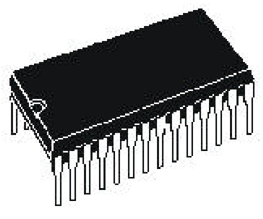
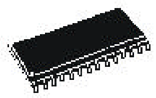
ST52T420/E420

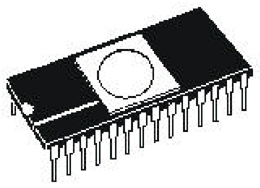
8-BIT *DuaLogic*Ô MCU WITH 4K BYTES OTP/EPROM, 128 BYTES RAM, WDG, ADC, 3 TIMER/PWM DRIVERS

PRELIMINARY DATA

- Digital Microcontroller with Fuzzy capabilities, 4 kbytes internal EPROM, 128 bytes of Data RAM
- On-chip 8 bit A/D Converter with an 8-channel multiplexer .
- 19 Configurable I/O PINs.
- Hardware multiplication and division.
- Two Programmable Timer/PWM with internal 16-bit Prescaler, featuring:
  - PWM output and Pulse generator mode
  - Complementary Outputs
- One Programmable Timer/PWM with internal 16-bit Prescaler, featuring:
  - 1 Input capture
  - 1 Output compare
  - External / Internal Clock
  - PWM output and Pulse generator mode
  - Complementary Outputs
- Watchdog timer.
- Capability to perform boolean, arithmetic operations and fuzzy algorithms.
- 46 basic instructions.
- Power Saving Features.
- Software tools and Emulators availability
- EPROM readout protection
- 28 pin Plastic Package


PDIP28


PSO28


CDIP28W

Features	ST52x420G0	ST52x420G1	ST52x420G2
Program Memory - bytes	1k	2k	4k
RAM - bytes	128		
PWM / TIMER	3		
ADC	8 channels / 8 bit		
Other Peripherals	Watchdog		
Operating Supply	3 to 5.5 V		
CPU frequency	up to 20 Mhz		
Temperature reange	-45 to +85 °C		
Packages	PDIP28 - PSO28 - CDIP28W		

GENERAL DESCRIPTION

ST52x420 is a member of the ST52xxx family of 8-bit *DuaLogic*Ô microcontrollers.

It is available in different versions, depending on memory size and peripheral.

ST52x420 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. It is produced by STMicroelectronics using the reliable high performance CMOSM6XE (O.5µm) process.

Thanks to Fuzzy Logic, ST52x420 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 complex systems with very high accuracy.

The linguistic approach is based on a set of IF-THEN rules, describing the control behaviour, 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 ST52x420 allows to interface with a wide range of external devices, like D/A converters or power control devices.

ST52x420 pins are configurable, allowing to set the input, or output, signals on each single pin, as shown in figures 1.3 and 1.4.

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 memory can be locked by user to avoid external operations.

It is possible to perform operations on data stored in the RAM (128 bytes), allowing to directly combine new inputs and feedback signals.

It is possible to store up to 341 Membership Functions, with triangular and trapezoidal shapes, or singleton values.

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

One of these programmable timers, with Internal Prescaler, can use internal or external START/STOP signals and clock.

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

In order to reduce the energy consumption, ST52x420 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 (MFs).

ST52x420 processes inputs and produces the related outputs according to the configuration loaded during the programming phase (stored into the EPROM).

ST52x420 includes an 8-bit Analog to Digital Converter with an 8-analog channel Multiplexer.

A powerful development environment consisting of board and software allows an easy configuration and use of ST52x420.

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.

ST52x420 is fully supported by FUZZYSTUDIO^{TM}4 allowing to graphically design a project and obtain an optimized microcode.

ST52x420 exploits a STMicroelectronics patented strategy to store the MFs in its internal memory.

FUNCTIONAL DESCRIPTION

ST52x420 works in two modes according to the control signal level.

ST52x420 is a programmable product and its functionment phases are:

- Memory Programming Phase
- Working Phase

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

- RESET
- TEST
- V_{PP}

Memory Programming Phase

ST52x420 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:

