IN1 and IN2; see CoolRisc documentation for more information).

Additional functions are given that allow fast detection of the highest priority interrupt that has been activated.

8.3 Register map

Register name
RegIrqHig
RegIrqMid
RegIrqLow
RegIrqEnHig
RegIrqEnMid
RegIrqEnLow
RegIrqPriority
RegIrqIrq

Table 8-1: IRQ handler registers

pos.	RegIrqHig	rw	reset	function
7	RegIrqHig[7]	r c1	0 nresetglobal	interrupt #23 (high priority) clear interrupt #23 when 1 is written
6	RegIrqHig[6]	r c1	0 nresetglobal	interrupt #22 (high priority) clear interrupt #22 when 1 is written
5	RegIrqHig[5]	r c1	0 nresetglobal	interrupt #21 (high priority) clear interrupt #21 when 1 is written
4	RegIrqHig[4]	r c1	0 nresetglobal	interrupt #20 (high priority) clear interrupt #20 when 1 is written
3	RegIrqHig[3]	r c1	0 nresetglobal	interrupt #19 (high priority) clear interrupt #19 when 1 is written
2	RegIrqHig[2]	r c1	0 nresetglobal	interrupt #18 (high priority) clear interrupt #18 when 1 is written
1	RegIrqHig[1]	r c1	0 nresetglobal	interrupt #17 (high priority) clear interrupt #17 when 1 is written
0	RegIrqHig[0]	r c1	0 nresetglobal	interrupt #16 (high priority) clear interrupt #16 when 1 is written

Table 8-2: **RegIrqHig**

pos.	RegIrqMid	rw	reset	function
7	RegIrqMid[7]	r c1	0 nresetglobal	interrupt #15 (mid priority) clear interrupt #15 when 1 is written
6	RegIrqMid[6]	r c1	0 nresetglobal	interrupt #14 (mid priority) clear interrupt #14 when 1 is written
5	RegIrqMid[5]	r c1	0 nresetglobal	interrupt #13 (mid priority) clear interrupt #13 when 1 is written
4	RegIrqMid[4]	r c1	0 nresetglobal	interrupt #12 (mid priority) clear interrupt #12 when 1 is written
3	RegIrqMid[3]	r c1	0 nresetglobal	interrupt #11 (mid priority) clear interrupt #11 when 1 is written
2	RegIrqMid[2]	r c1	0 nresetglobal	interrupt #10 (mid priority) clear interrupt #10 when 1 is written
1	RegIrqMid[1]	r c1	0 nresetglobal	interrupt #9 (mid priority) clear interrupt #9 when 1 is written
0	RegIrqMid[0]	r c1	0 nresetglobal	interrupt #8 (mid priority) clear interrupt #8 when 1 is written

Table 8-3: **RegIrqMid**

pos.	RegIrqLow	rw	reset	function
7	RegIrqLow[7]	r c1	0 nresetglobal	interrupt #7 (low priority) clear interrupt #7 when 1 is written
6	RegIrqLow[6]	r c1	0 nresetglobal	interrupt #6 (low priority) clear interrupt #6 when 1 is written
5	RegIrqLow[5]	r c1	0 nresetglobal	interrupt #5 (low priority) clear interrupt #5 when 1 is written
4	RegIrqLow[4]	r c1	0 nresetglobal	interrupt #4 (low priority) clear interrupt #4 when 1 is written
3	RegIrqLow[3]	r c1	0 nresetglobal	interrupt #3 (low priority) clear interrupt #3 when 1 is written
2	RegIrqLow[2]	r c1	0 nresetglobal	interrupt #2 (low priority) clear interrupt #2 when 1 is written
1	RegIrqLow[1]	r c1	0 nresetglobal	interrupt #1 (low priority) clear interrupt #1 when 1 is written
0	RegIrqLow[0]	r c1	0 nresetglobal	interrupt #0 (low priority) clear interrupt #0 when 1 is written

Table 8-4: **RegIrqLow**

pos.	RegIrqEnHig	rw	reset	function
7	RegIrqEnHig[7]	rw	0	1= enable interrupt #23
6	RegIrqEnHig[6]	rw	0	1= enable interrupt #22
5	RegIrqEnHig[5]	rw	0	1= enable interrupt #21
4	RegIrqEnHig[4]	rw	0	1= enable interrupt #20
3	RegIrqEnHig[3]	rw	0	1= enable interrupt #19
2	RegIrqEnHig[2]	rw	0	1= enable interrupt #18
1	RegIrqEnHig[1]	rw	0	1= enable interrupt #17
0	RegIrqEnHig[0]	rw	0	1= enable interrupt #16

Table 8-5: **RegIrqEnHig**

pos.	RegIrqEnMid	rw	reset	function
7	RegIrqEnMid[7]	rw	0	1= enable interrupt #15
6	RegIrqEnMid[6]	rw	0	1= enable interrupt #14
5	RegIrqEnMid[5]	rw	0	1= enable interrupt #13
4	RegIrqEnMid[4]	rw	0	1= enable interrupt #12
3	RegIrqEnMid[3]	rw	0	1= enable interrupt #11
2	RegIrqEnMid[2]	rw	0	1= enable interrupt #10
1	RegIrqEnMid[1]	rw	0	1= enable interrupt #9
0	RegIrqEnMid[0]	rw	0	1= enable interrupt #8

Table 8-6: **RegIrqEnMid**

pos.	RegIrqEnLow	rw	reset	function
7	RegIrqEnLow[7]	rw	0	1= enable interrupt #7
6	RegIrqEnLow[6]	rw	0	1= enable interrupt #6
5	RegIrqEnLow[5]	rw	0	1= enable interrupt #5
4	RegIrqEnLow[4]	rw	0	1= enable interrupt #4
3	RegIrqEnLow[3]	rw	0	1= enable interrupt #3
2	RegIrqEnLow[2]	rw	0	1= enable interrupt #2
1	RegIrqEnLow[1]	rw	0	1= enable interrupt #1
0	RegIrqEnLow[0]	rw	0	1= enable interrupt #0

Table 8-7: **RegIrqEnLow**

pos.	RegIrqPriority	rw	reset	function
7-0	RegIrqPriority	r	11111111	code of highest priority set

Table 8-8: **RegIrqPriority**

pos.	RegIrqIrq	rw	reset	function
7-3	-	r	00000	unused
2	IrqHig	r	0	one or more high priority interrupts is set
1	IrqMid	r	0	one or more mid priority interrupts is set
0	IrqLow	r	0	one or more low priority interrupts is set

Table 8-9: **RegIrqIrq**

8.4 Detailed description

The CoolRISC core has 3 different interrupt levels IN0, IN1 and IN2 (Figure 8-1). When these interrupts are triggered, the program counter (PC) is loaded with a fixed address. In case more than one interrupt occurs simultaneously, the execution order is IN0, IN1, IN2.

The masking, setting and clearing of these interrupts can be done in the stat register (see chapter describing the CPU).

The interrupt handler bundles a certain number of interrupt sources and routes them to one of these three interrupts and provides the possibility to enable/disable each of them individually. The definition of the interrupt sources is given in the memory mapping chapter.

RegIrqHig, RegIrqMid, and RegIrqLow are 8-bit registers containing flags for the interrupt sources. Those flags are set when the interrupt is enabled (i.e. if the corresponding bit in the registers **RegIrqEnHig**, **RegIrqEnMid** or **RegIrqEnLow** is set) and a rising edge is detected on the corresponding interrupt source.

Once memorized, an interrupt flag can be cleared by writing a '1' in the corresponding bit of **RegIrqHig**, **RegIrqMid** or **RegIrqLow**. Writing a '0' does not modify the flag. To definitively clear the interrupt, one has to clear the CoolRISC interrupt in the CoolRISC stat register. All interrupts are automatically cleared after a reset.

Two registers are provided to facilitate the writing of interrupt service software. **RegIrqPriority** contains the number of the highest priority set (its value is 0xFF when no interrupt is memorized). **RegIrqIrq** indicates the priority level of the currently activated interrupts.

All interrupt sources are sampled by the highest frequency in the system. A CPU interruption is generated and memorized when an interrupt becomes high. Between the rising edge of the interrupt on the peripheral and the rising edge on the CoolRISC core, there is a latency of one clock cycle.

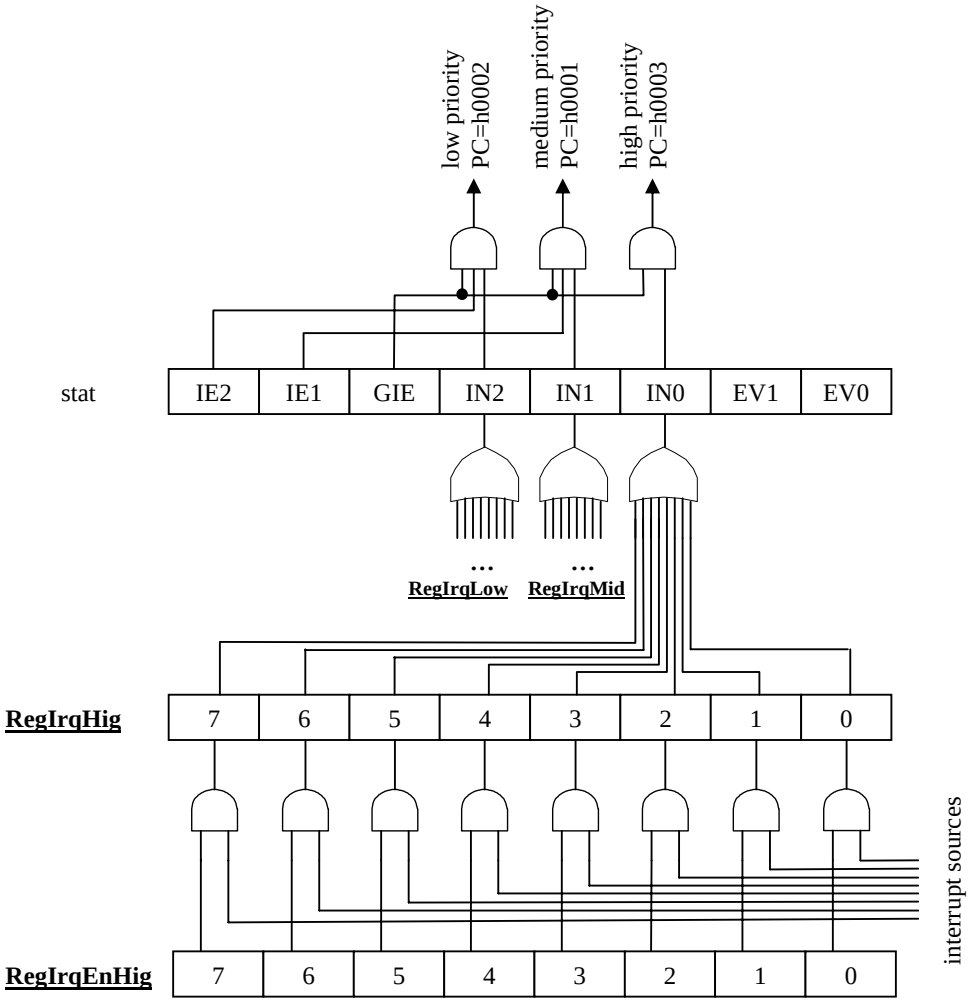


Figure 8-1. Principle of the interrupt handler.

8.5 Interrupt handling software

This chapter describes an example of the software used for the interrupt handler. This software is present by default in the software development environments. It represents only one of several possible ways of handling the interrupts.

First of all, the jump addresses are defined at the beginning of the crt0.s file. In our case, all three interrupt levels jump to the same place (defined by the _interrupt label), but this can be changed if required.

```
##############################################################################
## Reset & interrupt vectors
##############################################################################
_start:
    jump  main_init   ; reset
    jump  _interrupt  ; IN1
    jump  _interrupt  ; IN2
    jump  _interrupt  ; IN0
```

The first thing to do when an interrupt is activated is to save the context. You have to start with saving the contents of the accumulator, then the flags and finally the internal CPU registers. You will find this part of the code in the IRQComon_xx.s file.

```
_interrupt:
##############################################################################
## Save all registers and flags
##############################################################################
      move    -(i3), a
      move    a, r0
      sflag
      move    -(i3), a
      move    -(i3), ipl
      move    -(i3), iph
      move    -(i3), i0l
      move    -(i3), i0h
      move    -(i3), i1l
      move    -(i3), i1h
      move    -(i3), i2l
      move    -(i3), i2h
      move    -(i3), r0
      move    -(i3), r1
      move    -(i3), r2
      move    -(i3), r3
```

Next step is to determine which interrupt is activated. In this case, we use the value in the **RegIrqPriority** register to determine the highest priority interrupt that was activated. Other ways can be used, especially when the priority order fixed in the hardware needs to be changed. You will find this part of the code in the IRQComon_xx.s file. In this example, the labels are used as defined for the XE8802.

```
##############################################################################
## The following lines enables the adress calculation of the interrupt
## table. Where RegIrqPriority is the addres offset for the table.
## The RegIrqPriority valid values are between 0x00 until 0x017. The
## 0xFF value should never exist.
##############################################################################
    move  r0,RegIrqPriority
    calls _interrupttab               ; save pc+1 in ip
_interrupttab:
    add   ipl,#0x05                 ; add the offset, nb instr. before table
    addc  iph,#0x00                   ; propagate carry
    add   ipl,r0                      ; add the offset of the regirqpriority
    addc  iph,#0x00                   ; propagate carry
    rets                              ; put ip in pc

    ; interrupt table
    jump  ret_int                     ;  RegIrqPriority = 0x00
    jump  ret_int                     ;  RegIrqPriority = 0x01
    jump  Irq_Pa2                     ;  RegIrqPriority = 0x02
    jump  Irq_Pa3                     ;  RegIrqPriority = 0x03
    jump  Irq_CntD                    ;  RegIrqPriority = 0x04
    jump  Irq_CntB                    ;  RegIrqPriority = 0x05
```

```
        jump  Irq_Pa6                   ;  RegIrqPriority = 0x06
        jump  Irq_Pa7                   ;  RegIrqPriority = 0x07
        jump  Irq_Pa0                   ;  RegIrqPriority = 0x08
        jump  Irq_Pa1                   ;  RegIrqPriority = 0x09
        jump  Irq_Vld                   ;  RegIrqPriority = 0x0A
        jump  Irq_1Hz                   ;  RegIrqPriority = 0x0B
        jump  Irq_Pa4                   ;  RegIrqPriority = 0x0C
        jump  Irq_Pa5                   ;  RegIrqPriority = 0x0D
        jump  Irq_UsrtCond1             ;  RegIrqPriority = 0x0E
        jump  Irq_UsrtCond2             ;  RegIrqPriority = 0x0F
        jump  Irq_UartRx                ;  RegIrqPriority = 0x10
        jump  Irq_UartTx                ;  RegIrqPriority = 0x11
        jump  Irq_Cmpd                  ;  RegIrqPriority = 0x12
        jump  Irq_CntC                  ;  RegIrqPriority = 0x13
        jump  Irq_CntA                  ;  RegIrqPriority = 0x14
        jump  Irq_Spi                   ;  RegIrqPriority = 0x15
        jump  Irq_128Hz                 ;  RegIrqPriority = 0x16
        jump  Irq_AC                    ;  RegIrqPriority = 0x17
```

The next steps are to clear the interrupt flag in the interrupt handler, to call the specific function for the identified interrupt source and to clear the interrupt in the stat register. This code can be found in the file IRQSave0_xx.s.

```
#############################################################################
Irq_AC:
    move  RegIrqHig, #0x80
    calls Handle_Irq_AC
    jump  ret_int0
#############################################################################
Irq_128Hz:
    move  RegIrqHig, #0x40
    calls Handle_Irq_128Hz
    jump  ret_int0
#############################################################################
Irq_Spi:
    move  RegIrqHig, #0x20
    calls Handle_Irq_Spi
    jump  ret_int0

…

ret_int0:
    clrb  stat, #2
    jump  ret_int
```

Finally, the context and the PC have to be restored. This code can be found in the IRQComon_xx.s file.

```
ret_int:
#############################################################################
## Restore all registers and flags
#############################################################################
       move    r3, (i3)+
       move    r2, (i3)+
       move    r1, (i3)+
       move    r0, (i3)+
       move    i2h, (i3)+
       move    i2l, (i3)+
       move    i1h, (i3)+
       move    i1l, (i3)+
       move    i0h, (i3)+
       move    i0l, (i3)+
       move    iph, (i3)+
       move    ipl, (i3)+
       rflag   (i3)+
       move    a, (i3)+
       reti
#############################################################################
## End of interrupt handlers
#############################################################################
```

9. Event Handler

9.1 Features

The XE8000 chips support 8 event sources, divided into 2 levels of priority.

9.2 Overview

An event is different from an interrupt in that it does not modify the program counter (PC). Events are used by two microcontroller instructions.

First of all, events are useful to wake-up the microcontroller when it is in HALT. The software execution then simply resumes at the instruction next to the HALT instruction. The second instruction is the conditional jump on event (JEV). The jump is executed if one of the event flags in the stat register is set. In all other cases, the occurrence of an event has no effect.

The event handler allows 8 event sources to be managed individually. The 8 event sources are divided into 2 levels of priority: High (4 event sources) and Low (4 event sources). Those 2 levels of priority are directly mapped to those supported by the CoolRisc® (EV0and IN1; see CoolRisc documentation for more information).

Additional functions are given that allow fast detection of the highest priority event that has been activated.

9.3 Register map

The addresses given in Table 9-1 are the default values and may be different in some products.

Register name
RegEvn
RegEvnEn
RegEvnPriority
RegEvnEvn

Table 9-1: EVN handler registers.

pos.	RegEvn	rw	reset	function
7	RegEvn[7]	r c1	0 nresetglobal	event #7 (high priority) clear event #7 when written 1
6	RegEvn[6]	r c1	0 nresetglobal	event #6 (high priority) clear event #6 when written 1
5	RegEvn[5]	r c1	0 nresetglobal	event #5 (high priority) clear event #5 when written 1
4	RegEvn[4]	r c1	0 nresetglobal	event #4 (high priority) clear event #4 when written 1
3	RegEvn[3]	r c1	0 nresetglobal	event #3 (low priority) clear event #3 when written 1
2	RegEvn[2]	r c1	0 nresetglobal	event #2 (low priority) clear event #2 when written 1
1	RegEvn[1]	r c1	0 nresetglobal	event #1 (low priority) clear event #1 when written 1
0	RegEvn[0]	r c1	0 nresetglobal	event #0 (low priority) clear event #0 when written 1

Table 9-2: **RegEvn**

pos.	RegEvnEn	rw	reset	function
7	RegEvnEn[7]	rw	0 nresetglobal	1= enable event #7
6	RegEvnEn[6]	rw	0 nresetglobal	1= enable event #6
5	RegEvnEn[5]	rw	0 nresetglobal	1= enable event #5
4	RegEvnEn[4]	rw	0 nresetglobal	1= enable event #4
3	RegEvnEn[3]	rw	0 nresetglobal	1= enable event #3
2	RegEvnEn[2]	rw	0 nresetglobal	1= enable event #2
1	RegEvnEn[1]	rw	0 nresetglobal	1= enable event #1
0	RegEvnEn[0]	rw	0 nresetglobal	1= enable event #0

Table 9-3: **RegEvnEn**

pos.	RegEvnPriority	rw	reset	function
7-0	RegEvnPriority	r	11111111 nresetglobal	code of highest event set FF if no event present.

Table 9-4: **RegEvnPriority**

pos.	RegEvnEvn	rw	reset	function
7-2	-	r	00000	unused
1	EvnHig	r	0 nresetglobal	one or more high priority event is set
0	EvnLow	r	0 nresetglobal	one or more low priority event is set

Table 9-5: **RegEvnEvn**

9.4 Detailed description

The CoolRISC core has 2 different event levels EV0 and EV1 (Figure 9-1).

The setting and clearing of these events can be done in the stat register (see chapter describing the CPU).

The event handler bundles a certain number of event sources and routes them to one of these two events and provides the possibility to enable/disable each of them individually. The definition of the event sources is given in the memory mapping chapter.

RegEvn is an 8-bit register containing flags for the event sources. Those flags are set when the event is enabled (i.e. if the corresponding bit in the registers **RegEvnEn** is set) and a rising edge is detected on the corresponding event source.

Once memorized, writing a '1' in the corresponding bit of **RegEvn** clears an event flag. Writing a '0' does not modify the flag. All interrupts are automatically cleared after a reset.

Two registers are provided to facilitate the writing of event service software. **RegEvnPriority** contains the number of the highest event set (its value is 0xFF when no event is memorized). **RegEvnEvn** indicates the priority level of the current events.

All event sources are sampled by the highest frequency in the system. A CPU event is generated and memorized when an event becomes high. The 8 event sources are divided into 2 levels of priority: High (4 event sources) and Low (4 event sources). Those 2 levels of priority are directly mapped to those supported by the CoolRisc (EV0 and EV1; see CoolRisc documentation for more information).

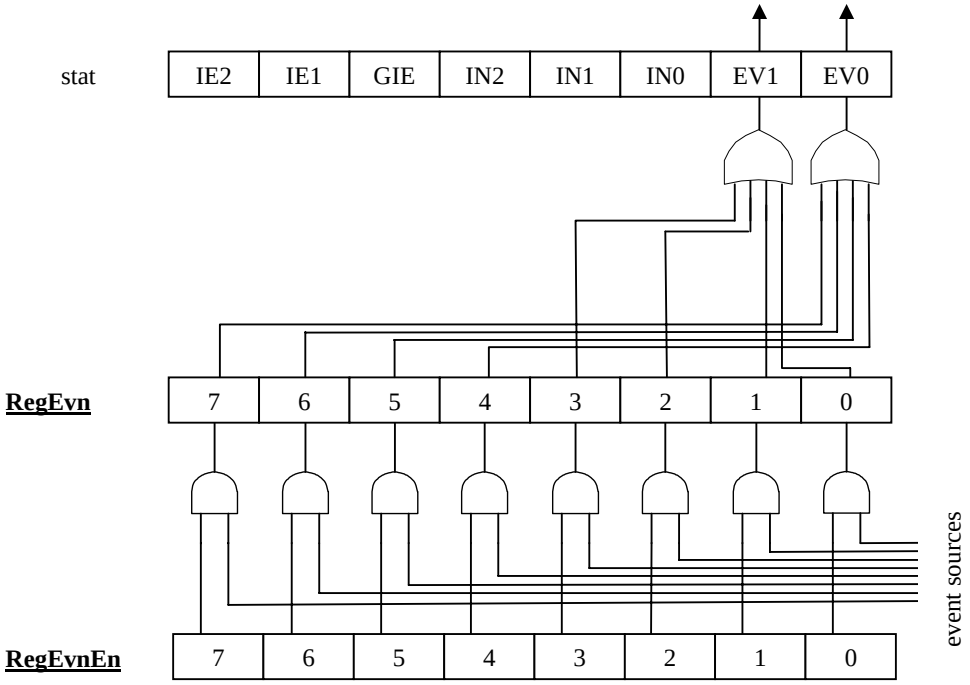


Figure 9-1. Event handler principle.

10. Low Power RAM

10.1 Features

10.1.1 Overview

In order to save power consumption, 8 8-bit registers are provided in page 0. These memory locations should be reserved for often-updated variables. As they are real registers and not RAM, power consumption is greatly reduced.

10.2 Register map

pos.	Reg00	rw	reset	function
7-0	Reg00	rw	0	low-power data memory
7-0	Reg01	rw	0	low-power data memory
7-0	Reg02	rw	0	low-power data memory
7-0	Reg03	rw	0	low-power data memory
7-0	Reg04	rw	0	low-power data memory
7-0	Reg05	rw	0	low-power data memory
7-0	Reg06	rw	0	low-power data memory
7-0	Reg07	rw	0	low-power data memory

Table 10-1: Low Power RAM

11. Port A

11.1 Features

- input port, 8 bits wide
- each bit can be set individually for debounced or direct input
- each bit can be set individually for pull-up or not
- snap-to-rail option for each input
- each bit is an interrupt request source on the rising or falling edge
- a system reset can be generated on an input pattern
- PA[0] and PA[1] can generate two events for the CPU, individually maskable
- PA[0] to PA[3] can be used as clock inputs for the counters/timers/PWM (product dependent)

11.2 Overview

PortA is a general purpose 8 bit wide digital input port, with interrupt capability. Figure 11-1 shows its structure.

