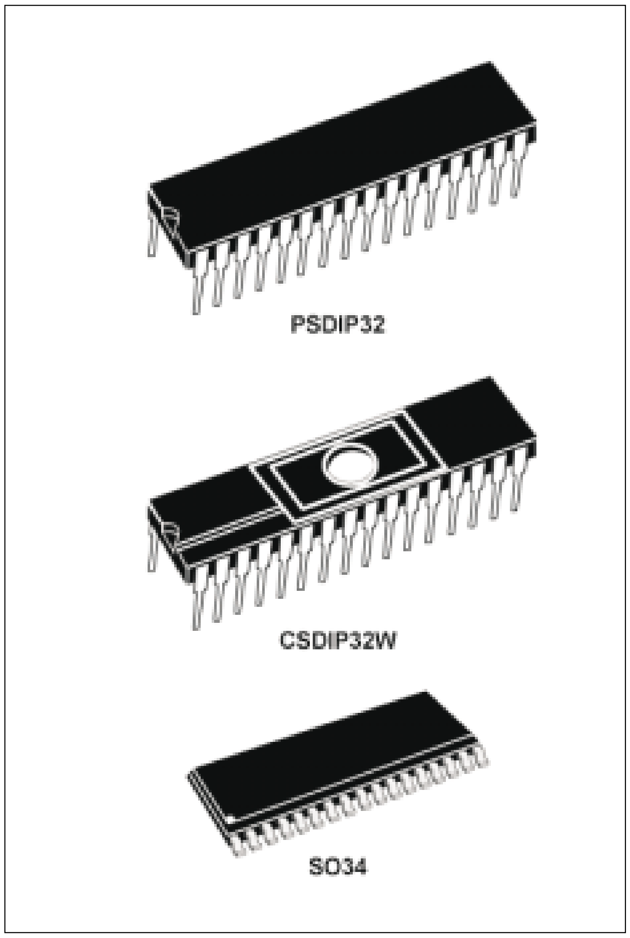
ST52T430/E430

8-BIT *DuaLogic*™ MICROCONTROLLER WITH THREE TIMER/PWM DRIVERS and SCI

PRELIMINARY DATA

- Digital Microcontroller with embedded Fuzzy capabilities, up to 8 Kbytes internal EPROM, 256 bytes of Data RAM
- On-chip 8 bit A/D Converter with an 8-channel multiplexer
- 23 configurable I/O PINs
- Hardware multiplication and division
- Two Programmable Timer/PWMs with internal 16-bit Prescaler and 8-bit counter, featuring:
  - PWM output and Pulse generator mode
  - Negated Outputs
- One Programmable Timer/PWM with internal 16-bit Prescaler and 8-bit counter, featuring:
  - 1 Input capture
  - 1 Output compare
  - External/Internal Clock
  - PWM output and Pulse generator mode
  - Negated Output
- Watchdog timer
- Serial Communication Interface with asynchronous protocol (UART)
- Capability to perform boolean, arithmetic operations and fuzzy algorithms
- 46 basic instructions
- Power Saving Features
- Software tools and Emulators availability
- EPROM readout protection

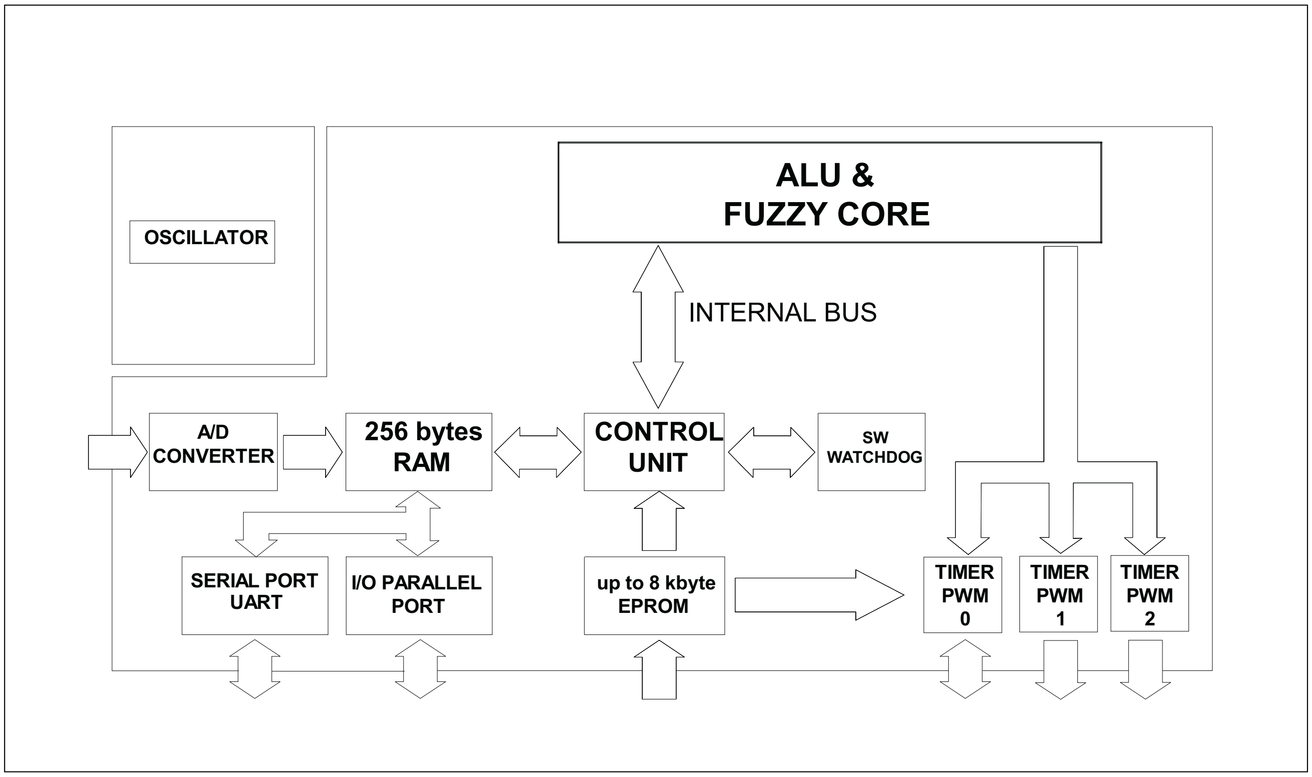


Features	**ST52430K1**	**ST52430K2**	**ST52430K3**
Program Memory - bytes	2K	4K	8K
RAM - bytes	256		
PWM/Timers	Three		
ADC	8 Channels / 8 bit		
Other Peripherals	Watchdog, SCI		
Operating Supply	3 to 5.5 V		
CPU Frequency	Up to 20 MHz		
Temperature Range	-40 to + 85 °C		
Package	SSO34 - PSDIP32		

January 2001

Table of Contents

Figure 1.1 ST52x430 Architectural Block Diagram



1 GENERAL DESCRIPTION

ST52x430 is a member of the ST52 family of 8-bit *DuaLogic™* microcontrollers. It is able to perform, in an efficient way, both boolean and fuzzy algorithms, in order to reach the best performances that the two methodologies allow.

Hardware multiplier and divider are available to implement complex functions by using a single instruction thus optimizing the program memory utilization and the computational speed.

It is produced by STMicroelectronics using the reliable high performance CMOSM6XE (O.5µm) process.

Thanks to Fuzzy Logic, ST52x430 allows to describe a problem using a linguistic model instead of a mathematical model. In this way it is very useful and easy to modelize and control complex systems with very high accuracy. It is possible to implement fuzzy expert systems for the overall system management and real time controller at a very competitive cost.

The linguistic approach is based on a set of IF-THEN rules, describing the control behavior, and on Membership Functions associated to input and output variables.

Fuzzy Inference is a set of operations which computes the output values according with the truth values of the involved rules.

The flexible I/O configuration of ST52x430 allows to interface with a wide range of external devices, like D/A converters or power control devices.

ST52x430 pins are configurable (Alternate Function), allowing to set the input, or output, signals on each single pin, as shown in figures 1.2 and 1.3.

The OTP (One Time Programmable) device is fully compatible with the EPROM windowed version, which may be used for the prototyping and pre-production phases of development.

The EPROM can be locked by user to prevent external undesired operations.

It is possible to store up to 341 Membership Functions, with triangular and trapezoidal shapes, or singleton values if fuzzy algorithms have to be implemented.

Three TIMER/PWM drivers allow to manage power devices and timing signals, implementing different operating modes and high frequency PWM (Pulse With Modulation) controls.

One of these programmable Timers, with Internal Prescaler, can utilize internal or external START/STOP signals and clock. It includes Input Capture (IC) and Output Compare (OC) functions.

An internal programmable watchdog is available to avoid loop errors and reset the microcontroller.

In order to reduce the energy consumption, ST52x430 is able to perform two different power saving features : Wait mode and Halt mode.

The EPROM contains the microcontroller configuration, in terms of I/O number, microcode, Fuzzy Rules and Membership Functions (Mbfs).

ST52x430 processes inputs and produces the related outputs according to the configuration loaded during the programming phase (stored into the EPROM).

ST52x430 includes an 8-bit Analog to Digital Converter with an 8-analog channel Multiplexer.

A Serial Communication peripheral (SCI) using the UART protocol allows to transfer data from the ST52x430 to other external devices.

A powerful development environment consisting of board and software allows an easy configuration and use of ST52x430.

Operations on the data stored in the RAM (256 bytes), allowing to directly combine new inputs and feedback data can be easily performed by using a Registers File approach. All the 256 bytes of RAM are used like Registers File.

It is possible to perform 16 bit over 8 bit arithmetical divisions, with 8 bit result and 8 bit remainder and 8 bit by 8 bit arithmetical multiplications, with 16-bit result.

ST52x430 is fully supported by FUZZYSTUDIO^{TM}4.0 allowing to graphically design a project and obtain an optimized microcode.

ST52x430 exploits a STMicroelectronics patented strategy to store the Mbfs in its internal memory.

1.1 Functional Description

ST52x430 works in two modes according to the control signal level.

ST52x430 is a programmable product and its operation phases are:

- Memory Programming Phase
- Working Phase

These phases are selected by using the following signals (see pins description):

RESET

TEST

V_{PP}

1.1.1 Memory Programming Phase

ST52x430 memory is loaded in Memory Programming Phase. All fuzzy and standard instructions are written inside the memory.

This phase starts with the setting of the control signals as follows:

RESET	TEST	V_{PP}
V_{SS}	V_{SS}	12V/V_{DD}

When this phase starts, ST52x430 core is set to the RESET status. This allows to program and/or to test the internal EPROM (see EPROM programming).

1.1.2 Working mode

In this mode the control signals are the following :

RESET	TEST	V_{PP}
V_{DD}	V_{SS}	V_{SS}

The processor starts the working phase following the instructions which have been previously loaded in the memory.

Figure 2.4 shows the internal structure of ST52x430. It is composed by one computational block: the CONTROL UNIT (CU) / DPU block, which allows the implementation of the fuzzy calculus and the performing of boolean functions.

The CU/DPU is able to manage up to 341 different Membership Functions for the antecedent part of the fuzzy rules. The rules consequents are “crisp” values (real numbers). The number of possible rules is related with the dimensions of the implemented standard algorithm. Smaller standard algorithms allow to define bigger fuzzy algorithms with more rules and viceversa. The program memory is then shared between fuzzy and standard algorithms.

The Control Unit (CU) reads the information and the status incoming from the peripherals.

The arithmetic calculus can be performed on these values by using the internal CU and the 256 bytes RAM, which supports all computations.

The inputs of the peripherals can be the fuzzy and/or arithmetic outputs, or the values contained in Data RAM and EPROM locations.

Figure 1.2 ST52x430 Block Diagram

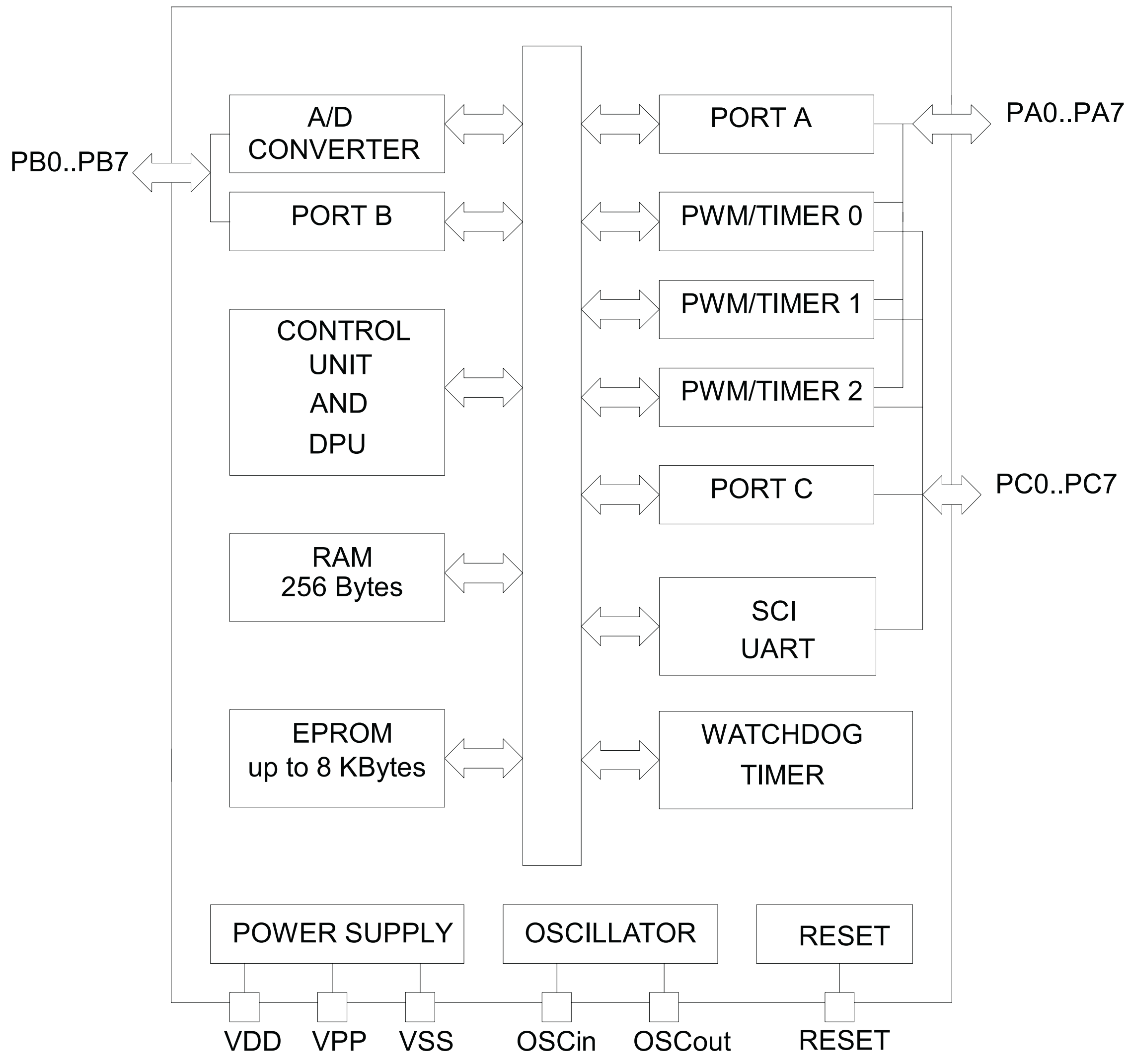


Figure 1.3. SO34 Pin Configuration

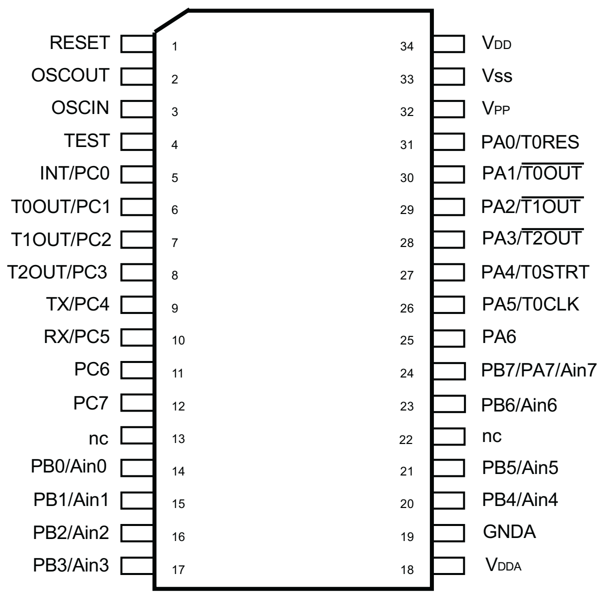


Figure 1.4 PSDIP32 Pin Configuration

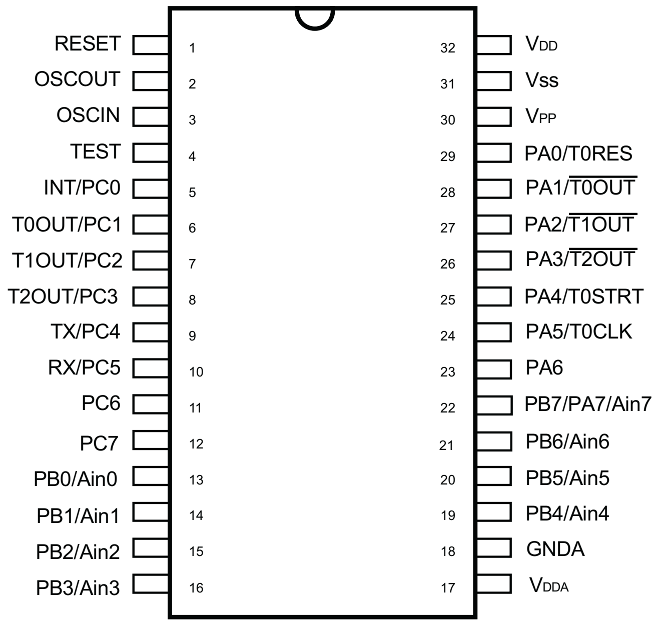


Table 1.1 SO-34 and SDIP-32 Pin Configuration

PIN SO34	PIN SDIP 32	NAME	TYPE	Programming Phase	Working Phase
1	1	RESET		General Reset	General Reset
2	2	OSCOUT		Oscillator Output	Oscillator Output
3	3	OSCIN		Oscillator Input	Oscillator Input
4	4	TEST		Must be tied to V_{ss}	Must be tied to V_{ss}
5	5	INT / PC0	I/O	Phase	External interrupt / Digital I/O
6	6	T0OUT / PC1	I/O		Timer/PWM 0 output / Digital I/O
7	7	T1OUT / PC2	I/O		Timer/PWM 1 output / Digital I/O
8	8	T2OUT / PC3	I/O		Timer/PWM 2 output / Digital I/O
9	9	TX / PC4	I/O		Digital I/O / SCI Output
10	10	RX / PC5	I/O		Digital I/O SCI Input
11	11	PC6	I/O		Digital I/O
12	12	PC7	I/O		Digital I/O
13		nc			
14	13	Ain0 / PB0	I/O	RST_ADD	Analog Input / Digital I/O
15	14	Ain1 / PB1	I/O	INC_ADD	Analog Input / Digital I/O
16	15	Ain2 / PB2	I/O	RST_CONF	Analog Input / Digital I/O
17	16	Ain3 / PB3	I/O	INC_CONF	Analog Input / Digital I/O
18	17	V_{DDA}		Analog Power Supply	Analog Power Supply
19	18	GNDA		Analog Ground	Analog Ground
20	19	Ain4 / PB4	I/O		Analog Input / Digital I/O
21	20	Ain5 / PB5	I/O		Analog Input / Digital I/O
22		nc			
23	21	Ain6 / PB6	I/O		Analog Input / Digital I/O
24	22	Ain7/PB7/PA7	I/O	I/O EPROM Data	Analog Input / Digital I/O
25	23	PA6	I/O	I/O EPROM Data	Digital I/O
26	24	T0CLK / PA5	I/O	I/O EPROM Data	Timer/PWM 0 clock / Digital I/O
27	25	T0STRT / PA4	I/O	I/O EPROM Data	Timer/PWM 0 start/stop / Digital I/O
28	26	$\overline{\text{T2OUT}}$ / PA3	I/O	I/O EPROM Data	Timer/PWM 2 negated Output /Digital I/O
29	27	$\overline{\text{T1OUT}}$ / PA2	I/O	I/O EPROM Data	Timer/PWM1 negated Output / Digital I/O
30	28	$\overline{\text{T0OUT}}$ / PA1	I/O	I/O EPROM Data	Timer/PWM 0 negated Output /Digital I/O
31	29	T0RES / PA0	I/O	I/O EPROM Data	Timer/PWM 0 Reset / Digital I/O
32	30	V_{PP}		EPROM Programming Power supply (12V $\pm$5%)	EPROM V_{DD} or Vss
33	31	V_{SS}		Digital Ground	Digital Ground
34	32	V_{DD}		Digital Power Supply	Digital Power Supply

1.2 Pin Description

ST52x430 pins are configurable by means of configuration registers:

V_{DD}, V_{SS}, V_{DDA}, GNDA, V_{PP}. In order to avoid noise disturbances, the power supply of the digital part is kept separated from the power supply of the analog part.

V_{DD}. Main Power Supply Voltage (5V± 10%).

V_{SS}. Digital circuit ground.

V_{DDA}. Analog V_{DD} of the Analog to Digital Converter.

GNDA. Analog V_{SS} of the Analog to Digital Converter. *Must be tied to V_{SS}.*

V_{PP}. Main Power Supply for the internal EPROM. (12.5V±5%, in programming phase) and MODE selector. During the Programming phase (programming), V_{PP} must be set at 12V. In the Working phase V_{PP} must be equal to **V_{SS}**.

OSCin and **OSCout.** These pins are internally connected with the on-chip oscillator circuit. A quartz crystal or a ceramic resonator can be connected between these two pins in order to allow the correct operations of ST52x430 with various stability/cost trade-offs. An external clock signal can be applied to OSCin, in this case OSCout must be floating.

RESET. This signal is used to restart ST52x430 at the beginning of its program. It also allows to select the program mode for the EPROM.

Ain0-Ain8. These 8 lines are connected to the inputs of the analog multiplexer. They allow to acquire 8 analog inputs. During the Programming phase, Ain0, Ain1, Ain2 and Ain3 are used to manage EPROM operation.

PA0-PA7, PB0-PB7, PC0-PC7.. These lines are organized as I/O port. Each pin can be configured as input or output. PA7/PB7 are tied to the same output. During the Programming phase PA port is used for the EPROM data read/write.

T0RES, **T0CLK**, **T0STRT**. These pins are related with the internal Programmable Timer/PWM 0. This Timer can be reset externally by using T0RES. In Working Mode, T0RES resets the address counter of the Timer. T0RES is active at low level.

The Timer 0 Clock can be the internal clock or can be supplied externally by using the pin T0CLK.

An external Start/Stop signal can be used to control the Timer through the pin T0STRT.

T0OUT, T1OUT, T2OUT. The TIMER/PWM outputs are available on these pins.

$\overline{\text{T0OUT}}$, **$\overline{\text{T1OUT}}$**, **$\overline{\text{T2OUT}}$**. The TIMER/PWM inverted outputs are available on these pins.

Tx. Serial data output of SCI transmitter block

Rx. Serial data input of the SCI receiver block.

TEST. During the Programming and Working phase it must be set to Vss.

2 INTERNAL ARCHITECTURE

ST52x430 is made up by the following blocks and peripherals:

- Control Unit (CU) and Data Processing Unit (DPU)
- ALU / Fuzzy Core
- EPROM
- 256 Byte RAM
- Clock Oscillator
- Analog Multiplexer and A/D Converter
- 3 PWM / Timers
- SCI
- Digital I/O port

2.1 ST52x430 Operating Modes

ST52x430 works in two modes, Programming and Working Modes, depending on the control signals level RESET, TEST and V_{PP}

The Operating modes are selected by setting the control signal level as specified in the Control Signals Setting table.

Table 2.1. Control Signals setting

Control Signal	Pro-gramming	Reset	Working
RESET	V_{SS}	V_{SS}	V_{DD}
TEST	V_{SS}	V_{SS}	V_{SS}
V_{PP}	12 V	V_{SS}	V_{SS}

2.2 Control Unit and Data Processing Unit

The Control Unit (CU) formally includes five main blocks. Each block decodes a set of instructions then generating the appropriate control signals. The main parts of the CU are shown in the figure 2.1.

The five different parts of the CU manage the Loading, Logic/Arithmetic, Jump, Control and Fuzzy instructions set.

The block called “Collector” manages the signals coming from the different parts of the CU then defines the signals for the Data Processing Unit (DPU) and for the different peripherals of the microcontroller.

The block called “Arbiter” manages the different parts of the CU in order to have only one part of the system activated during the working mode.

The CU structure is very flexible. It was designed with the aim to easily adapt the core of the microcontroller to the market needs. New instructions set or new peripherals can be easily included without changing the structure of the microcontroller then maintaining the code compatibility.

The CU reads the stored instructions on the EPROM (Fetch) and decode them.The Arbiter according to the instructions type, activates one of the main blocks of the CU. Then all the control signals for the DPU are generated.

A set of 46 different arithmetic, fuzzy and logic instructions is available. Each instruction requires from 6 (fuzzy instructions) to 26 (DIVISION) clock pulses to be performed.

The DPU receives, stores and sends the instructions coming from the EPROM, the RAM or from the peripherals in order to execute them.

Figure 2.1 CU Block Diagram

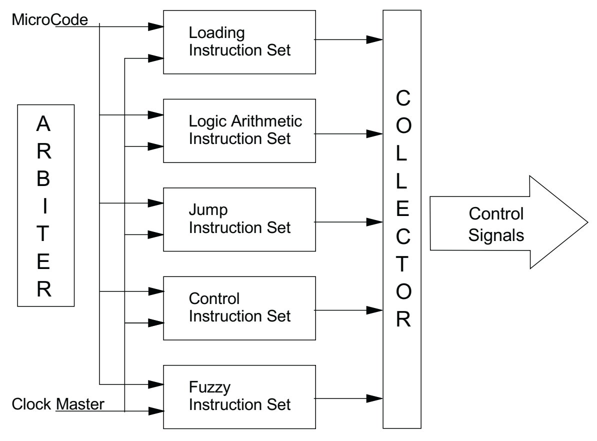


Figure 2.2 Data Processing Unit (DPU)

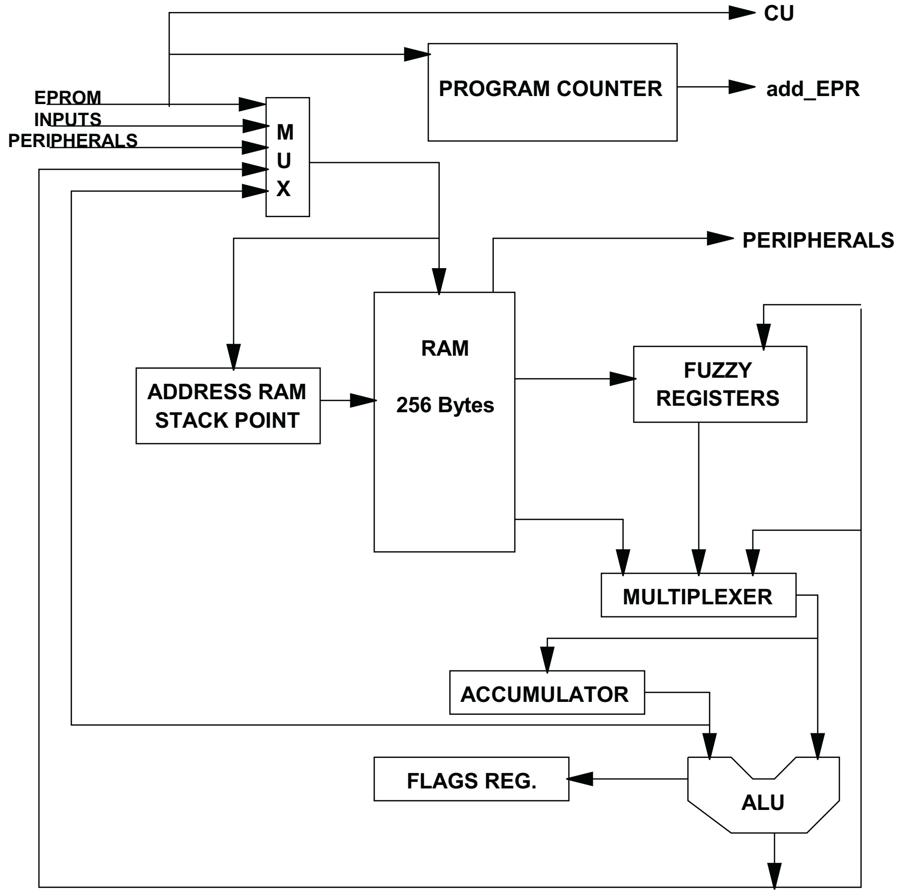


Figure 2.3 CU/DPU Block diagram

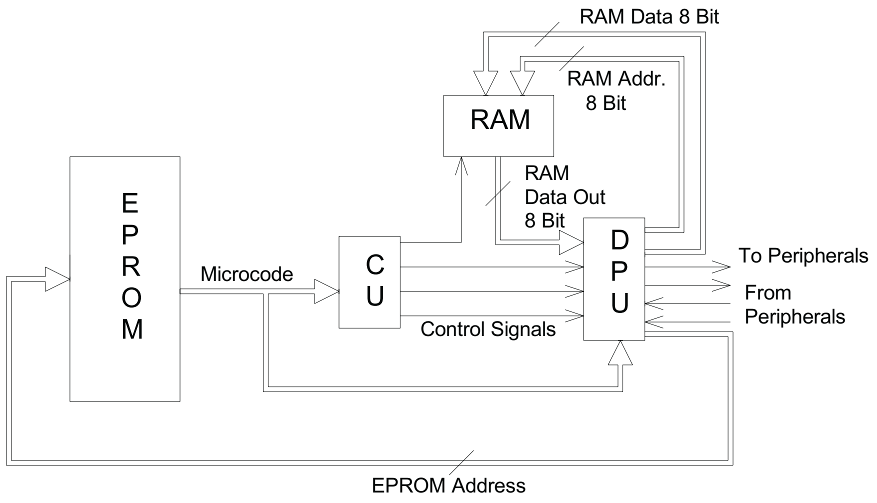
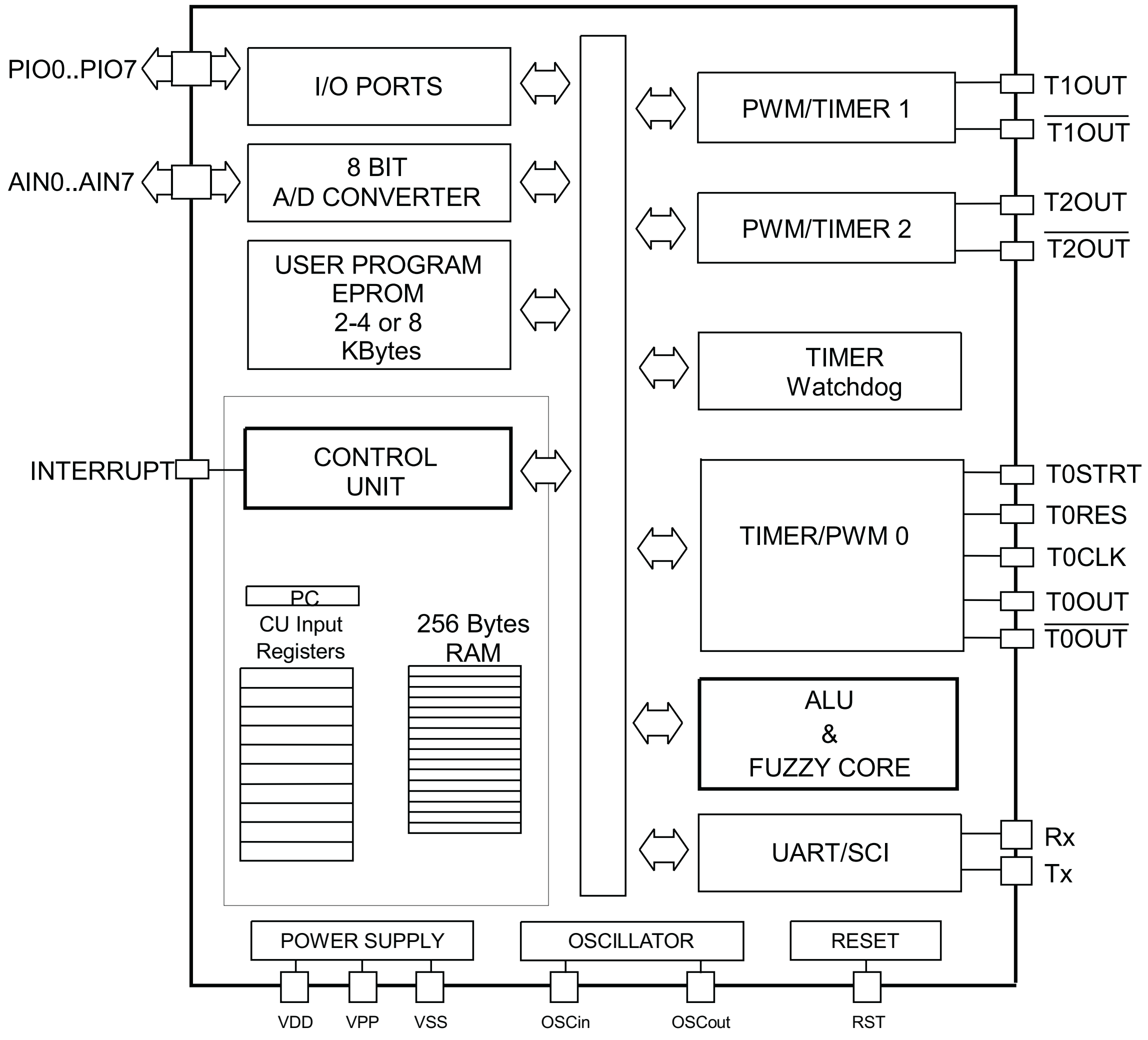


Figure 2.4 ST52x430 Peripherals Block Diagram



2.2.1 Program Counter

The Program Counter (PC) is a 13-bit register that contains the address of the next memory location to be processed by the core. This memory location may be an opcode, an operand or an address of an operand.

The 13-bit length allows the direct addressing of maximum 8192 bytes in the program space.

After having read the current instruction address, the PC value is incremented. The result of this operation is shifted back into the PC.

The PC can be changed in the following ways:

▪ JP (Jump)	PC = Jump Address
▪ Interrupt	PC = Interrupt Vector
▪ RETI	PC = Pop (stack)
▪ RET	PC = Pop (stack)
▪ CALL	PC = Subroutines address
▪ Reset	PC = Reset Vector
▪ Normal Instruction	PC = PC + 1

2.2.2 Flags

The ST52x430 core includes different set of flags that correspond to 2 different modes: normal mode and interrupt mode. Each set of flags consist of a CARRY flag (C), a ZERO flag (Z) and SIGN flag (S). One set (CN, ZN, SN) is used during normal operation and one is used during the interrupt mode (CI, ZI, SI). **Formally the user has to manage only a set of flag: C, Z and S**.

The ST52x430 core uses the flags that correspond to the actual mode: as soon as an interrupt is generated, the ST52x430 core uses the interrupt flags instead of the normal flags.

Each interrupt level has its own set of flags, that it is saved in the STACK together with the Program Counter. These flags are restored from the STACK automatically, when a RETI instruction is executed.

If the MCU was in the normal mode before an interrupt, when the RETI instruction is executed, the normal flags are restored.

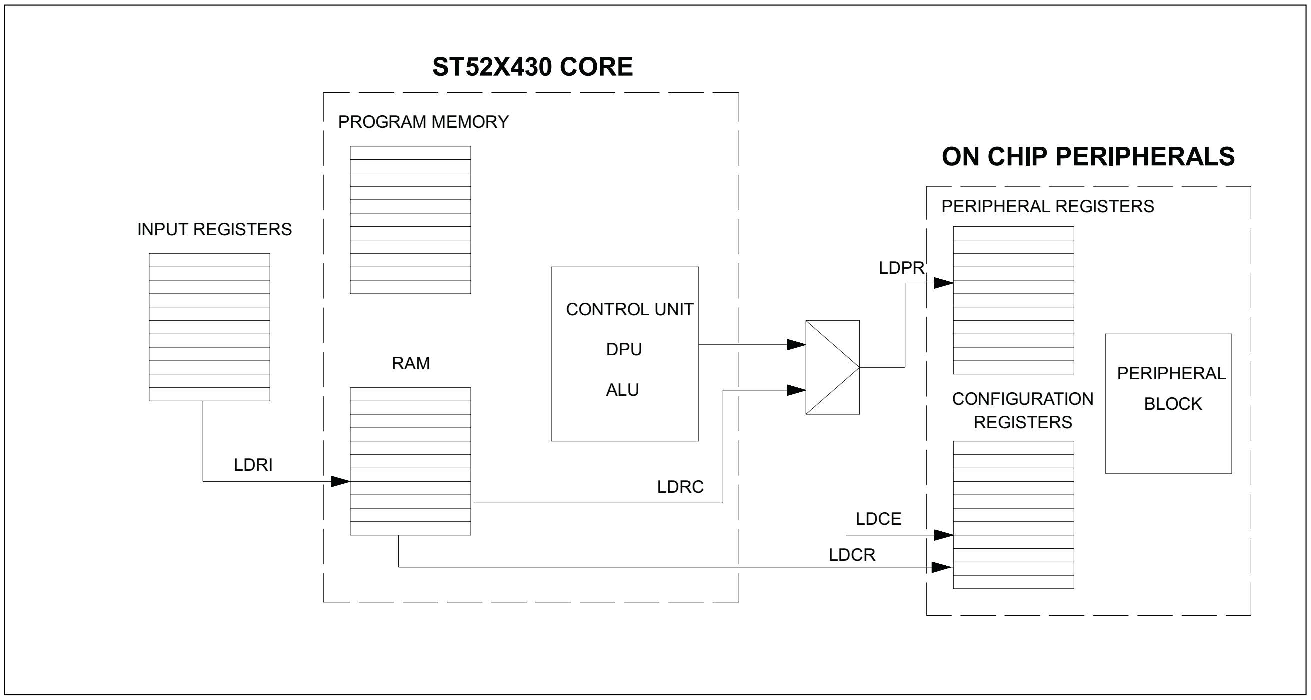
Note: *A CALL subroutine is a normal mode execution. For this reason a RET instruction, consequent to a CALL instruction, does not affect, the normal mode set of flags.*

The flags are not cleared during the context switching and remain in the state they were at the exit of the last interrupt routine switching.