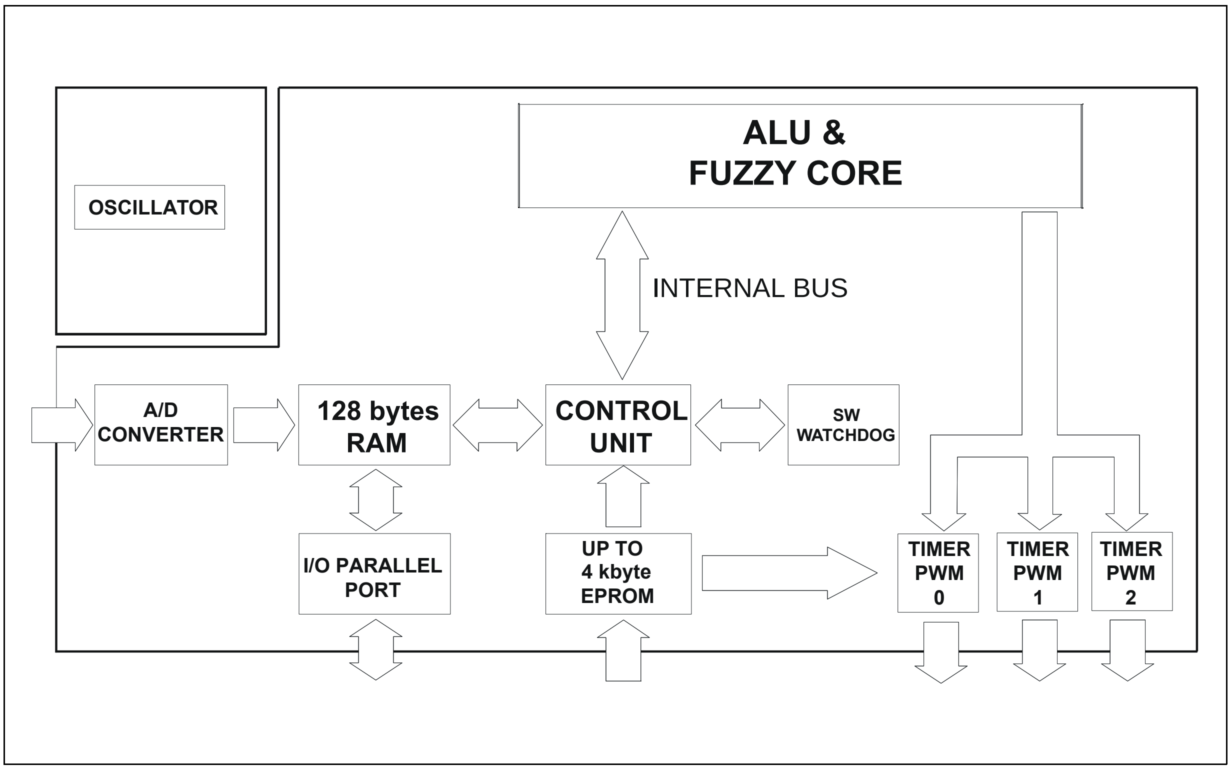
RESET=Vss

TEST=Vss

V_{PP}=12 V

When this phase starts, ST52x420 core is set to the RESET status. This allows to program and/or to test the internal Eprom. The signal INC_ADD is used to increment the address of the memory (see Eprom programming).

Figure 1.1 ST52x420 Architectural Block Diagram



Working Mode

In this mode the control signals are the following :

RESET=V_{DD}

TEST=Vss

V_{PP}=Vss

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

Figure 1.2 shows the internal structure of ST52x420. 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 fuzzy rules antecedent part. 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 4 kbytes of the Eprom are 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 128 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 ST52x420 Block Diagram