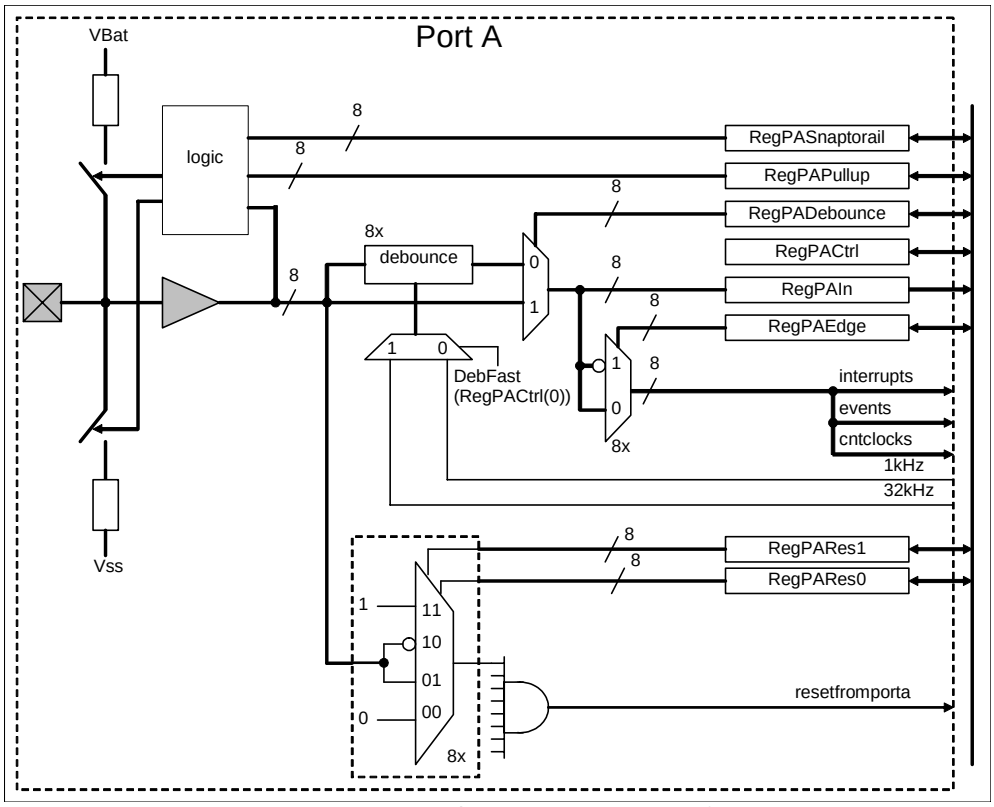


Figure 11-1: structure of Port A

11.3 Register map

There are eight registers in Port A (PA), namely **RegPAIn**, **RegPADebounce**, **RegPAPullup**, **RegPAEdge**, **RegPARes0**, **RegPARes1**, **RegPACtrl** and **RegPASnaptorail**. Table 11-2 to Table 11-9 show the mapping of control bits and functionality of these registers.

register name
RegPAIn
RegPADebounce
RegPAEdge
RegPAPullup
RegPARes0
RegPARes1
RegPACtrl
RegPASnaptorail

Table 11-1: PA registers

pos.	RegPAIn	rw	reset	description
7:0	PAIn[7:0]	r	x	pad PA[7] to PA[0] input value

Table 11-2: RegPAIn

pos.	RegPADebounce	rw	reset	description
7:0	PADebounce[7:0]	r w	0 nresetpconf	PA[7] to PA[0] 1: debounce enabled 0: debounce disabled

Table 11-3: RegPADebounce

pos.	RegPAEdge	rw	reset	description
7:0	PAEdge[7:0]	r w	0 nresetglobal	PA[7] to PA[0] edge configuration 0: positive edge 1: negative edge

Table 11-4: RegPAEdge

pos.	RegPAPullup	rw	reset	description
7:0	PAPullup[7:0]	r w	1 nresetpconf	PA[7] to PA[0] pullup enable 0: pullup disabled 1: pullup enabled

Table 11-5: RegPAPullup

pos.	RegPARes0	rw	reset	description
7:0	PARes0[7:0]	r w	0 nresetglobal	PA[7] to PA[0] reset configuration

Table 11-6: RegPARes0

pos.	RegPARes1	rw	reset	description
7:0	PARes1[7:0]	r w	0 nresetglobal	PA[7] to PA[0] reset configuration

Table 11-7: RegPARes

pos.	RegPACtrl	rw	reset	description
7:1	[7:1]	r	0000000	Unused
0	DebFast	r w	0 nresetpconf	0 = slow debounce, 1 = fastdebounce

Table 11-8: RegPACtrl

pos.	RegPASnaptorail	rw	reset	description
7:0	PASnaptorail[7:0]	rw	0 nresetpconf	set snap-to-rail input on

Table 11-9: RegPASnaptorail

Note*:* Depending on the status of the **EnResetPConf** bit in **RegSysCtrl**, **RegPAEdge, RegPADebounce** and **RegPACtrl** can be reset by any of the possible system resets or only with power-on reset and NRESET pad.

11.4 Interrupts and events map

Interrupt source	Default mapping in the interrupt manager	Default mapping in the event manager
pa_irqbus[5]	RegIrqMid[5]	
pa_irqbus[4]	RegIrqMid[4]	
pa_irqbus[1]	RegIrqMid[1]	RegEvn[4]
pa_irqbus[0]	RegIrqMid[0]	RegEvn[0]
pa_irqbus[7]	RegIrqLow[7]	
pa_irqbus[6]	RegIrqLow[6]	
pa_irqbus[3]	RegIrqLow[3]	
pa_irqbus[2]	RegIrqLow[2]	

11.5 Port A (PA) Operation

The Port A input status (debounced or not) can be read from **RegPAin**.

Debounce mode:
Each bit in Port A can be individually debounced by setting the corresponding bit in **RegPADebounce**. After reset, the debounce function is disabled. After enabling the debouncer, the change of the input value is accepted only if height consecutive samples are identical. Selection of the clock is done by bit **DebFast** in Register **RegPACtrl**.

DebFast	Clock filter
0	1kHz
1	32kHz

Table 11-10**: debounce frequency selection**

Note: The tolerance on the debounce frequency depends on the selected clock source. When the external clock is used, the pulse width will be correct if the input of the low prescaler is set to a frequency close to 32kHz (see clock block documentation).

Pullups/Snap-to-rail:
Different functions are possible depending on the value of the registers **RegPAPullup** and **RegPASnaptorail**. When the corresponding bit in **RegPAPullup** is set to 0, the inputs are floating (pullup and pulldown resistors are disconnected). When the corresponding bit in **RegPAPullup** is 1 and in **RegPASnaptorail** is 0, a pullup resistor is connected to the input pin. Finally, when the corresponding bit in **RegPAPullup** is 1 and in **RegPASnaptorail** is 1, the snap-to-rail function is active.

The snap-to-rail function connects a pullup or pulldown resistor to the input pin depending on the value forced on the input pin. This function can be used for instance when the input port is connected to a tristate bus. When the bus is floating, the pullup or pulldown maintains the bus in the last low impedance state before it became floating until another low impedance output drives the bus. It also reduces the power consumption with respect to a classic pullup since it selects the pullup or pulldown resistor so that it confirms the detected input state.

The state of input pin is summarized in the table below.

PAPullup[x]	PASnaptorail[x]	(last) externally forced PA[x] value	PA[x] pull
0	x	x	floating
1	0	x	pullup
1	1	0	*pulldown*
1	1	1	*pullup*

Table 11-11**: Snap-to-rail**

Port A starts up with the pullup resistor connected and the snap-to-rail function disabled.

Port A as an interrupt source:
Each Port A input is an interrupt request source and can be set on rising or falling edge with the corresponding bit in **RegPAEdge**. After reset, the rising edge is selected for interrupt generation by default. The interrupt source can be debounced by setting register **RegPADebounce**. The interrupt signals are sampled on the fastest clock in the circuit. In order to guarantee that the circuit detects the interrupt, the minimal pulse length should be 1 cycle of this clock.

Note: care must be taken when modifying **RegPAEdge** because this register performs an edge selection. The change of this register may result in a transition, which may be interpreted as a valid interruption.

Port A as an event source:
The interrupt signals of the pins PA[0] and PA[1] are also available as events on the event controller.

Port A as a clock source (product dependent):
Images of the PA[0] to PA[3] input ports (debounced or not) are available as clock sources for the counter/timer/PWM peripheral.

Port A as a reset source:
Port A can be used to generate a system reset by placing a predetermined word on Port A externally. The reset is built using a logical and of the 8 PARes[x] signals:

resetfromportA = PAReset[7] AND PAReset[6] AND PAReset[5] AND ... AND PAReset[0]

PAReset[x] is itself a logical function of the corresponding pin PA[x]. One of four logical functions can be selected for each pin by writing into two registers **RegPARes0** and **RegPARes1** as shown in Table 11-12.

PARes1[x]	PARes0[x]	PAReset[x]
0	0	0
0	1	PA[x]
1	0	not(PA[x])
1	1	1

Table 11-12: Selection bits for reset signal

A reset from Port A can be inhibited by placing a 0 on both **PARes1[x]** and **PARes0[x]** for at least 1 pin. Setting both **PARes1[x]** and **PARes0[x]** to 1, makes the reset independent of the value on the corresponding pin. Setting both registers to hFF, will reset the circuit independent from the Port A input value. This makes it possible to do a reset by software.

Note: depending of the value of PA[0] to PA[7], changes to **RegPARes0** and **RegPARes1** can cause a reset. Therefore it is safe to have always one (RegPARes0[x], RegPARes1[x]) equal to '00' during the setting operations.

11.6 Port A electrical specification

sym	description	min	typ	max	unit	Comments
V_{INH}	Input high voltage	0.7*VBAT		VBAT	V	VBAT≥2.4V
V_{INL}	Input low voltage	VSS		0.2*VBAT	V	VBAT≥2.4V
R_{PU}	Pull-up resistance	20	50	80	kΩ	
Cin	Input capacitance		2.5		pF	Note 1

Note 1: this value is indicative only since it depends on the package.

Table 11-13. Electrical specification

12. Port B

12.1 Features

• Input / output / analog port, 8 bits wide
• Each bit can be set individually for input or output
• Each bit can be set individually for open-drain or push-pull
• Each bit can be set individually for pull-up or not (for input or open-drain mode)
• In open-drain mode, pull-up is not active when corresponding pad is set to zero
• The 8 pads can be connected individually to four internal analog lines (4 line analog bus)
• Two internal freq. (16 kHz and cpuck) can be output on PB[2] and PB[3]

Product dependant:
• Two PWM signal can be output on pads PB[0] and PB[1]
• The synchronous serial interface (USRT) uses pads PB[5], PB[4]
• The UART interface uses pads PB[6] and PB[7] for Tx and Rx

12.2 Overview

Port B is a multi-purpose 8 bit Input/output port. In addition to digital behavior, all pins can be used for analog signals. Each port terminal can be individually selected as digital input or output or as analog for sharing one of four possible analog lines.

12.3 Register map

Table 12-1 shows the Port B registers.

register name
RegPBOut
RegPBIn
RegPBDir
RegPBOpen
RegPBPullup
RegPBAna

Table 12-1: Default Port B registers

pos.	RegPBOut	rw	reset	description in digital mode	description in analog mode
7 – 0	PBOut[7-0]	r w	0 nresetpconf	Pad PB[7-0] output value	Analog bus selection for pad PB[7-0]

Table 12-2: RegPBOut

pos.	RegPBIn	rw	reset	description in digital mode	description in analog mode
7 – 0	PBIn[7-0]	r w	X	Pad PB[7-0] input status	Unused

Table 12-3: RegPBIn

pos.	RegPBDir	rw	reset	description in digital mode	description in analog mode
7 – 0	PBDir [7-0]	r w	0 nresetpconf	Pad PB[7-0] direction (0=input)	Analog bus selection for pad PB[7-0]

Table 12-4**:** RegPBDir

pos.	RegPBOpen	rw	reset	description in digital mode	description in analog mode
7 – 0	PBOpen[7-0]	r w	0 nresetpconf	Pad PB[7-0] open drain (1 = open drain)	Unused

Table 12-5: RegPBOpen

pos.	RegPBPullup	rw	reset	description in digital mode	description in analog mode
7 –0	PBPullup[7]	r w	1 nresetpconf	Pull-up for pad PB[7-0] (1=active)	Connect pad PB[7-0] on selected ana bus

Table 12-6: RegPBPullup

pos.	RegPBAna	rw	reset	description in digital mode	description in analog mode
7 – 0	PBAna [7-0]	r w	0 nresetpconf	Set PB[7-0] in analog mode	Set PB[7-0] in analog mode

Table 12-7: RegPBAna

Note: Depending on the status of the **EnResPConf** bit in **RegSysCtrl**, the reset conditions of the registers are different. See the reset block documentation for more details on the nresetpconf signal.

12.4 Port B capabilities

Port B	utilization (priority)		
name	high (analog)	medium (functions)	low (digital) (default)
PB[7]	analog	uart Rx	I/O (with pull-up)
PB[6]	analog	uart Tx	I/O (with pull-up)
PB[5]	analog	usrt S1	I/O (with pull-up)
PB[4]	analog	usrt S0	I/O (with pull-up)
PB[3]	analog	16 kHz	I/O (with pull-up)
PB[2]	analog	clock CPU	I/O (with pull-up)
PB[1]	analog	PWM1 Counter C (C+D)	I/O (with pull-up)
PB[0]	analog	PWM0 Counter A (A+B)	I/O (with pull-up)

Table 12-8: Different Port B functions

Table 12-8 shows the different usages that can be made of port B with the order of priority. If a pin is selected to be analog, it overwrites the function and digital set-up. If the pin is not selected as analog, but a function is enabled, it overwrites the digital set-up. If neither the analog nor function is selected for a pin, it is used as an ordinary digital I/O. This is the default configuration at start-up.

Note: the presence of the functions is product dependent.

12.5 Port B analog capability

12.5.1 Port B analog configuration

Port B terminals can be attached to a 4 line analog bus by setting the **PBAna[x]** bits to 1 in the **RegPBAna** register.

The other registers then define the connection of these 4 analog lines to the different pads of Port B. These can be used to implement a simple LCD driver or A/D converter. Analog switching is available only when the circuit is powered with sufficient voltage (see specification below). Below the specified supply voltage, only voltages that are close to VSS or VBAT can be switched.

When **PBAna[x]** is set to 1, one pad of the Port B terminals is changed from digital I/O mode to analog. The usage of the registers **RegPBPullup, RegPBOut** and **RegPBDir** define the analog configuration (see Table 12-9).

When **PBAna[x]** = 1**,** then **PBPullup[x]** connects the pin to the analog bus. **PBDir[x]** and **PBPOut[x]** select which of the 4 analog lines is used.

analog bus selection		PBPullup[x]	PB[x] selection on
PBDir[x]	PBout[x]		
0	0	1	analog line 0
0	1	1	analog line 1
1	0	1	analog line 2
1	1	1	analog line 3
X	X	0	High impedance

Table 12-9: Selection of the analog lines with **RegPBDir**, **RegPBout** and **RegPBPullup** when **PBAna[x]** = 1

Example:

Set the pads PB[2] and PB[5] on the analog line 3. (the values X depend on the configuration of others pads)

- apply high impedance in the analog mode (move RegPBPullup,#0bXX0XX0XX)
- go to analog mode (move RegPBAna,#0bXX1XX1XX)
- select the analog line3 (move RegPBDir,#0bXX1XX1XX and move RegPBOut,#0bXX1XX1XX)
- apply the analog line to the output (move RegPBPullup,#0bXX1XX1XX)

12.5.2 Port B analog function specification

The table below defines the on-resistance of the switches between the pin and the analog bus for different conditions. The series resistance between 2 pins of Port B connected to the same analog line is twice the resistance given in the table.

sym	description	min	typ	max	unit	Comments
Ron	switch resistance			11	kΩ	Note 1
Ron	switch resistance			15	kΩ	Note 2
Cin	input capacitance (off)		3.5		pF	Note 3
Cin	input capacitance (on)		4.5		pF	Note 4

Table 12-10. Analog input specifications.

Note 1: This is the series resistance between the pad and the analog line in 2 cases
1. VBAT ≥ 2.4V and the VMULT peripheral is present on the circuit and enabled.
2. VBAT ≥ 3.0V and the VMULT peripheral is not present on the circuit.

Note 2: This is the series resistance in case VBAT ≥ 2.8V and the peripheral VMULT is not present on the circuit.
Note 3: This is the input capacitance seen on the pin when the pin is not connected to an analog line. This value is indicative only since it is product and package dependent.
Note 4: This is the input capacitance seen on the pin when the pin is connected to an analog line and no other pin is connected to the same analog line. This value is indicative only since it is product and package dependent.

12.6 Port B function capability

The Port B can be used for different functions implemented by other peripherals. The description below is applicable only in so far the circuit contains these peripherals.

When the counters are used to implement a PWM function (see the documentation of the counters), the PB[0] and PB[1] terminals are used as outputs (PB[0] is used if **CntPWM0** in **RegCntConfig1** is set to 1, PB[1] is used if **CntPWM1** in **RegCntConfig1** is set to 1) and the PWM generated values override the values written in **RegPBout**. However, **PBDir(0)** and **PBDir(1)** are not automatically overwritten and have to be set to 1.

If **Output16k** is set in **RegSysMisc**, the frequency is output on PB[3]. This overrides the value contained in **PBOut(3)**. However, **PBDir(3)** must be set to 1. The frequency and duty cycle of the clock signal are given in Figure 12-1. f_{max} is the frequency of fastest clock present in the circuit.

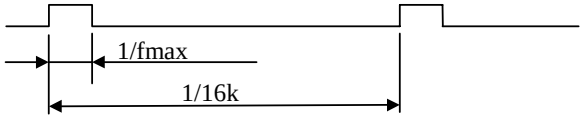


Figure 12-1. 16 kHz output clock timing

Similarly, if **OutputCkCpu** is set in **RegSysMisc**, the CPU frequency is output on PB[2]. This overrides the value contained in **PBOut(2)**. However, **PBDir(2)** must be set to 1.

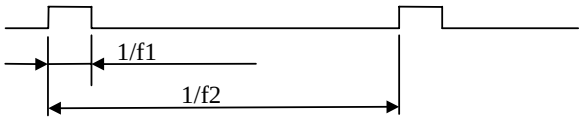


Figure 12-2. CPU output clock timing.

The timing of the CPU clock (Figure 12-2) depends on the selection of the **CpuSel** bit in the **RegSysClock** register and is given in Table 12-11. f_{max} is the frequency of fastest clock present in the circuit. Note that the tolerance on the 32 kHz depends on the selected clock source (see clock block documentation).

CpuSel	f1	f2
0	$f_{max}/4$	f_{max}
1	f_{max}	32 kHz

Table 12-11. CPU clock timing parameters.

Pins PB[5] and PB[4] can be used for S1 and S0 of the USRT (see USRT documentation) when the **UsrtEnable** bit is set in **RegUsrtCtrl**. The PB[5] and PB[4] then become open-drain. This overrides the values contained in **PBOpen(5:4)**, **PBOut(5:4)** and **PBDir(5:4)**. If there is no external pull-up resistor on these pins, internal pull-ups should be selected by setting **PBPullup(5:4)**. When S0 is an output, the pin PB[4] takes the value of **UsrtS0** in **RegUsrtS0**. When S1 is an output, the pin PB[5] takes the value of **UsrtS1** in **RegUsrtS1**.

Pins PB[6] and PB[7] can be used by the UART (see UART documentation). When **UartEnTx** in **RegUartCtrl** is set to 1, PB[6] is used as output signal Tx. When **UartEnRx** in **RegUartCtrl** is set to 1, PB[7] is used as input signal Rx. This overrides the values contained in **PBOut(7:6)** and **PBDir(7:6)**.

12.7 Port B digital capabilities

12.7.1 Port B digital configuration

The direction of each bit within Port B (input only or input/output) can be individually set using the **RegPBDir** register. If **PBDir[x]** = 1, both the input and output buffer are active on the corresponding Port B. If **PBDir[x]** = 0, the corresponding Port B pin is an input only and the output buffer is in high impedance. After reset (nresetpconf) Port B is in input only mode **(PBDir[x]** are reset to 0).

The input values of Port B are available in **RegPBIn** (read only). Reading is always direct - there is no debounce function in Port B. In case of possible noise on input signals, a software debouncer with polling or an external hardware filter have to be realized. The input buffer is also active when the port is defined as output and the effective value on the pin can be read back.

Data stored in **RegPBOut** are outputted at Port B if **PBDir[x]** is 1. The default values after reset is low (0).

When a pin is in output mode (**PBDir[x]** is set to 1), the output can be a conventional CMOS (Push-Pull) or a N-channel Open-drain, driving the output only low. By default, after reset (nresetpconf) the **PBOpen[x]** in **RegPBOpen** is cleared to 0 (push-pull). If **PBOpen[x]** in **RegPBOpen** is set to 1 then the internal P transistor in the output buffer is electrically removed and the output can only be driven low (**PBOut[x]**=0). When **PBOut[x]**=1, the pin is high Impedance. The internal pull-up or an external pull-up resistor can be used to drive to pin high.
Note: Because the P transistor actually exists (this is not a real Open-drain output) the pull-up range is limited to VDD + 0.2V (avoid forward bias the P transistor / diode).

Each bit can be set individually for pull-up or not using register **RegPBPullup**. Input is pulled up when its corresponding bit in this register is set to 1. Default status after (nresetpconf) is 1, which means with pull up. To limit power consumption, pull-up resistors are only enabled when the associated pin is either a digital input or an N-channel open-drain output with the pad set to 1. In the other cases (push-pull output or open-drain output driven low), the pull up resistors are disabled independent of the value in **RegPBPullup**.

After power-on reset, the Port B is configured as an input port with pull-up. During power-on reset (see reset block documentation) however, the pin PB[1] is pulled down in stead of pulled up. Once the power-on reset completed, the pin PB[1] is pulled up, exactly as the other Port B pins.

The input buffer is always active, except in analog mode. This means that the Port B input should be a valid digital value at all times unless the pin is set in analog mode. Violating this rule may lead to high power consumption.

12.7.2 Port B digital function specification

sym	description	min	typ	max	unit	Comments
V_{INH}	Input high voltage	0.7*VBAT		VBAT	V	VBAT≥2.4V
V_{INL}	Input low voltage	VSS		0.2*VBAT	V	VBAT≥2.4V
V_{OH}	Output high voltage	VBAT-0.4		VBAT	V	VBAT=1.2V, I_{OH} =0.3mA VBAT=2.4V, I_{OH} =5.0mA VBAT=4.5V, I_{OH} =8.0mA
V_{OL}	Output low voltage	VSS		VSS+0.4	V	VBAT=1.2V, I_{OL} =0.3mA VBAT=2.4V, I_{OL} =12.0mA VBAT=4.5V, I_{OL} =15.0mA
R_{PU}	Pull-up resistance	20	50	80	kΩ	
Cin	Input capacitance		3.5		pF	Note 1

Note 1: this value is indicative only since it depends on the package.

Table 12-12. Digital port specification

12.8 Low power comparators

If the low power comparator (CMPD) peripheral is present in the circuit, the signals on the pins PB[7:4] can be used as inputs for these low power comparators. Although the comparators are functional independent of the Port B configuration, it is recommended to set the pins that are used for the CMPD in analog mode without selecting any analog lines. This to avoid high power consumption in the digital input buffer when analog or slowly varying digital signals are applied.

13. Port D

13.1 Features

- input / output port, 8 bits wide
- each bit can be set individually for input or output
- pull-ups are available in input mode
- snap-to-rail option in input mode

13.2 Overview

Port D (PD) is a general purpose 8 bit input/output digital port.
Figure 13-1 shows its structure.