The Carry flag is set when an overflow occurs during arithmetic operations, otherwise it is cleared.

The Sign flag is set when an underflow occurs during arithmetic operations, otherwise it is cleared.

Figure 2.5 Address Spaces Description



2.3 Address Spaces

ST52x430 has four separate address spaces:

- RAM: 256 Bytes
- Input Registers: 20 8-bit registers
- Output Registers 10 8-bit registers
- Configuration Registers: 21 8-bit registers
- Program memory up to 8K Bytes

The Program memory will be described in further details in the MEMORY section

2.3.1 RAM and STACK

The RAM memory consists of 256 general purpose 8-bit RAM registers.

All the registers in the RAM can be specified by using a decimal address, e.g. 0 identifies the first register of the RAM.

To read or write the RAM registers the LOAD instructions must be used. See table 2.6

Each interrupt level has its own set of flags, that is saved in the STACK together with the Program Counter. These flags are restored from the STACK automatically, when a RETI instruction is executed.

When the instructions like Interrupt request or CALL are executed, a STACK level is used to push the PC .

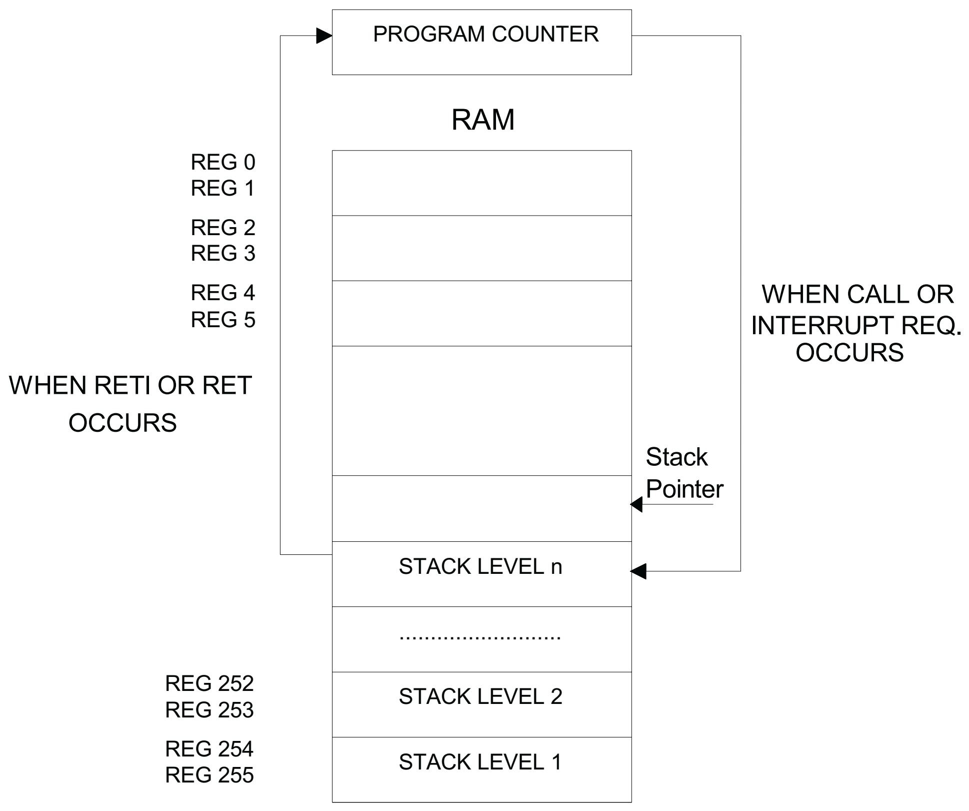
The STACK is located in the RAM. For each level of stack 2 bytes of the RAM are used. The values of this stack are stored from the last RAM register (address 255). **The maximum level of stack must be less than 128**.

The STACK POINTER indicates the first level available to store data. When a subroutine call or interrupt request occurs, the content of the PC and the current set of flag are stored into the level located by the STACK POINTER. When a interrupt return occurs (RETI instruction), the data stored in the highest stack level are restored back into the PC and current flags. Instead when a subroutine return occurs (RET instruction) the data stored in the highest stack level are restored in the PC not affecting the flags.

These operating modes are described in the Figure 2.6.

Note: *User must take care to avoid the overwriting of the RAM locations, where the STACK could be stored* .

Figure 2.6. Stack Operation



2.3.2 Input Registers Bench

The Input Registers (IR) bench consists of 20 8-bit registers containing data or status of the peripherals.

All the registers can be specified by using a decimal address, e.g. 0 identifies the first register of the IR.

The assembler instruction:

```
LDRI RAM_Reg. IR_i
```

loads the value of the i-th IR in the RAM location identified by the address RAM_Reg.

The first input register is dedicated to store the value of the stack pointer. The next 8 registers (ADC_OUT_0:7) of the IR are dedicated to the 8 converted values coming from the ADC. The last 9 registers contain data from the I/O ports and PWM/Timers. The following table summaries the IR address and the relative peripheral. For simplicity reasons a mnemonic name is assigned to the registers. The same name is used in FUZZYSTUDIO^{TM}4.0 development tools.

Table 2.2 Input Registers

IR MNEMONIC NAME	PERIPHERAL REGISTER	ADDRESS
STACK_POINTER	STACK POINTER	0
CHAN 0	A/D CHANNEL 0	1
CHAN 1	A/D CHANNEL 1	2
CHAN 2	A/D CHANNEL 2	3
CHAN 3	A/D CHANNEL 3	4
CHAN 4	A/D CHANNEL 4	5
CHAN 5	A/D CHANNEL 5	6
CHAN 6	A/D CHANNEL 6	7
CHAN 7	A/D CHANNEL 7	8
PORT_A	PORT A INPUT REGISTER	9
PORT_B	PORT B INPUT REGISTER	10
PORT_C	PORT C INPUT REGISTER	11
PWM_0_COUNT	PWM/TIMER 0 COUNTER	12
PWM_0_STATUS	PWM/TIMER 0 STATUS REGISTER	13
PWM_1_COUNT	PWM/TIMER 1 COUNTER	14
PWM_1_STATUS	PWM/TIMER 1 STATUS REGISTER	15
PWM_2_COUNT	PWM/TIMER 2 COUNTER	16
PWM_2_STATUS	PWM/TIMER 2 STATUS REGISTER	17
SCI_RX	SCI DATA REGISTER	18
SCI_STATUS	SCI STATUS REGISTER	19

2.3.3 Configuration Registers

The ST52x430 Configuration Registers allow to configure all the blocks of the fuzzy microcontroller. Table 2.3 describes the functions and the related peripherals of each Configuration Registers. By using the load instructions it is possible to set the Configuration Registers by using values stored in the Program Memory (EPROM) or in the RAM.

Use and meaning of each register will be described in further details in the corresponding section.

Table 2.3. Configuration Registers description

CONFIGURATION REGISTER	PERIPHERAL	DESCRIPTION
REG_CONF 0	INTERRUPT MASK	Interrupts mask setting
REG_CONF 1	N.U.	N.U.
REG_CONF 2	WATCHDOG TIMER	Watchdog Timer configuration
REG_CONF 3	A/D CONVERTER	A/D configuration
REG_CONF 4	PORT A	Set the relative bit like digital input or digital output
REG_CONF 5	PWM/TIMER 0	PWM/TIMER 0 Working mode Configuration
REG_CONF 6	PWM/TIMER 0	PWM/TIMER 0 Prescaler configuration and output waveform selection.
REG_CONF 7	PWM/TIMER 0	PWM/TIMER 0 Working mode Configuration
REG_CONF 8	PWM/TIMER 1	PWM/TIMER 1 Working mode Configuration
REG_CONF 9	PWM/TIMER 1	PWM/TIMER 1 Prescaler configuration and output waveform selection.
REG_CONF 10	PWM/TIMER 2	PWM/TIMER 2 Working mode Configuration
REG_CONF 11	PWM/TIMER 2	PWM/TIMER 2 Prescaler configuration and output waveform selection.
REG_CONF 12	PORT A	Set the bit 0,1 and 2 like Digital I/O or negated Timers Output.
REG_CONF 13	PORT B	Set the relative bit like digital input or digital output.
REG_CONF 14	PORT B	Set the relative I/O like Digital or Analog
REG_CONF 15	PORT C	Set the relative I/O like digital input or digital output
REG_CONF 16	PORT C	Set the relative I/O like Digital I/O or Timers Output
REG_CONF 17	Interrupt Priority	Set the Interrupts priority
REG_CONF 18	Interrupt Priority	Set the Interrupts priority
REG_CONF 19	SCI	Set the SCI working mode
REG_CONF 20	SCI	Set the SCI working mode

2.3.4 Output Registers

The Output Registers (OR) consist of 10 registers containing data for the microcontroller peripherals including the I/O Ports.

All the registers can be specified by using a decimal address, e.g. 1 identifies the second OR.

By using the LOAD instructions it is possible to set the Output Registers (OR) by using values stored in the Program Memory (LDPE) or in the RAM (LDPR)

The assembler instruction:

```
LDPR OR_i RAM_Reg.
```

loads the value of the RAM location identified by the address RAM_Reg in the OR i-th

Table 2.4 describes the OR.

For simplicity reasons a mnemonic name is assigned to the OR. The same names are used in FUZZYSTUDIO^{TM}4.0 development tools.

Use and meaning of each register will be described in further details in the corresponding section.

Table 2.4 Output Registers

OR MNEMONIC NAME	PERIPHERAL REGISTER	ADDRESS
PORT_A	PORT A OR	0
PORT_B	PORT B OR	1
PORT_C	PORT C OR	2
PWM_0_COUNT	TIMER/PWM 0 COUNTER	3
PWM_0_RELOAD	TIMER/PWM 0 RELOAD REGISTER	4
PWM_1_COUNT	TIMER/PWM 1 COUNTER	5
PWM_1_RELOAD	TIMER/PWM 1 RELOAD REGISTER	6
PWM_2_COUNT	TIMER/PWM 2 COUNTER	7
PWM_2_RELOAD	TIMER/PWM 2 RELOAD REGISTER	8
SCI_TX_DATA	SCI DATA REGISTER	9

2.4 Fuzzy Capabilities

ST52x430 Fuzzy main features are:

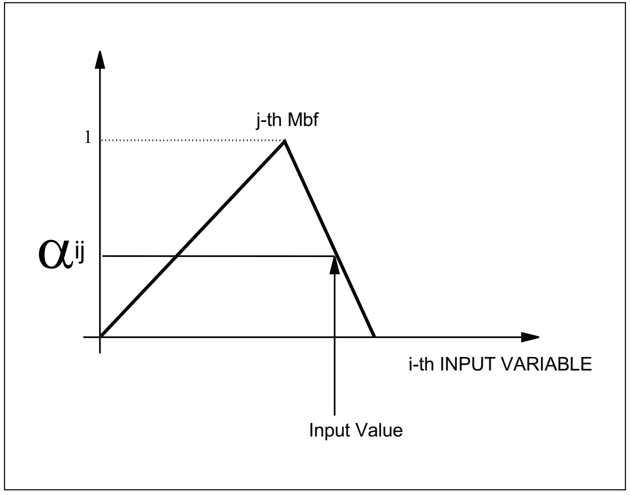
- Up to 8 Inputs with 8-bit resolution;
- 1 Kbyte of Program Memory (EPROM) available to store more than 300 to Membership Functions (Mbfs) for each Input;
- Up to 128 Outputs with 8-bit resolution;
- Possibility to process fuzzy rules with an UNLIMITED number of antecedents
- UNLIMITED number of Rules and Fuzzy Blocks.

The limits on the number of Fuzzy Rules and fuzzy Blocks are only related with the program memory size.

2.4.1 Fuzzy Inference

The block diagram shown in figure 2.8 describes the different steps performed during a fuzzy algorithm. ST52x430 Core allows to implement a MAMDANI type fuzzy inference with crisp consequents. The input for the fuzzy inference are stored in 8 dedicated Fuzzy input registers. The instruction LDFR is used to set the input fuzzy registers with the values stored in the RAM. The result of a fuzzy inference is directly stored in a location of the RAM

Figure 2.7. Alpha Weight calculation

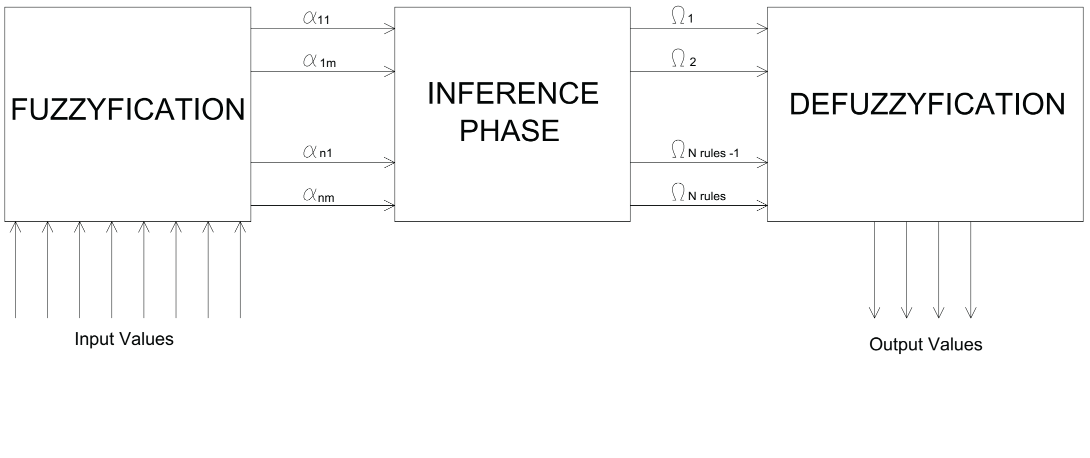


2.4.2 Fuzzyfication Phase

In this phase is performed the intersection (alpha weight) between the input values and the related Mbfs (fig. 2.7).

8 Fuzzy input registers are available for the fuzzy inferences.

Figure 2.8. Fuzzy Inference



After loading the input values by using the LDFR assembler instruction, the user can start the fuzzy inference by using the assembler instruction FUZZY.

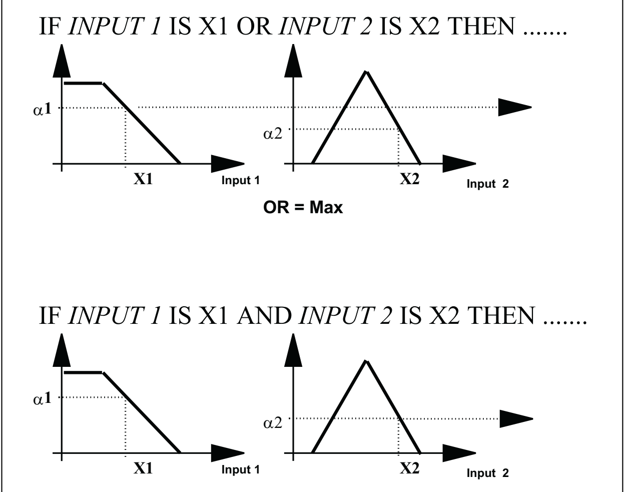
During the **fuzzyfication**: the input data are transformed in activation level (alpha weight) of the Mbfs.

2.4.3 Inference Phase

It manages the alpha weights obtained during the fuzzyfication phase to compute the truth value (ω) for each rule.

This is a calculation of the maximum (for the OR operator) and/or minimum (for the AND operator) performed on alpha values according to the logical connectives of fuzzy rules.

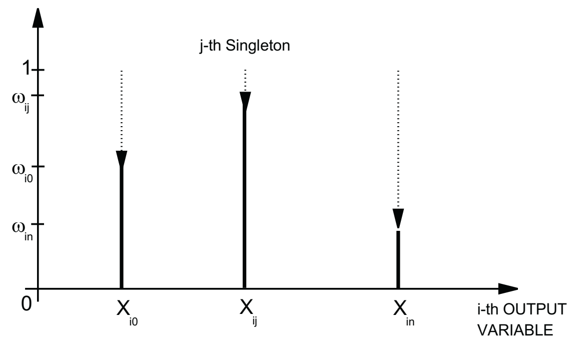
Figure 2.9. Fuzzyfication


IF INPUT 1 IS X1 OR INPUT 2 IS X2 THEN ....
α1
α2
X1
Input 1
X2
Input 2
OR = Max
IF INPUT 1 IS X1 AND INPUT 2 IS X2 THEN ....
α1
α2
X1
Input 1
X2
Input 2

It is possible to link together several conditions by linguistic connectives AND/OR, NOT operator and brackets.

The truth value ω and the related output singleton are passed to the Defuzzyfication phase to complete the inference calculation.

Figure 2.10 Output Membership Functions.



2.4.4 Defuzzyfication

In this phase the output crisp values are determined implementing the consequent part of the rules.

Each consequent Singleton X_i is multiplied by its weight values ω_i, calculated by the Fuzzy Inference Unit in order to compute the upper part of the defuzzification.

Each output value is deduced from the consequent crisp values (X_i) by using the defuzzification formula:

$$Y_i = \frac{\sum_j^N X_{ij}\,\omega_{ij}}{\sum_j^N \omega_{ij}}$$

where:

i = 0,1 identifies the current output variable

N = number of the active rules on the current output

ω_{ij} =weight of the j-th singleton

Xij = abscissa of the j-th singleton

The fuzzy outputs are stored in the RAM location i-th specified in the assembler instruction OUT i.

2.4.5 Input Membership Function

ST52x430 allows to manage triangular Mbfs. In order to define a Mbf it is necessary to store three different data on the program memory:

the vertex of the Mbf: **V**;

the length of the left semi-base: **LVD**;

the length of the right semi-base: **RVD**;

In order to reduce the size of the memory area and the computational effort the vertical dimension of the vertex is fixed to 15 (4 bits)

By using the previous memorization method it is possible to store different kinds of triangular Membership Functions. The figure 2.12 shows a typical example of Mbfs that can be defined in ST52x430.

Each Mbf is then defined storing 3 bytes in the first 1 Kbyte of the memory program.

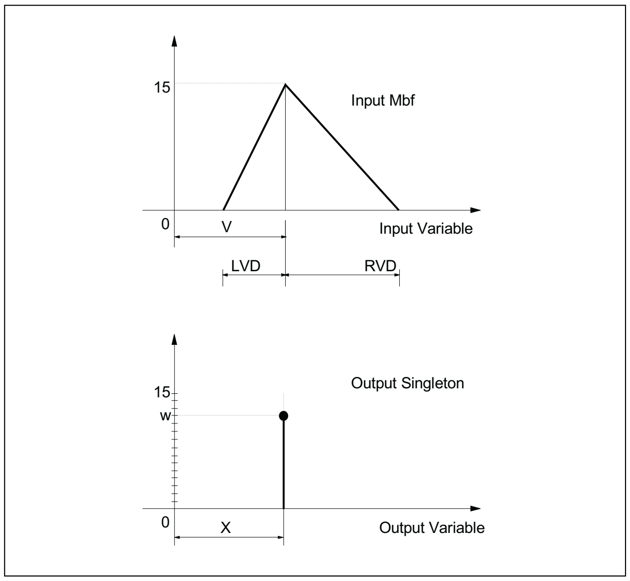
The Mbf is memorized by using the following instruction:

MBF n_mbf lvd v rvd

where

n_mbf identifies the Mbf, lvd, v, rvd are the parameters describing the Mbf's shape.

Figure 2.11. Mfs Parameters



2.4.6 Output Singleton

ST52x430 uses for the output variables a particular kind of membership function called Singleton. A Singleton has not a shape, like a traditional Mbf, and it is characterized by a single point identified by the couple (X, w), where the w is calculated by the Inference Unit as described before.

Often, a Singleton is simply identified with its Crisp Value X.

2.4.7 Fuzzy Rules

The rules can have the following structures:

if A op B op C...........then Z

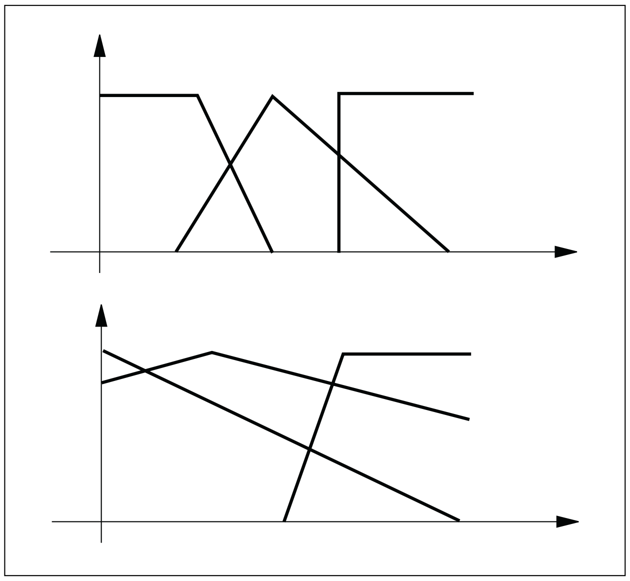
if (A op B) op (C op D op E...) ....then Z

where op is one of the possible linguistic operators (AND/OR)

In the first case the rule operators are managed sequentially; in the second one, the priority of the operator is fixed by the brackets.

Each rule is codified by using an instruction set, the inference time for a rule with 4 antecedents and 1 consequent is about 3 microseconds.

Figure 2.12. Example of valid Mbfs



The assembler Instruction Set allowing to manage the fuzzy instructions is reported in the following table:

Table 2.5. Fuzzy Instructions Set

Instruction	Description
MBFn_mbf lvd v rvd	Stores the Mbf n_mbf with the shape identified by the parameters lvd, v and rvd.
LDP n m	Fixes the alpha value of the input n with the Mbf m and stores it in internal registers.
LDN n m	Calculates the negated alpha value of the input n with the Mbf m and store the result in internal registers.
FZAND	Implements the fuzzy operation AND between the last two values stored in internal registers.
FZOR	Implements the fuzzy operation OR between the last two values stored in internal registers.
LDK	Stores the result of the last fuzzy operation executed in internal registers.
SKM	Loads the result of the last performed Fuzzy operation (stored in the temporary register K) in the temporary buffer M
LDM	Copies the value of the register M in the data stack.
CON crisp	Multiplies the crisp value with the last ω weight.
OUT n_out	Performs the defuzzification and store the fuzzy output in the RAM n_out location.
FUZZY	Starts the fuzzy algorithm.

Example 1:

IF $Input_1$ IS NOT Mbf_1 AND $Input_4$ is Mbf_{12} OR $Input_3$ IS Mbf_8 THEN $Crisp_1$

is codified by the following instructions:

LDN 1 1	calculates the NOT α value of $Input_1$ with Mbf_1 and stores the result in internal registers
LDP 4 12	fixes the α value of $Input_4$ with M_{12} and stores the result in internal registers
FZAND	adds the NOT α and α values obtained with the operations LDN1 1 and LDP 4 12
LDK	stores the result of the operation FZAND in internal registers
LDP 3 8	fixes the α value of $Input_3$ with Mbf_8 and stores the result in internal registers
FZOR	implements the operation OR between the results obtained with the operations LDK and LDP
CON $crisp_1$	multiplies the result of the last Ω operation with the crisp value $Crisp_1$

Example 2, the priority of the operator is fixed by the brackets:

IF ($Input_3$ IS Mbf_1 AND $Input_4$ IS NOT Mbf_{15}) OR ($Input_1$ IS Mbf_6 OR $Input_6$IS NOT Mbf_{14}) THEN $Crisp_2$

LDP 3 1	fixes the α value of $Input_3$ with Mbf_1 and stores the result in internal registers
LDN 4 15	calculates the NOT α value of $Input_4$ with Mbf_{15} and stores the result in internal registers
FZAND	adds NOT α and α values obtained with the operations LDP 3 1 and LDN 4 15
SKM	stores the result of the operation FZAND in internal registers
LDP 1 6	fixes the α value of $Input_1$ with Mbf_6 and stores the result in internal registers
LDN 2 14	calculates the NOT α value of $Input_6$ with Mbf_{14} and stores the result in internal registers
FZOR	implements the operation OR between the α and NOT α values obtained with the two previous operations (LDP 1 6 and LDN 2 14)
LDK	stores the result of the operation OR in internal registers
LDM	copies the value of the memory register M in internal registers
FZOR	implements the operation OR between the last two values stored in internal registers (LDK and LDM)
CON $crisp_2$	multiplies the result of the last Ω operation with the crisp value $Crip_2$

At the end of the fuzzy rules by using the instruction OUT RAM_reg a byte is set then the control of the algorithm goes back to the CU.

2.5 Arithmetic Logic Unit

The 8-bit Arithmetic Logic Unit (ALU) allows to perform arithmetic calculations and logic instructions which can be divided into 5 groups: Load, Arithmetic, Jump, Interrupts and Program Control instructions (refer to the ST52x430 Assembler Set for further details).

The computational time required for each instruction consists of one clock pulse for each Cycle plus 3 clock pulses for the decoding phase.

The ALU of the ST52x430 is able to perform multiplication (MULT) and division (DIV). The multiplication is performed by using 8 bit operands storing the result in 2 registers (16 bit values), see Figure 2.13.

Table 2.6. Arithmetic & Logic Instructions Set

Load Instructions						
Mnemonic	**Instruction**	**Bytes**	**Cycles**	**Z**	**S**	**C**
LDCE	LDCE conf, EPROM	3	17	-	-	-
LDCR	LDCR conf, RAM	3	14	-	-	-
LDFR	LDFR FUZZY_i, RAM	3	14	-	-	-
LDPE	LDPE per, EPROM	3	17	-	-	-
LDPR	LDPR reg, RAM	3	14	-	-	-
LDRC	LDRC RAM, const	3	14	-	-	-
LDRE	LDRE RAMi, EPROMi	3	16	-	-	-
LDRE	LDRE (RAMi), (EPROMj)	3	18	-	-	-
LDRI	LDRI RAM, inp_reg	3	15	-	-	-
LDRR	LDRR RAMi, RAMj	3	16	-	-	-
PGSET	PGSET const	2	9	-	-	-

Arithmetic Instructions						
Mnemonic	**Instruction**	**Bytes**	**Cycles**	**Z**	**S**	**C**
ADD	ADD regi, regj	3	17	I	-	I
ADDO	ADDO regi, regj	3	20	I	I	I
AND	AND regi, regj	3	17	I	-	-
ASL	ASL regi	2	15	I	-	I
ASR	ASR regi	2	15	I	I	-
DEC	DEC regi	2	15	I	I	-
DIV	DIV regi, regj	3	26	I	I	I
INC	INC regi	2	15	I	-	I
MULT	MULT regi, regj	3	19	I	-	-
NOT	NOT regi	2	15	I	-	-
OR	OR regi, regj	3	17	I	-	-
SUB	SUB regi, regj	3	17	I	I	-
SUBO	SUBO regi, regj	3	20	I	I	I
MIRROR	MIRROR regi	2	15	I	-	-

Table 2.7. Arithmetic & Logic Instructions Set (Continue)

Jump Instructions						
Mnemonic	**Instruction**	**Bytes**	**Cycles**	**Z**	**S**	**C**
CALL	CALL addr	3	18	-	-	-
JP	JP addr	3	12	-	-	-
JPC	JPC addr	3	10/12	-	-	-
JPNC	JPNC addr	3	10/12	-	-	-
JPNS	JPNS addr	3	10/12	-	-	-
JPNZ	JPNZ addr	3	10/12	-	-	-
JPS	JPS addr	3	10/12	-	-	-
JPZ	JPZ addr	3	10/12	-	-	-
RET	RET	1	13	-	-	-

Interrupt Instructions Set						
Mnemonic	**Instruction**	**Bytes**	**Cycles**	**Z**	**S**	**C**
HALT	HALT	1	7/15	-	-	-
MEGI	MEGI	1	7/15	-	-	-
MDGI	MDGI	1	6	-	-	-
RETI	RETI	1	12	-	-	-
RINT	RINT INT	2	8	-	-	-
UDGI	UDGI	1	6	-	-	-
UEGI	UEGI	1	7/15	-	-	-
WAITI	WAITI	1	7/14	-	-	-

Control Istructions set						
Mnemonic	**Instruction**	**Bytes**	**Cycles**	**Z**	**S**	**C**
FUZZY	FUZZY	1	5	-	-	-
NOP	NOP	1	6	-	-	-
WDTRFR	WDTRFR	1	7	-	-	-
WDTSLP	WDTSLP	1	6	-	-	-

Notes:
I affected
- not affected

The division is performed between a 16 bit dividend and an 8 bit divider, the result is stored in an 8 bit register (See Fig. 2.14)

Figure 2.13 Multiplication

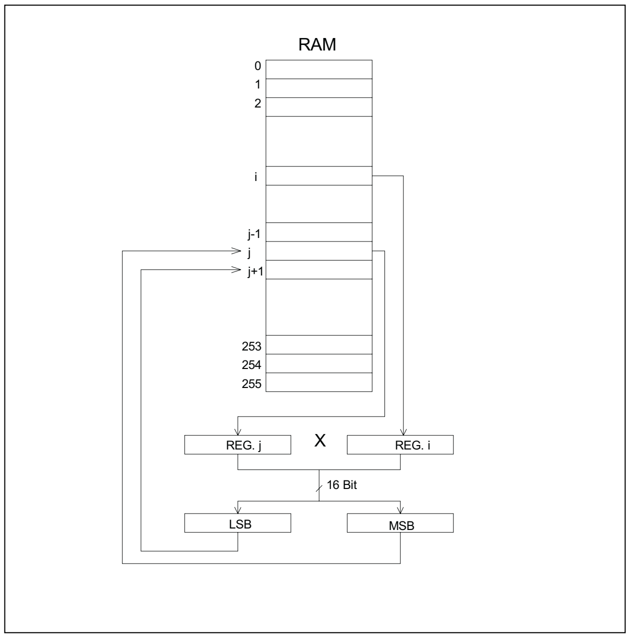
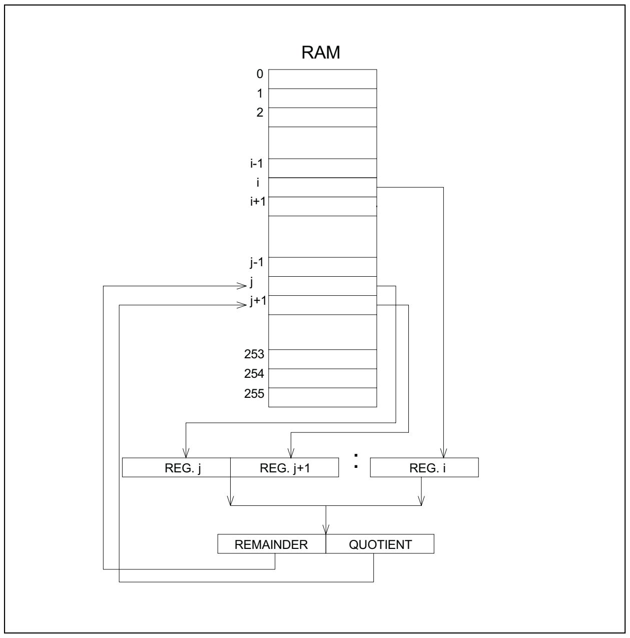


Figure 2.14 Division



3 EPROM

The EPROM memory provides an on-chip user-programmable non-volatile memory, that allows fast and reliable storage of user data.

The EPROM memory can be locked by user. In fact a memory location, called Lock Cell, is devoted to lock the EPROM and to avoid external operations. It is possible to write a software identification code, called ID CODE, to distinguish which software version is stored in the memory.

There are 64 kbits of memory space with an 8-bit internal parallelism (up to 8 kbytes) addressed by an 13-bit bus. The data bus is of 8 bits.

The memory has a double supply: V_{PP} is equal to 12V±5% in Programming Phase or to V_{SS} during Working Phase. V_{DD} is equal to 5V±10%.

The ST52x430 EPROM memory is divided into three main blocks (see Figure 3.1):

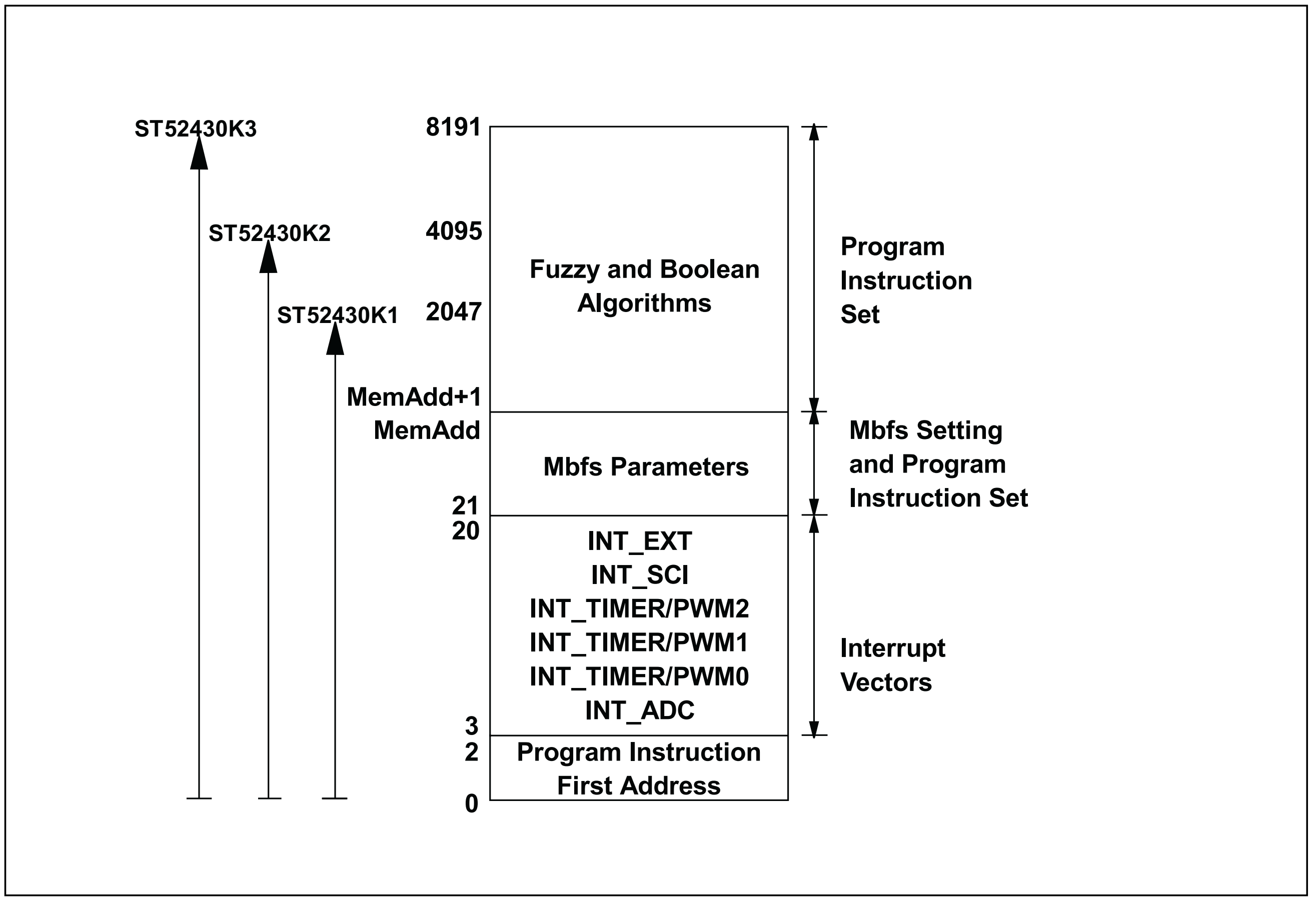
- *Interrupt Vectors memory block* (3 through 20) contains the addresses for the interrupt routines. Each address is composed of three bytes.
- *Mbfs Setting memory block* (21 through *MemAdd*) contains the coordinates of the vertexes of every Mbf defined in the program. The maximum value of *MemAdd* is 1023. This area is dynamically assigned according to the size of the fuzzy routines. The unused memory area, if any, is assigned to the Program Instruction Set memory block.
- *The Program Instruction Set memory block (MemAdd* through *MemAddx)* contains the instruction set of the user program. The following table summarize the values of *MemAddx* for the different devices

	ST52T430K1	ST52T430K2	ST52T430K3
Mem Addx	2047	4095	8191

The locations 0, 1 and 2 contain the address of the first microcode instruction.

The operations that can be performed on the EPROM during the Programming Phase are: Stand By, Memory Writing, Reading and

Figure 3.1 Global Interrupt Request generation



Verify/Margin Mode, Memory Lock, IDCode Writing and Verify.

Table 3.1. EPROM Configuration Register

OPERATION	REGISTER VALUE
Stand By	0
Memory Reading / Verify	1
Memory Unlock and Lock Status Reading	2
Memory Writing	3
Memory Lock	4
ID CODE Writing	5
Memory Lock Status Reading / Verify	9
ID CODE Reading / Verify	10

Above operations are managed by using an internal 4-bit configuration register and an EPROM Configuration Register. The reading phase is executed with V_{PP}= 5V±5%, while the verify/Margin Mode phase needs V_{PP}= 12V±5%. The Blank Check must be a reading operation with V_{PP}= 5V±5%.