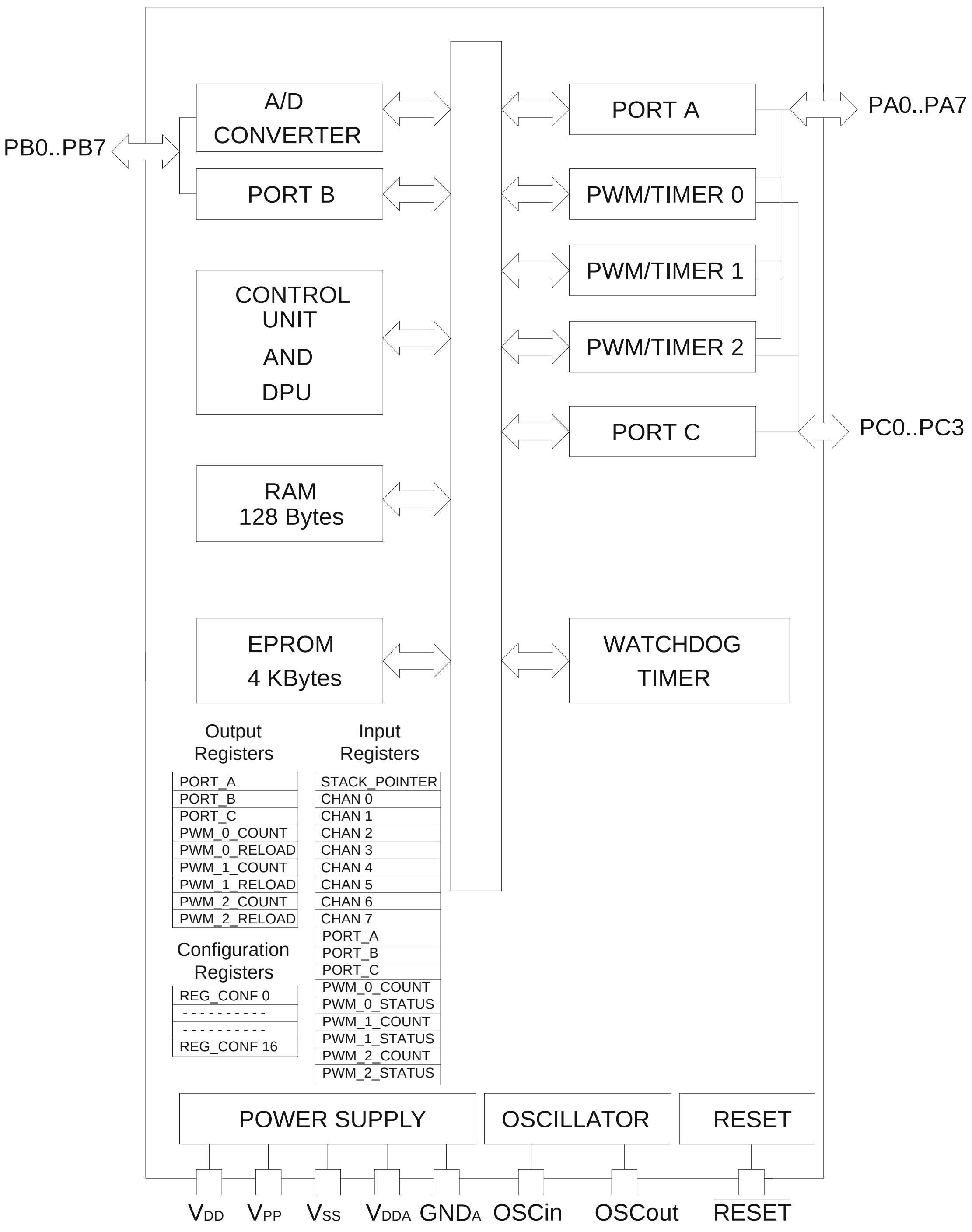


Figure 1.3a. ST52X420 SO28 Pin Configuration

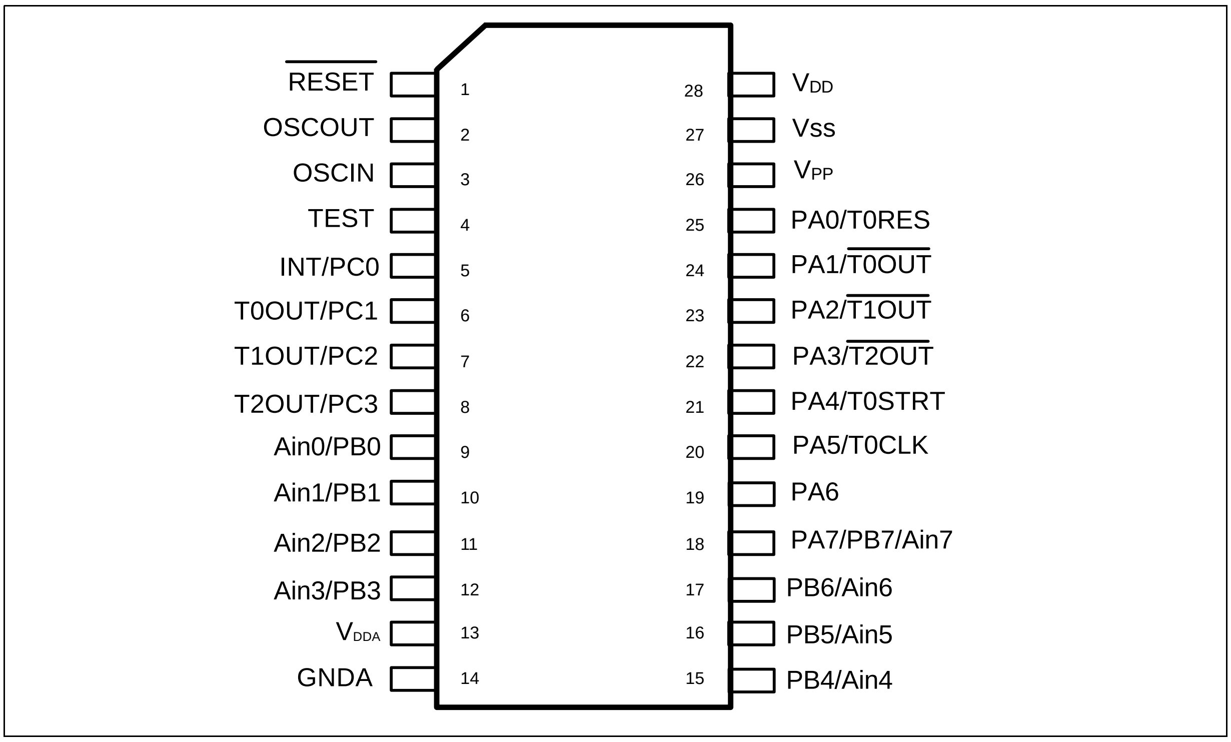


Figure 1.4a. ST52X420 DIP28 Pin Configuration

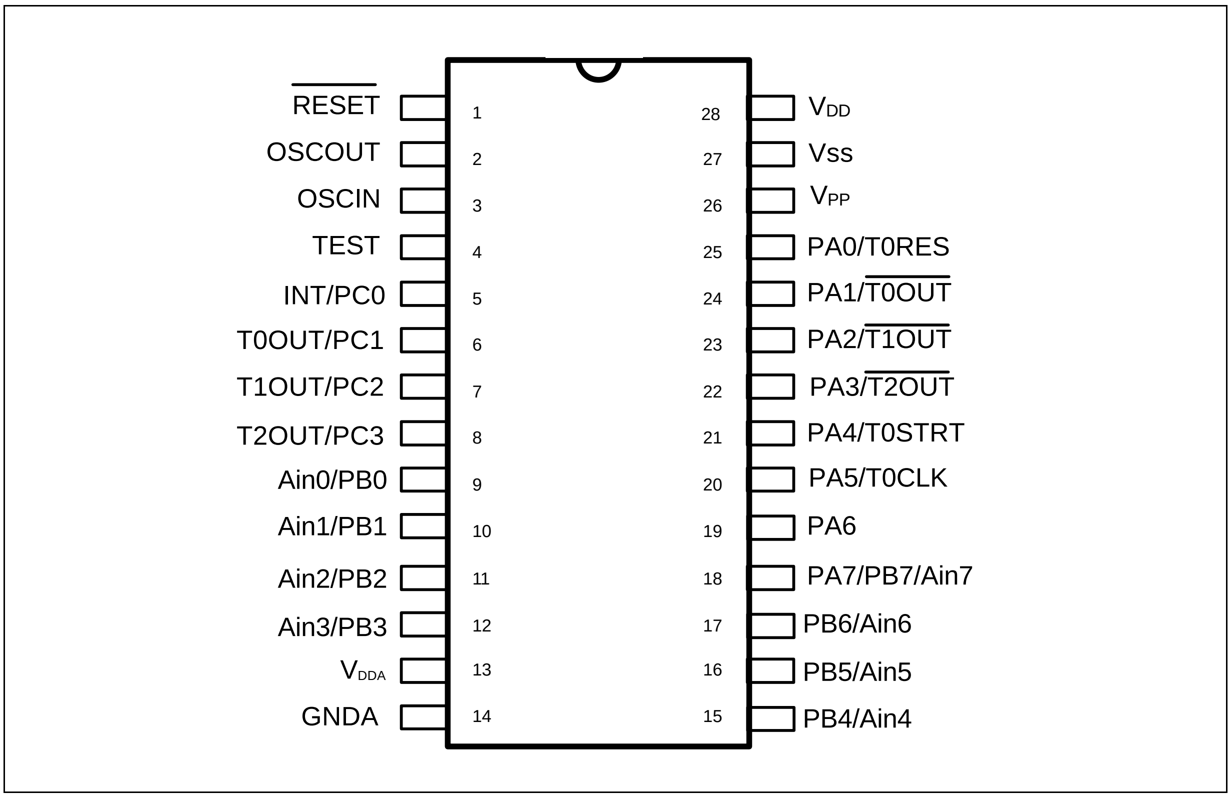


Table 1.1 ST52x420 SO-28 and DIP-28 Pin Configuration

PIN	NAME	TYPE	Programming Phase	Working Phase
1	$\overline{\text{RESET}}$	I	General Reset	General Reset
2	OSCOUT		Oscillator Output	Oscillator Output
3	OSCIN		Oscillator Input	Oscillator Input
4	TEST		Test Mode Selector	Test Mode Selector
5	INT / PC0	I/O	PHASE signal	External interrupt / Digital I/O
6	T0OUT / PC1	I/O		Timer/PWM 0 output / Digital I/O
7	T1OUT / PC2	I/O		Timer/PWM 1 output / Digital I/O
8	T2OUT / PC3	I/O		Timer/PWM 2 output / Digital I/O
9	Ain0 / PB0	I/O	Address RESET	Analog Input / Digital I/O
10	Ain1 / PB1	I/O	Address INCREMENT	Analog Input / Digital I/O
11	Ain2 / PB2	I/O	Configuration RESET	Analog Input / Digital I/O
12	Ain3 / PB3	I/O	Configuration INCREMENT	Analog Input / Digital I/O
13	V_{DDA}		Analog Power Supply	Analog Power Supply
14	GNDA		Analog Ground	Analog Ground
15	Ain4 / PB4	I/O		Analog Input / Digital I/O
16	Ain5 / PB5	I/O		Analog Input / Digital I/O
17	Ain6 / PB6	I/O		Analog Input / Digital I/O
18	Ain7 / PB7/ PA7	I/O	I/O EPROM Data	Analog Input / Digital I/O
19	PA6	I/O	I/O EPROM Data	Digital I/O
20	T0CLK / PA5	I/O	I/O EPROM Data	Timer/PWM 0 clock / Digital I/O
21	T0STRT / PA4	I/O	I/O EPROM Data	Timer/PWM 0 start/stop / Digital I/O
22	$\overline{\text{T2OUT}}$ / PA3	I/O	I/O EPROM Data	Timer/PWM 2 inverted output / Digital I/O
23	$\overline{\text{T1OUT}}$ / PA2	I/O	I/O EPROM Data	Timer/PWM 1 inverted output / Digital I/O
24	$\overline{\text{T0OUT}}$ / PA1	I/O	I/O EPROM Data	Timer/PWM 0 inverted output / Digital I/O
25	T0RES / PA0	I/O	I/O EPROM Data	Timer/PWM 0 Reset / Digital I/O
26	V_{PP}		EPROM Programming Power supply (12V ±5%)	EPROM V_{DD} or Vss
27	V_{SS}		Digital Ground	Digital Ground