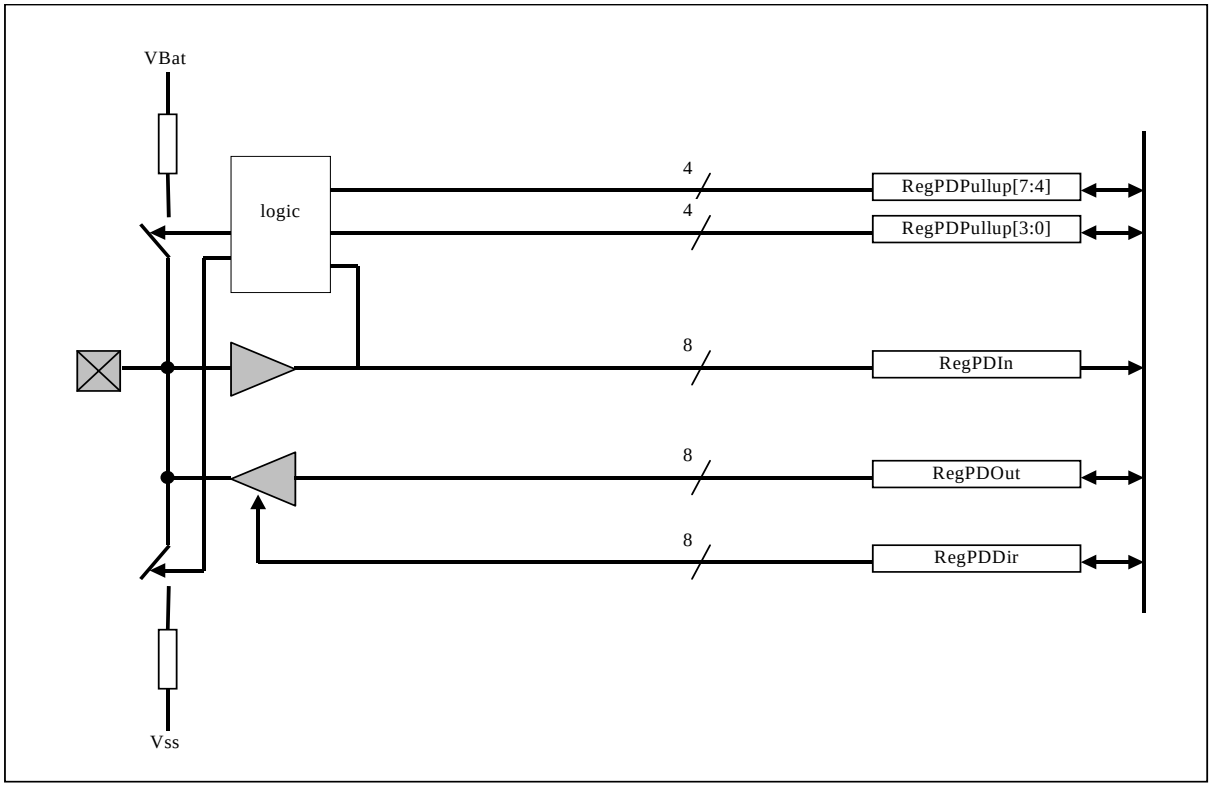


Figure 13-1 : structure of PortD

13.3 Register map

There are four registers in the Port D (PD), namely **RegPDIn**, **RegPDOut**, **RegPDDir** and **RegPDPullup**. Table 13-2 to Table 13-5 show the mapping of control bits and functionality of these registers.

register name
RegPDIn
RegPDOut
RegPDDir
RegPDPullup

Table 13-1: PD registers

Pos.	RegPDIn	Rw	Reset	Description
7:0	PDIn[7:0]	r	-	pad PD[7:0] input value

Table 13-2: RegPDIn

Pos.	RegPDOut	Rw	Reset	Description
7:0	PDOut[7:0]	r w	0 nresetpconf	pad PD[7:0] output value

Table 13-3: RegPDOut

Pos.	RegPDDir	Rw	Reset	Description
7:0	PDDir[7:0]	r w	0 nresetpconf	pad PD[7:0] direction (0=input)

Table 13-4: RegPDDir

Pos.	RegPDPullup	Rw	Reset	Description
7	PDSnapToRail[3]	r w	1 nresetpconf	snap-to-rail for pad PD[7] and PD[6] (1=active)
6	PDSnapToRail[2]	r w	1 nresetpconf	snap-to-rail for pad PD[5] and PD[4] (1=active)
5	PDSnapToRail[1]	r w	1 nresetpconf	snap-to-rail for pad PD[3] and PD[2] (1=active)
4	PDSnapToRail[0]	r w	1 nresetpconf	snap-to-rail for pad PD[1] and PD[0] (1=active)
3	PDPullup[3]	r w	1 nresetpconf	pullup for pad PD[7] and PD[6] (1=active)
2	PDPullup[2]	r w	1 nresetpconf	pullup for pad PD[5] and PD[4] (1=active)
1	PDPullup[1]	r w	1 nresetpconf	pullup for pad PD[3] and PD[2] (1=active)
0	PDPullup[0]	r w	1 nresetpconf	pullup for pad PD[1] and PD[0] (1=active)

Table 13-5: RegPDPullup

13.4 Port D (PD) Operation

The direction of each pin of Port D (input or input/output) can be individually set by using the **RegPDDir** register. If **PDDir[x]** = 1, the output buffer on the corresponding Port D pin is enabled. After reset, Port D is in input only mode (**PDDir[x]** are reset to 0). The input buffer is always enabled independently from the **RegPDDir** contents.

Output data:
Data are stored in **RegPDOut** prior to output at Port D.

Input data:
The status of Port D is available in **RegPDIn** (read only). Reading is always direct - there is no digital debounce function associated with Port D. In case of possible noise on input signals, a software debouncer or an external filter must be realised.

Pull-up/Snap to Rail:
When configured as an input (**PDDir[x]**=0), pull-ups are available on every pin. The pull-up function of the pins is controlled two by two by the **PDPullup** and **PDSnapToRail** bits in the register **RegPDPullup**. When a bit

PDPullup[x] is 0, the pull-ups on the pins PD[2x] and PD[2x+1] are disabled. When a bit **PDPullup[x]** is set to 1 and the bit **PDSnapToRail[x]** is set to 0, the pull-up resistor is connected to the pins PD[2x] and PD[2x+1]. When both **PDPullup[x]** and **PDSnapToRail[x]** are 1, the snap-to-rail function is active on the pins PD[2x] and PD[2x+1].

The snap-to-rail function connects a pullup or pulldown resistor to the input pin depending on the value forced on the input pin. This function can be used for instance when the input port is connected to a tristate bus. When the bus is floating, the pullup or pulldown maintains the bus in the last low impedance state before it became floating until another low impedance output is driving the bus. It also reduces the power consumption with respect to a classic pullup since it selects the pullup or pulldown resistor so that it confirms the detected input state.

The function is summarised in the table below as a function of the different register settings.

PDDir[2x(+1)]	PDPullup[x]	PDSnapToRail[x]	(last) externally forced PD[2x(+1)] value	PD[2x(+1)] pull resistor
1	x	x	x	not connected
0	0	x	x	not connected
0	1	0	x	pullup
0	1	1	0	pulldown
0	1	1	1	pullup

Table 13-6: Snap-to-rail and pullup function

At power-on reset, Port D is configured as an input port with all pull-ups active.

13.5 Port D electrical specification

sym	description	min	typ	max	unit	Comments
V_{INH}	Input high voltage	0.7*VBAT		VBAT	V	VBAT≥2.4V
V_{INL}	Input low voltage	VSS		0.2*VBAT	V	VBAT≥2.4V
V_{OH}	Output high voltage	VBAT-0.4		VBAT	V	VBAT=1.2V, I_{OH} =0.3mA VBAT=2.4V, I_{OH} =5.0mA VBAT=4.5V, I_{OH} =8.0mA
V_{OL}	Output low voltage	VSS		VSS+0.4	V	VBAT=1.2V, I_{OL} =0.3mA VBAT=2.4V, I_{OL} =12.0mA VBAT=4.5V, I_{OL} =15.0mA
R_{PU}	Pull-up resistance	20	50	80	kΩ	
Cin	Input capacitance		3.0		pF	Note 1

Note 1: this value is indicative only since it depends on the package.

Table 13-7. Port D electrical specification

14. RF Interface (BitJockey™)

14.1 Introduction

This block is a PCM Bit Stream Encoder / Decoder or RF Receiver / Transmitter Interface. In a normal microcontroller, the bits and low level coding of the RF protocol are done by software. This is a very tedious and CPU time consuming task since the processor has to handle bit by bit as they are received or as they have to be transmitted. This block simplifies the handling of the low level data handling for wireless data transmission systems very much in a similar way an UART interface does for wired transmission systems.

14.2 Features

- Two functions : Receiver and Transmitter (half duplex)
- Internal baud rate generator for all the standard baud rates
- Decode / encode 3 main PCM codes (NRZ, Bi-Phase and Miller), can be bypassed
- Buffered input / output data (FIFO)
- 4 interrupt sources
- Internal or external bit synchronization
- Internal or external start sequence detection

14.3 General overview

The RF Interface block is a bit stream receiver / transmitter for wireless transmission. It can be used with XEMICS' or third party RF transceiver circuits.

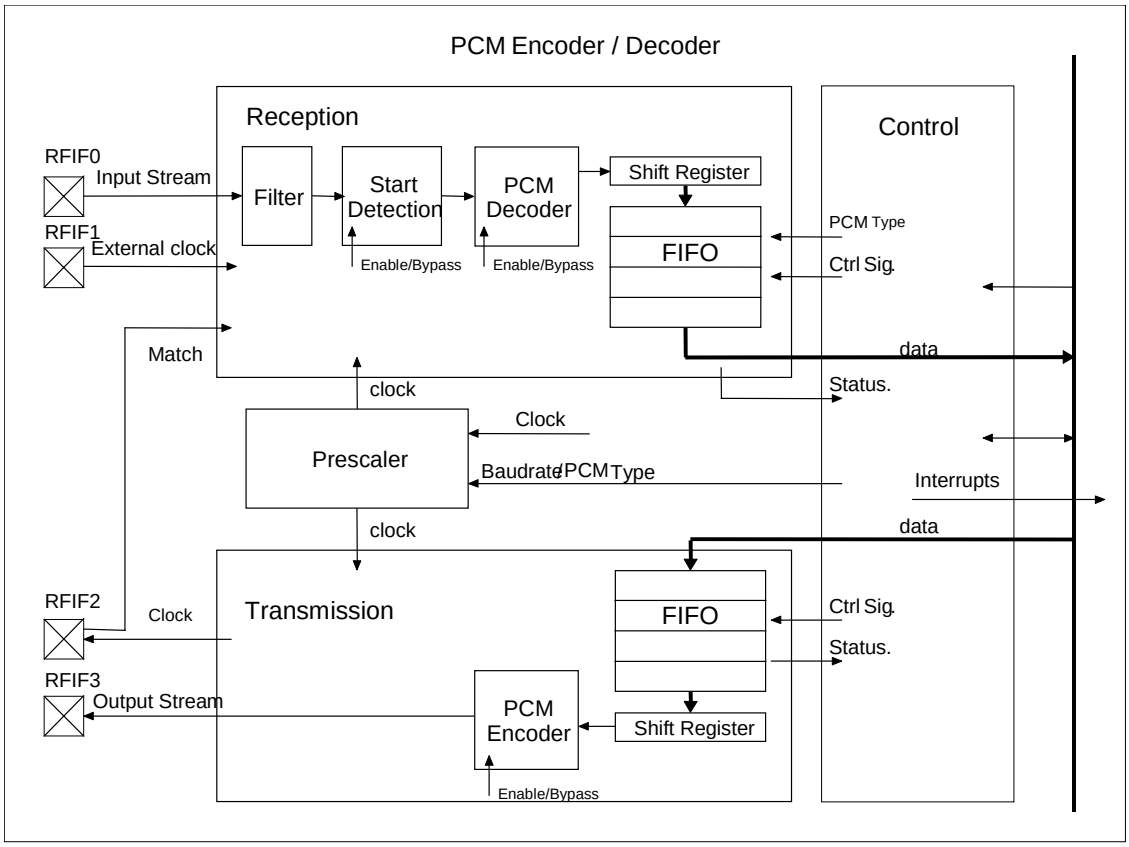


Figure 14-1 : PCM Encoder / Decoder Structure

The **Reception** block consists of:

- The “Filter”. The input data are filtered depending on the selected baud rate. The block extracts the bit synchronization clock. When an external bit synchronization clock is used, the filter is bypassed.
- The “Start Detection” block. This block detects a programmable start sequence and therefore synchronizes at the message start. The interface ignores incoming messages until the start sequence is detected. This block can be bypassed or an external message synchronization signal can be used (match in Figure 14-1).
- The “PCM Decoder” which decodes the 3 main types of PCM waveforms (i.e. NRZ, Bi-Phase, Miller). This block can be bypassed.
- The “Shift Register” and the FIFO, that do the serial to parallel conversion and memorize the input data.

The **Transmission** block consists of:

- The FIFO and the “Shift Register” which store the data and do the parallel to serial conversion.
- The “Encoder” which encodes the serial data into the chosen PCM type. This block can be bypassed.

The **Prescaler** generates the clock signals needed for the reception and transmission depending on the selected baud rate and PCM code type.

14.4 Register map

register name
RegRfifCmd1
RegRfifCmd2
RegRfifCmd3
RegRfifTx
RegRfifTxSta
RegRfifRx
RegRfifRxSta
RegRfifRxSPat

Table 14-1: Rfif registers

pos.	RegRfifCmd1	rw	reset	description
7-6	unused	-	-	-
5-4	RfifBRCoarse(1:0)	r/w	00 nresetglobal	Select baud rate (coarse selection)
3-0	RfifBRFine(3:0)	r/w	0000 nresetglobal	Select baud rate (fine selection)

Table 14-2: RegRfifCmd1

pos.	RegRfifCmd2	rw	reset	description
7-6	RfifEnStart(1:0)	r/w	00 nresetglobal	Start Detection mode in Rx mode (see page 14-8)
5	RfifEnCod	r/w	0 nresetglobal	‘1’ enables decoder (Rx) / encoder (Tx)
4	RfifRxClock	r/w	0 nresetglobal	‘1’ enables external clock input (Rx)
3	RfifTxClock	r/w	0 nresetglobal	‘1’ enables external clock output (Tx)
2-0	RfifPCM(2:0)	r/w	000 nresetglobal	Select PCM type (see page 14-6)

Table 14-3: RegRfifCmd2

pos.	RegRfifCmd3	rw	reset	description
7-5	RfifRxIrqEn	r/w	000 nresetglobal	'1' enable the Rx Irq sources (see page 14-11)
4-2	RfifRxIrqMem	r/c1	000 nresetglobal	Rx Irq Status (see page 14-11)
1	RfifEnRx	r/w	0 nresetpconf	'1' enable Rfif reception mode
0	RfifEnTx	r/w	0 nresetpconf	'1' enable Rfif transmission mode

Table 14-4: RegRfifCmd3

pos.	RegRfifTx	rw	reset	description
7-0	RfifTx	w	00000000 nresetglobal	Data to be sent FIFO, depth = 4

Table 14-5: RegRfifTx

pos.	RegRfifTxSta	rw	reset	description
7-4	-	-	-	-
3	RfifTxFifoOverrun	r/c1	0 nresetglobal	Tx Fifo overrun error. Write: clear FIFO
2	RfifTxFifoFull	r	0 nresetglobal	Tx Fifo full
1	RfifTxFifoEmpty	r	1 nresetglobal	Tx Fifo empty [IRQ]
0	RfifTxStopped	r	0 nresetglobal	Tx Fifo + Tx shift register empty: stopped

Table 14-6: RegRfifTxSta

pos.	RegRfifRx	rw	reset	description
7-0	RfifRx	r	00000000 nresetglobal	Received data FIFO, depth = 4

Table 14-7: RegRfifRx

pos.	RegRfifRxSta	rw	reset	description
7-5	-	-	-	-
4	RfifRxFifoOverrun	r/c1	0 nresetglobal	Rx Fifo overrun error. Write: clear FIFO
3	RfifRxFifoFull	r	0 nresetglobal	Rx Fifo full [IRQ]
2	RfifRxStartDet	r/c1	0 nresetglobal	Start Stream Detected [IRQ]
1	RfifRxBusy	r	0 nresetglobal	Rfif busy receiving
0	RfifRxReady = RfifRxNotEmpty	r	0 nresetglobal	**RegRfifRx** contains at least one byte to read. Cleared when reading **RegRfifRx**.[IRQ]

Table 14-8: RegRfifRxSta

pos.	RegRfifRxSPat	rw	reset	description
7-0	RfifRxSPat	r/w	00000000 nresetglobal	Programmable start pattern

Table 14-9: RegRfifRxSPat

14.5 Interrupts

Two interrupt lines exist: one for the transmitter and one for the receiver.

interrupt source
Irq_Rfif_Rx
Irq_Rfif_Tx

Table 14-10: Interrupts

14.6 External connections

The interface uses 4 I/O pins. The use of these pins depends on the set up of the interface. Table 14-11 shows the possible use of these pins.

In receive mode, the pin RFIF0 is the receiver data stream input. If the external bit synchronization clock mode is selected, the RFIF1 pin should be connected to the bit synchronizer clock of the receiver. If the receiver has a message synchronization signal, it can be connected to RFIF2.

In transmitter mode, the transmitter data pin has to be connected to RFIF3. A bit synchronization clock can be generated on RFIF2.

	RFIF0	RFIF1	RFIF2	RFIF3
Receive mode	data in	clock	match	-
Transmit mode	-	-	clock	data out

Table 14-11 : PAD management

In case the RFIF pins are multiplexed on a parallel digital I/O port, the interface overwrites all four digital pins as soon as the receive mode or transmitter mode of the interface are enabled (bit **RfifEnRx** or **RfifEnTx** in register **RegRfifCmd3** is set at 1).

Unused pins are configured as inputs with a weak pull-up or pull-down. These pins may remain floating or may be driven externally. No external pull-ups or pull-downs are required.

14.7 Supported PCM codes

Seven different PCM codes are supported by the interface. They are illustrated in the figure below.

NRZ Code: in this format **one message bit is represented by one chip** (i.e. the distance between two dotted lines in Figure 14-2). The coding does not include clock information. The bit sampling clock information has to be extracted from the message preamble.

Bi-Phase Code or Manchester code: in this format **one message bit is represented by two chips.** This code contains clock information.

Delay Modulation or Miller code: in this format **one message bit is represented by two chips**. This code contains clock information.

The interface can receive and transmit coded or uncoded messages. The coding/decoding function is enabled/disabled using the bit **RfifEnCod** (1 = enable) in the register **RegRfifCmd2**.

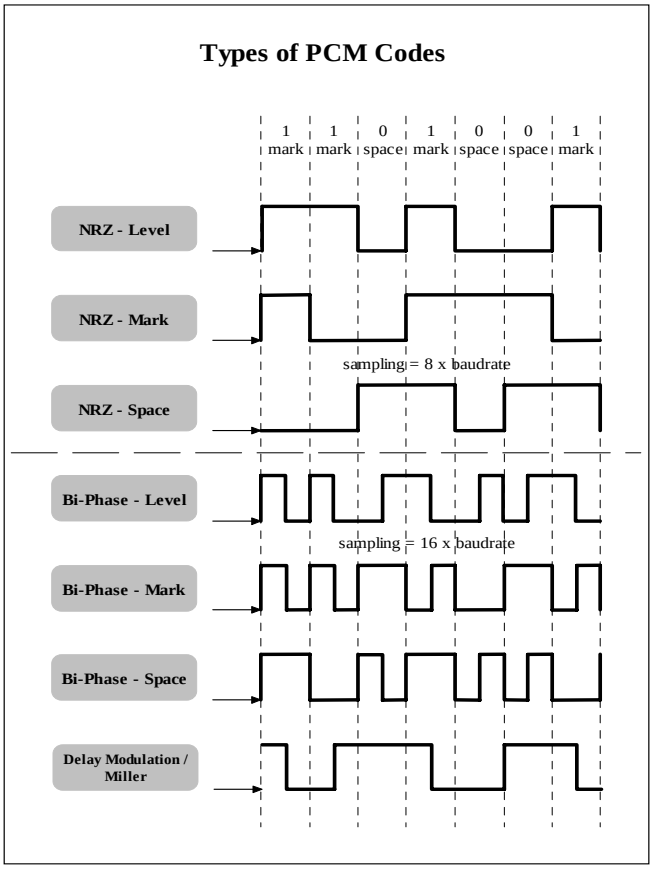


Figure 14-2 : Supported PCM Codes

The PCM code type is selected using the **RfifPCM[2:0]** control word in the register **RegRfifCmd2**. The selection control bits are given in Table 14-12.

PCM Codes	NRZ			Bi-Phase			Miller
	Level	Mark	Space	Level	Mark	Space	
RfifPCM[2:0]	000	001	010	011	100	101	11X

Table 14-12: PCM code selection

14.8 Reception mode: detailed description

The interface is configured in reception mode by setting the bit **RfifEnRx** (1 = enable) and clearing the bit **RfifEnTx** (0 = disable transmission) in the register **RegRfifCmd3**.

The input data stream coming from the receiver has to be connected to the RFIF0 pin.

14.8.1 Input data filter and bit synchronization

14.8.1.1 NRZ code

When the NRZ codes are used, the bit synchronization clock should be extracted from the preamble since the coding itself may not contain clock information. The input data filter has to be used (see description below in section 14.8.1.2) to extract the bit synchronization clock. The bit **RfifRxClock** in register **RegRfifCmd2** is to be cleared to 0. If the input data stream contains long periods without transitions, the internally generated bit synchronization clock could slip with respect to the bit rate resulting in transmission errors (see also specification in chapter 14.11). Tests have shown a very robust behavior with the internal bit synchronizer though.

An external bit synchronization clock can be used. This clock has to be delivered on the pin RFIF1. The bit stream on RFIF0 is then sampled on detection of the rising edge of the clock on the RFIF1 pin. This function has to be enabled by setting bit **RfifRxClock** in register **RegRfifCmd2** to 1. Selecting this function bypasses the input filter. In practice, it was found that the synchronization was more reliable with the internal synchronization filter than with the external bit clock.

14.8.1.2 Bi-phase and Miller code

The bi-phase and Miller code include clock information and no external bit synchronization clock is needed. The input data filter is used to filter the noise on the input data. The filter samples the input data at 16 times the selected baud rate and performs the following majority function on 8 successive samples:

```
if (number of '1' in the last 8 samples >= 6)
 data = 1
if (number of '1' in the last 8 samples <= 2)
 data = 0
else data = former data
```

The filter has some hysteresis since the output does not change when the number of 1's is between 2 and 6.

The input filter is always selected for the bi-phase and Miller coding independent from the value of the bit **RfifRxClock** in register **RegRfifCmd2**.

14.8.2 Start Sequence Detection or message synchronization

A full message is mainly composed of three parts:

1) a preamble or start sequence
2) the data itself
3) an optional stop sequence.

A start sequence is necessary to indicate the start of a new message.

The interface supports the detection of several types of start sequences. The user can chose between these start sequence detection modes using the bits **RfifEnStart[1:0]** in **RegRfifCmd2** as shown in Table 14-13. The actual implementation of the detection mode differs depending on the selected PCM coding and is described below.

A start sequence detection will set the bit **RfifRxStartDet** in the register **RegRfifRxSta** and generate an interrupt on the Irq_Rfif_Rx line. The contents of the reception FIFO is reset. The data of the preceding message that were not yet read at the occurrence of a new start sequence are lost. The first bit in the FIFO is the first valid bit decoded after the detected start sequence.

The bit **RfifRxStartDet** has to be cleared by software by writing a 1 to the bit.

start detection mode	
name	**RfifEnStart[1:0]**
“None”	00
“protocol”	01
“external”	10
“pattern 8bits”	11

Table 14-13. Start detection mode selection

14.8.2.1 No start sequence detection

In this case, the interface will not look for a start sequence. The incoming data stream will be decoded according to the selected PCM code and will be stored in the reception FIFO as soon as the reception mode is enabled. The message start has to be detected by software.

14.8.2.2 “Protocol” start detection

The NRZ coding has no defined start sequence. Selecting the “protocol” start sequence in NRZ has the same effect as no start detection.

In the bi-phase or Manchester code, any protocol violation is considered as a start sequence i.e. a 1-level or 0-level which is longer than the duration of a chip. An example is shown in Figure 14-3.

In the delay modulation or Miller coding, any protocol violation is considered as a start sequence i.e. a 1-level or 0-level which is longer than the duration of 2 chips.