Table 3.1 shows the EPROM Configuration Register codes used to identify the running operation.

3.1 EPROM Programming Phase Procedure

The Programming mode is selected by applying 12V±5% voltage or 5V±5% voltage to the V_{PP} pin and setting the control signal as following:

RESET =Vss

TEST =Vss

If the V_{PP} voltage is 5V±5% it is possible only to read.

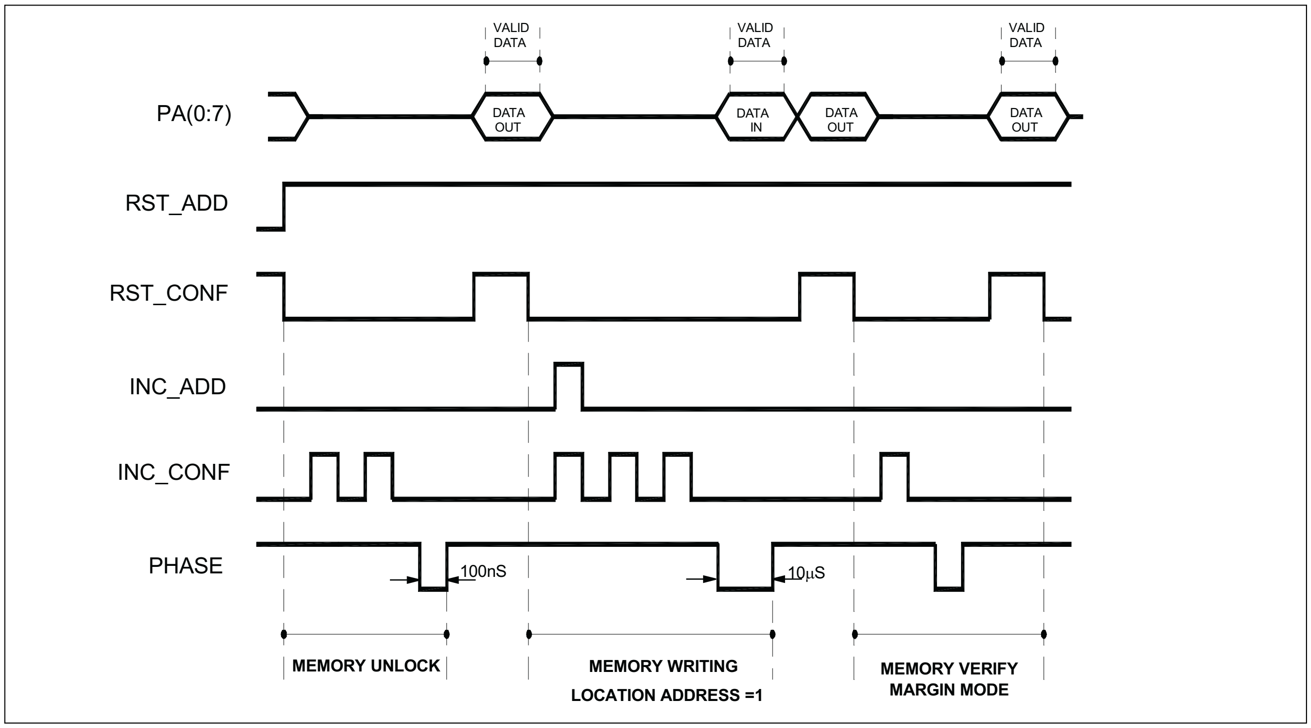
RST_ADD, INC_ADD, RST_CONF, INC_CONF and PHASE are the control signals used during the Programming Mode.

PHASE, RST_CONF and RST_ADD signals are active on level, the others are active on rising edge.

PHASE and RST_ADD signals are active low, RST_CONF signal is active high.

Port A is used for the memory data I/O. ***(See table 1.1 for pin reference on the different packages)***.

Figure 3.2. EPROM Programming Timing



It is possible to lock the memory by means of the Memory Lock Status, that is a flag used to enable the EPROM operations.

If Memory Lock Status is 1 all EPROM operations are enabled, otherwise, it is only possible to read (and verify) the OTP code and the Memory Lock Status.

Only If the EPROM is not locked by mean of Lock Cell (see paragraph 3.1.2), it is possible to enable the EPROM operations, changing the Memory Lock Status from 0 to 1 .

RST_ADD signal resets the memory address register and the Memory Lock Status. For this reason, when the RST_ADD becomes high, it is necessary to unlock the memory to read or write.

INC_ADD signal increments the memory address.

RST_CONF signal resets the EPROM Configuration Register. **When RST_CONF is high, the DATA I/O Port A is in output, otherwise it is always in input.**

INC_CONF signal increments the EPROM Configuration Register value.

PHASE signal validates the operation selected by means of EPROM Configuration Register value.

3.1.1 EPROM Operation

To execute one EPROM operation (See Table 3.1), the corresponding identification value must be loaded in the EPROM Configuration Register. The signal timing is the following: RST_ADD= high and PHASE= high, RST_CONF changes from low to high level, to reset the EPROM Configuration Register, and INC_CONF signal generates a number of positive pulses equal to the value to be loaded. After this sequence, a negative pulse of the PHASE signal will validate the selected operation. The minimum PHASE signal pulse width must be 10 µs for the EPROM Writing Operation and 100 ns for the others.

When RST_CONF is high, the DATA I/O Port A is enabled in output and the reading / verify operation results are available.

After a writing operation, when RST_CONF is high, the Port A is in output with no valid data.

3.1.2 EPROM Locking

The Memory Lock operation, that is identified with the number 4 in the EPROM Configuration Register, writes “0" in the Memory Lock Cell.

At the beginning of an External Operation, when RST_ADD signal changes from low level to high level, the Memory Lock Status is “0", therefore it is necessary to unlock it before to proceed.

To unlock the Memory Lock Status the operation, that is identified with the number 2 in the EPROM Configuration Register must be executed (see Figure 3.2).

The Memory Lock Status can be changed only if Memory Lock Cell is “1", therefore, for this reason, after a Memory Lock operation it is not possible to execute external operations except to read (or verify) the OTP Code and the Memory Lock Status.

3.1.3 EPROM Writing

When the memory is blank, all the bits are at logic level “1". The data are introduced by programming only the zeros in the desired memory location; however all input data must contain both ”1" and “0".

The only way to change “0" into ”1" is to erase the whole memory (by exposure to Ultra Violet light) and reprogram it.

The memory is in Writing mode when the EPROM Configuration Register value is 3.

The V_{PP} voltage must be 12V±5%, with stable data on the data bus PA(0:7).

The signals timing is the following (see Figure 3.2):

1) RST_ADD and RST_CONF change from low to high level,

2) two pulses on INC_CONF signal load the Memory Unlock operation code,

3) a negative pulse (100 ns) on the PHASE signal validates the Memory Unlock operation,

4) a negative pulse on RST_CONF signal resets the EPROM Configuration Register,

5) three positive pulses on INC_CONF load the Memory Writing operation code,

6) a train of positive pulses on INC_ADD signal increments the memory location address up to the requested value (generally this is a sequencial operation and only one pulse is used),

7) a negative pulse (10 µs) on the PHASE signal validates the Memory Writing operation,

3.1.4 EPROM Reading / Verify Margin Mode

The reading phase is executed with V_{PP}= 5V±5%, instead of verify phase that needs V_{PP}= 12V±5%.

The Memory Verify operation is available in order to verify the correctness of the data written. It is possible to execute a Memory Verify Margin Mode operation immediately after the writing of each byte and in this case (see Figure 3.2):

1) a positive pulse on RST_CONF signal resets the EPROM Configuration Register, if it was not already reseted

2) one positive pulse on INC_CONF load the Memory Reading/Verify operation code,

3) a negative pulse (100 ns) on the PHASE signal validates the Memory Reading / Verify operation,

4) a negative pulse on RST_CONF signal puts in the PA(0:7) port the value stored in the actual memory address and resets the EPROM Configuration Register.

Then, if any error in writing occurred, the user has to repeat the EPROM writing.

3.1.5 Stand by Mode

The EPROM has a standby mode which reduces the active current from 10mA (Programming mode) to less than 100 μA. The Memory is placed in standby mode by setting PHASE signal at high level or when the EPROM Configuration register value is 0 and PHASE signal is low.

3.1.6 ID code

It is possible to write a software identification code, called ID code, to distinguish which software version is stored in the memory.

64 Bytes are dedicated to store this code by using the address values from 0 to 63.

It is possible to read or verify the ID Code also if the Memory Lock Status is “0".

The signals timing is the same of a normal operation.

3.2 Eprom Erasure

Thanks to the transparent window available in the CSDIP32W package, its memory contents may be erased by exposure to UV light.

Erasure begins when the device is exposed to light with a wavelength shorter than 4000Å. It should be noted that sunlight, as well as some types of artificial light, includes wavelengths in the 3000-4000Å range which, on prolonged exposure, can cause erasure of memory contents. It is thus recommended that EPROM devices be fitted with an opaque label over the window area in order to prevent unintentional erasure.

The recommended erasure procedure for EPROM devices consists of exposure to short wave UV light having a wavelength of 2537Å. The minimum recommended integrated dose (intensity x expo-sure time) for complete erasure is 15Wsec/cm 2.

This is equivalent to an erasure time of 15-20 minutes using a UV source having an intensity of 12mW/cm 2 at a distance of 25mm (1 inch) from the device window.

4 INTERRUPTS

The Control Unit (CU) responds to peripheral events and external events through its interrupt channels.

When such an event occurs, if the related interrupt is not masked and according to a priority order, the current program execution can be suspended to allow the CU to execute a specific response routine.

Each interrupt is associated with an interrupt vector that contains the memory address of the related interrupt service routine. Each vector is located in the Program Space (EPROM Memory) at a fixed address (see Interrupt Vectors table fig. 4.2).

4.1 Interrupt Functionment

If, at the end of an arithmetic or logic instruction, there are pending interrupts, the one with the highest priority is passed. To pass an interrupt means to store the arithmetic flags and the current PC in the stack and execute the associated Interrupt routine, whose address is located in two bytes of the EPROM memory location between address 3 and 20.

The Interrupt routine is performed as a normal code checking, at the end of each instruction, if a higher priority interrupt has to be passed. An Interrupt request with the higher priority stops the lower priority Interrupt. The Program Counter and the arithmetic flags are stored in the stack.

With the instruction RETI (Return from Interrupt) the arithmetic flags and Program Counter (PC) are restored from the top of the stack. This stack was already described in the section 2.2.1.

An Interrupt request cannot stop the processing of the fuzzy rule, but this is passed only after the end of a fuzzy rule or at the end of a logic, or arithmetic, instruction.

REMARK: A fuzzy routine can be interrupted only in the Main program. An interrupt request cannot stop a Fuzzy function that is running inside another interrupt routine. For this reason, to use a Fuzzy function inside an interrupt routines, the user MUST include the Fuzzy function between an UDGI (MDGI) instruction and and UEGI (MEGI) instruction (see the following paragraphs), in order to disable the interrupt request during the execution of the fuzzy function.

4.2 Global Interrupt Request Enabling

When an Interrupt occurs, it generates a Global Interrupt Pending (GIP), that can be hanged up by software. After a GIP a Global Interrupt Request

Figure 4.1. Interrupt Flow

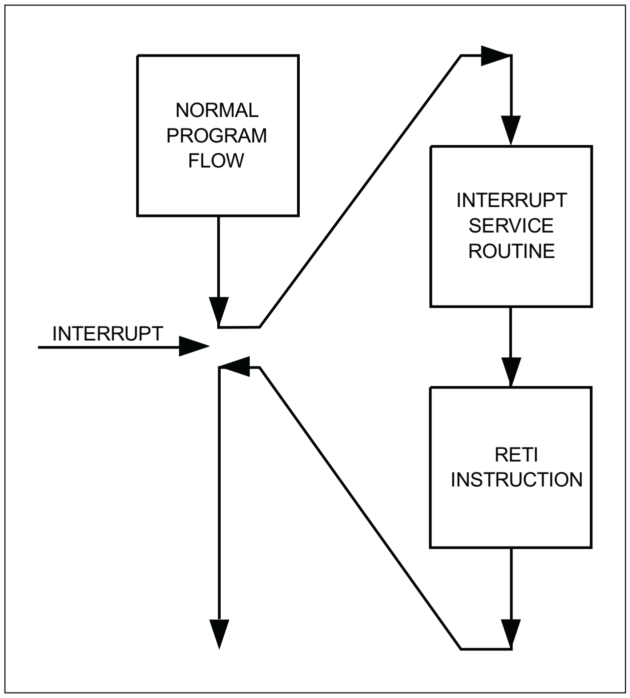


Figure 4.2. Interrupt Vectors Mapping

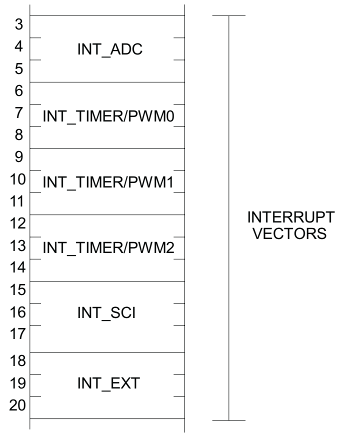
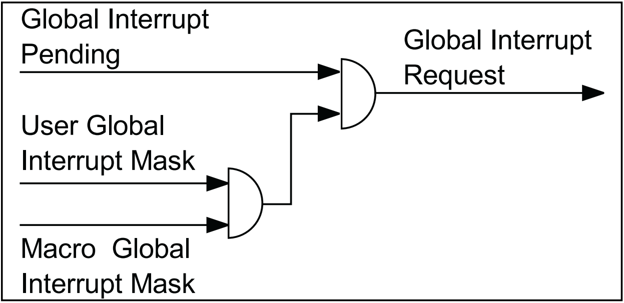


Figure 4.3 Global Interrupt Request generation



(GIR) will be generated and Interrupt Service Routine associated to the interrupt with higher priority will start.

In order to avoid possible conflicts between interrupt masking set in the main program, or inside macros, the GIP is hanged up through the User Global Interrupt Mask or the Macro Global Interrupt Mask (see fig.4.3).

UEGI/UDGI instruction switches on/off the User Global Interrupt Mask enabling/disabling the GIR for the main program.

MEGI/MDGI instructions switches on/off the Macro Global Interrupt Mask in order to ensure that the macro will not be broken.

4.3 Interrupt Sources

ST52x430 manages interrupt signals generated by the internal peripherals (PWM/Timers, UART and Analog to Digital Converter) or coming from the INT/PC0 pin. The External Interrupt can be programmed to be active on the rising or falling edge of INT/PC0 signal by setting the PEXTINT bit of the Configuration Register 0.

Each peripheral can be programmed in order to generate the associated interrupt; further details are described in the related chapter.

4.4 Interrupt Maskability

The interrupts can be masked by configuring the REG_CONF 0 by means of LDCR, or LDCE, instruction. The interrupt is enabled when the bit associated to the mask interrupt is “1". Viceversa, when the bit is ”0", the interrupt is masked and it is kept pendent.

For example:

```
LDRC 10,6 //load the constant 6 in the
RAM Register 10
```

```
LDCR 0, 10 // set the CONF_REG 0 with
the value stored in the RAM Register 10
```

the result is CONF_REG0 =00000110 thus enabling the interrupts coming from the ADC (INT_ADC) and from the PWM/TIMER 0 (INT_PWM/TIMER0).

Table 4.1. Configuration Register 0 Description

Bit	Name	Value	Description
0	MSKE	0	External Interrupt Masked
		1	External Interrupt Not Masked
1	MSKAD	0	A/D Converter Interrupt Masked
		1	A/D Converter Interrupt Not Masked
2	MSKTM0	0	PWM/TIMER 0Interrupt Masked
		1	PWM/TIMER 0 Interrupt Not Masked
3	MSKTM1	0	PWM/TIMER 1 Interrupt Masked
		1	PWM/TIMER 1 Interrupt Not Masked
4	MSKTM2	0	PWM/TIMER 2 Interrupt Masked
		1	PWM/TIMER 2 Interrupt Not Masked
5	MSCI	0	SCI Interrupt Not Masked
		1	SCI Interrupt Masked
6	PEXTINT	0	External Interrupt Polarity Active on Rising Edge
		1	External Interrupt Polarity Active on Falling Edge
7	Not used		

Reset Configuration ‘000000’

Table 4.2. Interrupts Description

Name	Description		Priority	Peripheral Code	Maskable	EPROM Locations
INT_ADC	ADC	Int	Programmable	000	yes	3-5
INT_PWM/TIMER0	PWM/TIMER 0	Int	Programmable	001	yes	6-8
INT_PWM/TIMER1	PWM/TIMER 1	Int	Programmable	010	yes	9-11
INT_PWM/TIMER2	PWM/TIMER 2	Int	Programmable	011	yes	12-14
INT_SCI	SCI	Int	Programmable	100	yes	15-17
INT_EXT	External Interrupt (INT)	Ext	Highest	-	yes	18-20

Figure 4.4. Interrupt Configuration Register 0

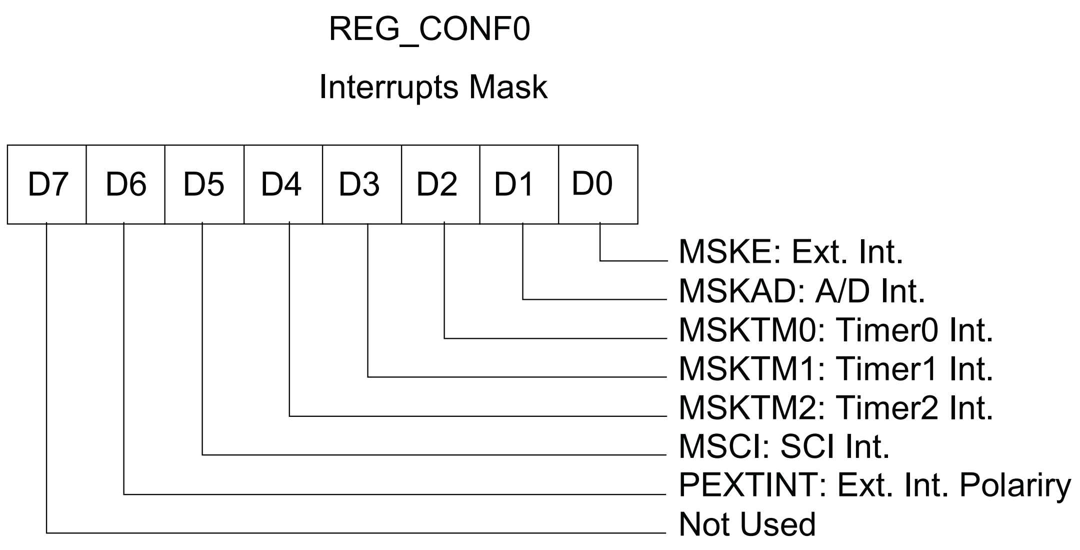
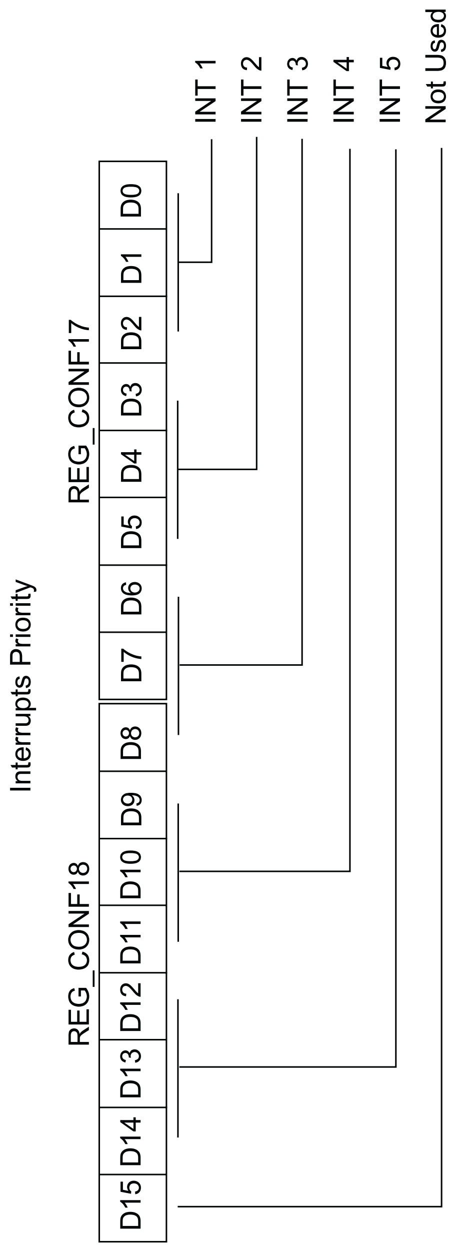


Figure 4.5 Interrupt Configuration Register 17 & 18



4.5 Interrupt Priority

Seven priority levels are available: level 6 has the lowest priority, level 0 has the highest priority.

Level 6 is associated to the Main Program, levels 5 to 1 are programmable by means of the priority registers called REG_CONF17 and REG_CONF18 (see fig.4.5 and table 4.3); whereas the higher level is related to the external interrupt (INT_EXT).

PWM/Timers, UART and ADC are identified by a three-bits Peripheral Code (see Table 4.2); in order to set the *i*-th priority level the user must write the peripheral label *i* in the related INT*i* priority level.

i.e.

```
LDRC  10,  193  //(load  the  value
193='11000001' in the RAM Register 10)
LDRC  11,  168  //(load  the  value
168='10101000' in the RAM Register 11)
LDCR 17, 10  // set  the REG_CONF17=
'11000001'
LDCR 18, 11  // set  the REG_CONF18=
'10101000'
```

thus defining the following priority levels:

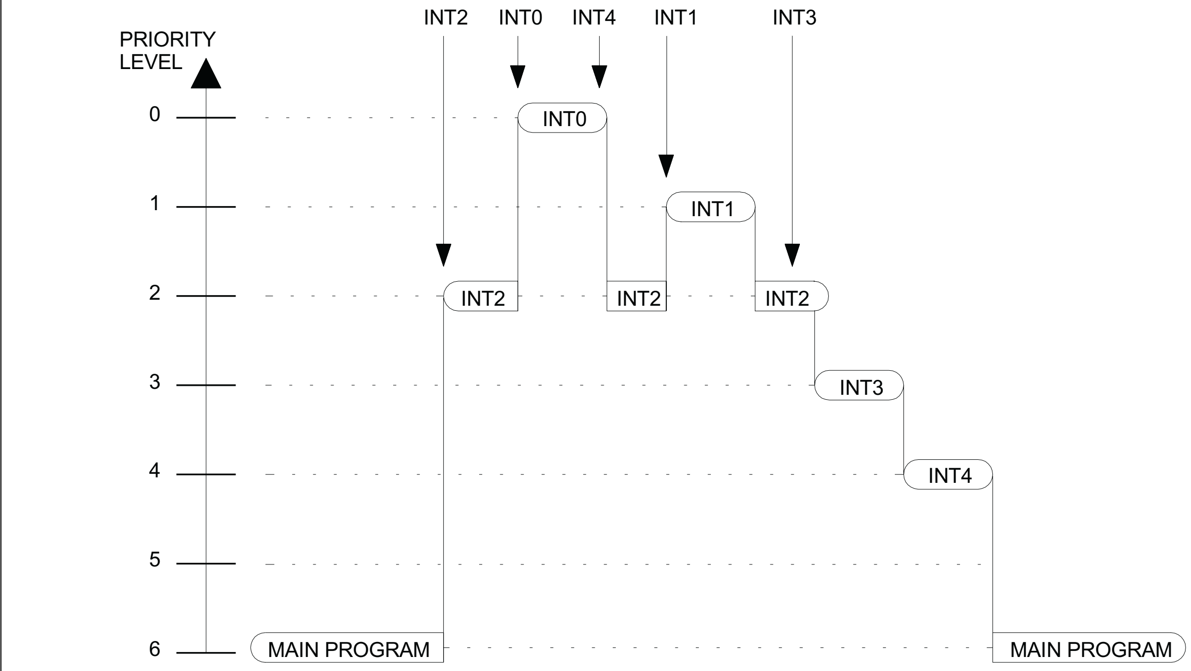
- Level 1: INT_PWM/TIMER0 (PWM/TIMER 0 Code: 001)
- Level 2: INT_ADC (ADC Code: 000)

Table 4.3. Conf. Register 17&18 Description

Bit	Name	Value	Level
0, 1,2	INT1	Peripheral Code	High
3, 4,5	INT2	Peripheral Code	Medium-High
6,7,8	INT3	Peripheral Code	Medium-Low
9,10,11	INT4	Peripheral Code	Low
12,13,14	INT5	Peripheral Code	Very Low

- Level 3:INT_PWM/TIMER2 (PWM/TIMER 2 Code: 011)
- Level 4: INT_UART (UART Code: 100)
- Level 5: INT_PWM/TIMER1 (PWM/TIMER 1 Code: 010)

Figure 4.6. Example of a Sequence of Interrupt Requests



REMARK: The Interrupt priority must be fixed ad the beginning of the main program, because at the RESET REG_CONF1='00000000', and this condition could generate wrong operations. During the program execution it is possible to modify the interrupt priority only with the following procedure:

STEP 1:

Mask the interrupts by means of a UDGI (or MDGI) instruction

STEP 2:

Change the REG_CONF 17&18 values to modify the interrupt priority

STEP 3:

Reset by means of RINT instructions all the pending interrupt routines

STEP 4:

Unmask the interrupts by means of a UEGI (or MEGI) instruction

When a source provides an Interrupt request, and the request processing is also enabled, the CU changes the normal sequential flow of a program by transferring program control to a selected service routine.

When an interrupt occurs the CU executes a JUMP instruction to the address loaded in the related location of the Interrupt Vector.

When the execution returns to the original program, it begins immediately following the interrupted instruction.

Table 4.4. RINT instruction code

Peripheral Name	Value
A/D Converter	0
PWM/TIMER 0	1
PWM/TIMER 1	2
PWM/TIMER 2	3
SCI	4
External Interrupt	5

4.6 Interrupts and Low power mode

All interrupts allow the processor to leave the WAIT low power mode. Only the external Interrupt, allows the processor to leave the HALT low power mode.

4.7 Interrupt RESET

An eventually pending interrupt can be reset with the instruction `RINT j`, which resets the interrupt *j*-th where *j* identifies the peripherals as described in the following table. (see table 4.4).

The assembler instruction:

```
RINT 2
```

Resets the PWM/Timer 1 interrupt.

REMARK: **RINT command must be preceded from a UDGI (or MDGI) command and followed by a UEGI (or MEGI) command**.

5 CLOCK, RESET & POWER SAVING MODE

5.1 System Clock

The ST52x430 Clock Generator module generates the internal clock for the internal Control Unit, ALU and on-chip peripherals and it is designed to require a minimum of external components.

The ST52x430 oscillator circuit generates an internal clock signal with the same period and phase as at the OSCin input pin. The maximum frequency allowed is **20 Mhz**.

REMARK: When the SCI peripheral is used, only 5, 10 or 20 MHz system clock must be used.

The system clock may be generated by using either a quartz crystal, a ceramic resonator or by means of an external clock.

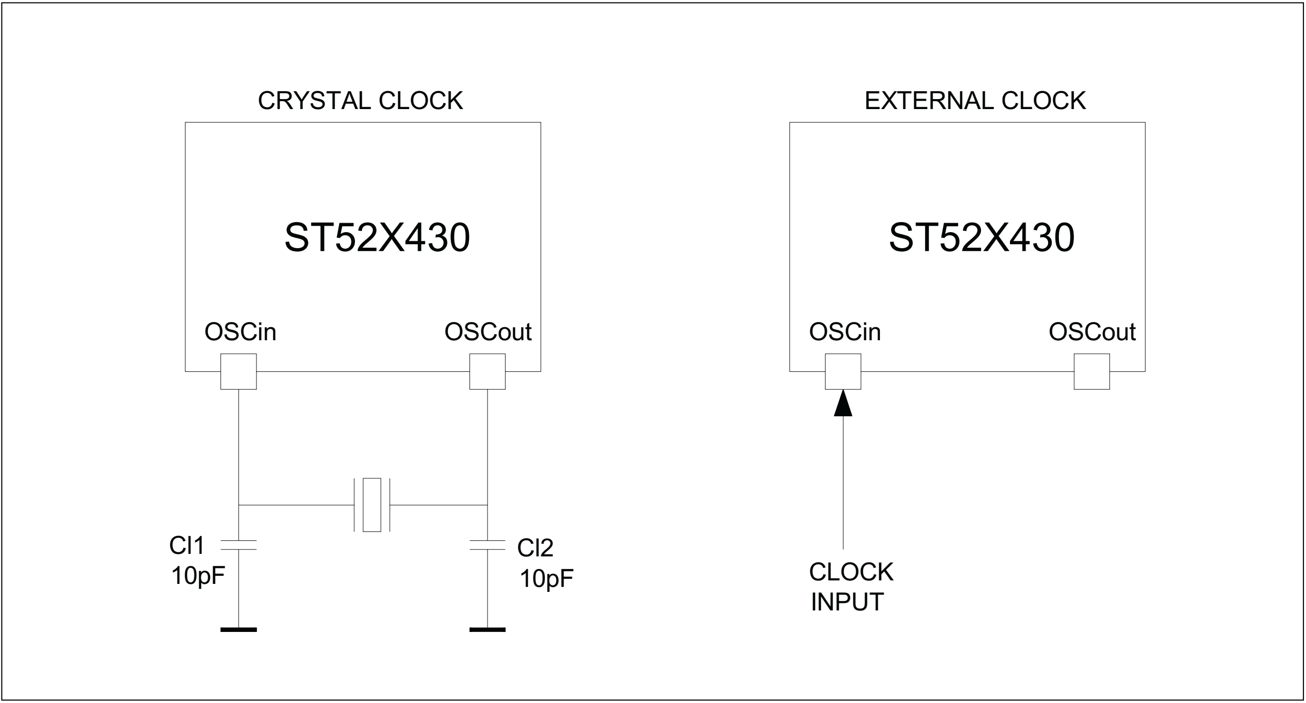
The different clock generator options connection methods are shown in Figure 5.1.

When an external clock is used, it must be connected on the pin OSCin while OSCout can be floating.

The crystal oscillator start-up time is a function of many variables: crystal parameters (especially R_S), oscillator load capacitance (CL), IC parameters, ambient temperature, supply voltage.

It must be observed that the crystal or ceramic leads and circuit connections must be as short as possible. Typical values for CL1, CL2 are 10pF for a 20 MHz crystal.

Figure 5.1 Oscillator Connections



5.2 RESET

There are two sources of Reset:

- RESET pin (external source.)

- WATCHDOG (internal source)

When a Reset event happens, the user program restarts from the beginning.

The Reset pin is an input. An internal reset does not affect this pin.

A Reset signal originated by external sources is instantaneously recognized. The RESET pin may be used to ensure V_{DD} has risen to a point where the MCU can operate correctly before the user program is run. In working mode the Reset must be set to ‘1’ (see Table 2.1)

Figure 5.2 Reset Block Diagram

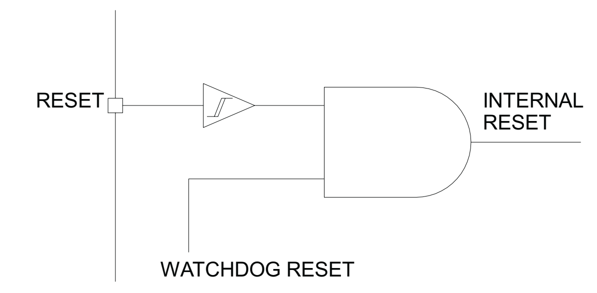
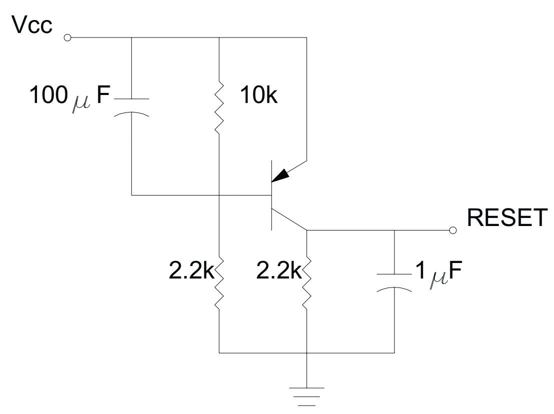


Figure 5.4 Simple Reset Circuit



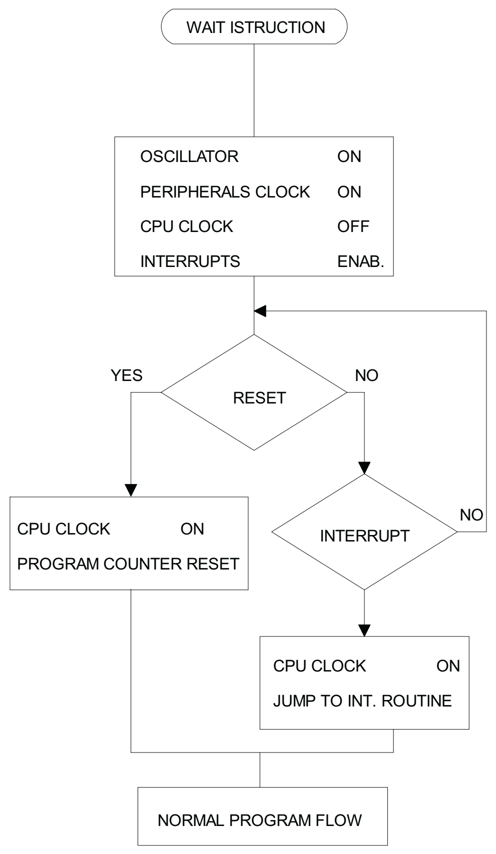
5.3 Power Saving Mode

There are two Power Saving modes: WAIT and HALT mode. These conditions may be entered using the WAIT or HALT instructions.

5.3.1 Wait Mode

Wait mode places the MCU in a low power consumption by stopping the CPU. All peripherals and the watchdog remain active. During the WAIT mode, the Interrupts are enabled. The MCU will remain in Wait mode until an Interrupt or a RESET occurs, whereupon the Program Counter jumps to the interrupt service routine or, if a RESET occurs, at the beginning of the user program.

Figure 5.3 WAIT Flow Chart



5.3.2 Halt Mode

The Halt mode is the MCU lowest power consumption mode. The Halt mode is entered by executing the HALT instruction. The internal oscillator is turned off, causing all internal processing to be stopped, including the operations of the on-chip peripherals. **The Halt mode cannot be used when the watchdog is enabled.** If the HALT instruction is executed while the watchdog system is enabled, it will be skipped without modifying the normal CPU operations.

In Halt mode the external interrupt is enabled. If an interrupt occurs, the CPU becomes active.

The MCU can exit the Halt mode upon reception of an external interrupt or a reset. The oscillator is then turned on and a stabilization time is provided before restarting the CPU operations. The stabilization time is 4096 CPU clock cycles. After the start up delay, the CPU restarts the operations.

Wake-Up from HALT

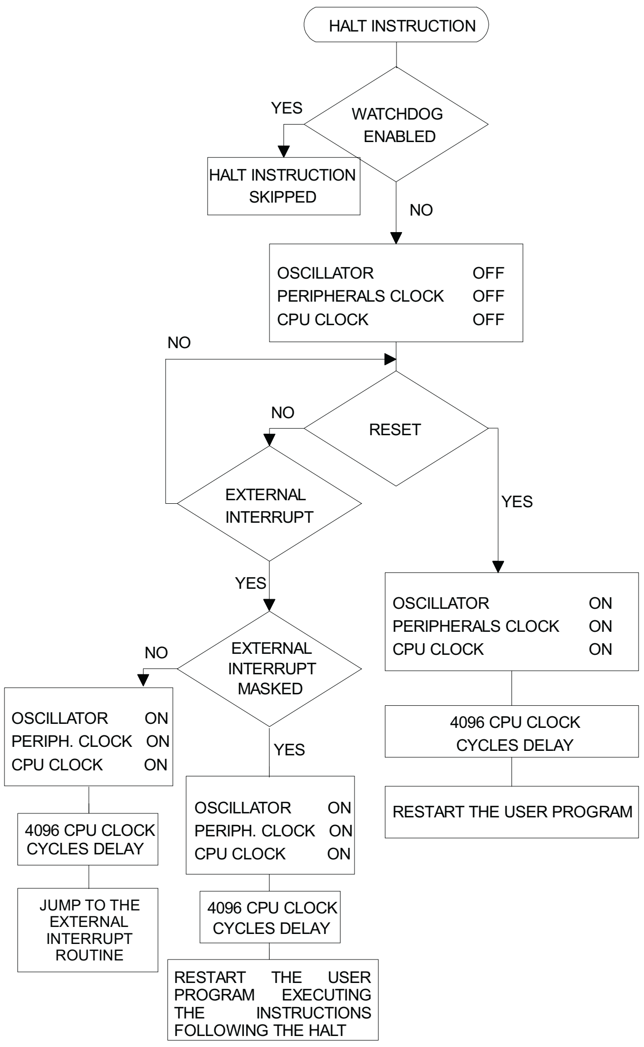
The device can wake up from HALT through one of the following events:

1) External interrupt

2) External reset by fetching the reset vector

Wake-up is regardless of the state of the External Interrupt mask. When the external interrupt is disabled, the device after the start up delay continues execution at the instruction after the HALT instruction. When the External interrupt is enabled, the device after the start up delay continues executing the External Interrupt service routine.

Figure 5.5 HALT Flow Chart



6 I/O PORTS

6.1 Introduction

ST52x430 devices feature flexible individually programmable multi functional input/output lines. Refer to the following figure for specific pin allocations.

