XEMICS

# Data Book

# XX-XE88LC01/03/05

**Ultra low-power mixed-signal microcontroller**
**300 uA at 1 MIPS**
**16 + 6 bits ADC**

20000327

Preliminary information

Printed in Switzerland

Date of release 03-00
DB004-74 - Data book XX-XE88LC01-03-05

# Table of contents

# List of figures

# List of tables

# 1 Introduction

The XE8000 is a family of microcontrollers (MCU) characterised by their very low power requirement. These MCUs are perfectly adapted to manage systems working on batteries or remotely powered.

The XE8000 is conceived as an evolutionary family of MCU that can address many applications. As there are always applications that can not be covered by such a product line, XEMICS also offers full custom ASICs based on the XE8000, including additional peripherals on request. The intellectual property that the XE8000 relies on, as well as the low power digital libraries, are available from XEMICS.

This document describes the functional components of the XE8000 family and their performance. Additional information relative to specific applications is available as “Application Notes”.

## 1.1 Conventions used in this document

The negative power supply is named VSS. VSS is internally connected to the substrate of the chip. Unless otherwise stated, all voltages are given with respect to VSS.

Current is positive when flowing into a pin. This pin is said to be “sinking” current. When the current is going out of the chip, its value is negative and the pin is said “sourcing” current.

All digital words are written from MSB to LSB (MSB left, LSB right). These words are expressed with all their digits either in binary, decimal or hexadecimal format. When expressed in binary, the word has a “b” at its beginning, when expressed in decimal, the word has nothing or a “d” at its beginning, when expressed in hexadecimal, the word has an “h” at its beginning.

Unless otherwise stated, all digital signals are active high.

| | |
|---|---|
| **RegSystem** | this is a register |
| **EnableRC** | this is a bit in a register |

An unreadable bit will output 0 when read.

| code | bit (register) read-write access |
|---|---|
| r | readable |
| w | write able |
| c1 | readable, cleared by writing 1 |
| c | readable, cleared by writing any value |

**Table 1.1: access code convention**

Sometimes the abbreviation for “micro-” is “u” instead of “µ”.

# 2 XE8000 MCU Family

## 2.1 Features

The main characteristics of the XE8000 MCU family are

- Ultra low power operation
- Low voltage operation (1.2 V or 2.4 V to 5.5 V)
- High efficiency CPU
    - 1 instruction per clock cycle, for all instructions
    - 22 bits wide instructions
    - Integrated 8x8 -> 16 bits multiplier
    - 8 bit data bus
    - 64k instruction program addressing space
    - 64kB data addressing space
    - 8 addressing modes
- MTP (multiple time programmable) memory available
- Dual clock (X-tal and/or RC)
- Each peripheral can be set on/off individually for minimal power consumption
- UART and synchronous serial interface
- Watch dog
- Four 8 bit timers with PWM ability
- Advanced acquisition path
    - Fully differential analog signal path for signal and reference
    - 4x2 or 7x1 + 1 signal input
    - 2x2 reference input
    - 0.5 - 1000 programmable gain amplifier
    - Offset programmed over +- 10 full scale
    - 5 - 16 bits resolution ADC
    - Low speed modes with reduced bias current for minimal power consumption
- Bias and signal DACs for resistive bridge sensing and analog output
- Complete development tools using Windows 95 or NT graphical interface
    - Assembler
    - ANSI-C compiler
    - Source level debugger
    - CPU Simulator
    - CPU Emulator XE8000HaCE
    - Starter kits (in preparation)
    - Programmer (ProStart, includes an eval board)
    - Hardware emulators (works with XE8000HaCE, in preparation)

## 2.2 Family

The XE8000 Family ultra low-power microcontroller is made up of several members, all using the same microprocessor core and differing by the peripherals available.

The XE88LC01 is a low-power sensing microcontroller, based on the XE88LC01, with an advanced acquisition path including diferential programmable gain amplifiers and a high resolution analog to digital converter. Its main applications are dataloggers and process control.

The XE88LC02 is a low-power sensing microcontroller, based on the XE88LC01, with an additionnal LCD driver. Its main applications are metering and dataloggers.

The XE88LC03 is a low-power, low-voltage, general purpose microcontroller. Its main features are the very efficient CoolRISC core, the low voltage function and the real time clock. Its main applications are low voltage control and supervision. XE88LC03 will be superseeded by the XE88LC06 later this year.

The XE88LC04 is a low-power, low-voltage, general purpose microcontroller, based on the XE88LC03, with an additionnal LCD driver. Its main features are the very efficient CoolRISC core, the low voltage function and the real time clock. Its main applications are low voltage control and supervision.

The XE88LC05 is a low power sensing microcontroller, based on the XE88LC01, with analog outputs. Its main applications are piezoresistive sensors and 4 - 20 mA loops systems.

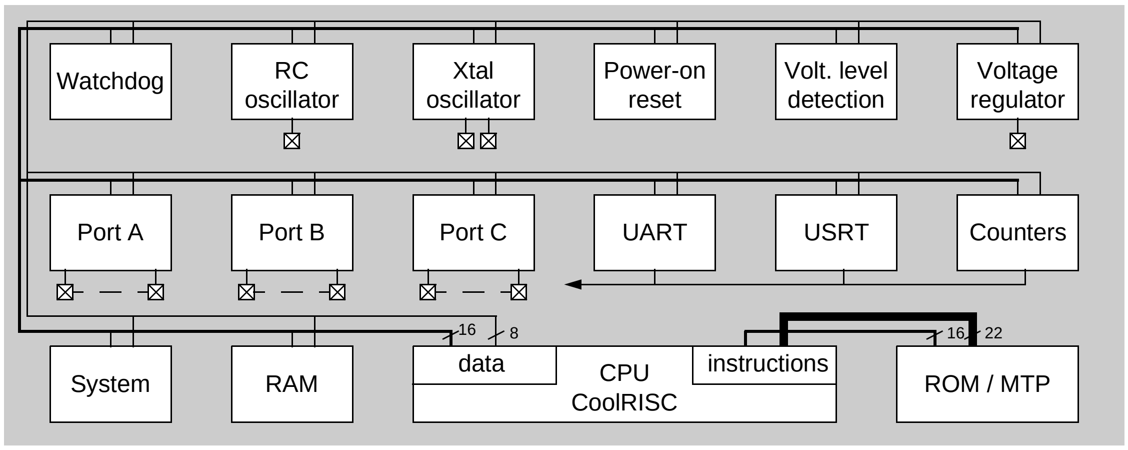


Figure 2.1: XE88LC03 block schematics

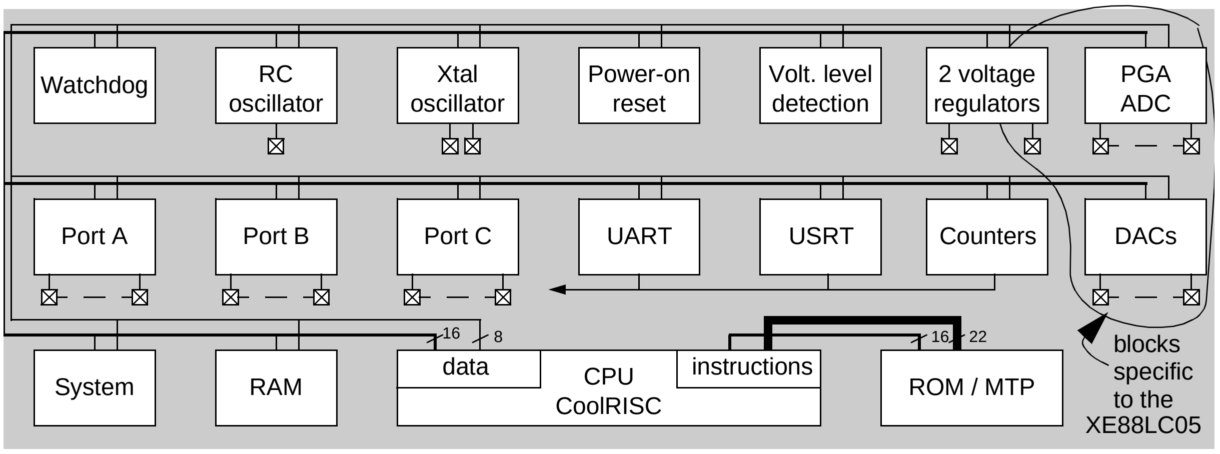


Figure 2.2: XE88LC05 block schematics.

Other microcontrollers derived from the XE8000 family members can be produced on request, either as new standard products or as customer specific circuits.

| | XE88LC01 | XE88LC02 | XE88LC03 | XE88LC04 | XE88LC05 | XE88LC06 | XE88LC07 |
|---|---|---|---|---|---|---|---|
| Supply voltage | 2.7 - 5.5 V | 2.4 - 5.5 V | 2.7 - 5.5 V | 1.2- 5.5 V for ROM<br>2.4 - 5.5 V for MTP | 2.7 - 5.5 V | 1.2- 5.5 V for ROM<br>2.4 - 5.5 V for MTP | 1.2- 5.5 V for ROM<br>2.4 - 5.5 V for MTP |
| Max speed | 2 MIPS | 4 MIPS | 2 MIPS | 4 MIPS at 2.4 V | 2 MIPS | 4 MIPS at 2.4 V | 4 MIPS at 2.4 V |
| Operating temperature | -40 - 85 °C<br>-40 - 125 °C | -40 - 85 °C<br>-40 - 125 °C | -40 - 85 °C | -40 - 85 °C | -40 - 85 °C<br>-40 - 125 °C | -40 - 85 °C<br>-40 - 125 °C | -40 - 85 °C<br>-40 - 125 °C |
| CPU | CoolRISC 816, 22 bits instructions 8 bits data HW multiplier | CoolRISC 816, 22 bits instructions 8 bits data HW multiplier | CoolRISC 816, 22 bits instructions 8 bits data HW multiplier | CoolRISC 816, 22 bits instructions 8 bits data HW multiplier | CoolRISC 816, 22 bits instructions 8 bits data HW multiplier | CoolRISC 816, 22 bits instructions 8 bits data HW multiplier | CoolRISC 816, 22 bits instructions 8 bits data HW multiplier |

**Table 2.1: List of the XE8000 family members functions**

| | XE88LC01 | XE88LC02 | XE88LC03 | XE88LC04 | XE88LC05 | XE88LC06 | XE88LC07 |
|---|---|---|---|---|---|---|---|
| Program memory | 8k Instructions = 22 kB MTP | 8k Instruction = 22 kB MTP or 6k Intructions = 16 kB ROM | 8k Instructions = 22 kB MTP | 8k Instructions = 22 kB MTP or 6k Intructions = 16 kB ROM | 8k Instructions = 22 kB MTP | 8k Instructions = 22 kB MTP or 6k Intructions = 16 kB ROM | 2k Instructions = 5.5 kB ROM or MTP |
| Data memory | 512 Bytes | 512 Bytes | 512 Bytes | 512 Bytes | 512 Bytes | 512 Bytes | 128 Bytes |
| Port A | 8 input and external interrupt | 8 input and external interrupt | 8 input and external interrupt | 8 input and external interrupt | 8 input and external interrupt | 8 input and external interrupt | 0-4 input and external interrupt |
| Port B | 8 input/output and analog | 8 input/output and analog | 8 input/output and analog | 8 input/output and analog | 8 input/output and analog | 8 input/output and analog | 8 input/output and analog |
| Port C | 8 input/output | 8 input/output | 4 to 8 input/output | 4 to 8 input/output | 8 input/output | 8 input/output | |
| Watchdog timer | yes | yes | yes | yes | yes | yes | yes |
| General purpose timers with PWM | 4 x 8 bits | 4 x 8 bits | 4 x 8 bits | 4 x 8 bits | 4 x 8 bits | 4 x 8 bits | 4 x 8 bits |
| UART | yes | yes | yes | yes | yes | yes | yes |
| 2-3 wires serial interface | transition detection + software | transition detection + software | transition detection + software | transition detection + software | transition detection + software | transition detection + software | transition detection + software |
| Voltage level detector | yes | yes | yes | yes | yes | yes | yes |
| Oscillators | 32 kHz quartz, internal RC | 32 kHz quartz, internal RC | 32 kHz quartz, internal RC | 32 kHz quartz, internal RC | 32 kHz quartz, internal RC | 32 kHz quartz, internal RC | 32 kHz quartz, internal RC |
| LCD drivers | | yes | | yes | | | |
| Analog mux | Port B and 4x2 or 7x1+1 | Port B and 4x2 or 7x1+1 | Port B | Port B | Port B and 4x2 or 7x1+1 | Port B | Port B |
| LP comparators | | 4 | | 4 | | 4 | 4 |
| PGA | gain 0.5 - 1000 | gain 0.5 - 1000 | | | gain 0.5 - 1000 | | |
| ADC | 5 - 16 bits resolution | 5 - 16 bits resolution | | | 5 - 16 bits resolution | | |
| DAC | PWM | PWM | PWM | PWM | PWM in timers 8 bit bias DAC, 4 - 16 bits signal DAC | PWM | PWM |
| Package | TQFP44, die | | SO28, TQFP32, die | | TQFP64, die | SO28, TQFP32, die | SO16, SO20, die |
| Availability | Q2/00 | samples Q1/01 | Q2/00 LC06 is a better choice for new designs | samples Q1/01 | yes | samples Q3/00 | |

**Table 2.1: List of the XE8000 family members functions**

# 3 Power supply

## 3.1 In circuit power supply principle

The power supply uses two regulators (Figure 3.1), Vreg should be connected to VDD for low voltage operation, Vmult should be connected to VDD for high voltage operation. There are several operation modes depending on the voltage range of the power supply (Figure 3.2). MTP and mixed signal blocks are limited to Middle and High voltage ranges.

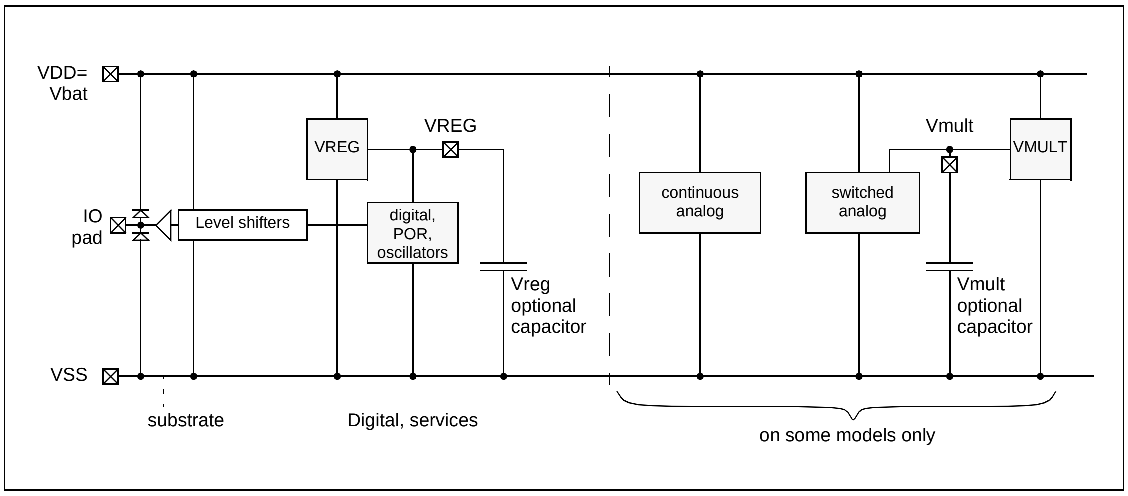


Figure 3.1: Power supply strategy.

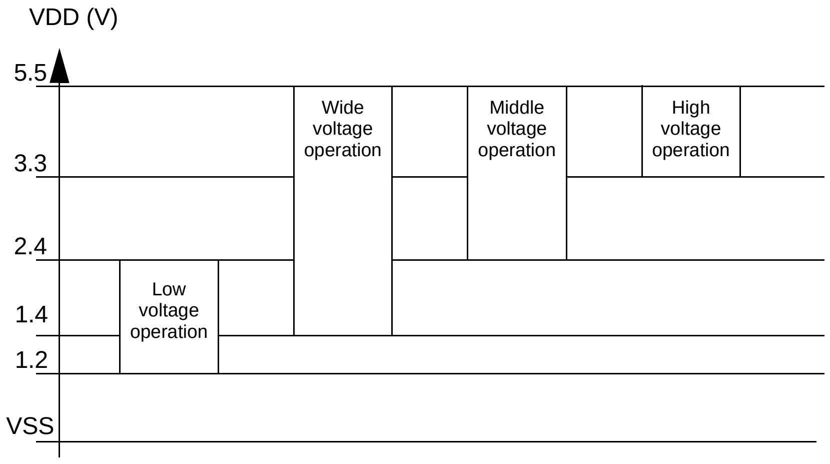


Figure 3.2: Selection of the operation mode with respect to the power supply range.

Voltage for the digital and for regular service blocks when operating in wide, middle and high voltage mode is regulated below power supply to have a minimal current requirement.

An additional high-voltage is generated when operating in middle voltage for controlling the internal analog switches.

## 3.2 Voltage regulator

All digital parts are powered through the voltage regulator. An external capacitor is needed for the regulated voltage. It should be bypassed to VDD if working in low voltage mode. The Vreg output value depends on the program memory implementation.

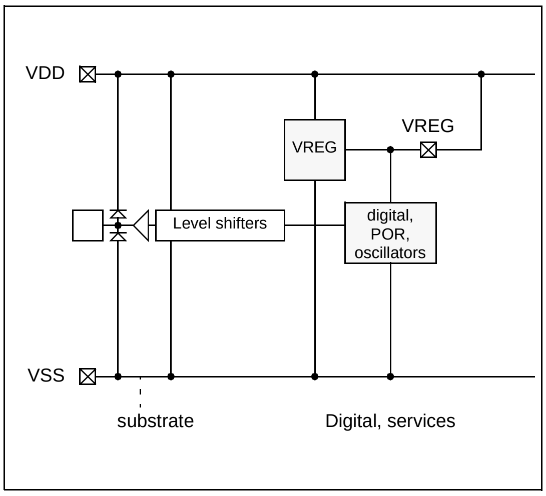


Figure 3.3: a) Power supply connection for low voltage operation.

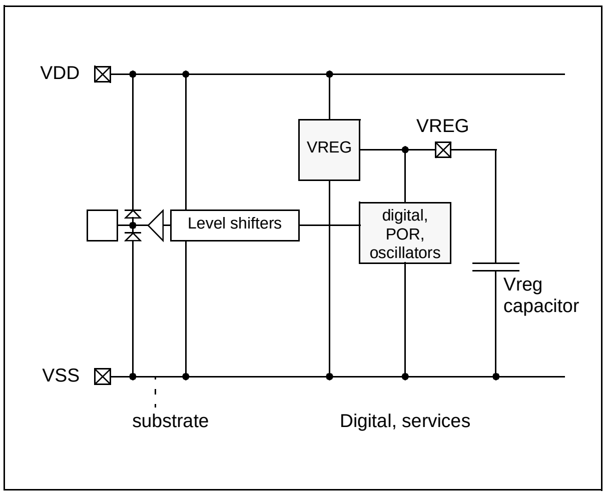


Figure 3.3: b) Power supply connection for wide voltage operation.

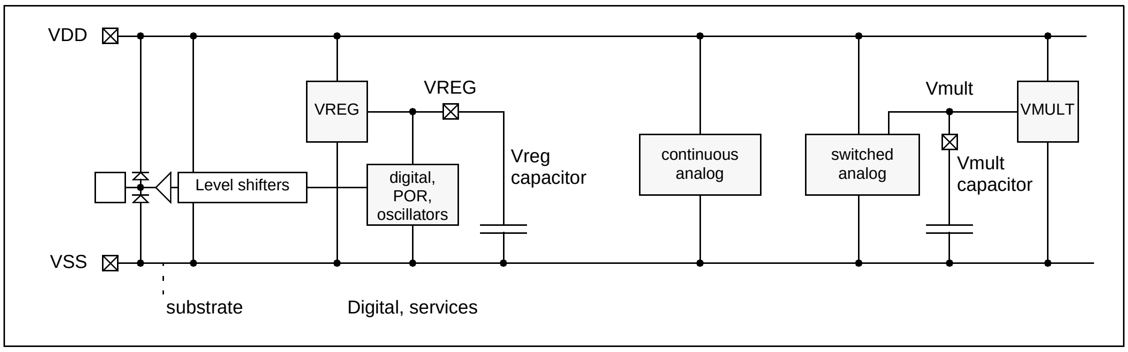


Figure 3.4: Power supply connection for middle voltage operation.

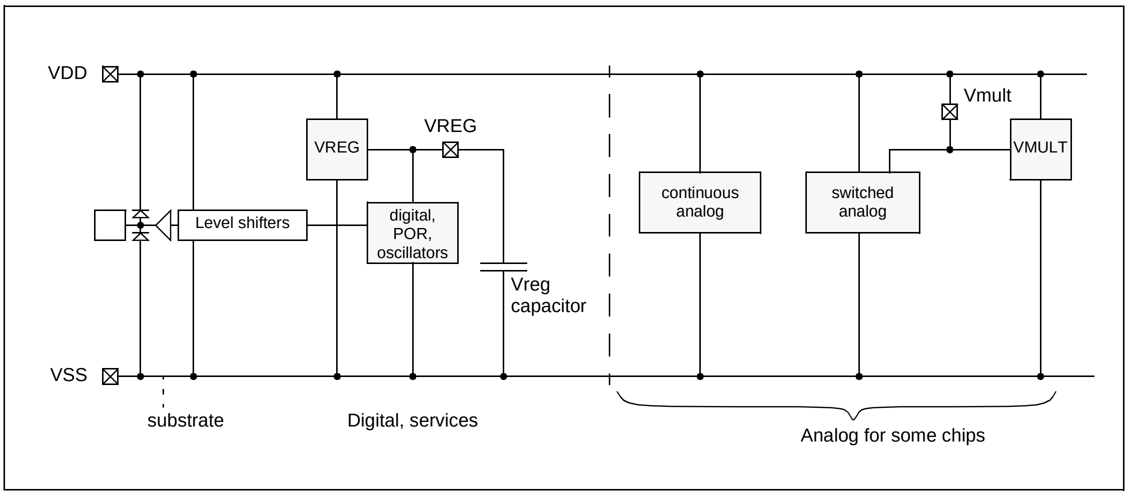


Figure 3.5: Power supply connection for high voltage operation.

| symbol | description | min | typ | max | unit | comments |
|---|---|---|---|---|---|---|
| $V_{REG}$ | regulated voltage | 1.4 | | 1.9 | V | |
| $t_{START}$ | start-up time | | | 0.5 | ms | |
| $C_{reg}$ | external load capacitor | 80 | 100 | 120 | nF | |

**Table 3.1: Voltage Regulator specifications for ROM**

| symbol | description | min | typ | max | unit | comments |
|---|---|---|---|---|---|---|
| $V_{REG}$ | regulated voltage | | 2 | | V | |
| $t_{START}$ | start-up time | | tbd | | ms | |
| $C_L$ | external load capacitor | | tbd | | nF | |

**Table 3.2: Voltage Regulator specifications for MTP**

## 3.3 Voltage multiplier

The Vmult block generates a voltage that is higher or equal to the supply voltage. The output voltage is intended for use in analog switch drivers, for example in the ADC and PGA block.

The voltage multiplier should be on when using switched analog blocks, like ADC, DAC or analog properties of the Port B under middle voltage conditions.

The clock source of Vmult is selected from the 2-bit register Vmult_fin. The normal usage is with the clock frequency of the acquisition chain. Other settings are reserved. An example of setting the regulator is as follows:

```
MOVE      RegVmultCfg0, #0b00000100; sets Vmult enable bit
```

An example of setting the regulator off follows:

```
MOVE      RegVmultCfg0, #0b00000000; resets Vmult enable bit
```

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-3 | | 00000 | rw | reserved |
| 2 | enable | 0 | rw | 0: multiplier is stopped<br>1: multiplier is active |
| 1-0 | fin[1:0] | 00 | rw | Clock source for Vmult:<br>00 : identical to acquisition chain clock (see corresponding chapter)<br>01 : reserved<br>10 : reserved<br>11 : reserved |

**Table 3.3: RegVmultCfg0**

| symbol | description | min | typ | max | unit | comments |
|---|---|---|---|---|---|---|
| Tsu | start-up time | | | 1 | ms | defined as the time to reach the minimum output voltage Vout |
| Cext | external capacitor | 1.1 | 1.8 | 2.5 | nF | |

**Table 3.4: Vmult specifications**

## 3.4 Current requirement

| Operation conditions | XE88LC01R XE88LC01M | XE88LC03R XE88LC03M | XE88LC05R XE88LC05M | Remarks |
|---|---|---|---|---|
| CPU running at 1 MIPS | 310 uA | 310 uA | 310 uA | 1 |
| CPU running at 32 kHz on Xtal, RC off | 10 uA | 10 uA | 10 uA | 1 |
| CPU halt, timer on Xtal, RC off | 1 uA | 1 uA | 1 uA | 1 |
| CPU halt, timer on Xtal, RC ready | 1.7 uA | 1.7 uA | 1.7 uA | 1 |
| CPU halt, Xtal off timer on RC at 100 kHz | 1.4 uA | 1.4 uA | 1.4 uA | 1 |
| CPU halt, ADC 12 bits at 4 kHz, PGA off | 200 uA | | 200 uA | 1,4 |
| CPU halt, ADC 12 bits at 4 kHz, PGA gain 100 | 480 uA | | 480 uA | 1,4 |
| CPU halt, LCD on, timer on Xtal | | | | 1 |
| CPU at 1 MIPS, ADC 12 bits, signal DAC 10 bits at 4 kHz, PGA off | | | 725 uA | 3,4,5 |
| CPU at 1 MIPS, ADC 12 bits, signal DAC 10 bits at 4 kHz, PGA gain 10 | | | 870 uA | 3,4,5 |
| CPU at 1 MIPS, ADC 12 bits, signal DAC 10 bits at 4 kHz, PGA gain 100 | | | 1 mA | 3,4,5 |
| CPU at 1 MIPS, ADC 12 bits, signal DAC 10 bits at 4 kHz, PGA gain 1000 | | | 1.2 mA | 3,4,5 |
| Voltage level detection | 10 uA | 10 uA | 10 uA | 2 |

**Table 3.5: Current requirement of the XE8000 family members**

**Note:** 1) Over 2.4 - 5.5 V, at 27 °C, max values.
**Note:** 2) Additional current, duration of the request is shorter than 2 ms.
**Note:** 3) Output not loaded.
**Note:** 4) Current requirement can be divided by a factor of 2 or 4 by reducing the speed accordingly.
**Note:** 5) At 2.4 V, at 27 °C, max values.

# 4 Central processing unit

## 4.1 Introduction

The XE8000 family is built around the CoolRISC 816 processor core. This is a Harvard type RISC processor (Program address is separated from data address). It is extremely efficient using large instruction words (22 bits), one clock per cycle instruction set (inclusive multiplication) and efficient pipeline.

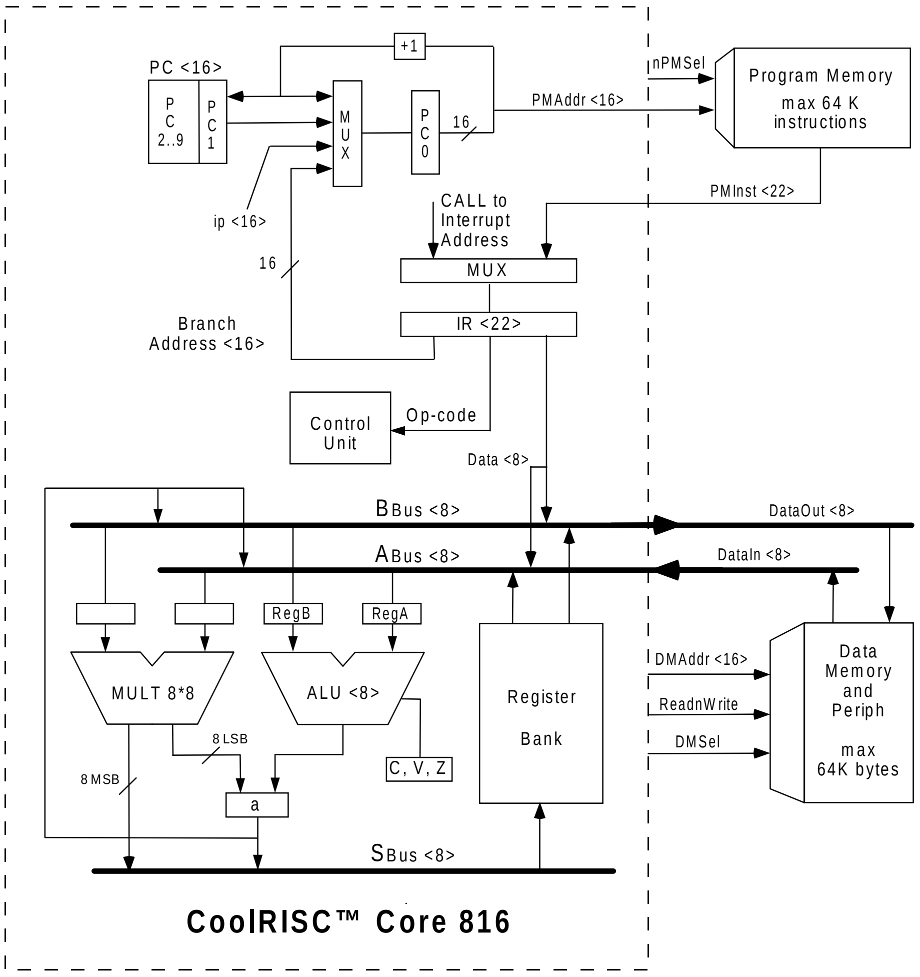


Figure 4.1: CoolRISC 816 core

### 4.1.1 Pipeline

The CoolRISC architecture is based on a 3-stage pipeline. One instruction enters the pipeline at each clock cycle and executes in a maximum of 3 cycles. The CoolRISC pipeline suffers no penalty such as delay slots or branch delays present in most RISC processors. Thus the clock count per instruction (CPI) is exactly one.
As a result the number of cycles needed to execute a task is easily determined, since it matches the number of executed instructions.

Figure 4.2 shows the timing diagram for the pipeline. Arithmetic instructions go through all three stages of the pipeline, thus executes in 3 clock cycles. A bypass mechanism is used to avoid any load delay[10].

It should be mentioned that existing 4-bit and 8-bit microprocessors typically need between 4 to 20 clocks per instruction (CPI), some newer CPU use 1 clock cycle for simple operations (MOVE) but 2 to 6 cycles for more complex operations (ADD, JPC). The efficiency of the CoolRISC architecture is far better than these microprocessors.

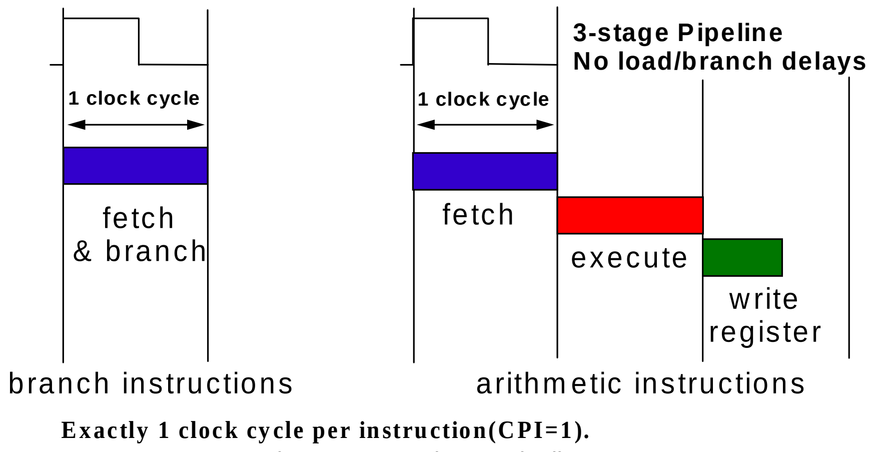


Figure 4.2: CoolRISC Pipeline

Figure 4.2 presents the timing diagram for the execution of different types of instructions.

The first instruction on Figure 4.3 is a typical ALU operation with a first operand in Data Memory (DM) and a second operand in a register. The result is stored in the destination register. During the first clock cycle, the Program Memory (PM) is pre-charged in the first phase and the instruction is read and is decoded in the second phase. During the second clock cycle, the register and the DM are read in phase 3 and the ALU operation is executed in phase 4. The last clock cycle contains only a single phase (phase 5) used to store the result in the destination register.

The second instruction shown is a Data Memory store instruction. The first clock cycle is identical for all instructions. The second clock cycle contains only phase 3 in which the value of a register is written into the DM.

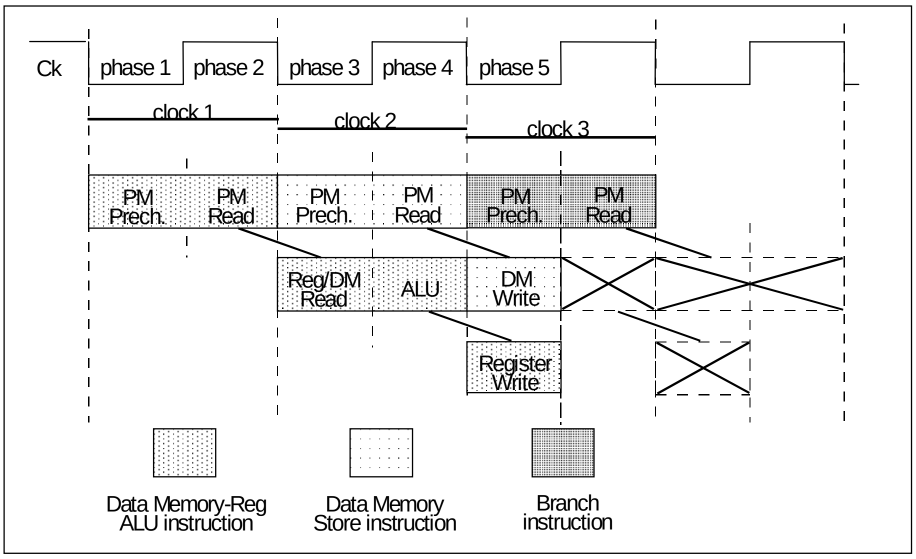


Figure 4.3: Pipeline execution of different instructions

The last instruction shown is a branch instruction. A single clock cycle is necessary for all branch instructions (conditional or unconditional jump, call, return). During phase 2, the next Program Memory address can be determined while considering an already computed test condition (computed during phase 4 of the previous instruction, which is phase 2 of the considered branch instruction). The new address is loaded into the PC at the high-to-low transition of the clock between phase 2 and 3.

Branch instructions executed in one clock do not result in branch delays that generally degrade the pipeline performance [10]. Thus, CPI=1 is not a peak value, but rather a characteristic of the CoolRISC© architecture.

Figure 4.3 shows a Data Memory-reg ALU instruction followed by a DM store instruction. The first instruction stores its result in a register during phase 5 which is phase 3 of the DM store instruction. A bypass mechanism allows the DM store instruction to read the register that is written by the preceding ALU instruction. Such a mechanism does not require load delays.

As the CoolRISC pipeline is not affected by branch or load delays [11], the pipeline hardware is simplified (no branch prediction needs to be performed [10]). This makes the CoolRISC© pipeline very efficient and low in power.

### 4.1.2 Gated clocks

The gated clock technique has been extensively used in the CoolRISC© design. It uses the ALU with input and control registers that are loaded only when an ALU operation has to be executed. During the execution of another type of instruction (branch, store, etc...), these registers are not clocked, thus no transitions occur in the ALU. This reduces power consumption.

A similar mechanism is used for the instruction registers, thus in a branch, which is executed only in the first pipeline stage, no transitions occur in the second and third stages of the pipeline.

Gated clocks can be advantageously combined with the pipeline architecture. When input and control registers have to be implemented to obtain a gated clock ALU, they are naturally used as pipeline registers.

### 4.1.3 Low frequency modes

The processor internal frequency can be reduced by a factor of 2, 4, 8 or 16. The division factor is both hardware and software controlled.

The **FREQ** instruction sets the basic division factor which is output on the processor **FreqOut[3:0]** bus. This value can be combined with other signals in an external hardware decoder to compute the final division factor which is then input on the **FreqIn[3:0]** bus.

Power consumption can be further decreased by putting the processor in the low-power standby mode with the **HALT** instruction. It will restart when an Event or an Interrupt occurs.