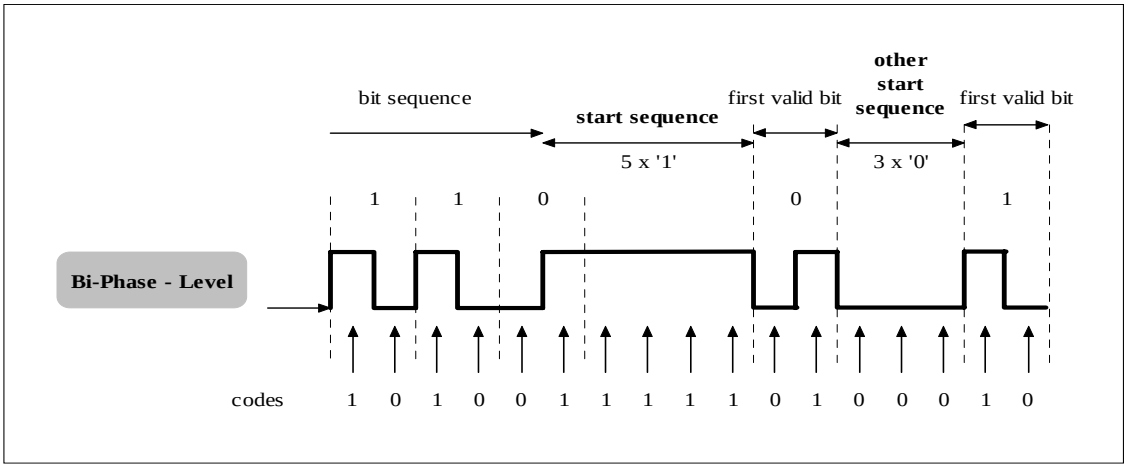


Figure 14-3 : Start sequence type in Bi-Phase-Level encoding

14.8.2.3 “External” start detection

An external start detection signal can be used. This signal should be connected to the pin RFIF2. The falling edge of RFIF2 (also called “match”, see Figure 14-1) indicates the first valid bit of the message. This edge has to occur during the first half of the first bit as shown in Figure 14-4. The minimal width of the pulse is half a bit.

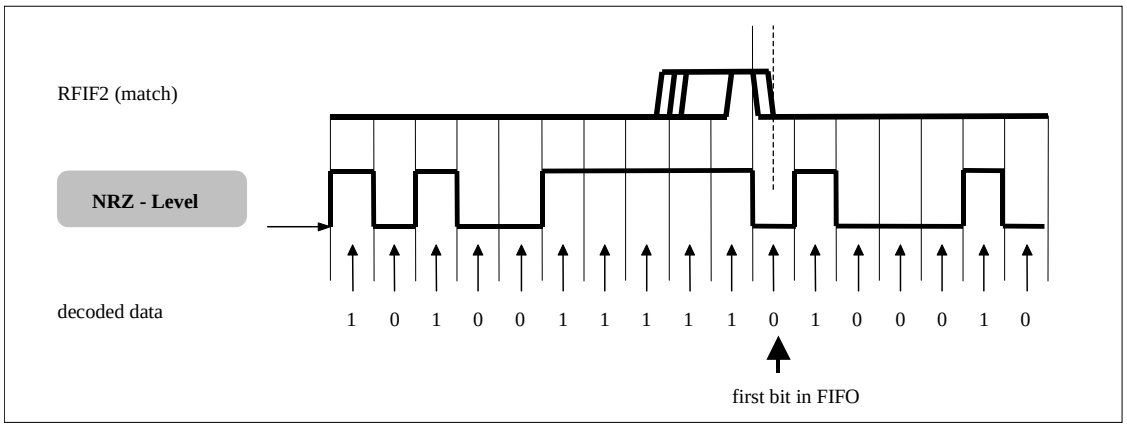


Figure 14-4. Timing of the external word synchronization signal

14.8.2.4 "Pattern" start detection

The start sequence is defined by the pattern **RfifRxSPat[7:0]** written in the register **RegFifRxSPat**. This pattern is compared to the incoming bit stream before decoding. This is shown in Figure 14-5 and Figure 14-6 where an identical input data bit stream and an identical pattern give the same start sequence detection independent from the decoded data.

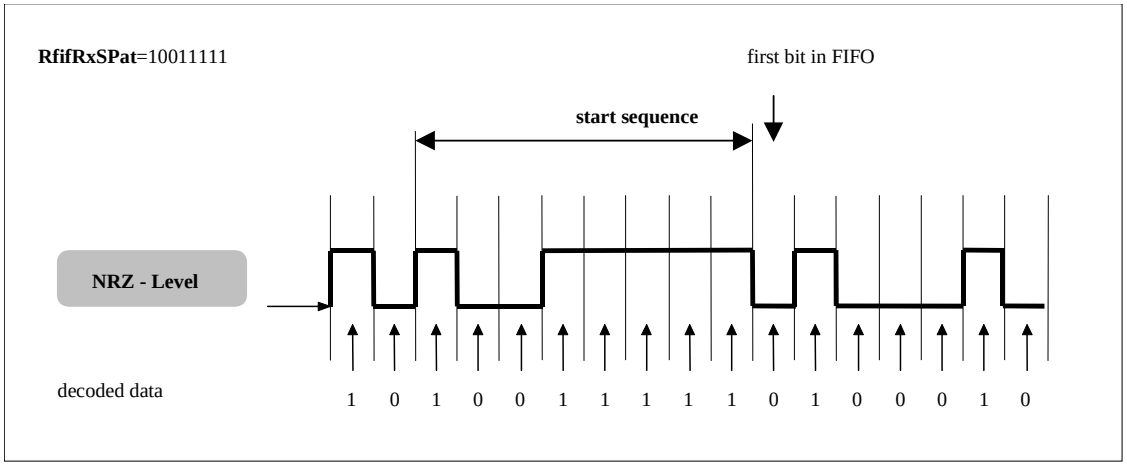


Figure 14-5. Pattern start detection in NRZ-Level for RfifRxSPat[7:0]=10011111

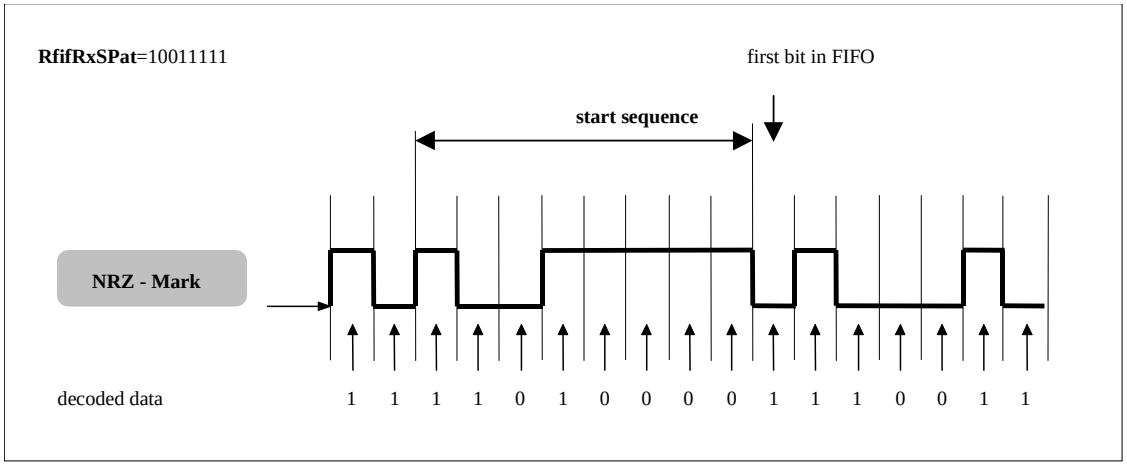


Figure 14-6. Pattern start detection in NRZ-Mark for RfifRxSPat[7:0]=10011111

The start pattern can not be less than 8 bits since no bits can be set as “don’t care”.

14.8.3 PCM decoding

The interface can receive coded or uncoded messages. The coding/decoding function is enabled/disabled using the bit **RfifEnCod** (1 = enable) in the register **RegRfifCmd2**.

The PCM code type is selected using the **RfifPCM[2:0]** control word in the register **RegRfifCmd2**. The selection control bits are given in Table 14-12.

PCM Codes	NRZ			Bi-Phase			Miller
	Level	Mark	Space	Level	Mark	Space	
RfifPCM[2:0]	000	001	010	011	100	101	11X

Table 14-14: PCM code selection

If the decoder detects invalid code sequences and the selected start detection mode is not the “protocol “ mode (**RfifEnStart[1:0]** $\neq$ 01, see Table 14-13), the decoder output generates an equivalent number of random data bits.

If the decoder is disabled (**RfifEnCod** = 0), the meaning of the **RfifPCM[2:0]** changes: if the NRZ coding is selected, the input bit stream is sampled at the selected bit rate. If the Manchester or Miller coding is selected, the input stream is sampled at twice the data rate. This allows the decoding of the bit stream by software.

14.8.4 Reception FIFO

The decoder output data are serially shifted in a shift register (see Figure 14-1). The bit **RfifRxBusy** = 1 in **RegRfifRxSta** indicates that new data are shifted in.

When 8 new bits are present in the shift register, the byte is loaded in the FIFO. The FIFO has a depth of 4. The bit **RfifRxReady** = 1 in **RegRfifRxSta** indicates that at least 1 byte is present in the FIFO. The software can read the data in the FIFO in **RegRfifRx**. When the bit **RfifRxFifoFull** is 1, the FIFO is full and data should be read by the software before the next byte is received. If not, data will be lost and the bit **RfifRxFifoOverrun** is set to 1. Writing a 1 to this bit will clear the bit and the FIFO contents. The detection of a new start sequence will also clear the **RfifRxFifoOverrun** bit and the FIFO contents.

The first bit that is received can be found in the position **RfifRx[0]**, the second bit in the position **RfifRx[1]** and so on.

14.8.5 Reception interrupts

Only one interrupt line is dedicated to the RF receiver interface. The interrupt can be generated by three different sources however. The bits **RfifRxIrqEn[2:0]** in **RegRfifCmd3** allow the selection of the interrupt source(s) as shown in Table 14-15. Possible interrupt sources are the detection of a message start, the shift of the first byte in the FIFO after the FIFO was cleared and finally when the FIFO is full. The bits **RfifRxIrqMem[2:0]** in **RegRfifCmd3** flag which interrupt source was active since these bits were last cleared. The interrupt flags are cleared by writing a 1. More than one interrupt source can be enabled at the same time.

RfifRxIrqEn[2:0] RfifRxIrqMem[2:0]	Interrupt source
XX1	Start sequence detection
X1X	A new byte is written to FIFO
1XX	FIFO is full

Table 14-15. Reception interrupt source selection

The first interrupt allows easy synchronization to the message start. The second interrupt can be used if one wants to download the message one byte at a time. The third interrupt can be used if one wants to download the message 4 bytes at a time.

14.9 Transmission mode

The interface is configured in reception mode by setting the bit **RfifEnTx** (1 = enable) and clearing the bit **RfifEnRx** (0 = disable reception) in the register **RegRfifCmd3**. When the transmission mode is switched off again, the transmission will stop when the transmission of the byte in the shift register is completed.

The output data stream will be generated on the RFIF3 pin.

14.9.1 Transmission FIFO

Data to be transmitted are written to the FIFO by writing to the register **RegRfifTx**. These data are then loaded in the shift register which sends the data bit by bit to the encoder (Figure 14-1). If the transmission FIFO is full, the bit **RfifTxFifoFull** in **RegRfifTxSta** is set to 1. If the software continues to write data to the FIFO while it is full, the flag **RfifTxFifoOverrun** will be set to 1 and the data will be lost. The **RfifTxFifoOverrun** flag is cleared by writing a 1 to it. This clears the contents of the transmission FIFO at the same time. When the last data byte present in the FIFO is loaded in the shift register, the flag **RfifTxFifoEmpty** is set to 1. If the software does not write new data to the FIFO and the shift register finished shifting the last bit to the encoder, the bit **RfifTxStopped** is set to 1. The RFIF3 pin remains in the last encoded state.

The first bit sent is the bit **RfifTx[0]**, the second bit is **RfifTx[1]** and so on.

14.9.2 Transmission interrupts

Only one interrupt source exists during the transmission: an interrupt can be generated at the moment the last byte of the FIFO is transferred to the transmission shift register. It means that the FIFO is empty and has to be refilled with data. The interrupt is enabled in the interrupt manager block.

14.9.3 Transmission encoding

The transmission encoder can be bypassed by clearing the bit **RfifEnCod** in register **RegRfifCmd2**. The bit stream on the RFIF3 pin is then directly connected to the output of the shift register.

If **RfifEnCod** in register **RegRfifCmd2** is set to 1, the bit stream coming from the shift register is encoded first before sending it to the RFIF3 pin. The type of protocol is selected using the **RfifPCM[2:0]** control word in the register **RegRfifCmd2**. The selection control bits are given in Table 14-14.

PCM Codes	NRZ			Bi-Phase/Manchester			Miller
	Level	Mark	Space	Level	Mark	Space	
RfifPCM[2:0]	000	001	010	011	100	101	11X

Table 14-16: PCM code selection

When the bit **RfifEnCod** is modified while a transmission is active, the modification will take effect only when a new byte is loaded from the FIFO into the shift register.

While the encoder is enabled, it is not possible to send a protocol violation as a start pattern. If a protocol violation is used as a start sequence in the Manchester or Miller protocol, the following sequence should be used:

1) wait until the transmission FIFO is empty (which means the last byte of the preceding message is being coded and sent),
2) clear **RfifEnCod**,
3) write the uncoded start pattern to the transmission FIFO
4) wait until the transmission FIFO is empty (which means the last byte of the uncoded start pattern is being sent)
5) set **RfifEnCod**
6) write the message bytes to the FIFO
7) back to 1.

Beware that in the case a protocol violation is used as a start sequence in Manchester-level or Miller coding, the used violation will depend on the value of the first bit of the message. As an example: if the Manchester level coding is used and the first bit of the message is a 0 (encoded 01), the start sequence needs to be a series of 1's. If a series of 0's would be used, the receiver would be unable to distinguish the leading 0 of the message from the start sequence.

When the encoder is bypassed (**RfifEnCod** = 0), bits **RfifPCM[2:0]** still have an influence on the output bit stream : when the NRZ coding is chosen, the bits are shifted out at the selected bit rate. If the Manchester or Miller codes are selected however, the bits are shifted out at twice the data rate. This allows the use of two bits in the FIFO to encode the bits by software.

14.9.4 Transmission synchronization clock

If required, a bit synchronization clock can be generated on the pin RFIF2 if the data is uncoded or if the NRZ coding is used (bits **RfifEnCod** and **RfifPCM[2:0]** in **RegRfifCmd2**). To generate this clock, the bit **RfifTxClock** in **RegRfifCmd2** has to be set. The timing of the generated clock is shown in Figure 14-7.

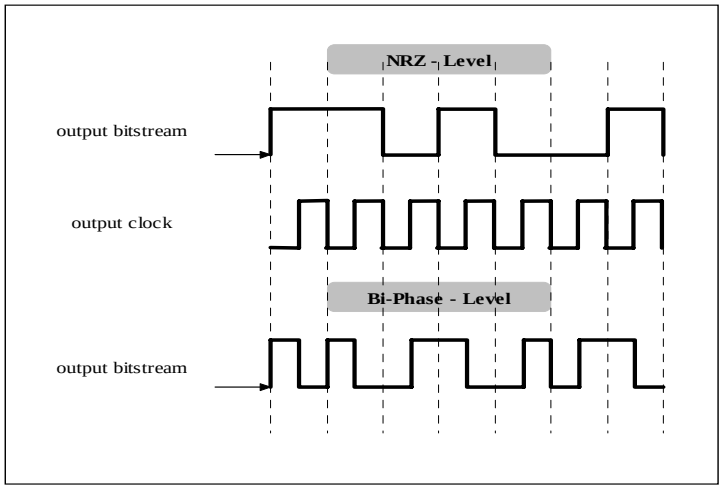


Figure 14-7 : Output clock in Transmission mode

14.10 Baud rate selection

In order to have correct baud rates, the interface has to be fed with a stable and trimmed clock source. The clock source can be an external clock source or the RC oscillator. The precision of the baud rate will depend on the precision of the selected clock source. The choice between the RC oscillator and the external clock source is made with the bit **EnExtClock** in **RegSysClock**. The precision of the obtained baud rate is directly proportional to the frequency deviation of the used clock source.

The bits **RfifBRCoarse[1:0]** and **RfifBRFine[3:0]** in **RegRfifCmd1** are used to select the appropriate baud rate. Several RC division factors are available.

The expression of the baud rate is the following:

$$baudrate = \frac{f_{RCExt}}{fine * coarse * 16}$$

with: f_{RCExt} : RC frequency or the half the external clock frequency.
fine and coarse: the division factors selected by **RfifBRCoarse** and **RfifBRFine**.

Note that the baud rate is the rate at which the chips are transmitted and is twice the bit rate in Manchester and Miller coding since these codes use two chips per bit (see chapter 14.7).

RfifBRFine[3:0]	fine
0000	1
0001	2
0010	3
0011	4
...	...
1101	14
1110	15
1111	16

Table 14-17 : Effect of RfifBRFine[3:0]

RfifBRCoarse[1:0]	coarse
00	1
01	4
10	8
11	16

Table 14-18 : Effect of RfifBRCoarse[1:0]

Even if the external bit synchronizer is used, the baud rate should be set correctly.

14.11 Specifications

If a frequency difference exists between the transmitter and the receiver, the receiver may be unable to decode the received messages correctly. This is especially the case if no protocol or if the NRZ protocols are used without an external bit synchronization clock since these signals do not contain clock information. Manchester and Miller coding are more robust since the decoder can resynchronize on the code itself.

The estimated maximal tolerated frequency difference between the transmitter and the RF interface are the following (theoretical values):

Mode	Maximal tolerated $\Delta f=f_{RX}-f_{TX}$
NRZ, no external clock	to receive 10 identical and consecutive bits: ±4%
NRZ, with external clock	-50%/+100%
Manchester	-17%/+11%
Miller	-9%/+8%

Table 14-19 : Frequency difference tolerance

In NRZ without external bit synchronizer, it should be noted that the tolerated frequency deviation is larger if the number of consecutive identical bits is lower than 10.

14.12 Application hints

Standard API functions to drive the XE1200 series are given in the TN8000.18. Using these API functions will allow you to set-up a transmission between two systems very quickly.

Below, you will find some examples on how to set-up the RF circuits and the RF interface (BitJockey™).

14.12.1 Using the RF interface with the XE1201A

This chapter gives some examples on how the RF interface can be used with the XE1201A circuit. It does not describe all possibilities but selects some representative cases.

14.12.1.1 Microcontroller – transceiver connections

For the external components required to operate the XE1201A, please refer to the XE1201A datasheet.

The connections between the XE8000 microcontroller and the XE1201A are shown in Figure 14-8. The transmission data pins are connected between the RF interface of the microcontroller and the transceiver. The configuration lines of the XE1201A can be driven by pins of a parallel port. The 3-wire configuration data bus of the XE1201A is used to set-up the circuit by writing in the registers A, B and C. The interface can be driven by a hardware SPI or by a software SPI on a parallel port. Pins of the RF interface RFIF0 to RFIF3 that are not used may remain floating.

If the XE1201A is used as a transmitter only, the connections RFIF0 – RXD and RFIF1 – CLKD are not required. The RXTX pin of the XE1201A can be permanently connected to the ground. RFIF0 and RFIF1 may remain floating.

If the XE1201A is used as a receiver only, the connection RFIF3 – TXD is not required. The pin TXD and the pin TXRX can be permanently connected to VDD. If the bit synchronizer in the XE1201A is not used, the connection RFIF1 – CLKD is not required.

If the XE1201A transmitter is permanently enabled, the pin EN can be permanently connected to VDD.

This documentation shows how to set-up the data transmission between the microcontroller and the XE1201A. For detailed information on the XE1201A functionality, external components, serial interface protocol and register set-up, please refer to the XE1201A datasheet.

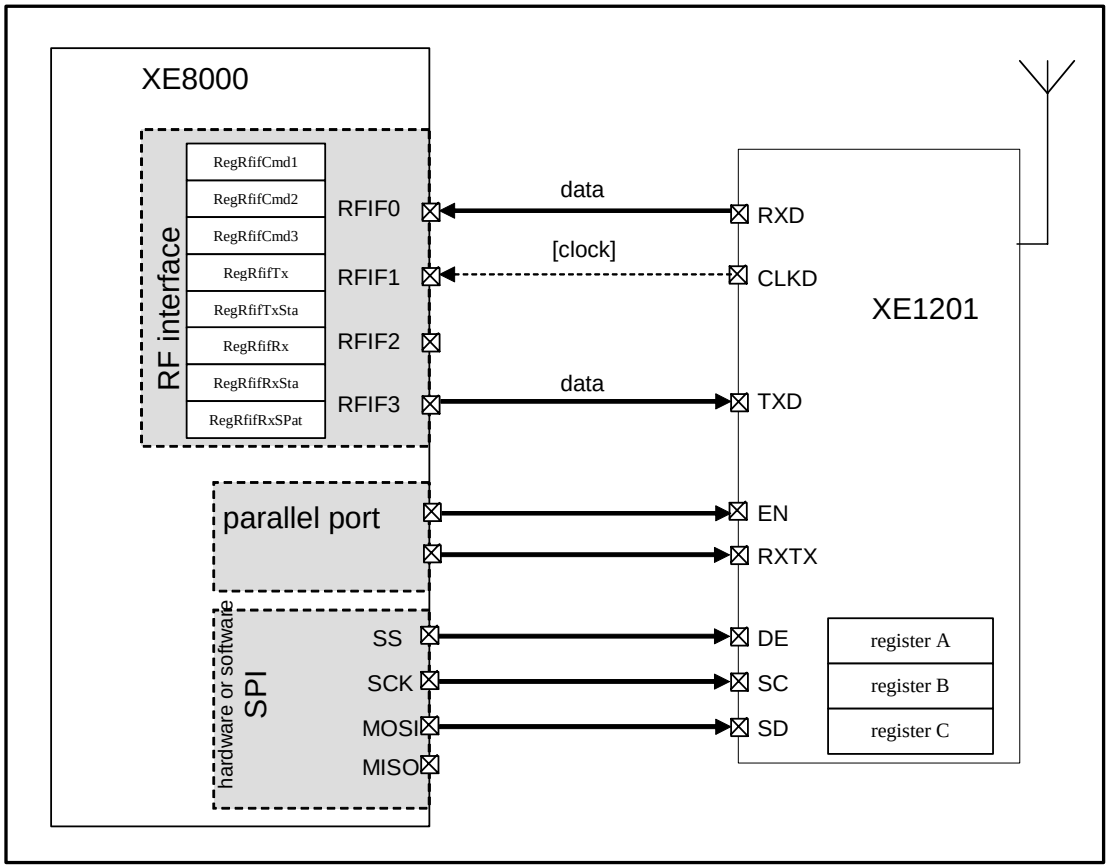


Figure 14-8 : Configuration of the connections between the microcontroller and XE1201A

14.12.1.2 Reception mode using NRZ coding and the XE1201A bit synchronizer

The protocol of the messages received is NRZ level. The start sequence of the message is a 11010111 pattern and the message length after the start sequence is 8 bytes. The messages are sent at a data rate of 16 kbit/s. The following paragraphs will show how to set-up the XE1201A, how to set-up the RF interface and how to handle the received data.

14.12.1.2.1 XE1201A set-up

To set the XE1201A in reception mode, set the pin EN to 1 and the pin RXTX to 1. The data rate of the XE1201A by default is 16kbit/s and the bit synchronizer is enabled by default. This means that the data registers of the XE1201A can remain in the default settings shown in Table 14-20. The serial interface connections of Figure 14-8 are therefore not required.

Register	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register A	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Register B	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Register C	0	1	0	1	0	1	0	0	1	0	0	0	0	0

Table 14-20. XE1201A default register set-up (see XE1201A datasheet for bit explanation)

14.12.1.2.2 RF interface set-up

Set-up the RF interface of the microcontroller circuit as a receiver (**RfifEnRx** = 1 and **RfifEnTx** = 0).

Assume that the RC clock frequency used in the microcontroller is 1.0 MHz. To select the correct baud rate of 16 kbit/s according to the equation in chapter 14.10, *fine*coarse*=1.0e06/(16*16e3)=3.9. This can be approximated at 4 (see specification in Table 14-19). This can be done by setting **RfifBRCoarse** = 01 and **RfifBRFine** = 0000.

The external bit synchronization clock is switched off by setting the bit **RfifRxClock** = 0.

The decoder is enabled and set to NRZ level decoding by setting **RfifEnCod** = 1 and **RfifPCM** = 000.

The start pattern detection is enabled by setting **RfifEnStart** = 11 and writing 11010111 to **RfifRxSPat**.

The start sequence detection interrupt is enabled by setting **RfifRxIrqEn** = 001.

The set-up of the interface is summarized in the Table 14-21.

Register	contents
RegRfifCmd1	00010000
RegRfifCmd2	11100000
RegRfifCmd3	00100010
RegRfifRxSPat	11010111

Table 14-21. RF interface set-up

14.12.1.2.3 Data reception

In order to handle the received data by interrupt, enable the RF interface reception interrupt in the interrupt handler of the circuit.

Data received before the first start pattern detection after the enabling of the interface are not relevant since we are not yet synchronized to the messages. Since the start detection interrupt has been enabled, nothing has to be done until the interrupt occurs.