23 I/O lines, grouped in 3 different ports, are available on the ST52x430:

PORT A = 7 or 8-bit ports (PA0 - PA7 pins)

PORT B = 7 or 8-bit ports (PB0 - PB7 pins)

PORT C = 8-bit port (PC0 - PC7 pins)

The PIN 24 in the SO34 or PIN 22 in the PDIP32 can be configured to belong to the port A or to the port B.

These I/O lines can be programmed to provide digital input/output and analog input, or to connect input/output signals to the on chip peripherals as alternate pin functions.

The input buffers are TTL compatible with Schmitt trigger in the port A and C while the port B is CMOS compatible without Schmitt trigger.

The output buffer is able to supply up to 8 mA.

The port cannot be configured to be at the same time input and output.

Each port is configured by using two configuration registers. The first is used to define if a pin is an input or output while the second defines the Alternate functions.

6.2 Input Mode

The input configuration is selected setting to “1" the corresponding configuration register bit (REG_CONF 4, 13 and 15) (see paragraph 6.5) . The ports are configured by using the configuration registers shown in the following table.

Table 6.1 I/O Port Configuration Registers

PORT A	PORT B	PORT C
Reg_Conf 4	Reg_Conf 13	Reg_Conf 15

The digital input data are automatically stored in the Input Registers, but it is not possible to read directly the single bit of the IR and it is necessary to copy the value in a RAM location. The digital data are stored in a RAM location by using the assembler instruction:

```
LDRI RAM_Reg Input_i
```

Figure 6.1 Ports A & C Functional Blocks

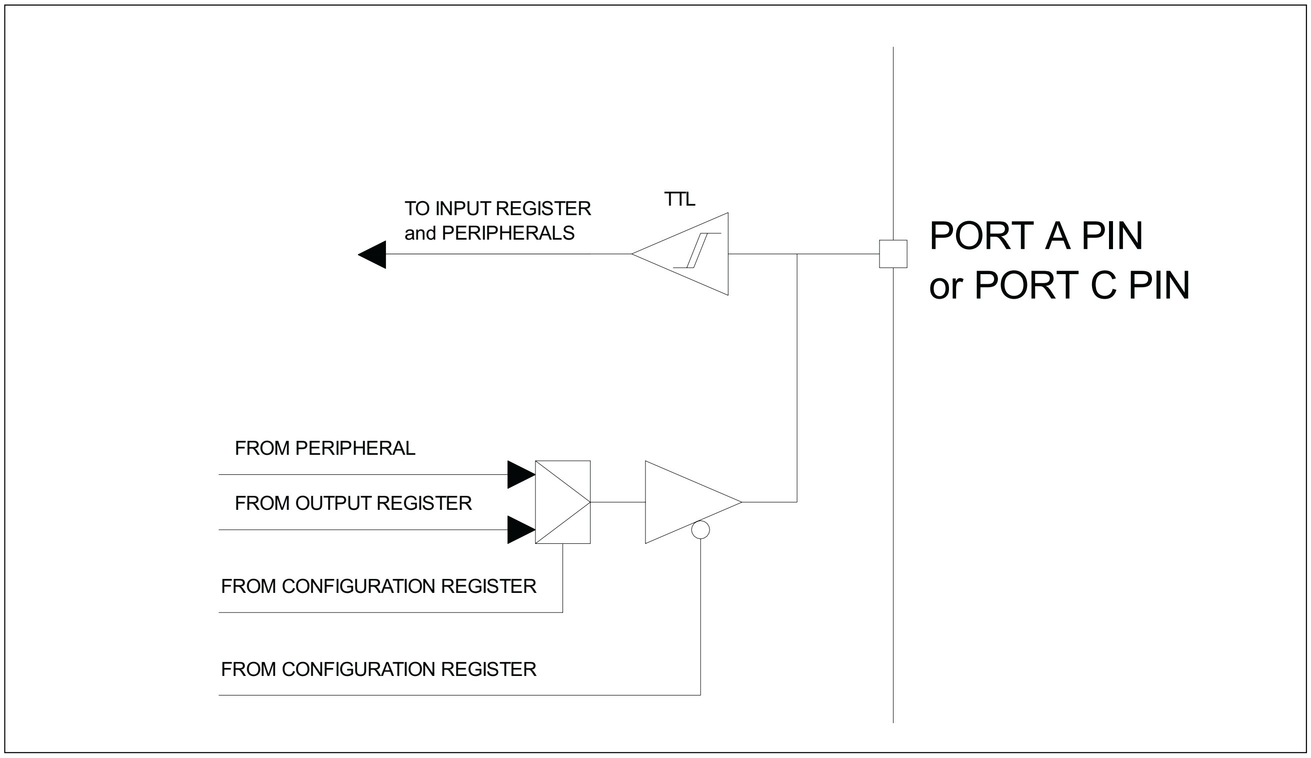


Figure 6.2 Port B Functional Blocks

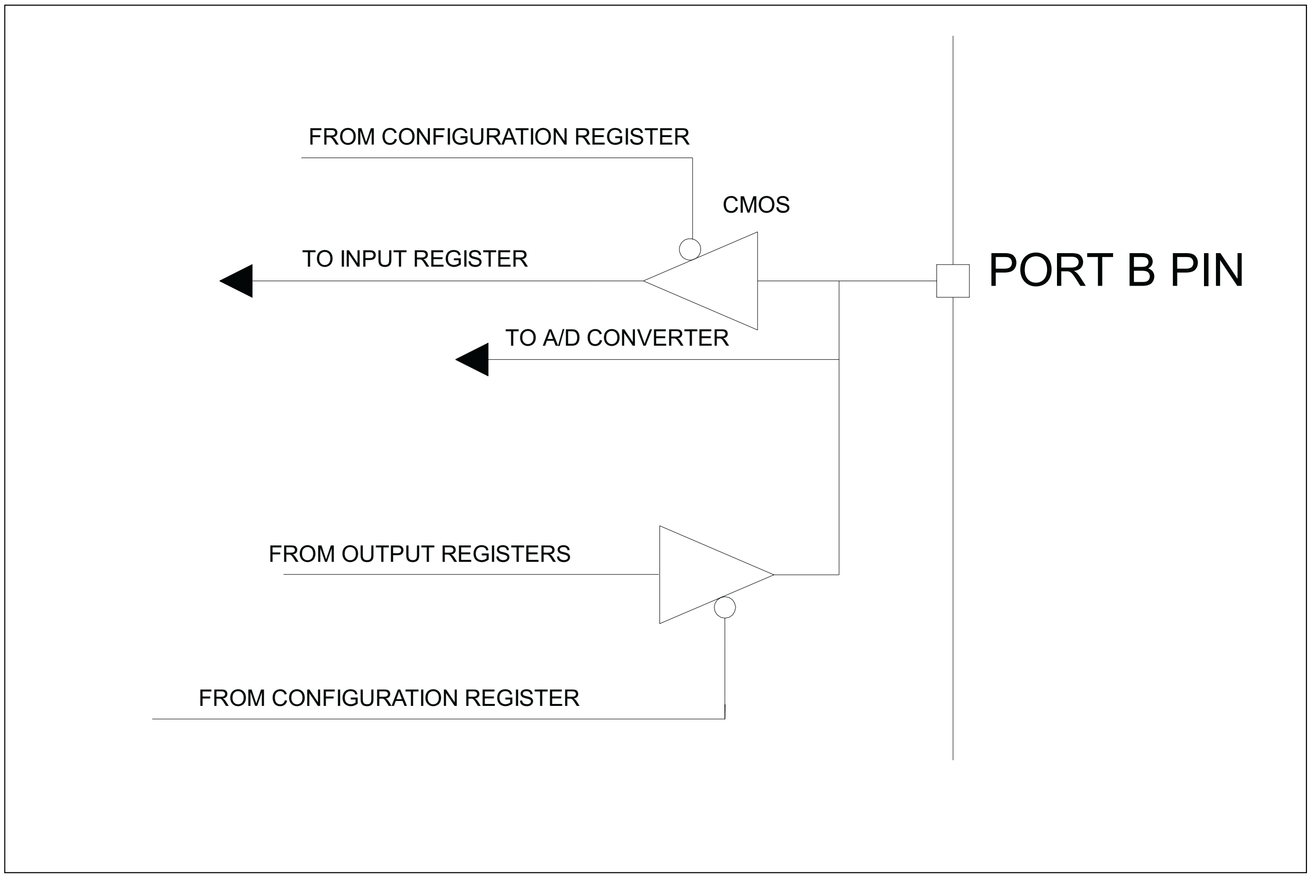


Table 6.2 Input Register and I/O Ports

PORT A	PORT B	PORT C
IR 9	IR 10	IR 11

6.3 Output Mode

The output configuration is selected setting to ‘0’ the corresponding configuration register bit (REG_CONF 4, 13 and 15) (see paragraph 6.5).

The digital data are transferred to the related I/O Port by means of the Output register, by using the assembler instructions LDPE or LDPR.

Table 6.3 Output Register and I/O Ports

PORT A	PORT B	PORT C
OR 0	OR 1	OR 2

6.4 Alternate Functions

Several ST52x430 pins are configurable to be used with different functions (see table 1.1).

When an on-chip peripheral is configured to use a pin, it is mandatory to select the correct I/O mode of the related pin.

For example: if the pin 26 (PA5/T0CLK in the SO34) has to be used like external PWM/Timer0 clock, the Reg_Conf 4(5) bit must be set to ‘1’.

When the signal is an input of an on-chip peripheral, the related I/O pin has to be configured in Input Mode.

When a pin is used as an A/D Converter input, the related I/O pin is automatically set in tristate. The analog multiplexer (controlled by the A/D configuration Register) switches the analog voltage present on the selected pin to the common analog rail which is connected to the ADC input.

It is recommended not to change the voltage level or loading on any port pin while conversion is running. Furthermore it is recommended not to have clocking pins located close to a selected analog pin.

6.5 I/O Port Configuration Registers

The I/O mode for each bit of the three ports are selected by using the Configuration Registers 4, 13 and 15 (See Table 6.1) The structure of these registers is shown in the following tables.

Each bit of the configuration registers sets the I/O mode of the related port pin.

Table 6.4 Ports A REG_CONF 4

Bit	Name	Value	Description
0	D0	0	Set the pin PA0/T0RES in Output Mode
		1	Set the pin PA0/T0RES in Input Mode
1	D1	0	Set the pin PA1/$\overline{\text{T0OUT}}$ in Output Mode
		1	Set the pin PA1/$\overline{\text{T0OUT}}$ in Input Mode
2	D2	0	Set the pin PA2/$\overline{\text{T1OUT}}$ in Output Mode
		1	Set the pin PA2/$\overline{\text{T1OUT}}$ in Input Mode
3	D3	0	Set the pin PA3/$\overline{\text{T2OUT}}$ in Output Mode
		1	Set the pin PA3/$\overline{\text{T2OUT}}$ in Input Mode
4	D4	0	Set the pin PA4/T0STRT in Output Mode
		1	Set the pin PA4/T0STRT in Input Mode
5	D5	0	Set the pin PA5/T0CLK in Output Mode
		1	Set the pin PA5/T0CLK in Input Mode
6	D6	0	Set the pin PA6 in Output Mode
		1	Set the pin PA6 in Input Mode
7	D7	0	Set the pin PB7/PA7/Ain7 in Output Mode
		1	Set the pin PB7/PA7/Ain7 in Input Mode
Reset Configuration ‘11111111’			

Table 6.5 Ports B REG_CONF 13

Bit	Name	Value	Description
0	D0	0	Set the pin PB0/Ain0 in Output Mode
		1	Set the pin PB0/Ain0 in Input Mode
1	D1	0	Set the pin PB1/Ain1 in Output Mode
		1	Set the pin PB1/Ain1 in Input Mode
2	D2	0	Set the pin PB2/Ain2 in Output Mode
		1	Set the pin PB2/Ain2 in Input Mode
3	D3	0	Set the pin PB3/Ain3 in Output Mode
		1	Set the pin PB3/Ain3 in Input Mode
4	D4	0	Set the pin PB4/Ain4 in Output Mode
		1	Set the pin PB4/Ain4 in Input Mode
5	D5	0	Set the pin PB5/Ain5 in Output Mode
		1	Set the pin PB5/Ain5 in Input Mode
6	D6	0	Set the pin PB6/Ain6 in Output Mode
		1	Set the pin PB6/Ain6 in Input Mode
7	D7	0	Set the pin PB7/PA7/Ain7 in Output Mode
		1	Set the pin PB7/PA7/Ain7 in Input Mode
Reset Configuration ‘11111111’			

Table 6.6 Port C REG_CONF 15

Bit	Name	Value	Description
0	D0	0	Set the pin INT/PC0 in Output Mode
		1	Set the pin INT/PC0 in Input Mode
1	D1	0	Set the pin T0OUT/PC1 in Output Mode
		1	Set the pin T0OUT/PC1 in Input Mode
2	D2	0	Set the pin T1OUT/PC2 in Output Mode
		1	Set the pin T1OUT/PC2 in Input Mode
3	D3	0	Set the pin T2OUT/PC3 in Output Mode
		1	Set the pin T2OUT/PC3 in Input Mode
4	D4	0	Set the pin Tx/PC4 in Output Mode
		1	Set the pin Tx/PC4 in Input Mode
5	D5	0	Set the pin Rx/PC5 in Output Mode
		1	Set the pin Rx/PC5 in Input Mode
6	D6	0	Set the pin PC6 in Output Mode
		1	Set the pin PC6 in Input Mode
7	D7	0	Set the pin PC7 in Output Mode
		1	Set the pin PC7 in Input Mode
Reset Configuration ‘11111111’			

Analog Input Option. The PB0-PB7 pins can be configured to be analog inputs according to the codes programmed in the configuration register REG_CONF 14 (See Table 6.7). These analog inputs are connected to the on chip 8-bit Analog to Digital Converter.

Table 6.7 Analog Inputs (REG_CONF 14)

Bit	Name	Value	Description
0	D0	0	pin PB0/Ain0 Digital I/O
		1	pin PB0/Ain0 Analog Input
1	D1	0	pin PB1/Ain1 Digital I/O
		1	pin PB1/Ain1 Analog Input
2	D2	0	pin PB2/Ain2 Digital I/O
		1	pin PB2/Ain2 Analog Input
3	D3	0	pin PB3/Ain3 Digital I/O
		1	pin PB3/Ain3 Analog Input
4	D4	0	pin PB4/Ain4 Digital I/O
		1	pin PB4/Ain4 Analog Input
5	D5	0	pin PB5/Ain5 Digital I/O
		1	pin PB5/Ain5 Analog Input
6	D6	0	pin PB6/Ain6 Digital I/O
		1	pin PB6/Ain6 Analog Input
7	D7	0	pin PB7/Ain7 Digital I/O
		1	pin PB7/Ain7 Analog Input
Reset Configuration ‘11111111’			

PWM/Timers Alternate Functions

The pins of the Port A and C can be configured to be I/O of the three PWM/Timers available on the ST52x430. The configuration of these pins is performed by using the Configuration Registers REG_CONF 12 and REG_CONF 16 if the related pin has to be outputs. When the related pin has to be used as peripherals' input the configuration is performed by the relative peripherals' configuration registers (See PWM/Timer Session).

Table 6.9 PWM/Timers REG_CONF 12

Bit	Name	Value	Description
0	PA1	1	Pin PA1/$\overline{\text{T0OUT}}$ is configured as PWM/Timer 0 negated output
		0	Pin PA1/$\overline{\text{T0OUT}}$ is configured as Port A Digital I/O
1	PA2	1	Pin PA2/$\overline{\text{T1OUT}}$ is configured as PWM/Timer 1 negated output
		0	Pin PA2/$\overline{\text{T1OUT}}$ is configured as Port A Digital I/O
2	PA3	1	Pin PA3/$\overline{\text{T2OUT}}$ is configured as PWM/Timer 2 negated output
		0	Pin PA3/$\overline{\text{T2OUT}}$ is configured as Port A Digital I/O
3	PASZ	1	PORT A bits = 8
		0	PORT A bits = 7
4-7	NC	x	Not Used
Reset Configuration '0000'			

Table 6.8 PWM/Timers REG_CONF 16

Bit	Name	Value	Description
0	PC1	1	Pin T0OUT/PC1 is configured as Port C Digital I/O
		0	Pin T0OUT/PC1 is configured as PWM/Timer 0 output T0OUT
1	PC2	1	PinT1OUT/PC2 is configured as Port C Digital I/O
		0	Pin T1OUT/PC2 is configured as PWM/Timer 1 output T1OUT
2	PC3	1	Pin T2OUT/PC3 is configured as Port C Digital I/O
		0	Pin T2OUT/PC3 is configured as PWM/Timer 2 output T2OUT
3	PC4	1	Pin Tx/PC4 is configured as Port C Digital I/O
		0	Pin Tx/PC4 is configured as SCI output Tx
4-7	NC	X	Not Used
Reset Configuration '1111'			

Figure 6.3 Configuration Register 12

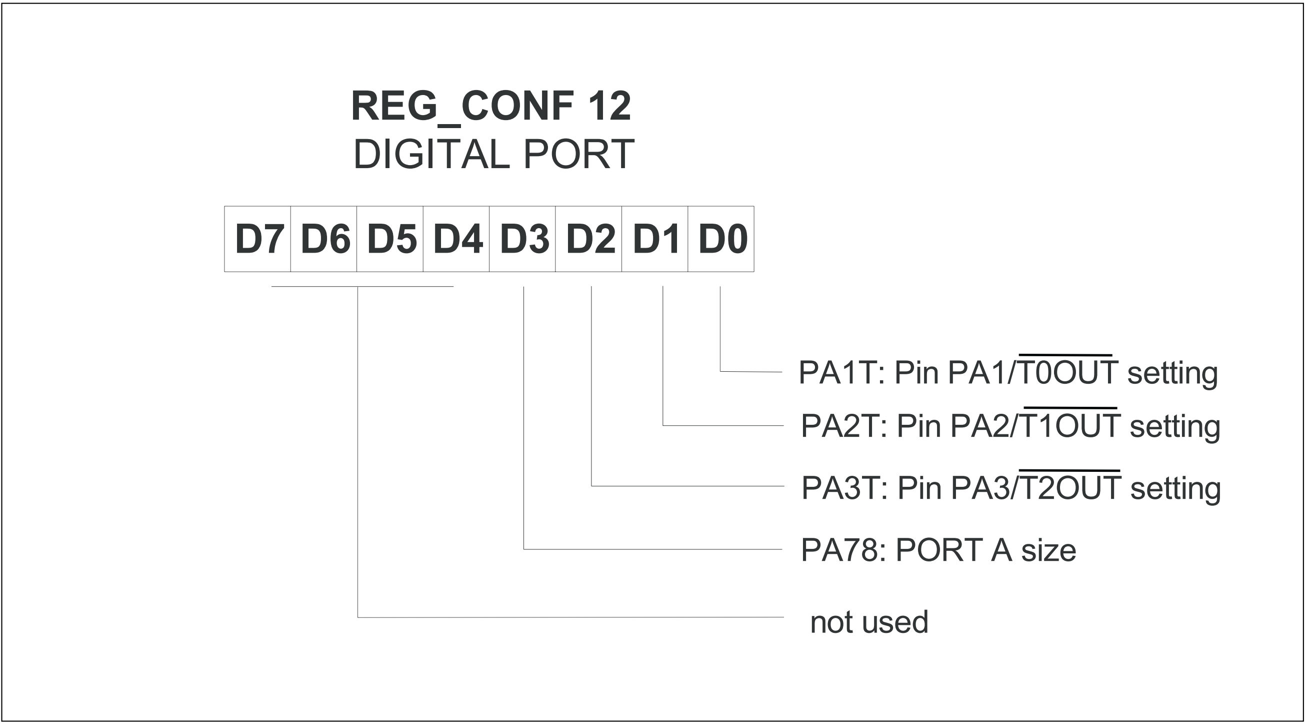
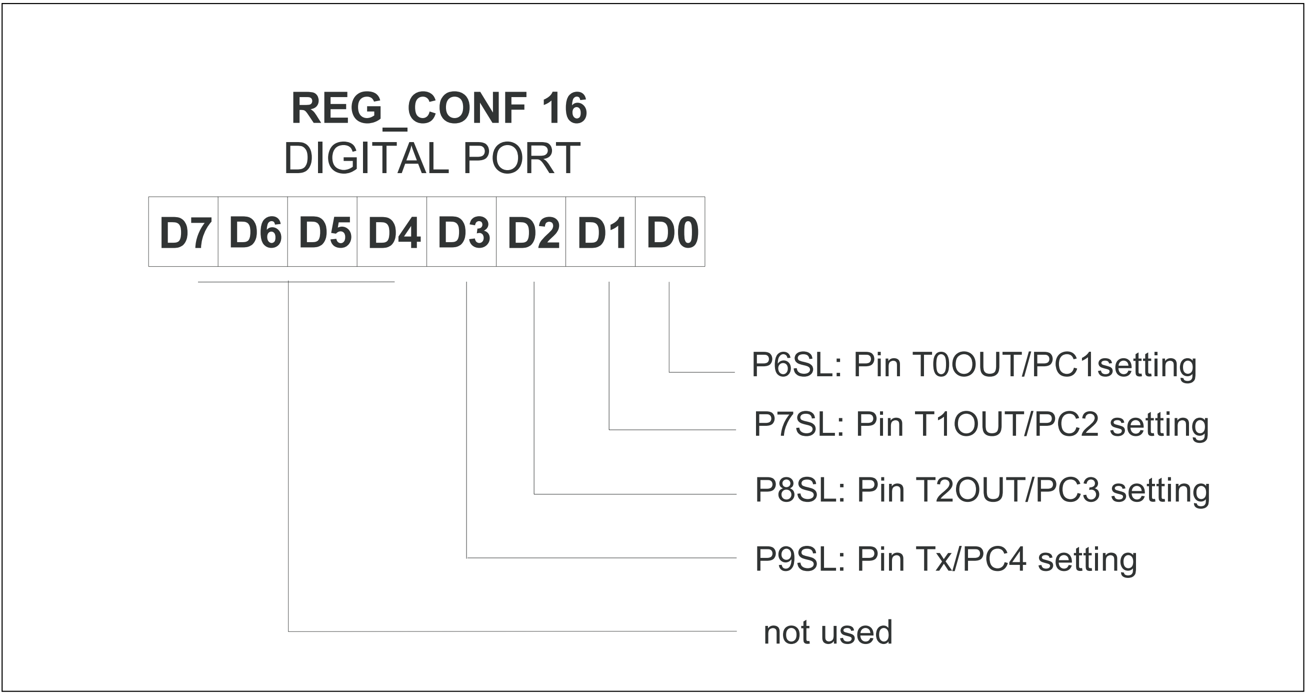

REG_CONF 12
DIGITAL PORT
D7 D6 D5 D4 D3 D2 D1 D0
PA1T: Pin PA1/$\overline{\text{T0OUT}}$ setting
PA2T: Pin PA2/$\overline{\text{T1OUT}}$ setting
PA3T: Pin PA3/$\overline{\text{T2OUT}}$ setting
PA78: PORT A size
not used

Figure 6.4 Configuration Register 16


REG_CONF 16
DIGITAL PORT
D7 D6 D5 D4 D3 D2 D1 D0
P6SL: Pin T0OUT/PC1setting
P7SL: Pin T1OUT/PC2 setting
P8SL: Pin T2OUT/PC3 setting
P9SL: Pin Tx/PC4 setting
not used

7 A/D CONVERTER

7.1 Introduction

The A/D Converter of ST52x430 is an 8-bit analog to digital converter with up to 8 analog inputs offering 8 bit resolution with a total accuracy of 1 LSB and a typical conversion time of 8.2 µs with a 20 MHz clock. This period also includes the 5.1 µs of the integral Sample and Hold circuitry, which minimizes the need for external components and allows quick sampling of the signal for the minimum warping effect and Integral conversion error.

A conversion is performed in 82 A/D clock pulses.

The A/D clock is derived from the clock master. The maximum A/D clock frequency has to be 10 MHz, for this reason when the master clock is higher than 10 MHz it has to be divided by 2 using the bit SCK of the A/D configuration register REG_CONF 3 (See table 7.1).

The A/D peripheral converts the input voltage with a process of successive approximations using a fixed clock frequency derived from the oscillator.

The conversion range is between the analog Vss and V_{DD} references.

The converter uses a fully differential analog input configuration for the best noise immunity and precision performances, along with one separate supply (V_{DDA}), allowing the best supply noise rejection.

Up to 8 multiplexed Analog Inputs are available. A group of signals can be converted sequentially by simply programming the starting address of the last analog channel to be converted.

Single or continuous conversion mode are available.

The result of the conversion is stored in an 8-bit Input Registers (from IR 1 to IR 8).

The A/D converter is controlled through the Configuration Register REG_CONF 3.

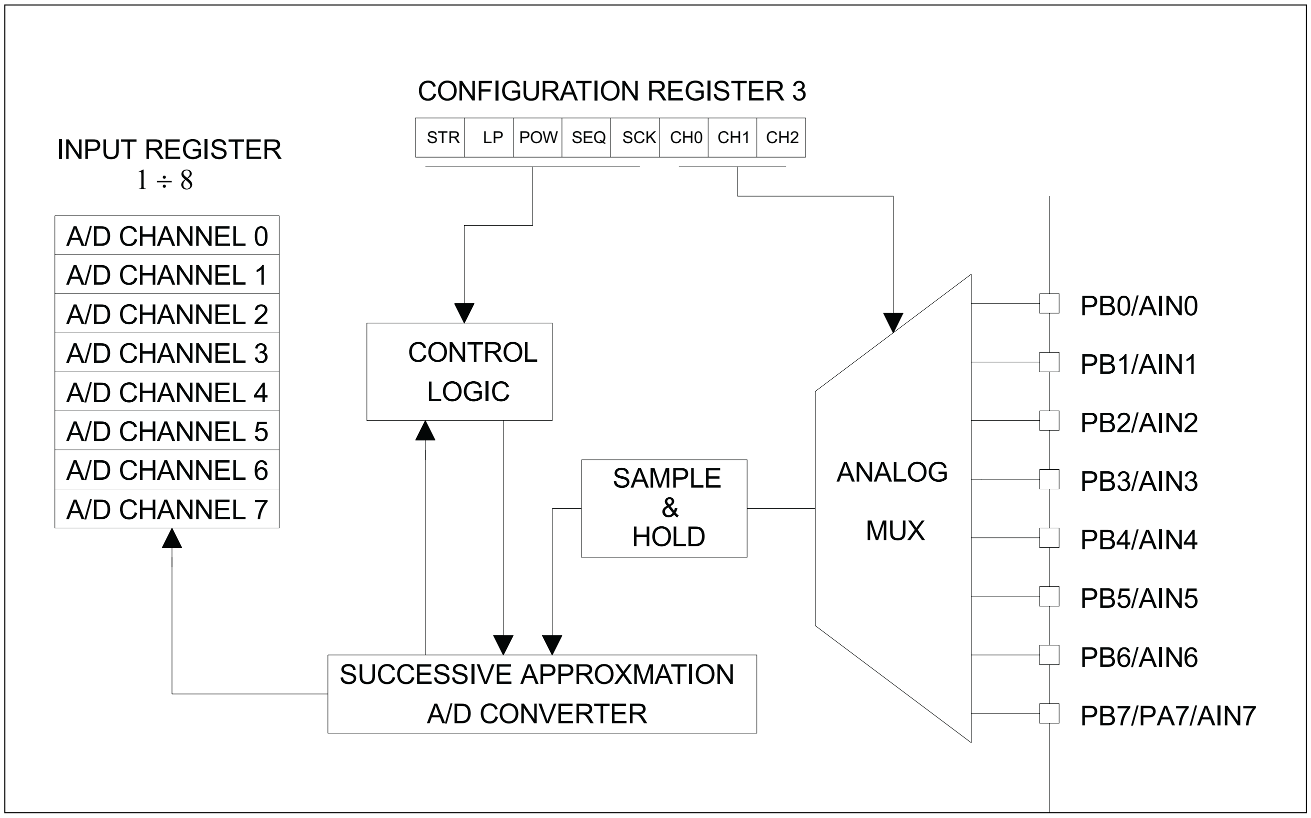
A Power-Down programmable bit allows to set the A/D converter to a minimum consumption idle status.

The ST52x430 Interrupt Unit provides one maskable channel for the End of Conversion (EOC)

7.2 Functional Description

The conversion is monotonic meaning the result never decreases if the analog input does not and never increases if the analog input does not.

Figure 7.1. A/D Converter Structure



If input voltage is greater than or equal to V_{dda} (Voltage Reference high) then the result is equal to FFh (full scale) without overflow indication.

If input voltage is less than Vss (voltage reference low) then the result is equal to 00h.

The A/D converter is linear and the digital result of the conversion is given by the formula:

$$Digital\ result = \frac{255xInput\ Voltage}{reference\ Voltage}$$

Where Reference Voltage is V_{dda} - V_{ss}.

The accuracy of the conversion is described in the Electrical Characteristics Section.

The A/D converter is not affected by the WAIT mode.

When the MCU enters HALT mode with A/D converter enabled, the converter is disabled until the HALT mode is exited and the start-up delay has elapsed. A stabilization time is also required before accurate conversions can be performed.

7.2.1 Operating Modes

Four main operating modes can be selected by setting the values of the LP and SEQ bit in the A/D configuration Register.

One Channel Single Mode

In this mode (**SEQ = '0'', LP = '0'**) the A/D provides an EOC signal after the end of channel i-th conversion; then the A/D waits for a new start event. The channel i-th is identified by the bit CH0, CH1, CH2.

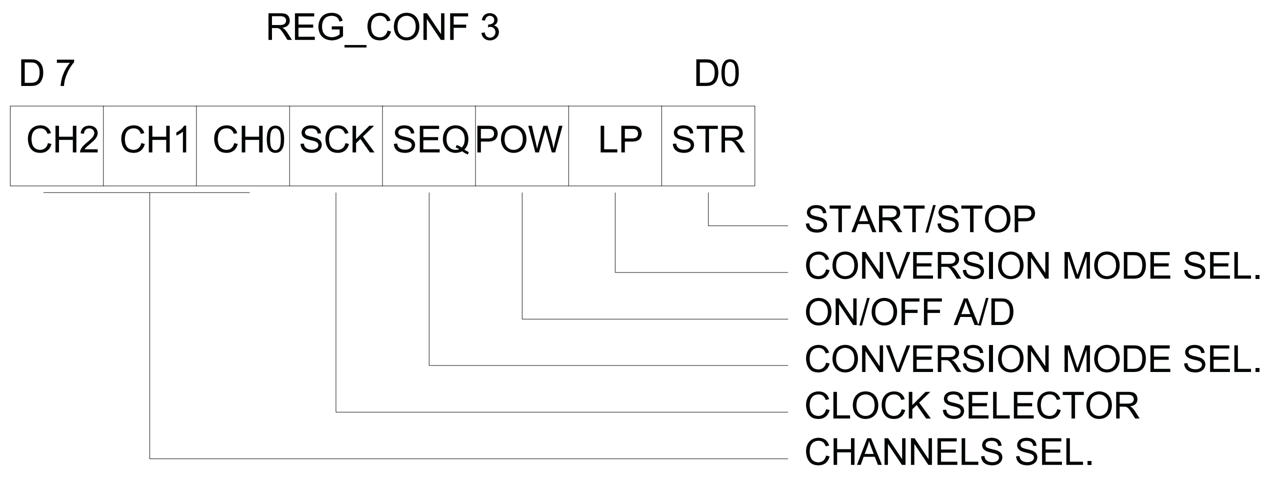
i.e CH(2:0) = '011' means conversion of channel 3 then stop.

Multiple Channels Single Mode

In this mode (**SEQ = '1', LP = '0'**) the A/D provides an EOC signal after the end of the channels sequence conversion identified by the bit CH0, CH1, CH2 ; then the A/D waits for a new start event.

i.e. CH(2:0) = '011' means conversion of channels 0,1,2 and 3 then stop.

Figure 7.2. A/D Configuration Register (REG_CONF 3)



One Channel Continuous Mode

In this mode (**SEQ = '0'', LP = '1'**) a continuous conversion flow is entered by a start event on the channel selected by the bit CH0, CH1, CH2.

i.e CH(2:0) = '011' means continuous conversion of channel 3. At the end of each conversion the relative IR is updated with the last conversion result, while the former value is lost.

To stop the conversion STR has to be set to '0'**.**

Multiple Channels Continuous Mode

In this mode (**SEQ = '1'', LP = '1'**) a continuous conversion flow is entered by a start event on the channels selected by the bits CH0, CH1, CH2.

i.e CH(2:0) = '011' means continuous conversion of channel 0,1,2 and 3.

At the end of each conversion the relative IRs are updated with the last conversion results, while the former values are lost.

To stop the conversion STR has to be set to '0'.

7.2.2 Power Down Mode

Before enabling any A/D operation mode, set the POW bit of the A/D configuration register to '1' at least 60 µs before the first conversion starts to enable the biasing circuit inside the analog section of the converter. Clearing the POW bit (POW = '0') is useful when the A/D is not used so reducing the total chip power consumption. This state is also the reset configuration and it is forced by hardware when the core is in HALT state (after a HALT instruction execution).

7.3 A/D Registers Description

The result of the conversions of the 8 available channels are loaded in the 8 Input Register from the decimal address 1 to the decimal address 8. (IR (1:8) see table 2.2)). Every IR(1:8) is reloaded with a new value at the end of the conversion of the correspondent analog input.

By using the assembler instruction:

```
LDRI RAM_Reg. IR_i
```

the value stored in the i-th IR is transferred on the RAM location RAM_Reg.

The A/D configuration register is the REG_CONF 3. The figure 6.2 shows the structure of this register. This register manages the A/D logic operation. The A/D configuration register (REG_CONF 3) is programmable as following:

b7-b5 = **CH2, CH1, CH0**: Last Conversion Address. These 3 bits define the last analog input. The first analog input is converted, then the address is incremented for the successive conversion, until the channel identified by CH0-CH2 is converted. The (CH2, CH1, CH0) bits define the group of channels to be scanned. When setting CH2=0 CH1=0 CH0=0 only channel 0 is converted.

b4 = **SCK**: Master clock divider. The ST52x430 is able to work with a clock frequency up to 20 MHz. The SCK must be set to '1' when the ST52x430 clock is higher then 10 MHz. It is useful to set SCK = '1' also when the clock master is lower than 10 MHz and a high accuracy is required.

b3 = **SEQ**: Multiple/Single channel. When SEQ is set to '0' a the channel identified by CH(2:0) is converted. If SEQ is set to '1' the group of channels identified by CH(2:0) are converted.

b2= **POW**: Power Up/ Power Down. A logical '1' enables the A/D logic and analog circuitry.

A logical level '0' disables all power consuming logic, thus allowing a low power idle status.

b1 = **LP**: Continuous/Single. When this bit is set to '1' (continuous mode), the first conversions sequence are started by the STR bit then a continuous conversion flow is processed.

When LP='0' (single mode) only one sequence of conversions is started when STR is set.

b0 = **STR**: Start/Stop. A logical level '1' enables the starting of a conversion sequence; a logical level '0' stops the conversion. When the A/D is running in the Single Modes (LP='0'), this bit is

Table 7.10 A/D Conf. Register (Reg_Conf 3)

Bit	Name	Value	Description
0	STR	0	Stop Conversion
		1	Start Conversion
1	LP	0	Single Conversion
		1	Continuous Conversions
2	POW	0	A/D OFF
		1	A/D ON
3	SEQ	0	Single Channel Conv.
		1	Multiple Channels Conv
4	SCK	0	Clock not Divided
		1	Clock Divided
5	CH(2:0)	000	Channel 0
		001	Channel 1
		010	Channel 2
6		011	Channel 3
		100	Channel 4
7		101	Channel 5
		110	Channel 6
		111	Channel 7

8 WATCHDOG TIMER

8.1 Functional Description

The Watchdog Timer (WDT) is used to detect the occurrence of a software fault, usually generated by external interference or by unforeseen logical conditions, which causes the application program to abandon its normal sequence. The WDT circuit generates an MCU reset on expiry of a programmed time period, unless the program refreshes the WDT before the end of the programmed time delay.

16 different delays can be selected by using the WDT configuration register.

If the WDT is activated (by using the assembler instruction `WDTSFR`) after the end of the delay programmed by the configuration register, it starts a reset cycle pulling low the reset pin.

The application program once activated the WDT has to refresh this peripheral (by the `WDTSFR` instruction) at regular intervals during normal operation to prevent an MCU reset.

To stop the WDT during the user program executions the instruction `WDTSLP` has to be used.