28	V_{DD}		Digital Power Supply	Digital Power Supply

1.2 PIN DESCRIPTION

ST52x420 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.

In the ST52x410 version the two V_{DD} pins must be connected togheter .

V_{SS}. Digital Circuit Ground.

In the ST52x410 version the two V_{SS} pins must be connected togheter

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

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

V_{PP}. Main Power Supply for the internal EPROM. (12 V $\pm 5\%$ in programming phase) and Operating Modes 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 ST52x420 with various stability/cost trade-offs. An external clock signal can be applied to OSCin, in this case OSCout must be kept floating.

RESET. This signal is used to restart ST52x420 at the beginning of its program. It also allows to select the program mode for the EPROM.

Ain0-Ain7. 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-PC3. These lines are organized as I/O port. Each pin can be configured as input or 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.

TEST. It enables the testing functionalities; during the Programming and Working phase it must be set to V_{SS}.

INT. This pin is used to start the External Interrupt routine.

2 INTERNAL ARCHITECTURE

ST52x420 is made up by the following blocks and peripherals:

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

ST52x420 Operating Modes

ST52x420 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	Programming	Reset	Working
RESET	Vss	Vss	V_{DD}
TEST	Vss	Vss	Vss
Vpp	12 V	Vss	Vss

2.1 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 figures 2.1 and 2.2.

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 mantaining the code compatibility.

The CU reads the stored instructions on the EPROM (Fetch) and decodifies 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) up 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/DPU Block diagram