### 4.1.4 Stand-by Mode

The **HALT** instruction puts the processor in standby mode in which power consumption is minimum. The clock is stopped at the entrance of the processor to prevent any transition in the core.

### 4.1.5 CoolRISC© Core Features

| CoolRISC© Core | CoolRISC816 as implemented in XE88LC01-03-05 | Maximal CoolRISC816 capabilities |
|---|---|---|
| CPI (clock per instruction) | 1 | 1 |
| Pipeline | 3 stages | 3 stages |
| Branch/Load delay | no | no |
| Data Width | 8 | 8 |
| No. of Registers | 8 | 16 |
| Max. Program Memory size | 8k * 22 (= 22 kBytes) | 64k * 22 (= 360 kBytes) |
| Max. Data Memory size | 512 * 8 | 64k * 8 |
| Instruction size | 22 | 22 |
| No. of Program Memory Index Registers | 1 | 1 |
| No. of Data Memory Index Registers | 4 | 4 |
| No. of Program Memory pages | 1 * 8k | 1 * 64k |
| No. of Data Memory pages | 2 * 256 | 256 * 256 |
| No. of Data Memory addressing modes | 8 | 8 |
| Software CALL (branch & link) | yes | yes |
| No. of nested hardware CALL | 4 | 8 |
| No. of Interrupt Levels | 3 | 3 |
| Nested Interrupts | yes | yes |
| No. of EVENT levels | 2 | 2 |
| Test access | serial | serial |
| Halt mode | yes | yes |
| 8 by 8 to 16 multiplication in one instruction | yes | yes |
| Barrel Shifter | yes | yes |
| Two-Complement capabilities | yes | yes |

**Table 4.1: CoolRISC core main characteristics**

## 4.2 Programmer's Model

### 4.2.1 CoolRISC© 816 Architecture

Figure 4.1 shows the CoolRISC© Core 816 architecture which is a 8-bit microprocessor core available with 16 registers and 22-bit wide instructions.

### 4.2.2 Instruction Set

Table 4.2 presents the instruction set of the CoolRISC© 816.

The CoolRISC© provides a RISC instruction set with four main categories:
- branch instructions
- transfer instructions
- arithmetic and logic instructions
- special instructions.

Unlike most RISC microprocessors, the CoolRISC© core provides instructions that can operate with operands stored either in registers or *in the Data Memory*. All arithmetic and logic instructions can be executed with a first operand in a register and a second operand either in the Data Memory or in a second register. The result can be stored either in a third register or in the first one.

Furthermore, unlike RISC microprocessors and similarly to classic 8-bit microprocessors, the CoolRISC© architecture provides an accumulator (**a**) located at the ALU output. This accumulator stores the last ALU result and should be used as an intermediate result for the next ALU operation. This accumulator is mapped in the register bank.

Similarly, both the Branch & Link instruction of RISC microprocessors (Software Call) and the classic hardware Call are provided by the CoolRISC© architecture.

CoolRISC© architecture can be used from the programming point of view, either as a true RISC architecture or as a more classic 8-bit architecture.

The CoolRISC© 816, with its overflow flag (**V**) and its arithmetic instructions (**SHRA, CMPA, MULA, MSHRA**) fully supports signed numbers in the two-complement representation. The **MUL** & **MULA** instructions execute a 8 by 8 multiplication. Because the result is on 16 bits, the 8 MSB bits are stored in the destination register and the 8 LSB bits are stored in the accumulator **a**. All the flags (**C, Z, V**) must be considered as unknown after these instructions.

The multiple shift instructions **MSHL**, **MSHR** & **MSHRA** use the multiplication instructions with an immediate operand. For this reason, the value of **a** is different from the destination register, as in the **MUL** & **MULA** instructions. This implies that the shifted "out" bits are never lost (they are either in **a** or in the destination register), and these instructions can be used to split a byte into two bytes, with a single instruction. For example, a "SWAP r0" can be implemented as follows:

```
                                                ; r0 = 0xYZ
MSHL                    r0, #4                  ; r0 <- 0Y, a <- Z0
ADD                     r0, a                   ; r0 <- ZY
```

The conditional move instructions (**CMVD** & **CMVS**) can be used to find the maximum (or minimum) value in a table. If **i0** is a pointer to the table, r0 will contain its maximum value after the following sequence:

```
CMP(A) r0, (i0)
CMVS r0, (i0)+          ; r0 <- DM(i0)  if  r0 < DM(i0)
CMP(A)r0, (i0)
CMVS r0, (i0)+          ; r0 <- DM(i0) if r0 < DM(i0)
...........
```

| NAME | Parameters | res | op1 | op2 | FUNCTION | MODIF. |
|---|---|---|---|---|---|---|
| JUMP | addr:16 | | | | PC0 <- addr | - , - , - , - |
| | ip | | | | PC0 <- ip | |
| Jcc | addr:16 | | | | if cc then PC0 <- addr | |
| | ip | | | | if cc then PC0 <- ip | |
| CALL | addr:16 | | | | PCn <- PCn-1 (n>1), PC1 <- PC0+1, PC0 <- addr | |
| | ip | | | | PCn <- PCn-1 (n>1), PC1 <- PC0+1, PC0 <- ip | |
| CALLS | addr:16 | | | | ip <- PC0+1, PC0 <- addr:16 | |
| | ip | | | | ip <- PC0+1, PC0 <- ip | |
| RET | | | | | PCn-1 (n>0) <- PCn | - , - , - , - |
| RETS | | | | | PC0 <- ip | |
| RETI | | | | | PCn-1 (n>0) <- PCn, GIE <- 1 | |
| PUSH | | | | | PCn <- PCn-1 (n>1), PC1 <- ip, PC0 <- PC0+1 | |
| POP | | | | | ip <- PC1, PCn-1 (n>1) <- PCn, PC0 <- PC0+1 | |
| MOVE | reg, data:8 | reg | data | | res <- op1 | - , - , Z , a |
| | reg1, reg2 | reg1 | reg2 | | | |
| | reg, eaddr | reg | eaddr | | | |
| | eaddr, reg | eaddr | reg | | | - , - , - , - |
| | addr:8, data:8 | addr | data | | | |
| CMVD | reg1, reg2 | reg1 | reg2 | | if C=0 then res <- op1 | - , - , Z , a |
| CMVS | reg, eaddr | reg | eaddr | | if C=1 then res <- op1 | |
| SHL | reg1, reg2 | reg1 | reg2 | | res(bitn) <- op1(bitn-1) (0<n<8), res(0) <- 0, C <- op1(7) | C, V, Z, a |
| | reg | reg | reg | | | |
| | reg, eaddr | reg | eaddr | | | |
| SHLC | reg1, reg2 | reg1 | reg2 | | res(bitn) <- op1(bitn-1) (0<n<8), res(0) <- C, C <- op1(7) | C, V, Z, a |
| | reg | reg | reg | | | |
| | reg, eaddr | reg | eaddr | | | |
| SHR | reg1, reg2 | reg1 | reg2 | | res(bitn-1) <- op1(bitn) (0<n<8), res(7) <- 0, C <- op1(0) | C, V, Z, a |
| | reg | reg | reg | | | |
| | reg, eaddr | reg | eaddr | | | |
| SHRC | reg1, reg2 | reg1 | reg2 | | res(bitn-1) <- op1(bitn) (0<n<8), res(7) <- C, C <- op1(0) | C, V, Z, a |
| | reg | reg | reg | | | |
| | reg, eaddr | reg | eaddr | | | |
| SHRA | reg1, reg2 | reg1 | reg2 | | res(bitn-1) <- op1(bitn) (0<n<8), res(7) <- op1(7), C <- op1(0) | C, V, Z, a |
| | reg | reg | reg | | | |
| | reg, eaddr | reg | eaddr | | | |
| CPL1 | reg1, reg2 | reg1 | reg2 | | res <- NOT (op1) | -, -, Z, a |
| | reg | reg | reg | | | |
| | reg, eaddr | reg | eaddr | | | |
| CPL2 | reg1, reg2 | reg1 | reg2 | | res <- NOT (op1) +1, if op1 = 0 then C = 1 | C, V, Z, a |
| | reg | reg | reg | | | |
| | reg, eaddr | reg | eaddr | | | |
| CPL2C | reg1, reg2 | reg1 | reg2 | | res <- NOT (op1) +C, if op1 = 0 then C = 1 | C, V, Z, a |
| | reg | reg | reg | | | |
| | reg, eaddr | reg | eaddr | | | |
| INC | reg1, reg2 | reg1 | reg2 | | res <- op1 +1, if overflow then C = 1 | C, V, Z, a |
| | reg | reg | reg | | | |
| | reg, eaddr | reg | eaddr | | | |
| INCC | reg1, reg2 | reg1 | reg2 | | res <- op1 +C, if overflow then C = 1 | C, V, Z, a |
| | reg | reg | reg | | | |
| | reg, eaddr | reg | eaddr | | | |
| DEC | reg1, reg2 | reg1 | reg2 | | res <- op1 -1, if underflow then C = 0 | C, V, Z, a |
| | reg | reg | reg | | | |
| | reg, eaddr | reg | eaddr | | | |
| DECC | reg1, reg2 | reg1 | reg2 | | res <- op1 -(1 -C), if underflow then C = 0 | C, V, Z, a |
| | reg | reg | reg | | | |
| | reg, eaddr | reg | eaddr | | | |

**Table 4.2: CoolRISC 816 Instruction Set**

| NAME | Parameters | res | op1 | op2 | FUNCTION | MODIF. |
|---|---|---|---|---|---|---|
| AND | reg, data:8 | reg | reg | data | res <- op1 AND op2 | -, -, Z, a |
| | reg1, reg2, reg3 | reg1 | reg2 | reg3 | | |
| | reg1, reg2 | reg1 | reg2 | reg1 | | |
| | reg | reg | reg | eaddr | | |
| OR | reg, data:8 | reg | reg | data | res <- op1 OR op2 | -, -, Z, a |
| | reg1, reg2, reg3 | reg1 | reg2 | reg3 | | |
| | reg1, reg2 | reg1 | reg2 | reg1 | | |
| | reg | reg | reg | eaddr | | |
| XOR | reg, data:8 | reg | reg | data | res <- op1 XOR op2 | -, -, Z, a |
| | reg1, reg2, reg3 | reg1 | reg2 | reg3 | | |
| | reg1, reg2 | reg1 | reg2 | reg1 | | |
| | reg | reg | reg | eaddr | | |
| ADD | reg, data:8 | reg | reg | data | res <- op1 + op2, if overflow then C=1 | C, V, Z, a |
| | reg1, reg2, reg3 | reg1 | reg2 | reg3 | | |
| | reg1, reg2 | reg1 | reg2 | reg1 | | |
| | reg | reg | reg | eaddr | | |
| ADDC | reg, data:8 | reg | reg | data | res <- op1 + op2 + C, if overflow then C=1 | C, V, Z, a |
| | reg1, reg2, reg3 | reg1 | reg2 | reg3 | | |
| | reg1, reg2 | reg1 | reg2 | reg1 | | |
| | reg | reg | reg | eaddr | | |
| SUBD | reg, data:8 | reg | reg | data | res <- op1 -op2, if underflow then C=0 | C, V, Z, a |
| | reg1, reg2, reg3 | reg1 | reg2 | reg3 | | |
| | reg1, reg2 | reg1 | reg2 | reg1 | | |
| | reg | reg | reg | eaddr | | |
| SUBDC | reg, data:8 | reg | reg | data | res <- op1 -op2 - (1-C), if underflow then C=0 | C, V, Z, a |
| | reg1, reg2, reg3 | reg1 | reg2 | reg3 | | |
| | reg1, reg2 | reg1 | reg2 | reg1 | | |
| | reg | reg | reg | eaddr | | |
| SUBS | reg, data:8 | reg | reg | data | res <- op2 -op1, if underflow then C=0 | C, V, Z, a |
| | reg1, reg2, reg3 | reg1 | reg2 | reg3 | | |
| | reg1, reg2 | reg1 | reg2 | reg1 | | |
| | reg | reg | reg | eaddr | | |
| SUBSC | reg, data:8 | reg | reg | data | res <- op2 -op1 - (1-C), if underflow then C=0 | C, V, Z, a |
| | reg1, reg2, reg3 | reg1 | reg2 | reg3 | | |
| | reg1, reg2 | reg1 | reg2 | reg1 | | |
| | reg | reg | reg | eaddr | | |
| MUL | reg, data:8 | reg | reg | data | res <- op1 * op2 (15:8), a <- op1 * op2 (7:0), unsigned | -, -, -, a |
| | reg1, reg2, reg3 | reg1 | reg2 | reg3 | | |
| | reg1, reg2 | reg1 | reg2 | reg1 | | |
| | reg | reg | reg | eaddr | | |
| MULA | reg, data:8 | reg | reg | data | res <- op1 * op2 (15:8), a <- op1 * op2 (7:0), signed (2 complement) | -, -, -, a |
| | reg1, reg2, reg3 | reg1 | reg2 | reg3 | | |
| | reg1, reg2 | reg1 | reg2 | reg1 | | |
| | reg | reg | reg | eaddr | | |
| MSHL | reg, shift:3 | | | | a(bitn) <- reg(bitn-shift) for (bitn >= shift), reg(bitn) <- reg (bitn+8-shift) for (bitn < shift) | -, -, -, a |
| MSHR | reg, shift:3 | | | | reg(bitn) <- reg(bitn+shift) for (bitn + shift < 8), a(bitn) <- reg (bitn-8+shift) for (bitn + shift >= 8) | -, -, -, a |
| MSHRA | reg, shift:3 | | | | a <- SHRA(shift,reg), a <- SHL(8-shift,reg), SHRA propagates sign, do not use with shift=0x01 | -, -, -, a |
| CMP | reg, data:8 | | reg | data | if op2 > op1 then C <- 0, V = C AND NOT(Z), unsigned | C, V, Z, a |
| | reg1, reg2 | | reg1 | reg2 | | |
| | reg, eaddr | | reg | eaddr | | |
| CMPA | reg, data:8 | | reg | data | if op2 > op1 then C <- 0, V = C AND NOT(Z), signed | C, V, Z, a |
| | reg1, reg2 | | reg1 | reg2 | | |
| | reg, eaddr | | reg | eaddr | | |
| TSTB | reg, bit:3 | | | | Z <- NOT(reg(bit)) | -, -, Z, a |
| SETB | reg, bit:3 | | | | reg(bit) <- 1 | -, -, Z, a |
| CLRB | reg, bit:3 | | | | reg(bit) <- 0 | -, -, Z, a |
| INVB | reg, bit:3 | | | | reg(bit) <- NOT(reg(bit)) | -, -, Z, a |
| SFLAG | | | | | a(7) <- C, a(6) <- C XOR V | -, -, -, a |

**Table 4.2: CoolRISC 816 Instruction Set**

| NAME | Parameters | res | op1 | op2 | FUNCTION | MODIF. |
|---|---|---|---|---|---|---|
| RFLAG | reg | | reg | | flags <- op1, SHL op1, SHL a | C, V, Z, a |
| | eaddr | | eaddr | | | |
| FREQ | divn:4 | | | | set cpu frequency divider | -, -, -, - |
| HALT | | | | | stops CPU | -, -, -, - |
| NOP | | | | | no operation | -, -, -, - |
| PMD | s:1 | | | | if s=1 then starts program dump, if s=0 stops program dump | -, -, -, - |

Table 4.2: CoolRISC 816 Instruction Set

| | Parameters | Data Memory (DM) access | Index update | Addressing mode name |
|---|---|---|---|---|
| eaddr | addr:8 | DM(addr) | | Direct addressing |
| | (ix) | DM(ix) | | Indexed addressing |
| | (ix, offset:8) | 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 post-modification of index |
| | (ix, offset:7)+ | DM(ix) | ix <- ix+offset | |
| | -(ix) | DM(ix-1) | ix <- ix-1 | Indexed addressing with pre-modification of index |
| | -(ix, offset:7) | DM(ix-offset) | ix <- ix-offset | |

Table 4.3: CoolRISC 816 addressing modes

| | 11 Conditions | Test |
|---|---|---|
| cc | CS | C = 1 |
| | CC | C = 0 |
| | ZS | Z = 1 |
| | ZC | Z = 0 |
| | VS | V = 1 |
| | VC | V = 0 |
| | EV | (EV0 OR EV1) = 1 |
| | After CMP d, s | |
| | EQ | d = s |
| | NE | d <> s |
| | GT | d > s |
| | GE | d >= s |
| | LT | d < s |
| | LE | d <= s |

Table 4.4: CoolRISC 816 conditional jump (Jcc) conditions

| | information type |
|---|---|
| addr:8 | 8-bit address |
| addr:16 | 16-bit address |
| ip | Program Memory Index |
| ix | 4 Data Memory (DM) Indexes i0, i1, i2, i3 |
| data:8 | 8-bit data |
| offset:8 | 8-bit positive offset |
| offset:7 | 7-bit positive offset |
| bit:3 | 3-bit Bit select value : 0..7 |
| shift:3 | 3-bit shift value : 2..7 |
| divn:4 | 0b0000: nodiv, 0b1000: div by 2, 0b1100: div by 4, 0b1110: div by 8, 0b1111: div by 16 |

Table 4.5: CoolRISC 816 instruction construction

|  | 16 Registers | Function |
|---|---|---|
| reg reg1 reg2 reg3 | r0 |  |
|  | r1 |  |
|  | r2 |  |
|  | r3 | DM offset |
|  | i0l | i0[7:0] |
|  | i0h | i0[15:8] |
|  | i1l | i1[7:0] |
|  | i1h | i1[15:8] |
|  | i2l | i2[7:0] |
|  | i2h | i2[15:8] |
|  | i3l | i3[7:0] |
|  | i3h | i3[15:8] |
|  | ipl | ip[7:0] |
|  | iph | ip[15:8] |
|  | stat | status |
|  | a | accu |

**Table 4.6: CoolRISC 816 internal registers**

| registers organization |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | register name |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
| PC |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | PC |
| iph |  |  |  |  |  |  |  | ipl |  |  |  |  |  |  |  | ip |
| i0h |  |  |  |  |  |  |  | i0l |  |  |  |  |  |  |  | i0 |
| i1h |  |  |  |  |  |  |  | i1l |  |  |  |  |  |  |  | i1 |
| i2h |  |  |  |  |  |  |  | i2l |  |  |  |  |  |  |  | i2 |
| i3h |  |  |  |  |  |  |  | i3l |  |  |  |  |  |  |  | i3 |
|  |  |  |  |  |  |  |  | a |  |  |  |  |  |  |  | a |
|  |  |  |  |  |  |  |  | r0 |  |  |  |  |  |  |  | r0 |
|  |  |  |  |  |  |  |  | r1 |  |  |  |  |  |  |  | r1 |
|  |  |  |  |  |  |  |  | r2 |  |  |  |  |  |  |  | r2 |
|  |  |  |  |  |  |  |  | r3 |  |  |  |  |  |  |  | r3 |
|  |  |  |  |  |  |  |  | IE2 | IE1 | GIE | IN2 | IN1 | IN0 | EV1 | EV0 | stat |

**Table 4.7: CoolRISC 816 registers organization**

| Interrupt | CALL address |
|---|---|
| IN0 | 3 |
| IN1 | 1 |
| IN2 | 2 |

**Table 4.8: CoolRISC 816 interrupts**

The **PUSH** & **POP** instructions allow the processor to read and write its hardware stack. This can be used to “extend the depth” of the stack when nested interrupts are needed. The software implementation of nested interrupts can be achieved with the following instructions :

```
Interrupt               ; PC1 <- return address
POP                     ; ip <- PC1
MOVE eaddr1, ipl        ; eaddr1 <- return. address
MOVE eaddr2, iph        ; eaddr2 <- return. address
......                  ; 1 stack level is now freed
MOVE ipl, eaddr1        ; ipl <- eaddr1
MOVE iph, eaddr2        ; iph <- eaddr2
PUSH                    ; PC1 <- ip
RET(I)                  ; PC0 <- PC1
```

## 4.2.3 Register bank

The register bank of the 8-bit CoolRISC core 816 is described in Table 4.6 and Table 4.7.

Four data registers and an accu register are available, as well as a two-byte Program Memory Index (**ip**) and four two-byte Data Memory Index (**ix**) registers. These Index registers allow the user to address up to 64k instructions and up to 64k Data bytes. **ip** is also used to save the return address with the Software Call instruction (**CALLS**). A Status register (**stat**) is used *only* to control Interrupts and Events. **r3** can also be used as an offset register in the indexed addressing mode of the Data Memory.

If some of the 10 Data & Program Memory Index registers are not used permanently as indexes, they can be used as data registers. Furthermore, if some of them are not used in a given routine, they can also be advantageously used as temporary data registers.

Table 4.9 summarises the names and the roles of the registers.

| Register Names | Data Regs | Data Mem. Index | Prog. Mem. Index | Soft. Call |
|---|---|---|---|---|
| a | "X" | | | |
| r0 | X | | | |
| r1 | X | | | |
| r2 | X | | | |
| r3 | X | DM offset | | |
| i0l | X | X | | |
| i0h | X | X | | |
| i1l | X | X | | |
| i1h | X | X | | |
| i2l | X | X | | |
| i2h | X | X | | |
| i3l | X | X | | |
| i3h | X | X | | |
| ipl | X | | X | X |
| iph | X | | X | X |
| stat | | | | |

**Table 4.9: Registers Roles**

## 4.2.4 Program Memory addressing modes

The CoolRISC 816 provides one 16-bit Program Memory index called **ip** in order to address indirectly the 64K of Program Memory (**ipl** for the LSB bits, **iph** for the MSB).

The address field in a direct Jump instruction is of 16 bits. This allows addressing directly to the whole Program Memory space.

## 4.2.5 Data Memory addressing modes

The CoolRISC© 816 provides four 16-bit Data Memory Indexes called **ix** (**i0, i1, i2** & **i3)** in order to address 64K bytes of Data Memory (**ixl** for the LSB bits, **ixh** for the MSB).

The Data Memory is organised as 256 pages of 256 bytes. The whole memory can be addressed indirectly using ix, while only page 0 can be addressed directly.

### 4.2.5.1 Direct addressing

*Data Memory access : DM(addr:8)*

This mode is limited to the addressing of page 0. The field addr:8 of the corresponding instructions is used as a direct address (DMAddr[7:0]) while the MSB bits of the Data Memory address (DMAddr[15:8]) are equal to 0.

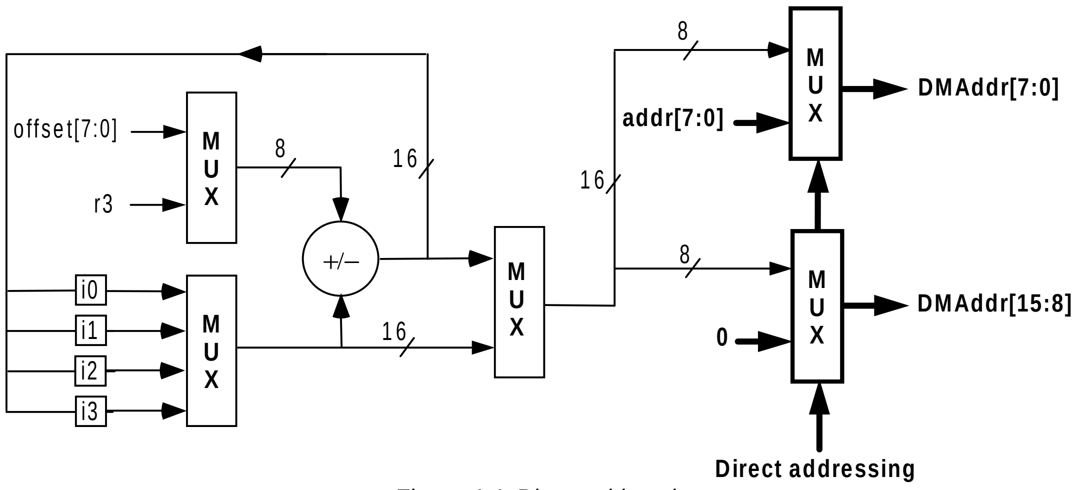


Figure 4.4: Direct addressing

### 4.2.5.2 Indexed addressing

*Data Memory access : DM(ix)*

The complete Data Memory space can be addressed with this mode. The value of the index **ix** is the Data Memory address (DMAddr[15:0]). The 8 LSB bits (**ixl**) define the offset in the page while the 8 MSB bits (**ixh**) define the page number. Therefore 256 pages of 256 bytes can be addressed.

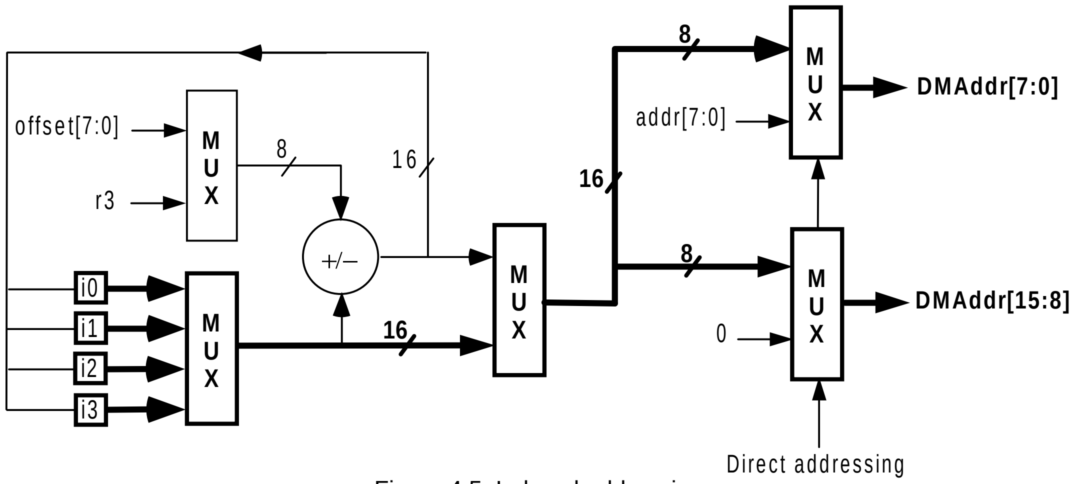


Figure 4.5: Indexed addressing

### 4.2.5.3 Indexed addressing with an immediate offset

*Data Memory access : DM(ix+offset:8)*

The 16-bit Data Memory address **DMAddr[15:0]** is calculated by the addition of an 8-bit positive **offset:8** taken in the instruction to one of the 16-bit Index **ix**.

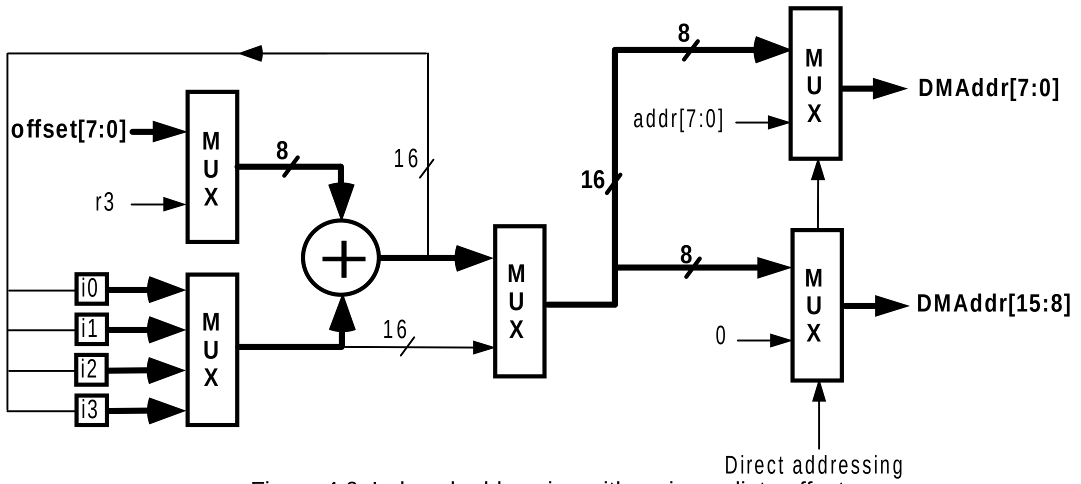


Figure 4.6: Indexed addressing with an immediate offset

### 4.2.5.4 Indexed addressing with a register offset

*Data Memory access : DM(ix+r3)*

The 16-bit Data Memory address **DMAddr[15:0]** is calculated by the addition of the 8-bit positive value of the **r3** register to one of the 16-bit Index **ix**.

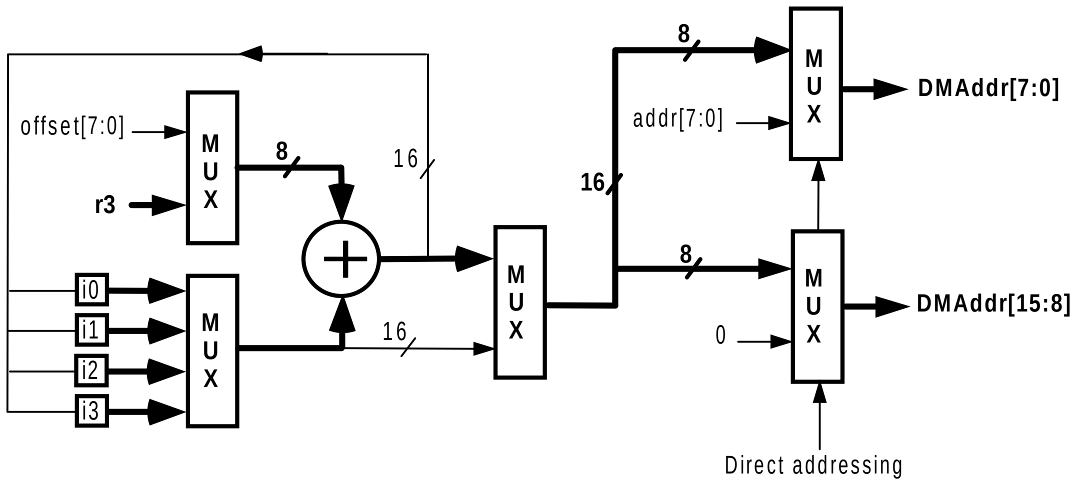


Figure 4.7: Indexed addressing with a register offset

### 4.2.5.5 Indexed addressing with Post-modification of the Index

*Data Memory access : DM(ix)*

| | |
|---|---|
| *Index Update :* | *ix <- ix+offset:7* |
| *Particular case :* | *offset:7 = 1* |

The 16-bit Data Memory address **DMAddr[15:0]** is given directly by the value of one of the 16-bit Index **ix**. The 16-bit Index **ix** is updated by the addition of the 7-bit positive **offset:7** field in the corresponding instruction.

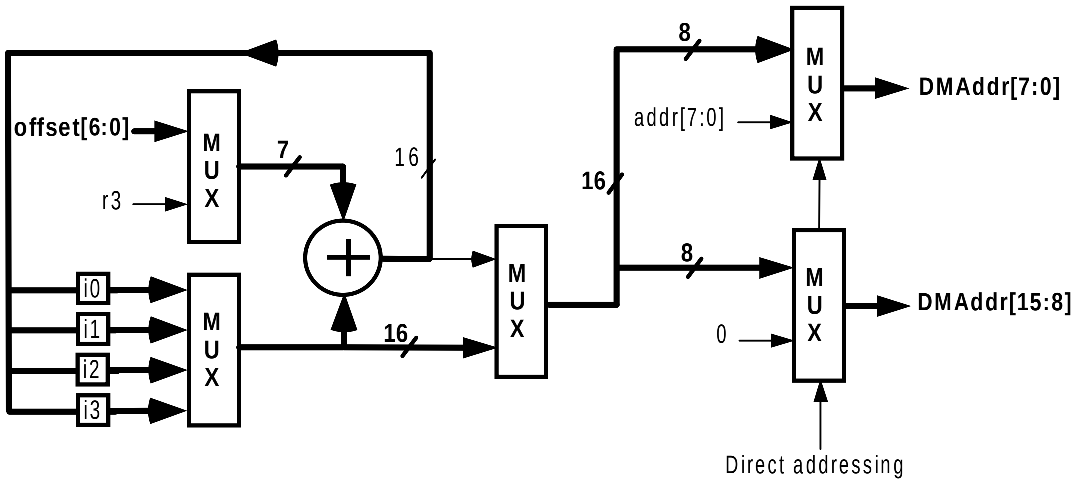


Figure 4.8: Indexed addressing with post-modification of the Index

### 4.2.5.6 Indexed addressing with Pre-modification of the Index

*Data Memory access : DM(ix-offset:7)*

| | |
|---|---|
| *Index Update :* | *ix <- ix-offset:7* |
| *Particular case :* | *offset:7 = 1* |

The 16-bit Data Memory address **DMAddr[15:0]** is calculated by the subtraction of the 7-bit positive **offset:7** from the 16-bit Index **ix**. The 16-bit Index **ix** is updated by the subtraction of the 7-bit positive **offset:7** field in the corresponding instruction.

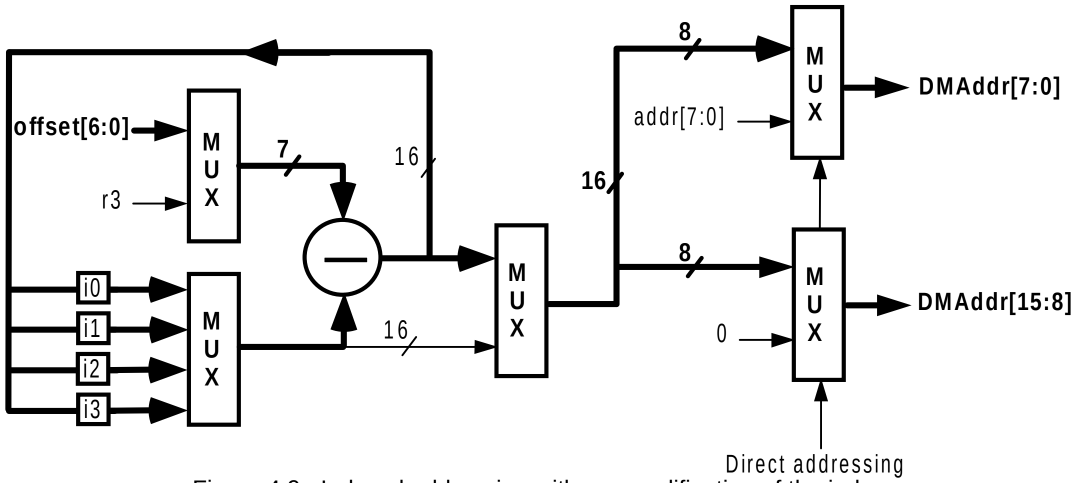


Figure 4.9: Indexed addressing with pre-modification of the index

#### 4.2.5.7 Remark on the Indexed addressing

In an Indexed Data Memory access, the Index used for the access may also be used as the destination register for the operation.

In the case of Pre/Post Index modification, the Index **ix** is updated *before* the result of the operation is stored in **ixl** or **ixh**.

### 4.2.6 Flags Z, C & V

The flag **Z** (zero) is modified by all ALU operations (including the **MOVE** into a register).

**Z** = 1 only if the ALU Output *(not the multiplier output*) is equal to 0.

The flags **C** (carry) and **V** (overflow) are only modified by arithmetic, comparison and shift operations.

In shift operations, **C** always contains the “shifted out” bit.

In arithmetic operations with unsigned numbers, **C** indicates whether an overflow or an underflow occurs (can be active either high or low depending on the operation).

In arithmetic and shift operations with signed numbers, **V** indicates whether an overflow or an underflow occurs.
**V** = 1 when overflow (or underflow).

After the comparison instructions “CMP(A) d, s” :

***C** = 0 if d > s and **V** = **C** * NOT(**Z**)*

### 4.2.7 ALU output register: a

The ALU Output Register (Figure 2.1) named **a** always contains the result of the last ALU operation. It is a temporary register that is *always* modified by ALU operations.

It is addressed as a normal register in the register bank. It should be used for temporary results, *as power-consumption is saved* if this low-power accumulator is used instead of another data register.

### 4.2.8 Program counter

The 16-bit program counter (PC) can address a Program Memory with up to 64K instructions. A hardware stack is provided for efficient subroutine and Interrupt support.

When the hardware stack is full, Interrupt is disabled until one level of the stack becomes free again (RET or RETI).

Additional subroutine levels are supported with no hardware cost through the use of the Software Call mechanism (CALLS instruction).

### 4.2.9 Branch conditions

After a comparison instruction (**CMP**), 6 conditional branch instructions are possible depending on the comparison result, after the other ALU operations, 6 conditional branch instructions are possible depending on the value of the flags. The **JEV** branch instruction is executed if one (or more) of the Event bits of the Status register is active (equal to 1).