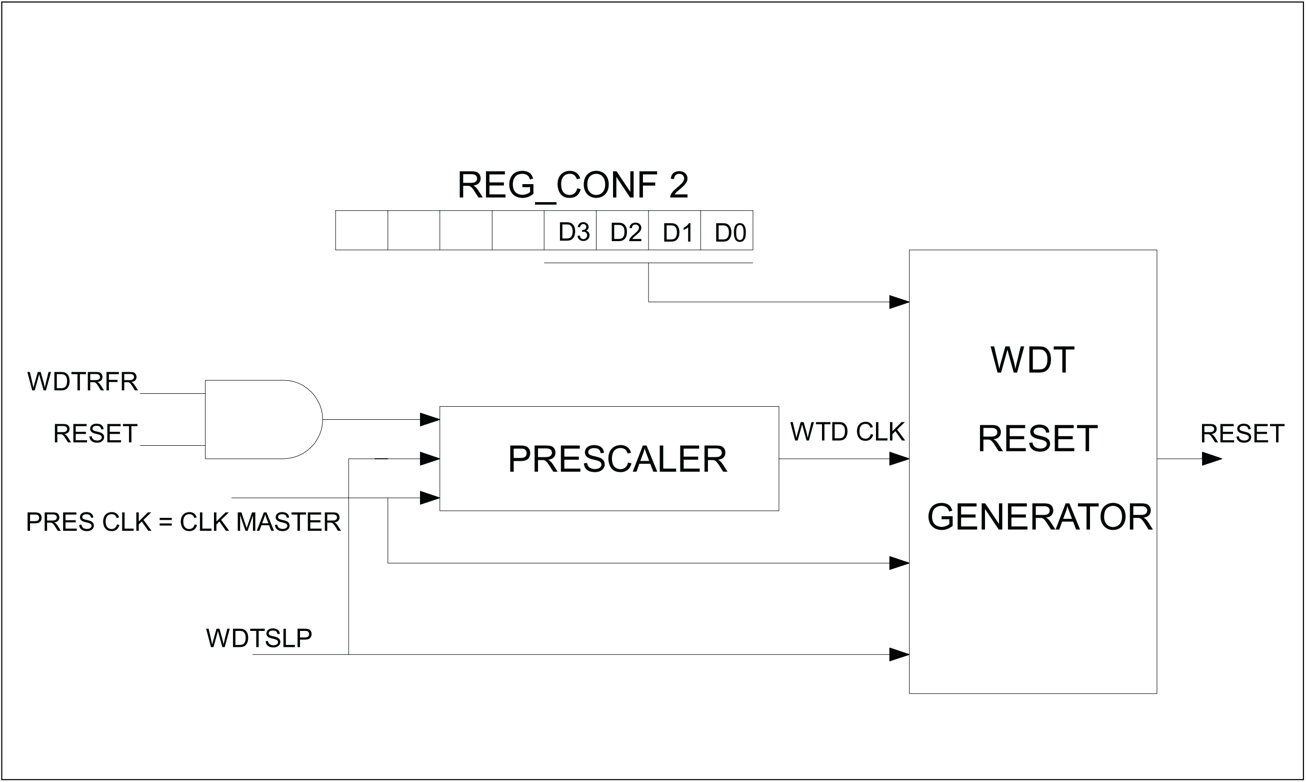
The working frequency of the WDT (PRES CLK in the Figure 8.1) is equal to the clock master. The clock master is then divided by 500 thus obtaining the WDT CLK signal that is used to fix the timeout of the WDT.

Table 8.1 Watchdog Timing range (CLK=5 MHz)

	WDT timeout period (ms)
min	0.1
max	937.5

According to the WDT configuration register values it possible to define a WDT delay between 0.1 ms and 937.5 mS when the clock master is 5 MHz. Changing the clock master frequency the timeout delay can be calculated according to the configuration register values REG_CONF 2 as described in the following section.

Figure 8.1 Watchdog Block Diagram



8.2 Register Description

The WDT timeout is defined setting the value of the REG_CONF 2. The first 4 bits of this register are used thus obtaining 16 different delays as shown in the table 8.2. In the table 8.2 the timeout is expressed by using the number of WDT CLK. The WDT CLK is derived from the clock master by a division factor of 500. The Timeout is then obtained by multiplying the WDT CLK pulse length for the number of pulses defined by the configuration register REG_CONF 2. The Table 8.4 shows the pulses length for typical values of the clock master.

The Table 8.3 shows the timeout WDT values when the Master Clock is 5 MHz.

Table 8.2 WDT REG_CONF 2

Bit	Name	Value	Timeout Values (WDT CLK pulses)
0	D(3:0)	0000	1
		0001	625
		0010	1250
		0011	1875
1		0100	2500
		0101	3125
		0110	3750
		0111	4375
2		1000	5000
		1001	5625
		1010	6250
		1011	6875
3		1100	7500
		1101	8125
		1110	8750
		1111	9375
4-7	NC	x	Not Used
Reset Configuration ‘0000’			

Table 8.3 Timeout Values with CLK=5 MHz

Bit	Name	Value	Timeout Values (ms)
0	D(3:0)	0000	0.1
		0001	62.5
		0010	125
		0011	187.5
1		0100	250
		0101	312.5
		0110	375
		0111	437.5
2		1000	500
		1001	562.5
		1010	625
		1011	687.5
3		1100	750
		1101	812.5
		1110	875
		1111	937.5
4-7	NC	x	Not Used
Reset Configuration ‘0000’			

Table 8.4 Typical WDT CLK Pulse Length

MASTER CLK (MHz)	WDT CLK (KHz)	WDT CLK PULSE LENGTH (ms)
4	0.8	0.125
5	1	0.1
8	1.6	0.0625
10	2	0.05
20	4	0.025

9 PWM/TIMER

ST52x430 offers three on-chip PWM/Timer peripherals :TIMER0, TIMER1 and TIMER2.

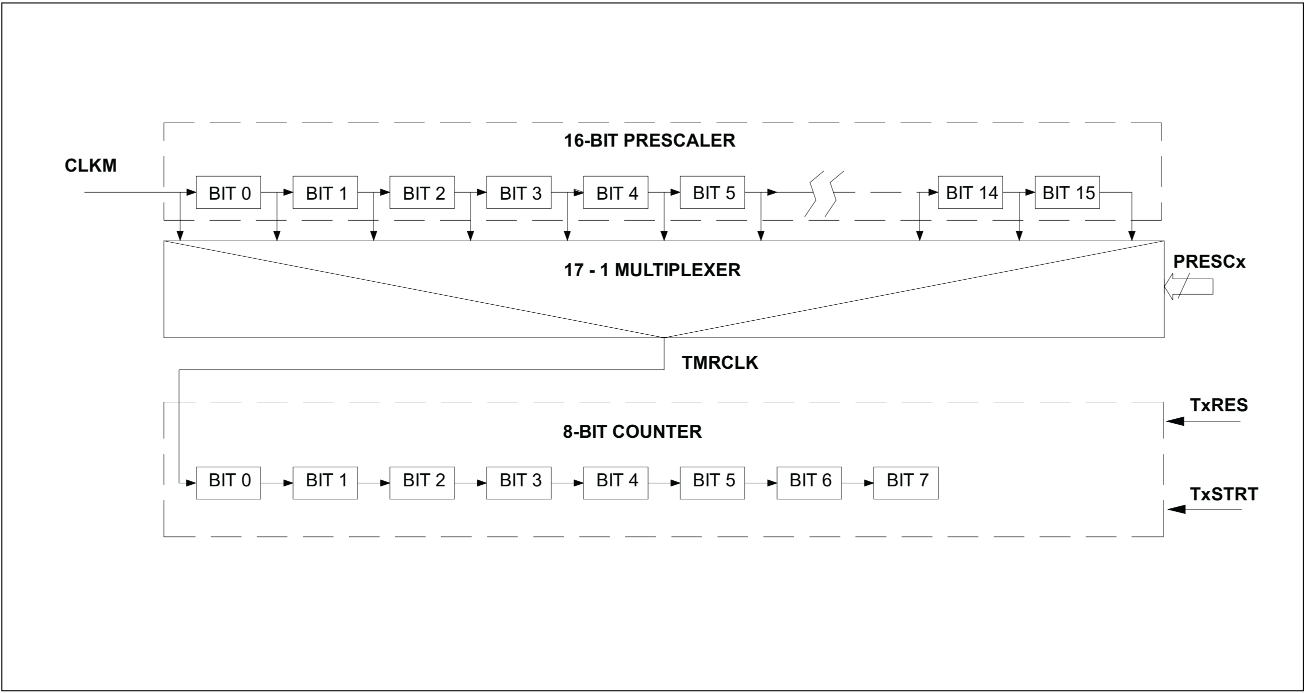
The ST52x430 timers have the same internal structure. Basically the timer consists of an 8-bit counter with a 16-bit programmable prescaler, thus giving a maximum count of 2^{24} (see figure 9.1).

CLOCK: PA4/T0STRT, PA0/T0RES and PA5/T0CLK pins.

REMARKS: To use T0RST, T0STR, T0CLK external signals the related pins must be configured in Input Mode by setting REG_CONF4 and REG_CONF7 registers (see table 6.4 and 9.3)

For each timer, the content of the 8-bit counter is incremented on the Rising Edge of the 16-bit prescaler output (PRESCOUT) and it can be read

Figure 9.1 Timer Peripheral Block Diagram



Following, the generic timer is called Timer x, where x can be 0, 1 or 2.

Each timer has two different working modes, that can be selected by setting the correspondent TxMODE bits of REG_CONF5, REG_CONF8 and REG_CONF10 registers: Timer Mode and PWM (Pulse Width Modulation) Mode.

All the Timers have Autoreload Functions in PWM Mode.

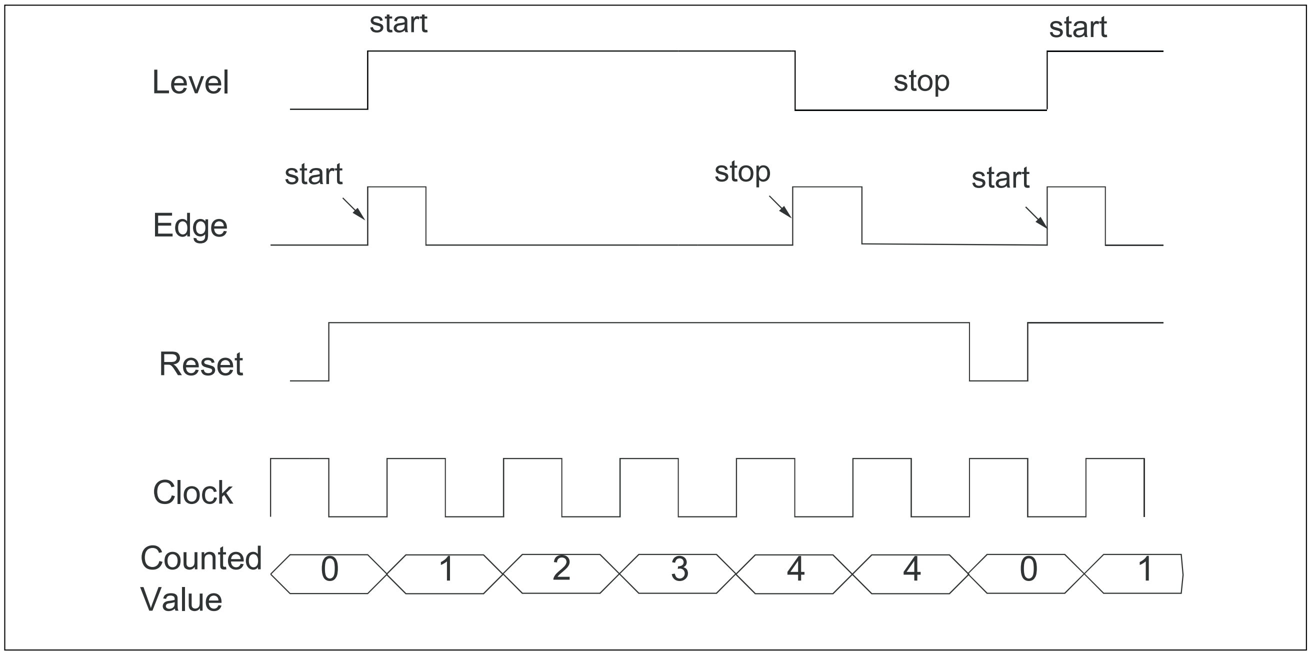
Each timer output is available, with its negated signal, on external pins, by setting PAx and PCx bits of REG_CONF12 and REG_CONF16 (see tables 9.8 and 9.9).

REMARKS: To enable a timer output (TxOUT or TxOUT) the related pin must be configured in Output Mode by setting REG_CONF4, and REG_CONF15 registers (see table 6.4 and 6.6)

In particular, TIMER0 can utilize also external START/STOP signals (Input capture and Output compare), external RESET signal, and external

at any instant of the counting phase then saved in a location of the RAM memory. The PWM/Timer x Counter value can be read from the Input Register PWM_x_COUNT (Input Registers 12, 14 or 16. See table 2.2)

Figure 9.2. Timer 0 External START / STOP Mode



The PWM/Timer x Status can be read from the Input Register PWM_x_STATUS (Input Registers 13, 15 or 17. See tables 2.2 and 9.10).

9.1 Timer Mode

Timer Mode is selected fixing TxMODE bit of REG_CONF5, REG_CONF8 and REG_CONF10 equal to 0 (see tables 9.1, 9.4 and 9.6).

Each TIMERx requires three signals: Timer Clock (TMRCLKx), Timer Reset (TxRES) and Timer Start (TxSTRT) (see Figure 9.1). Each of these signals can be generated internally, or, only for Timer 0, externally by setting T0RST, T0STR, T0CLK bits of REG_CONF7 register.

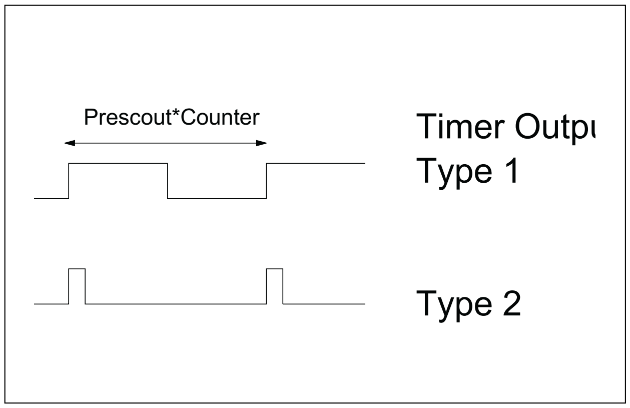
TMRCLKx is the Prescaler x output, that increments, on the rising edge, the Counter x value. TMRCLKx is obtained from the internal clock signal (CLKM) or, only for TIMER0, from the external signal provided on the PA5/T0CLK pin.

NOTE: The external clock signal, applied on T0CLK pin, must have a frequency at least two times smaller than the internal master clock.

The prescaler output can be selected by setting PRESCx bit of REG_CONF6, REG_CONF9 and REG_CONF11 registers (see tables 9.2, 9.5 and 9.7).

TxRES resets to zero the content of the 8-bit counter x. It is generated by the TIRSTx and TxMSK bits of REG_CONF5, REG_CONF7, REG_CONF8 and REG_CONF10 registers (see tables 9.1, 9.3, 9.4 and 9.6).

Figure 9.3. TIMEROUT Signal Type



TxSTRT signal start/stop the Timer x counting only if the peripherals are configured in Timer mode. This signal is forced by setting the correspondent TISTRx bit of REG_CONF5, REG_CONF8 and REG_CONF10 registers (see tables 9.1, 9.4 and 9.6).

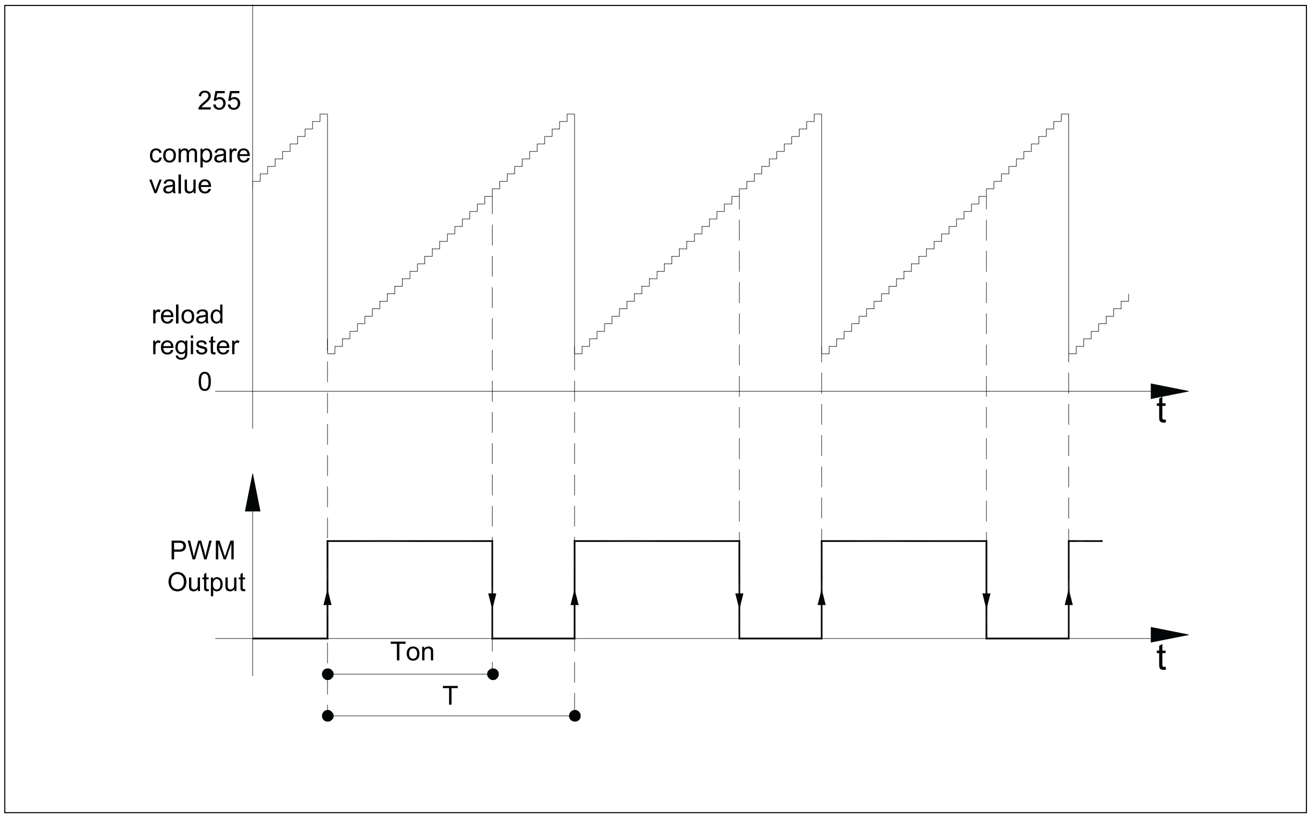
TxMSK bits mask the reset of each timer and, for this reason, they can be utilized to synchronize a simultaneous start of the timers, by means, for example, of the following procedure that starts three timers:

1) TIRST0 = TIRST1 = TIRST2 = 0,

2) TISTR0 = TISTR1 = TISTR2 = 0,

3) T0MSK = T1MSK = T2MSK = 1,

Fig. 9.4 PWM Mode with Auto Reload



4) TIRST0 = TIRST1 = TIRST2 = 1,

5) TISTR0 = TISTR1 = TISTR2 = 1,

6) T0MSK = T1MSK = T2MSK = 0,

(all timers start simultaneously)

when TxMSK is 1 the TIMER x is reset.

TIMER 0 START/STOP can be given externally on the T0STRT pin. In this case, T0STRT signal allows to work in two different modes, by setting the TESTR configuration bit of REG_CONF5 register (see figure 9.2) (Input capture):

LEVEL (Time Counter): If the T0STRT signal is high the Timer starts the count. When the T0STRT is low the counting is stopped and the current value is stored in the PWM_0_COUNT Input Register.

EDGE(Period Counter): After the reset, on the first T0STRT rising edge, the TIMER 0 starts the counting and, at the next rising edge, it is stopped. In this way it is possible to measure the period of an external signal.

The Timer x output signal, TIMERxOUT, is a signal with a frequency equal to the 16 bit-Prescaler x output signal, TMRCLKx, divided by the Output Register PWM_x_COUNT value (8 bit) (Output Registers 3, 5 or 7. See table 2.4), that is the value to count.

TIMERxOUT waveform can be of two types:

type 1: TIMERxOUT waveform equal to a square wave with a 50% duty-cycle

type 2: TIMERxOUT waveform equal to a pulse signal with the pulse duration equal to the Prescaler x output signal.

For each Timer x, the TIMERxOUT waveform type can be selected by setting the correspondent TMRWx bit of REG_CONF6, REG_CONF9 and REG_CONF11 registers (see tables 9.2, 9.5 and 9.7)

9.2 PWM Mode

The PWM working mode, for each timer, is obtained by setting at 1 the correspondent TxMODE bits of REG_CONF5, REG_CONF8 and REG_CONF10 registers (see tables 9.1, 9.4 and 9.6).

TIMERxOUT, in PWM Mode, consists of a signal, with a fixed period, whose duty cycle can be modified by the user.

The TIMERxOUT signal can be available on TxOUT pin and the $\overline{\text{TIMERx}}$OUT inverted signal can be available on $\overline{\text{TxOUT}}$ pin, by setting PxSL bits of

REG_CONF12 and REG_CONF16 (see tables 9.8 and 9.9)

The PWM TIMERxOUT period can be fixed, by setting the 16-bit prescaler x output and an initial autoreload 8-bit counter value stored in the Output Register PWM_x_RELOAD, as shown in figure 9.4.

The Output Register PWM_x_RELOAD value is automatically reloaded when Counter x restarts to count.

The 16-bit Prescaler x divides the master clock, CLKM, or, only for TIMER0, the external T0CLK signal, by the 16-bit Prescaler x.

NOTE: The external clock signal, applied on T0CLK pin, must have a frequency at least two times smaller than the internal master clock.

The Prescaler x output can be selected by setting PRESCx bit of REG_CONF6, REG_CONF9 and REG_CONF11 registers (see tables 9.2, 9.5 and 9.7).

When the Counter x reaches the Peripheral Register PWM_x_COUNT value (Compare Value), the TIMERxOUT signal changes from high to low level, up to the next counter start.

The period of the PWM signal is obtained by using the following relation:

$$T = (255 - PWM\_x\_RELOAD) \times TMRCLKx$$

where TMRCLKx is the output of the 16-bit prescaler x.

The duty cycle of the PWM signal is controlled by the Output Register PWM_x_COUNT:

$$Ton = (PWM\_x\_COUNT - PWM\_x\_RELOAD) * TMRCLKx$$

If the Output Register PWM_x_COUNT value is 255 the TIMERxOUT signal is always at high level.

If the Output Register PWM_x_COUNT is 0, or less than the PWM_x_RELOAD value, TIMERxOUT signal is always at low level.

NOTE. If PWM_x_RELOAD value increases the duty cycle resolution decreases.

By using a 20 MHz clock master it is possible to obtain a PWM frequency in the range 1.2 Hz to 78.43 Khz.

NOTE: The Timers, before to use a new values of the counter or of the reload, has to complete the previous counting phase. If the counter/reload value is changed during a counting, the new value of the timer counter is used only at the end of the previous counting phase. This happens both in timer and in PWM mode.

When the Timers are in Reset, or when the device is reset, the TxOut pins go in threestate, then it is recommended to put a pull-up or a pull-down resistor if these output are used to drive external devices.

9.3 Timer Interrupt

The TIMERx can be programmed to generate an Interrupt request until the end of the count or when there is an external TSTART signal. The Timer can generate programmable Interrupts into 4 different modes:

Interrupt mode 1: Interrupt on counter Stop.

Interrupt mode 2: Interrupt on Rising Edge of TIMEROUT.

Interrupt mode 3: Interrupt on Falling Edge of TIMEROUT.

Interrupt mode 4: Interrupt on both edges of TIMEROUT.

The Interrupt mode can be selected by means of INTSLx and INTEx bits of the REG_CONF5, REG_CONF8 and REG_CONF10 registers (see tables 9.1, 9.4 and 9.6).

Table 9.1. Configuration Register 5 Description

Bit	Name	Value	Description
0	TIRST0	0	Internal RESET enabled
		1	Internal RESET disabled
1	TERST	0	External RESET on Level
		1	External RESET on Edge
2	TISTR0	0	Internal STOP
		1	Internal START
3	TESTR	0	External START on Level
		1	External START on Edge
4	INTE0	00	TIMER0 Interrupt on TIMER0OUT Falling Edge
		01	TIMER0 Interrupt on TIMER0OUT Rising Edge
5		10	TIMER0 Interrupt on Both Edges of TIMER0OUT
		11	- not used
6	INTSL0	0	TIMER0 Interrupt on Counter Stop
		1	TIMER0 Interrupt on TIMER0OUT
7	T0MODE	0	TIMER MODE
		1	PWM MODE

Figure 9.5. Configuration Register 5

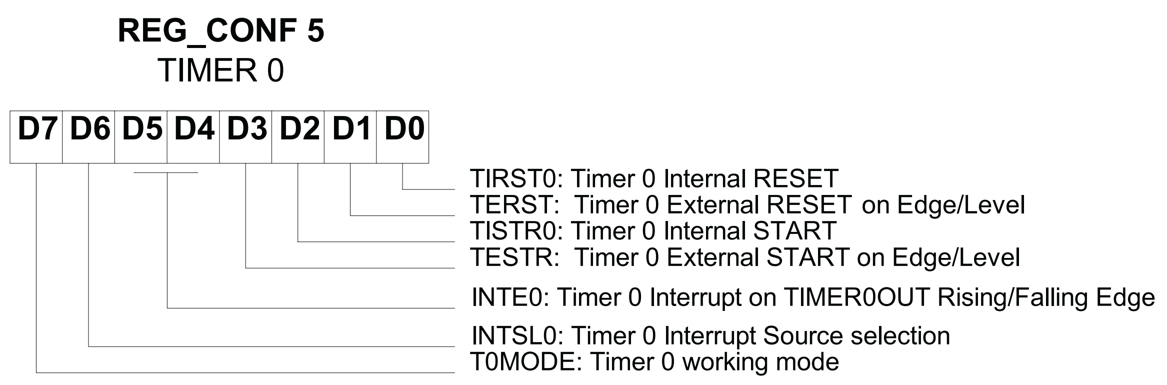

REG_CONF 5
TIMER 0
D7
D6
D5
D4
D3
D2
D1
D0
TIRST0: Timer 0 Internal RESET
TERST: Timer 0 External RESET on Edge/Level
TISTR0: Timer 0 Internal START
TESTR: Timer 0 External START on Edge/Level
INTE0: Timer 0 Interrupt on TIMER0OUT Rising/Falling Edge
INTSL0: Timer 0 Interrupt Source selection
T0MODE: Timer 0 working mode

Table 9.2. Configuration Register 6 Description

Bit	Name	Value	Description
0	PRESC0	00000	TIMER0 Clock = CLKM / 1
		00001	TIMER0 Clock = CLKM / 2
		00010	TIMER0 Clock = CLKM / 4
		00011	TIMER0 Clock = CLKM / 8
1		00100	TIMER0 Clock = CLKM / 16
		00101	TIMER0 Clock = CLKM / 32
		00110	TIMER0 Clock = CLKM / 64
2		00111	TIMER0 Clock = CLKM / 128
		01000	TIMER0 Clock = CLKM / 256
		01001	TIMER0 Clock = CLKM / 512
3		01010	TIMER0 Clock = CLKM/1024
		01011	TIMER0 Clock = CLKM/2048
		01100	TIMER0 Clock = CLKM/4096
4		01101	TIMER0 Clock = CLKM/8192
		01110	TIMER0 Clock=CLKM/16384
		01111	TIMER0 Clock=CLKM/32768
		10000	TIMER0 Clock=CLKM /65536
5	TMRW0	0	TIMER0OUT Waveform equal to pulse wave
		1	TIMER0OUT Waveform equal to square wave
6	-	-	- not used
7	-	-	- not used

Figure 9.6. Configuration Register 6

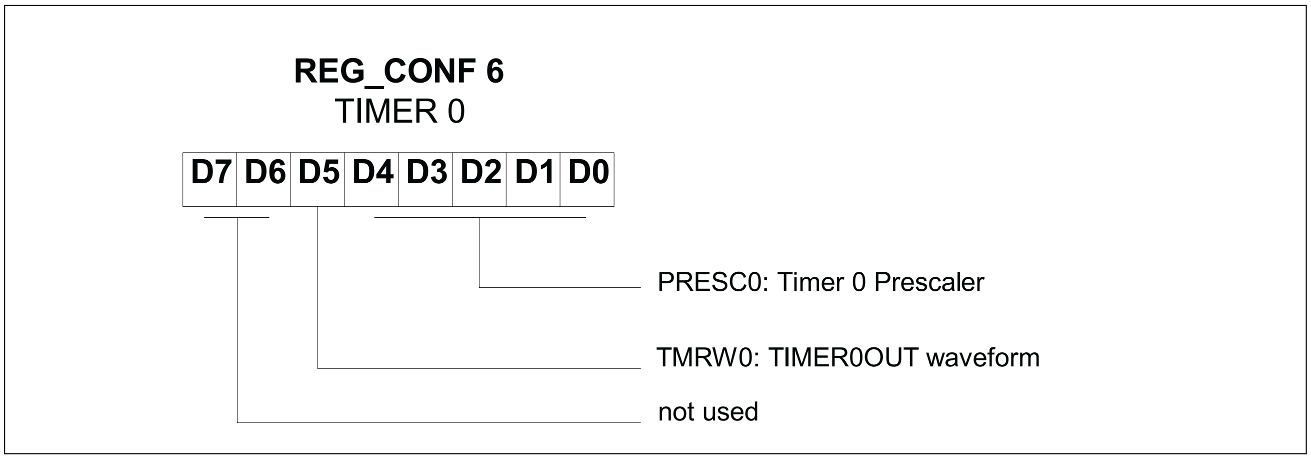


Table 9.3. Configuration Register 7 Description

Bit	Name	Value	Description
0	T0RST	00	TIMER0 RESET Internal
		01	TIMER0 RESET External
1		10	TIMER0 RESET External or Internal
		11	- not used
2	T0STR	00	TIMER0 START Internal
		01	TIMER0 START External
3		10	TIMER0 START External or Internal
		11	- not used
4	T0CLK	0	TIMER0 Clock Internal
		1	TIMER0 Clock External
5	T0MSK	0	TIMER 0 reset synchronization mask. TIMER 0 RESET enabled
		1	TIMER0 reset synchronization mask. TIMER0 RESET masked
6	T2MSK	0	TIMER2 reset synchronization mask. TIMER2 RESET enabled
		1	TIMER2 reset synchronization mask. TIMER2 RESET masked
7	T1MSK	0	TIMER1 reset synchronization mask. TIMER1 RESET enabled
		1	TIMER1 reset synchronization mask. TIMER1 RESET masked

Figure 9.7. Configuration Register 7

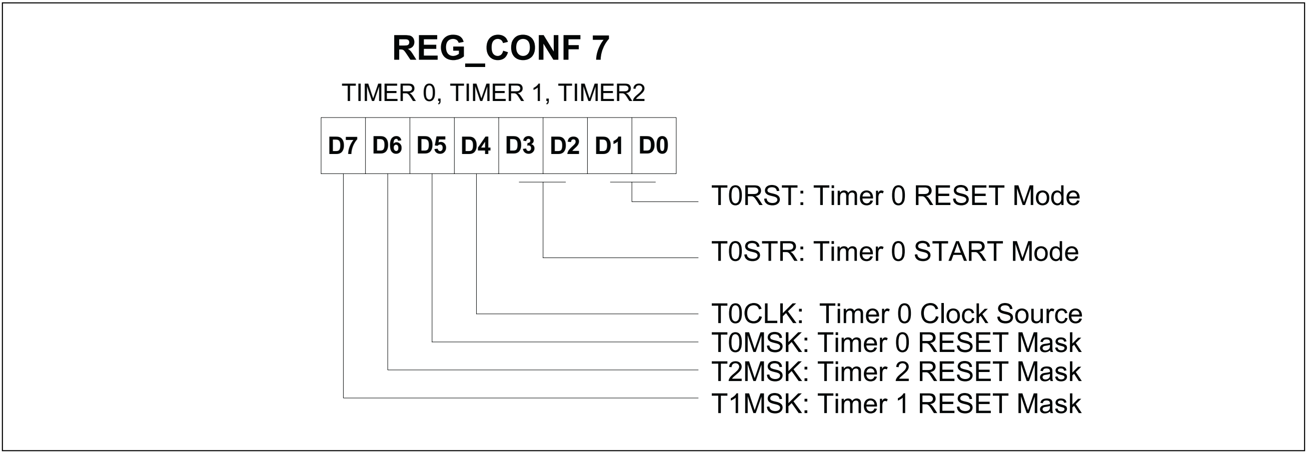

REG_CONF 7
TIMER 0, TIMER 1, TIMER2
D7
D6
D5
D4
D3
D2
D1
D0
T0RST: Timer 0 RESET Mode
T0STR: Timer 0 START Mode
T0CLK: Timer 0 Clock Source
T0MSK: Timer 0 RESET Mask
T2MSK: Timer 2 RESET Mask
T1MSK: Timer 1 RESET Mask

Table 9.4. Config. Register 8 Description

Bit	Name	Value	Description
0	TIRST1	0	TIMER 1 RESET enabled
		1	TIMER 1 RESET disabled
1	-	-	- not used
2	TISTR1	0	TIMER 1 STOP
		1	TIMER 1 START
3	-	-	- not used
4	INTE1	00	TIMER1 Interrupt on TIMER1OUT Falling Edge
		01	TIMER1 Interrupt on TIMER1OUT Rising Edge
5		10	TIMER1 Interrupt on Both Edges of TIMER1OUT
		11	- not used
6	INTSL1	0	TIMER1 Interrupt on Counter Stop
		1	TIMER1 Interrupt on TIMER1OUT
7	T1MODE	0	TIMER MODE
		1	PWM MODE

Figure 9.8. Configuration Register 8

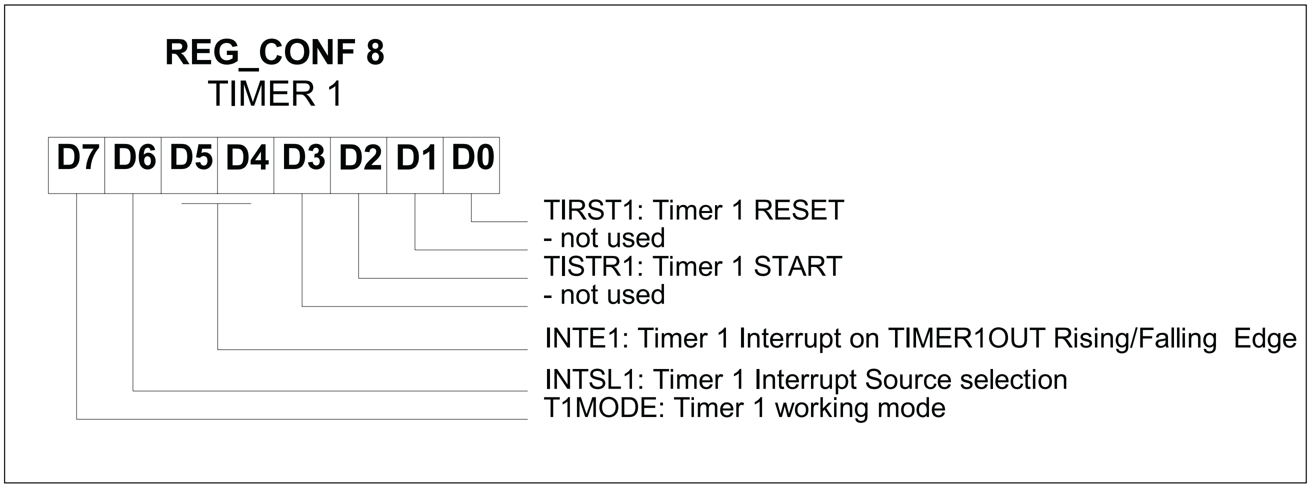

REG_CONF 8
TIMER 1
D7
D6
D5
D4
D3
D2
D1
D0
TIRST1: Timer 1 RESET
- not used
TISTR1: Timer 1 START
- not used
INTE1: Timer 1 Interrupt on TIMER1OUT Rising/Falling Edge
INTSL1: Timer 1 Interrupt Source selection
T1MODE: Timer 1 working mode

Table 9.5. Config.Register 9 Description

Bit	Name	Value	Description
0	PRESC1	00000	TIMER1 Clock = CLKM / 1
		00001	TIMER1 Clock = CLKM / 2
		00010	TIMER1 Clock = CLKM / 4
		00011	TIMER1 Clock = CLKM / 8
1		00100	TIMER1 Clock = CLKM / 16
		00101	TIMER1 Clock = CLKM / 32
		00110	TIMER1 Clock = CLKM / 64
2		00111	TIMER1 Clock = CLKM / 128
		01000	TIMER1 Clock = CLKM / 256
		01001	TIMER1 Clock = CLKM / 512
3		01010	TIMER1 Clock =CLKM / 1024
		01011	TIMER1 Clock =CLKM / 2048
		01100	TIMER1 Clock =CLKM / 4096
4		01101	TIMER1 Clock =CLKM / 8192
		01110	TIMER1 Clock =CLKM/16384
		01111	TIMER1 Clock=CLKM /32768
		10000	TIMER1 Clock=CLKM /65536
5	TMRW1	0	TIMER1OUT Waveform equal to pulse wave
		1	TIMER1OUT Waveform equal to square wave
6	-	-	- not used
7	-	-	- not used

Figure 9.9. Configuration Register 9

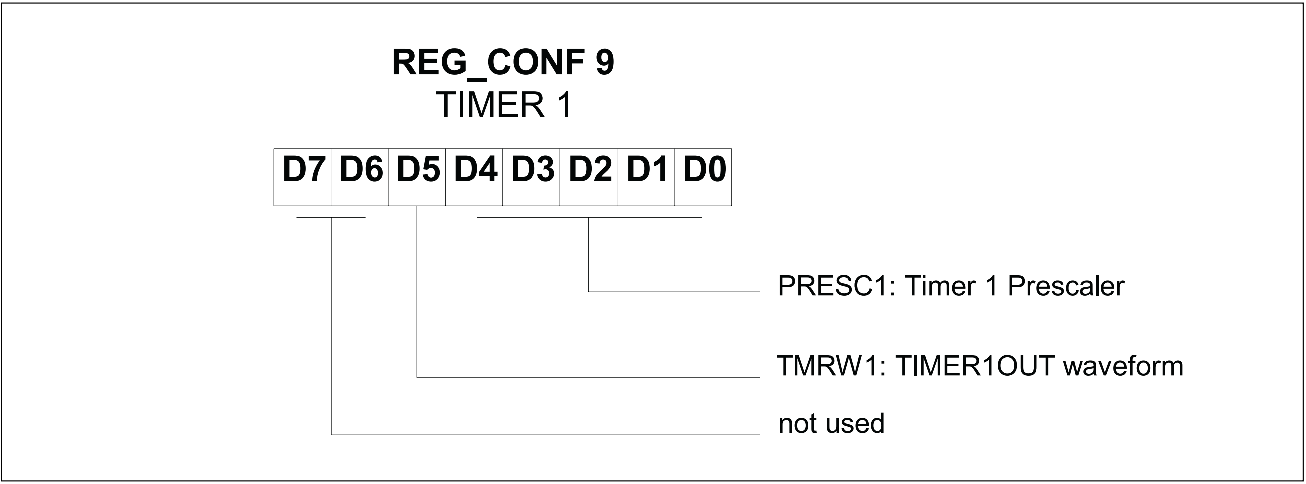


Table 9.6. Config. Register 10 Description

Bit	Name	Value	Description
0	TIRST2	0	TIMER 2 RESET enabled
		1	TIMER 2 RESET disabled
1	-	-	- not used
2	TISTR2	0	TIMER 2 STOP
		1	TIMER 2 START
3	-	-	- not used
4	INTE2	00	TIMER2 Interrupt on TIMER2OUT Falling Edge
		01	TIMER2 Interrupt on TIMER2OUT Rising Edge
5		10	TIMER2 Interrupt on Both Edges of TIMER2OUT
		11	- not used
6	INTSL2	0	TIMER2 Interrupt on Counter Stop
		1	TIMER2 Interrupt on TIMER2OUT
7	T2MODE	0	TIMER MODE
		1	PWM MODE