When the first interrupt occurs, we are synchronized to the messages. In order to read data in an efficient way, the interrupt source is modified and set to “Rx FIFO full” by writing 100 to **RfifRxIrqEn**. Once this is done, we can wait for the next interrupt to download the received message.

At each new interrupt, we can now read 4 bytes of the received message by reading the register **RegRfifRx** 4 consecutive times. The interrupt should be served before the next byte is received since otherwise data may be

lost by lack of space in the FIFO (overrun error which sets the flag **RfifRxFifoOverrun**) or because the start sequence of the next message is detected which resets the reception FIFO.

When the complete message is received, the start sequence detection interrupt may be enabled again (**RfifRxIrqEn** = 001) and the sequence starts all over again.

14.12.1.3 Reception mode using Manchester coding

The protocol of the messages received is Manchester Mark. The start sequence of the message is protocol violation. The messages are sent at a data rate of 20 kbit/s (40 kbaud/s in Manchester code). The following paragraphs will show how to set-up the XE1201A, how to set-up the RF interface and how to handle the received data.

14.12.1.3.1 XE1201A set-up

To set the XE1201A in reception mode, set the pin EN to 1 and the pin RXTX to 1. The bit synchronizer of the XE1201A is not used since the RF interface automatically synchronizes to the bit rate. This means that the data registers of the XE1201A can remain in the default settings shown in Table 14-20. However, in order to reduce the power consumption, the XE1201A bit synchronizer can be switched off by writing the bits A(9:6) = 0101 as shown in Table 14-22.

Register	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register A	0	0	0	0	0	1	0	1	0	1	0	0	0	0
Register B	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Register C	0	1	0	1	0	1	0	0	1	0	0	0	0	0

Table 14-22. XE1201A register set-up (see XE1201A datasheet for bit explanation)

14.12.1.3.2 RF interface set-up

Set-up the RF interface of the microcontroller circuit as a receiver (**RfifEnRx** = 1 and **RfifEnTx** = 0).

Assume that the RC clock frequency used in the microcontroller is 2.0 MHz. To select the correct baud rate of 20 kbit/s according to the equation in chapter 14.10 (attention: the baud rate is twice the bit rate in Manchester code), *fine*coarse*=2.0e06/(16*40e3)=3.12. We can approximate this at 3, which is close enough (see specification in Table 14-19). This can be done by setting **RfifBRCoarse** = 00 and **RfifBRFine** = 0010.

The external bit synchronization clock is switched off by clearing the bit **RfifRxClock** = 0.

The decoder is enabled and set to Manchester Mark decoding by setting **RfifEnCod** = 1 and **RfifPCM** = 100.

The start detection by protocol violation is enabled by setting **RfifEnStart** = 01.

The start sequence detection interrupt is enabled by setting **RfifRxIrqEn** = 001.

The set-up of the interface is summarized in the Table 14-23.

Register	contents
RegRfifCmd1	00000010
RegRfifCmd2	10100100
RegRfifCmd3	00100010

Table 14-23. RF interface set-up

14.12.1.3.3 Data reception

In order to handle the received data by interrupt, enable the RF interface reception interrupt in the interrupt handler of the circuit.

Data received before the first start pattern detection after the enabling of the interface are not relevant since we are not yet synchronized to the messages. Sine the start detection interrupt has been enabled, nothing has to be done until the interrupt occurs.

When the first interrupt occurs, we are synchronized to the messages. In order to read data in an efficient way, the interrupt source is modified and set to “Rx FIFO full” by writing 100 to **RfifRxIrqEn**. Once this is done, we can wait for the next interrupt to download the received message.

At each new interrupt, we can now read 4 bytes of the received message by reading the register **RegRfifRx** 4 consecutive times. The interrupt should be served before the next byte is received since otherwise data may be lost by lack of space in the FIFO (overrun error which sets the flag **RfifRxFifoOverrun**) or because the start sequence of the next message is detected which resets the reception FIFO.

When the complete message is received, the start sequence detection interrupt may be enabled again (**RfifRxIrqEn** = 001) and the sequence starts all over again.

14.12.1.4 Transmission mode using NRZ coding

The messages have to be sent encoded with NRZ Space. The start sequence of the message is 4 bytes of “01010101”. The messages are sent at a data rate of 64 kbit/s and the modulator frequency deviation is 125kHz and the output power is –5dbm.

The following paragraphs will show how to set-up the XE1201A, how to set-up the RF interface and how to handle the transmission data.

14.12.1.4.1 XE1201A set-up

To set the XE1201A in transmission mode, set the pin EN to 1 and the pin RXTX to 0. The modulator deviation frequency is 125kHz by default and the output power is –5dbm by default, so there is no need to modify the registers of the XE1201A.

14.12.1.4.2 RF interface set-up

Set-up the RF interface of the microcontroller circuit as a transmitter (**RfifEnRx** = 0 and **RfifEnTx** = 1).

Assume that the RC clock frequency used in the microcontroller is 1.0 MHz. To select the correct baud rate of 64 kbit/s according to the equation in chapter 14.10, *fine*coarse*=1.0e06/(16*64e3)=0.98. We can approximate this at 1, which is close enough (see specification in Table 14-19). This can be done by setting **RfifBRCoarse** = 00 and **RfifBRFine** = 0000.

The external bit synchronization clock is switched off by clearing the bit **RfifTxClock** = 0.

The encoder is enabled and set to NRZ space encoding by setting **RfifEnCod** = 1 and **RfifPCM** = 010.

The set-up of the interface is summarized in the Table 14-24.

Register	contents
RegRfifCmd1	00000000
RegRfifCmd2	00100010
RegRfifCmd3	00000001

Table 14-24. RF interface set-up

14.12.1.4.3 Data transmission

To handle the transmission data by interrupt, enable the RF interface transmission interrupt in the interrupt handler of the circuit.

We need to send the start sequence (preamble) first. The start sequence is defined as 4 bytes of "01010101". Since the NRZ Space encoder is enabled, four bytes of "00000000" should be written to the register **RegRfifTx**. After encoding, this will correspond to 4 bytes of "01010101". The FIFO is now full, and we can wait for the interrupt, which will indicate that the FIFO is empty.

At occurrence of the interrupt, 4 bytes of the message can be written to the FIFO until all message bytes are sent.

14.12.1.5 Transmission mode using Miller code

The messages have to be encoded using the Miller code. The start sequence of the message is a protocol violation. The messages are sent at a data rate of 5 kbit/s (or 10 kbaud/s in Miller code) and the modulator frequency deviation is 25kHz and the output power is +5dbm.

The following paragraphs will show how to set-up the XE1201A, how to set-up the RF interface and how to handle the transmission data.

14.12.1.5.1 XE1201A set-up

To set the XE1201A in transmission mode, set the pin EN to 1 and the pin RXTX to 0. The modulator deviation frequency is set to 23.4kHz by writing C(6:0) = 0000110. The output power is set to +5dbm by writing C(13:12) = 11. All other values remain default. The resulting register values are shown in Table 14-25.

Register	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register A	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Register B	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Register C	1	1	0	1	0	1	0	0	0	0	0	1	1	0

Table 14-25. XE1201A register set-up (see XE1201A datasheet for bit explanation)

14.12.1.5.2 RF interface set-up

Set-up the RF interface of the microcontroller circuit as a transmitter (**RfifEnRx** = 0 and **RfifEnTx** = 1).

Assume that the RC clock frequency used in the microcontroller is 2.0 MHz. To select the correct baud rate of 5 kbit/s according to the equation in chapter 14.10 (attention: the baud rate is twice the bit rate in the Miller code), *fine*coarse*=2.0e06/(16*10e3)=12.5. We can approximate this at 12, which gives us a data rate of 5.2 kbit/s. This can be done by setting **RfifBRCoarse** = 01 and **RfifBRFine** = 0010. If a data rate closer to the 5kbit/s target, the RC oscillator frequency could be slightly increased to 1.04 MHz which gives *fine*coarse*=13.

The external bit synchronization clock is switched off by clearing the bit **RfifTxClock** = 0.

The encoder is enabled and set to Miller encoding by setting **RfifEnCod** = 1 and **RfifPCM** = 111.
The set-up of the interface is summarized in Table 14-26.

Register	contents
RegRfifCmd1	00010010
RegRfifCmd2	00100111
RegRfifCmd3	00000001

Table 14-26. RF interface set-up

14.12.1.5.3 Data transmission

To handle the transmission data by interrupt, enable the RF interface transmission interrupt in the interrupt handler of the circuit.

We need to send the start sequence first. The start sequence is defined as a protocol violation. To send a protocol violation, the encoder has to be disabled by clearing **RfifEnCod** = 0. We then can write for instance “00000000” to **RfifTx**.

At occurrence of the interrupt, the start sequence was transferred to the shift register and the FIFO is empty. The encoder is to be enabled by setting **RfifEnCod** = 1. We do not need to wait until the start sequence transmission is completed, since the modification of the bit **RfifEnCod** is taken into account only at the transfer of the next byte from the FIFO to the transmission shift register. The first 4 bytes of the message can then be written to **RfifTx**.

At each occurrence of the interrupt, 4 bytes of the message can be written to the FIFO until the end of the message is reached. The whole operation starts over again for the next message.

14.12.2 Using the RF interface with the XE1202A

This chapter gives some examples on how the RF interface can be used with the XE1202A circuit. It does not describe all possibilities but selects some representative cases.

14.12.2.1 Microcontroller – transceiver connections

For the external components required to operate the XE1202A, please refer to the XE1202A datasheet.

The connections between the XE8000 microcontroller and the XE1202 are shown in Figure 14-9. The transmission data are connected between the RF interface of the microcontroller and the transceiver. The configuration lines of the XE1202A can be driven by pins of a parallel port. The 3-wire configuration data bus of the XE1202A is used to set-up the circuit by writing in the registers RTParam, FSParam, ADParam and Pattern. The interface can be driven by a hardware SPI or by a software SPI on a parallel port. Pins of the RF interface RFIF0 to RFIF3 that are not used may remain floating.

If the XE1202A is used as a transmitter only, the connections RFIF0 – RXD, RFIF1 – CLKD and RFIF2 – PATTERN are not required.

If the XE1202A is used as a receiver only, the connection RFIF3 – TXD is not required. If the bit synchronizer in the XE1202A is not used, the connection RFIF1 – CLKD is not required. If the start sequence detection is not used, the connection RFIF2 – PATTERN is not required.

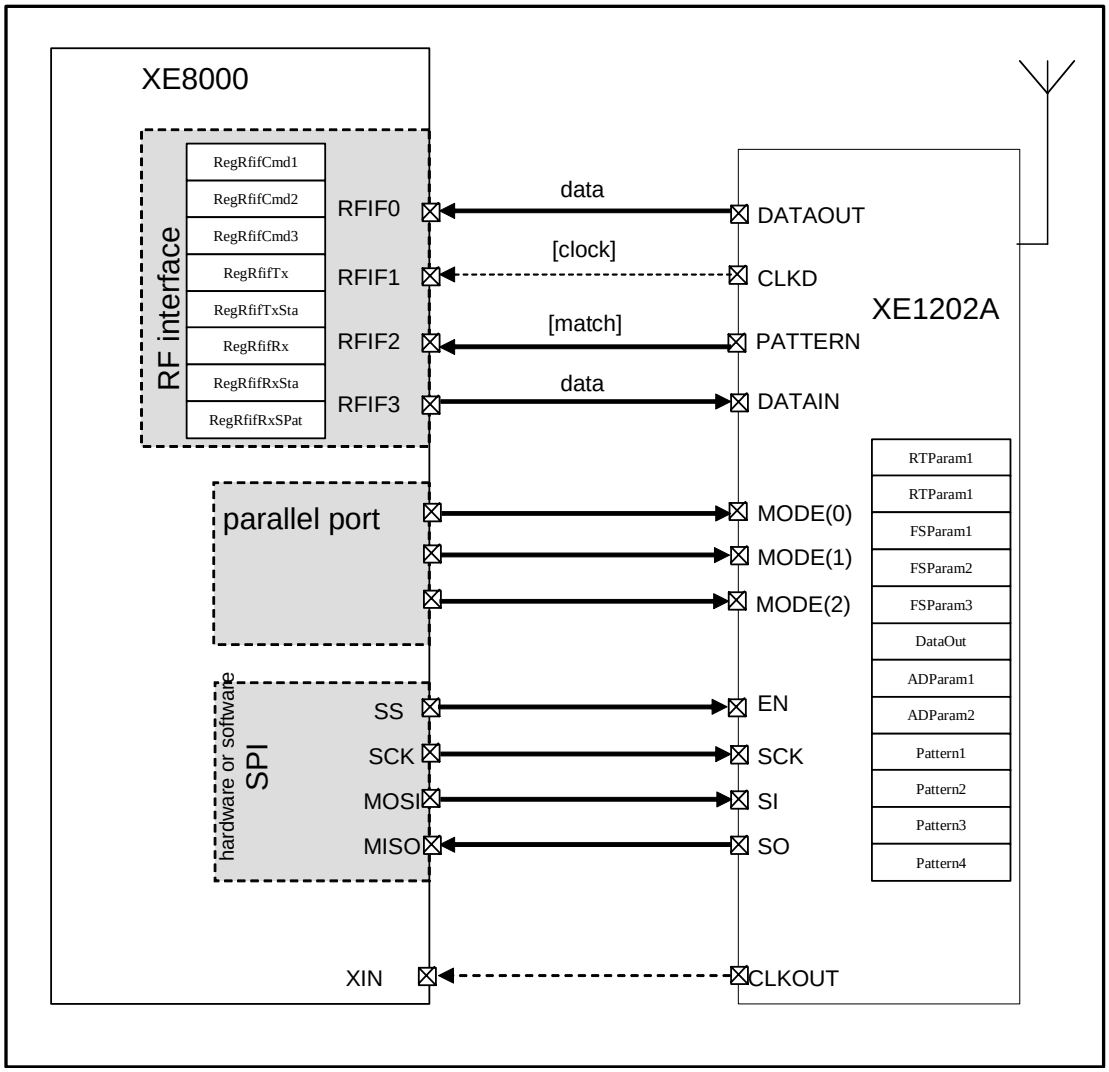


Figure 14-9 : Configuration of the connections between the microcontroller and XE1202A

This documentation shows how to set-up the data transmission between the microcontroller and the XE1202A. For detailed information on the XE1202A functionality, external components, serial interface protocol and register set-up, please refer to the XE1202A datasheet.

14.12.2.2 Microcontroller clock source derived from XE1202A crystal oscillator

The XE1202A has a programmable output clock derived from the 39MHz crystal oscillator. This clock can be routed to the external clock input of the microcontroller (see Figure 14-9 and the clock block documentation). In this case, a precise clock derived from the XE1202A can be used to run the microcontroller in stead of the internal RC oscillator. The advantage is that the crystal oscillator is much more precise than the internal RC but the overall power consumption of the system will be somewhat larger.

To enable the external clock output of the XE1202A, set the bit RTParam_Clkout=1 and use the bits ADParam_Clkfreq[1:0] to select the frequency between 9.75MHz and 1.22MHz (see XE1202A datasheet for details).

To select the external clock in the microcontroller, set the bit **EnExtClk** = 1 in the register **RegSysClk**. Be aware that the external clock is divided by 2 at the input (see the clock block documentation for more details).

The first example below (chapter 14.12.2.3) uses the XE1202A clock. In the other examples, the internal RC clock is used.

14.12.2.3 Reception mode using NRZ coding and the XE1202A bit and message synchronizer

This RF link uses a wide band transmission in the 915MHz frequency band. The protocol of the messages received is NRZ Mark. The start sequence of the message is a 4 byte pattern (10101010 10101010 10101010 11010100). The received messages have a data rate of 76.8 kbit/s with a frequency deviation of 100 kHz. The following paragraphs will show how to set-up the XE1202A, how to set-up the RF interface and how to handle the received data. The microcontroller derives its clock from the XE1202A.

14.12.2.3.1 XE1202A set-up

To set the XE1202A in reception mode, set the pins MODE(2:0) = 100.

To select the 915MHz band, the bits FSParam_band = 11. The bandwidth of the baseband filter is set to 200 kHz by writing RTParam_BW = 11. The bit rate is set to 76.8 kbit/s by writing FSParam_BR = 100. The bit synchronizer is enabled (RTParam_bits = 1). The pattern recognition is enabled (ADParam_Pattern =1) and the length of the pattern is set to 32 bit (ADParam_Psize = 11). The pattern is written to the 4 pattern registers. The output clock is enabled (RTParam_Clkout = 1) and its frequency is set to 2.44 MHz (ADParam_Clkfreq = 01).

Register	Register Address	7	6	5	4	3	2	1	0
RTParam1	00000	0	1	0	0	1	1	0	0
RTParam2	00001	0	1	0	0	0	0	0	1
FSParam1	00010	1	1	1	0	0	1	0	0
FSParam2	00011	0	0	0	0	0	0	0	0
FSParam3	00100	0	0	0	0	0	0	0	0
DataOut	00101	0	0	0	0	0	0	0	0
ADParam1	00110	1	1	1	0	0	0	1	0
ADParam2	00111	0	0	0	0	0	0	0	0
Pattern1	01000	1	0	1	0	1	0	1	0
Pattern2	01001	1	0	1	0	1	0	1	0
Pattern3	01010	1	0	1	0	1	0	1	0
Pattern4	01011	1	1	0	1	0	1	0	0

Table 14-27. XE1202 register set-up (see XE1202 datasheet for bit explanation)

14.12.2.3.2 RF interface set-up

We then set-up the RF interface of the microcontroller circuit as a receiver (**RfifEnRx** = 1 and **RfifEnTx** = 0).

The external clock frequency generated by the XE1202A is 2.44 MHz. The external clock source has to be selected by writing **SelExtClock** = 1 in **RegSysClock**. The frequency used in the microcontroller is half of this (see clock block documentation): 1.22 MHz. To select the correct baud rate of 76.8 kbit/s according to the equation in chapter 14.10, *fine*coarse*=1.22e06/(16*76.8e3)=0.99. We can approximate this at 1 (see specification in Table 14-19). This can be done by setting **RfifBRCoarse** = 00 and **RfifBRFine** = 0000.

The external bit synchronization clock is switched off by setting the bit **RfifRxClock** = 0.

The decoder is enabled and set to NRZ Mark decoding by setting **RfifEnCod** = 1 and **RfifPCM** = 001.

The external start pattern detection is enabled by setting **RfifEnStart** = 10.

The start sequence detection interrupt is enabled by setting **RfifRxIrqEn** = 001.

The set-up of the interface is summarized in the Table 14-28.

Register	contents
RegRfifCmd1	00000000
RegRfifCmd2	10100001
RegRfifCmd3	00100010

Table 14-28. RF interface set-up

14.12.2.3.3 Data reception

In order to handle the received data by interrupt, enable the RF interface reception interrupt in the interrupt handler of the circuit.

Data received before the first start pattern detection after the enabling of the interface are not relevant since we are not yet synchronized to the messages. Since the start detection interrupt has been enabled, nothing has to be done until the interrupt occurs.

When the first interrupt occurs, we are synchronized to the messages. In order to read data in an efficient way, the interrupt source is modified and set to “Rx FIFO full” by writing 100 to **RfifRxIrqEn**. Once this is done, we can wait for the next interrupt to download the received message.

At each new interrupt, we can now read 4 bytes of the received message by reading the register **RegRfifRx** 4 consecutive times. The interrupt should be served before the next byte is received since otherwise data may be lost by lack of space in the FIFO (overrun error which sets the flag **RfifRxFifoOverrun**) or because the start sequence of the next message is detected which resets the reception FIFO.

When the complete message is received, the start sequence detection interrupt may be enabled again (**RfifRxIrqEn** = 001) and the sequence starts all over again.

14.12.2.4 Reception mode using Manchester coding

This RF link uses a narrow band transmission in the 869MHz frequency band. The protocol of the messages received is Manchester Level. The start sequence of the message is a protocol violation. The received messages have a data rate of 2.4 kbit/s (or 4.8kbaud/s in Manchester coding) with a frequency deviation of 5kHz. The following paragraphs will show how to set-up the XE1202A, how to set-up the RF interface and how to handle the received data.

14.12.2.4.1 XE1202A set-up

To set the XE1202A in reception mode, set the pins MODE(2:0) = 100.

To select the 869MHz band, the bits FSParam_band = 10. The bandwidth of the baseband filter is set to 10kHz by writing RTParam_BW = 00. The baud rate is set to 4.8 kb/s by writing FSParam_BR = 000. The bit synchronizer is enabled (RTParam_bits = 1). The pattern recognition is disabled (ADParam_Pattern =0).

Register	Register Address	7	6	5	4	3	2	1	0
RTParam1	00000	0	1	0	0	0	0	0	0
RTParam2	00001	0	1	0	0	0	0	0	0
FSParam1	00010	1	0	0	0	0	0	0	0
FSParam2	00011	0	0	0	0	0	0	0	0
FSParam3	00100	0	0	0	0	0	0	0	0
DataOut	00101	0	0	0	0	0	0	0	0
ADParam1	00110	0	0	0	0	0	0	0	0
ADParam2	00111	0	0	0	0	0	0	0	0

Pattern1	01000	0	0	0	0	0	0	0	0
Pattern2	01001	0	0	0	0	0	0	0	0
Pattern3	01010	0	0	0	0	0	0	0	0
Pattern4	01011	0	0	0	0	0	0	0	0

Table 14-29. XE1202A register set-up (see XE1202A datasheet for bit explanation)

14.12.2.4.2 RF interface set-up

We then set-up the RF interface of the microcontroller circuit as a receiver (**RfifEnRx** = 1 and **RfifEnTx** = 0).

Assume that the RC clock frequency used in the microcontroller is 1.0 MHz. To select the correct baud rate of 4.8 kbit/s according to the equation in chapter 14.10 (attention: the baud rate is twice the bit rate in the Manchester code), *fine*coarse*=1.0e06/(16*4.8e3)=13.0. This can be done by setting **RfifBRCoarse** = 00 and **RfifBRFine** = 1100.

The external bit synchronization clock is switched off by clearing the bit **RfifRxClock** = 0.

The decoder is enabled and set to Manchester Level decoding by setting **RfifEnCod** = 1 and **RfifPCM** = 011.

The start detection by protocol violation is enabled by setting **RfifEnStart** = 01.

The start sequence detection interrupt is enabled by setting **RfifRxIrqEn** = 001.

The set-up of the interface is summarized in the Table 14-30.

Register	contents
RegRfifCmd1	00000010
RegRfifCmd2	10100100
RegRfifCmd3	00100010

Table 14-30. RF interface set-up

14.12.2.4.3 Data reception

In order to handle the received data by interrupt, enable the RF interface reception interrupt in the interrupt handler of the circuit.

Data received before the first start pattern detection after the enabling of the interface are not relevant since we are not yet synchronized to the messages. Sine the start detection interrupt has been enabled, nothing has to be done until the interrupt occurs.

When the first interrupt occurs, we are synchronized to the messages. In order to read data in an efficient way, the interrupt source is modified and set to “Rx FIFO full” by writing 100 to **RfifRxIrqEn**. Once this is done, we can wait for the next interrupt to download the received message.

At each new interrupt, we can now read 4 bytes of the received message by reading the register **RegRfifRx** 4 consecutive times. The interrupt should be served before the next byte is received since otherwise data may be lost by lack of space in the FIFO (overrun error which sets the flag **RfifRxFifoOverrun**) or because the start sequence of the next message is detected which resets the reception FIFO.

When the complete message is received, the start sequence detection interrupt may be enabled again (**RfifRxIrqEn** = 001) and the sequence starts all over again.

14.12.2.5 Transmission mode using NRZ coding

This RF link uses a wide band transmission in the 915MHz frequency band. The protocol of the messages to be sent is NRZ Level. The start sequence of the message is a 4 byte pattern (10101010 10101010 10101010 11010100). The messages have a data rate of 76.8 kbit/s with a frequency deviation of 100 kHz. The output power is 0 dBm. The following paragraphs will show how to set-up the XE1202A, how to set-up the RF interface and how to handle the received data.

14.12.2.5.1 XE1202A set-up

To set the XE1202A in transmission mode, set the pins MODE(2:0) = 111.

To select the 915MHz band, the bits FSParam_band = 11. The frequency deviation is set to 100kHz by writing RTParam_Dev = 11. The baud rate is set to 76.8 kbit/s by writing FSParam_BR = 100. The output power is set to 0 dBm by writing RTParam_Tpow = 00. The setup of the XE1202a is shown in Table 14-31.