The carry (**C**), the overflow (**V**) and the zero (**Z**) flags result from the "previous" ALU operation (which modified the flags!).

Table 4.10 summarises the different Branch conditions available.

| Branch | Test |
|---|---|
| Branch on ALU result | |
| JCS | C = 1 |
| JCC | C = 0 |
| JVS | V = 1 |
| JVC | V = 0 |
| JZS | Z = 1 |
| JZC | Z = 0 |
| Branch on Event | |
| JEV | (EV0 OR EV1) = 1 |
| Branch on CMP result | |
| JEQ | d = s |
| JNE | d <> s |
| JGT | d > s |
| JGE | d >= s |
| JLT | d < s |
| JLE | d <= s |

**Table 4.10: Branch Conditions**

The Branch **JEV** is executed if one (or more) of the Event bits of the Status register is active (equal to 1).

### 4.2.10 Call, Branch and Link

In most RISC microprocessors, only a Branch & Link mechanism is available. This mechanism saves the return address in a register. The programmer is responsible to save this address in the Data Memory before a new Software Call is used.

CoolRISC provides both Hardware and Software Call mechanisms. The Software Call stores the return address in a particular register which is the two-byte Program Memory Index **ip**. Therefore, the programmer has to save **ip** (if it is used) before a Software Call (**CALLS**).

### 4.2.11 Events and Interrupts

The 8-bit Status register (**stat**) contains the following booleans:

| | |
|---|---|
| bit0: | Event nb 0 (**EV0**) |
| bit1: | Event nb 1 (**EV1**) |
| bit2: | Interrupt nb 0 (**IN0**) |
| bit3: | Interrupt nb 1 (**IN1**) |
| bit4: | Interrupt nb 2 (**IN2**) |
| bit5: | Enable bit for all Interrupts (**GIE**) |
| bit6: | Enable bit for **IN1** (**IE1**) |
| bit7: | Enable bit for **IN2** (**IE2**) |

Events and Interrupts are Boolean flags which can be either hardware or software modified. A negative pulse on one of the **nEvent[1:0]** or **nInterrupt[2:0]** pins will set the corresponding bit to 1. In addition, a write to the Status register can either set or reset any of these bits. Therefore Interrupts and Events can be forced by software (see next chapter: “Pipeline Exception”). Note however that **a** is modified when “Software Interrupt” are generated (by an ALU operation).

Interrupts force a CALL to a fixed address (one specific address for each interrupt), save the Program counter (PC) on the stack (which will be restored by the **RETI** or **RET** instruction), and start the processor if it is in the HALT mode.

The designer must save the accumulator **a**, the flag C and the working registers (which are used in the Interrupt routine) at the beginning of the Interrupt routine, and restore them at the end as follows :

```
MOVE eaddr1, a                                ;save a & Z
SFLAG                                         ;a ( C & V
MOVE eaddr2, a                                ;save C & V
MOVE eaddr3, ri                               ;save ri
..........................
MOVE ri, eaddr3                               ;restore ri
RFLAG eaddr2                                  ;restore C & V
MOVE a, eaddr1                                ;restore a & Z
```

A disabled Interrupt (corresponding Enable bit at 0) cannot force a CALL, and cannot START the CPU if it is in the HALT mode. However, the request is taken into account and will be executed as soon as the corresponding Enable bit is set to 1, unless the interrupt bit has been successfully cleared by software in the meantime.

The general Interrupt Enable bit **GIE** (**stat[5]**), if 0, disables any Interrupt with the same principle as above.

When a CALL to an Interrupt routine is executed, **GIE** is automatically cleared in order to prevent the **CPU** executing another CALL to the same (or another) Interrupt.

When the **RETI** instruction is executed, **GIE** is automatically set to 1 in order to allow Interrupts to occur again. In order to return from an Interrupt, **RET** must be used instead of **RETI** if the programmer does not want to change the value of **GIE**.

The programmer may allow nested Interrupts by setting **GIE** to 1. But each time a new CALL to an Interrupt is executed, a new level of the hardware stack is used.

When the *hardware stack is full, Interrupts are disabled* independently of the value of **GIE**. As soon as a level of the stack is freed (**RET**, **RETI**), a pending Interrupt can be executed.

An action on the **nEvent[1:0]** pins restarts the processor if it is in the HALT mode. In contrast to the Interrupt, an Event does not force a call to a predefined address. It should be used as a handshake facility.

The **HALT** instruction is only effective if all Event bits and all non-masked Interrupt bits are cleared.

The **nEvent** and **nInterrupt** lines are active low, but a "short" negative pulse is sufficient to set the Event or Interrupt bit in the status register.

Clearing an Event or an Interrupt bit is only possible if the corresponding input line is not active. This allows the execution of an Interrupt routine as long as the corresponding input line is active.

In the Interrupt routine, it is recommended first to desactivate the Interrupt line (by commanding the corresponding peripheral to release the line), and then to clear the corresponding Interrupt bit. Thus, it will be possible to receive a new Interrupt (on the same input) as soon as the Interrupt bit is cleared.

If several Interrupts are pending, they are executed in order of priority.

Only **GIE** is reset by a hardware Reset. The other booleans must be initialised by the programmer.

Table 4.11 shows the call addresses and the priorities of the Interrupts.

| Inter. nb. | CALL ad. | Priority |
|---|---|---|
| Inter. nb 0 | 3 | Highest |
| Inter. nb 1 | 1 | Medium |

**Table 4.11: CALL addresses and priorities**

| Inter. nb. | CALL ad. | Priority |
|---|---|---|
| Inter. nb 2 | 2 | Lowest |

**Table 4.11: CALL addresses and priorities**

### 4.2.12 Pipeline exception

If an interrupt bit is set by the software (write into **stat**) the pipeline causes the next instruction to be executed "before" the CPU executes the interrupt routine. This allows to supply a parameter to a “trap” as follows :

```
SETB stat, #4                                    ; trap
MOVE a, #parameter      ; a ( parameter
```

If an Event bit is set by software, and a "JUMP on Event" (**JEV**) is the next instruction, *the first instruction will be ignored by the second.*

These are the only delays caused by the CoolRISC© pipeline.

### 4.2.13 HALT mode

The **HALT** instruction turns the processor into stand-by mode, in which power consumption is minimum. Only an Event, an Interrupt or a hardware Reset are able to wake up the microprocessor.

The **HALT** instruction is only effective if all Event bits and all enabled Interrupt bits are inactive (low).

When the processor stops because of a **HALT** instruction, the previous instruction is totally executed before the stand-by mode occurs. The next instruction will only begin when the processor restarts.

### 4.2.14 Hardware Reset

When the **nReset** signal goes low, the general Interrupt Mask bit **GIE** is reset, the CPU restarts if it was in the HALT mode, the frequency division factor is set to 1x, the PC stack is emptied, and the Test mode is reset as well as the Program Memory Dump mode.

Furthermore, at each rising edge of the external clock, a JUMP to address 0 is forced. The instruction located at address 0 will be executed when the **nReset** signal returns high.

### 4.2.15 Low frequency modes

As explained in chapter 1.6, the processor internal frequency can be reduced by a factor of 2, 4, 8 or 16. The division factor is both hardware and software controlled.

The **FREQ** instruction sets the basic division factor which is output on the **FreqOut[3:0]** bus. The instruction that follows **FREQ** is already executed with the new processor frequency.

# 5 Memory

## 5.1 Memory organisation

The MCU uses a Harvard architecture, so that memory is organised in two separated fields: program memory and data memory. As the memory is separated, the central processing unit can read/write data at the same time it loads an instruction. Peripherals and system control registers are mapped on data memory space.

Program memory is made in one page (program address bus is 16 bits wide). Data is made of several 256 bytes pages.

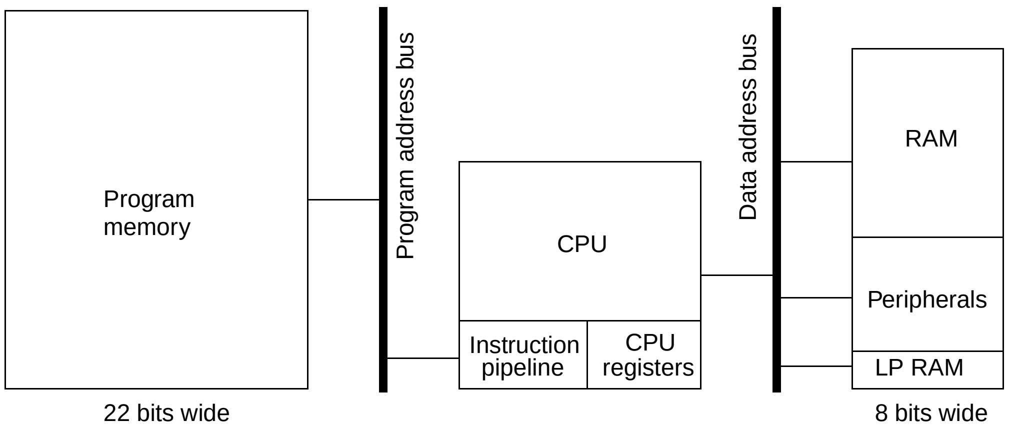


Figure 5.1: Memory organization

## 5.2 Program memory

The program memory is implemented as Multiple Time Programmable (MTP) Flash memory or Read-Only Memory (ROM).

The power consumption of MTP and ROM is linear with the access frequency (no static current).
Memory sizes :

- Flash MTP: 8192 x 22 bits
- ROM : 8192 x 22 bits

| block | size | address |
|---|---|---|
| ROM/MTP | 8192 x 22 | H0000 - H1FFF |

**Table 5.1: Program addresses**

## 5.3 Data memory

The data memory is implemented as static Random-Access Memory (RAM). The RAM size is 512 x 8 bits, plus a 8 x 8 low memory registers (named LP RAM) that require very low current when addressed. Programs using these registers instead of regular RAM will spare current.

**Note:** The RAM content is not defined at power-up.
**Note:** The registers in Data memory (LP RAM) are not related to the CPU registers.

| block | size | address |
|---|---|---|
| LP RAM | 8 x 8 | H0000 - H0007 |
| RAM | 512 x 8 | H0080 - H027F |

Table 5.2: RAM addresses

## 5.4 Peripherals mapping

| block | size | address | Page | Specific chapter |
|---|---|---|---|---|
| LP RAM | 8x8 | H0000-H0007 | Page 0 | “Memory” on page 43 (low power memory) |
| Reserved | 8x8 | H0008-H000F | | |
| System control | 16x8 | H0010-H001F | | “System operating modes” on page 47 |
| Port A | 8x8 | H0020-H0027 | | “Parallel IO ports” on page 69 |
| Port B | 8x8 | H0028-H002F | | |
| Port C | 4x8 | H0030-H0033 | | |
| Port D | 4x8 | H0034-H0037 | | |
| MTP | 4x8 | H0038-H003B | | “Memory” on page 43 (used only for MTP programming) |
| Event | 4x8 | H003C-H003F | | “System operating modes” on page 47 |
| Interrupts control | 8x8 | H0040-H0047 | | |
| USRT | 8x8 | H0048-H004F | | “Universal Synchronous Receiver/Transmitter (USRT)” on page 101 |
| UART | 8x8 | H0050-H0057 | | “Universal Asynchronous Receiver/Transmitter (UART)” on page 79 |
| Counters | 8x8 | H0058-H005F | | “Counters/timers” on page 87 |
| ADC | 8x8 | H0060-H0067 | | “Acquisition chain” on page 103 |
| Reserved | 12x8 | H0068-H0073 | | |
| DACs | 8x8 | H0074-H007B | | “Analog outputs” on page 111 |
| Other (Vmult, VLD) | 4x8 | H007C-H007F | | “Power supply” on page 21 and “Voltage Level Detector” on page 99 |
| RAM1 | 128x8 | H0080 - H00FF | | |
| RAM2 | 256x8 | H0100 - H01FF | Page 1 | |
| RAM3 | 128x8 | H0200 - H027F | Page 2 | |

Table 5.3: Peripherals addresses

## 5.5 Address pack

A standardized peripheral address naming is provided to help developpers. The address pack is shown in XEMICS application note AN8000.01.

## 5.6 MTP Flash memory programmation

### 5.6.1 Introduction

The programmation process uses the test interface of the CPU (the main clock of the processor, and its testin, testout and testck ports), a reset port (active high) and another clock (ptck). It requires that the XE88LC01/03/05 is in test mode (TEST/VPP above VDD), connecting internal signals to pins.

More detailed information is given in XEMICS application note AN8000.02.

Be aware that the XE88LC02/04/06 and above use another programming algorithm.

### 5.6.2 MTP Registers

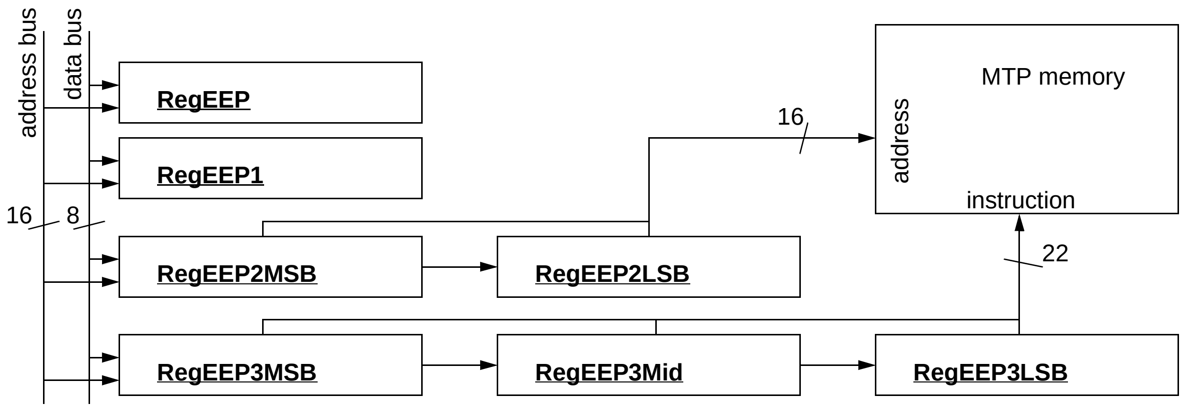


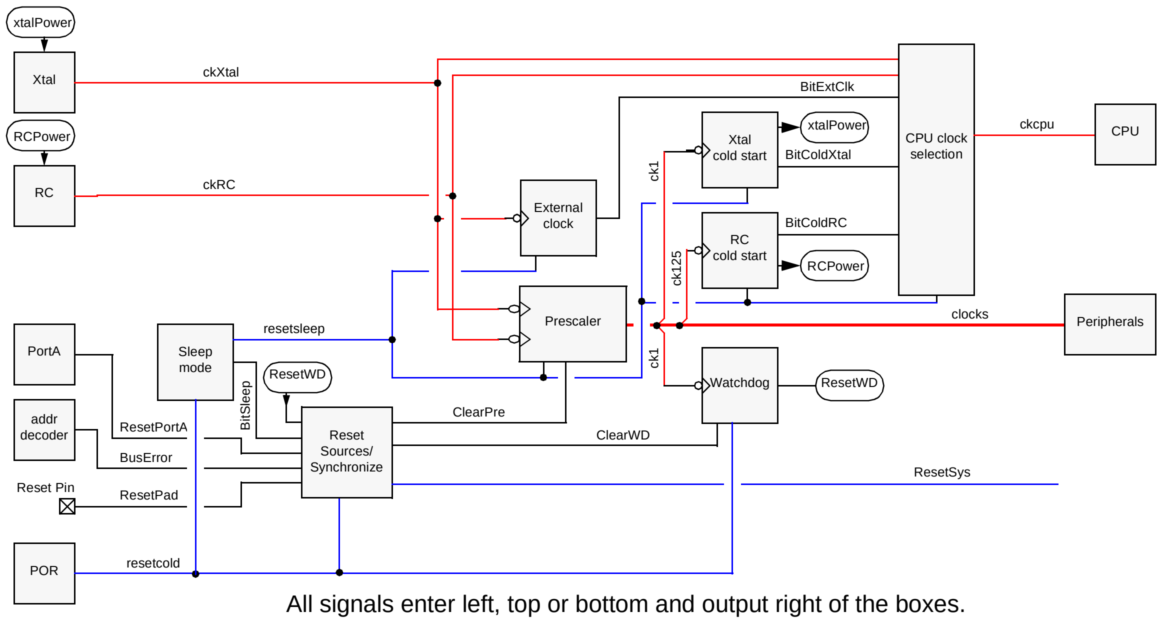
Figure 5.2: MTP registers organisation

| register name | address |
|---|---|
| RegEEP | h0038 |
| RegEEP1 | h0039 |
| RegEEP2 | h003A |
| RegEEP3 | h003B |

**Table 5.4: MTP Registers**

# 6 System operating modes

The system block controls resets, interrupts, events, clocks and operating modes.



All signals enter left, top or bottom and output right of the boxes.
Figure 6.1: System block

## 6.1 Operating modes

The XE8000 system can operate in three different modes from which two are low-power dissipation modes (StandBy and Sleep). In addition to that, most peripherals also have several low power modes.

- Active mode (normal operation)
- StandBy mode (CPU halt, oscillator on)
- Sleep mode (oscillator off, RAM state maintained)

### 6.1.1 Start-up and Reset states

These states can only be entered following reset (see Figure 6.3). A correct start-up of the whole microcontroller is ensured.

### 6.1.2 Active mode

This is the running mode where all peripherals can work and the CPU executes instructions.

### 6.1.3 Standby mode

Executing a HALT instruction puts the XE8000 into the StandBy mode. The voltage regulators, Oscillators, Watchdog timer, interrupts, events and counters can be operated.
However, the CPU stops, since the clock related to instruction fetching and execution stops. Registers, RAM, and I/O pins retain their states prior to StandBy mode.
StandBy is cancelled by a RESET (except BusError reset) or an Interrupt/Event request if enabled.

**Note:** Unless disabled, watchdog timer is running in standby mode.

### 6.1.4 Sleep mode

This is a very low-power mode in which all peripherals are stopped. All internal registers and RAM keep the value before Sleep mode. This mode can be used when no time-keeping is necessary. Both oscillators are stopped.

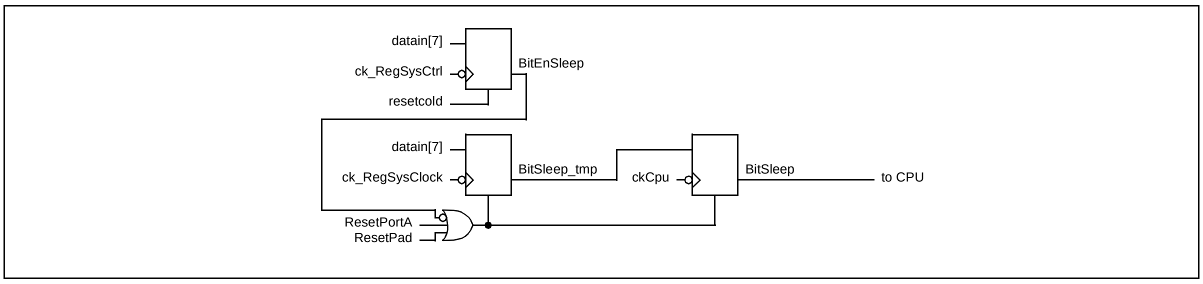


Figure 6.2: sleep mode structure

To put the XE8000 in Sleep mode first the **SleepEn** (sleep enable) bit in **RegSysCtrl** must be set to 1. The only Reset which clears this bit is the Power-On-Reset. It can be cleared at any time also by writing 0 to this location. Once the **SleepEn** bit is set to one, writing the **Sleep** bit in **RegSysReset** immediately puts the XE8000 in the Sleep mode.

There are only three possible ways to wake-up from the Sleep mode :

1. In case of Power-Down followed by Power-On, the Power-On-Reset initializes the whole system. In this case, the peripheral register contents and the RAM information is lost.
2. RESET pad, no information loss.
3. PortA reset combination, no information loss.

If the Reset pin or PortA reset is the Wake-Up condition, the **SleepEn** bit is not cleared and this therefore provides the information that the circuit was previously running and that the XE8000 should now wake up from Sleep mode.

**Note:** Due to the pipelined architecture, the instruction following the writing of the **Sleep** Bit may be executed under certain conditions. Therefore, the writing of the **Sleep** bit must be followed by a NOP.

**Note:** Watchdog timer is stopped in sleep mode.

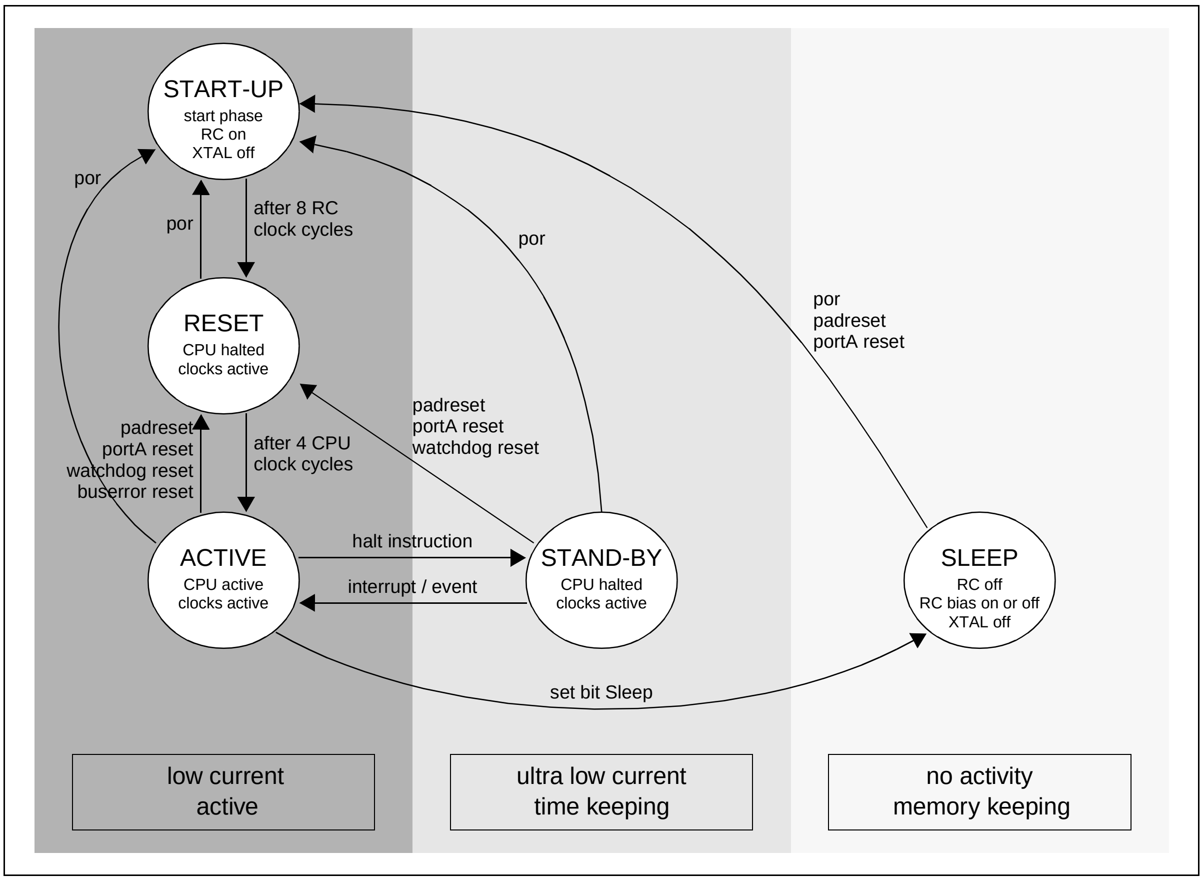


Figure 6.3: System operating modes. Most peripherals also have several low power modes

| register name | address (hex) | comments |
|---|---|---|
| RegSysCtrl | H0010 | System sleep and reset flags |
| RegSysReset | H0011 | System sleep and reset enable |
| RegSysClock | H0012 | Clock selection |
| RegSysMisc | H0013 | Special functions |
| RegSysWd | H0014 | Watchdog |
| RegSysPre0 | H0015 | Prescaler |
| RegSysPre1 | H0016 | |
| RegSysTest1 | H0017 | Reserved |
| RegSysTest2 | H0018 | |
| RegSysTest3 | H0019 | |
| RegSysTestAna | H001A | |
| RegSysRCTrim1 | H001B | RC clock frequency select |
| RegSysRCTrim2 | H001C | |
| RegSysVIreg | H001D | Internal regulator setting |
| - | H001E | Reserved |
| RegSysWarm | H001F | Reserved |

**Table 6.1: System registers**

## 6.2 Resets

To initialize the XE8000, a system reset must be executed.

This can be performed in 5 ways:

- initial reset from the Power-On Reset (POR)
- external RESET from the RESET pin
- port A reset combination (programmable)
- watchdog timer reset (programmable enable)
- wrong data memory address reset = BusError reset

Power-on reset is treated as a special case with an additional ColdStart delay on RC oscillator. This ColdStart delay guarantees a stable oscillator clock once it is finished (see Figure 6.3). The circuit always starts with the RC oscillator because its Start-Up time is very short compared to the Quartz oscillator. Once the RC ColdStart delay of several RC clock is finished, the oscillator selector detecting only the RC clock needs an additional RC period to switch to RC oscillator for System clock. From now on all Reset sources go via the Reset Synchronizer logic which prolongs any reset by 3 whole System clock periods and then releases the System reset on the falling edge of the System clock. The program counter for Instruction words is cleared to 0 and first Jump Main instruction is executed after the end of System Reset.

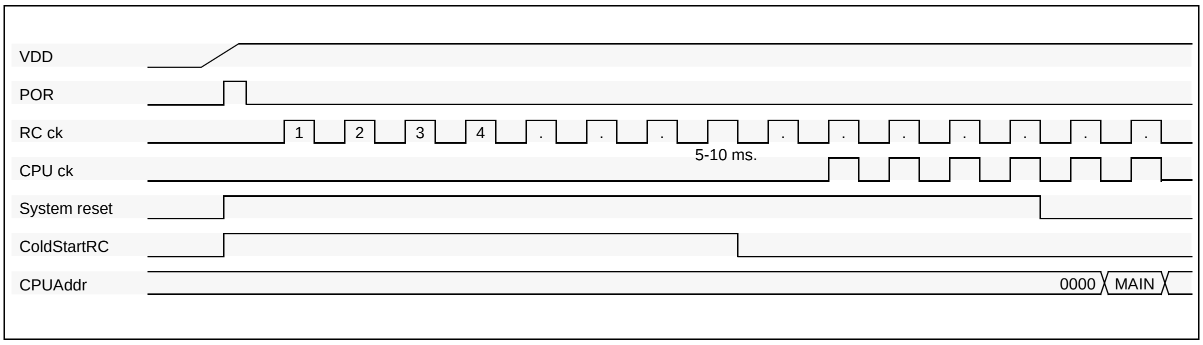


Figure 6.4: power-on reset start

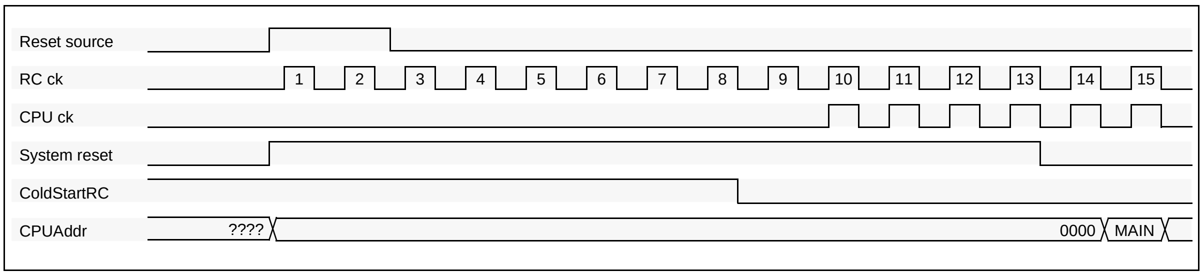


Figure 6.5: wake from sleep (short reset)

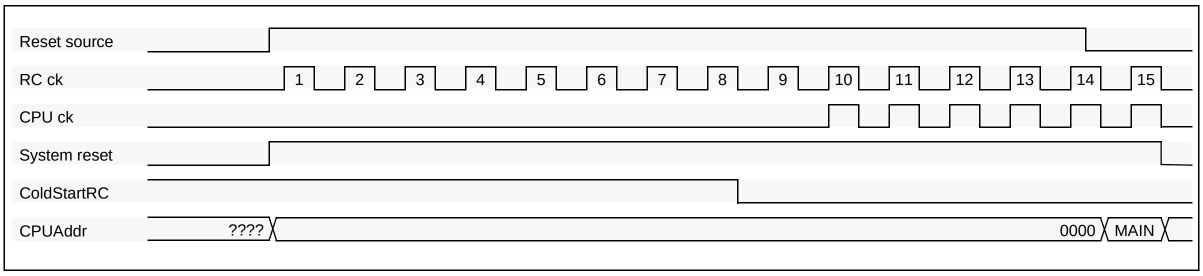


Figure 6.6: wake from sleep (long reset)

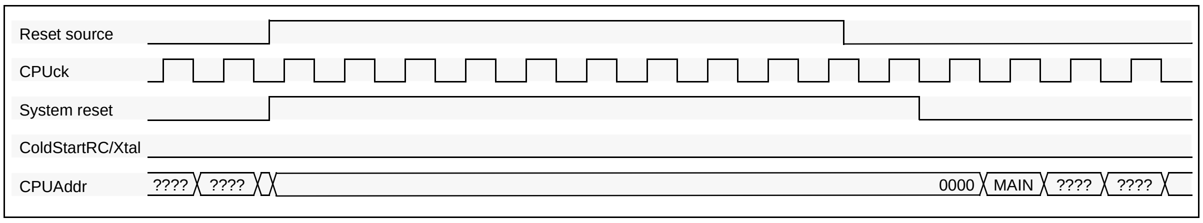


Figure 6.7: reset in active mode

### 6.2.1 Initial reset from the Power-On-Circuit

At power-on the POR circuit supplied by the internal Voltage regulator (which also supplies the whole digital part ) generates a Reset until the regulated voltage is high enough for the IC to work.

### 6.2.2 External Reset from the RESET pin

System reset can be activated by using the RESET pin.

### 6.2.3 PortA programmed Reset combination.

As described in the PortA description a PortA reset can be programmed. When Input PortA status matches the programmed combination in **RegPARes0** and **RegPARes1** a reset is generated which lasts until Reset portA status is removed from PortA inputs. Direct non-debounced PortA status is used for this reset.

### 6.2.4 Watch-Dog timer Reset

The Watchdog Timer will generate a RESET if it is not cleared on time. See chapter WATCHDOG TIMER for details.

### 6.2.5 BusError reset

Address space is distributed as shown in Register mapping to different peripherals. If an unused address region was addressed, then the BusError reset is executed.

### 6.2.6 Reset Registers

Two registers are dedicated for reset status and control, **RegSysReset** and **RegSysCtrl**. The bits **Sleep** and **SleepEn** are also located in those registers but have nothing to do with reset and are described in a dedicated chapter.

**RegSysReset** gives reset source information.

| | |
|---|---|
| • **ResBusError** | an undefined peripheral register was addressed. |
| • **ResWD** | the Watchdog timer generated a reset. |
| • **ResPortA** | a PortA combination generated a reset. |
| • **ResPad** | Reset pad was used . |
| • **ResPadSleep** | Reset pad was used during Sleep mode. |

**RegSysCtrl** enables reset sources

| | |
|---|---|
| • **EnBusError** | enables reset source from bus error |
| • **EnResWD** | enables reset source from watchdog |
| • **EnResPConf** | has a special function. It is a reset source selection for Port Configuration reset. It can be activated either with power-on reset only or with a combination of power-on reset and any other reset source. |

| EnResPConf | resetpconf |
|---|---|
| 1 | any reset source |
| 0 | por ONLY |

**Table 6.2: EnResPConf**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | Sleep | resetpor or resetPortA or resetPad or resetCold | w | Sleep flag (read value is 0) |
| 6 | ResPor | 0 | r t | Por status in test mode |
| 5 | ResBusError | 0 resetcold | r c | reset source is BusError |
| 4 | ResWD | 0 resetcold | r c | reset source is Watchdog |
| 3 | ResPortA | 0 resetcold | r c | reset source is PortA combination |
| 2 | ResPad | 0 resetcold | r c | reset source is debounced pad reset |
| 1 | ResPadSleep | 0 resetcold | r c | reset source is pad reset (in sleep mode) |
| 0 | | 0 | r | reserved |

**Table 6.3: RegSysReset**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | SleepEn | 0 resetpor | r w | enable Sleep flag |
| 6 | EnResPConf | 0 resetcold | rw | enable reset port config when resetsystem |
| 5 | EnBusError | 0 resetcold | rw | enable reset from BusError |
| 4 | EnResWD | 0 resetcold | rw | enable reset from Watchdog this bit can not be set to 0 by software |
| 3-0 | -- | 0000 | r | unused |

**Table 6.4: RegSysCtrl**

### 6.2.7 Oscillators and prescaler control

This is described in the Oscillators chapter.

### 6.2.8 Other features

- **RCOnPA0** if set to one, an event (or interrupt) on pad PA[0] will start the RC oscillator by setting **Ena-bleRC**
- **DebFast** debouncer clock selection. This clock is used for all digital debouncer in the XE8000
- **OutputCKXtal** output Xtal clock on portPB[3]
- **OutputCKCpu** output CpuCkout clock on portPB[2].

| DebFast | debouncer frequency |
|---|---|
| 1 | 8KHz |
| 0 | 256Hz |

**Table 6.5: Debouncer frequency**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | DisResPad | 0 testpad | r wt | disable resetpad in test mode |
| 6-4 | -- | 000 | r | unused |
| 3 | RCOnPA0 | 0 resetsleep | r w | Start RC on PA[0] |
| 2 | DebFast | 0 resetsleep | r w | debouncer clock speed (0=slow) |
| 1 | OutputCkXtal | 0 resetsleep | r w | output CkXtal on pad PB[3] |
| 0 | OutputCkCpu | 0 resetsleep | r w | output CKOutCpu on pad PB[2] |

**Table 6.6: RegSysMisc**

## 6.3 Interrupt

### 6.3.1 Features

The XE8000 support 16 interrupt sources, namely:

- 8 external inputs from PortA
- 4 from the internal Counters A,B,C and D
- 2 from the internal Prescaler (128Hz and 1Hz )
- 1 from internal Voltage Level Detector
- 1 from Acquisition chain

These interrupts have 3 levels of priority.

### 6.3.2 Overview

The interrupts are divided into 3 categories with different priorities. The priorities are fixed and are described below :

- High: acquisition chain, counterA, counterC, prescaler 128Hz, UartTx, UartRx
- Mid: PortA[0,1,4,5], prescaler 1Hz, VLD
- Low: PortA[2,3,6,7], counterB, counterD.

There is an interrupt flag register associated with each priority. The interrupt flag registers are **RegIrqHig**, **RegIrqMid** and **RegIrqLow**.

The rising edge of the interrupt (IRQ) source sets the Interrupt flag if it is enabled.

The Interrupt flag is automatically cleared following system Resets and can be cleared by software. This is achieved by writing the corresponding Flag in the RegIrqXXX register to 1. For definitively clearing the interrupt, one has to disable the corresponding bit in the CoolRISC status register . For example if an interrupt is generated by the pad 0 of port A:

```
move                    RegIrqMid, #0x01
clrb                    stat,#3
```

Each interrupt (IRQ) source can be enabled or disabled by software with the help of the Interrupt enable registers **RegIrqEnHig**, **RegIrqEnMid** and **RegIrqEnLow**. There is a bit within each of these registers corresponding to the list above. For example, within **RegIrqEnHig**, there is a bit corresponding to each of CounterA, CounterC, Prescaler 128Hz, UartTx and UartRx that enables/disables the associated interrupt accordingly.

### 6.3.3 PortA interrupts

It is possible to use each pin of PortA (8 pins) as an independent edge triggered IRQ input.
Bitwise configuration (debounced or non-debounced / falling or rising edge) is performed using reserved registers in portA.
The debouncer frequency can be either 256Hz (slow debouncer clock) or 8KHz (fast-debouncer clock). The debounce clock is common for all 8 PortA inputs (and also for the RESET pad).
When debounced, the signal on portA has to be stable for one to two periods of selected internal debounce clock to ensure that the new logical value is accepted.

### 6.3.4 Counters A, B, C and D interrupts

Counters generate Interrupts only when they work as normal counters (as opposed to PWMs).