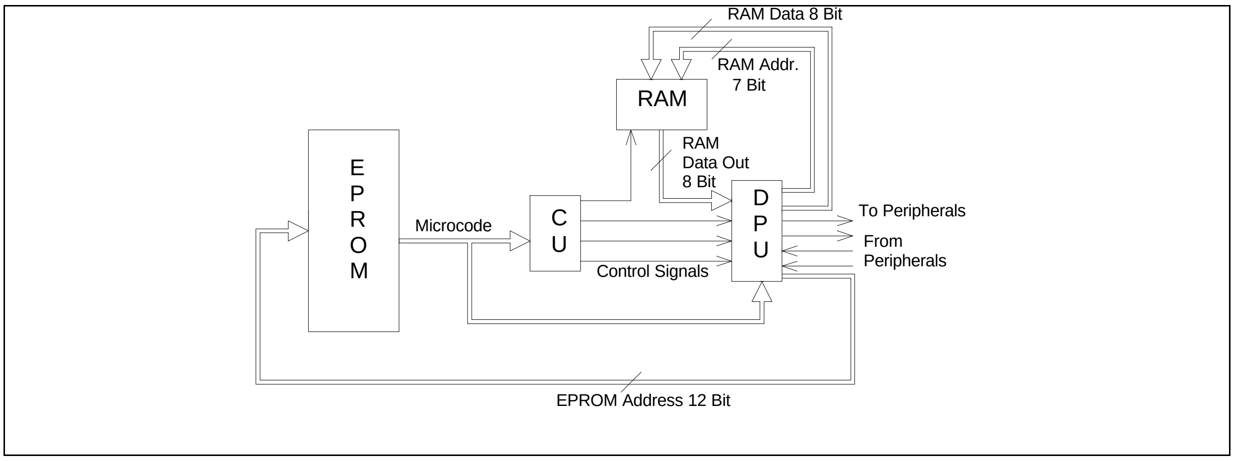


Figure 2.2 CU Block Diagram

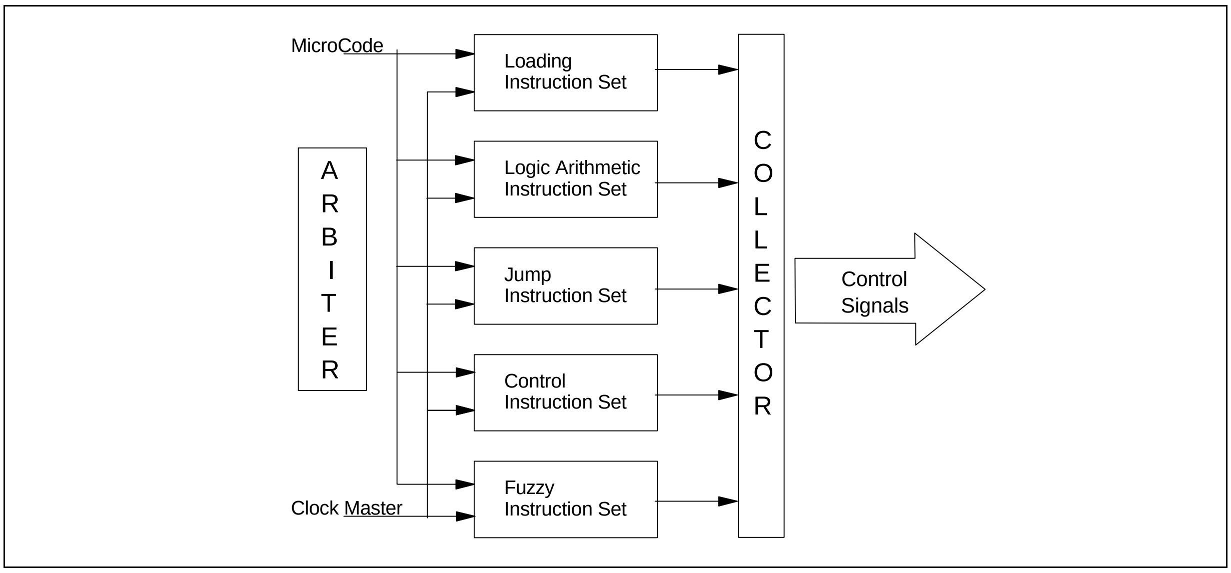


Figure 2.3 Data Processing Unit (DPU)