Register	Register Address	7	6	5	4	3	2	1	0
RTParam1	00000	0	0	0	0	0	0	0	0
RTParam2	00001	0	1	0	0	0	0	0	0
FSParam1	00010	1	1	1	0	0	1	0	0
FSParam2	00011	0	0	0	0	0	0	0	0
FSParam3	00100	0	0	0	0	0	0	0	0
DataOut	00101	0	0	0	0	0	0	0	0
ADParam1	00110	0	0	0	0	0	0	0	0
ADParam2	00111	0	0	0	0	0	0	0	0
Pattern1	01000	0	0	0	0	0	0	0	0
Pattern2	01001	0	0	0	0	0	0	0	0
Pattern3	01010	0	0	0	0	0	0	0	0
Pattern4	01011	0	0	0	0	0	0	0	0

Table 14-31. XE1202A register set-up (see XE1202A datasheet for bit explanation)

14.12.2.5.2 RF interface set-up

We then set-up the RF interface of the microcontroller circuit as a transmitter (**RfifEnRx** = 0 and **RfifEnTx** = 1).

Assume that the RC clock frequency used in the microcontroller is 1.2 MHz. To select the correct baud rate of 76.8 kbit/s according to the equation in chapter 14.10, *fine*coarse*=1.2e06/(16*76.8e3)=0.98. We can approximate this at 1 (see specification in Table 14-19). This can be done by setting **RfifBRCoarse** = 00 and **RfifBRFine** = 0000.

The external bit synchronization clock is switched off by clearing the bit **RfifTxClock** = 0.

The encoder is enabled and set to NRZ Level encoding by setting **RfifEnCod** = 1 and **RfifPCM** = 000.

The set-up of the interface is summarized in the Table 14-32.

Register	contents
RegRfifCmd1	00000000
RegRfifCmd2	00100010
RegRfifCmd3	00000001

Table 14-32. RF interface set-up

14.12.2.5.3 Data transmission

To handle the transmission data by interrupt, enable the RF interface transmission interrupt in the interrupt handler of the circuit.

We need to send the start sequence (preamble) first. The start sequence is defined as 10101010 10101010 10101010 11010100. Since the NRZ Level encoder is enabled, as “10101010”, “10101010”, “10101010” and “11010100” should be written to the register **RegRfifTx**. The FIFO is now full, and we can wait for the interrupt which will indicate that the FIFO is empty. At occurrence of the interrupt, 4 bytes of the message can be written to the FIFO until all message bytes are sent.

14.12.2.6 Transmission mode using Manchester code

This RF link uses a transmission in the 433MHz frequency band. The protocol of the messages to be sent is Manchester Space. The start sequence of the message is a code violation. The messages have a data rate of 19.2 kbit/s with a frequency deviation of 40 kHz. The output power is 15 dBm. The following paragraphs will show how to set-up the XE1202A, how to set-up the RF interface and how to handle the received data.

14.12.2.6.1 XE1202A set-up

To set the XE1202A in transmission mode, set the pins MODE(2:0) = 111.

To select the 433 MHz band, the bits FSParam_band = 01. The frequency deviation is set to 40 kHz by writing RTParam_Dev = 10. The baud rate is set to 38.4 kb/s by writing FSParam_BR = 011. The output power is set to 15 dbm by writing RTParam_Tpow = 11. The setup of the XE1202A is shown in Table 14-33.

Register	Register Address	7	6	5	4	3	2	1	0
RTParam1	00000	0	0	0	0	0	0	1	1
RTParam2	00001	0	1	0	0	0	0	0	0
FSParam1	00010	0	1	0	1	1	0	1	1
FSParam2	00011	0	0	0	0	0	0	0	0
FSParam3	00100	0	0	0	0	0	0	0	0
DataOut	00101	0	0	0	0	0	0	0	0
ADParam1	00110	0	0	0	0	0	0	0	0
ADParam2	00111	0	0	0	0	0	0	0	0
Pattern1	01000	0	0	0	0	0	0	0	0
Pattern2	01001	0	0	0	0	0	0	0	0
Pattern3	01010	0	0	0	0	0	0	0	0
Pattern4	01011	0	0	0	0	0	0	0	0

Table 14-33. XE1202A register set-up (see XE1202A datasheet for bit explanation)

14.12.2.6.2 RF interface set-up

We then set-up the RF interface of the microcontroller circuit as a transmitter (**RfifEnRx** = 0 and **RfifEnTx** = 1).

Assume that the RC clock frequency used in the microcontroller is 1.2 MHz. To select the correct baud rate of 19.2 kbit/s according to the equation in chapter 14.10 (attention: the baud rate is twice the bit rate in the Manchester code), *fine*coarse*=1.2e06/(16*38.4e3)=1.95. We can approximate this at 2. This can be done by setting **RfifBRCoarse** = 00 and **RfifBRFine** = 0001.

The external bit synchronization clock is switched off by clearing the bit **RfifTxClock** = 0.

The encoder is enabled and set to Manchester Space encoding by setting **RfifEnCod** = 1 and **RfifPCM** = 010.

The set-up of the interface is summarized in Table 14-34.

Register	contents
RegRfifCmd1	00000001
RegRfifCmd2	00100010
RegRfifCmd3	00000001

Table 14-34. RF interface set-up

14.12.2.6.3 Data transmission

To handle the transmission data by interrupt, enable the RF interface transmission interrupt in the interrupt handler of the circuit.

We need to send the start sequence first. The start sequence is defined as a protocol violation. To send a protocol violation, the encoder has to be disabled by clearing **RfifEnCod** = 0. We then can write for instance “00000000” to **RfifTx**.

At occurrence of the interrupt, the start sequence was transferred to the shift register and the FIFO is empty. The encoder is to be enabled by setting **RfifEnCod** = 1. We do not need to wait until the start sequence transmission is completed, since the modification of the bit **RfifEnCod** is taken into account only at the transfer of the next byte from the FIFO to the transmission shift register. The first 4 bytes of the message can then be written to **RfifTx**.

At each occurrence of the interrupt, 4 bytes of the message can be written to the FIFO until the end of the message is reached. The whole operation starts over again for the next message.

15. Universal Asynchronous Receiver/Transmitter (UART)

15.1 Features

- Full duplex operation with buffered receiver and transmitter.
- Internal baudrate generator with 10 programmable baudrates (300 - 153600).
- 7 or 8 bits word length.
- Even, odd, or no-parity bit generation and detection
- 1 stop bit
- Error receive detection: Start, Parity, Frame and Overrun
- Receiver echo mode
- 2 interrupts (receive full and transmit empty)
- Enable receive and/or transmit
- Invert pad Rx and/or Tx

15.2 Overview

The UART pins are PB[7], which is used as Rx - receive and PB[6] as Tx - transmit.

15.3 Registers map

register name
RegUartCtrl
RegUartCmd
RegUartTx
RegUartTxSta
RegUartRx
RegUartRxSta

Table 15-1: Uart register default addresses

pos.	RegUartCmd	rw	reset	description
7	SelXtal	r/w	0 nresetglobal	Select input clock: 0 = RC/external, 1 = xtal
6	-	r	0	Unused
5-3	UartRcSel(2:0)	r/w	000 nresetglobal	RC prescaler selection
2	UartPM	r/w	0 nresetglobal	Select parity mode: 1 = odd, 0 = even
1	UartPE	r/w	0 nresetglobal	Enable parity: 1 = with parity, 0 = no parity
0	UartWL	r/w	1 nresetglobal	Select word length: 1 = 8 bits, 0 = 7 bits

Table 15-2: RegUartCmd

pos.	RegUartCtrl	rw	reset	description
7	UartEcho	r/w	0 nresetglobal	Enable echo mode: 1 = echo Rx->Tx, 0 = no echo
6	UartEnRx	r/w	0 nresetglobal	Enable uart reception
5	UartEnTx	r/w	0 nresetglobal	Enable uart transmission
4	UartXRx	r/w	0 nresetglobal	Invert pad Rx
3	UartXTx	r/w	0 nresetglobal	Invert pad Tx
2-0	UartBR(2:0)	r/w	101 nresetglobal	Select baud rate

Table 15-3: RegUartCtrl

pos.	RegUartTx	rw	reset	description
7-0	UartTx	r/w	00000000 nresetglobal	Data to be send

Table 15-4: RegUartTx

pos.	RegUartTxSta	rw	reset	description
7-2	-	r	000000	Unused
1	UartTxBusy	r	0 nresetglobal	Uart busy transmitting
0	UartTxFull	r	0 nresetglobal	**RegUartTx** full Set by writing to **RegUartTx** Cleared when transferring **RegUartTx** into internal shift register

Table 15-5: RegUartTxSta

pos.	RegUartRx	rw	reset	description
7-0	UartRx	r	00000000 nresetglobal	Received data

Table 15-6: RegUartRx

pos.	RegUartRxSta	rw	reset	description
7-6	-	r	00	Unused
5	UartRxSErr	r	0 nresetglobal	Start error
4	UartRxPErr	r	0 nresetglobal	Parity error
3	UartRxFErr	r	0 nresetglobal	Frame error
2	UartRxOErr	r/c	0 nresetglobal	Overrun error Cleared by writing **RegUartRxSta**
1	UartRxBusy	r	0 nresetglobal	Uart busy receiving
0	UartRxFull	r	0 nresetglobal	**RegUartRx** full Cleared by reading **RegUartRx**

Table 15-7: RegUartRxSta

15.4 Interrupts map

interrupt source	default mapping in the interrupt manager
Irq_uart_Tx	**IrqHig(1)**
Irq_uart_Rx	**IrqHig(0)**

Table 15-8: Interrupts map

15.5 UART baud rate selection

In order to have correct baud rates, the UART interface has to be fed with a stable and trimmed clock source. The clock source can be an external clock source, the RC oscillator or the crystal oscillator. The precision of the baud rate will depend on the precision of the selected clock source.

15.6 UART on the RC oscillator or external clock source

To select the external clock or RC oscillator for the Uart, the bit **SelXtal** in **RegUartCmd** has to be 0. The choice between the RC oscillator and the external clock source is made with the bit **EnExtClock** in **RegSysClock**.

In order to obtain a correct baud rate, the RC oscillator or external clock frequency have to be set to one of the frequencies given in the table below. The precision of the obtained baud rate is directly proportional to the frequency deviation of the used clock source with respect to the values in the table below.

Frequency selection for correct Uart baud rates	
RC oscillator (Hz)	**External clock (Hz)**
2'457'600	4'915'200
1'228'800	2'457'600
614'400	1'228'800
307'200	614'400
153'600	307'200
76'800	153'600

Table 15-9a

For each of these frequencies, the baud rate can be selected with the bits **UartBR(2:0)** in **RegUartCtrl** and **UartRcSel(2:0)** in **RegUartCmd** as shown in Table 15-9.

RC frequency (Hz)		2457600	1228800	614400	307200	153600	76800
External clock freq. (Hz)		4915200	2457600	1228800	614400	307200	153600
UartRcSel	**UartBR**						
000	111	153600	76800	38400	19200	9600	4800
	110	76800	38400	19200	9600	4800	2400
	101	38400	19200	9600	4800	2400	1200
	100	19200	9600	4800	2400	1200	600
	011	9600	4800	2400	1200	600	300
	010	4800	2400	1200	600	300	-
	001	2400	1200	600	300	-	-
	000	1200	600	300	-	-	-
010	001	600	300	-	-	-	-
	000	300	-	-	-	-	-

Table 15-9: Uart baud rate with RC clock or external clock

Note 1 : Although not documented here, the coding of the baud rate used in the circuits XE8801, XE8803 and XE8805 can also be used.

Note 2 : The precision of the baud rate is directly proportional to the frequency deviation of the used clock from the ideal frequency given in the table. In order to increase the precision and stability of the RC oscillator, the DFLL (digital frequency locked loop) can be used with the crystal oscillator as a reference.

15.7 UART on the crystal oscillator

In order to use the crystal oscillator as the clock source for the UART, the bit **SelXtal** in **RegUartCmd** has to be set. The crystal oscillator has to be enabled by setting the **EnableXtal** bit in **RegSysClock**. The baud rate selection is done using the **UartBR** bits as shown in Table 15-10.

Xtal freq. (Hz)	32768
UartBR	
011	2400
010	1200
001	600
000	300

Table 15-10: UART baud rate with Xtal clock

Due to the odd ratio between the crystal oscillator frequency and the baud rate, the generated baud rate has a systematic error of –2.48%.

15.8 Function description

15.8.1 Configuration bits

The configuration bits of the Uart serial interface can be found in the registers **RegUartCmd** and **RegUartCtrl**.

The bit **SelXtal** is used to select the clock source (see chapter 15.5). The bits **UartSelRc** and **UartBR** select the baud rate (see chapter 15.5).

The bits **UartEnRx** and **UartEnTx** are used to enable or disable the reception and transmission.

The word length (7 or 8 data bits) can be chosen with **UartWL**. A parity bit is added during transmission or checked during reception if **UartPE** is set. The parity mode (odd or even) can be chosen with **UartPM**.

Setting the bits **UartXRx** and **UartXTx** inverts the Rx respectively Tx signals.

The bit **UartEcho** is used to send the received data automatically back. The transmission function becomes then: Tx = Rx XOR **UartXRx** XOR **UartXTx**.

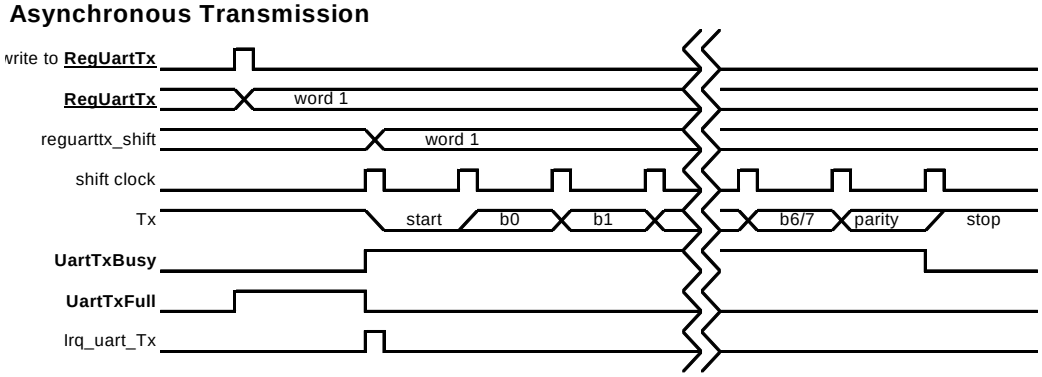
15.8.2 Transmission

In order to send data, the transmitter has to be enabled by setting the bit **UartEnTx**. Data to be sent have to be written to the register **RegUartTx**. The bit **UartTxFull** in **RegUartTxSta** then goes to 1, indicating to the transmitter that a new word is available. As soon as the transmitter has finished sending the previous word, it then loads the contents of the register **RegUartTx** to an internal shift register and clears the **UartTxFull** bit. An interrupt is generated on Irq_uart_Tx at the falling edge of the **UartTxFull** bit. The bit **UartTxBusy** in **RegUartTxSta** shows that the transmitter is busy transmitting a word.

A timing diagram is shown in Figure 15-1. Data is sent LSB first.

New data should be written to the register **RegUartTx** only while **UartTxBusy** is 0, otherwise data will be lost.

Asynchronous Transmission



Asynchronous Transmission (back to back)

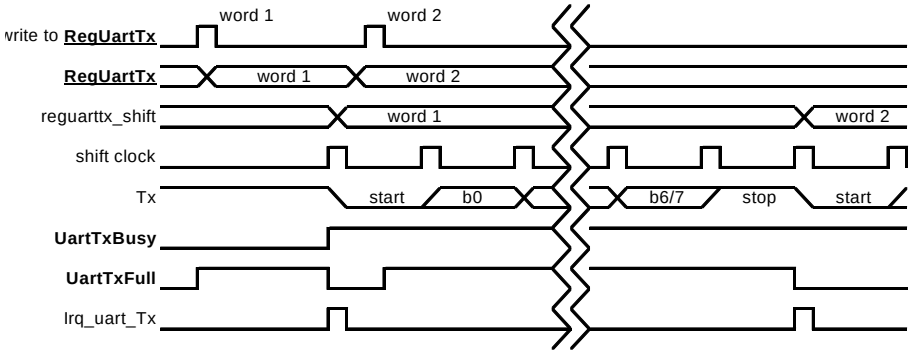


Figure 15-1. UART transmission timing diagram.

15.8.3 Reception

On detection of the start bit, the **UartRxBusy** bit is set. On detection of the stop bit, the received data are transferred from the internal shift register to the register **RegUartRx**. At the same time, the **UartRxFull** bit is set and an interrupt is generated on Irq_uart_Rx. This indicates that new data is available in **RegUartRx**. The timing diagram is shown in Figure 15-2.

The **UartRxFull** bit is cleared when **RegUartRx** is read. If the register was not read before the receiver transfers a new word to it, the bit **UartRxOErr** (overflow error) is set and the previous contents of the register are lost. **UartRxOErr** is cleared by writing any data to **RegUartRxSta**.

The bit **UartRxSErr** is set if a start error has been detected. The bit is updated at data transfer to **RegUartRx**.

The bit **UartRxPErr** is set if a parity error has been detected, i.e. the received parity bit is not equal to the calculated parity of the received data. The bit is updated at data transfer to **RegUartRx**.

The bit **UartRxFErr** in **RegUartRxSta** shows that a frame error has been detected. No stop bit has been detected.

Asynchronous Reception

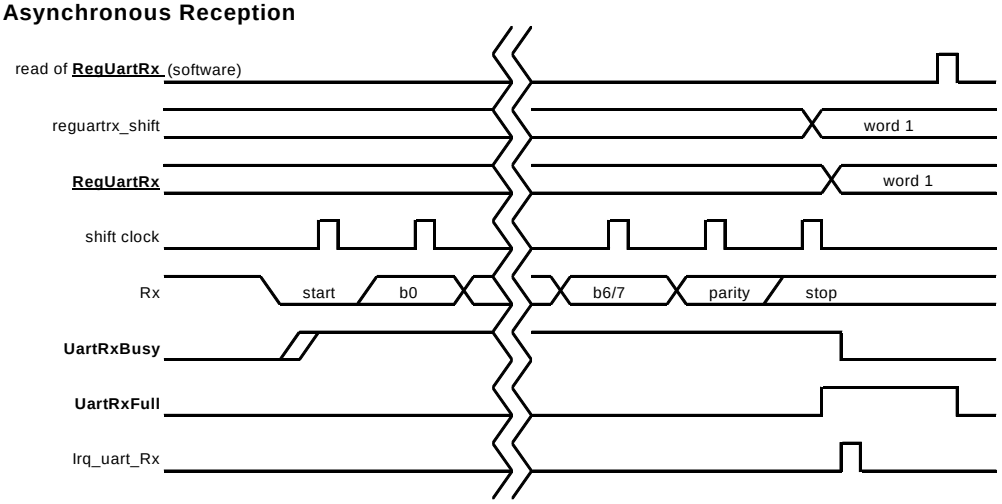


Figure 15-2. UART reception timing diagram.

15.8.4 Interrupt or polling

The transmission and reception software can be driven by interruption or by polling the status bits.

Interrupt driven reception: each time an Irq_uart_Rx interrupt is generated, a new word is available in **RegUartRx**. The register has to be read before a new word is received.
Interrupt driven transmission: each time the contents of **RegUartTx** is transferred to the transmission shift register, an Irq_uart_Tx interrupt is generated. A new word can then be written to **RegUartTx**.

Reception driven by polling: the **UartRxFull** bit is to be read and checked. When it is 1, the **RegUartRx** register contains new data and has to be read before a new word is received.
Transmission driven by polling: the **UartTxFull** bit is to read and checked. When it is 0, the **RegUartTx** register is empty and a new word can be written to it.

15.9 Software hints

Example of program for a transmission with polling:

1. The **RegUartCmd** register and the **RegUartCtrl** register are initialized (for example: 8 bit word length, odd parity, 9600 baud, enable UART transmission).
2. Write a byte to **RegUartTx**.
3. Wait untill the **UartTxFull** bit in **RegUartTxSta** register equals 0.
4. Jump to 2 to writing the next byte if the message is not finished.
5. End of transmission.

Example of program for a transmission with interrupt:

1. The **RegUartCmd** register and the **RegUartCtrl** register are initialized (for example: 8 bit word length, odd parity, 9600 baud, enable UART transmission).
2. Write a byte to **RegUartTx**.
3. After an interrupt and if the message is not finished, jump to 2
4. End of transmission.

Example of program for a reception with polling:

1. The **RegUartCmd** register and the **RegUartCtrl** register are initialized (for example: 8 bit word length, odd parity, 9600 baud, enable UART reception).
2. Wait until the **UartRxFull** bit in the **RegUartRxSta** register equals 1.
3. Read the **RegUartRxSta** and check if there is no error.
4. Read data in **RegUartRx**.
5. If data is not equal to End-Of-Line, then jump to 2.
6. End of reception.

Example of program for a reception with interrupt:

1. The **RegUartCmd** register and the **RegUartCtrl** register are initialized (for example: 8 bit word length, odd parity, 9600 baud, enable UART reception).
2. When there is an interrupt, jump to 3
3. Read **RegUartRxSta** and check if there is no error.
4. Read data in **RegUartRx**.
5. If data is not equal to End-Of-Line, then jump to 2.
6. End of reception.

16. Universal Synchronous Receiver/Transmitter (USRT)

16.1 Features

The USRT implements a hardware support for software implemented serial protocols:

- Control of two external lines S0 and S1 (read/write).
- Conditional edge detection generates interrupts.
- S0 rising edge detection.
- S1 value is stored on S0 rising edge.
- S0 signal can be forced to 0 after a falling edge on S0 for clock stretching in the low state.
- S0 signal can be stretched in the low state after a falling edge on S0 and after a S1 conditional detection.

16.2 Overview

The USRT block supports software universal synchronous receiver and transmitter mode interfaces.

External lines S0 and S1 respectively correspond to clock line and data line. S0 is mapped to PB[4] and S1 to PB[5] when the USRT block is enabled. It is independent of **RegPBdir** (Port B can be input or output). When USRT is enabled, the configurations in port B for PB[4] and PB[5] are overwritten by the USRT configuration. Internal pull-ups can be used by setting the **PBPullup[5:4]** bits.

Conditional edge detections are provided.

RegUsrtS1 can be used to read the S1 data line from PB[5] in receive mode or to drive the output S1 line PB[5] by writing it when in transmit mode. It is advised to read S1 data when in receive mode from the **RegUsrtBufferS1** register, which is the S1 value sampled on a rising edge of S0.

16.3 Register map

Register name
RegUsrtS1
RegUsrtS0
RegUsrtCtrl
RegUsrtCond1
RegUsrtCond2
RegUsrtBufferS1
RegUsrtEdgeS0

Table 16-1: USRT Registers

Block configuration registers:

pos.	RegUsrtS1	rw	reset	function
7-1	“0000000”	r	-	Unused
0	UsrtS1	r/w	1 nresetglobal	Write: data S1 written to pad PB[5]), Read: value on PB[5] (not **UsrtS1** value).

Table 16-2: RegUsrtS1

pos.	RegUsrtS0	rw	reset	function
7-1	“0000000”	r	-	Unused
0	UsrtS0	r/w	1 nresetglobal	Write: clock S0 written to pad PB[4], Read: value on PB[4] (not **UsrtS0** value).

Table 16-3: RegUsrtS0

The values that are read in the registers **RegUsrtS1** and **RegUsrtS0** are not necessarily the same as the values that were written in the register. The read value is read back on the circuit pins not in the registers themselves. Since the outputs are open drain, an external circuit on the circuit pins may force a value different from the register value.

pos.	RegUsrtCtrl	rw	reset	function
7-4	“0000”	r	-	Unused
3	UsrtWaitS0	r	0 nresetglobal	Clock stretching flag (0=no stretching), cleared by writing **RegUsrtBufferS1**
2	UsrtEnWaitCond1	r/w	0 nresetglobal	Enable stretching on UsrtCond1 detection (0=disable)
1	UsrtEnWaitS0	r/w	0 nresetglobal	Enable stretching operation (0=disable)
0	UsrtEnable	r/w	0 nresetglobal	Enable USRT operation (0=disable)

Table 16-4: RegUsrtCtrl

pos.	RegUsrtCond1	rw	reset	function
7-1	“0000000”	r	-	Unused
0	UsrtCond1	r/c	0 nresetglobal	State of condition 1 detection (1 =detected), cleared when written.

Table 16-5: RegUsrtCond1

pos.	RegUsrtCond2	rw	reset	function
7-1	“0000000”	r	-	Unused
0	UsrtCond2	r/c	0 nresetglobal	State of condition 2 detection (1 =detected), cleared when written.