Loops Counters generate interrupts regularly, depending on their mode of operation (up/down counting). If more interrupts arrive from the same counter before its interrupt is served, second and subsequent interrupts are ignored. This can be the case when interrupts arrive too quickly or the interrupt service routine is currently serving some higher priority interrupts.

### 6.3.5 Prescaler interrupts

Two interrupts are available from the Prescaler, 128Hz and 1Hz.

Both are generated by the low 15-stage divider.This means that they can be accurate at 128Hz and 1Hz only when the Xtal 32768 Quartz oscillator is enabled. If the Xtal is not enabled, they are derived from the RC oscillator frequency.

The Prescaler can be resynchronized at any time by resetting it.

### 6.3.6 Voltage Level Detector interrupt

The Voltage Level Detector generates an interrupt following supply voltage comparison in response to a comparison request.

The interrupt indicates the comparison is complete.

### 6.3.7 Acquisition chain interrupt

The Acquisition chain generates an interrupt when the acquisition is completed.

### 6.3.8 Interrupt priorities

Two registers have been provided to facilitate the writing of interrupt service software namely :

- **RegIrqPriority**
- **RegIrqIrq**

The first, **RegIrqPriority,** contains the ID-code of the highest Priority interrupt. The table below shows these priorities which correspond to organization of interrupts in the registers above and their ID-code.

| RegIrqPriority | Irq on CPU (priority) | highest priority interrupt set |
|---|---|---|
| HFF | none | no interrupt |
| H17 | IN0 (high) | IrqAc |
| H16 | IN0 (high) | IrqPre1 |
| H14 | IN0 (high) | IrqCntA |
| H13 | IN0 (high) | IrqCntC |
| H11 | IN0 (high) | IrqUartTx |
| H10 | IN0 (high) | IrqUartRx |
| H0F | IN1 (mid) | reserved |
| H0E | IN1 (mid) | reserved |
| H0D | IN1 (mid) | IrqPA[5] |
| H0C | IN1 (mid) | IrqPA[4] |
| H0B | IN1 (mid) | IIrqPre2 |
| H0A | IN1 (mid) | IrqVLD |
| H09 | IN1 (mid) | IrqPA[1] |
| H08 | IN1 (mid) | IrqPA[0] |
| H07 | IN2 (low) | IrqPA[7] |
| H06 | IN2 (low) | IrqPA[6] |
| H05 | IN2 (low) | IrqCntB |
| H04 | IN2 (low) | IrqCntD |
| H03 | IN2 (low) | IrqPA[3] |
| H02 | IN2 (low) | IrqPA[2] |

**Table 6.7: all interrupts and their priorities**

ID-codes H15, H12, H11, H10, H1, and H0 are not used in this product. Code H17 is only used in some products. After any system Reset the **RegIrqPriority** value is HFF indicating no interrupt request has been issued subsequently to the Reset.

The second, **RegIrqIrq,** indicates the priority levels of the current IRQs.

The three levels of interrupts map directly to those supported by the CoolRISC (IN0, IN1 & IN2).

For more information, see the documentation about the CoolRISC816 core.

| register name | address (hex) |
|---|---|
| RegIrqHig | H0040 |

**Table 6.8: Interrupt registers**

| register name | address (hex) |
|---|---|
| RegIrqMid | H0041 |
| RegIrqLow | H0042 |
| RegIrqEnHig | H0043 |
| RegIrqEnMid | H0044 |
| RegIrqEnLow | H0045 |
| RegIrqPriority | H0046 |
| RegIrqIrq | H0047 |

**Table 6.8: Interrupt registers**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | IrqAc | 0 resetsystem | r c1 | interrupt from acquisition chain (when available) |
| 6 | IrqPre1 | 0 resetsystem | r c1 | interrupt from prescaler (128 Hz) |
| 5 | -- | 0 | r | unused |
| 4 | IrqCntA | 0 resetsystem | r c1 | interrupt from CounterA |
| 3 | IrqCntC | 0 resetsystem | r c1 | interrupt from CounterC |
| 2 | -- | 0 | r | unused |
| 1 | IrqUartTx | 0 resetsystem | r c1 | interrupt from Uart Transmitter |
| 0 | IrqUartRx | 0 resetsystem | r c1 | interrupt from Uart Receiver |

**Table 6.9: RegIrqHig**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | reserved | 0 resetsystem | r c1 | |
| 6 | reserved | 0 resetsystem | r c1 | |
| 5 | IrqPA[5] | 0 resetsystem | r c1 | interrupt from Pad PA[5] |
| 4 | IrqPA[4] | 0 resetsystem | r c1 | interrupt from Pad PA[4] |
| 3 | IrqPre2 | 0 resetsystem | r c1 | interrupt from prescaler (1 Hz) |
| 2 | IrqVld | 0 resetsystem | r c1 | interrupt from Voltage Level Detector |
| 1 | IrqPA[1] | 0 resetsystem | r c1 | interrupt form Pad PA[1] |
| 0 | IrqPA[0] | 0 resetsystem | r c1 | interrupt form Pad PA[0] |

**Table 6.10: RegIrqMid**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | IrqPA[7] | 0 resetsystem | r c1 | interrupt from Pad PA[7] |
| 6 | IrqPA[6] | 0 resetsystem | r c1 | interrupt from Pad PA[6] |
| 5 | IrqCntB | 0 resetsystem | r c1 | interrupt from CounterB |
| 4 | IrqCntD | 0 resetsystem | r c1 | interrupt from CounterD |
| 3 | IrqPA[3] | 0 resetsystem | r c1 | interrupt from Pad PA[3] |
| 2 | IrqPA[2] | 0 resetsystem | r c1 | interrupt from Pad PA[2] |
| 1 | -- | 0 | r | unused |
| 0 | -- | 0 | r | unused |

**Table 6.11: RegIrqLow**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | IrqEnAc | 0 resetsystem | r w | enable interrupt from acquisition chain (when available) |
| 6 | IrqEnPre1 | 0 resetsystem | r w | enable interrupt from prescaler (128 Hz) |
| 5 | -- | 0 | r | unused |
| 4 | IrqEnCntA | 0 resetsystem | r w | enable interrupt from CounterA |
| 3 | IrqEnCntC | 0 resetsystem | r w | enable interrupt from CounterC |
| 2 | -- | 0 | r | unused |
| 1 | IrqEnUartTx | 0 resetsystem | r w | enable interrupt from Uart Transmitter |
| 0 | IrqEnUartRx | 0 resetsystem | r w | enable interrupt from Uart Receiver |

**Table 6.12: RegIrqEnHig**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | reserved | 0 resetsystem | r w | bit should not be used |
| 6 | reserved | 0 resetsystem | r w | bit should not be used |
| 5 | IrqEnPA[5] | 0 resetsystem | r w | enable interrupt from Pad PA[5] |
| 4 | IrqEnPA[4] | 0 resetsystem | r w | enable interrupt from Pad PA[4] |
| 3 | IrqEnPre2 | 0 resetsystem | r w | enable interrupt from prescaler (1 Hz) |
| 2 | IrqEnVld | 0 resetsystem | r w | enable interrupt from Voltage Level Detector |
| 1 | IrqEnPA[1] | 0 resetsystem | r w | enable interrupt from Pad PA[1] |
| 0 | IrqEnPA[0] | 0 resetsystem | r w | enable interrupt from Pad PA[0] |

**Table 6.13: RegIrqEnMid**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | IrqEnPA[7] | 0 resetsystem | r w | enable interrupt from Pad PA[7] |
| 6 | IrqEnPA[6] | 0 resetsystem | r w | enable interrupt from Pad PA[6] |
| 5 | IrqEnCntB | 0 resetsystem | r w | enable interrupt from CounterB |
| 4 | IrqEnCntD | 0 resetsystem | r w | enable interrupt from CounterD |
| 3 | IrqEnPA[3] | 0 resetsystem | r w | enable interrupt from Pad PA[3] |
| 2 | IrqEnPA[2] | 0 resetsystem | r w | enable interrupt from Pad PA[2] |
| 1 | -- | 0 | r | unused |
| 0 | -- | 0 | r | unused |

**Table 6.14: RegIrqEnLow**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | IrqPriority | HFF resetsystem | r | code of highest priority set interrupt |

**Table 6.15: RegIrqPriority**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-3 | -- | 0 | r | unused |
| 2 | IrqHig | 0 resetsystem | r | one (or more) high priority interrupt is set |
| 1 | IrqMid | 0 resetsystem | r | one (or more) middle priority interrupt is set |
| 0 | IrqLow | 0 resetsystem | r | one (or more) low priority interrupt is set |

**Table 6.16: RegIrqIrq**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | IrqTest | 00000000 resetsystem | wt | for test only |

**Table 6.17: RegIrqTest**

## 6.4 Events

### 6.4.1 Features

The XE8000 support 8 event sources, namely:

- 2 external inputs from PortA
- 4 from the internal Counters A,B,C and D
- 2 from the internal Prescaler (128Hz and 1Hz )

Events are used to restart the CPU after a HALT instruction, without having to handle a complete interrupt cycle.

These events display 2 levels of priority.

### 6.4.2 Overview

The events are divided into 2 categories with different priorities. The priorities are fixed and are described below :

- High: counterA, counterC, prescaler 128Hz & pad PA[1]
- Low: CounterB, CounterD, prescaler 1Hz & pad PA[0]

There is an event flag register associated with each priority. The event flag register is **RegEvn**.

The rising edge of the event (EVN) source sets the event flag if it is enabled.

The event flag is automatically cleared following system Resets and can be cleared by software. This is achieved by writing the corresponding Flag in the **RegEvn** register to 1 (the CoolRISC events are active low). For clearing definitively the event, one has to disable the corresponding bit in the CoolRISC status register. For example if a event is generated by the pad 0 of port A:

```
move                    RegEvn, #0x01
clrb                    stat,#1
```

Each Event (EVN) source can be enabled or disabled by software with the help of the Event enable registers **RegEvnEn**.

### 6.4.3 PortA events

It is possible to use pin PA[0] & PA[1] of PortA as an independent edge triggered EVN input.
Bitwise configuration (debounced or non-debounced / falling or rising edge) is performed using reserved registers in portA.
The debouncer frequency can be either 256Hz (slow debouncer clock) or 8KHz (fast-debouncer clock). The debounce clock is common for all 8 PortA inputs (and also for the RESET pad).
When debounced, the signal on portA has to be stable for one to two periods of selected internal debounce clock to ensure that the new logical value is accepted.

### 6.4.4 Counters A, B, C and D events

Counters generate Events only when they work as normal counters (not when set to PWM).

Loop Counters generate events regularly, depending on their mode of operation (up/down counting). It is for the software programmer to handle the processing of events.

### 6.4.5 Prescaler events

Two events are available from the Prescaler, 128Hz and 1Hz.

Both are generated by the low 15-stage divider.This means that they can be accurate at 128Hz and 1Hz only when the Xtal 32768 Quartz oscillator is enabled. If the Xtal is not enabled, they are derived from the RC oscillator frequency.

The Prescaler can be resynchronized at any time by resetting it.

### 6.4.6 Event priorities

Two registers have been provided to facilitate the writing of software used for handling events, namely :

- **RegEvnPriority**
- **RegEvnEvn**

The first, **RegEvnPriority,** contains the ID-code of the highest activated Priority event. The table below shows these priorities which correspond to organization of event in the registers above and their ID-code.

| RegEvnPriority | Event priority (on CPU) | highest priority event set |
|---|---|---|
| HFF | none | no event |
| H07 | EV0 (high) | EvnCntA |
| H06 | EV0 (high) | EvnCntC |
| H05 | EV0 (high) | EvnPre1 |
| H04 | EV0 (high) | EvnPA[1] |
| H03 | EV1 (low) | EvnCntB |
| H02 | EV1 (low) | EvnCntD |
| H01 | EV1 (low) | EvnPre2 |
| H00 | EV1 (low) | EvnPA[0] |

**Table 6.18: all events and their priorities**

After any system Reset the **RegEvnPriority** value is HFF indicating no event request has been issued subsequent to the Reset.

The second, **RegEvnEvn,** indicates the priority levels of the current events.

The two levels of events map directly to those supported by the CoolRISC (EV0 & EV1).

For more information, see the documentation about the CoolRISC816 core.

| register name | address (hex) |
|---|---|
| RegEvn | H003C |
| RegEvnEn | H003D |
| RegEvnPriority | H003E |
| RegEvnEvn | H003F |

**Table 6.19: Event registers**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | EvnCntA | 0 resetsystem | rc1 | event from Counter A |
| 6 | EvnCntC | 0 resetsystem | rc1 | event from Counter C |
| 5 | EvnPre1 | 0 resetsystem | rc1 | event from prescaler (128 Hz) |
| 4 | EvnPA[1] | 0 resetsystem | rc1 | event from Pad PA[1] |
| 3 | EvnCntB | 0 resetsystem | rc1 | event from Counter B |
| 2 | EvnCntD | 0 resetsystem | rc1 | event from Counter D |
| 1 | EvnPre2 | 0 resetsystem | rc1 | event from prescaler (1 Hz) |
| 0 | EvnPA[0] | 0 resetsystem | rc1 | event from Pad PA[0] |

**Table 6.20: RegEvn**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | EvnEnCntA | 0 resetsystem | rw | enable event from CounterA |
| 6 | EvnEnCntC | 0 resetsystem | rw | enable event from CounterC |
| 5 | IrqREnPre1 | 0 resetsystem | rw | enable event from prescaler (128 Hz) |
| 4 | EvnEnPA[1] | 0 resetsystem | rw | enable event from Pad PA[1] |
| 3 | EvnEnCntB | 0 resetsystem | rw | enable event from CounterB |
| 2 | EvnEnCntD | 0 resetsystem | rw | enable event from CounterD |
| 1 | IrqREnPre1 | 0 resetsystem | rw | enable event from prescaler (1 Hz) |
| 0 | EvnEnPA[0] | 0 resetsystem | rw | enable event from Pad PA[0] |

**Table 6.21: RegEvnEn**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | EvnPriority | HFF resetsystem | r | code of highest priority set interrupt |

**Table 6.22: RegEvnPriority**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-2 | -- | 000000 | r | unused |
| 1 | EvnHig | 0 resetsystem | r | one (or more) high priority event is set |
| 0 | EvnLow | 0 resetsystem | r | one (or more) low priority event is set |

**Table 6.23: RegEvnEvn**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | EvnTest | 00000000 resetsystem | wt | for test only |

**Table 6.24: RegEvnTest**

## 6.5 Miscellaneous

### 6.5.1 Port configuration reset

Port configuration (pins in input or output mode) is controlled by registers described in the Ports chapters. These registers are initialized by signal ResetPConf that the user can configure as active or inactive during system reset with bit **EnResPConf** in register **RegSysCtrl**. The user can also choose if these registers are initialized for each reset or only on power-on reset.

| BitEnResPConf | Port config reset |
|---|---|
| 1 | reset with any reset source |
| 0 | reset with resetcold only (power-on reset) |

**Table 6.25: Port config reset**

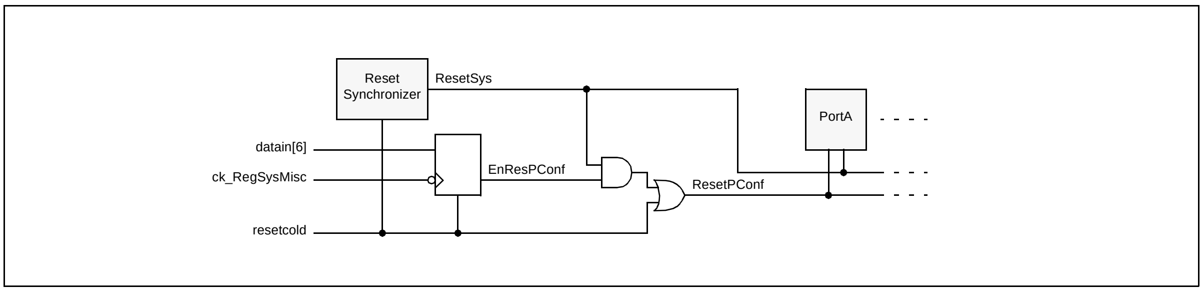


Figure 6.8: Port configuration reset

### 6.5.2 Prescaler interrupt synchronization

Prescaler generates two interrupts at frequencies 128 Hz and 1 Hz. Interrupt signal is generated on negative slope so that 1Hz interrupt is generated one second after prescaler initialization, and then subsequently each second.

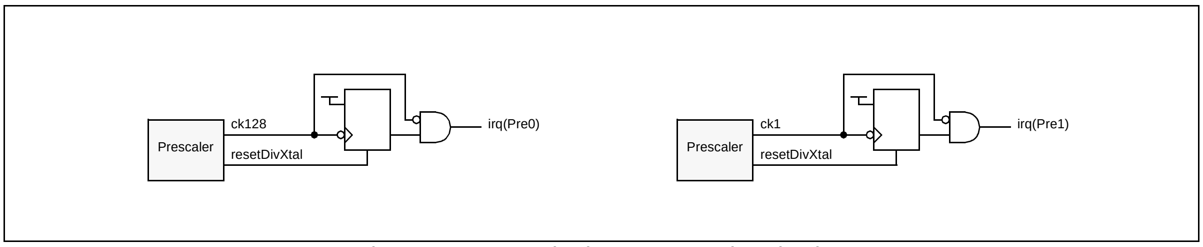


Figure 6.9: Prescaler interrupt synchronization

## 6.6 Digital debouncer

A digital debouncer can be used to filter input pins of PortA. Period of the debouncer is set by bit **DebFast** in register **RegSysMisc**.

### 6.6.1 Description

At power-on, signal resetcold is activated by power-on reset block. **DebFast** is initialized at 0 (ckdeb = 4 ms) digital filter is initialized at 0 (not active).

Debouncer period can be set to 125 us or 4 ms with bit **DebFast**.

| DebFast | debouncer period |
|---|---|
| 1 | 125 us |
| 0 | 3.9 ms |

**Table 6.26: Debouncer frequency**

Input signal must be stable during the complete debouncer period to be transmitted.

**Note:** Debouncer clock is taken from the prescaler, and so may vary with actual RC frequency.
**Note:** Debouncer delay varies with the respective phases of the input signal and the debouncer clock.

## 6.7 System peripheral addresses

| register | addresses |
|---|---|
| RegSysCtrl | h0010 |
| RegSysReset | h0011 |
| RegSysClock | h0012 |
| RegSysMisc | h0013 |
| RegSysWD | h0014 |
| RegSysPre0 | h0015 |
| RegSysPre1 | h0016 |
| reserved | h0017 |
| reserved | h0018 |
| reserved | h0019 |
| reserved | h001A |
| RegSysRCTrim1 | h001B |
| RegSysRCTrim2 | h001C |
| RegSysVIreg | h001D |
| reserved | h001E |
| RegSysWarm | h001F |

**Table 6.27: system address ranges**

# 7 Oscillators

There are two oscillators: one fast programmable RC oscillator, and a precise crystal oscillator. Registers for controlling the oscillators are described in the SYSTEM chapter.

## 7.1 RC Oscillator

### 7.1.1 RC oscillator principle

The RC Oscillator is always turned on at power-on reset and can be turned off after the optional Xtal oscillator has been started. The RC oscillator has two frequency ranges: sub-MHz (100KHz to 1MHz) and above-MHz (1MHz to 10MHz). Inside a range, the frequency can be tuned by software for coarse and fine adjustment.

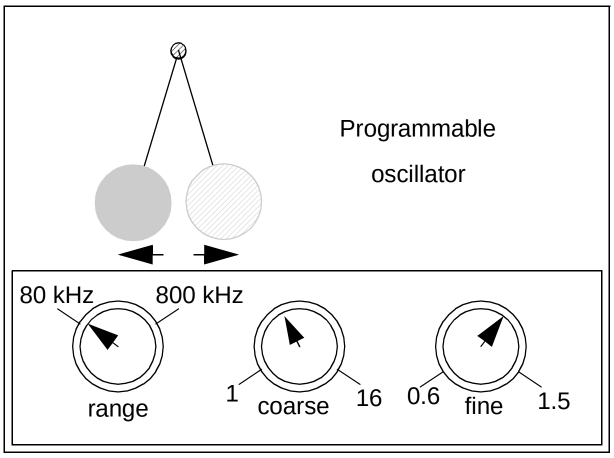


Figure 7.1: RC programming principle

No external components are required as both the resistor and the capacitor are on chip.

The RC oscillator can be in 3 modes. In mode 1(RC on), the RC oscillator and its bias are on. In mode 2 (RC ready), the RC oscillator is off and the bias is on. In mode 3 (RC off), the RC oscillator and the bias are off. RC ready mode is a compromise between power consumption and start-up time.

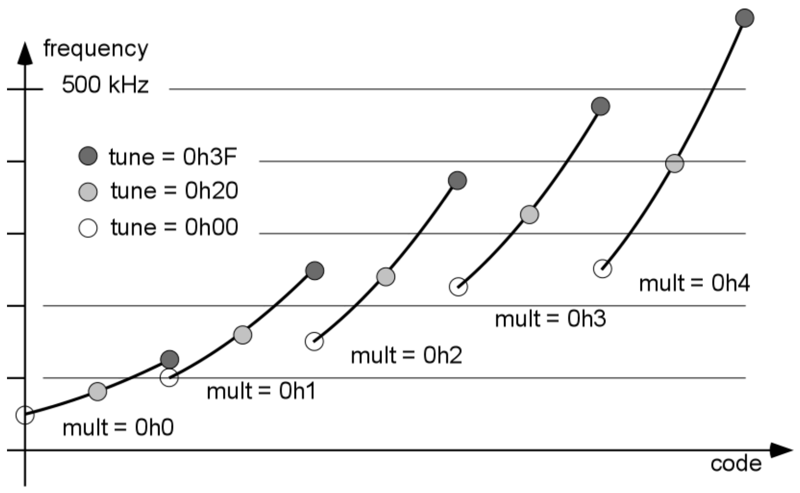


Figure 7.2: RC frequencies programming example for low range (typical values)

| symbol | description | min | typ | max | unit | comments |
|---|---|---|---|---|---|---|
| $F_{st}$ | frequency at start-up | 50 | 80 | 110 | kHz | |
| range | range selection | 1 | | 10 | | multiplies $F_{st}$ |
| mult[3:0] | coarse tuning range | 1 | | 16 | | 4 bits, multiplies $F_{st}$ * range |
| tune[5:0] | fine tuning range | 0.65 | | 1.5 | | 6 bits, multiplies $F_{st}$ * range * mult |
| | fine tuning step | | 1.4 | 2 | % | |
| $T_{st}$ | start-up time | | 30 | 50 | µs | bias current is off (RC off) |
| $O_{st}$ | overshoot at start-up | | | 50 | % | bias current is off (RC off) |
| $T_{wu}$ | wakeup time | | 3 | 5 | µs | bias current is on (RC ready) |
| $O_{wu}$ | overshoot at wakeup | | | 50 | % | bias current is on (RC ready) |
| jit | jitter rms | | 2 | | $^{o}/_{oo}$ | |

**Table 7.1: RC specifications**

### 7.1.2 RC divider cold start

During RC oscillator start-up, the coldstart structure masks the first system clock cycles. Two bits **EnableRC** and **ColdRC** in register **RegSysClock** control this coldstart structure.

At power-on, resetcold signal initialize **EnableRC** and **ColdRC** to 1 (selects RC oscillator). RC oscillator starts and some cycles later bit **ColdRC** is reset and the clock is made available to the system after one additionnal clock cycle delay.

RC oscillator can be stopped by the user by resetting bit **EnableRC**. In this case, bit **ColdRC** goes immediately to 1. By setting bit **EnableRC**, one restarts the RC oscillator. Eight cycles later, bit **ColdRC** is reset and clock is made available to the system after one additionnal clock cycle delay.

When user sets sleep mode, signal **Sleep** is set together with bits **EnableRC** and **ColdRC**. Signal rcPower is masked to maintain oscillator stopped. When coming out of sleep mode, usual start-up happens.

RC oscillator starts on Port A event if **ResOnPA0** is set in **RegSysMisc**.

**Note:** Prescaler is initialized by signal **BitSleep**.

**Note:** RC oscillator bias is set by bit **BitBiasRC** in **RegSysClock**.

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-6 | -- | 00 | r | unused |
| 5 | EnRCRext | 0 resetcold | rw | enable external resistor for trimming |
| 4 | RCFreqRange | 0 resetcold | rw | low- / high-frequency RC (0=low) |
| 3 | RCFreqCoarse[3] | 0 resetcold | rw | RC frequency coarse trimming bit 3 |
| 2 | RCFreqCoarse[2] | 0 resetcold | rw | RC frequency coarse trimming bit 2 |
| 1 | RCFreqCoarse[1] | 0 resetcold | rw | RC frequency coarse trimming bit 1 |
| 0 | RCFreqCoarse[0] | 0 resetcold | rw | RC frequency coarse trimming bit 0 |

**Table 7.2: RegSysRCTrim1**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-6 | -- | 00 | r | unused |
| 5 | RCFreqFine[5] | 1 resetcold | rw | RC frequency fine trimming bit 5 |
| 4 | RCFreqFine[4] | 0 resetcold | rw | RC frequency fine trimming bit 4 |
| 3 | RCFreqFine[3] | 0 resetcold | rw | RC frequency fine trimming bit 3 |
| 2 | RCFreqFine[2] | 0 resetcold | rw | RC frequency fine trimming bit 2 |
| 1 | RCFreqFine[1] | 0 resetcold | rw | RC frequency fine trimming bit 1 |
| 0 | RCFreqFine[0] | 0 resetcold | rw | RC frequency fine trimming bit 0 |

**Table 7.3: RegSysRCTrim2**

## 7.2 Xtal Oscillator

### 7.2.1 General description

The Xtal Oscillator operates with an external crystal of 32’768 Hz.

| symbol | description | min | typ | max | unit | comments |
|---|---|---|---|---|---|---|
| f_clk32k | nominal frequency | | 32768 | | Hz | |
| st_x32k | oscillator start-up time | | 1 | 2 | s | |
| duty_clk32k | duty cycle on the digital output | 30 | 50 | 70 | % | |
| fstab_1 | relative frequency deviation from nominal, for a crystal with CL=8.2 pF and temperature between -40° and +85°C | -100 | | +300 | ppm | not included: crystal frequency tolerance and aging crystal frequency - temperature dependence |

**Table 7.4: Xtal oscillator specifications.**

**Note:** Board layout recommendations for safer crystal oscillation and lower current consumption:
- Keep lines xtal_in and xtal_out short and insert a VSS line between them.
- Connect package of the crystal to VSS.
- No noisy or digital lines near xtal_in and xtal_out.
- Insert guards at VSS where needed.

### 7.2.2 Typical external component

The crystal oscillator needs a 32’768 Hz crystal with specifications as in Table 7.5 (normal watch crystal). Attention has to be made when designing the board, especially to the parameters given in Table 7.6.

| symbol | description | min | typ | max | unit | comments |
|---|---|---|---|---|---|---|
| Fs | Resonance frequency | | 32768 | | Hz | |
| CL | CL for nominal frequency | | 8.2 | | 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 | |
| Q | quality factor | 40k | 80k | 400k | - | |

**Table 7.5: External crystal specifications. 32 kHz Xtal outside these specifications will must probably delivers correct frequency, but some precision specifications will be released.**

| symbol | description | min | typ | max | unit |
|---|---|---|---|---|---|
| rh_xin | external board parasitic resistance xtal_in - VSS | 10 | | | MΩ |
| rh_xout | external board parasitic resistance xtal_out - VSS | 10 | | | MΩ |
| rh_xin_xout | external board parasitic resistance xtal_in - xtal_out | 50 | | | MΩ |
| cp_xin | external board parasitic capacitance on xtal_in - VSS | 0.5 | | 3.0 | pF |
| cp_xout | external board parasitic capacitance on xtal_out - VSS | 0.5 | | 3.0 | pF |
| cp_xin-xout | external board parasitic capacitance xtal_in - xtal_out | 0.2 | | 1.0 | pF |

**Table 7.6: Board design specifications**

### 7.2.3 Xtal divider cold start

During Xtal oscillator start-up, a cold start structure masks the first 32768 clock cycles. The two bits **EnableXtal** and **BitColdXtal** in register **RegSysClock** control this structure.

### 7.2.4 Description

At power-on, signal resetcold resets **EnableXtal** and sets **ColdXtal** (Xtal oscillator is not selected at start-up).

Xtal oscillator can be started by the user by setting **EnableXtal**. Bit **EnExtClk** should be reset first to disable external clock detection. If an external clock is already detected, Xtal oscillator can not be started anymore (blocked by **ExtClk**). When Xtal oscillator starts, bit **ColdXtal** is reset after 32768 cycles and clock is made available to the system. Xtal oscillator can be stopped by user by resetting bit **EnableXtal**. In this case, bit **ColdXtal** is immediately set.

When the user chooses sleep mode, signal **Sleep** going to 1 resets **EnableXtal** and sets **ColdXtal**. When going out of sleep mode the oscillator is stopped until being started by the user.

**Note:** Prescaler is initialised by signal **Sleep**.

## 7.3 External clock

An external clock can be provided by the user instead of the Xtal oscillator. The clock is input through the pin OSCIN. If the external clock is present, it is detected after 4 cycles (ExtClk set) and the Xtal oscillator is disabled (not possible to set **EnableXtal**).

**Note:** After power-up, the external clock detection is disabled (**EnExtClk** is off). If the user wants to use the external oscillator, the bit **EnExtClk** has to be set to 1.
**Note:** **ExtClk** can be reset by power-on reset only.

## 7.4 Oscillators control

Both RC and Xtal oscillators can be controlled by changing the states of selected bits with **RegSysClock**. After power-on reset, only the RC oscillator is running.

### 7.4.1 CPU clock

As there are three different clock sources available for the CPU clock, it is possible to select one of them. The RC clock is always selected after power-up or after Sleep mode. The CPU clock selection is done with the bit **CpuSel** in **RegSysClock** (0=RC clock, 1 = Xtal clock or external clock).

**Note:** If a clock is selected but its oscillator is in cold start phase, the CPU clock is stopped during that time.

**Note:** Switching from RC clock to Xtal clock and stopping the RC must be performed using 2 MOVE instructions to **RegSysClock**. First select the Xtal clock and then stop the RC. When switching form Xtal to RC clock, first select the RC clock and then stop the Xtal.

**Note:** If only one clock is active (RC or Xtal), that clock is selected per default.

At power-on, or when going to sleep mode, signal resetsleep (resetcold OR BitSleep) resets **CpuSel**, RC oscillator is chosen.

| CpuSel | ColdRC | ColdXtal | BitExtClk | sel | ckcpu |
|---|---|---|---|---|---|
| X | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | X | X | 0 | ckRC |
| 1 | 0 | 1 | 0 | 0 | ckRC |
| 0 | 1 | 0 | X | 1 | ckXtal |
| 0 | 1 | 1 | 1 | 1 | extClk |
| 1 | X | 0 | X | 1 | ckXtal |
| 1 | X | 1 | 1 | 1 | extClk |

**Table 7.7: CPU clock selection**

**Note:** Selection of clock is made without glitches thanks to a specific block (see Figure 7.3).

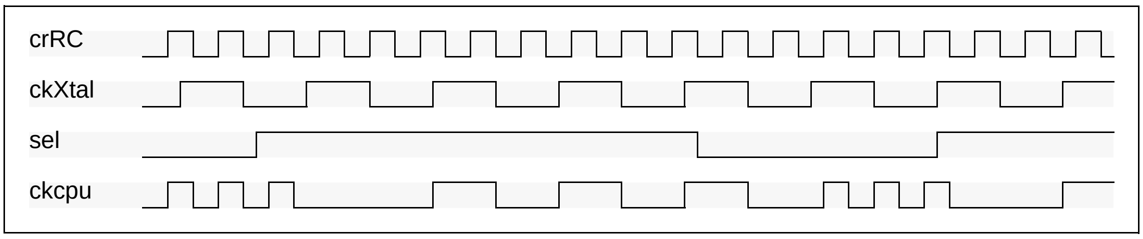


Figure 7.3: CPU clock selection

### 7.4.2 Oscillator register

- **BiasRC** enables the RC oscillator bias current for a fast start-up
- **ColdXtal** this bit is set during the cold start of the Xtal oscillator. The delay is 32'768 clock cycles
- **ColdRC** this bit is set during the cold start of the RC oscillator. The delay is 8 clock cycles
- **EnableXtal** enables the Xtal oscillator. This bit is set after power-on.
- **EnableRC** enables the RC oscillator

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | CpuSel | 0 resetsleep | r w | CkCpu selection (0=RC) |
| 6 | ExtClk | 0 resetcold | r | external clock detected |
| 5 | EnExtClk | 0 resetcold | r w | enable Oscin to be driven from ext. |
| 4 | BiasRC | 1 resetcold | r w | enable RC bias when EnableRC = 0 |
| 3 | ColdXtal | 1 resetsleep | r | Xtal in starting phase for 32768 cycles |
| 2 | ColdRC | 1 resetsleep | r | RC in starting phase for 8 cycles |
| 1 | EnableXtal | 0 resetsleep | r w | enable Xtal oscillator (cannot be set if EnExtClk=1 or ExtClk=1) |
| 0 | EnableRC | 1 resetsleep | r w | enable RC oscillator |

**Table 7.8: RegSysClock, address h0012**

**Note:** If the **ExtClk** bit is one, then the **EnExtClk** bit must not be set to 0.

## 7.5 Prescaler

The prescaler is a divider chain providing frequencies down to 1Hz based on the RC oscillator or based on the Xtal oscillator. Two registers **RegSysPre0** and **RegSysPre1** control the prescaler.

The prescaler is made of one asynchronous 8 stage divider and one asynchronous 15 stage divider, all chained, and of one initialization structure.

### 7.5.1 Features

- basic 15 stage divider in case of 32768Hz -> 1Hz
- low 15 stages can be reset with a reset prescaler command
- all 15 divided clocks are outputs which can be used in other blocks like timer/counter, debouncer, EOL logic, Oscillation detector, WD, used outputs from 15 stage dividers are 32kHz, 1024Hz, 256Hz, 128Hz, 1Hz
- RC oscillator itself has a divider by 32 to accept 1MHz clock and output for other peripherals 1MHz, 250 kHz and 31 kHz clocks
- Prescaler as divider chain and clock selector can be tested separately. (this means independent of other blocks)
- with test register we can read state of the divider chain after each 4 dividers, (after 2 dividers for hi-frequency RC clock)
- Prescaler has a Reset input to reset the whole divider chain. This can be the sum of POR and Reset pad

### 7.5.2 Description

The 8 stages divider is directly connected to the RC oscillator. The 15 stages dividers is clocked by the Xtal oscillator when it is on, or by a frequency near to 32 kHz taken from the first divider based on the RC oscillator trimming. See examples below (Figure 7.4, Table 7.9). Additionnal circuitry, not shown on the figure, prevents unwanted transitions on ck32k when switching the RC oscillator frequency.