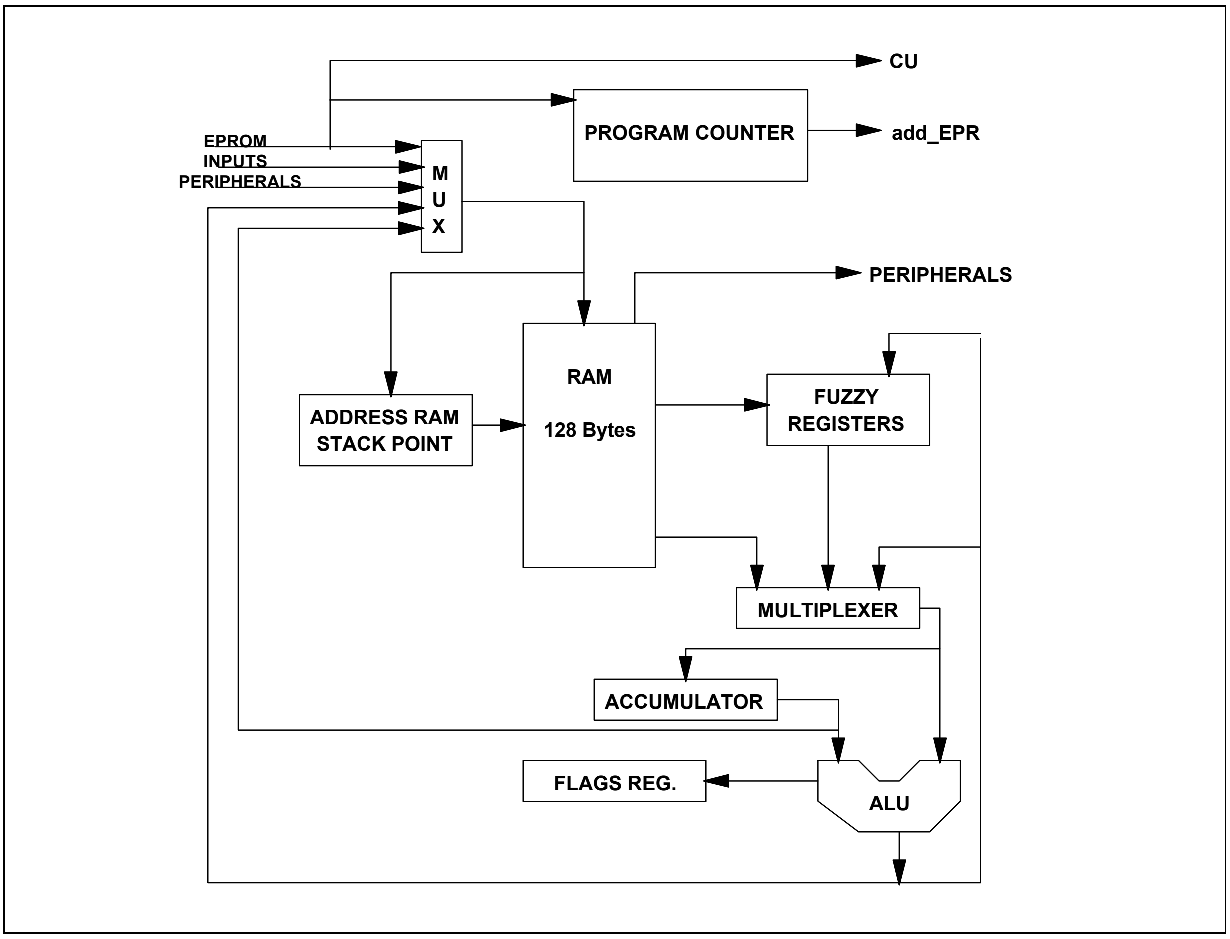
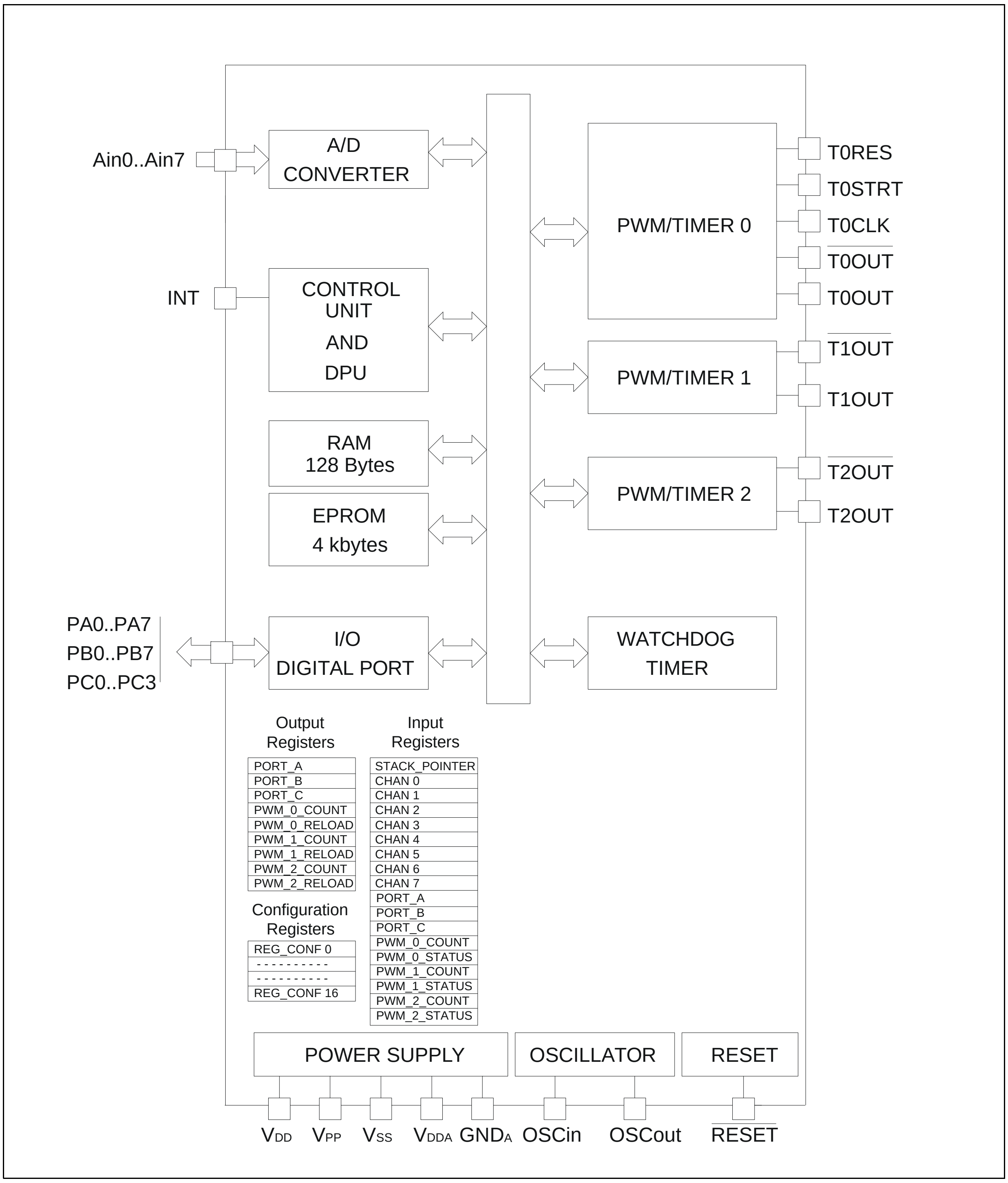


Figure 2.4 ST52x420 Peripherals Block Diagram



Note: ST52x410 version does not have the A/D Converter.

2.1.1 Program Counter

The Program Counter (PC) is a 12-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 12-bit length allows the direct addressing mode of 4096 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) instruction	PC = Jump Address
■ Interrupt	PC = Interrupt Vector
■ RETI instruction	PC = Pop (stack)
■ Reset	PC = Reset Vector
■ Normal Instruction	PC = PC + 1

2.1.2 Flags

The ST52x420 core includes different sets of flags. Each set of flags consist of a CARRY flag (C), a ZERO flag (Z) and SIGN flag (S).

Each interrupt level and the main level have their own set of flags, that are saved in the STACK together with the Program Counter.

If an interrupt occurs, the ST52x420 core uses the associated set of flags, and stores, in the STACK, the actual set in use .

These flags are restored from the STACK *automatically*, when a RETI instruction is executed.

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.