Table 16-6: RegUsrtCond2

pos.	RegUsrtBufferS1	rw	reset	function
7-1	“0000000”	r	-	Unused
0	UsrtBufferS1	r	0 nresetglobal	Value on S1 at last S0 rising edge.

Table 16-7: RegUsrtBufferS1

pos.	RegUsrtEdgeS0	rw	reset	function
7-1	“0000000”	r	-	Unused
0	UsrtEdgeS0	r	0 nresetglobal	State of rising edge detection on S0 (1=detected). Cleared by reading **RegUsrtBufferS1**

Table 16-8: RegUsrtEdgeS0

16.4 Interrupts map

interrupt source	default mapping in the interrupt manager
Irq_cond2	RegIrqMid(7)
Irq_cond1	RegIrqMid(6)

Table 16-9: Interrupts map

16.5 Conditional edge detection 1

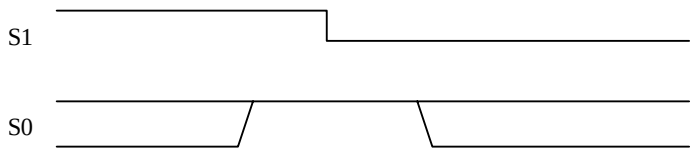


Figure 16-1: Condition 1

Condition 1 is satisfied when S0=1 at the falling edge of S1. The bit **UsrtCond1** in **RegUsrtCond1** is set when the condition 1 is detected and the USRT interface is enabled (**UsrtEnable**=1). Condition 1 is asserted for both modes (receiver and transmitter). The **UsrtCond1** bit is read only and is cleared by all reset conditions and by writing any data to its address.

Condition 1 occurrence also generates an interrupt on Irq_cond1.

16.6 Conditional edge detection 2

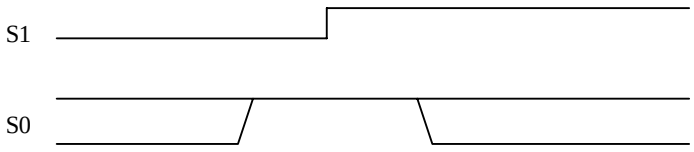


Figure 16-2: Condition 2

Condition 2 is satisfied when S0=1 at the rising edge of S1. The bit **UsrtCond2** in **RegUsrtCond2** is set when the condition 2 is detected and the USRT interface is enabled. Condition 2 is asserted for both modes (receiver and transmitter). The **UsrtCond2** bit is read only and is cleared by all reset conditions and by writing any data to its address.

Condition 2 occurrence also generates an interrupt on Irq_cond2.

16.7 Interrupts or polling

In receive mode, there are two possibilities to detect condition 1 or 2: the detection of the condition can generate an interrupt or the registers can be polled (reading and checking the **RegUsrtCond1** and **RegUsrtCond2** registers for the status of USRT communication).

16.8 Function description

The bit **UsrtEnable** in **RegUsrtCtrl** is used to enable the USRT interface and controls the PB[4] and PB[5] pins. This bit puts these two port B lines in the open drain configuration requested to use the USRT interface.

If no external pull-ups are added on PB[4] and PB[5], the user can activate internal pull-ups by setting **PBPullup[4]** and **PBPullup[5]** in **RegPBPullup**.

The bits **UsrtEnWaitS0**, **UsrtEnWaitCond1**, **UsrtWaitS0** in **RegUsrtCtrl** are used for transmitter/receiver control of USRT interface.

Figure 16-3 shows the unconditional clock stretching function which is enabled by setting **UsrtEnWaitS0.**

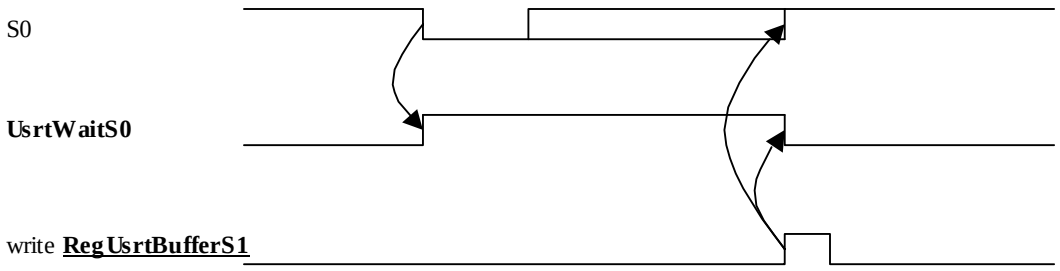


Figure 16-3: S0 Stretching (UsrtEnWaitS0=1)

When **UsrtEnWaitS0** is 1, the S0 line will be maintained at 0 after its falling edge (clock stretching). **UsrtWaitS0** is then set to 1, indicating that the S0 line is forced low. One can release S0 by writing to the **RegUsrtBufferS1** register.

The same can be done in combination with condition 1 detection by setting the **UsrtEnWaitCond1** bit. Figure 16-4 shows the conditional clock stretching function, which is enabled by setting **UsrtEnWaitCond1**.

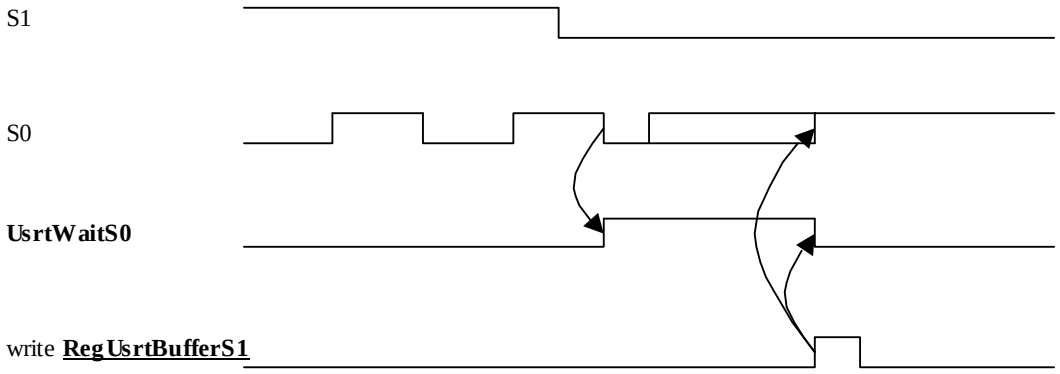


Figure 16-4: Conditional stretching (UsrtEnWaitCond1=1)

When **UsrtEnWaitCond1** is 1, the S0 signal will be stretched in its low state after its falling edge if the condition 1 has been detected before (**UsrtCond1**=1). **UsrtWaitS0** is then set to 1, indicating that the S0 line is forced low. One can release S0 by writing to the **RegUsrtBufferS1** register.

Figure 16-5 shows the sampling function implemented by the **UsrtBufferS1** bit. The bit **UsrtBufferS1** in **RegUsrtBufferS1** is the value of S1 sampled on PB[4] at the last rising edge of S0. The bit **UsrtEdgeS0** in **RegUsrtEdgeS0** is set to one on the same S0 rising edge and is cleared by a read operation of the **RegUsrtBufferS1** register. The bit therefor indicates that a new value is present in the **RegUsrtBufferS1** which was not yet read.

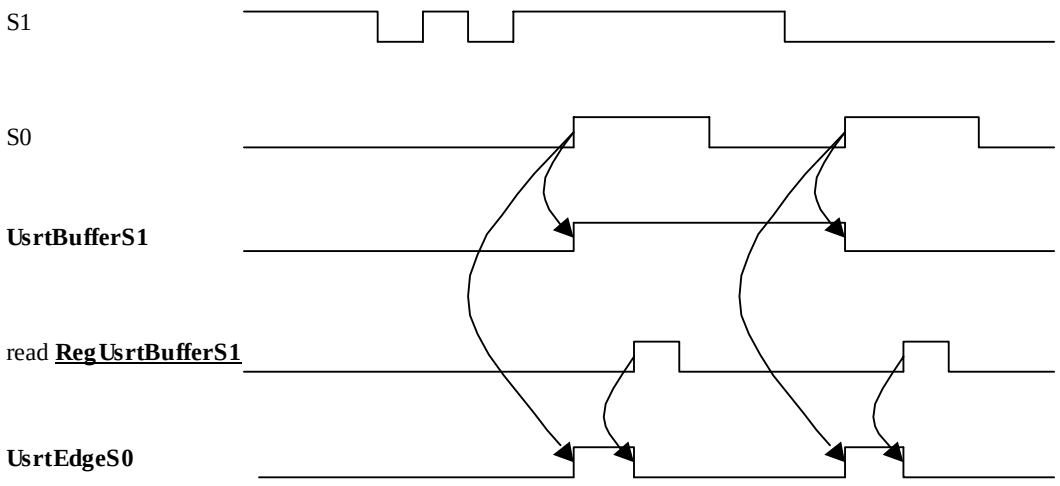


Figure 16-5: S1 sampling

17. Counters/PWM

17.1 Features

- 4 x 8-bits timer/counter modules or 2 x 16-bits timers/counter modules
- Each with 4 possible clock sources
- Up/down counter modes
- Interrupt and event generation
- Capture function (internal or external source)
- Rising, falling or both edge of capture signal (except for xtal 32 kHz, only rising edge)
- PA[3:0] can be used as clock inputs (debounced or direct, frequency divided by 2 or not)
- 2 x 8 bits PWM or 2 x 16 bits PWM
- PWM resolution of 8, 10, 12, 14 or 16 bits
- Complex mode combinations are possible

17.2 Overview

Counter A and Counter B are 8-bits counters and can be combined to form a 16-bit counter. Counter C and Counter D exhibit the same feature.

The counters can also be used to generate two PWM outputs on PB[0] and PB[1]. In PWM mode one can generate PWM functions with 8, 10, 12, 14 or 16 bits wide counters.

The counters A and B can be captured by events on an internal or an external signal. The capture can be performed on both 8-bit counters running individually on two different clock sources or on both counters chained to form a 16-bit counter. In any case, the same capture signal is used for both counters.

When the counters A and B are not chained, they can be used in several configurations: A and B as counters, A and B as captured counters, A as PWM and B as counter, A as PWM and B as captured counter.

When the counters C and D are not chained, they can be used either both as counters or counter C as PWM and counter D as counter.

17.3 Register map

register name
RegCntA
RegCntB
RegCntC
RegCntD
RegCntCtrlCk
RegCntConfig1
RegCntConfig2
RegCntOn

Table 17-1. Counter Registers

bit	RegCntA	rw	reset	function
7-0	CounterA	R	00000000 nresetglobal	8-bits counter value
7-0	CounterA	W	00000000 nresetglobal	8-bits comparison value

Table 17-2. **RegCntA**

bit	RegCntB	rw	reset	function
7-0	CounterB	R	00000000 nresetglobal	8-bits counter value
7-0	CounterB	W	00000000 nresetglobal	8-bits comparison value

Table 17-3. **RegCntB**

Note: When writing to **RegCntA** or **RegCntB**, the processor writes the counter comparison values. When reading these locations, the processor reads back either the actual counter value or the last captured value if the capture mode is active.

bit	RegCntC	rw	reset	function
7-0	CounterC	R	00000000 nresetglobal	8-bits counter value
7-0	CounterC	W	00000000 nresetglobal	8-bits comparison value

Table 17-4. **RegCntC**

bit	RegCntD	rw	reset	function
7-0	CounterD	R	00000000 nresetglobal	8-bits counter value
7-0	CounterD	W	00000000 nresetglobal	8-bits comparison value

Table 17-5. **RegCntD**

Note: When writing **RegCntC** or **RegCntD**, the processor writes the counter comparison values. When reading these locations, the processor reads back the actual counter value.

bit	RegCntCtrlCk	rw	reset	function
7-6	CntDCkSel(1:0)	rw	00 nresetglobal	Counter d clock selection
5-4	CntCCkSel(1:0)	rw	00 nresetglobal	Counter c clock selection
3-2	CntBCkSel(1:0)	rw	00 nresetglobal	Counter b clock selection
1-0	CntACkSel(1:0)	rw	00 nresetglobal	Counter a clock selection

Table 17-6. **RegCntCtrlCk**

bit	RegCntConfig1	rw	reset	function
7	CntDDownUp	rw	0 nresetglobal	Counter d up or down counting (0=down)
6	CntCDownUp	rw	0 nresetglobal	Counter c up or down counting (0=down)
5	CntBDownUp	rw	0 nresetglobal	Counter b up or down counting (0=down)
4	CntADownUp	rw	0 nresetglobal	Counter a up or down counting (0=down)
3	CascadeCD	rw	0 nresetglobal	Cascade counter c & d (1=cascade)
2	CascadeAB	rw	0 nresetglobal	Cascade counter a & b (1=cascade)
1	CntPWM1	rw	0 nresetglobal	Activate pwm1 on counter c or c+d (PB(1))
0	CntPWM0	rw	0 nresetglobal	Activate pwm0 on counter a or a+b (PB(0))

Table 17-7. **RegCntConfig1**

bit	RegCntConfig2	rw	reset	function
7-6	CapSel(1:0)	rw	00 nresetglobal	Capture source selection
5-4	CapFunc(1:0)	rw	00 nresetglobal	Capture function
3-2	Pwm1Size(1:0)	rw	00 nresetglobal	Pwm1 size selection
1-0	Pwm0Size(1:0)	rw	00 nresetglobal	Pwm0 size selection

Table 17-8. **RegCntConfig2**

bit	RegCntOn	rw	reset	function
7	CntDExtDiv	rw	0 nresetglobal	Divide PA(3) frequency by 2 (1=divide)
6	CntCExtDiv	rw	0 nresetglobal	Divide PA(2) frequency by 2 (1=divide)
5	CntBExtDiv	rw	0 nresetglobal	Divide PA(1) frequency by 2 (1=divide)
4	CntAExtDiv	rw	0 nresetglobal	Divide PA(0) frequency by 2 (1=divide)
3	CntDEnable	rw	0 nresetglobal	Enable counter d
2	CntCEnable	rw	0 nresetglobal	Enable counter c
1	CntBEnable	rw	0 nresetglobal	Enable counter b
0	CntAEnable	rw	0 nresetglobal	Enable counter a

Table 17-9. **RegCntOn**

17.4 Interrupts and events map

Interrupt source	Default mapping in the interrupt manager	Default mapping in the event manager
IrqA	RegIrqHigh(4)	RegEvn(7)
IrqB	RegIrqLow(5)	RegEvn(3)
IrqC	RegIrqHigh(3)	RegEvn(6)
IrqD	RegIrqLow(4)	RegEvn(2)

Table 17-10. Default interrupt and event mapping.

17.5 Block schematic

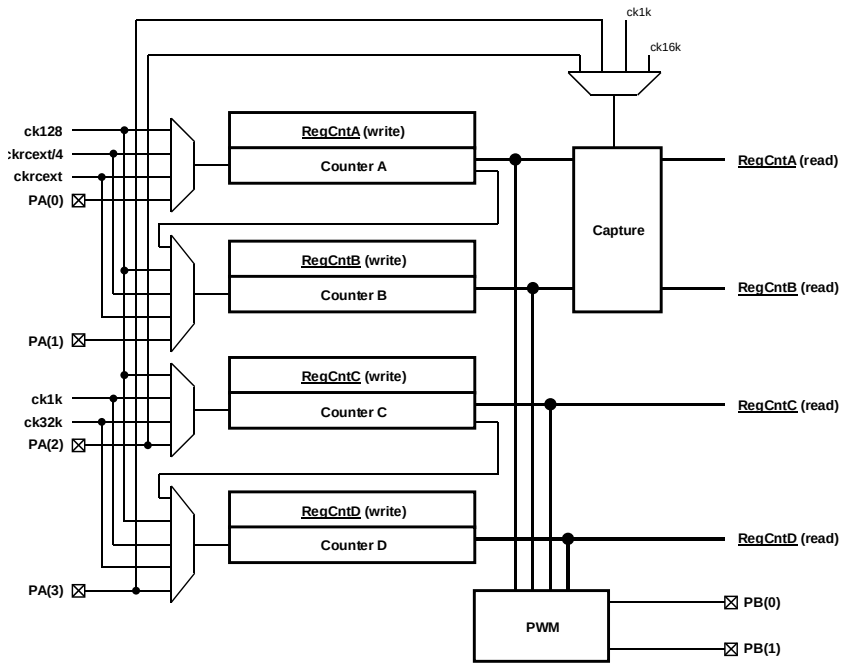


Figure 17-1: Counters/timers block schematic

17.6 General counter registers operation

Counters are enabled by **CntAEnable**, **CntBEnable**, **CntCEnable**, and **CntDEnable** in **RegCntOn**.

To stop the counter X, **Cnt**X**Enable** must be reset. To start the counter X, **Cnt**X**Enable** must be set. When counters are cascaded, **CntAEnable** and **CntCEnable** also control respectively the counters B and D.

All counters have a corresponding 8-bit read/write register: **RegCntA**, **RegCntB**, **RegCntC**, and **RegCntD**. When read, these registers contain the counter value (or the captured counter value). When written, they modify the counter comparison values.

It is possible to read any counter at any time, even when the counter is running. The value is guaranteed to be correct when the counter is running on an internal clock source. For a correct acquisition of the counter value when running on an external clock source, use one of the three following methods:

1) For slow operating counters (typically at least 8 times slower than the CPU clock), oversample the counter content and perform a majority operation on the consecutive read results to select the correct actual content of the counter.

2) Stop the concerned counter, perform the read operation and restart the counter. While stopped, the counter content is frozen and the counter does not take into account the clock edges delivered on the external pin.

3) Use the capture mechanism.

When a value is written into the counter register while the counter is in counter mode, both the comparison value is updated and the counter value is modified. In upcount mode, the register value is reset to zero. In downcount mode, the comparison value is loaded into the counter. Due to the synchronization mechanism between the processor clock domain and the external clock source domain, this modification of the counter value can be postponed until the counter is enabled and that it receives it's first valid clock edge.

In the PWM mode or in the capture mode, the counter value is not modified by the write operation in the counter register. Changing to the counter mode, does not update the counter value (no reset in upcount, no load in downcount mode).

17.7 Clock selection

The clock source for each counter can be individually selected by writing the appropriate value in the register **RegCntCtrlCk**.

Table 17-11 gives the correspondence between the binary codes used for the configuration bits **CntACkSel(1:0)**, **CntBCkSel(1:0), CntCCkSel(1:0)** or **CntDCkSel(1:0)** and the clock source selected respectively for the counters A, B, C or D.

CntX**CkSel(1:0)**	**Clock source for**			
	CounterA	**CounterB**	**CounterC**	**CounterD**
11	Ck128			
10	CkRcExt/4		Ck1k	
01	CkRcExt		Ck32k	
00	PA(0)	PA(1)	PA(2)	PA(3)

Table 17-11: Clock sources for counters A, B, C and D

The CkRcExt clock is the RC oscillator or external clock. The clocks below 32kHz can be derived from the RC oscillator, the external clock source or the crystal oscillator (see the documentation of the clock block). A separate external clock source can be delivered on PortA for each individula counter.

The external clock sources can be debounced or not by properly setting the PortA configuration registers. Additionally, the external clock sources can be divided by two in the counter block, thus enabling higher external clock frequencies, by setting the **Cnt**X**ExtDiv** bits in the **RegCntOn** register.

Switching between an internal and an external clock source can only be performed while the counter is stopped. The enabling or disabling of the external clock frequency division can only be performed while the counter using this clock is stopped, or when this counter is running on an internal clock source.

17.8 Counter mode selection

Each counter can work in one of the following modes:

1) Counter, downcount & upcount
2) Captured counter, downcount & upcount (only counters A&B)
3) PWM, downcount & upcount
4) Captured PWM, downcount and upcount

The counters A and B or C and D can be cascaded or not. In cascaded mode, A and C are the LSB counters while B and D are the MSB counters.

Table 17-12 shows the different operation modes of the counters A and B as a function of the mode control bits. For all counter modes, the source of the down or upcount selection is given (either the bit **CntADownUp** or the bit **CntBDownUp**). Also, the mapping of the interrupt sources IrqA and IrqB and the PWM output on PB(0) in these different modes is shown.

CascadeAB	**CountPWM0**	**CapFunc(1:0)**	**Counter A mode**	**Counter B mode**	**IrqA source**	**IrqB source**	**PB(0) function**
0	0	00	Counter 8b Downup: A	Counter 8b Downup: B	Counter A	Counter B	PB(0)
1	0	00	Counter 16b AB Downup: A		Counter AB	-	PB(0)
0	1	00	PWM 8b Downup: A	Counter 8b Downup: B	-	Counter B	PWM A
1	1	00	PWM 10 – 16b AB Downup A		-	-	PWM AB
0	0	1x or x1	Captured counter 8b Downup: A	Captured counter 8b Downup: B	Capture A	Capture B	PB(0)
1	0	1x or x1	Captured counter 16b AB Downup: A		Capture AB	Capture AB	PB(0)
0	1	1x or x1	Captured PWM 8b Downup: A	Captured counter 8b Downup: B	Capture A	Capture B	PWM A
1	1	1x or x1	Captured 10 – 16b PWM (captured value on 16b) Downup: A		Capture AB	Capture AB	PWM AB

Table 17-12: Operating modes of the counters A and B

Table 17-13 shows the different operation modes of the counters C and D as a function of the mode control bits. For all counter modes, the source of the down or upcount selection is given (either the bit **CntCDownUp** or the bit **CntDDownUp**). The mapping of the interrupt sources IrqC and IrqD and the PWM output on PB(1) in these different modes is also shown.

The switching between different modes must be done while the concerned counters are stopped. While switching capture mode on and off, unwanted interrupts can appear on the interrupt channels concerned by this mode change.

CascadeCD	CountPWM1	Counter C mode	Counter D mode	IrqC source	IrqD source	PB(1) function
0	0	Counter 8b Downup: C	Counter 8b Downup: D	Counter C	Counter D	PB(1)
1	0	Counter 16b CD Downup: C		Counter CD	-	PB(1)
0	1	PWM 8b Downup: C	Counter 8b Downup: D	-	Counter D	PWM C
1	1	PWM 10 – 16b CD Downup: C		-	-	PWM CD

Table 17-13: Operating modes of the counters C and D

17.9 Counter / Timer mode

The counters in counter / timer mode are generally used to generate interrupts after a predefined number of clock periods applied on the counter clock input.

Each counter can be set individually either in upcount mode by setting **Cnt**X**DownUp** in the register **RegCntConfig1** or in downcount mode by resetting this bit. Counters A and B can be cascaded to behave as a 16 bit counter by setting **CascadeAB** in the **RegCntConfig1** register. Counters C and D can be cascaded by setting **CascadeCD**. When cascaded, the up/down count modes of the counters B and D are defined respectively by the up/down count modes set for the counters A and C.

When in upcount mode, the counter will start incrementing from zero up to the target value which has been written in the corresponding **RegCnt**X register(s). When the counter content is equal to the target value, an interrupt is generated at the next counter clock pulse and the counter is loaded again with the zero value (Figure 17-2).

When in downcount mode, the counter will start decrementing from the initial load value which has been written in the corresponding **RegCnt**X register(s) down to the zero value. Once the counter content is equal to zero, an interrupt is generated at the next counter clock pulse and the counter is loaded again with the load value (Figure 17-2).

Be careful to select the counter mode (no capture, not PWM, specify cascaded or not and up or down counting mode) before writing any target or load value to the **RegCnt**X register(s). This ensures that the counter will start from the correct initial value. When counters are cascaded, both counter registers must be written to ensure that both cascaded counters will start from the correct initial values.

The stopping and consecutive starting of a counter in counter mode without a target or load value write operation in between can generate an interrupt if this counter has been stopped at the zero value (downcount) or at it's target value (upcount). This interrupt is additional to the interrupt which has already been generated when the counter reached the zero or the target value.