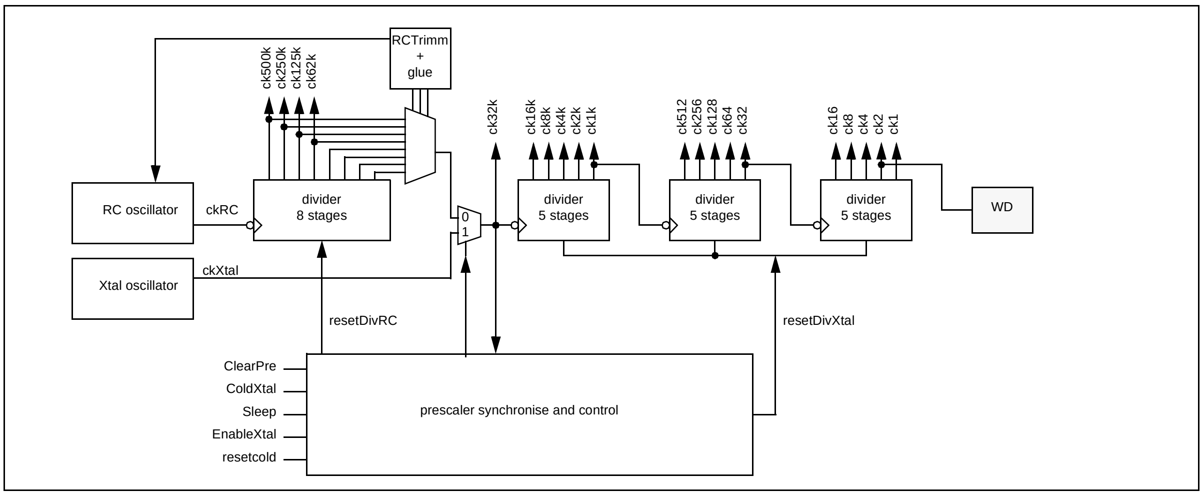


Figure 7.4: prescaler principle

| freq name | clocks | RC@4MHz, Xtal off | RC@2.5MHz, Xtal off | RC@1MHz, Xtal off | RC@1MHz, Xtal on | RC off, Xtal on |
|---|---|---|---|---|---|---|
| coarse (3:0) | | 0100b | | | | N/A |
| fine (5:0) | | 100001b | | | | N/A |
| ckRC | 20 | 4’000’000 | 2’457’600 | 1’040’000 | 1’040’000 | OFF |
| ck500k | 19 | 2’000’000 | 1’228’800 | 520’000 | 520’000 | OFF |
| ck250k | 18 | 1’000’000 | 614’400 | 260’000 | 260’000 | OFF |
| ck125k | 17 | 500’000 | 307’200 | 130’000 | 130’000 | OFF |
| ck62k | 16 | 250’000 | 153’600 | 65’000 | 65’000 | OFF |
| ckXtal | | OFF | OFF | OFF | 32’768 | 32’768 |
| ck64k to ck32k divider | | 8 | 4 | 2 | N/A | N/A |
| ck32k | 15 | 31’250 | 38’400 | 32’500 | 32’768 | 32’768 |
| ck16k | 14 | 15’625 | 19’200 | 16’250 | 16’384 | 16’384 |
| ck8k | 13 | 7813 | 9600 | 8125 | 8192 | 8192 |
| ck4k | 12 | 3’906 | 4800 | 4062 | 4096 | 4096 |
| ck2k | 11 | 1’953 | 2400 | 2031 | 2048 | 2048 |
| ck1k | 10 | 977 | 1200 | 1016 | 1024 | 1024 |
| ck512 | 9 | 488 | 600 | 508 | 512 | 512 |
| ck256 | 8 | 244 | 300 | 254 | 256 | 256 |
| ck128 | 7 | 122 | 150 | 127 | 128 | 128 |
| ck64 | 6 | 61 | 75 | 63.5 | 64 | 64 |
| ck32 | 5 | 31 | 37.5 | 31.7 | 32 | 32 |
| ck16 | 4 | 15.3 | 18.8 | 15.9 | 16 | 16 |
| ck8 | 3 | 7.63 | 9.38 | 7.93 | 8 | 8 |
| ck4 | 2 | 3.81 | 4.69 | 3.97 | 4 | 4 |
| ck2 | 1 | 1.91 | 2.34 | 1.98 | 2 | 2 |
| ck1 | 0 | 0.95 | 1.17 | 0.992 | 1 | 1 |
| watchdog | -- | 4.19 seconds | 3.41 seconds | 4.03 seconds | 4 seconds | 4 seconds |

**Table 7.9: frequency examples, typical values for RC setting, range is set to 1**

Signal resetcold initializes all dividers at power-on.

The 8-bits divider and the remaining 5-bits dividers use different initialization signals. The 3 low frequency dividers can be reset simultaneously by writing bXXXXXXX1 in register **RegSysPre0**. These dividers are also initialized at

Xtal oscillator start (writing bXXXXXXX1X in register **RegSysClock**). Initialization signal is synchronized to the low frequency clock in order to ensure correct divider initialization.

**Note:** Dividers are initialised in mode ‘Sleep’.

**Note:** Low frequency part of prescaler is not initialised by the reset Synchronizer (ClearPre) when Xtal oscillator is in coldstart so that the coldstart delay is not disturbed.

**Note:** Low frequency part of prescaler is always set after the high frequency part (connected to RC) when external clock is selected.

| EnXtal | RC Trimming | | RC frequency | | | division factor | Low Prescaler input freq | | |
|---|---|---|---|---|---|---|---|---|---|
| | Range | Coarse | RegSysRC-Trim2 = 00 | RegSysRC-Trim2 = 20 | RegSysRC-Trim2 = 3F | | RegSysRC-Trim2 = 00 | RegSysRC-Trim2 = 20 | RegSysRC-Trim2 = 3F |
| 1 | X | XXXX | X | X | X | X | | 32768 | |
| 0 | 0 | 0000 | 56560 | 80000 | 112800 | 2 | 28280 | 40000 | 56400 |
| 0 | 0 | 0001 | 113120 | 160000 | 225600 | 4 | 28280 | 40000 | 56400 |
| 0 | 0 | 0010 | 169680 | 240000 | 338400 | 8 | 21210 | 30000 | 42300 |
| 0 | 0 | 0011 | 226240 | 320000 | 451200 | 8 | 28280 | 40000 | 56400 |
| 0 | 0 | 0100 | 282800 | 400000 | 564000 | 16 | 17675 | 25000 | 35250 |
| 0 | 0 | 0101 | 339360 | 480000 | 676800 | 16 | 21210 | 30000 | 42300 |
| 0 | 0 | 0110 | 395920 | 560000 | 789600 | 16 | 24745 | 35000 | 49350 |
| 0 | 0 | 0111 | 452480 | 640000 | 902400 | 16 | 28280 | 40000 | 56400 |
| 0 | 0 | 1000 | 509040 | 720000 | 1015200 | 16 | 31815 | 45000 | 63450 |
| 0 | 0 | 1001 | 565600 | 800000 | 1128000 | 32 | 17675 | 25000 | 35250 |
| 0 | 0 | 1010 | 622160 | 880000 | 1240800 | 32 | 19443 | 27500 | 38775 |
| 0 | 0 | 1011 | 678720 | 960000 | 1353600 | 32 | 21210 | 30000 | 42300 |
| 0 | 0 | 1100 | 735280 | 1040000 | 1466400 | 32 | 22978 | 32500 | 45825 |
| 0 | 0 | 1101 | 791840 | 1120000 | 1579200 | 32 | 24745 | 35000 | 49350 |
| 0 | 0 | 1110 | 848400 | 1200000 | 1692000 | 32 | 26513 | 37500 | 52875 |
| 0 | 0 | 1111 | 904960 | 1280000 | 1804800 | 32 | 28280 | 40000 | 56400 |
| 0 | 1 | 0000 | 565600 | 800000 | 1128000 | 32 | 17675 | 25000 | 35250 |
| 0 | 1 | 0001 | 1131200 | 1600000 | *2256000* | 64 | 17675 | 25000 | 35250 |
| 0 | 1 | 0010 | 1696800 | *2400000* | *3384000* | 64 | 26513 | 37500 | 52875 |
| 0 | 1 | 0011 | *2262400* | *3200000* | *4512000* | 128 | 17675 | 25000 | *35250* |
| 0 | 1 | 0100 | *2828000* | *4000000* | *5640000* | 128 | 22094 | 31250 | *44063* |
| 0 | 1 | 0101 | 3393600 | *4800000* | *6768000* | 128 | 26513 | *37500* | *52875* |
| 0 | 1 | 0110 | 3959200 | *5600000* | *7896000* | 128 | 30931 | *43750* | *61688* |

**Table 7.10: automatic input frequency selection, typical values. Values in *italic* are not allowed and may result in unpredictable CPU behaviour.**

**Note:** UNLESS ck32k IS TAKEN FROM THE Xtal OSCILLATOR, ITS FREQUENCY CAN SIGNIFICANTLY DIFFER FROM ITS NOMINAL VALUE.

### 7.5.3 Registers

The **RegSysPre0** controls some of the prescaler functions. The bits within the registers define the following:

- **CpuClk** internal CPU clock in internal test mode only
- **ResPre** When 1 is written at this address only the low 15 stage “Xtal” divider is reset to 0. This reset takes one Xtal clock period

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-2 | -- | 000000 | r | unused |
| 1 | CpuClk | 0 | r | CpuClk in test mode |
| 0 | ResPre | 0 resetcold | w | reset the Xtal Prescaler when written to 1 |

**Table 7.11: RegSysPre0**

# 8 Parallel IO ports

## 8.1 Port A

### 8.1.1 Features

- input port, 8 bits wide
- RegPAin read the input status after the debouncer option
- each bit can be set individually for debounced or direct input
- each bit can be set individually for pullup or not
- each bit is an interrupt request source and can be set on rising or falling edge
- a system reset can be generated on a port configuration, each bit as direct input, inverted input, zero or one
- 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

### 8.1.2 Overview

Port A is a general purpose 8 bit wide input port, with interrupt capability. It can be debounced or not.

Internal Pull-Up resistors can be connected to its inputs. It can be used to connect external clock sources for the internal counters (also event counter capability). Rising or falling edge for interrupt generation can be selected, PA[0] and PA[1] can generate events for the CPU.

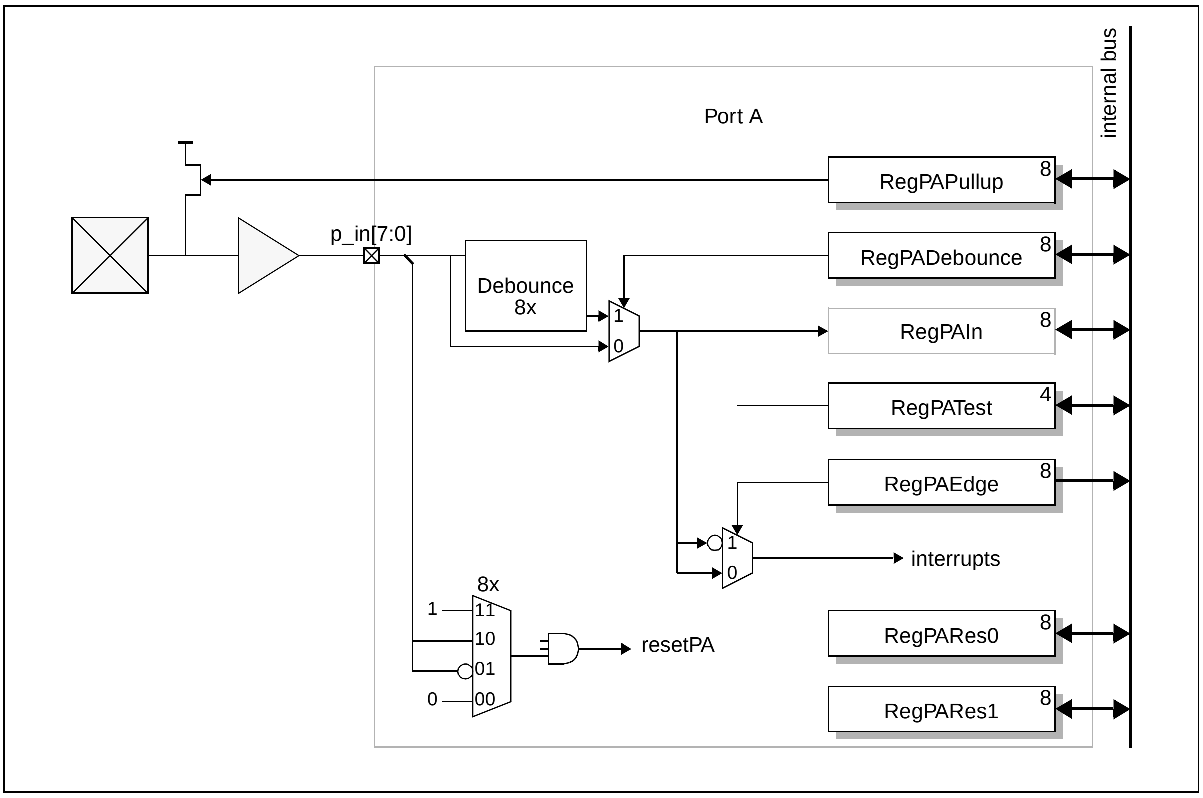


Figure 8.1: Port A

### 8.1.3 Port A configuration

The PortA input status can be read from **RegPAin**. Each bit of PortA can be set individually to be a debounced or a direct input using register **RegPADebounce** (Input is debounced if 1 is written in). Following reset, this register is 0.

**Note:** Depending on the status of an **EnResPConf** bit in **RegSysCtrl**, **RegPADebounce** can be reset by any of the possible system resets or only with power-on reset (POR).

Each PortA input is an interrupt request source and can be set on rising or falling edge with a RegPAEdge. Following reset, the rising edge is selected for Interrupt generation by default.

**Note:** Care must be taken when modifying **RegPAEdge** because this register performs a selection, the dynamic result of which may be interpreted incorrectly as a transition. It is therefore better to remove the corresponding PortA IrqMask in **RegIrqEnMid** and **RegIrqEnLow** when changing **RegPAEdge**. Interrupt flag is set only when its corresponding IrqMask bit is set to 1.

Each bit can be set individually for pullup or not using Register **RegPAPullup**. Input is Pulled-Up when its corresponding bit in this register is set to 1.

Default status after reset is 0 what means no Pull_Up.

**Note:** Depending on the status of an **EnResPConf** bit in **RegSysCtrl**, **RegPAEdge** can be reset by any of the possible system resets or only with PowerOnReset (POR). See Table 8.1.

PortA can be used to generate a system reset by placing a predetermined word on PortA externally. The precise code that will cause a system reset can be set with the aid of table 8.1.

| PARes1[x] | PARes0[x] | PARes[x] |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | NOT PA[x] |
| 0 | 1 | PA[x] |
| 0 | 0 | 0 |

**Table 8.1: reset selection for each pin**

Two registers **RegPARes0** and **RegPARes1** (bit as direct input, inverted input, zero or one) can select one of four possibilities shown in this table.
The Logical AND combination of all 8 input lines with their selection (shown in the column PARes[x]) can give a resetportA as one of system reset sources.
resetportA = PARes[7] AND PARes[6] AND PARes[5] AND ... AND PARes[0]

**Note:** a reset via Port A can be inhibited by placing a 0 on both **PARes1[x]** and **PARes0[x]** (for a particular x).

PA[0] to PA[3] can be used as clock input for the counters, A, B, C & D respectively. In this case counters can be used as event counters of PA[0]..PA[3].

| PortA input | external clock for counter |
|---|---|
| PA[0] | CounterA |
| PA[1] | CounterB |
| PA[2] | CounterC |
| PA[3] | CounterD |

**Table 8.2: clock inputs for counters**

Counter inputs can be debounced or not, depending on **RegPADebounce**. Additionally, the counting edge (falling or Rising) can be selected using RegPAEdge.

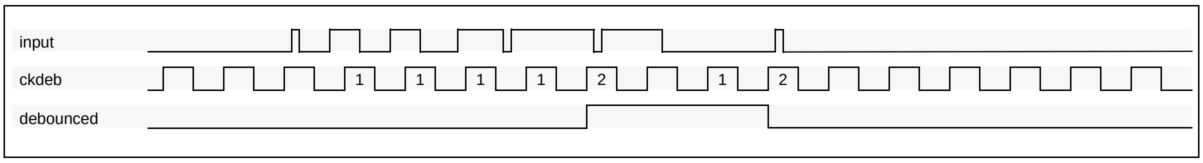


Figure 8.2: digital debouncer

### 8.1.4 PortA registers

| register name | address (hex) |
|---|---|
| RegPAIn | H0020 |
| RegPADebounce | H0021 |
| RegPAEdge | H0022 |
| RegPAPullup | H0023 |
| RegPARes0 | H0024 |
| RegPARes1 | H0025 |
| RegPATest | H0027 |

**Table 8.3: Port A registers**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | PAIn[7-0] | xxxxxxxx | r | pad PA[7-0] input status |

**Table 8.4: Register RegPAIn**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | PADeb[7] | 00000000 resetpconf | rw | debounce for PA[7-0] (1=debounce) |

**Table 8.5: Register RegPADebounce**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | PAEdge[7-0] | 00000000 resetsystem | rw | edge selection for IrqPA[7-0] (0=rise) |

**Table 8.6: Register RegPAEdge**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | PAPullup[7-0] | 00000000 resetpconf | rw | pullup for pad PA[7-0] (1=active) |

**Table 8.7: Register RegPAPullup**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | PARes0[7-0] | 00000000 resetsynch | rw | reset selection bit 0 for pad PA[7-0] |

**Table 8.8: RegPARes0**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | PARes1[7-0] | 0 resetsynch | rw | reset selection bit 1 for pad PA[7-0] |

**Table 8.9: RegPARes1**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-4 | -- | 0000 | r | unused |
| 3 | PATest[3] | 0 resetsystem | r | resetportA |
| 2 | PATest[2] | 0 resetsystem | rw | PATest = Irqbus in test mode |
| 1 | PATest[1] | 0 resetsystem | rw | 8 bits of Irqbus in test mode |
| 0 | PATest[0] | 0 resetsystem | rw | Irqbus = 8 x PATest1 in test mode |

**Table 8.10: RegPATest**

## 8.2 Port B

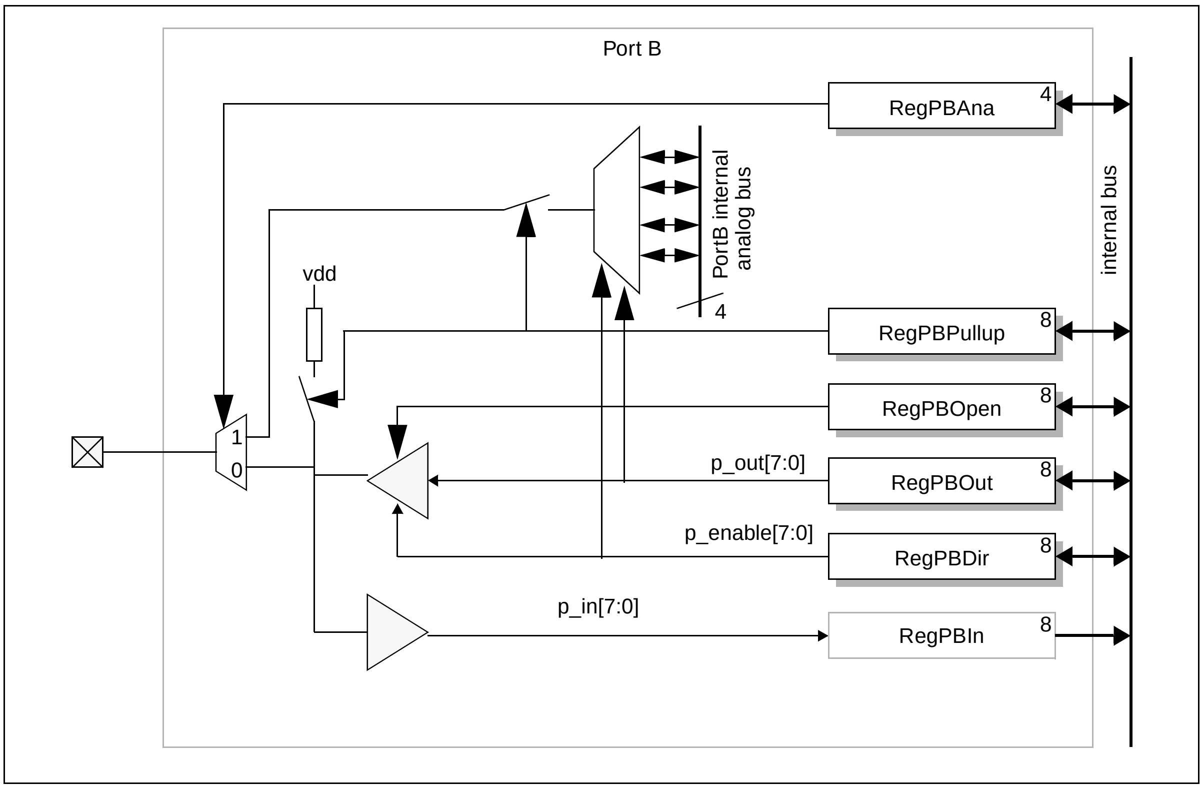


Figure 8.3: Port B

### 8.2.1 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 pullup or not
- four pairs of pads which can be connected individually to four internal analog lines (4 line analog bus)
- pullup is not active when corresponding pad is set to zero
- two PWM can be output on pads PB[0] and PB[1]
- two internal freq (Xtal and cpu) can be output on PB[2] and PB[3]
- the synchronous serial interface uses pads PB[5], PB[4] and PB[3] for SIN, SCL and SOU
- PB[0] is used in test mode for cpu test output
- the UART interface uses pads PB[6] and PB[7] for Tx and Rx

### 8.2.2 Overview

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

### 8.2.3 Port B digital capabilities

When used as logic outputs, each can be individually set as an N-channel open-drain or a push-pull (conventional) output.
Each pin can be individually set to use internal pull-ups.

Data is stored in **RegPBOut** prior to output at PortB.This occurs provided that **PBDir[x]** has been set high (1), accordingly. Its default following reset is low (0).

The status of PortB is available in **RegPBIn** (read only). Reading is always direct - there is no debounce function associated with PortB. In case of possible noise on input signals, a software debouncer with pooling must be realized.

The direction of each bit within PortB (input or output) can be individually set using the **RegPBDir** register. If **PBDir[x]** = 1, the corresponding PortB pin becomes an output. Following reset PortB is in input mode (**PBDir[x]** are reset to 0).

When a pin is in output mode (its **PBDir[x]** is set to 1), the output can be conventional CMOS (Push-Pull) or N-channel Open-drain. driving the output only Low.
By default, after POR 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. When the output should be high the pin becomes high Impedance. The internal pull-up or external pull-up resistor can be used to drive to pin high if required.
**Note:** Because the P transistor actually exists (this not a real Open-drain output) the pull-up range is limited to avoid forward bias the P transistor / diode (VDD + 0.2V).

Each bit can be set individually for pullup or not using Register **RegPBPullup**. Input is Pulled-Up when its corresponding bit in this register is set to 1.
Default status after reset is 0 which means no Pull_Up.
To conserve power, pull-ups are only enabled when the associated pin is either a digital input or on an N-channel open-drain output.

When the counters are used to implement a PWM function, the PB[0] and PB[1] terminals are declared as outputs and override other values written in **RegPBout**. **PBDir(0)** and **PBDir(1)** must be set to 1.

If **OutputCkXtal** is set in **RegSysMisc** the Xtal frequency is output on PB[3]. This overrides the value contained in **RegPBOut**.
Similarly, if **OutputCkCpu** is set in **RegSysMisc**, the CPU frequency is output on PB[2]. This overrides the value contained in **RegPBOut**. **PBDir(2)** and **PBDir(3)** must be set to 1.

Pins PB[5] and PB[4] can be used for Serial Data (SIN) and Serial Clock (SCL) when the **EnableUSRT** bit is set in **USRTctrl**. When **EnableUSRT** is set the PB[5] and PB[4] bidirectional become open-drain (**RegPBopen** is not changed). If there is no external Pull-Up resistor on these pins, this must be set to be pull-ups in **RegPBPullup**. When used as output and in synchronous serial mode, outputs take the value of **USRTSIN** and **USRTScl** bits from their respective registers **RegUSRTSIN** and **RegUSRTSCL**.

When **EnTx** in **RegUartCtrl** is set, PB[6] is output signal Tx.
When **EnRx** in **RegUartCtrl** is set, PB[7] is input signal Rx.

| Port B name | utilisation (priority) | | |
|---|---|---|---|
| | high | medium | low |
| PB[7] | analog | usart Rx | I/O |
| PB[6] | | usart Tx | I/O |
| PB[5] | analog | synchronous serial | I/O |
| PB[4] | | | I/O |
| PB[3] | analog | clock 32KHz | I/O |
| PB[2] | | clock CPU | I/O |
| PB[1] | analog | PWM1 Counter C (C+D) | I/O |
| PB[0] | | PWM0 Counter A (A+B) | I/O |

**Table 8.11: different PortB functions**

### 8.2.4 Port B analog capability

PortB terminals can be attached to 4 line analog bus in pairs by setting the **PBAna[x]** bits in **RegPBAna** register. With other registers we can program sharing of this 4 analog lines between different pads of PortB. This can be used to implement simple LCD driver or A/D converters. Analog switching is available only when the circuit is powered with sufficient voltage.

When **PBAna[x]** is set to 1, changing one pair of the PortB terminals from digital I/O mode to analog, the corresponding **RegPBPullup, RegPBOut** and **RegPBdir** change their functions.

Example: When **PBAna[0]** is set to 1 then the interpretation of two LSBs (bit0 and Bit1) in **RegPBOut** and **RegPBdir** change. PB[0] and PB[1] become analog pins. The other PortB pins (PB[7]..PB[2]) stay digital.

The selection is performed with **RegPBout** and **RegPBDir**. The analog line to which the appropriate signal is connected is determined by the codes placed on **RegPBout** and **RegPBDir** according to the following table.

| Selection bits from **RegPBDir** or **RegPBou** | | PB[x] selection on |
|---|---|---|
| 0 | 0 | analog line 0 |
| 0 | 1 | analog line 1 |
| 1 | 0 | analog line 2 |
| 1 | 1 | analog line 3 |

**Table 8.12: selection for analog lines with RegPBDir (pads B0, B2, B4 and B6) or RegPBout (pads B1, B3, B5 and B7)**

In the context of the example where **PBAna[0]** is set to 1: to connect PB[0] to analog line 3 and PB[1] to analog line 2 would require that f(**RegPBDir**[1], **RegPBDir**[0]) = (1,1) and that f(**RegPBOut**[1], **RegPBOut**[0]) = (1,0).

**RegPBTest** is used for internal test purposes.It can not be influenced by user outside test mode. In normal mode it is always read as 0.

**Note:** Depending on the status of an **EnResPConf** bit in **RegSysCtrl**, some registers can be reset by any of the possible system resets or only with PowerOnReset (POR).

| register name | address (hex) |
|---|---|
| RegPBOut | H0028 |
| RegPBIn | H0029 |
| RegPBDir | H002A |
| RegPBOpen | H002B |
| RegPBPullup | H002C |
| RegPBAna | H002D |
| reserved | H002E<br>H002F |

**Table 8.13: Port B registers**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | PBIn[7-0] | x | r | pad PB[7-0] input status |

**Table 8.14: RegPBIn**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | PBOpen[7-0] | 0 resetpconf | rw | pad PB[7-0] opendrain (1=opendrain) |

**Table 8.15: RegPBOpen**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-4 | -- | 0000 | r | used |
| 3 | PBAna[3] | 0 resetpconf | rw | set PB[7] and PB[6] in analog mode |
| 2 | PBAna[2] | 0 resetpconf | rw | set PB[5] and PB[4] in analog mode |
| 1 | PBAna[1] | 0 resetpconf | rw | set PB[3] and PB[2] in analog mode |
| 0 | PBAna[0] | 0 resetpconf | rw | set PB[1] and PB[0] in analog mode |

**Table 8.16: RegPBAna**

| bit | name | reset | rw | description in digital mode | descriptionin analog mode |
|---|---|---|---|---|---|
| 7-0 | PBPullup[7-0] | 0 resetpconf | rw | pullup for pad PB[7-0] (1=active) | connect pad PB[7-0] on selected ana bus |

**Table 8.17: RegPBPullup**

| bit | name | reset | rw | description in digital mode | description in analog mode |
|---|---|---|---|---|---|
| 7 | PBOut[7] | 0 resetpconf | rw | pad PB[7] output value | analog bus selection bit 1 for pad PB[7] |
| 6 | PBOut[6] | 0 resetpconf | rw | pad PB[6] output value | analog bus selection bit 0 for pad PB[7] |
| 5 | PBOut[5] | 0 resetpconf | rw | pad PB[5] output value | analog bus selection bit 1 for pad PB[5] |
| 4 | PBOut[4] | 0 resetpconf | rw | pad PB[4] output value | analog bus selection bit 0 for pad PB[5] |
| 3 | PBOut[3] | 0 resetpconf | rw | pad PB[3] output value | analog bus selection bit 1 for pad PB[3] |
| 2 | PBOut[2] | 0 resetpconf | rw | pad PB[2] output value | analog bus selection bit 0 for pad PB[3] |
| 1 | PBOut[1] | 0 resetpconf | rw | pad PB[1] output value | analog bus selection bit 1 for pad PB[1] |
| 0 | PBOut[0] | 0 resetpconf | rw | pad PB[0] output value | analog bus selection bit 0 for pad PB[1] |

**Table 8.18: RegPBOut**

| bit | name | reset | rw | description in digital mode | description in analog mode |
|---|---|---|---|---|---|
| 7 | PBDir[7] | 0 resetpconf | rw | pad PB[7] direction (0=input) | analog bus selection bit 1 for pad PB[6] |
| 6 | PBDir[6] | 0 resetpconf | rw | pad PB[6] direction (0=input) | analog bus selection bit 0 for pad PB[6] |
| 5 | PBDir[5] | 0 resetpconf | rw | pad PB[5] direction (0=input) | analog bus selection bit 1 for pad PB[4] |
| 4 | PBDir[4] | 0 resetpconf | rw | pad PB[4] direction (0=input) | analog bus selection bit 0 for pad PB[4] |
| 3 | PBDir[3] | 0 resetpconf | rw | pad PB[3] direction (0=input) | analog bus selection bit 1 for pad PB[2] |
| 2 | PBDir[2] | 0 resetpconf | rw | pad PB[2] direction (0=input) | analog bus selection bit 0 for pad PB[2] |
| 1 | PBDir[1] | 0 resetpconf | rw | pad PB[1] direction (0=input) | analog bus selection bit 1 for pad PB[0] |
| 0 | PBDir[0] | 0 resetpconf | rw | pad PB[0] direction (0=input) | analog bus selection bit 0 for pad PB[0] |

**Table 8.19: RegPBDir**

## 8.3 Port C

### 8.3.1 Features

- input / output port, 8 bits wide
- each bit can be set individually for input or output
- push-pull output only

### 8.3.2 Overview

PortC is a general purpose 8 bit Input/output port.

Data is stored in **RegPCOut** prior to output at PortC. This occurs provided that **PCDir[x]** has been set high (1), accordingly. Its default following reset is low (0).

The status of PortC is available in **RegPCIn** (read only). Reading is always direct - there is no digital debounce function associated with PortC. In case of possible noise of input signals, a software debouncer with pooling must be realized.

The direction of each bit within PortC (input or output) can be individually set using the **RegPCDir** register. If **PCDir[x]** = 1, the corresponding PortC pin becomes an output. Following reset PortC is in input mode (**PCDir[x]** are reset to 0).

**Note:** Depending on the status of an **EnResPConf** bit in **RegSysCtrl**, this register can be reset by any of the possible system resets or only with PowerOnReset (POR). See table Table 8.1.

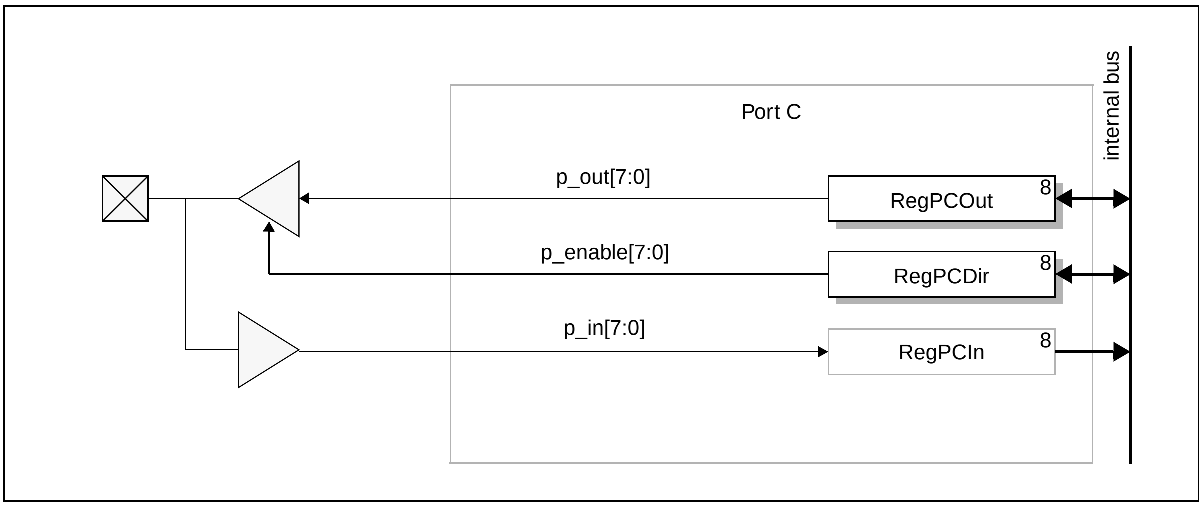


Figure 8.4: Port C

| register name | address (hex) |
|---|---|
| RegPCOut | H0030 |
| RegPCIn | H0031 |
| RegPCDir | H0032 |
| reserved | H0033 |

**Table 8.20: Port C registers**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | PCOut[7] | 0 resetpconf | r w | pad PC[7] output value |
| 6 | PCOut[6] | 0 resetpconf | r w | pad PC[6] output value |
| 5 | PCOut[5] | 0 resetpconf | r w | pad PC[5] output value |
| 4 | PCOut[4] | 0 resetpconf | r w | pad PC[4] output value |
| 3 | PCOut[3] | 0 resetpconf | r w | pad PC[3] output value |
| 2 | PCOut[2] | 0 resetpconf | r w | pad PC[2] output value |
| 1 | PCOut[1] | 0 resetpconf | r w | pad PC[1] output value |
| 0 | PCOut[0] | 0 resetpconf | r w | pad PC[0] output value |

**Table 8.21: RegPCOut**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | PCIn[7] | x | r | pad PC[7] input status |
| 6 | PCIn[6] | x | r | pad PC[6] input status |
| 5 | PCIn[5] | x | r | pad PC[5] input status |
| 4 | PCIn[4] | x | r | pad PC[4] input status |
| 3 | PCIn[3] | x | r | pad PC[3] input status |
| 2 | PCIn[2] | x | r | pad PC[2] input status |
| 1 | PCIn[1] | x | r | pad PC[1] input status |

**Table 8.22: RegPCIn**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 0 | PCIn[0] | x | r | pad PC[0] input status |

**Table 8.22: RegPCIn**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | PCDir[7] | 0 resetpconf | r w | pad PC[7] direction (0=input) |
| 6 | PCDir[6] | 0 resetpconf | r w | pad PC[6] direction (0=input) |
| 5 | PCDir[5] | 0 resetpconf | r w | pad PC[5] direction (0=input) |
| 4 | PCDir[4] | 0 resetpconf | r w | pad PC[4] direction (0=input) |
| 3 | PCDir[3] | 0 resetpconf | r w | pad PC[3] direction (0=input) |
| 2 | PCDir[2] | 0 resetpconf | r w | pad PC[2] direction (0=input) |
| 1 | PCDir[1] | 0 resetpconf | r w | pad PC[1] direction (0=input) |
| 0 | PCDir[0] | 0 resetpconf | r w | pad PC[0] direction (0=input) |

**Table 8.23: RegPCDir**

### 8.3.3 Port D

Is identical to port C.

# 9 Universal Asynchronous Receiver/Transmitter (UART)

## 9.1 Features

- Full duplex operation with buffered receiver and transmitter.
- Internal baudrate generator with 8 programmable baudrate (300 - 38400).
- 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

## 9.2 Overview

The Uart hardware is tied to PB[7] which is used as Rx - receive and PB[6] as Tx - transmit.

## 9.3 Uart Prescaler

In order to have correct baudrates, the Uart interface has to fed with a stable and trimmed clock source. It can be issued by the Xtal oscillator or by the RC oscillator (bit **SelXtal** in **RegUartCmd**). The following RC frequencies are suitable for correct Uart protocols.

| rc freq [Hz] |
|---|
| 9’830’400 |
| 4’915’200 |
| 2’457’600 |
| 1’228’800 |
| 614’400 |

**Table 9.1: RC frequencies for Uart**

The internal prescaler of the Uart has to be set according to the RC frequency :

| UartRcSel[2:0] | prescaler division | rc freq [Hz] |
|---|---|---|
| 111 | reserved | |
| 110 | reserved | |
| 101 | reserved | |
| 100 | : 16 | 9’830’400 |
| 011 | : 8 | 4’915’200 |
| 010 | : 4 | 2’457’600 |
| 001 | : 2 | 1’228’800 |
| 000 | no division | 614’400 |

**Table 9.2: uart internal prescaler**

**Note:** the bit **UartRcSel[2:0]** in **RegUartCmd** only sets the internal prescaler of the Uart. It has no influence on the RC oscillator trimming or on the CPU prescaler.

## 9.4 Function description

The bit **UartTxFull** in **RegUartTxSta** goes to 0 when the uart transfers data from the **RegUartTx** register to the transmitter shift register, and goes to 1 when the processor writes new data onto the **RegUartTx**.
An interrupt is generated when there is a falling edge of the **UartTxFull** bit.