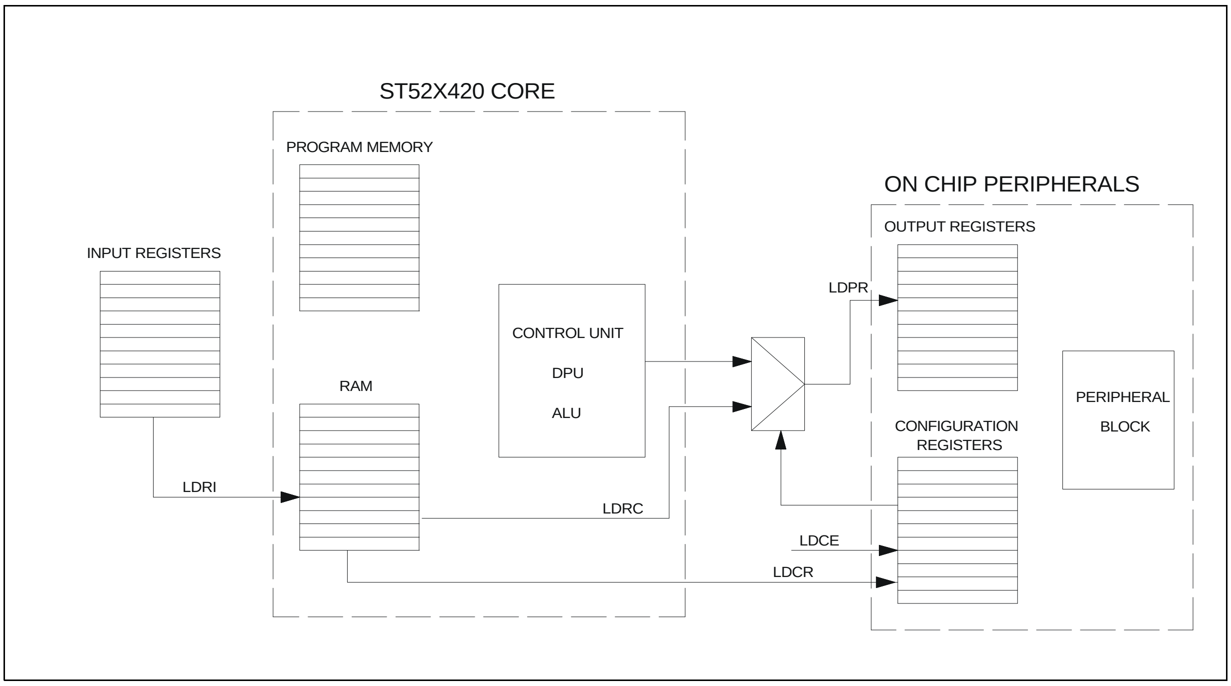
Note: *A CALL subroutine does not store in the STACK the current set of flags. For this reason a RET instruction, consequent to a CALL instruction, does not affect, the set of flags in use.*

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

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

Zero flag is set when a result equal to 0 occurs during arithmetic operations, otherwise it is cleared.

Figure 2.5 Address Spaces Description



2.2 ADDRESS SPACES

ST52x420 has five separate address spaces:

- RAM: 128 Bytes
- Input Registers: 18 8-bit registers
- Output Registers 9 8-bit registers
- Configuration Registers: 17 8-bit registers
- Program memory up to 4K Bytes

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

2.2.1 RAM and STACK

The RAM memory consists of 128 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.

A RAM register is read in 2 cycles and is written in 3 cycles. 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.

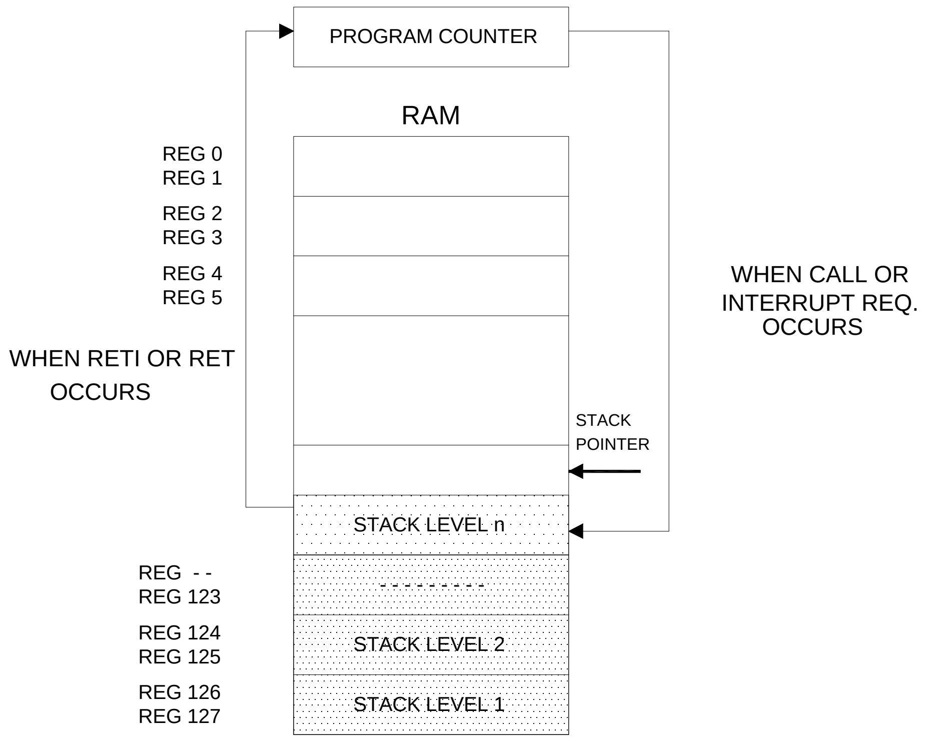
When the instructions like Interrupt request or CALL are executed, a STACK level is used to push the PC and flags.

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 127). **The maximum level of stack must be less than 64**.

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 subroutine or interrupt return occurs (RET or RETI instructions), the data stored in the highest stack level are restored back into the PC and current 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



2.2.2 Input Registers Bench

The Input Registers (IR) bench consists of 18 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, STACK_POINTER, 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 summarises 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	PERIPHERALS	
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

The input register PORT_C contains 4 bits.

2.2.3 Configuration Registers

The ST52x420 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.

Table 2.3. Configuration Registers description

CONFIGURATION REGISTER	PERIPHERAL	DESCRIPTION
REG_CONF 0	INTERRUPT MASK	Interrupts mask setting
REG_CONF 1	INTERRUPT PRIORITY	Interrupts priority setting
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 complementary 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

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

2.2.4 Output Registers

The Output Registers (OR) consist of 9 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 name is used in FUZZYSTUDIO^{TM} 4 development tools.