Figure 9.10. Configuration Register 10

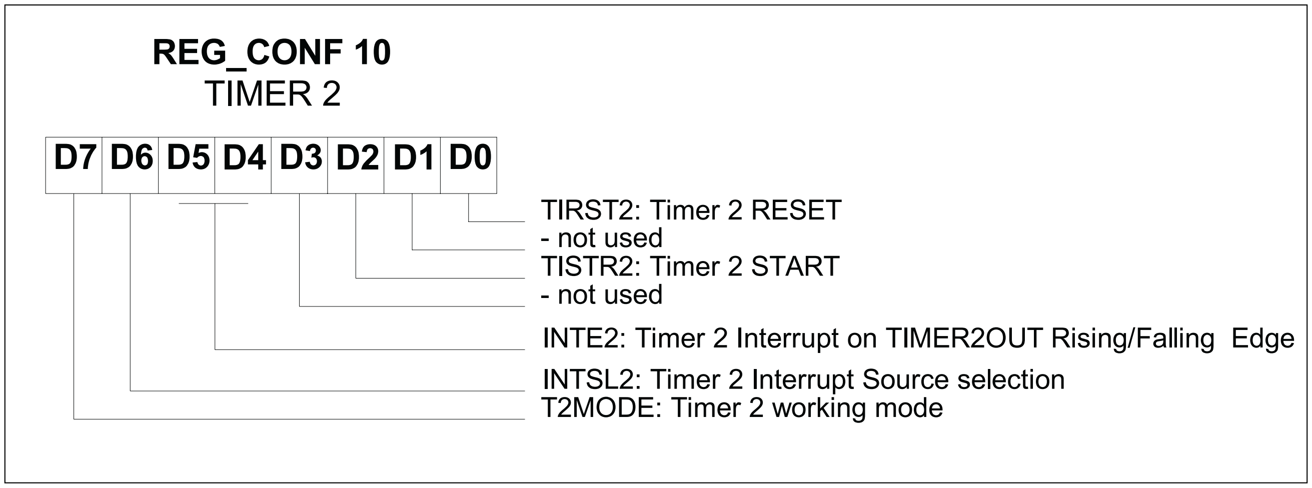

REG_CONF 10
TIMER 2
D7 D6 D5 D4 D3 D2 D1 D0
TIRST2: Timer 2 RESET
- not used
TISTR2: Timer 2 START
- not used
INTE2: Timer 2 Interrupt on TIMER2OUT Rising/Falling Edge
INTSL2: Timer 2 Interrupt Source selection
T2MODE: Timer 2 working mode

Table 9.7. Config. Register 11 Description

Bit	Name	Value	Description
0	PRESC2	00000	TIMER2 Clock = CLKM / 1
		00001	TIMER2 Clock = CLKM / 2
		00010	TIMER2 Clock = CLKM / 4
		00011	TIMER2 Clock = CLKM / 8
1		00100	TIMER2 Clock = CLKM / 16
		00101	TIMER2 Clock = CLKM / 32
		00110	TIMER2 Clock = CLKM / 64
2		00111	TIMER2 Clock = CLKM / 128
		01000	TIMER2 Clock = CLKM / 256
		01001	TIMER2 Clock = CLKM / 512
3		01010	TIMER2 Clock = CLKM /1024
		01011	TIMER2 Clock = CLKM/ 2048
		01100	TIMER2 Clock = CLKM/ 4096
4		01101	TIMER2 Clock = CLKM/ 8192
		01110	TIMER2 Clock= CLKM/16384
		01111	TIMER2 Clock =CLKM/32768
		10000	TIMER2 Clock =CLKM/65536
5	TMRW2	0	TIMER2OUT Waveform equal to pulse wave
		1	TIMER2OUT Waveform equal to square wave
6	-	-	- not used
7	-	-	- not used

Figure 9.11. Configuration Register 11

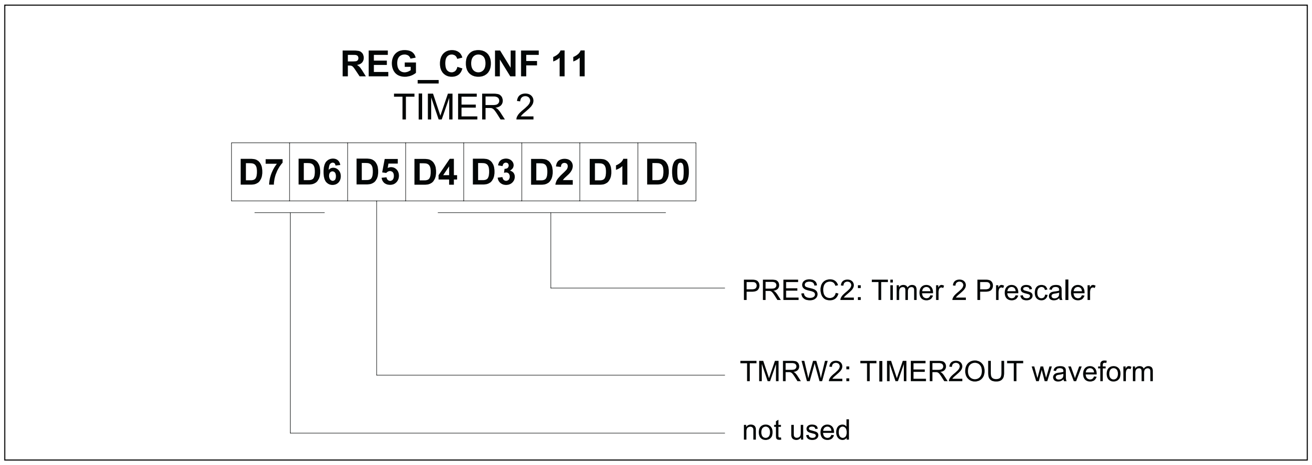


Table 9.8. Config. Register 12 Description

Bit	Name	Value	Description
0	PA1	0	Pin PA1/ $\overline{\text{T0OUT}}$ equal to PORT A Digital I/O
		1	Pin PA1/ $\overline{\text{T0OUT}}$ equal to $\overline{\text{T0OUT}}$
1	PA2	0	Pin PA2/ $\overline{\text{T1OUT}}$ equal to PORT A Digital I/O
		1	Pin PA2/ $\overline{\text{T1OUT}}$ equal to $\overline{\text{T1OUT}}$
2	PA3	0	Pin PA3/ $\overline{\text{T2OUT}}$ equal to PORT A Digital I/O
		1	Pin PA3/ $\overline{\text{T2OUT}}$ equal to $\overline{\text{T2OUT}}$
3	PASZ	0	PORT A bits = 7
		1	PORT A bits = 8
4	-	-	- not used
5	-	-	- not used
6	-	-	- not used
7	-	-	- not used

Figure 9.12. Configuration Register 12

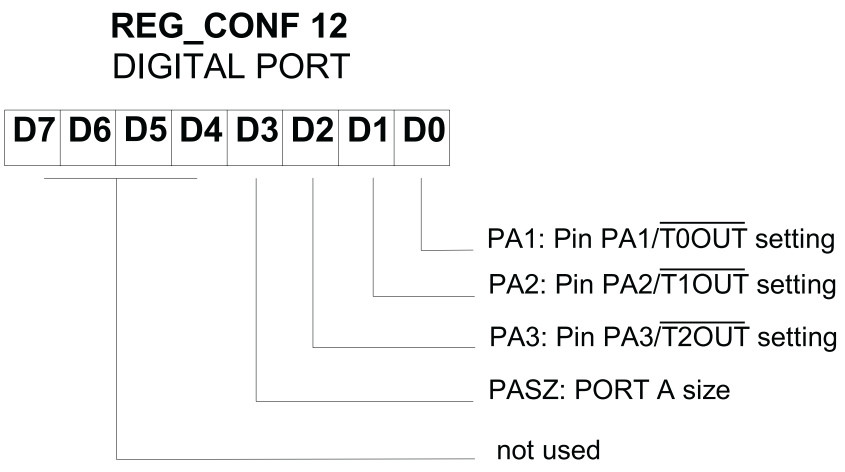

REG_CONF 12
DIGITAL PORT
D7 D6 D5 D4 D3 D2 D1 D0
PA1: Pin PA1/$\overline{\text{T0OUT}}$ setting
PA2: Pin PA2/$\overline{\text{T1OUT}}$ setting
PA3: Pin PA3/$\overline{\text{T2OUT}}$ setting
PASZ: PORT A size
not used

Table 9.9. Config. Register 16 Description

Bit	Name	Value	Description
0	PC1	1	Pin T0OUT/PC1 equal to PORT C Digital I/O
		0	Pin T0OUT/PC1 equal to T0OUT
1	PC2	1	Pin T1OUT/PC2 equal to PORT C Digital I/O
		0	Pin T1OUT/PC2 equal to T1OUT
2	PC3	1	Pin T2OUT/PC3 equal to PORT C Digital I/O
		0	Pin T2OUT/PC3 equal to T2OUT
3	PC4	1	Pin Tx/PC4 is configured as Port C Digital I/O
		0	Pin Tx/PC4 is configured as SCI output Tx
4-7	-	-	- not used

Figure 9.13. Configuration Register 16

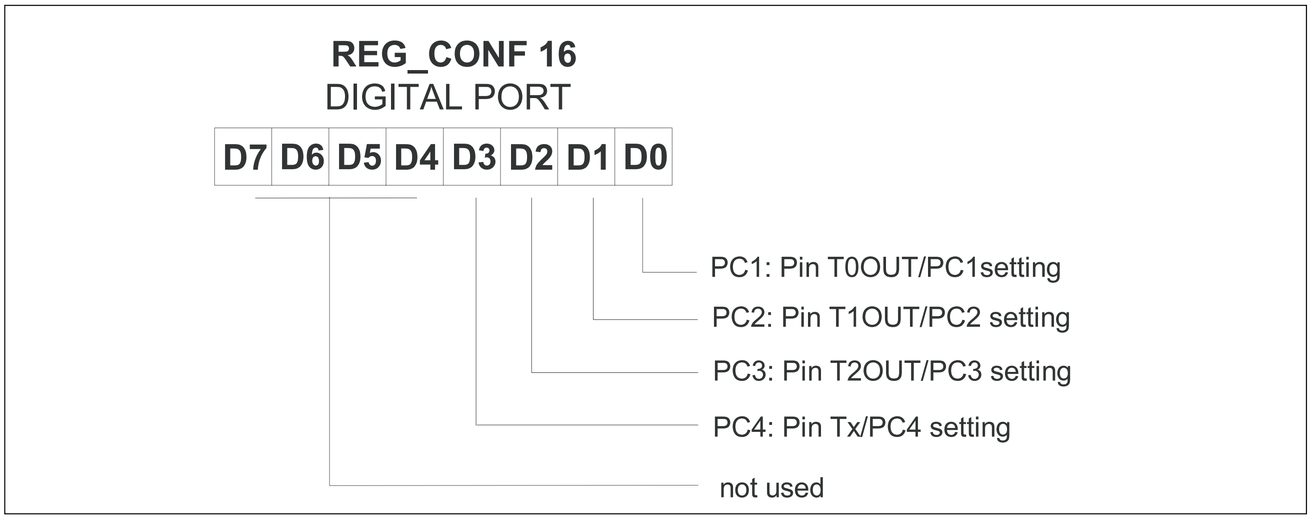

REG_CONF 16
DIGITAL PORT
D7 D6 D5 D4 D3 D2 D1 D0
PC1: Pin T0OUT/PC1setting
PC2: Pin T1OUT/PC2 setting
PC3: Pin T2OUT/PC3 setting
PC4: Pin Tx/PC4 setting
not used

Table 9.10 Input Registers 13. PWM_0_STATUS

Bit	Name	Value	Description
0	STR0ST	0	TIMER 0 is STOP
		1	TIMER 0 START
1	RST0ST	0	TIMER 0 is RESET
		1	TIMER 0 is NOT RESET
2	-	-	- not used
3	-	-	- not used
4	-	-	- not used
5	-	-	- not used
6	-	-	- not used
7	-	-	- not used

Table 9.12 Input Registers 17. PWM_2_STATUS

Bit	Name	Value	Description
0	STR2ST	0	TIMER 2 is STOP
		1	TIMER 2 is START
1	RST2ST	0	TIMER 2 is RESET
		1	TIMER 2 is NOT RESET
2	-	-	- not used
3	-	-	- not used
4	-	-	- not used
5	-	-	- not used
6	-	-	- not used
7	-	-	- not used

Table 9.11 Input Registers 15. PWM_1_STATUS

Bit	Name	Value	Description
0	STR1ST	0	TIMER 1 is STOP
		1	TIMER 1 is START
1	RST1ST	0	TIMER 1 is RESET
		1	TIMER 1 is NOT RESET
2	-	-	- not used
3	-	-	- not used
4	-	-	- not used
5	-	-	- not used
6	-	-	- not used
7	-	-	- not used

10 SERIAL COMMUNICATION INTERFACE

The Serial Communication Interface (SCI) integrated into the fuzzy processor ST52x430 provides a general purpose shift register peripheral, that allows to link several widely distributed MCUs, through their SCI subsystem. The SCI gives a serial interface providing communication with common baud rates, up to 38400, and flexible character format.

The SCI is a full-duplex UART-type asynchronous system with standard Non Return to Zero (NRZ) format for the transmitted/received bit. The length of the transmitted word is 10/11 bits (1 start bit, 8/9 data bits, 1 stop bit).

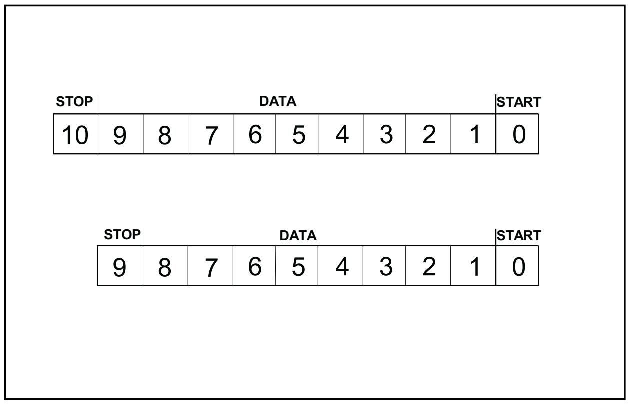
The SCI is composed of three modules: Receiver, Transmitter and Baud-Rate Generator and it is configured by means of Configuration Registers 19 and 20

WARNING: TO WORK CORRECTLY WITH SCI PERIPHERAL, MAINTAINING THE DESIRED BAUD RATE, IT IS NECESSARY TO USE ONLY 5, 10 OR 20 MHz SYSTEM CLOCK.

10.1 SCI Receiver block

The SCI Receiver block manages the synchronization of the serial data stream and stores the data characters. The SCI Receiver is mainly formed by two sub-systems: Recovery Buffer Block and SCDR_RX Block.

Figure 10.1 SCI transmitted word structures

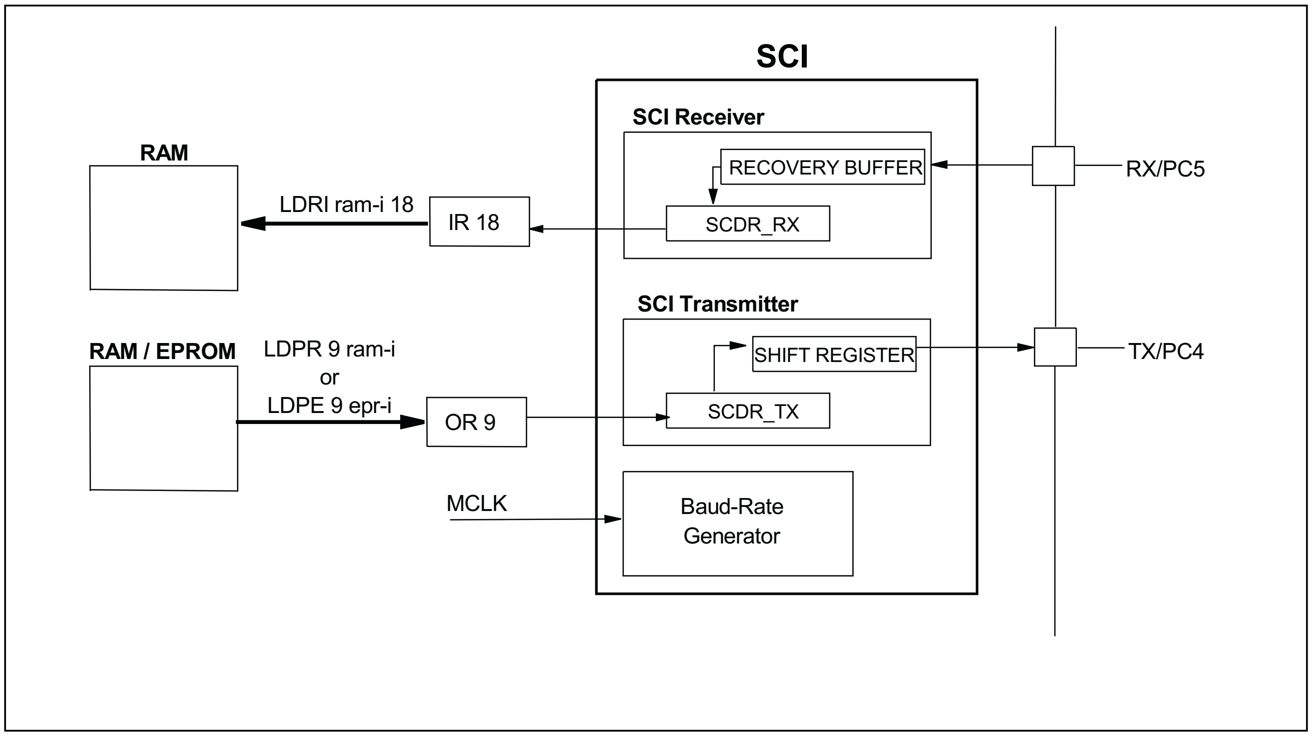


The RE configuration bit set to “1" (bit 1 of the Configuration Register 20) enables the SCI Receiver.

The SCI receives data coming from the RX/PC5 pin and drives the Recovery Buffer Block, that is a high-speed shift register operating at a clock frequency (CLOCK_RX) 16 times higher than the fixed baud rate (CLOCK_TX). This sampling rate, higher than the Baud Rate clock, allows to detect the START condition, the Noise error and the Frame error.

When the SCI Receiver is in IDLE status, it is waiting for the START condition, that is obtained with a logic level 0, consecutive to a logic level 1.

Figure 10.2 SCI Block Diagram



This condition is detected, if, with the fixed sampling time, a logic level 0 is sampled after three logic levels 1.

The recognition of the START bit forces the SCI Receiver Block to enter in a data acquisition sequence. The data acquisition sequence is configured by the Configuration Register 20 as follows.

The 2 bits, M, of the Configuration Register 20 allow to define the serial mode with the convention shown in table 10.1.

The bit, T8, in case of M=10 is used to set the parity check to perform, as indicated in the table 10.1.

The recognition of STOP condition allows to transfer the received data, from Recovery Buffer to SCDR_RX buffer, adding the eventual ninth data bit, according to the meaning shown in the previous table 10.1. After this operation, RXF flag of SCI Status Input Register 19 (fig.10.3) is set to logic level 1. The Control Unit reads the data from SCDR_RX buffer (in read-only mode) with LDRI instruction, addressing the Input Register 18, and provides a reset at logic level 0 to RXF flag.

If a data of Recovery Buffer is ready to be transferred into SCDR_RX buffer, but the previous one was not yet read by the Core, an OVERRUN Error takes place: the status flag OVERR indicates the error condition. In this case the information stored in SCDR_RX buffer is not altered, but the one that has caused the OVERRUN error can be overwritten by a new data coming from the serial data line.

Recovery Buffer Block

This block is structured as a synchronized finite state machine on the CLOCK_RX signal.

When the Recovery Buffer Block is in IDLE state it waits for the reception of the correct 1 and 0 sequence representing the START.

The recognition takes place by sampling the input RX/PC5 at CLOCK_RX frequency, that has a frequency 16 times higher than CLOCK_TX. For this reason, while the external transmitter sends a single bit, the Recovery Buffer Block samples 16 states (from SAMPLE1 to SAMPLE16).

The analysis of RX/PC5 input signal is carried out looking three samples for each bit received.

If these three samples are not equal, then the noise error flag, NSERR, of Input Register 19 is set to 1 and the received data value will be the one assumed by the majority of the samples.

By means of the procedure described above, to avoid SCI becomes IDLE, because of a limited

Table 10.1 Configuration Register 20 Setting

Bit	Name	Value	Description
0	TE	0	Transmission DISABLED
		1	Transmission ENABLED
1	RE	0	Receiver DISABLED
		1	Receiver ENABLED
2	M	00	8, No Parity, 1 bit stop
		01	8, No Parity, 2 bit stop
3		10	8, Parity, 1 bit stop
		11	9, No Parity, 1 bit stop
4	T8	0	Parity Odd, if Parity is selected (M = 10); otherwise 9th Data bit
		1	Parity Even, if Parity is selected (M = 10); otherwise 9th Data bit
5	BRSL	000	600 baud
		001	1200 baud
		010	2400baud
6		011	4800 baud
		100	9600 baud
7		101	19200 baud
		110	38400 baud
		111	Not Used

noise due to an erroneous sampling, the transmission is recognized as correct and the noise flag error is set.

At the end of the cycle relative to the reception of a bit, Recovery Buffer Block will repeat the same steps 9 times: one step for each received bit, plus one for the stop acquisition (10 times in case of 9-bit data, double stop or parity check).

At the end of data reception, Recovery Buffer Block, will supply information on eventual frame errors by setting to 1 FRERR flag bit of Input Register 19.

A frame error can occur if the parity check has not been successfully achieved or if STOP bit has not been detected.

If Recovery Buffer Block receives 10 consecutive bits at logic level 0, a break error occurs, and interrupt routine request starts.

SCDR_RX Block

It is a finite state machine synchronized with the clock master signal, CKM.

The SCDR_RX block waits the signal of complete reception, from the Recovery Buffer, to load the word received. Moreover, the SCDR_RX block loads the values of FRERR and NSERR flag bits (Input Register 19), and sets the RXF flag to 1.

Using LDRI instruction the data are transferred to RAM and RXF flag is reset to 0, to indicate SCDR_RX block is empty.

If a new data arrives before the previous one has been transferred to Register File, an overrun error occurs and OVERR flag, of Input Register 19, is set to 1.

10.2 SCI Transmitter Block

The SCI Transmitter Block consists of the following blocks: SCDR_TX and SHIFT REGISTER, synchronized, respectively, with the clock master signal (CKM) and the CLOCK_TX.

The whole block receives through Configuration Register 20 (M bits) the settings for the following transmission modes (see table 10.1):

- 8-bit word and a single stop signal
- 8-bit word plus a parity bit and a single stop signal
- 8-bit word plus a double stop signal
- 9-bit word

In case of 9 bit frame transmission, the most significative bit arrives through T8 of the Configuration Register 20.

Table 10.2 Configuration Register 19 Setting

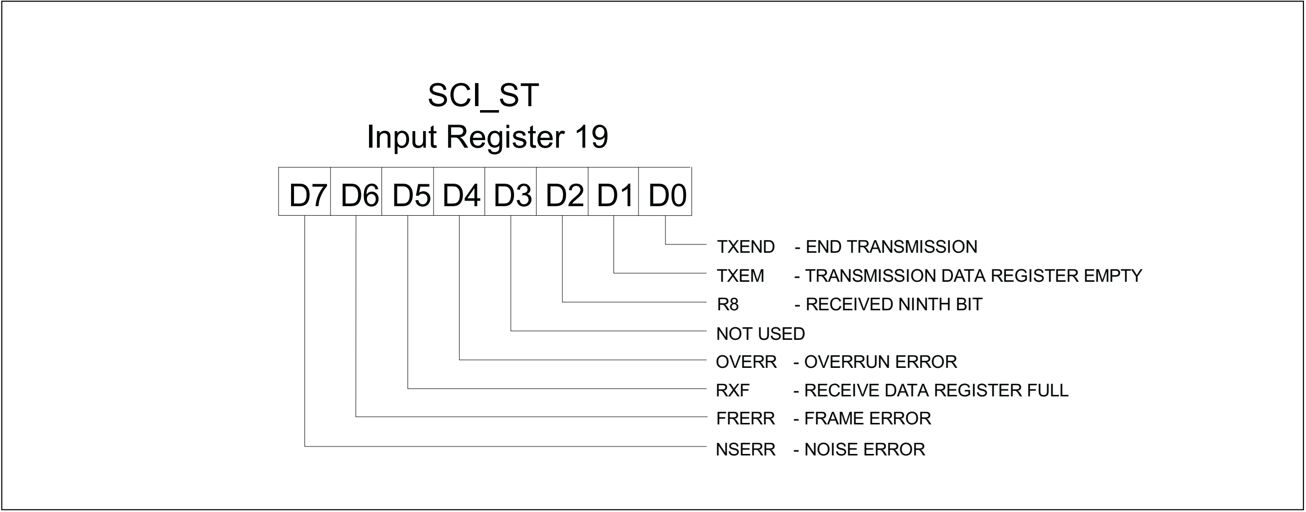
Bit	Name	Value	Description
0		-	Not used
1	ECKF	00	5 MHz
		01	10 MHz
2		10	20 MHz
		11	20 MHz
3	TXC	0	SCI End Transmission Interrupt Disabled
		1	SCI End Transmission Interrupt Enabled
4	TDRE	0	SCI Transmission Data Register Empty Interrupt Disabled
		1	SCI Transmission Data Register Empty Interrupt Enabled
5	BRK	0	SCI Break Error Interrupt Disabled
		1	SCI Break Error Interrupt Enabled
6	OVR	0	SCI Overrun Error Interrupt Disabled
		1	SCI Overrun Error Interrupt Enabled
7	RDRF	0	SCI Received Data Register Full Interrupt Disabled
		1	SCI Received Data Register Full Interrupt Enabled

In an 8-bit transmission, instead, T8 is used to configure the SCI, according to information contained in M (see table 10.1): in particular to choose the polarity control (even or odds) to implement the parity check.

After a RESET signal, RST, the SCDR_TX block is in IDLE state until it receives enabling signal, TE=1, of Configuration Register 20.

The data is loaded on the peripheral register (OR 9) by using the instruction LPPR or LDPE. If TE=1, the data, to be transmitted, are transferred from Output Register 9 (OR 9) to SCDR_TX block and the flag of Input Register 19, TXEM, is reset to 0, to indicate SCDR_TX block is full.

Figure 10.3. SCI Status Input Register



If the core supplies a new data, this could not be loaded in the SCDR_TX block until the current data has not been unloaded on the Shift Register block. This means that only when TXEM is 1, it is possible to load data in the SCDR_TX Block.

When the SHIFT REGISTER Block loads the data to be transmitted on an internal buffer, TXEND is reset to 0 to indicate the beginning of a new transmission. At the end of transmission TXEND is set to 1, allowing to load in the SHIFT REGISTER a new data coming from SCDR_TX.

It is important to underline that TXEND = 1 does not mean SCDR_TX is ready to receive a new data. For this reason it is better to utilize the TXEM signal to synchronize the LDPR instruction to the SCI TRANSMITTER block

If ST52x430 core resets TE to 0, the transmission is interrupted, but the SCI Transmitter block completes the transmission in progress before to reset.

10.3 Baud Rate Generator Block

The Baud Rate Generator Block performs the division of the clock master signal (CKM), in a set of synchronism frequencies for the serial bit reception/transmission on the external line.

Table 10.1. shows the set of frequencies selected by means of BRSL (Configuration Register 20).

Reception frequency (CLOCK_RX) is 16 times higher than transmission frequency (CLOCK_TX) .

The following example shows a simple way to use the SCI to receive and transmit data:

```
LDRC 1 251
LDCR 20 1    These instruction loads the value 251
               on the Configuration Register 20
               fixing the Baud Rate=9600, 8 bit
               data, TE=1, RE=1; Parity; 1 stop
               bit.
LDRC 1 252
LDCR 19 1    SCI Interrupts enabled, clock
               frequency 20 MHz
LDRC 1 170
LDPR 9 1     Send data to transmission buffer
WAITI
LDRI 6 19    Save the SCI status register on the
             RAM
LDRI 1 18    Save the received data on a RAM
             register
```

11 ELECTRICAL CHARACTERISTICS

11.1 Parameter Conditions

Unless otherwise specified, all voltages are referred to V_{SS}

11.1.1 Minimum and Maximum values

Unless otherwise specified the minimum and maximum values are guaranteed in the worst conditions of ambient temperature, supply voltage and frequencies by test in production on 100% of the devices with an ambient temperature at T_A=25°C and T_A=$T_A$max (given by the selected temperature range).

Data based on characterization results, design simulation and/or technology characteristics are indicated in the table footnotes and are not tested in production. Based on characterization, the minimum and maximum values refer to sample tests and represent the mean value plus or minus three times the standard deviation (mean ±3Σ).

11.1.2 Typical values

Unless otherwise specified, typical data are based on T_A=25°C, V_{DD}=5V (for the $4.5 \leq V_{DD} \leq 5.5$V voltage range). They are given only as design guideline and are not tested.

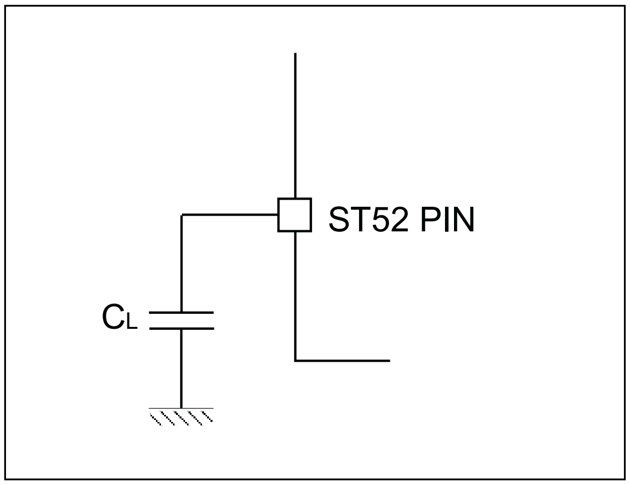
11.1.3 Typical curves

Unless otherwise specified, all typical curves are given only as design guidelines and are not tested.

11.1.4 Loading capacitor

The loading condition used for pin parameter measurement is shown in Figure 11.1.

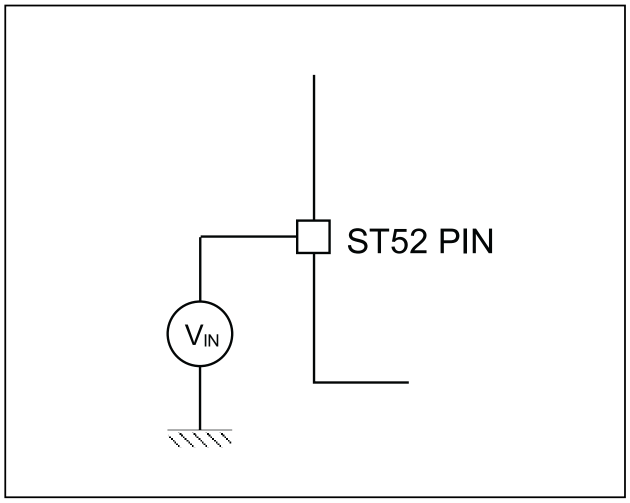
Figure 11.1. Pin loading conditions



11.1.5 Pin input voltage

The input voltage measurement on a pin of the device is described in Figure 11.2

Figure 11.2 Pin input Voltage



11.2 Absolute Maximum Ratings

Stresses above those listed as "absolute maximum ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device under these conditions is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

Table 11.1 Voltage Characteristics

Symbol	Ratings	Maximum Value	Unit
V_{DD}-V_{SS}	Supply voltage	6.5	V
V_{DDA}-V_{SSA}	Analog reference voltage($V_{DD} \geq V_{DDA}$)	6.5	
$\|\Delta V_{DDA}\|$ and $\|\Delta V_{SSA}\|$	Variation between different digital power pins	50	mV
$\|V_{SSA}$-$V_{SSX}\|$	Variation between digital and analog ground pins	50	
V_{IN}	Input voltage on Vpp	V_{SS}-0.3 to 13	V
	Input voltage on any other pin 1) & 2)	V_{SS}-0.3 to V_{DD}+0.3	
V_{DESD}	Electro-static discharge voltage	2000	

Table 11.2 Current Characteristics

Symbol	Ratings	Maximum Value	Unit
I_{VDD}	Total current in V_{DD} power lines (source)3)	100	mA
I_{VSS}	Total current in V_{SS} ground lines (sink)3)	100	
I_{IO}	Output current sunk by any standard I/O and control pin	25	
	Output current source by any I/Os and control pin	-25	
$I_{INJ(PIN)}$	Injected current on V_{PP} pin	±5	
	Injected current on RESET pin	±5	
	Injected current on OSCin and OSCout pins	±5	
	Injected current on any other pin 4)	±5	
$\Sigma I_{INJ(PIN)}$	Total Injected current (sum of all I/O and control pins) 4)	±20	

Table 11.3 Thermal Characteristics

Symbol	Ratings	Maximum Value	Unit
T_{STG}	Storage temperature range	-65 to +150	°C
T_J	Maximum junction temperature	150	°C

Notes:

1. Directly connecting the RESET and I/O pins to V_{DD} or V_{SS} could damage the device if an unintentional internal reset is generated or an unexpected change of I/O configuration occurs (for example, due to a corrupted program counter). To guarantee safe operation, this connection has to be done through a pull-up or pull-down resistor (typical: 4.7kΩ for RESET, 10kΩ for I/Os). Unused I/O pins must be tied in the same way to V_{DD} or V_{SS} according to their reset configuration.

2. When the current limitation is not possible, the V_{IN} absolute maximum rating must be respected, otherwise refer to $I_{INJ(PIN)}$ specification. A positive injection is induced by $V_{IN}>V_{DD}$ while a negative injection is induced by $V_{IN}<V_{SS}$.

3. All power (V_{DD})and ground (V_{SS}) lines must always be connected to the external supply.

4. When several inputs are submitted to a current injection, the maximum $\Sigma I_{INJ(PIN)}$ is the absolute sum of the positive and negative injected currents (instantaneous values)

11.3 Recommended Operating Condition

Operating condition: $V_{DD}=5V\pm10\%$; $T_A=0 \div 125$ °C, unless otherwise specified

Table 11.4 Recommended Operating Conditions

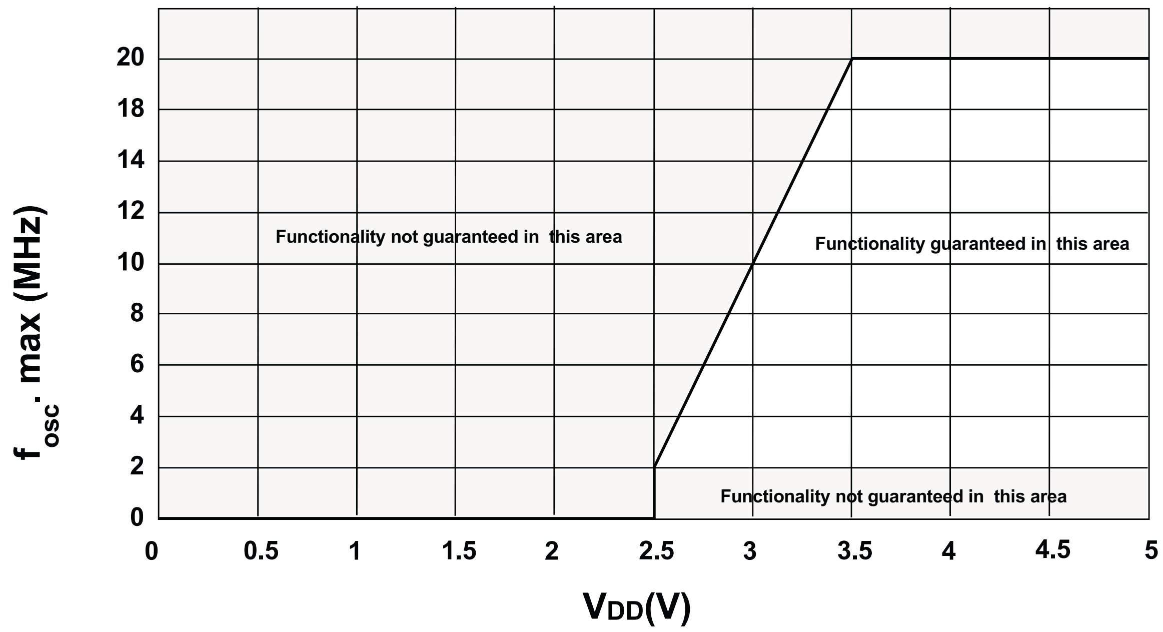
Symbol	Parameter	Test Condition	Min.	Typ.	Max	Unit
V_{DD} 2)	Operating Supply Voltage	Refer to the Fig. 11.3	3		5.5	V
V_{PP}	Programming Voltage		11.4	12	12.6	
V_O	Output Voltage		V_{SS}		V_{DD}	
$V_{DDA,}$	Analog Supply Voltage		V_{DD}-0.3	V_{DD}	V_{DD}+0.3	
V_{SSA}	Analog Ground		V_{SS}-0.3	V_{SS}	V_{SS}+0.3	
f_{OSC} 1), 2)	Oscillator Frequency		2		20	MHz

Notes:

1. To use correctly the SCI maintaining the programmed baud rates it is necessary to set fosc to 5, 10 or 20 Mhz.

2. It is possible to use a lower V_{DD} decreasing f_{osc} (see figure 11.3). The data shown in the following figure are preliminary and not guaranteed.