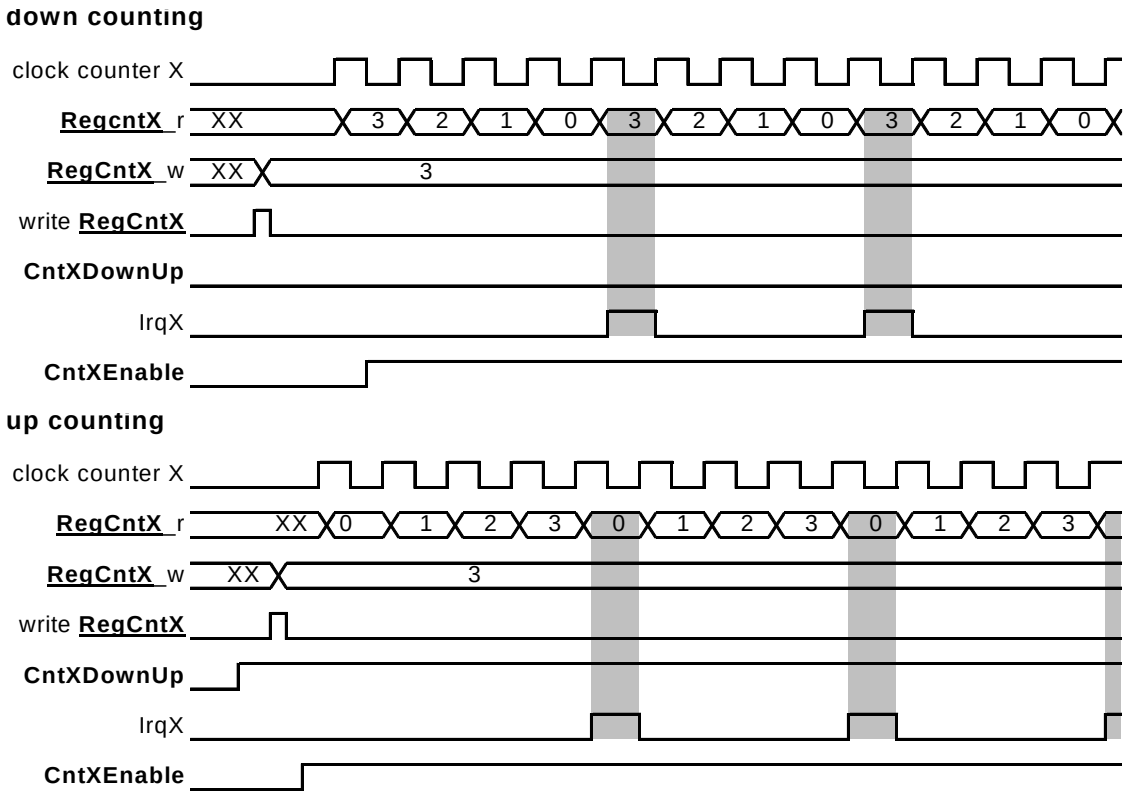


Figure 17-2. Up and down count interrupt generation.

17.10 PWM mode

The counters can generate PWM signals (Pulse Width Modulation) on the PortB outputs PB(0) and PB(1).

The PWM mode is selected by setting **CntPWM1** and **CntPWM0** in the **RegCntConfig1** register. See Table 17-12 and Table 17-13 for an exact description of how the setting of **CntPWM1** and **CntPWM0** affects the operating mode of the counters A, B, C and D according to the other configuration settings.

When **CntPWM0** is enabled, the PWMA or PWMAB output value overrides the value set in bit 0 of **RegPBOut** in the Port B peripheral. When **CntPWM1** is enabled, the PWMC or PWMCD output value overrides the value set in bit 1 of **RegPBOut**. The corresponding ports (0 and/or 1) of Port B must be set in digital mode and as output and either open drain or not and pull up or not through a proper setting of the control registers of the Port B.

Counters in PWM mode count down or up, according to the **Cnt**X**DownUp bit** setting. No interrupts and events are generated by the counters that are in PWM mode. Counters do count circularly: they restart at zero or at the maximal value (either 0xFF when not cascaded or 0xFFFF when cascaded) when respectively an overflow or an underflow condition occurs in the counting.

The internal PWM signals are low as long as the counter contents are higher than the PWM code values written in the **RegCnt**X registers. They are high when the counter contents are smaller or equal to these PWM code values. In order to have glitch free outputs, the PWM outputs on PB(0) and PB(1) are sampled versions of these internal PWM signals, therefore delayed by one counter clock cycle.

The PWM resolution is always 8 bits when the counters used for the PWM signal generation are not cascaded. **PWM0Size(1:0)** and **PWM1Size(1:0)** in the **RegCntConfig2** register are used to set the PWM resolution for the counters A and B or C and D respectively when they are in cascaded mode. The different possible resolutions in cascaded mode are shown in Table 17-14. Choosing a 16 bit PWM code which is higher than the maximum value that can be represented by the number of bits chosen for the resolution results in a PWM output which is always tied to 1.

PwmXsize(1:0)	Resolution
11	16 bits
10	14 bits
01	12 bits
00	10 bits

Table 17-14: Resolution selection in cascaded PWM mode

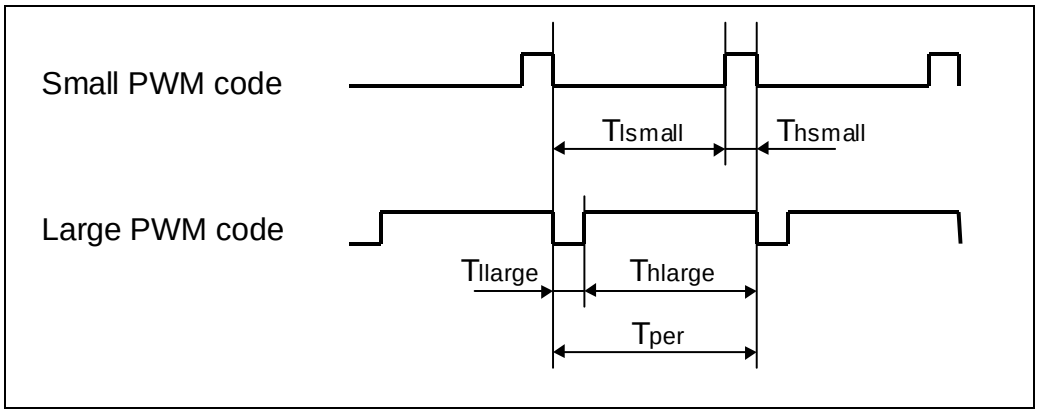


Figure 17-3: PWM modulation examples

The period of the PWM signal is given by the formula:

$$Tper = \frac{2^{resolution}}{f_{ckcnt}}$$

The duty cycle ratio DCR of the PWM signal is defined as:

$$DCR = \frac{Th}{Tper}$$

DCR can be selected between $\frac{100}{2^{resolution}}$ % and 100 %.

DCR in % in function of the RegCntX content(s) is given by the relation:

$$DCR = MIN\left(\frac{100(1+\mathrm{RegCntX})}{2^{resolution}}, 100\right)$$

17.11 Capture function

The 16-bit capture register is provided to facilitate frequency measurements. It provides a safe reading mechanism for the counters A and B when they are running. When the capture function is active, the processor does not read anymore the counters A and B directly, but instead reads shadow registers located in the capture block. An interrupt is generated after a capture condition has been met when the shadow register content is updated. The capture condition is user defined by selecting either internal capture signal sources derived from the prescaler or from the external PA(2) or PA(3) ports. Both counters use the same capture condition.

When the capture function is active, the A and B counters can either upcount or downcount. They do not count circularly: they restart at zero or at the maximal value (either 0xFF when not cascaded or 0xFFFF when cascaded) when respectively an overflow or an underflow condition occurs in the counting. The capture function is also active on the counters when used to generate PWM signals.

CapFunc(1:0) in register **RegCntConfig2** determines if the capture function is enabled or not and selects which edges of the capture signal source are valid for the capture operation. The source of the capture signal can be selected by setting **CapSel(1:0)** in the **RegCntConfig2** register. For all sources, rising, falling or both edge sensitivity can be selected. Table 17-15 shows the capture condition as a function of the setting of these configuration bits.

CapSel(1:0)	Selected capture signal	CapFunc	Selected condition	Capture condition
11	1 K	00	**Capture disabled**	-
		01	Rising edge	1 K rising edge
		10	Falling edge	1 K falling edge
		11	Both edges	2 K
10	16 K	00	**Capture disabled**	-
		01	Rising edge	16 K rising edge
		10	Falling edge	16 K falling edge
		11	Both edges	32 K
01	PA3	00	**Capture disabled**	-
		01	Rising edge	PA3 rising edge
		10	Falling edge	PA3 falling edge
		11	Both edges	PA3 both edges
00	PA2	00	**Capture disabled**	-
		01	Rising edge	PA2 rising edge
		10	Falling edge	PA2 falling edge
		11	Both edges	PA2 both edges

Table 17-15: Capture condition selection

CapFunc(1:0) and **CapSel(1:0)** can be modified only when the counters are stopped otherwise data may be corrupted during one counter clock cycle.

Due to the synchronization mechanism of the shadow registers and depending on the frequency ratio between the capture and counter clocks, the interrupts may be generated one or only two counter clock pulses after the effective capture condition occurred. When the counters A and B are not cascaded and do not operate on the same clock, the interruptions on IrqA and IrqB which inform that the capture condition was met, may appear at different moments. In this case, the processor should read the shadow register associated to a counter only if the interruption related to this counter has been detected.

An edge is detected on the capture signals only if the minimal pulse widths of these signals in the low and high states are higher than a period of the counter clock source.

17.12 Specifications

Parameter	Min	Typ	Max	Unit	Conditions
Pulse width in the low and high states for an external clock source, frequency division by 2 disabled	500			ns	@ 1.2V
	125			ns	@ 2.4V
Pulse width in the low and high states for an external clock source, frequency division by 2 enabled	100			ns	@ 1.2V
	25			ns	@ 2.4V
Pulse width of external capture signals	$\frac{1}{fckcnt}$			s	

Table 17-16: Timing specifications for the counters

18 The Voltage Level Detector

18.1 Features

- can be switched off, on or simultaneously with CPU activities
- generates an interrupt if power supply is below a pre-determined level

18.2 Overview

The Voltage Level Detector monitors the state of the system battery. It returns a logical high value (an interrupt) in the status register if the supplied voltage drops below the user defined level (Vsb).

18.3 Register map

There are two registers in the VLD, namely **RegVldCtrl** and **RegVldStat**. Table 18-2 shows the mapping of control bits and functionality of **RegVldCtrl** while Table 18-3 describes that for **RegVldStat**.

register name
RegVldCtrl
RegVldStat

Table 18-1: Vld registers

pos.	RegVldCtrl	rw	reset	function
7-4	--	r	0000	reserved
3	VldRange	r w	0 nresetglobal	VLD detection voltage range for VldTune = “011”: 0 : 1.3V 1 : 2.55V
2-0	VldTune[2:0]	r w	000 nresetglobal	VLD tuning: 000 : +19 % 111 : -18 %

Table 18-2: RegVldCtrl

pos.	RegVldStat	rw	reset	function
7-3	--	r	00000	reserved
2	VldResult	r	0 nresetglobal	is 1 when battery voltage is below the detection voltage
1	VldValid	r	0 nresetglobal	Indicates when VldResult can be read
0	VldEn	r w	0 nresetglobal	VLD enable

Table 18-3: RegVldStat

18.4 Interrupt map

interrupt source	default mapping in the interrupt manager
IrqVld	RegIrqMid(2)

Table 18-4: Interrupt map

18.5 VLD operation

The VLD is controlled by **VldRange**, **VldTune** and **VldEn**. **VldRange** selects the voltage range to be detected, while **VldTune** is used to fine-tune this voltage level in 8 steps. **VldEn** is used to enable (disable) the VLD with a 1(0) value respectively. Disabled, the block will dissipate no power.

symbol	description	min	typ	max	unit	comments	
Vth	Threshold voltage	Note 1			V	trimming values:	
						VldRange	VldTune
			1.53			0	000
			1.44			0	001
			1.36			0	010
			1.29			0	011
			1.22			0	100
			1.16			0	101
			1.11			0	110
			1.06			0	111
			3.06			1	000
			2.88			1	001
			2.72			1	010
			2.57			1	011
			2.44			1	100
			2.33			1	101
			2.22			1	110
			2.13			1	111
T_{EOM}	duration of measurement		2.0	2.5	ms	Note 2	
T_{PW}	Minimum pulse width detected		875	1350	us	Note 2	

Table 18-5: Voltage level detector operation

Note 1: absolute precision of the threshold voltage is ±10%.
Note 2: this timing is respected in case the internal RC or crystal oscillators are selected. Refer to the clock block documentation in case the external clock is used.

To start the voltage level detection, the user sets bit **VldEn**. The measurement is started.After 2ms, the bit **VldValid** is set to indicate that the measurement results are valid. From that time on, as long as the VLD is enabled, a maskable interrupt request is sent if the voltage level falls below the threshold. One can also poll the VLD and monitor the actual measurement result by reading the **VldResult** bit of the **RegVldStat**. This result is only valid as long as the **VldValid** bit is '1'.

Figure 18-1 shows the timing of the VLD. An interrupt is generated on each rising edge of **VldResult**.

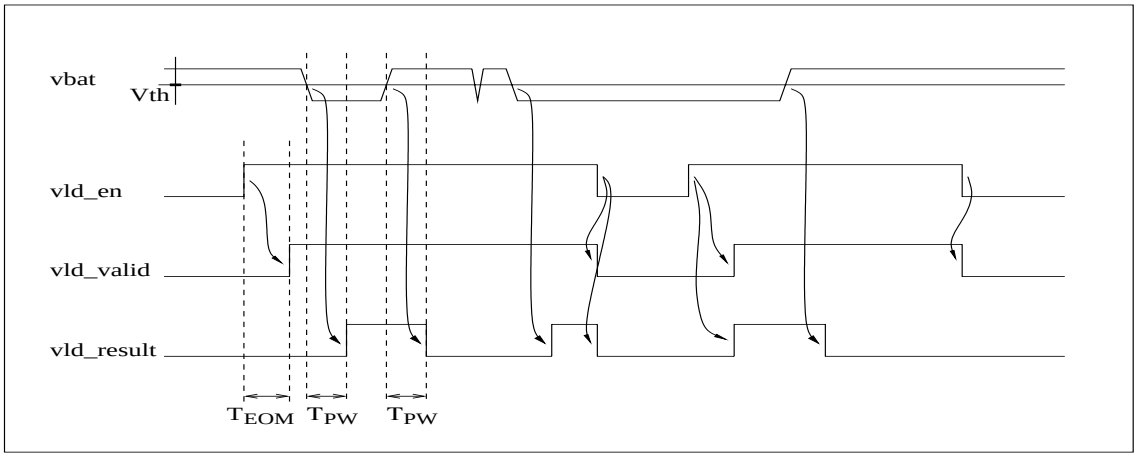


Figure 18-1: VLD timing

The threshold value should not be changed during the measurement.

19. Low Power Comparators

19.1 Features

The cmpd peripheral implements four low power comparators.

- Quiescent current consumption of 1.5µA
- Very low switching current
- Per channel configurable interrupt
- Hysteresis
- 1 MHz operation

19.2 Overview

Figure 19-1 gives an overview of this block:

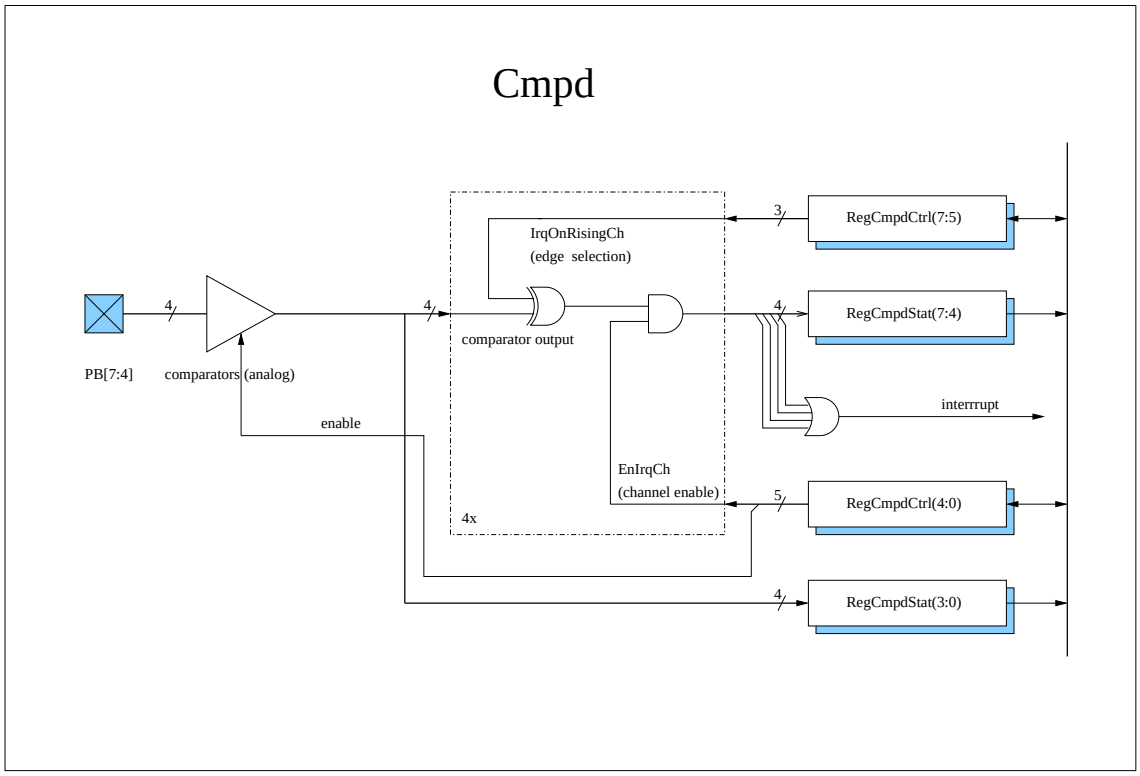


Figure 19-1: Structure of Cmpd

The cmpd peripheral is a 4-channel low power comparator. It is intended to compare analog input signals with an internally set threshold voltage. The comparator maintains low current consumption even if the input signal is very close to the threshold. The comparison result of each channel can be used to generate an interrupt and/or is available for polling.

The comparator can be enabled or disabled by programming the **Enable** bit in the **RegCmpdCtrl** register. When disabled, the block consumes no current.

The peripheral has a single interrupt output which is a combination of the four channels. The combination can be chosen by programming the **RegCmpdCtrl** register. The **EnIrqCh[3:0]** bits select the channel that can activate the interrupt. The **IrqOnRisingCh[2:0]** bits indicate if the interrupt is generated on detection of the rising or falling edge of the channel.

The comparison results of the peripheral can be read in the **RegCmpdStat** register. The bits **CmpdOut[3:0]** are the value of the comparisons at the moment the register is read. The **CmpdStat[3:0]** indicates which channel generated an interrupt since the register was last read.

Comparator specifications:

Sym	description	min	typ	max	unit	comments
t_{pulse}	Required input pulse width	500			ns	VBAT≥1.2V
IDD_q	Quiescent current		0.8	1.5	µA	1
IDD_{stat}	Maximal static current		1.5		µA	2
V_{th}	Threshold voltage	0.7		1.1	V	3
$\Delta V_{th}/\Delta T$	Threshold temperature drift		-0.9		mV/°C	
V_{hyst}	Threshold hysteresis		13		mV	

Table 19-1: Comparator specifications

Comments:
1. The quiescent current is defined for a static input voltage <0.5V or >1.3V. The specified consumption is the sum for all 4 channels.
2. The maximal static current is defined for any static input voltage between VDD and VSS. The specified consumption is the sum for all 4 channels.
3. Defined with respect to VSS.

How to start the cmpd:
To avoid unwanted irqs one has first to configure the rising / falling edge of the detection (bit IrqOnRisingCh[2:0]) and to enable the comparator (bit Enable). Only after that may the user enable the channel interrupts with bit EnIrqCh[3:0].

19.3 Register map

There are two registers in the Cmpd, namely **RegCmpdStat** and **RegCmpdCtrl**. Table 19-3 and Table 19-4 show the mapping of the control bits and the functionality of these registers.

register name
RegCmpdStat
RegCmpdCtrl

Table 19-2: Cmpd registers

pos.	RegCmpdStat	rw	reset	function
7	CmpdStat[3]	rc	0 nresetglobal	1: if the channel 3 generated an interrupt since last read of this register
6	CmpdStat[2]	rc	0 nresetglobal	1: if the channel 2 generated an interrupt since last read of this register
5	CmpdStat[1]	rc	0 nresetglobal	1: if the channel 1 generated an interrupt since last read of this register
4	CmpdStat[0]	rc	0 nresetglobal	1: if the channel 0 generated an interrupt since last read of this register
3	CmpdOut[3]	r	0 nresetglobal	Channel 3 comparator output
2	CmpdOut[2]	r	0 nresetglobal	Channel 2 comparator output
1	CmpdOut[1]	r	0 nresetglobal	Channel 1 comparator output
0	CmpdOut[0]	r	0 nresetglobal	Channel 0 comparator output

Table 19-3: RegCmpdStat

pos.	RegCmpdCtrl	rw	reset	function
7	IrqOnRisingCh[2]	rw	0 nresetglobal	1: an interrupt is generated on the rising edge of channels 2 and 3. 0: an interrupt is generated on the falling edge of channels 2 and 3.
6	IrqOnRisingCh[1]	rw	0 nresetglobal	1: an interrupt is generated on the rising edge of channel 1. 0: an interrupt is generated on the falling edge of channel 1.
5	IrqOnRisingCh[0]	rw	0 nresetglobal	1: an interrupt is generated on the rising edge of channel 0. 0: an interrupt is generated on the falling edge of channel 0.
4	EnIrqCh[3]	rw	0 nresetglobal	1 enables interrupt on channel 3
3	EnIrqCh[2]	rw	0 nresetglobal	1 enables interrupt on channel 2
2	EnIrqCh[1]	rw	0 nresetglobal	1 enables interrupt on channel 1
1	EnIrqCh[0]	rw	0 nresetglobal	1 enables interrupt on channel 0
0	Enable	rw	0 nresetglobal	Enables the comparator

Table 19-4: RegCmpdCtrl

19.4 Interrupt map

interrupt source	default mapping in the interrupt manager
cmpd_irq	RegIrqHigh[2]

Table 19-5: Interrupt map

20 Physical Dimensions

20.1 SO type package

The SO type package dimensions are give in Figure 20-1 and Table 20-1

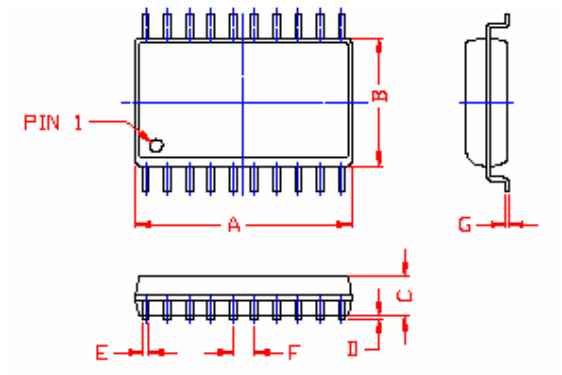


Figure 20-1. SO type package

package	A inch (mm)	B inch (mm)	C inch (mm)	D inch (mm)	E inch (mm)	F inch (mm)	G inch (mm)
SO-24	0.606 (15.39)	0.294 (7.47)	0.1 (2.54)	0.007 (0.18)	0.017 (0.43)	0.05 (1.27)	0.01 (0.26)
SO-28	0.705 (17.90)	0.294 (7.47)	0.1 (2.54)	0.007 (0.18)	0.017 (0.43)	0.05 (1.27)	0.01 (0.26)

Table 20-1. SO package dimensions

20.2 QFP type package

The QFP package dimensions are given in Figure 20-2 and Table 20-2

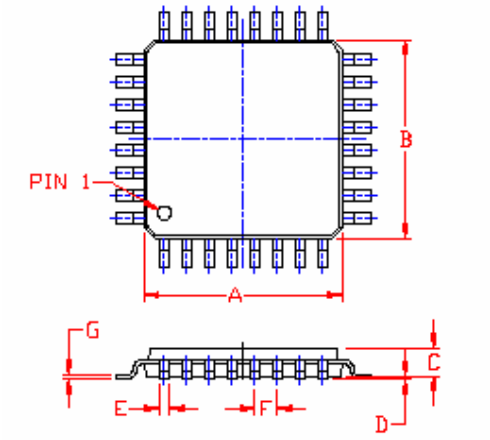


Figure 20-2. QFP type package

package	A inch (mm)	B inch (mm)	C inch (mm)	D inch (mm)	E inch (mm)	F inch (mm)	G inch (mm)
TQFP-32	0.276 7.0	0.276 7.0	0.039 1.0	0.004 0.1	0.015 0.37	0.031 0.8	0.006 0.15

Table 20-2. Typical QFP package dimensions

Contact Information

Semtech Corporation
Wireless and Sensing Products Division
200 Flynn Road, Camarillo, CA 93012
Phone (805) 498-2111 Fax : (805) 498-3804