The bit **UartTxBusy** in **RegUartTxSta** shows that the transmitter has transmitted data.

The bit **UartRxFull** in **RegUartRxSta** goes to 1 when the uart transfers data from the receiver shift register to **RegUartRx**, and goes to a 0 when the processor reads the **RegUartRx**.
An interrupt is generated when there is a rising edge of the **UartRxFull** bit.

The bit **UartRxBusy** in **RegUartRxSta** shows that the receiver has received data.

The bit **UartRxSErr** in **RegUartRxSta** shows that a start error has been detected.

The bit **UartRxPErr** in **RegUartRxSta** shows that a parity error has been detected. The received parity is not equal to the calculated parity with the received data.

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

The bit **UartRxOErr** in **RegUartRxSta** shows that an overrun error has been detected. The reception buffer is not empty when a new data is received. This bit is cleared by all reset conditions and by writing any data to its address.

The bit **UartEcho** in **RegUartCtrl** is used to return the Rx signal on the Tx signal. Tx = Rx XOR UartXRx XOR UartXTx.
UartEnRx (register RegUartCtrl bit 6) must be zero if the reception of data in echo mode is not wanted.

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

The bits **UartXRx** and **UartXTx** in **RegUartCtrl** are used to enable or disable the reversal on the Rx and Tx signal.

The bits **UartBR** in **RegUartCtrl** are used to select the internal baudrate.

The **RegUartCmd** register is used to select the word length (7-8 data bit) **UartWL**, the enable or disable parity **UartPE** and the parity mode (odd or even) **UartPM**.

## 9.5 Interrupt or polling

There are two possibilities for transmit or receive a message.
First by interrupt, when the **RegUartRx** is full an interrupt is generated, the **UartRx** register must be read. And when the **RegUartTx** is empty an interrupt is generated too, the **UartTx** register can be load.
Second by polling, reading and checking the **UartRxFull** bit and when it is at 1 the **RegUartRx** register must be read. Also, reading and checking the **UartTxdFull** bit and when it is 0 the **RegUartTx** register can be load.

## 9.6 Software hints

Example of programme for a transmission with polling:
1. The **RegUartCmd** register and the **RegUartCtrl** register are initialized (for example: 8 bit WordLen, Odd Parity, 9600 baud, enable Uart Transmission).
2. Write a byte in **RegUartTx**.
3. Wait on the **UartTxFull** bit in **RegUartTxSta** register equal 0.
4. jump to 2 for 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 WordLen, Odd Parity, 9600 baud, enable Uart Transmission).
2. Write a byte in **RegUartTx**.
3. When there is an interrupt, if the message is not finished then write the next byte in **RegUartTx** else 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 WordLen, Odd Par-

ity, 9600 baud, enable Uart Reception).
2. Wait on the **UartRxFull** bit in **RegUartRxSta** register equal 1.
3. Reading the **RegUartRxSta** and checking if there is no error.
4. Read data in **RegUartRx**.
5. If data is not equal at End-Of-Line then jump to 5.
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 WordLen, Odd Parity, 9600 baud, enable Uart Reception).
2. When there is an interrupt jump to 3
3. Reading the **RegUartRxSta** and checking if there is no error.
4. Read data in **RegUartRx**.
5. If data is not equal at End-Of-Line then jump to 2.
6. End of reception.

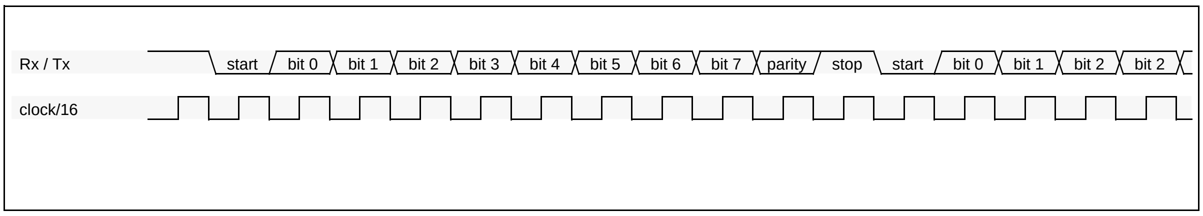


Figure 9.1: example of UART messages

| UartBR[2-0] | baud rate, RC input | baud rate, Xtal input |
|---|---|---|
| 111 | 38400 | - |
| 110 | 19200 | - |
| 101 | 9600 | - |
| 100 | 4800 | - |
| 011 | 2400 | 2400 |
| 010 | 1200 | 1200 |
| 001 | 600 | 600 |
| 000 | 300 | 300 |

**Table 9.3: baud rate selection**

| UartWL | word length |
|---|---|
| 1 | 8 bits |
| 0 | 7 bits |

**Table 9.4: word length**

| UartPM | parity mode |
|---|---|
| 1 | even |
| 0 | odd |

**Table 9.5: parity mode**

| UartPE | parity enable |
|---|---|
| 1 | with parity |
| 0 | no parity |

**Table 9.6: parity enable**

| register name | address (hex) |
|---|---|
| RegUartCtrl | H0050 |
| RegUartCmd | H0051 |
| RegUartTx | H0052 |
| RegUartTxSta | H0053 |
| RegUartRx | H0054 |
| RegUartRxSta | H0055 |

**Table 9.7: UART registers**

| UartEcho | echo |
|---|---|
| 1 | echo Rx -> Tx |
| 0 | no echo |

**Table 9.8: echo modes**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | SelXtal | 0 resetsystem | r w | Select input clock; 0->RC, 1->Xtal |
| 6 | UartWakeup | 0 resetsystem | r w | Uart reception enabled on falling on Rx |
| 5-3 | UartRcSel[2:0] | 000 resetsystem | r w | Rc prescaler selection |
| 2 | UartPM | 0 resetsystem | r w | select parity mode |
| 1 | UartPE | 0 resetsystem | r w | enable parity |
| 0 | UartWL | 1 resetsystem | r w | select word length |

**Table 9.9: RegUartCmd**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | UartEcho | 0 resetsystem | r w | enable Echo Mode |
| 6 | UartEnRx | 0 resetsystem | r w | enable Uart reception |
| 5 | UartEnTx | 0 resetsystem | r w | enable Uart Transmission |
| 4 | UartXRx | 0 resetsystem | r w | invert pad Rx |
| 3 | UartXTx | 0 resetsystem | r w | invert pad Tx |
| 2-0 | UartBR[2-0] | 101 resetsystem | r w | select baud rate |

**Table 9.10: RegUartCtrl**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | UartRx | xxxxxxxx | r | data received |

**Table 9.11: RegUartRx**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-6 | -- | 00 | r | unused |
| 5 | UartRxSErr | x | r | start error |
| 4 | UartRxPErr | x | r | parity error |
| 3 | UartRxFErr | x | r | frame error |
| 2 | UartRxOErr | 0 resetsystem | c | overrun error<br>cleared by writing RegUartRxSta |
| 1 | UartRxBusy | 0 resetsystem | r | Uart is busy receiving |
| 0 | UartRxFull | 0 resetsystem | r | RegUartRx is Full (irq on full)<br>cleared by reading RegUartRx |

**Table 9.12: RegUartRxSta**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | UartTx | 00000000 resetsystem | r w | data to send |

**Table 9.13: RegUartTx**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-2 | -- | 000000 | r | unused |
| 1 | UartTxBusy | 0 resetsystem | r | Uart is busy transmitting |
| 0 | UartTxFull | 0 resetsystem | r | RegUartTx is full (irq on empty)<br>set by wrinting RegUartTx |

**Table 9.14: RegUartTxSta**

# 10 Universal Synchronous Receiver/Transmitter (USRT)

## 10.1 Overview

The XE8000 includes hardware to support the software implementation of several bidirectional synchronous serial interfaces, for Master and Slave modes.

The serial transceiver hardware is connected to PB[5] (SIN: Synchronous INput), PB[3] (SOU: Synchronous OUtput) and PB[4] (SCL: Synchronous CLock).

The register **RegUSRTSIN** can be used to read SIN when the serial interface is in Receive = Slave mode. It is advised to read the SIN when in Slave mode from the **RegUSRTData** register (data is stored on SCL rising edge in this register).

The **RegUSRTSCL** register is used to read SCL when interface is in Receive mode or to drive SCL by writing it when in Transmit mode.

### 10.1.1 Enabling the serial interface

The bit **EnableUSRT** in **RegUSRTCtrl** is used to enable the serial interface and controls the SIN and SCL lines. This bit puts these two PortB lines in open drain configuration.

If no external SIN and SCL Pull-ups are added, the user has to add them by software by setting **PBPullup[5**] and **PBPullup[4]** in **RegPBPullup** (see chapter “Parallel IO ports” ).

### 10.1.2 Reading the serial interface

The bit **USRTData** in **RegUSRTData** is the prefered way to read the SIN data line because this data is stored on the rising edge of the SCL and will change only on the next rising SCL edge.

The bit **USRTEdgeSCL** in **RegUSRTEdgeSCL** is set to one on SCL rising edge and is cleared on any reset as well as by reading the data from the **RegUSRTData**. It is the prefered way of knowing if data has been stored.

## 10.2 Registers

| register name | address (hex) |
|---|---|
| RegUSRTSIN | H0060 |
| RegUSRTSCL | H0061 |
| RegUSRTCtrl | H0062 |
| Reserved | H0063 |
| Reserved | H0064 |
| RegUSRTData | H0065 |
| RegUSRTEdgeSCL | H0066 |
| Reserved | H0067 |

**Table 10.1: serial interface registers**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-1 | Reserved | 0000000 | r | reserved |
| 0 | USRTSIN | 1 resetsystem | r w | Serial Data (SIN on pad PB[5]) |

**Table 10.2: RegUSRTSIN**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-1 | Reserved | 0000000 | r | reserved |
| 0 | USRTSCL | 1 resetsystem | r w | Serial Clock (SCL on pad PB[4]) |

**Table 10.3: RegUSRTSCL**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-3 | Reserved | 00000 | r | reserved |
| 2-1 | Reserved | 00 | r w | reserved, should always be set to 0 |
| 0 | USRTEnable | 0 resetsystem | r w | enable hardware and pads for USRT (0=disable) |

**Table 10.4: RegUSRTCtrl**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-1 | Reserved | 0000000 | r | reserved |
| 0 | USRTData | x | r | last received data |

**Table 10.5: RegUSRTData**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-1 | Reserved | 0000000 | r | reserved |
| 0 | USRTEdgeSCL | 0 resetsystem | r | edge detection on SCL (1 = edge detected) cleared by reading RegUSRTData |

**Table 10.6: RegUSRTEdgeSCL**

# 11 Counters/timers

## 11.1 Introduction

They are several types of counters in the XE8000:

- Watchdog
- 4 general purpose counters/timers
- Prescaler

The watchdog is a counter that sends an interrupt when it is not correctly addressed within a given time. It is used to recover from abnormal situations, like endless software loops.

The general purpose counters/timers are used for counting events, counting time, measuring frequency (capture function) and generating analog-like outputs (PWM).

The prescaler (see “Prescaler” on page 65) is used to extract all necessary signals from the different clocks.

## 11.2 Watchdog

Watchdog is a timer which has to be cleared at least every 4 seconds by the software to prevent program overrun.

The watchdog can be enabled by software, and it is disabled at power-on. Once enabled, it cannot be disabled by software (only with power-on reset). See register **RegSysCtrl** (bit **EnResWD**).

The watchdog is made of a 3 stage divider chain clocked by a 2 Hz signal from the prescaler and gives a system reset if not cleared within 4 seconds. The watchdog timer can be cleared by writing two successive **WDKeys** to **RegSysWD** register. This sequence must be respected. Only writing hxA followed by hx3 resets the WD.

Example :
```
move                      AddrRegSysWD, #0x0A
move                      AddrRegSysWD, #0x03
```

If some other write instruction is done to **RegSysWD** between the hxA and hx3, the watchdog timer will not be reset.

It is possible to read the status of the **RegSysWD** register. The watchdog timer doesn’t perform a full range count for the 4 bits associated with it. The watchdog is a 4 bit counter, however the count range is 0 to 7 and following the count of 7, the system reset is generated concurrently with the setting of **WDCount[3]**.

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-4 | -- | 0 | r | reserved |
| 3 | WDKey[3] | | w | Watchdog Key bit 3 |
| | WDCounter[3] | 0 resetpor | r | Watchdog counter bit 3 (1/4 Hz) |
| 2 | WDKey[2] | | w | Watchdog Key bit 2 |
| | WDCounter[2] | 0 resetpor | r | Watchdog counter bit 2 (1/2 Hz) |
| 1 | WDKey[1] | | w | Watchdog Key bit 1 |
| | WDCounter[1] | 0 resetpor | r | Watchdog counter bit 1 (1 Hz) |
| 0 | WDKey[0] | | w | Watchdog Key bit 0 |
| | WDCounter[0] | 0 resetpor | r | Watchdog counter bit 0 (2 Hz) |

**Table 11.1: RegSysWD**

## 11.3 Counters

### 11.3.1 Overview

There are four general purpose 8-bit wide Up/Down counters, CounterA, CounterB, CounterC & CounterD.

Each counter has four possible clock inputs. Three of them are internal clocks provided by the prescaler and the forth is a PortA pin. The clock input of PortA is set as a normal portA input (rising/falling, debounced/not debounced).

CounterA and CounterB can be combined to form a 16-bit counter and similarly with CounterC and CounterD.

When not in PWM mode, counters can generate an interrupt when reaching a predefined value.

Counters can also be used to generate two PWM outputs on PB[1] and PB[0]. In PWM mode one can generate PWM functions on 8 to 16 bits width.

**Note:** If **PowerXtal** is set when the counter is set to use one of the low frequency clocks (32kHz or lower), counter switches immediately on Xtal. The Xtal frequency takes time to stabilize. **ColdXtal** goes low when the Xtal is considered stable. Additionally, the low 15-stage divider of the Prescaler is cleared when **PowerXtal** is set.

### 11.3.2 Features

- four independent 8-bit up/down counters - each with 4 possible clock selections
- PA[3:0] can be used as clock inputs (debounced or direct)
- counters can be set in pairs to form blocks of 16-bit counters
- generate interrupts when used as normal counters
- PWM function on two simple (8-bit) or combined (16-bit) counters.
- PWM function on 8 / 10 / 12 / 14 / or 16 bit width
- Capture function (internal or external)
- PWM output on PB[1] and PB[0]

### 11.3.3 Block schematics

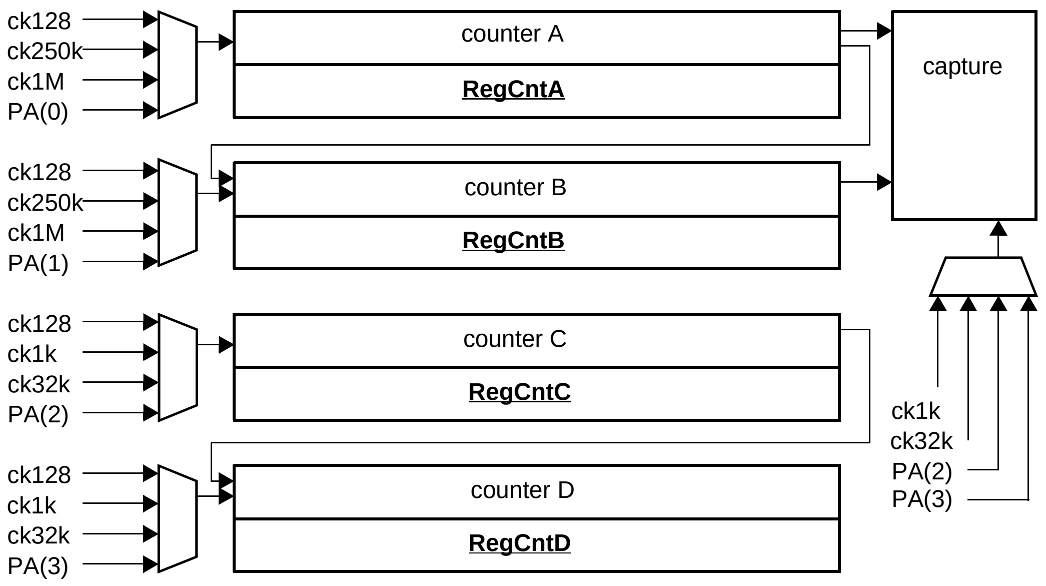


Figure 11.1: Counters/timers block schematics

### 11.3.4 Counter Registers

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

Each counter has a corresponding 8 bit read/write register **RegCntA**, **RegCntB**, **RegCntC**, and **RegCntD**. These registers contain the counter value for the purposes of reading. When written by the user, they contain the comparison values. They can only be reliably modified when the counter is stopped.

To stop the counter, **Cnt**X**Enable** must be reset. To start the counter, **Cnt**X**Enable** must be set. The only operation allowed when the counter is stopped is loading the counter with data. If no new data has been written, the counter remains unchanged. So the smallest sequence is a STOP/MODIFY/START.

It is possible to read any counter at any time, even when the counter is running. However, the value is only guaranteed correct when the counter is static. For precise counter acquisition, one should use the capture function (see below).

### 11.3.5 Clock selection

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

Each value can be 1 of 4 (2 bits) : each represents one of the different clock options. Table 7.1, 7.2, 7.3 & 7.4 describe the code for each counter clock source combination. See chapter “Prescaler” on page 65, and Figure 7.4 for more explainations about the ck128, ck1k, ck32k, ck250k, and ck1M signals.

| CntACkSel[1:0] | clock source |
|---|---|
| 11 | ck128 |
| 10 | ck250k |
| 01 | ck1M |
| 00 | PA[0] |

**Table 11.2: clock source for Counter A**

| CntBCkSel[1:0] | clock source |
|---|---|
| 11 | ck128 |
| 10 | ck250k |
| 01 | ck1M |
| 00 | PA[1] |

**Table 11.3: clock source for Counter B**

| CntCCkSel[1:0] | clock source |
|---|---|
| 11 | ck128 |
| 10 | ck1k |
| 01 | ck32k |
| 00 | PA[2] |

**Table 11.4: clock source for Counter C**

| CntDCkSel[1:0] | clock source |
|---|---|
| 11 | ck128 |
| 10 | ck1k |
| 01 | ck32k |
| 00 | PA[3] |

**Table 11.5: clock source for Counter D**

The clock source must be changed ONLY WHEN THE COUNTER IS STOPPED. Once the counter is restarted/started, the circuit wait for a falling edge on the clock signal (internally generated clock or external source), to start counting. The counter is modified at the clock rising edge.

Depending when the start of the counter related to its selected clock arrives, the first counter clock might not be counted because the first falling edge is used for synchronization and counter preparation.

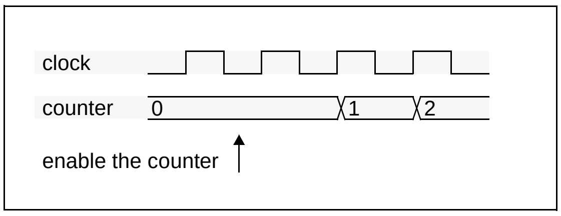


Figure 11.2: start synchronization

When the counter arrives at IRQ condition (if count down when it change to 00), it does not give an IRQ at that moment but one half period of selected clock later.

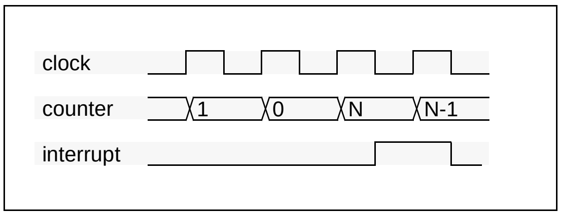


Figure 11.3: interrupt generation

If corresponding CounterMask bits are set to 1, interrupts are generated on the clock's falling edge after the counter has reached the expected value. The counter is stopped immediately by setting start at 0. If a rising edge occurs exactly at this time, we cannot predict if the counter will be modified or not. For this reason, there is always a 1 lsb uncertainty on the counter value.

**Note:** You can read the counter properly ONLY WHEN THE COUNTER IS STOPPED.

Reading on run can give false values. This case is not advised, use only when the counter counts at least 8 times lower than CPU clock. In this case user can do a software filer of at least three readings to determine the counter value.
Be careful to select the counter mode BEFORE writing any value in it.

Value is not stored in the counter in the “UP-COUNT” mode. In “Down-COUNT” or “PWM” mode, the value is stored in the counter.

When you write a value into the counter, the counter will be cleared if you are in UP-COUNT mode. Otherwise, the counter will be loaded with the value. If you do not write a value, the counter will not be modified, even if you are modifying the mode.

### 11.3.6 16 bit counters

**CascadeAB** and **CascadeCD** in **RegCntConfig1** are used to set the count ranges.

When **CascadeAB** is set, the CounterA is connected to CounterB to form a 16-bit wide counter. In this case, CounterA is the LSB and CounterB is the MSB. When not cascaded (CascadeAB = 0), the counters are independent.

CounterC and CounterD can be set similarly with **CascadeCD**.

There is no default values, so these bits must be set before using any of the counters.

| CascadeAB | counters |
|---|---|
| 1 | 16 bits counter AB |
| 0 | 8 bit counter A<br>8 bit counter B |

**Table 11.6: cascading counter A & B**

| CascadeCD | counters |
|---|---|
| 1 | 16 bits counter CD |
| 0 | 8 bit counter C<br>8 bit counter D |

**Table 11.7: cascading counter C & D**

Only CounterA and CounterC IRQ are generated in 16 bit mode.

## 11.3.7 Up/down counting

The counters can be up- or down-counting.

### 11.3.7.1 Up-counting

For 8 bits up-counting, we see the following behaviour:
- Counter A: 0,1,2,...,ValueA-1,ValueA,0,1,....,ValueA-1,ValueA,...
- Counter B: 0,1,2,...,ValueB-1,ValueB,0,1,....,ValueB-1,ValueB,...

For 16 bits up-counting, we see the following behaviour:
- Counter (B,A): (00, 00), (00, 01),..., (00, FF), (01, 00), (01, 01), .., (ValueB, ValueA), (00,00), ...

In 16 bits mode, counter B is incremented only when counter A is FF

IRQ generated when counter reaches VALUE.

### 11.3.7.2 Down-counting

For 8 bits down-counting, we see the following behaviour:
- Counter A: ValueA,...,2,1,0,ValueA,ValueA-1,....,2,1,0,ValueA,...
- Counter B: ValueB,...,2,1,0,ValueB,ValueB-1,....,2,1,0,ValueB,...

For 16 bits down-counting, we see the following behaviour:
- Counter (B,A): (ValueB, ValueA), (ValueB, ValueA-1),..., (ValueB, 00), (ValueB-1, FF), .., (00,00), (ValueB, ValueA), ...

IRQ generated when counter reaches 0.

### 11.3.7.3 Registers for up/down counting

**CntADownUp**, **CntBDownUp**, **CntCDownUp** & **CntDDownUp** in **RegCntConfig1** are used to individually set up and down counting for Counters A, B ,C & D.

**Note: There is no default values, so these bits must be set before using any of the Counters.**

| CntXDownUp | CounterX |
|---|---|
| 1 | up-counting |
| 0 | down-counting |

**Table 11.8: selection for up/down-counting**

An IRQ will be generated every “VALUE+1” clock cycles

For example in 8 bits mode, if you load the counter with a value h5B, with a 4 MHz clock, an interrupt will be generated every 23 us; in 16 bits mode, if you load the counter will a value h5BA7, with a 4 MHz clock, an interrupt will be generated every 5866 us.

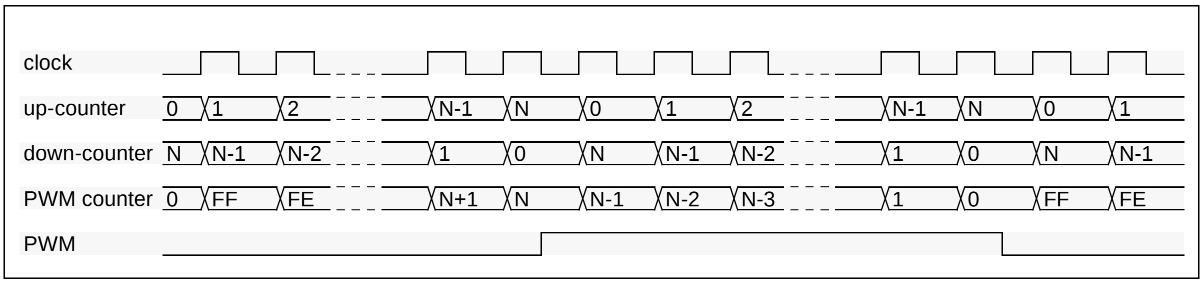

clock
up-counter
0 1 2 N-1 N 0 1 2 N-1 N 0 1
down-counter
N N-1 N-2 1 0 N N-1 N-2 1 0 N N-1
PWM counter
0 FF FE N+1 N N-1 N-2 N-3 1 0 FF FE
PWM

Figure 11.4: counter examples

### 11.3.8 Capture functions

#### 11.3.8.1 Overview

The 16 bit capture register is provided to facilitate frequency measurements. It captures the data held in counters A and B and is split into CaptureA and CaptureB. The instant of capture for both CaptureA and CaptureB are user defined by selecting either internal sources derived from the prescaler or from a chip pin (PA[2] or PA[3]). Both registers use the same capture source. For all sources, rising edge sensitivity, falling edge sensitivity or both can be selected dynamically. This is especially useful for synchronizing with signals for which duty cycle measurements are required.
The source of the capture signal and the edge sensitivity are determined in **RegCntConfig2**.

When counters A or B are active, reads performed on their associated addresses come from the capture register. Figure 11.5 shows a representation of the capture block within the context of counter A.

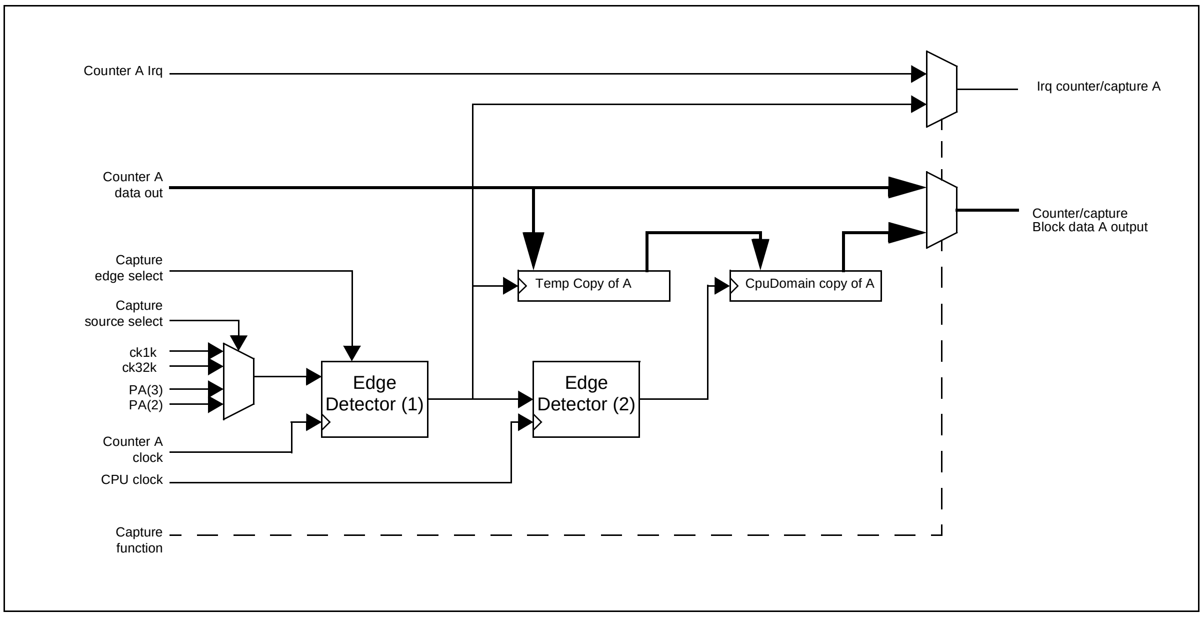


Figure 11.5: Capture Architecture (counter A)

### 11.3.8.2 Detailed implementation

Several edge detectors and temporary registers are implemented to ensure proper operation of the capture function despite the asynchronous multiple clocks. Understanding of the detailed implementation is only required for very precise measurements. Other users may directly go to the registers chapter.

When in capture mode the capture block becomes the source of interrupts from the counter/capture block A. Additionally, the global data register address associated with the counter/capture block A becomes that labelled as CPU-domain copy of A within the figure. Within the Capture architecture there are two edge detector blocks, namely:

- Edge detector (1)
- Edge detector (2)

Edge detector (1) generates the strobe pulse that captures the contents of the counter into the register marked Temp copy of A within Figure 11.5. This strobe signal is also used to communicate the availability of data within that register to the Edge detector (2) block which in turn generates the strobe signal that copies the data into the cpu clock domain.

The architectures of Edge detector (1) and Edge detector (2) are represented in Figure 11.6 and Figure 11.7 respectively.

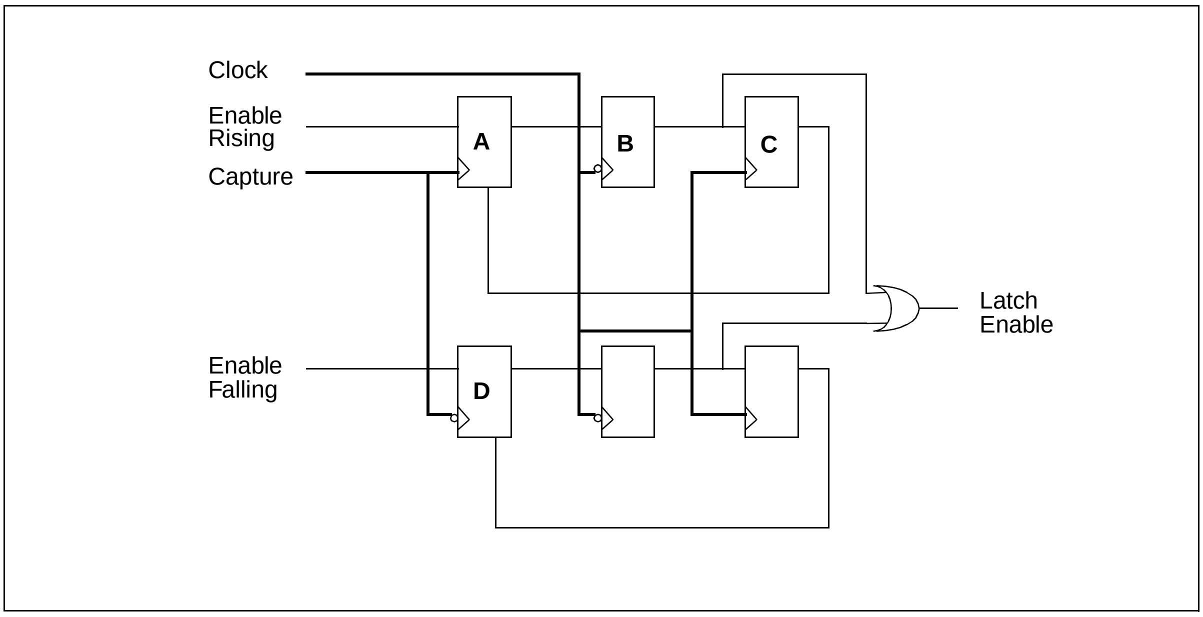


Figure 11.6: Edge detector (1) principle

The Edge detector (1) represented in Figure 11.6 may be divided into two sections that are associated with rising edge detection (top row of cells) and falling edge detection (bottom row of cells). To explain the circuit functionality the rising edge detection is used as an example. The state of Enable Rising is user defined. Assuming that it is disabled (set to 0) then no contribution is made to the Latch Enable output circuit.

If the Enable Rising signal is active (set to 1), then the output of d-flip-flop A (subsequently referred to as **A**) becomes high within a flip-flop delay time of the rising edge of the capture signal. This signal remains until **A** is reset. Following the next falling edge of the Clock signal, the output of **A** is copied to **B**. The next rising edge of clock copies this further to **C**. The output of **C** causes **A** to be reset; only to become active following the detection of the next rising edge of the capture signal (if enabled). This structure is based on conventional techniques for exchanging data between different clock domains. Within the bottom row of cells, **D** is sensitive to the falling edge of Capture.

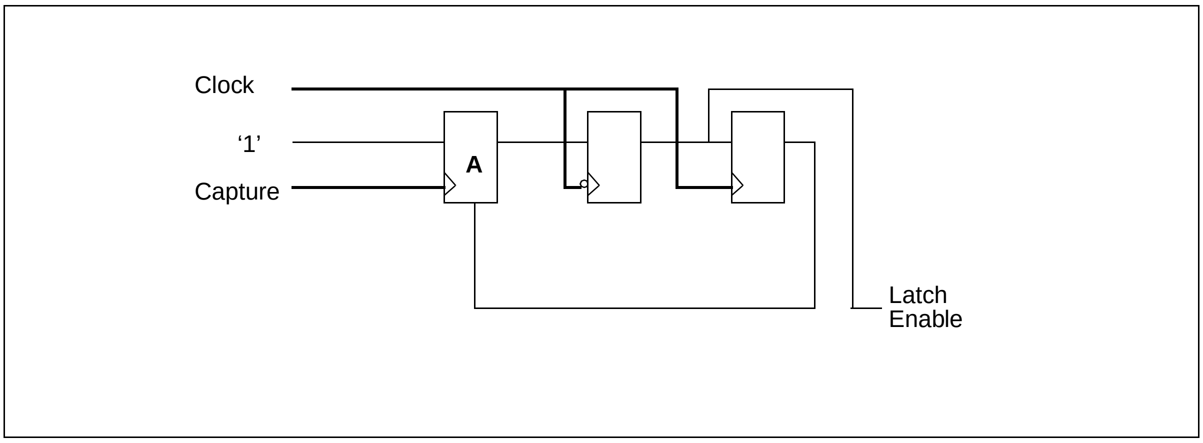


Figure 11.7: Edge detector (2) principle

The behavior of the Edge Detector (2) is similar to that previously described, with the exception that it is always enabled and the output of cell **A** is always set following the rising edge of the flip flop generated by Edge Detector (1).

In both circuits, the outputs of cells **A** and **D** can only be reset (during normal operation) as a result of acknowledgment of the request signal (**A** or **D**) by downstream cells.

#### 11.3.8.3 Registers

Table 11.9 and Table 11.10 describe the mapping between the codes placed on CapSel[1:0] (Capture source select) and CapFunc[1:0] (Capture Function) and their meanings.

| CapSel[1:0] | source |
|---|---|
| 11 | ck1k (RC divider) |
| 10 | ck32k (RC divider) |
| 01 | PA[3] |
| 00 | PA[2] |

**Table 11.9: capture source**

| CapFunc[1:0] | selection |
|---|---|
| 11 | both edges |
| 10 | falling edge |
| 01 | rising edge |
| 00 | disabled |

**Table 11.10: capture function selection**

### 11.3.9 PWM functions

#### 11.3.9.1 General

The counters can generate PWM (pulse width modulation) PB(0) and PB(1) outputs.

PWM mode override normal outputs settings on PB(0) and PB(1) (except analog mode).

The counters can be used in 8 or 16 bits mode. Counters in PWM mode always count down, independently of **CntX-DownUp** setting.

Output is low as long as the value of the counter is bigger than the stored value, and high when the value of the counter is smaller or equal to the stored value.

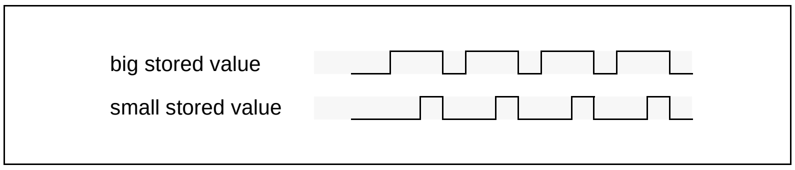


Figure 11.8: PWM output examples

**CntPWM1** and **CntPWM0** in **RegCntConfig1** are used to set PWM functions. These two bits are by default after any reset cleared to 0 disabling the PWM function.

| CntPWM1 | PWM1 | PB[1] |
|---|---|---|
| 1 | active | outputs PWM1 |
| 0 | stopped | normal function |

**Table 11.11: PWM1**

| CntPWM0 | PWM0 | PB[0] |
|---|---|---|
| 1 | active | outputs PWM0 |
| 0 | stopped | normal function |

**Table 11.12: PWM0**

PWM resolution is always 8 bits when the counters are in 8 bits mode. **PWM1Size** and **PWM0Size** in **RegCntConfig2** are used to set the PWM resolution when the counters are in 16 bits mode (**CascadeAB** or/and **CascadeCD** set).

| PWM1Size[1:0] | Size of PWM1 |
|---|---|
| 11 | 16 bits |
| 10 | 14 bits |
| 01 | 12 bits |
| 00 | 10 bits |

**Table 11.13: PWM1 size selection**

| PWM0Size[1:0] | Size of PWM0 |
|---|---|
| 11 | 16 bits |
| 10 | 14 bits |
| 01 | 12 bits |
| 00 | 10 bits |

**Table 11.14: PWM0 size selection**

**Note:** Counters used for PWM do not generate any IRQ.

#### 11.3.9.2 PWM in 8 bits mode

With “Value” in the range from 0 - FF one selects the duty cycle of the output between 0 and 99.6%.
h00 = 0%, h3F = 25%, h7F = 50%, hFF = 255/256.