Figure 11.3 f_{OSC} Maximum Operating Frequency versus V_{DD} supply



11.4 Supply Current Characteristics

The supply current is mainly a function of the operating voltage and frequency. Other factors such as I/O pin loading and switching rate, oscillator type, internal code execution pattern and temperature also have an impact on the current consumption.

The test condition in RUN mode for all the I_{DD} measurements are:

OSCin = external square wave, from rail to rail;

OSCout = floating;

All I/O pins tristated pulled to V_{DD}

Table 11.5 Supply Current in RUN and WAIT Mode

Symbol	Parameter	Conditions		Typ	Max[3)]	Unit
I_{DD}	Supply current in RUN mode [1)]	V_{DD}=5V±5%	f_{osc}=2 Mhz	4	4.3	mA
			f_{osc}=4 Mhz	7	7.5	
			f_{osc}=5 Mhz,	7.7	8.0	
			f_{osc}=10 MHz	13.7	14.0	
			f_{osc}=20 MHz	26.4	27.0	
	Supply current in WAIT mode[2)]		f_{osc}=2 MHz	1.5	1.6	
			f_{osc}=4 MHz	3.0	3.3	
			f_{osc}=5 MHz	3.7	4.0	
			f_{osc}=10 MHz	6.9	7.2	
			f_{osc}=20 MHz	10.6	14.5	

Notes:

1. CPU running with memory access, all I/O pins in input mode with a static value at V_{DD} or V_{SS} (no load), all peripherals switched off; clock input (OSCin driven by external square wave).

2. CPU in WAIT mode with all I/O pins in input mode with a static value at V_{DD} or V_{SS} (no load), all peripherals switched off; clock input (OSCin driven by external square wave).

3. Data based on characterization results, tested in production at V_{DDmax} and f_{oscmax}.

Figure 11.4. Typical I_{DD} in RUN vs f_{OSC}

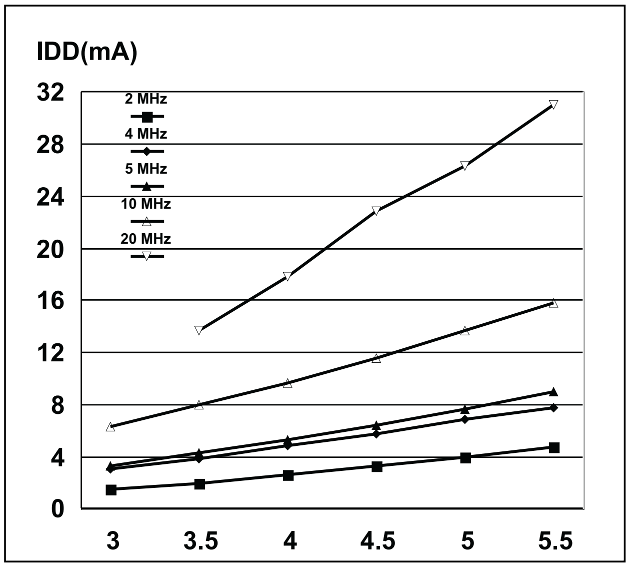
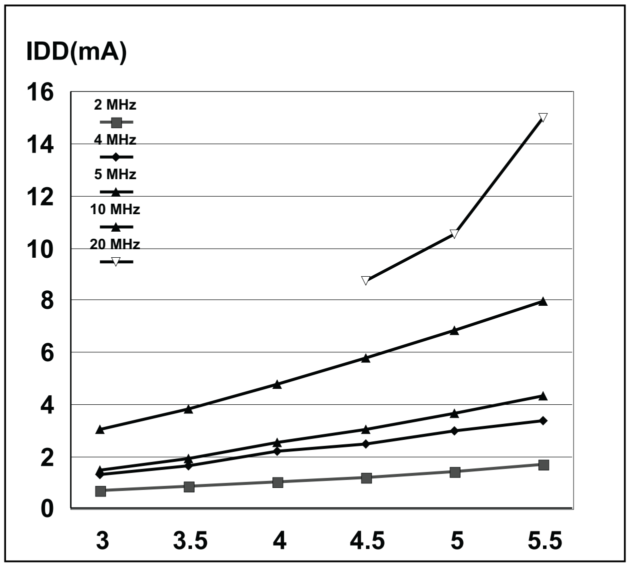


Figure 11.5. Typical I_{DD} in WAIT vs f_{OSC}



Supply Current Characteristics (Cont'd)

Tabel 11.6 Supply Current in HALT Mode

Symbol	Parameter	Conditions	Typ[1]	Max	Unit
I_{DD}	Supply current in HALT mode [2]	$3.0\text{ V} \leq V_{DD} \leq 5.5\text{ V}$	1	10	μA

Notes:

1. Typical data are based on T_A=25°C.
2. All I/O pins in input mode with a static value at V_{DD} or V_{SS} (no load)

Table 11.7 On-Chip Peripheral

Symbol	Parameter	Conditions	Typ[3]	Max[4]	Unit
I_{DDA}	ADC Supply current when converting	fosc=20MHz, V_{DDA} = 5 ±5% V, V_{SSA}=V_{SS}	1	2	mA

Notes:

3. Typical data are based on T_A=25°C, V_{DDA}=5 V.
4. Data based on characterization results, not tested in production

11.5 Clock and Timing Characteristics

Operating Conditions: VDD=5V±5%, TA=0 C t0 125 C, unless otherwise specified

Table 11.8 General Timing Parameters

Symbol	Parameters	Test Condition	Min	Typ.	Max	Unit
f_{osc}	Oscillator Frequency		1		20	MHz
t_{CLH}	Clock High		25		500	nS
t_{CLL}	Clock Low		25		500	
t_{SET}	Setup	See Fig. 11.6		5		
t_{HLD}	Hold	See Fig. 11.6		5		
t_{WRESET}	Minimum Reset Pulse Width	f_{osc}=20MHz	100			
t_{WINT}	Minimum External Interrupt Pulse Width	f_{osc}=20MHz	100			
t_{IR}	Input Rise Time	See Fig. 11.7			15	
t_{IF}	Input Fall Time	See Fig. 11.7			15	
t_{OR}	Output Rise Time	C_{LOAD}=10pF See Fig. 11.6		10		
t_{OF}	Output Fall	C_{LOAD}=10pF See Fig. 11.6		10		

Figure 11.6 Data Input Timing

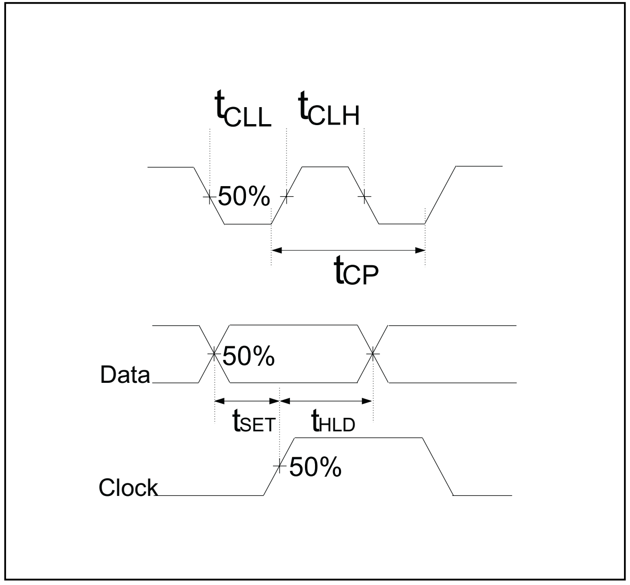
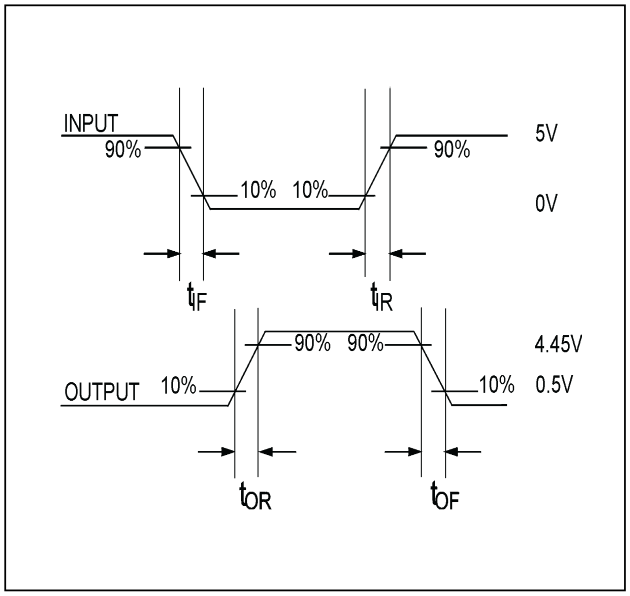


Figure 11.7 I/O Rise and Fall Timing



11.6 Memory Characteristics

Subject to general operating condition for V_{DD}, f_{osc} and T_A unless otherwise specified.

Table 11.8 RAM and HARDWARE Registers

Symbol	Parameter	Conditions	Min.	Typ.	Max	Unit
V_{RM}	Data retention mode 1)	HALT mode (or RESET)	1.6			V

Table 11.9 EPROM Program Memory

Symbol	Parameter	Conditions	Min.	Typ.	Max	Unit
W_{ERASE}	UV lamp	Lamp wavelength 2537 Å		15		Watt.sec /cm^2
t_{ERASE}	Erase Time 2)	UV lamp is placed 1 inch from the device window without any interposed filters	15		20	min
t_{ret}	Data Retention	T_A=+55°C	20			years

Notes:

1. Minimum V_{DD} supply voltage without losing data stored into RAM (in HALT mode or under RESET) or into hardware registers (only in HALT mode). Guaranteed by construction, not tested in production.

2. Data given only as guidelines

11.7 ESD Pin Protection Strategy

To protect an integrated circuit against Electro-Static Discharge the stress must be controlled to prevent degradation or destruction of the circuit elements. The stress generally affects the circuit elements which are connected to the pads but can also affect the internal devices when the supply pads receive the stress. The elements to be protected must not receive excessive current, voltage or heating within their structure.

An ESD network combines the different input and output protections. This network works, by allowing safe discharge paths for the pins subjected to ESD stress. Two critical ESD stress cases are presented in the Figures 11.8 and Figure for standard pins.

11.7.1 Standard Pin Protection

To Protect the output structure the following elements are added:

- A diode to V_{DD} (3a) and a diode from V_{SS} (3b)

- A protection device between V_{DD} and V_{SS} (4)

To protect the input structure the following elements are added:

- A resistor in series with the pad (1)

- A diode to V_{DD} (2a) and a diode from V_{SS} (2b)

- A protection device between V_{DD} and V_{SS} (4)

Figure 11.8. Safe discharge path for pins subjected to ESD stress

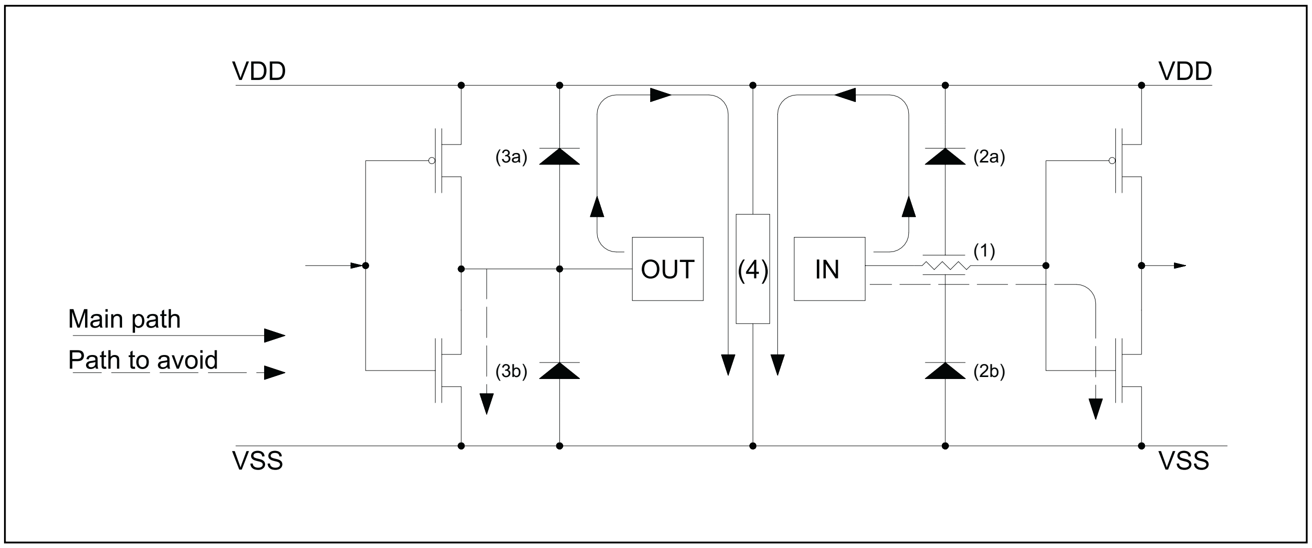
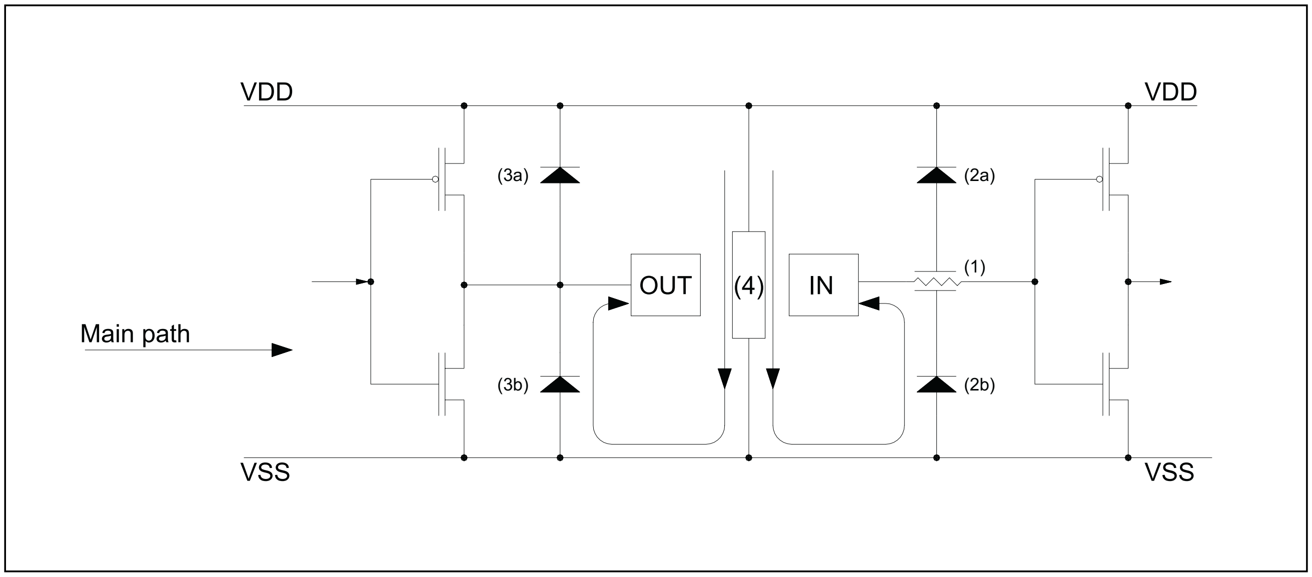


Figure 11.9 Negative Stress on a Standard Pad vs. V_{DD}

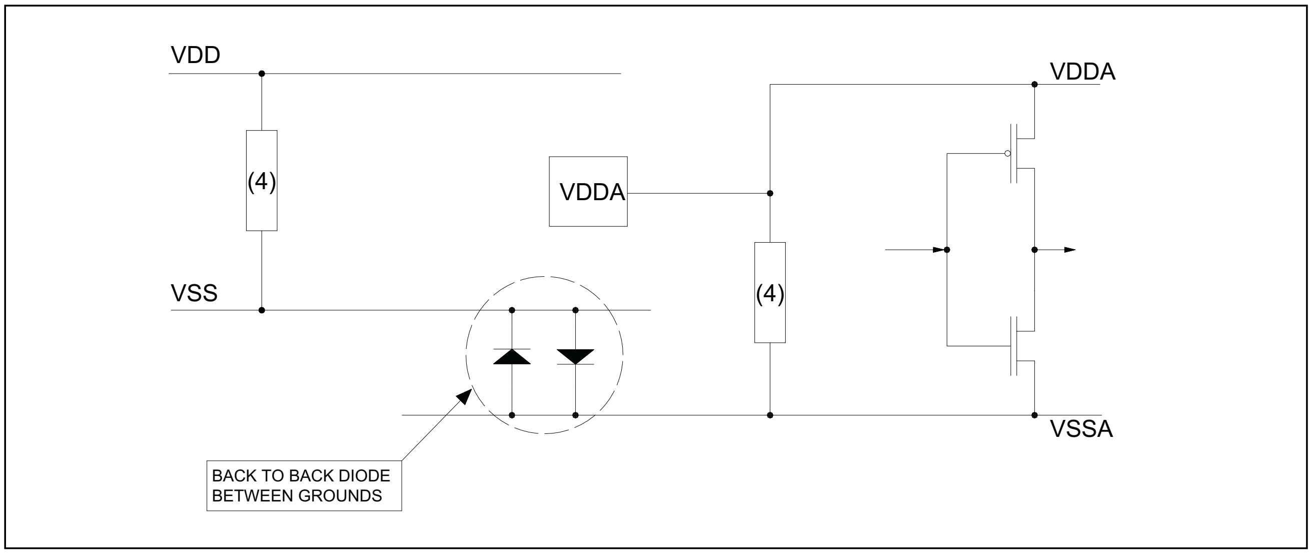


ESD Pin Protection Strategy (Cont'd)

11.7.2 Multisupply Configuration

When several types of ground (V_{SS}, V_{SSA},...) and power supply (V_{DD}, V_{DDA},...) are available for any reason (better noise immunity...), the structure shown in Figure 11.10 is implemented to protect the device against ESD.

Figure 11.9 ESD Protection for Multisupply Configuration



11.8 Port Pin Characteristics

11.8.1 General characteristics

Subject to general operating condition for V_{DD}, $f_{osc,}$ and T_A unless otherwise specified.

Symbol	Parameter	Condition	Min	Typ[1)]	Max	Unit
V_{IL}	CMOS type low level input voltage. Port B pins. (See Fig 11.13)				1.5	V
	TTL type Schmitt trigger low level input voltage. Port A and Port C pins. (See Fig. 11.12)				0.8	
V_{IH}	CMOS type high level input voltage. Port B pins. (See Fig 11.13)		3.3			
	TTL type Schmitt trigger high level input voltage. Port A and Port C pins. (See Fig. 11.12)		2.2			
V_{hys}	Schmitt trigger voltage hysteresis [2)]			1		
I_L	Input leakage current	$V_{SS} \leq V_{IN} \leq V_{DD}$			±1	μA
I_S	Static current consumption[3)]	Floating input mode			200	

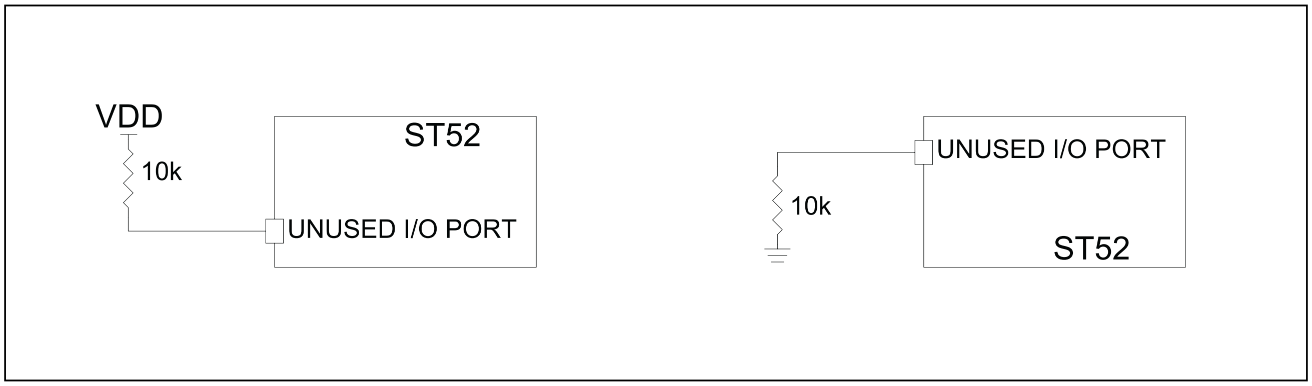
Notes:

1. Unless otherwise specified, typical data are based on T_A=25 °C and V_{DD}=5 V

2. Hysteresis voltage between Schmitt trigger switching level. Based on characterization results, not tested in production.

3. Configuration not recommended, all unused pins must be kept at a fixed voltage: using the output mode of the I/O for example or an external pull-up or pull-down resistor (see Figure 11.10). Data based on design simulation and/or technology characteristics, not tested in production.

Figure 11.10 Unused pins recommended configuration



Port Pin Characteristics (Cond't)

Subject to general operating condition for V_{DD}, $f_{osc,}$ and T_A unless otherwise specified.

Table 11.11 Output Driving Current

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{OL}1)	Output low level voltage for standard I/O pin when 8 pins are sunk at same time.	V_{DD}=5V, I_{IO}=+8mA			V_{SS}+0.4	V
V_{OH}2)	Output high level voltage for standard I/O pin when 8 pins are sourced at same time.	V_{DD}=5V, I_{IO}=- 8mA	V_{DD}-0.5			

Notes:

1. The I_{IO} current sunk must always respect the absolute maximum rating specified in Section 11.2 and the sum of I_{IO} (I/O ports and control pins) must not exceed I_{VSS}

2. The I_{IO} current sourced must always respect the absolute maximum rating specified in Section 11.2 and the sum of I_{IO} (I/O ports and control pins) must not exceed $I_{VDD.}$

Figure 11.12 TTL-Level input Schmitt Trigger

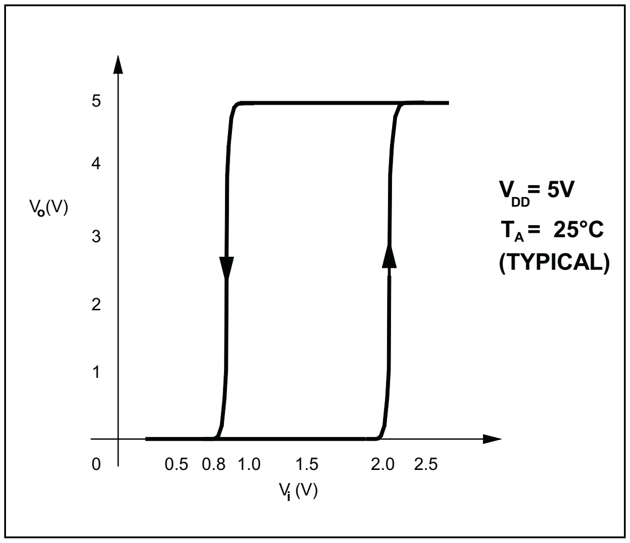
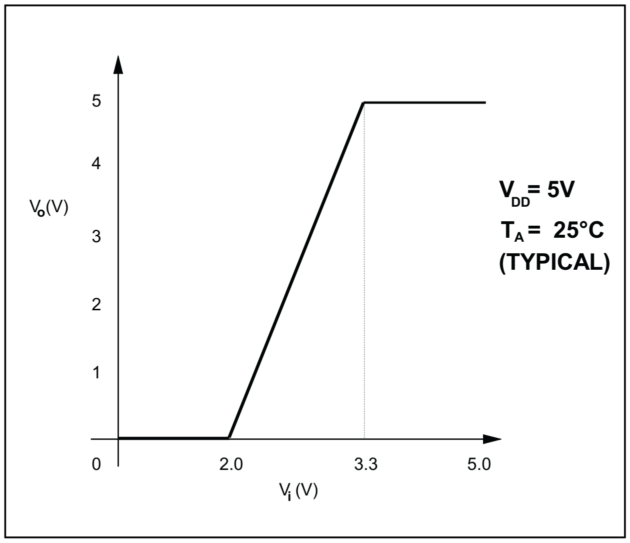


Figure 11.13 Port B pins CMOS-level input



Port Pin Characteristics (Cond't)

Subject to general operating condition for V_{DD}, $f_{osc,}$ and T_A unless otherwise specified.

Table 11.11 AC Electrical Characteristics

Symbol	Parameter	Test Condition	Min	Typ	Max	Unit
R_S	Input protection resistor	All input Pins		1		kΩ
C_S	Pin Capacitance	All inputs Pins		5		pF

Figure 11.12 Port A and Port C Pin Equivalent Circuit

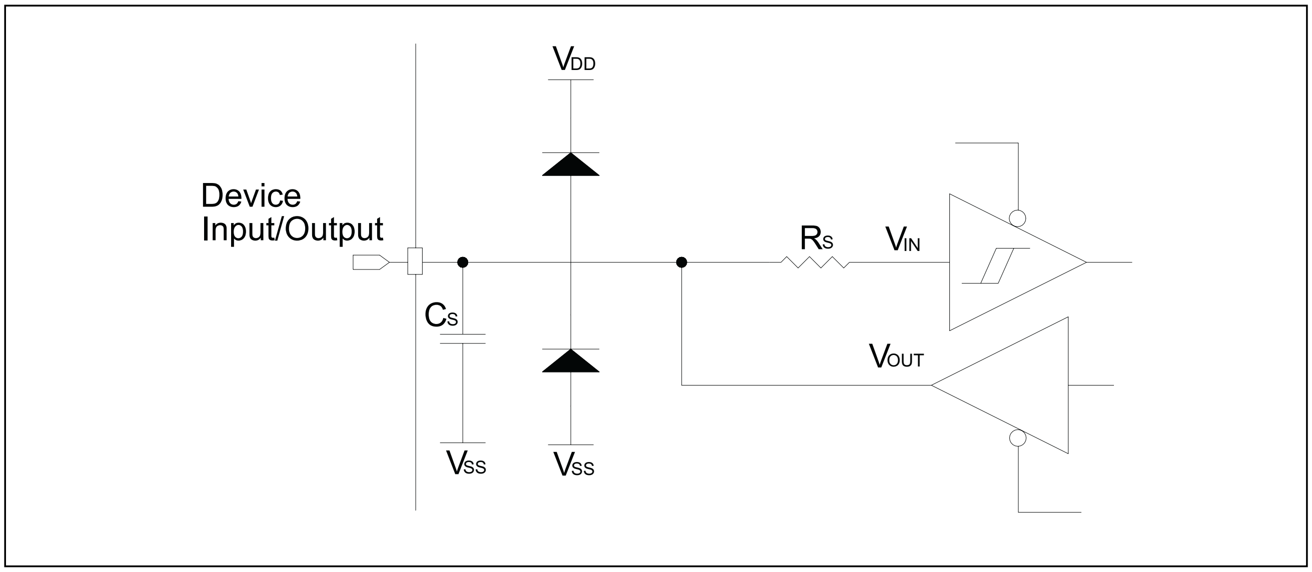
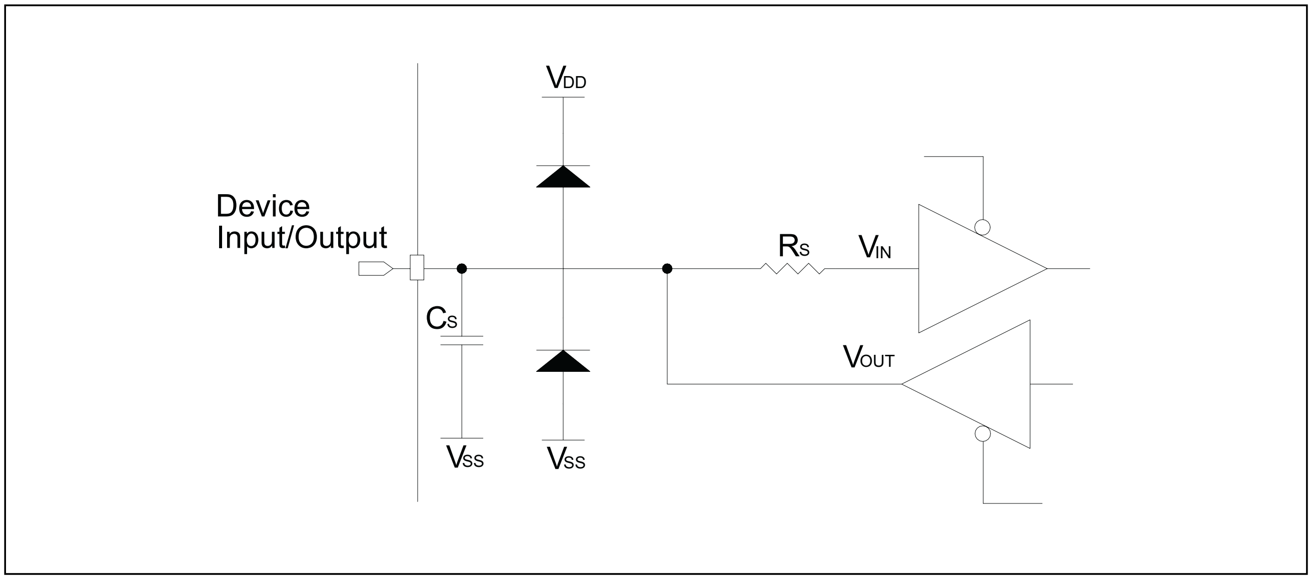


Figure 11.13 Port B Pin Equivalent Circuit



11.9 Control Pin Characteristics

11.9.1 RESET pin

Subject to general operating condition for V_{DD}, f_{osc}, and T_A unless otherwise specified.

Table 11.12 Reset pin

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IL}	Input low level voltage[1)]	V_{DD}=5 V			1.8	V
V_{IH}	Input high level voltage[1)]	V_{DD}=5 V	2.8			
V_{hys}	Schmitt trigger voltage hysteresis[2)]	V_{DD}=5 V		0.8		
$t_{w(RSTL)out}$	General reset pulse duration			30		μS
$t_{h(RSTL)int}$	External reset pulse hold time		20			

11.9.2 V_{PP} pin

Subject to general operating condition for V_{DD}, f_{osc}, and T_A unless otherwise specified.

Table 11.13 V_{PP}[4)] pin

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IL}	Input low level voltage[3)]		V_{SS}		0.2	V
V_{IH}	Input high level voltage[3)]		V_{DD}-0.1		12.6	

Notes:

1.Data based on characterization results not tested in production.

2. Hysteresis voltage between Schmitt trigger switching level. Based on characterization results not tested in production.

3. Data based on design simulation and/or technology characteristics, not tested in production.

4. In working mode V_{PP} must be tied to V_{SS}

11.10 8-bit A/D Characteristics

Subject to general operating condition for V_{DD}, f_{osc}, and T_A unless otherwise specified.

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
Res	Resolution			8		bit
A_{TOT}	Total Accuracy[1)]	1 MHz<f_{ADC}< 20 MHz		$\pm$1		LSB
t_C	Conversion Time			82/f_{ADC}		µS
V_{AN}	Conversion Range		V_{SSA}		V_{DDA}	V
V_{ZI}	Zero Scale Voltage	Conversion result = 00 Hex		V_{SSA}		V
V_{FS}	Full Scale Voltage	Conversion result = FF Hex		V_{DDA}		V
AD_I	Analog Input Current during Conversion	f_{ADC}=20MHz		0.1		µA
AC_{IN}	Analog Input Capacitance				25	pF
f_{ADC}	ADC Clock frequency		f_{osc}/2		f_{osc}	MHz

Notes:

1. Noise on V_{DDA}, V_{SSA} < 40 mV

INSTRUCTION SET

ADD

Addition

Format: **add** dst, src

Operation: dst ⇐ dst + src

Description: The content of the RAM location specified as source is added to the content of the destination location, leaving the result in the destination.

Flags:

Z	sets if result is zero, cleared otherwise.
C	sets if overflow, cleared otherwise.
S	not affected.

Bytes: 3
Cycles: 17

Example: If the RAM location 20 contains the value 45 and the RAM location 11 contains the value 15, then the instruction

add 20, **11** 0010000 000010100 00001011

causes the location 20 of the RAM to be loaded with the value 60.

If the location 20 contains the value 200 and the location 11 contains the value 100, the instruction causes the location 20 to be loaded with the value 44 (result-256) and the C flag to be set.

ADDO

Addition with Offset

Format: **addo** dst, src

Operation: dst ⇐ dst + src - 128

Description: The content of the RAM location specified as source is added to the content of the destination location, the value 128 is subtracted from the result that is stored in the destination. This operation allows the use of the signed byte considering the values between 0 and 127 as negative, 128 as 0, and the values between 129 and 255 as positive.

Flags:

Z	sets if result is zero, cleared otherwise.
C	sets if overflow, cleared otherwise.
S	sets if underflow, cleared otherwise.

Bytes: 3

Cycles: 20

Example: If the RAM location 20 contains the value 100 and the RAM location 11 contains the value 40, then the instruction

```
addo 20, 11    00100001        00010100        00001011
```

causes the location 20 of the RAM to be loaded with the value 12.

If the location 20 contains the value 100 and the location 11 contains the value 10, the instruction causes the location 20 to be loaded with the value 238 (256+result) and the S flag to be set. If the location 20 contains the value 200 and the location 11 contains the value 228, the instruction causes the location 20 to be loaded with the value 44 (result-256) and the C flag to be set.

AND

Logical AND

Format: **and** dst, src

Operation: **Dst $\Leftarrow$ dst AND src**

Description: The instruction logically ANDs the content of the RAM locations specified as source and as destination, leaving the result in the destination.

Flags:

Z	sets if result is zero, cleared otherwise.
C	not affected
S	not affected

Bytes: 3
Cycles: 17

Example: If the RAM location 20 contains the value 240 (11110000b) and the RAM location 11contains the value 85 (01010101b), then the instruction

and 20, 11	00100010	00010100	00001011

causes the location 20 of the RAM to be loaded with the value 80 (01010000b).

ASL

Arithmetic Shift Left

Format:	**asl** dst
Operation:	**C $\Leftarrow$ dst (7)** **dst (0) $\Leftarrow$ 0** **dst (n+1) $\Leftarrow$ dst(n) where n = 0-6**
Description:	The instruction shifts one bit left the content of the RAM location specified as destination. The most significative bit is placed in the C flag and the less significative bit is loaded with 0.
Flags:	Z sets if result is zero, cleared otherwise. C sets if MSB is set, cleared otherwise. S not affected.
Bytes:	2
Cycles:	15

Example: if the RAM location 20 contains the value 85 (01010101b), then the instruction:

asl 20	00101001	00010100

causes the location 20 of the RAM to be loaded with the value 170 (10101010b).

If the RAM location 20 contains the value 150 (10010110b), then the instruction causes the location 20 of the RAM to be loaded with the value 44 (00101100b) and the C flag to be set.

ASR

Arithmetic Shift Right

Format:	**asr** dst
Operation:	**S ⇐ dst (0)** **dst (7)⇐ 0** **dst (n) ⇐ dst (n+1) where n = 0-6**
Description:	The instruction shifts one bit right the content of the RAM location specified as destination. The less significative bit is placed in the S flag and the most significative bit is loaded with 0.
Flags:	Z sets if result is zero, cleared otherwise. C not affected. S sets if LSB is set, cleared otherwise.
Bytes:	2
Cycles:	15

Example: If the RAM location 20 contains the value 170 (10101010b), then the instruction:

asr 20	00101010	00010100

causes the location 20 of the RAM to be loaded with the value 85 (01010101b).

If the RAM location 20 contains the value 85 (01010101b), then the instruction causes the location 20 of the RAM to be loaded with the value 42 (00101010b) and the S flag to be set.

CALL

Subroutine Call

Format:	**call** label

Operation: **SP ⇐ SP -2** (SP = Stack Pointer)
(SP) ⇐ PC (PC = Program Counter)
PC ⇐ label

Description: The content of the Program Counter (PC) is pushed to the top of the System Stack and the location address specified by the symbol label is loaded into the PC in order to point to the first instruction of the subroutine.

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes: 3
Cycles: 18

Example: If the label “subx”, that indicates the first location of a subroutine, is located to the address 2500 (00001001 11000100), then the instruction:

call subx	01000111	00001001	11000100

causes the PC to be loaded with the value 2500 and the program to jump to the subroutine labelled “subx”.

DEC

Decrement

Format:	**dec** dst
Operation:	**dst $\Leftarrow$ dst - 1**
Description:	The content of the specified RAM location is decremented by 1.
Flags:	Z sets if result is zero, cleared otherwise.
	C not affected.
	S sets if underflow, cleared otherwise.
Bytes:	2
Cycles:	15

Example: If the RAM location 20 contains the value 50, then the instruction:

dec 20	00101100	00010100

causes the location 20 of the RAM to be loaded with the value 49.