#### 11.3.9.3 PWM in 16 bits mode

The counter counts from 0, MAX, MAX-1, ....., 2, 1, 0, MAX, MAX-1, .......
MAX is depending on the PWM size setting. It is hFFFF for 16 bits, h3FFF for 14 bits, h0FFF for 12 bits and h03FF for 12 bits.

### 11.3.10 Counter registers

| register name | address (hex) |
|---|---|
| RegCntA | H0058 |
| RegCntB | H0059 |
| RegCntC | H005A |
| RegCntD | H005B |
| RegCntCtrlCk | H005C |
| RegCntConfig1 | H005D |
| RegCntConfig2 | H005E |
| RegCntOn | H005F |

**Table 11.15: Counters registers**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | CounterA | xxxxxxxx | r w | 8-bit Counter A |

**Table 11.16: RegCntA**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | CounterB | xxxxxxxx | r w | 8-bit Counter B |

**Table 11.17: RegCntB**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | CounterC | xxxxxxxx | r w | 8-bit Counter C |

**Table 11.18: RegCntC**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-0 | CounterD | xxxxxxxx | r w | 8-bit Counter D |

**Table 11.19: RegCntD**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-6 | CntDCkSel[1:0] | xx | r w | CounterD clock selection |
| 5-4 | CntCCkSel[1:0] | xx | r w | CounterC clock selection |
| 3-2 | CntBCkSel[1:0] | xx | r w | CounterB clock selection |
| 1-0 | CntACkSel[1:0] | xx | r w | CounterA clock selection |

**Table 11.20: RegCntCtrlCk**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7 | CntDDownUp | x | r w | CounterD down- or up-counting (0=down) |
| 6 | CntCDownUp | x | r w | CounterC down- or up-counting (0=down) |
| 5 | CntBDownUp | x | r w | CounterB down- or up-counting (0=down) |
| 4 | CntADownUp | x | r w | CounterA down- or up-counting (0=down) |
| 3 | CascadeCD | x | r w | cascade Counter C & Counter D (1=cascade) |
| 2 | CascadeAB | x | r w | cascade Counter A & Counter B (1=cascade) |
| 1 | CntPWM1 | 0 resetsystem | r w | activate PWM1 on Counter C or C+D (PB[1]) |
| 0 | CntPWM0 | 0 resetsystem | r w | activate PWM0 on Counter A or A+B (PB[0]) |

**Table 11.21: RegCntConfig1**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-6 | CapSel[1:0] | 00 resetsystem | r w | capture source selection |
| 5-4 | CapFunc[1:0] | 00 resetsystem | r w | capture function |

**Table 11.22: RegCntConfig2**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 3-2 | PWM1Size[1:0] | xx | r w | PWM1 size selection |
| 1-0 | PWM0Size[1:0] | xx | r w | PWM0 size selection |

**Table 11.22: RegCntConfig2**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-4 | -- | 0000 | r | reserved |
| 3 | CntDEnable | 0 resetsystem | r w | enable Counter D |
| 2 | CntCEnable | 0 resetsystem | r w | enable Counter C |
| 1 | CntBEnable | 0 resetsystem | r w | enable Counter B |
| 0 | CntAEnable | 0 resetsystem | r w | enable Counter A |

**Table 11.23: RegCntOn**

# 12 Voltage Level Detector

## 12.1 Features

- can be switched off and on
- generates an interrupt if power supply is below a pre-determined level at end of measurement

## 12.2 Overview

The Voltage level detector monitors the state of the system battery. It returns a logical value depending on the result. If the supplied voltage drops below a user defined value (Vsb) then an interrupt request (‘1’ on **VldIrq**) is generated.

## 12.3 VLD operation

The VLD is controlled by **VldMult**, **VldTune** and **VldEn**. **VldMult** selects the voltage range applicable to the product, while VldTune is used to programme the reference voltage level. **VldEn** is used to enable (disable) the VLD with a 1(0) value respectively.

The user tunes the Vsb level using the **VldTune** bits. In low and high range, 8 values of Vsb can be selected.

| symbol | description | min | typ | max | unit | comments |
|---|---|---|---|---|---|---|
| Vsb | threshold voltage for VLD_tune = h100 | 1.2<br>2.4 | 1.3<br>2.55 | 1.4<br>2.7 | V<br>V | low range<br>high range |
| dVsb | step size of tuning | | 6 | | % | relative to Vsb |
| $t_{EOM}$ | duration of measurement | | | 2 | ms | time between setting **VldEn** and rise of **VldValid** |

**Table 12.1: Voltage level detector operation**

To start the voltage level detection, the user sets bit **VldEn**. Analog part is powered and measurement is started. After the measurement, Bit **VldValid** is set to signal validity of **VldIrq**, a maskable interrupt request is sent if voltage level is below threshold. By reading bit **VldIrq**, the user knows the power supply status.

Figure 12.1 describes the functionality of the **VldValid** and **VldIrq** signals with respect to the battery level (Vbat), the measurement made by the analog part of the vld (vld_out), the vld enable signal, the vld_valid signal and the vld_irq signal.The VLD is intended to be polled, namely: the user requests that a measurement be taken by **VldEn**. The bit **VldValid** becomes active indicating the validity of **VldIrq**. Once the **VldIrq** has been read, the user can disable the VLD by setting the **VldEn** to 0.

Figure 12.1 shows that the **VldValid** signal rises to its active state with the sixteenth rising ck8k edge following the rise of **VldEn**. The intermediate time is required for the vld to stabilize. Digital filtering is used to filter the output signal.

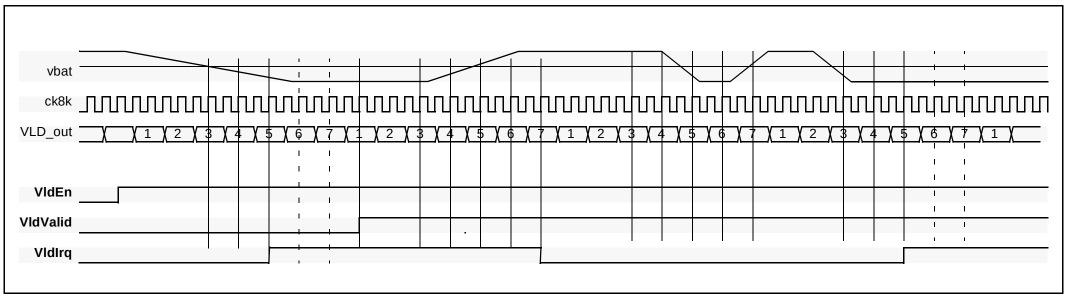


Figure 12.1: VLD timing

Figure 12.1 also shows that if the VLD remains in the enabled state, it continues to generate interrupt requests as appropriate. However, it should be noted that these are based on the existence of three consecutive agreeing measurements being made, and therefore the transitions occuring on **VldIrq** are not always equi-distant.

## 12.4 Registers

The CPU interface comprises two 8 bit registers that are globally accessible, namely **RegVldCtrl** and **RegVldStat**. Table 12.2 shows the mapping of control bits and functionality of **RegVldCtrl** while Table 12.3 describes that for **RegVldStat**.

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-4 | -- | 0000 | r | reserved |
| 3 | VldMult | 0 resetcold | r w | VLD double level dectector |
| 2-0 | VldTune[2:0] | 000 resetcold | r w | VLD level tuning |

**Table 12.2: RegVldCtrl**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-3 | -- | 00000 | r | reserved |
| 2 | VldIrq | 0 resetsystem | r | VLD irq |
| 1 | VldValid | 0 resetsystem | r | VLD valid result |
| 0 | VldEn | 0 resetsystem | r w | VLD enable |

**Table 12.3: RegVldStat**

# 13 Power-on reset

The Power-on reset (POR) block fixes the conditions for the beginning of the power-on sequence (see SYSTEM chapter).

At start-up of the circuit, the POR block generates a reset signal during $t_{on}$. In normal mode if the voltage supply falls below a critical value, the reset signal is activated.

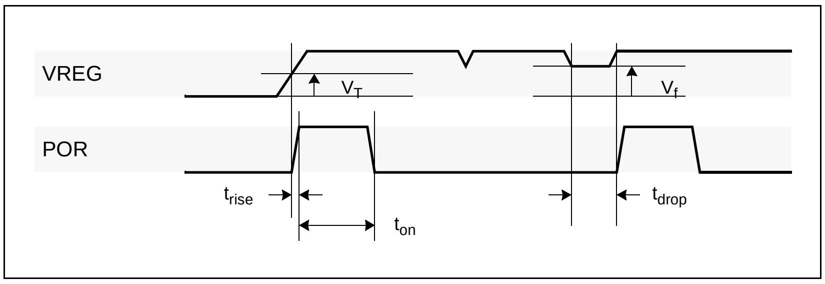


Figure 13.1: reset conditions

| symbol | description | min | typ | max | unit | comments |
|---|---|---|---|---|---|---|
| $t_{on}$ | reset duration | 5 | | 20 | ms | 3 |
| $t_{drop}$ | drop duration | 10 | | | µs | $V_f$ = 0.4 VREG |
| $V_T$ | start threshold voltage | 0.7 | | 1.1 | V | 1 |
| Vdd_sl_M | voltage slope for activation of MTP versions | 20 | | | V/ms | 2 |
| Vdd_sl_R | voltage slope for activation of ROM versions | 0.25 | | | V/ms | 2 |

**Table 13.1: POR specifications**

**Note:** 1) Threshold voltage for the power-on reset does not imply that this voltage is sufficient for correct operation of the CPU. This has to be checked with the voltage level detector during operation of the device, and to be ensured by design for the start-up of the chip.

**Note:** 2) The Vdd_sl defines a minimum slope on Vreg. The power-on reset behaviour of the circuit is not guaranteed if this slope is too slow as system could start before program memory has sufficient voltage to insure correct behaviour. In such a case, a delay has to be built using the RESET pin.

**Note:** 3) The power-on sequence (see SYSTEM chapter) follows the reset duration.

# 14 Acquisition chain

## 14.1 Introduction

The Acquisition Chain is made of an Analog Multiplexer (AMUX) entering a Programmable Gain Amplifier (PGA) followed by an Analog-to-Digital Converter (ADC). It is intended to sense a differential voltage at its input, which can be connected for instance to a Wheatstone-bridge type resistive sensor, and to deliver a signed 16 bits-wide word in 2's complement. A busy signaland a maskable interrupt inform the CPU about the state of the conversion.

The PGA can provide both amplification and offset compensation. The ADC converter is oversampled and incremental, i.e. it uses several input samples to generate one output data and it is reset before each conversion. Its oversampling ratio can be programmed as a power of 2 to choose the appropriate trade-off between conversion duration and resolution. Input swiching is immediate when PGA is off, it requires a short delay when PGA is on.

The block can be operated in two ways: "on request" upon a Start Conversion signal or "continuously running".

## 14.2 Block diagram

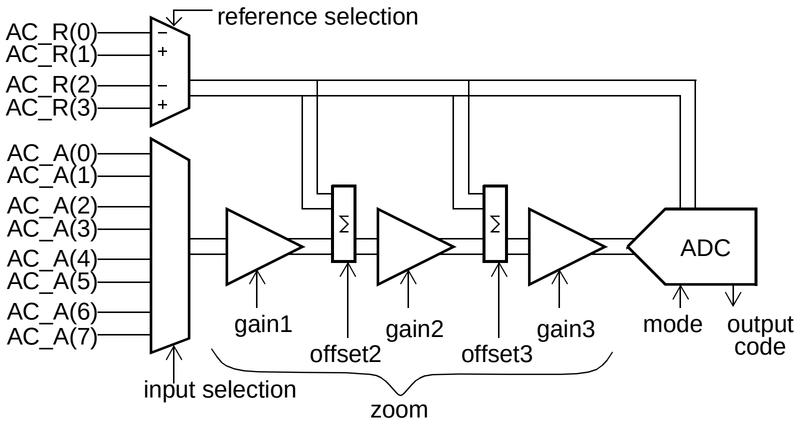


Figure 14.1: Acquisition channel block diagram with ZoomingADC™

Figure 14.1 shows the general block diagram of the peripheral. The PGA selects a combination of input signals, modulates the input voltage and amplifies it through 1 to 3 stages, 2 of these providing offset compensation. Each amplifier can be bypassed. The ADC delivers the output in 2's complement. The acquisition channel also includes a control block that manages all communication with the CPU, sets the configuration of the peripheral and implements the different test modes.

## 14.3 Input signal multiplexing

There are 8 inputs named AC_A[0] to AC_A[7]. Inputs can be used either as four differential channels (Vin1=AC_A[1]-AC_A[0], …, Vin4=AC_A[7]-AC_A[6]) or AC_A[0] can be used as a common reference, providing 7 signal paths (AC_A[1]-AC_A[0], …, AC_A[7]-AC_A[0]), all referred to AC_A[0]. Default input is Vin1.

On top of these settings, inputs can be crossed or not. All multiplexing combinations are summarised in the following table (see Table 14.1) :

| uni/bi-polar | sign | channel selection | | | selected differential input | |
|---|---|---|---|---|---|---|
| AMUX(4) | AMUX(3) | AMUX(2) | AMUX(1) | AMUX(0) | VIN- | VIN+ |
| 0 | 0 | unused | 0 | 0 | A(0) | A(1) |
| | | | 0 | 1 | A(2) | A(3) |
| | | | 1 | 0 | A(4) | A(5) |
| | | | 1 | 1 | A(6) | A(7) |
| | 1 | unused | 0 | 0 | A(1) | A(0) |
| | | | 0 | 1 | A(3) | A(2) |
| | | | 1 | 0 | A(5) | A(4) |
| | | | 1 | 1 | A(7) | A(6) |
| 1 | 0 | 0 | 0 | 0 | A(0) | A(0) |
| | | 0 | 0 | 1 | A(0) | A(1) |
| | | 0 | 1 | 0 | A(0) | A(2) |
| | | 0 | 1 | 1 | A(0) | A(3) |
| | | 1 | 0 | 0 | A(0) | A(4) |
| | | 1 | 0 | 1 | A(0) | A(5) |
| | | 1 | 1 | 0 | A(0) | A(6) |
| | | 1 | 1 | 1 | A(0) | A(7) |
| | 1 | 0 | 0 | 0 | A(0) | A(0) |
| | | 0 | 0 | 1 | A(1) | A(0) |
| | | 0 | 1 | 0 | A(2) | A(0) |
| | | 0 | 1 | 1 | A(3) | A(0) |
| | | 1 | 0 | 0 | A(4) | A(0) |
| | | 1 | 0 | 1 | A(5) | A(0) |
| | | 1 | 1 | 0 | A(6) | A(0) |
| | | 1 | 1 | 1 | A(7) | A(0) |

**Table 14.1: AMUX selection**

## 14.4 Input reference multiplexing

One must select one of two differential signal as reference signal (Vref1=AC_R[1]-AC_R[0], Vref2=AC_R[3]-AC_R[2]). Default is Vref1.

## 14.5 Amplifier chain

The 3 stages transfer functions are:

```
VD3 = GD3·VD2 - GDoff3·Vref
VD2 = GD2·VD1 - GDoff2·Vref
      VD1 = GD1·Vin
```

| | |
|---|---|
| where: | Vin=Selected input voltage |
| | Vref=Selected reference voltage |
| | VD1=Differential voltage at the output of first amplifier |
| | VD2=Differential voltage at the output of second amplifier |
| | VD3=Differential voltage at the output of third amplifier |
| | GD1=Differential gain of stage 1 |
| | GD2=Differential gain of stage 2 |
| | GD3=Differential gain of stage 3 |
| | GDoff2= Offset gain of stage 2 |
| | GDoff3=Offset gain of stage 3 |

and therefore the whole transfer function is:

```
Vout of PGA = VD3 = GD3·GD2·GD1·Vin - (GDoff3 + GDoff2·GD3)·Vref
```

**Note:** As the offset compensation is realized together with the amplification on the same summing node, the only voltages that have to stay within the supplies are Vref and the VD$_i$ · GD$_i$ · VD$_{i-1}$ and GDoff$_i$ · Vref can be larger without any saturation.

**Note:** All stages use a fully differential architecture and all gain and offset settings are realized with ratios of capacitors.

**Note:** As the ADC also provides a gain (2 nominal), the total chain transfer function is:

$$Data\_out = 2 \cdot GD3 \cdot GD2 \cdot GD1 \cdot \frac{Vin}{Vref} - 2 \cdot (GDoff3 + GDoff2 \cdot GD3)$$

Each stage is called PGAi. Features of these stages are:

- Gain can be chosen between 1 and 10 (between 0 and 10 for PGA3)
- Offset can be compensated for in PGA2 (a little) and in PGA3 (to a large extent)
- Granularity of settings is rough for PGA1, medium for PGA2, fine for PGA3
- Zero, one, two or three of the PGA stages can be used.

A functional example of one of the stages is given on Figure 14.2.

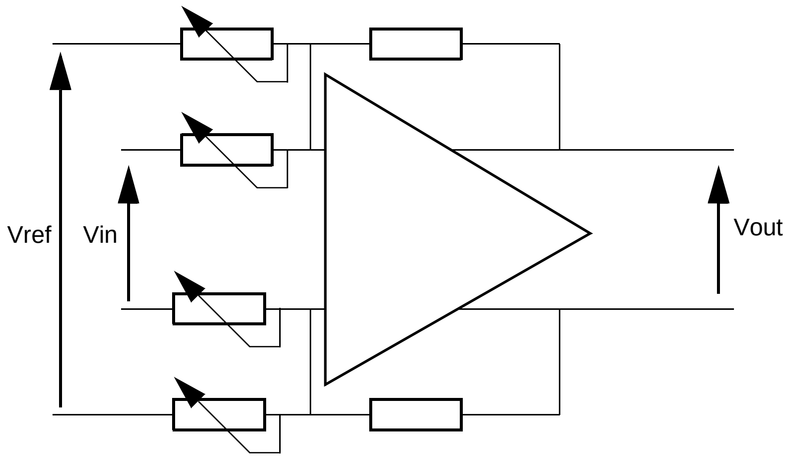


Figure 14.2: PGA stage principle implementation

### 14.5.1 PGA 1

| symbol | description | min | typ | max | unit | Comments |
|---|---|---|---|---|---|---|
| GD_preci | Precision on gain settings | -5 | | +5 | % | |
| GD_TC | Temperature dependency of gain settings | -5 | | +5 | ppm/°C | |
| fs | input sampling frequency | 5 | | 512 | kHz | |
| Zin1 | Input impedance | 150 | | | kΩ | 1 |
| Zin1p | Input impedance for gain 1 | 1500 | | | kΩ | 1 |
| VN1 | Input referred noise | | | 18 | nV/ sqrt(Hz) | 2 |

**Table 14.2: PGA1 Performances**

**Note:** 1) Measured with block connected to inputs through AMUX block. Normalized input sampling frequency for input impedance is 512 kHz. This figure has to be multiplied by 2 for fs = 256 kHz and 4 for fs = 128 kHz.

**Note:** 2) Input referred rms noise is 10 uV per input sample. This corresponds to 18 nV/sqrt(Hz) for fs = 512 kHz.

### 14.5.2 PGA2

| sym | description | min | typ | max | unit | Comments |
|---|---|---|---|---|---|---|
| GDoff2 | PGA2 Offset Gain | -1 | | 1 | FS | |
| GDoff2_step | GDoff2(code+1) – GDoff2(code) | 0.18 | 0.2 | 0.22 | - | |

**Table 14.3: PGA2 Performances**

| sym | description | min | typ | max | unit | Comments |
|---|---|---|---|---|---|---|
| GD_preci | Precision on gain settings | -5 | | +5 | % | valid for GD2 and GDoff2 |
| GD_TC | Temperature dependency of gain settings | -5 | | +5 | ppm/°C | |
| fs | Input sampling frequency | 5 | | 512 | kHz | |
| Zin2 | Input impedance | 150 | | | kΩ | 1 |
| VN2 | Input referred noise | | | 36 | nV/ sqrt(Hz) | 2 |

**Table 14.3: PGA2 Performances**

**Note:** 1) Measured with block connected to inputs through AMUX block. Normalized input sampling frequency for input impedance is 512 kHz. This figure has to be multiplied by 2 for fs = 256 kHz and 4 for fs = 128 kHz.
**Note:** 2) Input referred rms noise is 26uV per sample.This corresponds to 36 nV/sqrt(Hz) max for fs = 512 kHz.

### 14.5.3 PGA3

| sym | description | min | typ | max | unit | Comments |
|---|---|---|---|---|---|---|
| GDoff3 | PGA3 Offset Gain | -5 | | 5 | FS | |
| GD3_step | GD3(code+1) - GD3(code) | 0.075 | 0.08 | 0.085 | - | |
| GDoff3_step | GDoff2(code+1) – GDoff2(code) | 0.075 | 0.08 | 0.085 | - | |
| GD_preci | Precision on gain settings | -5 | | +5 | % | valid for GD3 and GDoff3 |
| GD_TC | Temperature dependency of gain settings | -5 | | +5 | ppm/°C | |
| fs | Input sampling frequency | 5 | | 512 | kHz | |
| Zin3 | Input impedance | 150 | | | kΩ | 1 |
| VN3 | Input referred noise | | | 36 | nV/ sqrt(Hz) | 2 |

**Table 14.4: PGA3 Performances**

**Note:** 1) Measured with block connected to inputs through AMUX block. Normalized input sampling frequency for input impedance is 512 kHz. This figure has to be multiplied by 2 for fs = 256 kHz and 4 for fs = 128 kHz.
**Note:** 2) Input referred rms noise is 26uV per sample. This corresponds to 36 nV/sqrt(Hz) max for fs = 512 kHz.

## 14.6 ADC

### 14.6.1 Input-Output relation

The ADC block is used to convert the differential input signal into a 16 bits 2’s complement output format. The output code corresponds to the ratio:

$$\text{Output code} = \frac{\text{Vin}}{\text{Vref}} \cdot \frac{\text{smax} + 1}{\text{smax}}$$

smax being the number of samples used to generate one output sample per elementary conversion. smax is set by **OSR** on **RegACCfg0**.

Vref can be selected up to the power supply rails and must be positive. The corresponding 2’s complement output code is given in hexadecimal notation by 8000 (negative full scale) and 7FFF (positive full scale). Code outside the range are saturated to the closest full scale value.

**Note:** The output code is normalized into a 16 bits format by shifting the result left or right accordingly.

### 14.6.2 Operation mode

The mode can be either “on request” or “continuously running”.

In the “on request” mode, after a request, an initialization sequence is performed, then an algorithm is applied and an output code is produced. The converter is idle until the next request.

In the “continuously running“ mode, an internal conversion request is generated each time a conversion is finished, so that the converter is never idle. The output code is updated at a fixed rate corresponding to 1/Tout, with Tout being the conversion time.

## 14.6.3 Conversion sequence

The whole conversion sequence is basically made of an initialisation, a set of $N_{umconv}$ elementary incremental conversions and finally a termination phase (**$N_{umCONV}$** is set by the 2 **NELCONV** bits on **RegACCfg0**). The result is a mean of the results of the elementary conversions.

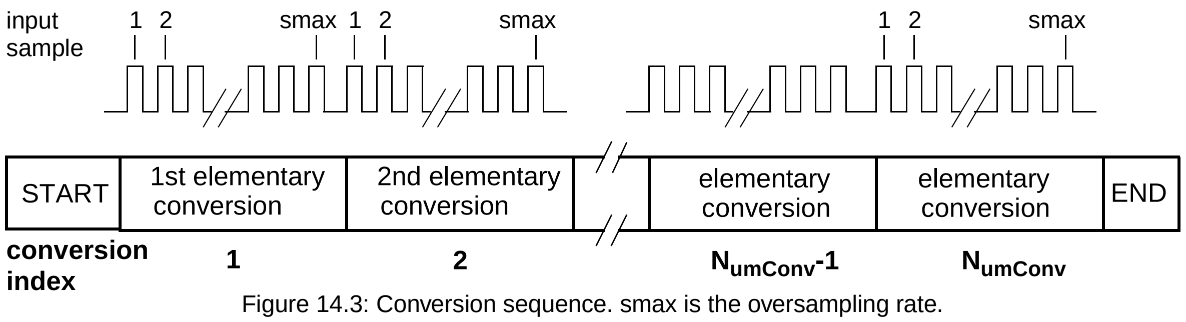


Figure 14.3: Conversion sequence. smax is the oversampling rate.

**Note:** **$N_{umCONV}$** elementary conversions are performed, each elementary conversion being made of smax+1 input samples.

$$N_{umConv} = 2^{NELCONV}$$

$$smax = 8 * 2^{OSR}$$

During the elementary conversions, the operation of the converter is the same as in a sigma delta modulator. During one conversion sequence, the elementary conversions are alternatively performed with direct and crossed PGA-ADC differential inputs, so that when two elementary conversions or more are performed, the offset of the converter is cancelled.

**Note:** The sizing of the decimation filter puts some limits on the total number of conversions and it is not possible to combine the maximum number of elementary conversions with the maximum oversampling (see the Nelconv*smax specification).

Some additional clock cycles ($N_{INIT}$+$N_{END}$) clock cycles are required to initiate and terminate the conversion properly.

## 14.6.4 Conversion duration

The conversion time is given by :

$$T_{OUT} = (N_{umConv} * Smax + N_{INIT} + N_{END}) / fs$$

## 14.6.5 Resolution

As far as it is not limited by thermal noise and internal registers width, the resolution is given by :

$$\text{Resolution (in bits)} = 6 + 2 * OSR + NELCONV$$

## 14.6.6 ADC performances

| sym | description | min | typ | max | unit | Comments |
|---|---|---|---|---|---|---|
| VINR | Input range | -0.5 | | 0.5 | Vref | |

**Table 14.5: ADC Performances**

| sym | description | min | typ | max | unit | Comments |
|---|---|---|---|---|---|---|
| Resol | Resolution | 12 | | | bits | 1 |
| NResol | Numerical resolution | | | 16 | bits | 4 |
| INL | Integral non-linearity | | | 4 | LSB | 1,3, LSB at 12bits |
| TC | Temperature dependency of sensitivity | -5 | | +5 | ppm/°C | |
| fs | sampling frequency | 10 | | 512 | kHz | |
| smax | Oversampling Ratio | 8 | | 1024 | - | 2 |
| $N_{UMCONV}$ | Number of elementary conversions | 1 | | 8 | - | 2 |
| Ninit | Number of periods for incremental conversion initialization | | | 5 | - | |
| Nend | Number of periods for incremental conversion termination | | | 5 | - | |

**Table 14.5: ADC Performances**

**Note:** 1) Resolution specification also includes thermal noise and differential non-linearity (DNL) for a reference signal of 2.4 V. It is defined for default operating mode ( See “Default operation mode (not yet implemented)” on page 108.)

**Note:** 2) Only powers of 2

**Note:** 3) INL is defined as the deviation of the DC transfer curve from the best fit straight line. This specification holds over the full scale.

**Note:** 4) NResol is the maximal readable resolution of the digital filter. Input noise may be higher than NResol.

## 14.7 Control part

### 14.7.1 Starting a convertion

A conversion is started each time **START** or **DEF** is set.

PGAs are reset after each writing operation to registers **RegACCfg1-5**. The ADCs must be started after a PGA common-mode stabilisation delay. This is done by writing bit **START** several cycles after PGA settings modification. Delay between PGA start and ADC start should be equivalent to **smax** number of cycles. (This delay does not apply to conversions made without the PGAs)

### 14.7.2 Clocks generation

The clock of the acquisition path is derived from the RC oscillator clock.

```
fs = psck/4
```

psck can be chosen among four prescaler clocks (bit **FIN** of **RegACCfg2**), see Table 14.7.

### 14.7.3 Default operation mode (not yet implemented)

The **DEF** bit (**RegACCfg5**) allows the use of the ADC in a default mode without any gain nor offset adjustment (see values in the right column of Table 14.7). The only action to launch the ADC default conversion is to write a ‘b01AAAAAV’ in **RegACCfg5**, AAAAA being the **AMUX** selection and V the **VMUX** selection. This default mode is used in specifications to define resolution and INL.

### 14.7.4 Registers

Eight registers control this peripheral. Two registers are for the data output, six for peripheral general set-up. Registers are defined in Table 14.6 and Table 14.7.

| register | data | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| RegACOutLSB | ADC_OUT_L | | | | | | | |
| RegACOutMSB | ADC_OUT_H | | | | | | | |
| RegACCfg0 | START | NELCONV | | OSR | | | CONT | reserved |
| RegACCfg1 | IB_AMP_ADC | | IB_AMP_PGA | | ENABLE | | | |
| RegACCfg2 | FIN | | PGA2_GAIN | | PGA2_OFF | | | |
| RegACCfg3 | PGA1_GAIN | PGA3_GAIN | | | | | | |
| RegACCfg4 | reserved | PGA3_OFF | | | | | | |

**Table 14.6: Peripheral register memory map**

| register | data | | | |
|---|---|---|---|---|
| RegACCfg5 | BUSY | DEF | AMUX | VMUX |

Table 14.6: Peripheral register memory map

| Name | Register | rm | description | Default (reset and DEF mode) |
|---|---|---|---|---|
| ADC_OUT(15:0) | RegACOutLSB RegACOutMSB | r | data output<br>data is shifted left for having the MSB on the MSB of RegACOutMSB for any resolution. Therefore LSBs may be fixed at 0 ifdigital resolution is below 16 bits. | 0000h |
| AMUX(4:0) | RegACCfg5 | wr | Selection of PGA inputs | 00000 (reset only) |
| BUSY | RegACCfg5 | r | ‘1’ : conversion is in progress<br>‘0’ : data is available | 0 |
| CONT | RegACCfg0 | wr | ‘1’ : continuous operation.<br>‘0’ : one shot mode | 0 |
| DEF | RegACCfg5 | wr0 | Default Operation bit (not yet implemented)<br>‘1’: All registers but VMUX and AMUX are reset and default values are used<br>‘0’: Normal operation | N/A |
| ENABLE(3:0) | RegACCfg1 | wr | bit3 : PGA3, bit2 : PGA2, bit1 : PGA1, bit 0 : ADC<br>If a bit is ‘1’, the block is powered. If not, the block is switched off and all internal digital signals are reset.<br>Concerning the PGAs, ENABLE=0 means also that inputs and outputs are wired together and that the acquisition chain is not perturbed by the block. | 0000 (reset) 0001 (DEF mode) |
| FIN(1:0) | RegACCfg2 | wr | ‘00’ : RC is used as master clock (psck)<br>‘01’ : RC / 2 is used as master clock (psck)<br>‘10’ : RC / 8 is used as master clock (psck)<br>‘11’ : RC / 32 is used as master clock (psck)<br>Rem: do not select a clock that results in fs faster than 512 kHz. | 00 |
| IB_AMP_PGA(1:0) | RegACCfg1 | wr | PGA amplifiers biasing current reduction factor<br>‘00’ : current = 0.25 nominal current<br>‘01’ : current = 0.5 nominal current<br>‘10’ : current = 0.75 nominal current<br>‘11’ : current = nominal current | 11 |
| IB_AMP_ADC(1:0) | RegACCfg1 | wr | ADC amplifiers biasing current reduction factor.<br>Tuning identical to IB_AMP_PGA | 11 |
| NELCONV(1:0) | RegACCfg0 | wr | Number of elementary conversions setting<br>‘00’ : 1 conversion, ‘01’ : 2 conversions<br>‘10’ : 4 conversions, ‘11’ : 8 conversions | 01 |
| OSR(2:0) | RegACCfg0 | wr | OverSampling Ratio setting. Defined as fs/fout. smax = $8*2^{OSR(2:0)}$.<br>‘000’ : oversampling = 8, ..., ‘111’ : oversampling = 1024 | 010 |
| PGA1_GAIN | RegACCfg3 | wr | signal gain of first PGA stage (GD1)<br>‘1’ : nominal gain is 10.<br>‘0’ : nominal gain is 1 | 0 |
| PGA2_GAIN(1:0) | RegACCfg2 | wr | signal gain of second PGA stage (GD2)<br>‘11’ : nominal gain is 10<br>‘10’ : nominal gain is 5<br>‘01’ : nominal gain is 2<br>‘00’ : nominal gain is 1 | 00 |
| PGA2_OFF(3:0) | RegACCfg2 | wr | offset gain of second PGA stage (GDoff2)<br>bit 3: offset sign (‘0’ : GDoff2 > 0, ‘1’ : GDoff2 < 0)<br>bits (2:0) : offset amplitude<br>‘01x1’ : GDoff2 = 1.0 nominal, ‘0100’ : GDoff2 = 0.8 nominal,<br>‘0011’ : GDoff2 = 0.6 nominal, ..., ‘0000’ : GDoff2 = 0.0 nominal,<br>‘1001’’ : GDoff2 = -0.2 nominal, ..., ‘11x1’ : GDoff2 = -1.0 nominal | 0000 |
| PGA3_GAIN(6:0) | RegACCfg3 | wr | signal gain of third stage (GD3)<br>GD3 = 0.08*PGA3_GAIN(6:0)<br>Nominal values : 0 (‘000 0000’), ..., 10 (‘111 1000’) | 000 1100 |
| PGA3_OFF(6:0) | RegACCfg4 | wr | offset gain of third stage (GDoff3)<br>bit 6: offset sign (‘0’ : GDoff3 > 0, ‘1’ : GDoff3 < 0)<br>GDoff3 = 0.08*PGA3_OFF(5:0), maximum = 5.04<br>Nominal values : -5.04 (‘111 1111’), 0 (‘x00 0000’), +5.04 (‘011 1111’) | 000 0000 |
| START | RegACCfg0 | wr0 | writing a “1” in START bit restarts the ADC. It does not affect the PGAs. | 0 |
| TEST | RegACCfg0 | | reserved | 0 |
| VMUX | RegACCfg5 | wr | VREF selection multiplexer<br>‘0’ : VREF0 is used, ‘1’ : VREF1 is used | 0 (reset only) |

Table 14.7: Peripheral register memory map, bits description

## 14.8 Acquisition of a sample

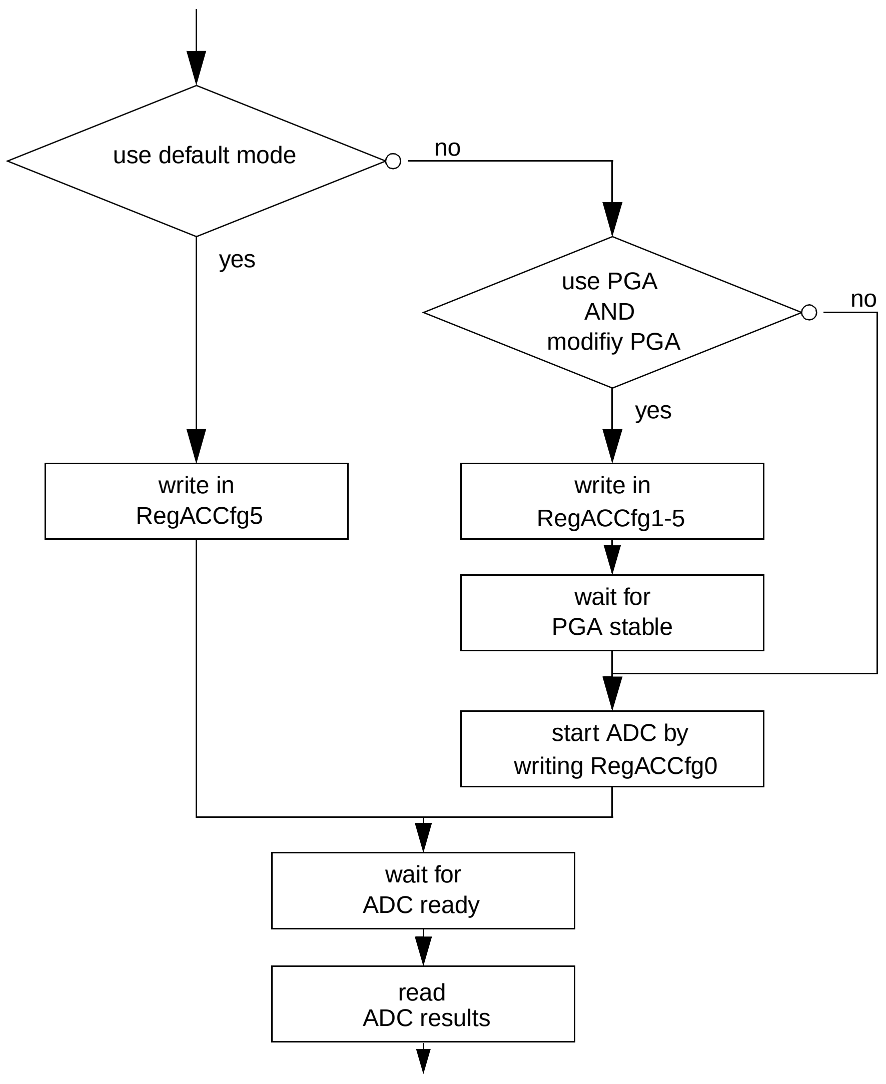


Figure 14.4: Acquisition flow

# 15 Analog outputs

The PWM DACs available in all XE8000 products are described in the TIMERS chapter.

The XE88LC05 has two additional digital to analog converters (DAC)s: a signal DAC able to pass a 4 kHz signal with 10 bits precision (16 bits resolution at low frequency), and a bias DAC able to output 10 mA to bias a resistive bridge sensor on 8 bits resolution. Both are DACs formed from a generic DAC and an amplifier. This makes possible current and voltage drive and lets full freedom for the user to choose the preferred filtering scheme.

Please refer to XEMICS application note AN8000.03 for more hints on how to use the XE88LC05 DACs.

## 15.1 Signal DAC

### 15.1.1 Application

The DAC signal block described in this document converts a digital signal into an analog output signal (voltage output or 4-20mA loop).

### 15.1.2 Typical external components

External components are needed for the filtering of the generic DAC output. These external components depend on the characteristics the customer wants to obtain. Some application examples are shown in the application note AN8000.03.

### 15.1.3 Block diagram

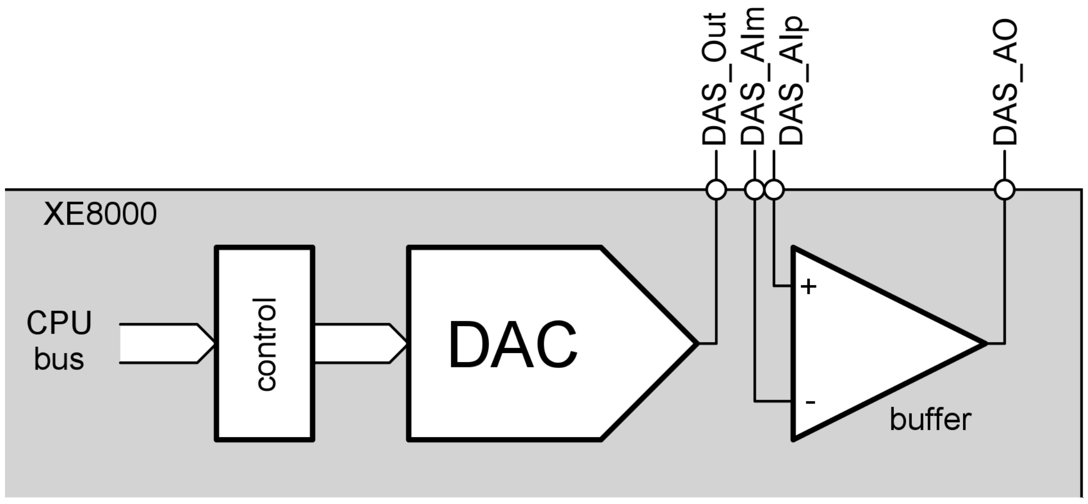


Figure 15.1: General block diagram

Figure 15.1 shows the general block diagram of the peripheral. It consists of a control block that manages all communication with the CPU, sets the configuration of the peripheral and implements the different test modes. The DAC converts the digital data in a PWM output signal. A completely independent amplifier is added. This allows the user to build a low impedance voltage output after the (external, probably passive) filter. It also allows to build a 4-20mA loop. (See amplifier section below for examples)

### 15.1.4 The generic DAC

The generic DAC block consists of two major parts: the noise shaper (sigma-delta modulator) and the PWM modulator as shown in Figure 15.2.

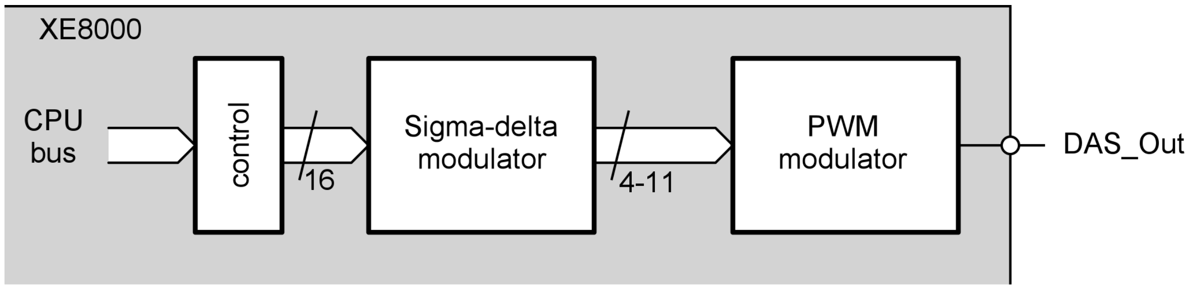


Figure 15.2: The DAC signal structure

The DAC word width at the input is 16 bit. If the word is narrower, 0's have to be added after the LSB to fill the 16 bits. To maintain maximum flexibility, the order of the noise shaper and the resolution of the PWM modulation are programmable by writing the codes **code_lmax** and **ns_order** to the configuration register. The possible noise shaper order is 0, 1 or 2. With noise shaper order 0, the generic DAC is a conventional PWM DAC which resolution can be set between 4 and 11 bits. Higher resolutions are reached with higher orders of the noise shaper.

The role of the sigma-delta modulator is to vary the code sent to the PWM modulator, so that its mean value is exactly equal to the code set in the DAC input.

Regular way of using the DAC is to set **code_lmax** for 4 bits (000) and **ns_order** to 2 (10).

DACs and filter settings are explained in application note AN8000.03 “XE88LC05 DACs usage”.

### 15.1.5 The amplifier

The amplifier can be used in several configurations. Therefore, it is not connected internally.

| sym | description | min | typ | max | unit | Comments |
|---|---|---|---|---|---|---|
| gain | gain at DC | 80 | | | dB | 1 |
| GBW0 | gain bandwidth product | 25 | | | kHz | 7 |
| cl0 | capacitive load | | | 5 | nF | 7 |
| GBW1 | gain bandwidth product | 125 | | | kHz | 8 |
| cl1 | capacitive load | | | 200 | pF | 8 |
| fm | phase margin | 55 | | | ° | 9 |
| rl | resistive load | 5 | | | kohm | 6 |
| SR | slew rate | 10 | | | kV/s | 10 |
| CMR | common mode input range | vss-0.2 | | vdd-1.2 | V | 2 |
| OR | output range | vss+0.2 | | vdd-0.2 | V | |
| voff | offset | | | ±5 | mV | |
| CMRR | common mode rejection | 60 | | | dB | 3 |
| noise | integrated input noise | | | 100 | uVrms | |
| PSRR | power supply rejection ratio | 20 | | | dB | 4 |
| ibias | quiescent bias current | | 200 | 500 | uA | |
| ioff | off current | | | 1 | uA | |

**Table 15.1: DAC signal amplifier performances**

**Note:** 1. For the minimal resistive load and the maximal capacitive load
**Note:** 2. The amplifier common mode is vss in the 4-20mA loop (Figure 6).
**Note:** 3. At DC
**Note:** 4. At DC. Only a low rejection ratio is needed since the DAC output refers directly to the power supplies.
**Note:** 6. Short circuit protection at ~5mA.
**Note:** 7. GBW when the maximal load is cl0 and with the bit BW=0
**Note:** 8. GBW when the maximal load is cl1 and with the bit BW=1
**Note:** 9. In both cases BW=0 and BW=1 for the maximal capacitive load and the minimal resistive load.

**Note:** 10. For maximal load cl0, BW=0 and maximal resistive load rl

### 15.1.6 Signal DAC registers

| register name | address (hex) | comments |
|---|---|---|
| RegDasInLsb | H0068 | input code (LSB) |
| RegDasInMsb | H0069 | input code (MSB) |
| RegDasCfg0 | H006A | DAC settings |
| RegDasCfg1 | H006B | |

**Table 15.2: Signal DAC registers**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-6 | NsOrder | 0 resetsystem | r w | modulator order |
| 5-3 | CodeImax | 0 resetsystem | r w | PWM resolution |
| 2-1 | Enable | 0 resetsystem | r w | DAC and buffer enable |
| 0 | Fin | 0 resetsystem | r w | input frequency selection |

**Table 15.3: RegDasCfg0**

| bit | name | reset | rw | description |
|---|---|---|---|---|
| 7-2 | reserved | 0 resetsystem | r w | |
| 1 | BW | 0 resetsystem | r w | output bandwidth selection |
| 0 | Inv | 0 resetsystem | r w | output polarity selection |

**Table 15.4: RegDasCfg1**

The CPU communicates with the peripheral through registers. Two registers are needed for the data input, two are needed for the set-up of the peripheral. The registers are defined in Table 15.2. The data in the set-up registers code for the noise shaper order (ns_order), the resolution of the PWM pulse modulation (code_lmax), the peripheral status (enable), clock input frequency (fin), if the PWM pulse is active low or high (inv) and the amplifier bandwidth.

The input data will have to be resynchronized with respect to the peripheral clock. In order to guarantee the synchronization of the MSB and LSB part of the input data, a validity flag is reset when the CPU is writing in the LSB register and set again when the CPU is writing to the MSB register. The contents of the registers is not copied to the DAC while the validity flag is reset. This means that the CPU always has to write the LSB register before writing the MSB register.

The different set-up words (set-up register definition in Table 15.2) are coded as indicated in the tables below.

| ns_order(1:0) | Noise shaping order |
|---|---|
| 00 | 0 (PWM operation) |
| 01 | 1 |
| 1x | 2 |

**Table 15.5: Noise shaping setting**

| code_lmax | m (PWM resolution in bits) |
|---|---|
| 000 | 4 |
| 001 | 5 |
| 010 | 6 |
| 011 | 7 |
| 100 | 8 |
| 101 | 9 |
| 110 | 10 |
| 111 | 11 |

**Table 15.6: PWM setting**

| enable(1:0) | Peripheral status |
|---|---|
| 00 | DAC: switched off; Amplifier switched off |

**Table 15.7: DAC status**

| enable(1:0) | Peripheral status |
|---|---|
| 01 | DAC: normal operation; Amplifier: switched off |
| 10 | DAC: switched off; Amplifier: normal operation |
| 11 | normal operation |

**Table 15.7: DAC status**

The **Fin** bit allows the choice between two clock inputs that can be connected in two different places of the prescaler of the XE8000. The control logic has to guarantee correct switching from one to the other.

| fin | used clock input |
|---|---|
| 0 | RC clock |
| 1 | RC clock div 2 |

**Table 15.8: Clock setting**

Finally, the inv bit indicates if the PWM output pulse is active low or active high. This allows the use of the output amplifier in an inverting or not inverting configuration.

| inv | PWM pulse |
|---|---|
| 0 | active high |
| 1 | active low |

**Table 15.9: PWM polarity**

## 15.2 Bias DAC

### 15.2.1 Application

The bias DAC block converts a digital signal into an analog output signal in DC. It can be used to bias resistive sensors. In some cases it could also be used to do a rough offset calibration of a resistive bridge.

### 15.2.2 Typical external components

No other external devices are needed in case of voltage controlled bridge bias. An additional resistor is needed for current controlled bridge bias.

### 15.2.3 Block diagram

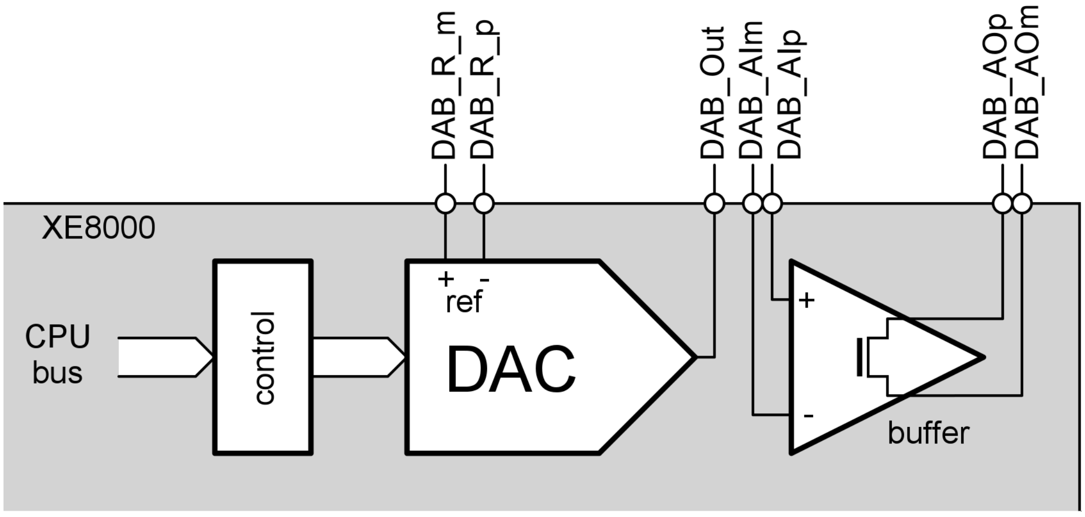


Figure 15.3: General block diagram of the bias DAC

Figure 15.3 shows the general block diagram of the peripheral. It consists of a control block that manages all communication with the CPU, sets the configuration of the peripheral and implements the different test modes. The DAC converts the digital data in an analog output signal. An amplifier is added in order to be able to deliver large currents.

### 15.2.4 The DAC

The DAC convertor is a resistive divider connected between pads DAB_R_m and DAB_R_p.

| sym | description | min | typ | max | unit | Comments |
|---|---|---|---|---|---|---|
| wda | number of input bits | | 8 | | bits | |
| tstep | step response | | | 100 | ms | 1 |
| range | DAC output range | DAB_R_m | | DAB_R_p | | 2 |

**Table 15.10: DAC performances**

**Note:** 1) Time to reach the final value within 5%.
**Note:** 2) In most cases DAB_R_m will be connected to vss and DAB_R_p to vdd.

### 15.2.5 The amplifier

The amplifier can be used in several configurations as for biasing a bridge in voltage or current. Application examples are given in application note AN8000.03.

| sym | description | min | typ | max | unit | Comments |
|---|---|---|---|---|---|---|
| gain | gain at DC | 60 | | | dB | 1 |
| GBW | gain bandwidth product | 100 | | | Hz | 1 |

**Table 15.11: Amplifier performances**

| sym | description | min | typ | max | unit | Comments |
|---|---|---|---|---|---|---|
| fm | phase margin | 60 | | | ° | 1 |
| rl | resistive load | 300 | | 100000 | ohm | 5,6 |
| cl | capacitive load | | | 1 | nF | |
| CMR | common mode input range | vss | | vdd | V | |
| OR | output range | vss+0.2 | | vdd-0.2 | V | 3,4 |
| outp vr | outp pin voltage range | vss+2.3 | | vdd | V | |
| voff | offset | | | ±10 | mV | |
| noise | integrated input noise | | | 100 | uVrms | |
| isourc | max source current | | | 10 | mA | 5 |
| PSRR | power supply rejection ratio | 40 | | | dB | 2 |
| ibias | quiescent bias current | | 2 | 5 | uA | 6 |
| ioff | off current | | | 1 | uA | |

**Table 15.11: Amplifier performances**

**Note:** 1) For all possible combinations of resistive load and capacitive load.
**Note:** 2) At DC.
**Note:** 3) For voltage controlled bias control. For current controlled operation the voltage drop on the pMOS output transistor has to be less than 200mV at maximum current.
**Note:** 4) Special analog output pads without series resistor will be needed in order to get the specification. Care has to be taken with the layout so that ESD and latchup specifications can be met.
**Note:** 5) Short circuit protection at ~80mA.
**Note:** 6) This amplifier must be loaded for correct operation. Ibias is without load current.

### 15.2.6 Bias DAC registers

| register name | address (hex) | comments |
|---|---|---|
| RegDab1In | H006C | input code |
| RegDab1Cfg | H006D | DAC settings |

**Table 15.12: Bias DAC registers**

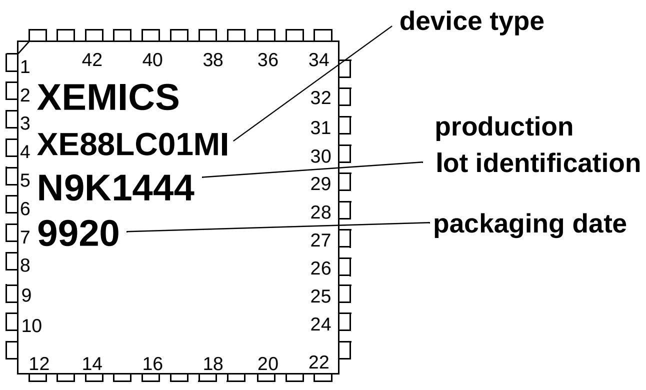
| enable(1:0) | Peripheral status |
|---|---|
| 00 | DAC disabled, amplifier disabled |
| 01 | DAC: normal operation; Amplifier: switched off |
| 10 | DAC: switched off; Amplifier: normal operation |
| 11 | DAC: enabled; Amplifier: enabled |

**Table 15.13: RegDab1Cfg**

# 16 Pin-out, package and electrical specifications

All chips of the XE8000 family are available either as bare die or packaged.

## 16.1 XE88LC01 pin-out



Pinout of the XE88LC01 in TQFP44 package

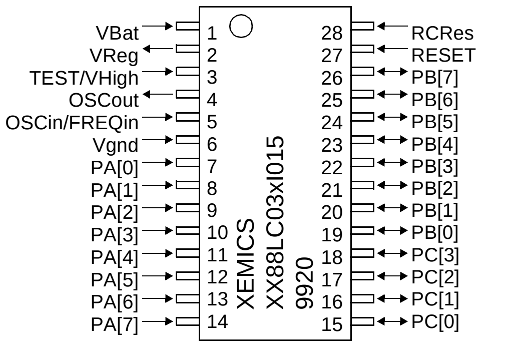
| Pin | | | | Description |
|---|---|---|---|---|
| Position in TQFP44 | Function name | Second function name | Type | |
| 1 | PA(5) | | Input | Input of Port A |
| 2 | PA(6) | | Input | Input of Port A |
| 3 | PA(7) | | Input | Input of Port A |
| 4 | PC(0) | | Input/Output | Input-Output of Port C |
| 5 | PC(1) | | Input/Output | Input-Output of Port C |
| 6 | PC(2) | | Input/Output | Input-Output of Port C |
| 7 | PC(3) | | Input/Output | Input-Output of Port C |
| 8 | PC(4) | | Input/Output | Input-Output of Port C |
| 9 | PC(5) | | Input/Output | Input-Output of Port C |
| 10 | PC(6) | | Input/Output | Input-Output of Port C |
| 11 | PC(7) | | Input/Output | Input-Output of Port C |
| 12 | PB(0) | testout | Input/Output/Analog | Input-Output-Analog of Port B/ Data output for test and MTP programing/ PWM output |
| 13 | PB(1) | | Input/Output/Analog | Input-Output-Analog of Port B/ PWM output |
| 14 | PB(2) | | Input/Output/Analog | Input-Output-Analog of Port B |
| 15 | PB(3) | SOU | Input/Output/Analog | Input-Output-Analog of Port B, Output pin of USRT |
| 16 | PB(4) | SCL | Input/Output/Analog | Input-Output-Analog of Port B/ Clock pin of USRT |
| 17 | PB(5) | SIN | Input/Output/Analog | Input-Output-Analog of Port B/ Data input or input-output pin of USRT |
| 18 | PB(6) | Tx | Input/Output/Analog | Input-Output-Analog of Port B/ Emission pin of UART |
| 19 | PB(7) | Rx | Input/Output/Analog | Input-Output-Analog of Port B/ Reception pin of UART |
| 20 | VPP/TEST | Vhigh | Special | Test mode/High voltage for MTP programing |
| 21 | AC_R(3) | | Analog | Highest potential node for 2nd reference of ADC |
| 22 | AC_R(2) | | Analog | Lowest potential node for 2nd reference of ADC |
| 23 | AC_A(7) | | Analog | ADC input node |

**Pin-out of the XX-XE88LC01 in TQFP44**

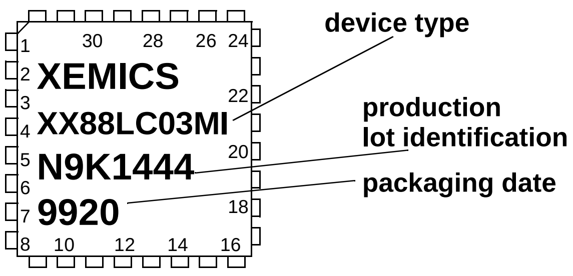
| Pin | | | | Description |
|---|---|---|---|---|
| Position in TQFP44 | Function name | Second function name | Type | |
| 24 | AC_A(6) | | Analog | ADC input node |
| 25 | AC_A(5) | | Analog | ADC input node |
| 26 | AC_A(4) | | Analog | ADC input node |
| 27 | AC_A(3) | | Analog | ADC input node |
| 28 | AC_A(2) | | Analog | ADC input node |
| 29 | AC_A(1) | | Analog | ADC input node |
| 30 | AC_A(0) | | Analog | ADC input node |
| 31 | AC_R(1) | | Analog | Highest potential node for 1st reference of ADC |
| 32 | AC_R(0) | | Analog | Lowest potential node for 1st reference of ADC |
| 33 | VSS | | Power | Negative power supply, connected to substrate |
| 34 | Vbat | | Power | Positive power supply |
| 35 | Vreg | | Analog | Regulated supply |
| 36 | RESET | | Input | Reset pin (active high) |
| 37 | Vmult | | Analog | Pad for optional voltage multiplier capacitor |
| 38 | OscIn | ck_cr | Analog/Input | Connection to Xtal/ CoolRISC clock for test and MTP programing |
| 39 | OscOut | ptck | Analog/Input | Connection to Xtal/ Peripheral clock for test and MTP programing |
| 40 | PA(0) | testin | Input | Input of Port A/ Data input for test and MTP programing/ Counter A input |
| 41 | PA(1) | testck | Input | Input of Port A/ Data clock for test and MTP programing/ Counter B input |
| 42 | PA(2) | | Input | Input of Port A/ Counter C input/ Counter capture input |
| 43 | PA(3) | | Input | Input of Port A/ Counter D input/ Counter capture input |
| 44 | PA(4) | | Input | Input of Port A |

**Pin-out of the XX-XE88LC01 in TQFP44**

## 16.2 XE88LC03 pin-out



Pinout of the XX-XE88LC03 in SOP28 package



Pinout of the XX-XE88LC03 in TQFP32

| Pin | | | | | Description |
|---|---|---|---|---|---|
| Position in SO28 | Position in TQFP32 | Function name | Second function name | Type | |
| 1 | 13 | Vbat | | Power | Positive power supply |
| 2 | 14 | Vreg | | Analog | Regulated supply |
| 3 | 15 | TEST/Vhigh | Vhigh | Special | Test mode/High voltage for MTP programing |
| 4 | 16 | OscOut | ptck | Analog/Input | Connection to Xtal/ Peripheral clock for test and MTP programing |
| 5 | 17 | OscIn | ck_cr | Analog/Input | Connection to Xtal/ CoolRISC clock for test and MTP programing |
| 6 | 18 | Vss | | Power | Negative power supply, connected to substrate |
| 7 | 19 | PA(0) | testin | Input | Input of Port A/ Data input for test and MTP programing/ Counter A input |
| 8 | 20 | PA(1) | testck | Input | Input of Port A/ Data clock for test and MTP programing/ Counter B input |
| 9 | 21 | PA(2) | | Input | Input of Port A/ Counter C input/ Counter capture input |
| 10 | 22 | PA(3) | | Input | Input of Port A/ Counter D input/ Counter capture input |
| 11 | 23 | PA(4) | | Input | Input of Port A |
| 12 | 24 | PA(5) | | Input | Input of Port A |
| 13 | 25 | PA(6) | | Input | Input of Port A |
| 14 | 26 | PA(7) | | Input | Input of Port A |
| 15 | 27 | PC(0) | | Input/Output | Input-Output of Port C |
| 16 | 28 | PC(1) | | Input/Output | Input-Output of Port C |
| 17 | 29 | PC(2) | | Input/Output | Input-Output of Port C |
| 18 | 30 | PC(3) | | Input/Output | Input-Output of Port C |
| | 31 | PC(4) | | Input/Output | Input-Output of Port C |
| | 32 | PC(5) | | Input/Output | Input-Output of Port C |
| | 1 | PC(6) | | Input/Output | Input-Output of Port C |
| | 2 | PC(7) | | Input/Output | Input-Output of Port C |
| 19 | 3 | PB(0) | testout | Input/Output/Analog | Input-Output-Analog of Port B/ Data output for test and MTP programing/ PWM output |
| 20 | 4 | PB(1) | | Input/Output/Analog | Input-Output-Analog of Port B/ PWM output |
| 21 | 5 | PB(2) | | Input/Output/Analog | Input-Output-Analog of Port B |
| 22 | 6 | PB(3) | SOU | Input/Output/Analog | Input-Output-Analog of Port B, Output pin of USRT |
| 23 | 7 | PB(4) | SCL | Input/Output/Analog | Input-Output-Analog of Port B/ Clock pin of USRT |

**Pin-out of the XX-XE88LC03 in SO28 and TQFP32**

Preliminary information

| Pin | | | | | Description |
|---|---|---|---|---|---|
| Position in SO28 | Position in TQFP32 | Function name | Second function name | Type | |
| 24 | 8 | PB(5) | SIN | Input/Output/Analog | Input-Output-Analog of Port B/ Data input or input-output pin of USRT |
| 25 | 9 | PB(6) | Tx | Input/Output/Analog | Input-Output-Analog of Port B/ Emission pin of UART |
| 26 | 10 | PB(7) | Rx | Input/Output/Analog | Input-Output-Analog of Port B/ Reception pin of UART |
| 27 | 11 | RESET | | Input | Reset pin (active high) |
| 28 | 12 | RCRes | | Analog | Optional external resistor for RC oscillator |

**Pin-out of the XX-XE88LC03 in SO28 and TQFP32**

## 16.3 XE88LC05 pin-out

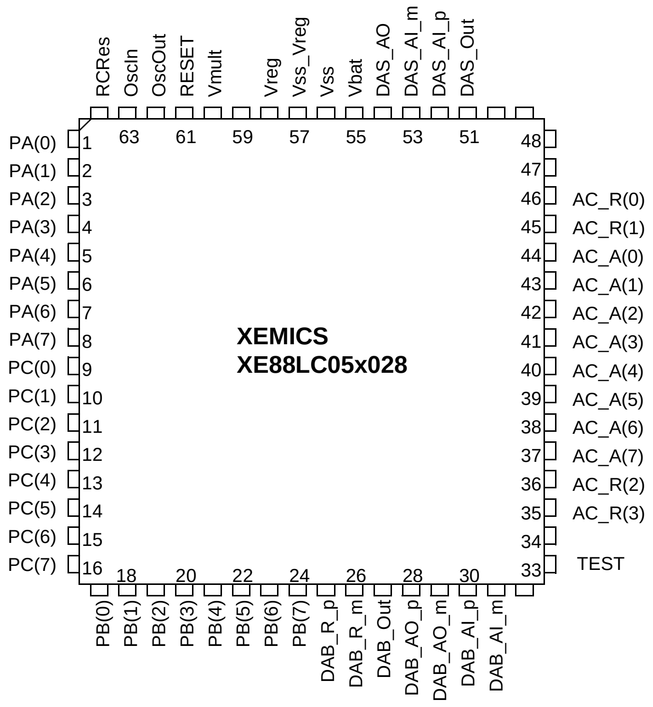


Figure 16.1: Pinout of the XE88LC05 in TQFP64 package

| Pin | | | | Description |
|---|---|---|---|---|
| Position | Function name | Second function name | Type | |
| 1 | PA(0) | testin | Input | Input of Port A/ Data input for test and MTP programing/ Counter A input |
| 2 | PA(1) | testck | Input | Input of Port A/ Data clock for test and MTP programing/ Counter B input |
| 3 | PA(2) | | Input | Input of Port A/ Counter C input/ Counter capture input |
| 4 | PA(3) | | Input | Input of Port A/ Counter D input/ Counter capture input |
| 5 | PA(4) | | Input | Input of Port A |
| 6 | PA(5) | | Input | Input of Port A |
| 7 | PA(6) | | Input | Input of Port A |
| 8 | PA(7) | | Input | Input of Port A |
| 9 | PC(0) | | Input/Output | Input-Output of Port C |
| 10 | PC(1) | | Input/Output | Input-Output of Port C |
| 11 | PC(2) | | Input/Output | Input-Output of Port C |
| 12 | PC(3) | | Input/Output | Input-Output of Port C |
| 13 | PC(4) | | Input/Output | Input-Output of Port C |
| 14 | PC(5) | | Input/Output | Input-Output of Port C |
| 15 | PC(6) | | Input/Output | Input-Output of Port C |
| 16 | PC(7) | | Input/Output | Input-Output of Port C |
| 17 | PB(0) | testout | Input/Output/Analog | Input-Output-Analog of Port B/ Data output for test and MTP programing/ PWM output |
| 18 | PB(1) | | Input/Output/Analog | Input-Output-Analog of Port B/ PWM output |
| 19 | PB(2) | | Input/Output/Analog | Input-Output-Analog of Port B |

**Table 16.1: Pin-out of the XE88LC05 in TQFP64**

Preliminary information

| Pin | | | | Description |
|---|---|---|---|---|
| Position | Function name | Second function name | Type | |
| 20 | PB(3) | SOU | Input/Output/Analog | Input-Output-Analog of Port B,<br>Output pin of USRT |
| 21 | PB(4) | SCL | Input/Output/Analog | Input-Output-Analog of Port B/<br>Clock pin of USRT |
| 22 | PB(5) | SIN | Input/Output/Analog | Input-Output-Analog of Port B/<br>Data input or input-output pin of USRT |
| 23 | PB(6) | Tx | Input/Output/Analog | Input-Output-Analog of Port B/<br>Emission pin of UART |
| 24 | PB(7) | Rx | Input/Output/Analog | Input-Output-Analog of Port B/<br>Reception pin of UART |
| 25 | DAB_R_p | | Analog | Positive reference of bias DAC |
| 26 | DAB_R_m | | Analog | Negative reference of bias DAC |
| 27 | DAB_Out | | Analog | Output of bias DAC |
| 28 | DAB_AO_p | | Analog | Highest potential output of bias DAC buffer |
| 29 | DAB_AO_m | | Analog | Lowest potential output of bias DAC buffer |
| 30 | DAB_AI_p | | Analog | Positive input of bias DAC buffer |
| 31 | DAB_AI_m | | Analog | Negative input of bias DAC buffer |
| 32 | | | Not connected | Spare pin to be connected to negative power supply |
| 33 | TEST/Vhigh | Vhigh | Special | Test mode/High voltage for MTP programing |
| 34 | | | Not connected | Spare pin to be connected to negative power supply |
| 35 | AC_R(3) | | Analog | Highest potential node for 2nd reference of ADC |
| 36 | AC_R(2) | | Analog | Lowest potential node for 2nd reference of ADC |
| 37 | AC_A(7) | | Analog | ADC input node |
| 38 | AC_A(6) | | Analog | ADC input node |
| 39 | AC_A(5) | | Analog | ADC input node |
| 40 | AC_A(4) | | Analog | ADC input node |
| 41 | AC_A(3) | | Analog | ADC input node |
| 42 | AC_A(2) | | Analog | ADC input node |
| 43 | AC_A(1) | | Analog | ADC input node |
| 44 | AC_A(0) | | Analog | ADC input node |
| 45 | AC_R(1) | | Analog | Highest potential node for 1st reference of ADC |
| 46 | AC_R(0) | | Analog | Lowest potential node for 1st reference of ADC |
| 47-50 | | | Not connected | Spare pins to be connected to negative power supply |
| 51 | DAS_Out | | Analog | Output of signal DAC |
| 52 | DAS_AI_p | | Analog | Positive input of signal DAC buffer |
| 53 | DAS_AI_m | | Analog | Negative input of signal DAC buffer |
| 54 | DAS_AO | | Analog | Output of signal DAC buffer |
| 55 | Vbat/VDD | | Power | Positive power supply |
| 56 | Vss | | Power | Negative power supply, connected to substrate |
| 57 | Vss_Reg | | Power | Digital negative power supply, must be equal to Vss |
| 58 | Vreg | | Analog | Regulated supply |
| 59 | | | Not connected | Spare pin to be connected to negative power supply |
| 60 | Vmult | | Analog | Pad for optional voltage multiplier capacitor |
| 61 | RESET | | Input | Reset pin (active high) |
| 62 | OscOut | ptck | Analog/Input | Connection to Xtal/<br>Peripheral clock for test and MTP programing |
| 63 | OscIn | ck_cr | Analog/Input | Connection to Xtal/<br>CoolRISC clock for test and MTP programing |
| 64 | RCRes | | Analog | Optional external resistor for RC oscillator |

**Table 16.1: Pin-out of the XE88LC05 in TQFP64**

## 16.4 Electrical specifications

### 16.4.1 Absolute maximum ratings

| name | value |
|---|---|
| Maximal voltage applied between any pin (but VPP/TEST) and VSS | 5.5 V |
| Voltage applied to any pin (but VPP/TEST) | VSS - 0.3 V to VDD + 0.3 V |
| Storage temperature (no programmed) | 150 °C |
| Storage temperature (programmed) | 85 °C |

**Table 16.2: Absolute maximum ratings**

### 16.4.2 Operating conditions

| name | value |
|---|---|
| Voltage applied between VDD and VSS (ROM version, without ADC or DAC) | 1.2 - 5.5 V |
| Voltage applied between VDD and VSS (MTP version, or ROM version with ADC or DAC) | 2.4 - 5.5 V |
| Operating temperature (ROM) | -40 - 125°C |
| Operating temperature (MTP) | -40 - 85 °C |

**Table 16.3: Operating conditions**

### 16.4.3 IO pins operation

| sym | description | condition | min | typ | max | unit | Comments |
|---|---|---|---|---|---|---|---|
| | Port A: low threshold limit | Vbat = 1.2 V | | | | V | |
| | Port A: high threshold limit | | | | | V | |
| | output drop when sinking 1 mA | | | | 0.4 | V | |
| | output drop when sourcing 1 mA | | | | 0.4 | V | |
| | Port A: low threshold limit | Vbat = 2.4 V | | 1 | | V | |
| | Port A: high threshold limit | | | 1.5 | | V | |
| | output drop when sinking 1 mA | | | | | V | |
| | output drop when sinking 8 mA | | | | 0.4 | V | |
| | output drop when sourcing 1 mA | | | | | V | |
| | output drop when sourcing 8 mA | | | | 0.4 | V | |
| | Port A: low threshold limit | Vbat = 5.0 V | | 2 | | V | |
| | Port A: high threshold limit | | | 3 | | V | |
| | output drop when sinking 1 mA | | | | | V | |
| | output drop when sinking 8 mA | | | | 0.4 | V | |
| | output drop when sourcing 1 mA | | | | | V | |
| | output drop when sourcing 8 mA | | | | 0.4 | V | |
| | pull-up, pull-down resistor | | 50 | | 150 | kohm | |

**Table 16.4: IO pins performances**

# 17 Index

# 18 Contact

One can contact XEMICS at info@xemics.com or on our web site at http://www.xemics.com.

List of contacts, representatives and distributors can be found on the web at http://www.xemics.com/contact.html.

Application notes can be found on the web at http://www.xemics.com/downldata.html.

XEMICS Headquarters is

XEMICS SA
Maladière 71
2007 Neuchâtel
Switzerland