If the RAM location 20 contains the value 0, then the instruction causes the location 20 to be loaded with the value 255 and the S flag to be set.

DIV

Division (16/8)

Format:	**div** dst, src
Operation:	**[dst dst+1] / src :** **dst ⇐ remainder** **dst + 1 ⇐ result**
Description:	The content of the destination RAM location pair (the 16 bit dividend is composed by the dst (MSByte) and dst+1(LSByte) locations) is divided by the source. The LSByte of the destination location pair (dst+1) is loaded with the result, the MSByte (dst) is loaded with the remainder. In case of overflow the MSByte and the LSByte are loaded both with 255.

Flags:

Z	sets if result is zero, cleared otherwise.
C	sets if overflow, cleared otherwise.
S	sets if remainder is zero, cleared otherwise.

Bytes: 3
Cycles: 26

Example: If the RAM location pair 20 and 21 contains the value 1523 and the location 40 contains the value 30, then the instruction:

div 20, 40	00100011	00010100	00101000

causes the location 21 of the RAM to be loaded with the value 50 and the location 20 with the value 23.

FUZZY

Fuzzy Computation

Format:	**fuzzy**
Operation:	**Start fuzzy output computation**
Description:	This instruction transfers the control to the Fuzzy Computation Unit for the evaluation of a single fuzzy output. After this instruction, only fuzzy instructions can be inserted until the instruction OUT is specified. If more fuzzy output have to be computed, the instruction FUZZY should be specified again after the instruction OUT.

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes: 1
Cycles: 6

Example: The following instruction:

fuzzy 10000000

starts a fuzzy computation section.

HALT

Halt

Format:	**halt**
Operation:	**Clock Master halted.**
Description:	This instruction stops the clock master so that the CPU and the peripherals are turned-off. It is possible to exit from the halt mode by means of an external interrupt or a chip reset.

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes: 1
Cycles: 7 - 15

Example: After the instruction:

Halt 00110111

the device is put in halt mode and the program is stopped until an external interrupt or a chip resets.

INC

Increment

Format:	**inc** dst
Operation:	**dst ⇐ dst + 1**
Description:	The content of the specified RAM location is incremented by 1.
Flags:	Z sets if result is zero, cleared otherwise. C sets if overflow, cleared otherwise. S not affected.
Bytes:	2
Cycles:	15

Example: If the RAM location 20 contains the value 50, then the instruction:

inc 20	00101101	00010100

causes the location 20 of the RAM to be loaded with the value 51.

If the RAM location 20 contains the value 255, then the instruction causes the location 20 to be loaded with the value 0 and the S and Z flags to be set.

JP

Unconditional Jump

Format:	**jp** label
Operation:	**PC** ⇐ **label** (PC = Program Counter)
Description:	This instruction causes the address value specified by the symbol “label” to be loaded into the Program Counter (PC) and the Program jumps to the instruction located at the address labeled with “label”.
Flags:	Z not affected. C not affected. S not affected.
Bytes:	3
Cycles:	12

Example: If the Program Memory location 2500 (00001001b 11000100b) is labeled with “labelx”, then the instruction:

jp labelx	01000000	00001001	11000100

loads the value 2500 into the PC and transfers the program control to that location.

JPC

Jump if C Flag Set

Format:	**jpc** label
Operation:	**if C=1, PC ⇐ label** (PC = Program Counter)
Description:	If C flag is set, this instruction causes the address value specified by the symbol “label” to be loaded into the Program Counter (PC) and the Program jumps to the instruction located at the address labeled with “label”. Otherwise the control passes to the next instruction.
Flags:	Z not affected. C not affected. S not affected.
Bytes:	3
Cycles:	12 if jump, 10 otherwise
Example:	If the Program Memory location 2500 (00001001b 11000100b) is labeled with “labelx”, and the C flag is set then the instruction:

jpc labelx	01000101	00001001	11000100

loads the value 2500 into the PC and transfers the program control to that location.

JPNC

Jump if C Flag Not Set

Format: **jpnc** label

Operation: **if C=0, PC ⇐ label** (PC = Program Counter)

Description: If C flag is not set, this instruction causes the address value specified by the symbol “label” to be loaded into the Program Counter (PC) and the Program jumps to the instruction located at the address labeled with “label”. Otherwise the control passes to the next instruction.

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes: 3

Cycles: 12 if jump, 10 otherwise

Example: If the Program Memory location 2500 (00001001b 11000100b) is labeled with “labelx”, and the C flag is not set then the instruction:

jpnc labelx	01000110	00001001	11000100

loads the value 2500 into the PC and transfers the program control to that location.

JPNS

Jump if S Flag Not Set

Format:	**jpns** label
Operation:	**if S=0, PC ⇐ label** (PC = Program Counter)
Description:	If S flag is not set, this instruction causes the address value specified by the symbol “label” to be loaded into the Program Counter (PC) and the Program jumps to the instruction located at the address labeled with “label”. Otherwise the control passes to the next instruction.

Flags:	Z	not affected.
	C	not affected.
	S	not affected.

Bytes: 3
Cycles: 12 if jump, 10 otherwise

Example: If the Program Memory location 2500 (00001001b 11000100b) is labeled with “labelx”, and the S flag is not set then the instruction:

jpns labelx	01000010	00001001	11000100

loads the value 2500 into the PC and transfers the program control to that location.

JPNZ

Jump if Z Flag Not Set

Format:	**jpnz** label
Operation:	if Z=0, PC ⇐ label (PC = Program Counter)
Description:	If Z flag is not set, this instruction causes the address value specified by the symbol “label” to be loaded into the Program Counter (PC) and the Program jumps to the instruction located at the address labelled with “label”. Otherwise the control passes to the next instruction.
Flags:	Z not affected. C not affected. S not affected.
Bytes:	3
Cycles:	12 if jump, 10 otherwise

Example: If the Program Memory location 2500 (00001001b 11000100b) is labelled with “labelx”, and the Z flag is not set then the instruction:

```
jpnz labelx      01000100          00001001          11000100
```

loads the value 2500 into the PC and transfers the program control to that location.

JPS

Jump if S Flag Set

Format: **jps** label

Operation: **if S=1, PC ⇐ label** (PC = Program Counter)

Description: If S flag is set, this instruction causes the address value specified by the symbol “label” to be loaded into the Program Counter (PC) and the Program jumps to the instruction located at the address labeled with “label”. Otherwise the control passes to the next instruction.

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes: 3
Cycles: 12 if jump, 10 otherwise

Example: If the Program Memory location 2500 (000
01001b 11000100b) is labeled with “labelx”, and the S flag is set then the instruction:

jps labelx 01000001 00001001 1000100

loads the value 2500 into the PC and transfers the program control to that location.

JPZ

Jump if Z Flag Set

Format: **jpz** label

Operation: **if Z=1, PC $\Leftarrow$ label** (PC = Program Counter)

Description: If Z flag is set, this instruction causes the address value specified by the symbol “label” to be loaded into the Program Counter (PC) and the Program jumps to the instruction located at the address labeled with “label”. Otherwise the control passes to the next instruction.

Flags:		
	Z	not affected.
	C	not affected.
	S	not affected.

Bytes: 3
Cycles: 12 if jump, 10 otherwise

Example: If the Program Memory location 2500 (00001001b 11000100b) is labeled with “labelx”, and the Z flag is set then the instruction:

jpz labelx	01000011	00001001	11000100

loads the value 2500 into the PC and transfers the program control to that location.

LDCE

Load Configuration, EPROM

Format:	**ldce** dst, src
Operation:	**dst ⇐ src**
Description:	The instruction loads into the configuration register specified as destination the data contained in the Program Memory source location in the current page, specified with the PGSET instruction.
Flags:	Z not affected. C not affected. S not affected.
Bytes:	3
Cycles:	17
Example:	if the Program Memory location 300 contains the value 240 and the current page is set to 1 (256+44=300), then the instruction:

ldce 12, 44	00011010	00001100	00101100

causes the configuration register 12 to be loaded with the value 240.

LDCR

Load Configuration, RAM

Format	**ldcr** dst, src
Operation:	**dst ⇐ src**
Description:	The instruction loads into the configuration register specified as destination the data contained in the RAM location specified as source.
Flags:	Z not affected.
	C not affected.
	S not affected.
Bytes:	3
Cycles:	14

Example: If the RAM location 80 contains the value 64 then the instruction

ldcr 12, 80	00010100	00001100	01010000

causes the configuration register 12 to be loaded with the value 64.

LDFR

Load Fuzzy, RAM

Format:	**ldfr** dst, src
Operation:	**dst ⇐ src**
Description:	The instruction loads into Fuzzy input registers (0 to 7) specified as destination the data contained in the RAM location specified as source.
Flags:	Z not affected. C not affected. S not affected.
Bytes:	3
Cycles:	14

Example: If the RAM location 80 contains the value 64 then the instruction

ldfr 2, 80	00011000	00000010	01010000

causes the fuzzy input register 2 to be loaded with the value 64, that is used as crisp input value of the third fuzzy variable.

LDPE

Load Peripheral, EPROM Indirect

Format:	**ldpe** dst, (src)
Operation:	**dst $\Leftarrow$ (src)**
Description:	The instruction loads into the Output Peripheral Register specified as destination the data contained in the EPROM location which address (in the page set with the PGSET instruction) is contained in the location specified as source.
Flags:	Z not affected. C not affected. S not affected.
Bytes:	3
Cycles:	17
Example:	If the currently EPROM page set is 2, the RAM location 30 contains the value 10 and the EPROM location 522 (256*2+10) contains the value 100, then the instruction:

ldpe 2, (30)	00010110	00000010	00011110

causes the Output Peripheral Register 2 to be loaded with the value 100.

LDPR

Load Peripheral, RAM

Format:	**ldpr** dst, src
Operation:	**dst ⇐ src**
Description:	The instruction loads into the Output Peripheral Register specified as destination the data contained in the RAM location specified as source.

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes: 3
Cycles: 14

Example: If the RAM location 30 contains the value 100, then the instruction:

ldpr 2, 30	00010101	00000010	00011110

causes the Output Peripheral Register 2 to be loaded with the value 100.

LDRC

Load RAM, Constant

Format: **ldrc** dst, const

Operation: **dst ⇐ const**

Description: The instruction loads into the RAM location specified as destination the constant specified as source.

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes: 3
Cycles: 14

Example: The following instruction:

ldrc 24, 130   00010000   00011000   10000010

causes the RAM location 24 to be loaded with the value 130.

LDRE

Load RAM, EPROM

Format:	**ldre** dst, src
Operation:	**dst ⇐ src**
Description:	The instruction loads into the RAM location specified as destination the contents of the EPROM location specified as source (in the page set with the PGSET instruction).
Flags:	Z not affected. C not affected. S not affected.
Bytes:	3
Cycles:	16

Example: If the currently set EPROM page is 2 and the address 522 (256*2+10) contains the value 100, then the following instruction:

```
ldre 24, 10     00010001          00011000          00001010
```

causes the RAM location 24 to be loaded with the value 100.

(LDRE)

Load RAM Indirect, EPROM Indirect

Format: **ldre** (dst), (src)

Operation: **(dst) ⇐ (src)**

Description: The instruction loads into the RAM location, which address is contained in the RAM location specified as destination, the contents of the EPROM location, which address is contained in the RAM location specified as source (in the page set with the PGSET instruction).

Flags:	Z	not affected.
	C	not affected.
	S	not affected.

Bytes: 3
Cycles: 18

Example: If the currently set EPROM page is 2, the RAM location 20 contains the value 10, the address 522 (256*2+10) contains the value 100 and the RAM location 24 contains the value 50, then the following instruction:

```
ldre (24), (20)      00010010      00011000            00001010
```

causes the RAM location 50 to be loaded with the value 100.

LDRI

Load RAM, Peripheral Input

Format:	**ldri** dst, src
Operation:	**dst ⇐ src**
Description:	The instruction loads into the RAM location specified as destination the contents of the Input Peripheral Register specified as source.

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes: 3
Cycles: 15

Example: If the Input Peripheral Register 10 contains the value 100, then the following instruction:

ldri 24, 10	00010011	00011000	00001010

causes the RAM location 24 to be loaded with the value 100.

LDRR

Load RAM, RAM

Format:	**ldrr** dst, src
Operation:	**dst $\Leftarrow$ src**
Description:	The instruction loads into the RAM location specified as destination the contents of RAM location specified as source.
Flags:	Z not affected. C not affected. S not affected.
Bytes:	3
Cycles:	16

Example: if the RAM location 10 contains the value 100, then the following instruction:

ldrr 24, 10	00010111	00011000	00001010

causes the RAM location 24 to be loaded with the value 100.

MDGI

Macro Disable Global Interrupts

Format:	**mdgi**
Operation:	**all interrupts disabled**
Description:	This instruction is used by the FUZZYSTUDIO Compiler in order to disable the interrupts at the beginning of a Compiler Macro.
Flags:	Z not affected.
	C not affected.
	S not affected.
Bytes:	1
Cycles:	7 if GI already disabled, 15 otherwise
Example:	After the instruction:

mdgi 00110100

interrupts cannot be serviced until the Global Interrupt Mask (GI) is again enabled with a MEGI instruction.

MEGI

Macro Enable Global Interrupts

Format: **megi**

Operation: **not masked interrupts enabled**

Description: This instruction is used by the FUZZYSTUDIO Compiler in order to enable not masked interrupts after the end of a Compiler Macro. Interrupts cannot be enabled if a UDGI instruction, not followed by a UEGI instruction, has been specified.

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes: 1

Cycles: 7 if GI already enabled, 15 otherwise

Example: If a UDGI instruction, not followed by a UEGI instruction, has not been specified, after the instruction:

megi 00110101

not masked interrupts are enabled.

MIRROR

Byte Mirror

Format: **mirror** dst

Operation: **dst(n) ⇐ dst(7-n)**

Description: This instruction modifies the content of the specified RAM location, inverting the order of the bits.

Flags:	Z	set if result is zero, cleared otherwise.
	C	not affected.
	S	not affected.

Bytes: 2
Cycles: 15

Example: If the RAM location 24 contains the value 142 (10001110b), after the instruction:

mirror 24 00101011 00011000

the RAM locations will contain the value 113 (01110001b).

MULT

Multiplication (8 X 8)

Format: **mult** dst, src

Operation: **[dst dst+1] ⇐ dst * src**

Description: The instruction computes the product between the values contained in the RAM locations specified as destination and as source. The result is a 16 bit number which the most significative byte is stored in the destination location and the least significative is stored in the location after the destination.

Flags:

Z	set if result is zero, cleared otherwise.
C	not affected.
S	not affected.

Bytes: 3
Cycles: 19

Example: If the RAM location 20 contains the value 100 and the location 40 contains the value 30, then the instruction:

mult 20, 40 00100100 00010100 00101000

causes the location 20 of the RAM to be loaded with the value 11 (MSB) and the location 21 with the value 184 (256*11+184=30*100=3000).

NOP

No Operation

Format:	**nop**
Operation:	**No operation.**
Description:	No operation is carried out with this instruction. It is typically used for timing delay.

Flags:	Z	not affected.
	C	not affected.
	S	not affected.

Bytes:	1
Cycles:	6

Example: The instruction:

nop	10000001

causes the program control to pass to the next instruction after 6 clock cycles.

NOT

Logical NOT

Format:	**not** dst
Operation:	**dst ⇐ 255-dst**
Description:	This instruction negates each bit of the location specified as destination.
Flags:	Z sets if result is zero, cleared otherwise.
	C not affected.
	S not affected.
Bytes:	2
Cycles:	15
Example:	If the location 24 contains the value 100 (01100100b), the instruction:

not 24 00100101 00011000

causes the location 24 to be loaded with the value 155 (10011011b).

OR

Logical OR

Format:	**or** dst, src
Operation:	**dst ⇐ dst OR src**
Description:	The instruction logically ORs the content of the RAM locations specified as source and as destination, leaving the result in the destination.

Flags:

Z	sets if result is zero, cleared otherwise.
C	not affected.
S	not affected.

Bytes: 3
Cycles: 17

Example: If the location 24 contains the value 100 (01100100b), and the location 10 contains the value 15 (00001111b), then the instruction:

or 24, 10	00100110	00011000	00001010

causes the location 24 to be loaded with the value 111 (01101111b).

PGSET

Page Set

Format:	**pgset** const
Operation:	**Page pointer setting.**
Description:	This instruction sets the current EPROM page to the const page, so that the locations that can be addressed are in the range [256*const , 256*cost+255]

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes: 2
Cycles: 9

Example: The instruction:

pgset 4	00011001	00000100

sets the current page to the fifth page (addresses 1024-1279).

RET

Return from Subroutine

Format:	**ret**
Operation:	**PC ⇐ (SP)** (PC = Program Counter) **SP ⇐ SP+ 2** (SP = Stack Pointer)
Description:	This instruction performs the return from a subroutine. It determines the jump of the program to the line after the subroutine call instruction.
Flags:	Z not affected. C not affected. S not affected.
Bytes:	1
Cycles:	13

Example: If the value to the top of the stack is 0e4h, the instruction:

ret	01001000

determines the PC to be loaded with the value 0e4h and the previous value to be lost.

RETI

Return from Interrupt

Format: **reti**

Operation:

PC ⇐ (SP)   (PC = Program Counter)
SP ⇐ SP + 2 (SP = Stack Pointer)
Flags ⇐ saved flags

Description: This instruction performs the return from a interrupt service routine. It determines the return of the device to the state it was before the interrupt. The value of the PC is popped from the top of the stack, together with the saved flags.

Flags:

Z	restored.
C	restored.
S	restored.

Bytes: 1
Cycles: 12

Example: If the value to the top of the stack is 0e4h, the instruction:

reti 00110000

determinates the PC to be loaded with the value 0e4h, the previous value to be lost and the flags status before the interrupt to be restored.

RINT

Reset Interrupt

Format: **rint** const

Operation: **Interrupt No. const Pending bit ∂ 0**

Description: This instruction resets the pending bit of the interrupt No.const. After this instruction the request of interrupt is cancelled and will not be acknowledged

Flags:

z	not affected.
C	not affected.
S	not affected.

Bytes: 2

Cycles: 8

Example: If the interrupt 3 source has generated an interrupt request remaining pending (being the interrupt masked or globally disabled), after the instruction

rint 3 00110001 00000011

the interrupt request is cancelled and will be serviced when enabled only if a successive request is sent.

SUB

Subtraction

Format:	**sub** dst, src
Operation:	**dst ⇐ dst - src**
Description:	The content of the RAM location specified as source is subtracted to the contents of destination location, leaving the result in the destination.

Flags:

Z	sets if result is zero, cleared otherwise.
C	not affected.
S	sets if underflow, cleared otherwise.

Bytes: 3
Cycles: 17

Example: if the RAM location 20 contains the value 45 and the RAM location 11 contains the value 15, then the instruction

sub 20, 11	00100111	00010100	00001011

causes the location 20 of the RAM to be loaded with the value 30.

If the location 20 contains the value 80 and the location 11 contains the value 100, the instruction causes the location 20 to be loaded with the value 236 (256 + result) and the S flag to be set.

SUBO

Subtraction with Offset

Format:	**subo** dst, src
Operation:	**dst $\Leftarrow$ dst + 128 - src**

Description: The value 128 is added to the content of the RAM location specified as destination, then the content of source location is subtracted to the result and stored into the destination location. This operation allows the use of the signed byte considering the values between 0 and 127 as negative, 128 as 0, and the values between 129 and 255 as positive.

Flags:

Z	sets if result is zero, cleared otherwise.
C	sets if overflow, cleared otherwise.
S	sets if underflow, cleared otherwise.

Bytes:	3
Cycles:	20

Example: if the RAM location 20 contains the value 45 and the RAM location 11 contains the value 65, then the instruction

subo 20, 11	00101000	00010100	00001011

causes the location 20 of the RAM to be loaded with the value 108.

If the location 20 contains the value 200 and the location 11 contains the value 20, the instruction causes the location 20 to be loaded with the value 52 (result-256) and the C flag to be set.If the location 20 contains the value 20 and the location 11 contains the value 200, the instruction causes the location 20 to be loaded with the value 204 (256+result) and the S flag to be set.

UDGI

User Disable Global Interrupts

Format:	**udgi**
Operation:	**all interrupts disabled**
Description:	This instruction can be used by the User in order to disable globally the interrupts.

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes:	1
Cycles:	6

Example: After the instruction:

udgi 00110010

interrupts cannot be serviced until the Global Interrupt Mask (GI) is again enabled with a UEGI instruction.

UEGI

User Enable Global Interrupts

Format:	**uegi**
Operation:	**not masked interrupts enabled**
Description:	This instruction can be used by the user in order to enable not masked interrupts. Interrupts cannot be enabled if a MDGI instruction, not followed by a MEGI instruction, has been specified.
Flags:	Z not affected.
	C not affected.
	S not affected.
Bytes:	1
Cycles:	7 if GI already enabled, 15 otherwise

Example: If a MDGI instruction, not followed by a MEGI instruction, has not been specified, after the instruction:

uegi 00110011

not masked interrupts are enabled.

WAITI

Wait for Interrupt

Format: **waiti**

Operation: **wait for interrupt**

Description: This instruction stops the program execution until an interrupt from an active source is requested. During the wait state some functionalities of the device are turned off in order to lower the power consumption.

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes: 1
Cycles: 7 - 14

Example: The instruction:

wait 00110110

puts the chip in wait mode and stops the program execution, waiting for an interrupt signal. If there are no active interrupt sources, the device can exit from the wait mode only with a reset.

WDTRFR

Watchdog Refresh

Format: **wdtrfr**

Operation: **Watchdog counter enabled or refreshed**

Description: If the Watchdog is disabled, this instruction enables the watchdog and the counter starts to count from the configured value. If the watchdog is already enabled, this instruction restarts the counting from the beginning.

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes: 1
Cycles: 7

Example: After the instruction:

wtdrfr 10000010

the Watchdog is enabled and the value of counting stored in the Configuration Register 2 is loaded in the Watchdog counter.

WDTSLP

Watchdog Sleep

Format:	**wdtslp**
Operation:	**Watchdog disabled**
Description:	This instruction disables the Watchdog, avoiding the chip reset.

Flags:

Z	not affected.
C	not affected.
S	not affected.

Bytes: 1
Cycles: 6

Example: After the instruction:

wtdslp 10000011

the Watchdog is disabled stopping the counter .

ST52x420 Assembler Pseudo Instructions:

The Assembler pseudo instructions have not direct correspondence with the machine code; this is obtained after the elaboration of the supplied data by means of the Assembler.

The Assembler pseudo instructions are used to set the data for the Fuzzy Computation, the Assembler then optimizes these data considering the code format used from the Fuzzy Computation Unit.

There are also the pseudo instructions to set data and to set the current location in EPROM Memory.

CON

Consequent

Format: **con** const

Operation: **Dividend Register ⇐ Dividend register + Teta * const**
Divisor Register ⇐ Divisor Register + Teta

Description: This instruction computes the values to add in the defuzzyfication registers, at the end of the single rule. The specified constant is the crisp value representing the output crisp membership function: it is multiplied by the last fuzzy operation result.

DATA

EPROM Data

Format: **data** page, addr, value

Operation: **none**

Description: This pseudo instruction indicates to the Assembler to store data in the EPROM. The location in the address of the specified page is loaded with the specified value.

IRQ

Interrupt Request Vector

Format: **irq** int, label

Operation: **none**

Description: This pseudo-instruction indicates the interrupt vectors to the Assembler.

The argument represents respectively the interrupt and the relative interrupt service routine first address, pointed with a label.

FZAND

Fuzzy AND

Format: **fzand**

Operation: **K ⇐ MIN(stack(0) , stack(1))**

Description: This instruction computes the Fuzzy AND operation (minimum) between the two values stored in the Fuzzy stack, previously loaded with LDP, LDN or LDK instructions, and stores the result in the register K.

FZOR

Fuzzy OR

Format: **fzor**

Operation: **K ⇐ MAX(stack(0) , stack(1))**

Description: This instruction computes the Fuzzy OR operation (maximum) between the two values stored in the Fuzzy stack, previously loaded with LDP, LDN or LDK instructions, and stores the result in the register K.

LDK

Load Stack with K Register

Format: **ldk**

Operation: **stack ⇐ K**

Description: The instruction loads in the Fuzzy stack the value temporarily stored in the Fuzzy register K that is the result of the last Fuzzy operation.

LDM

Load Stack with M Register

Format: **ldm**

Operation: **stack ⇐ M**

Description: The instruction loads in the Fuzzy stack the value temporarily stored in the Fuzzy register M with a previous SKM operation.

LDN

Load Negative Alpha Value

Format: **ldn** var, mbf

Operation: **stack ⇐ 15 - computed alpha value related to mbf M.F. of var Variable**

Description: The instruction performs the fuzzyfication and loads in the stack the negated alpha value of the mbf M.F. of the var Variable.

LDP

Load Positive Alpha Value

Format: **ldp** var, mbf

Operation: **stack ⇐ computed alpha value related to mbf M.F. of var Variable**

Description: The instruction performs the fuzzyfication and loads in the stack the alpha value of the mbf M.F. of the var Variable.

MBF

Membership Function

Format: **mbf** num, lvd, vtx, rvd

Operation: **none**

Description: This pseudo instruction indicates to the Assembler to store a Membership Function data in the EPROM Memory. The M.F. number is specified as first argument, followed by the left semibase width, the vertex position and the right semibase width. The first (of three) EPROM location where the data are stored is the current program line.

OUT

Fuzzy Output

Format: **out** dst

Operation: **dst ⇐ current fuzzy output defuzzyfication result.**

Description: This instruction performs the defuzzyfication for the computation of the current fuzzy output and store the result in the destination RAM location.

SETMEM

Set Memory

Format: **setmem** page, addr

Operation: **none**

Description: This pseudo-instruction indicates that the next current program line must be the one in the specified address of the specified page.

SKM

Store K Register in M Register

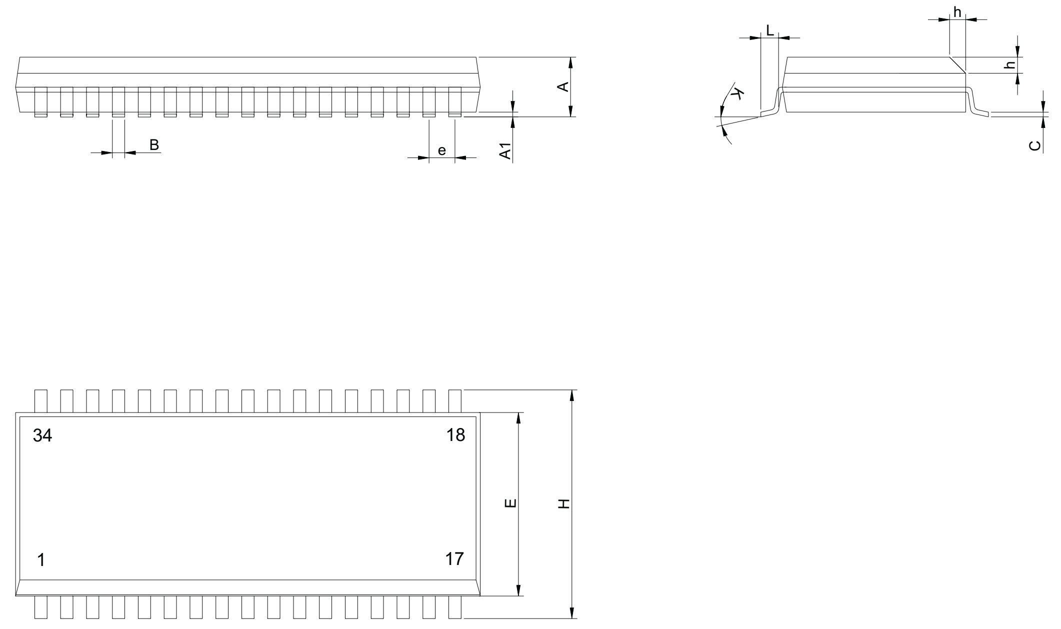
Format: **skm**

Operation: **M $\Leftarrow$ K**

Description: This instruction loads the result of the last performed Fuzzy operation (stored in the temporary register K) in the temporary buffer M.

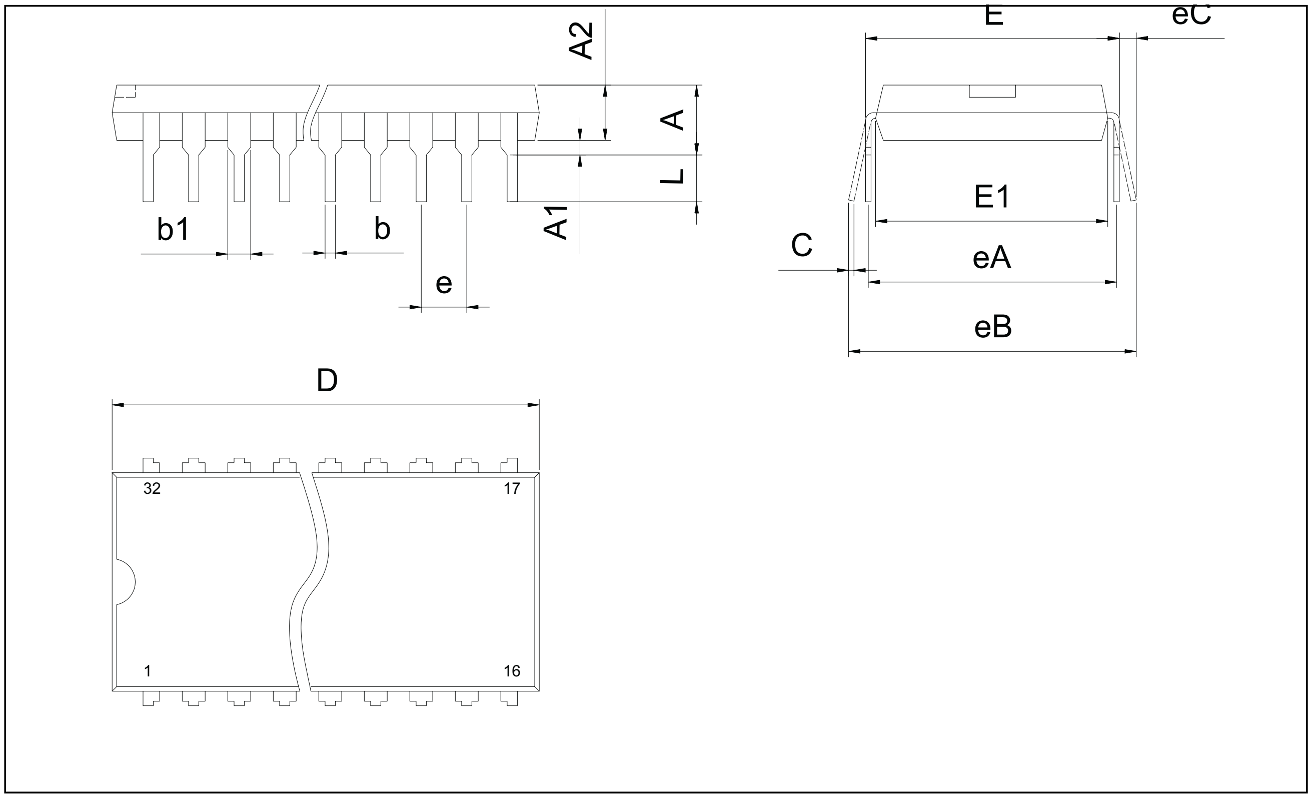
SSO34 PACKAGE MECHANICAL DATA

DIM	mm			inch.		
	MIN	TYP.	MAX	MIN	TYP.	MAX
A	2.4638		2.6416	0.097		0.104
A1	0.127		0.2921	0.005		0.0115
B	0.3556		0.4826	0.014		0.019
C	0.23114		0.3175	0.0091		0.0125
D	17.7292		18.0594	0.698		0.711
E	7.4168		7.5946	0.292		0.299
e		1.016			0.040	
H	10.16		10.414	0.400		0.410
h	0.635		0.7366	0.025		0.029
k	0°		8°	0°		8°
l	0.6096		1.016	0.024		0.040



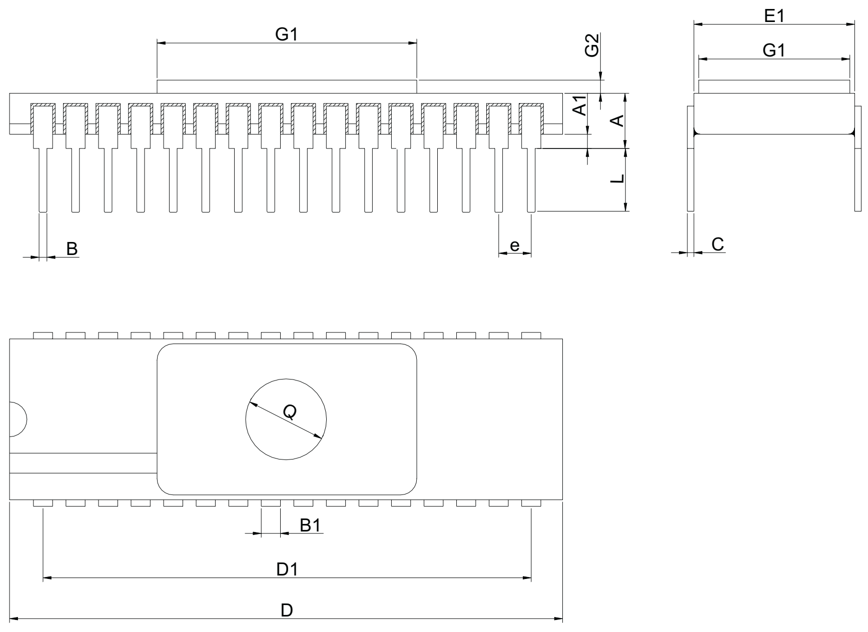
PDIP32 Shrink PACKAGE MECHANICAL DATA

DIM	mm			inch.		
	MIN	TYP.	MAX	MIN	TYP.	MAX
A	3.556	3.7592	5.08	0.140	0.148	0.200
A1	0.508			0.02		
A2	3.048	3.556	4.572	0.120	0.140	0.18
b	0.3556	0.4572	0.5842	0.014	0.018	0.023
b1	0.762	1.016	1.397	0.030	0.040	0.055
C	0.2032	0.254	0.3556	0.008	0.010	0.014
D	27.432	27.94	28.448	1.080	1.100	1.120
E	9.906	10.414	11.049	0.390	0.410	0.435
E1	7.62	8.89	9.398	0.300	0.350	0.370
e		1.778			0.070	
eAl		10.16			0.400	
eB			12.7			0.500
eC			1.397			0.055
L	2.54	3.048	3.81	0.100	0.120	0.150



CSDIP32W Shrink PACKAGE MECHANICAL DATA

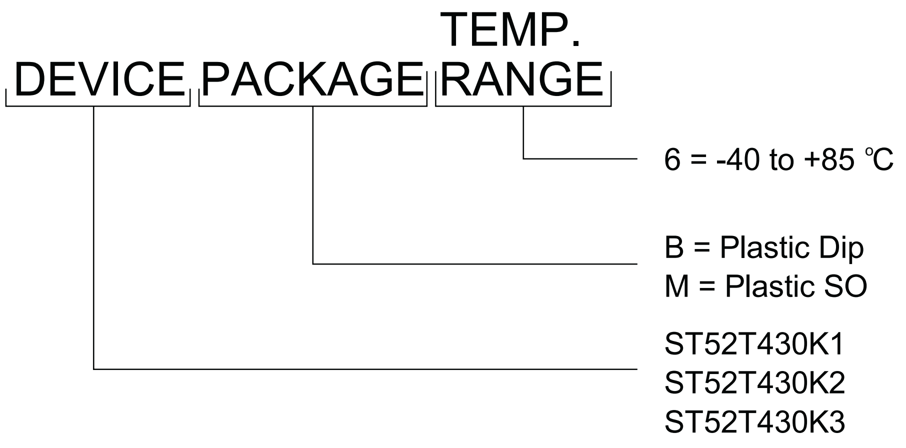
DIM	mm			inch.		
	MIN	TYP.	MAX	MIN	TYP.	MAX
A	2.4638	2.921	3.3782	0.097	0.115	0.133
A1	0.635	0.889	1.143	0.025	0.035	0.045
B	0.4064	0.4572	0.508	0.016	0.018	0.020
B1	0.889			0.035		
C	0.2032	0.254	0.3048	0.008	0.010	0.012
D	29.6672	29.972	30.2768	1.168	1.180	1.192
D1	26.4668	26.67	26.8732	1.042	1.050	1.058
E1	9.7028	9.906	10.1092	0.382	0.390	0.398
e	1.651	1.778	1.905	0.065	0.070	0.075
G		9.525			0.375	
G1		14.732			0.580	
G2		1.1176			0.044	
L	4.318	4.445	4.572	0.170	0.175	0.180
Q			7.366			0.290



ORDERING INFORMATION

Each device is available for production in user programmable version (OTP) as well as in factory programmed version (FASTROM). OTP device are shipped to customer with a default blank content FFh, while FASTROM factory programmed parts contain the code sent by customer. There is one common EPROM version for debugging and prototyping which features the maximum memory size and peripherals of the family. Care must be taken to only use resources available on the target device.

Figure 12.1 OTP User Programmable Device Types



PART NUMBER	TEMPERATURE RANGE	PACKAGE
ST52T430K1M6	-40TO +85 °C	SSO34
ST52T430K2M6	-40TO +85 °C	SSO34
ST52T430K3M6	-40TO +85 °C	SSO34
ST52T430K1B6	-40TO +85 °C	PSDIP32
ST52T430K2B6	-40TO +85 °C	PSDIP32
ST52T430K3B6	-40TO +85 °C	PSDIP32
ST52E430K3D6	-40TO +85 °C	CSDIP32W
ST52X430/KIT		DEVELOPMENT KIT