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

Table 2.4 Output Registers

OR MENMONIC NAME	PERIPHERAL	ADDRESS
PORT_A	PORT A OR	0
PORT_B	PORT B OR	1
PORT_C	PORT C OR	2
PWM_O_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

2.3 FUZZY CAPABILITIES

ST52x420 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 (MFs) for each Input;
- Up to 128 Outputs with 8-bit resolution;
- Possibility to process fuzzy rules with an high number of antecedents
- High number of Rules and Fuzzy Blocks.

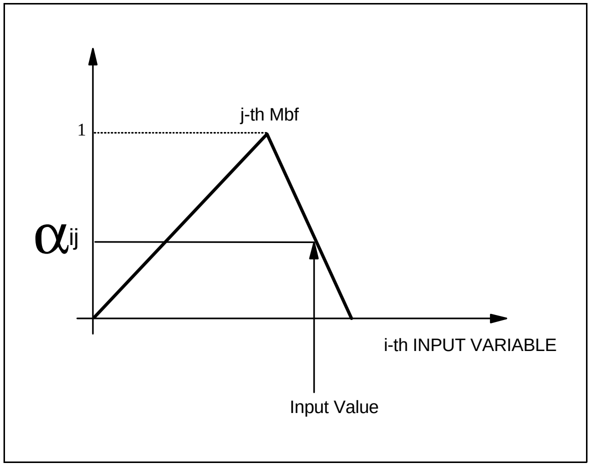
The limits on the number of Fuzzy Rules and Fuzzy Blocks are only related with the program memory size.

2.3.1 Fuzzy Inference

The block diagram shown in figure 2.8 describes the different steps performed during a fuzzy algorithm. ST52x420 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 Weigth calculation



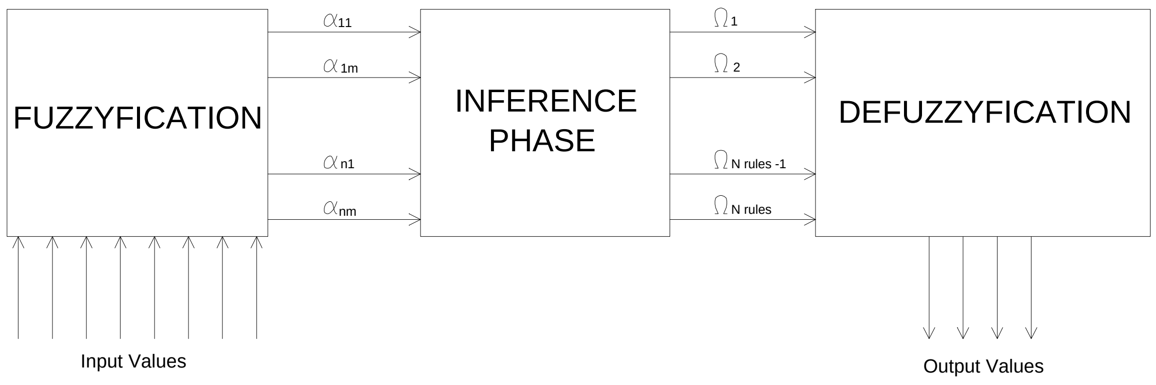
2.3.2 Fuzzyfication Phase

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

8 Fuzzy input registers are available for the fuzzy inferences.

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

Figure 2.8. Fuzzy Inference



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

2.3.3 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.

It is possibile 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.

2.3.4 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.

Figure 2.9. Fuzzyfication

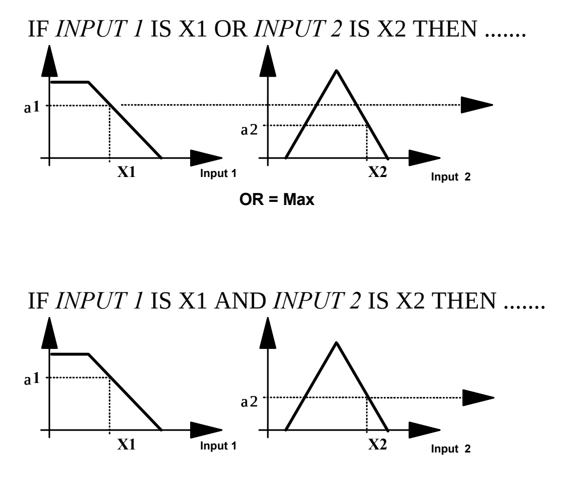
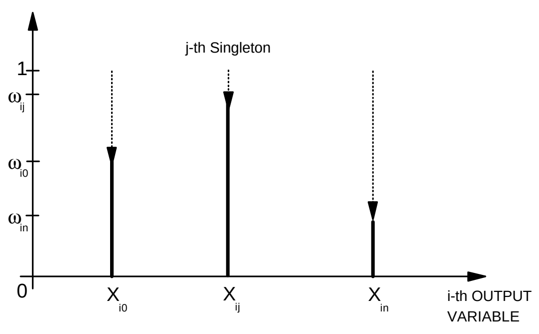

IF INPUT 1 IS X1 OR INPUT 2 IS X2 THEN ....
a1
X1
Input 1
a2
X2
Input 2
OR = Max
IF INPUT 1 IS X1 AND INPUT 2 IS X2 THEN ....
a1
X1
Input 1
a2
X2
Input 2

Figure 2.10 Output Membership Functions.



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 = identifies the current output variable

N = number of the active rules on the current output

ω_{ij} =weigth 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.

2.3.5 Input Membership Function

ST52x420 allows to manage triangular MFs. In order to define a MF it is necessary to store three different data on the program memory:

the vertex of the MF: **V**;

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

the lenght 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 MFs that can be defined in ST52x420.

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

The MF is memorized by using the following instruction:

MF n_MF lvd v rvd

where

n_MF identifies the MF, lvd, v, rvd are the parameters describing the MF's shape. In a trapezoidal MF LVD, or RVD, is 0.

2.3.6 Output Singleton

ST52x420 uses for the output variables a particular kind of membership function called Singleton. A Singleton has not a shape, like a traditional MF, and it is characterized by a single point identified by the couple (X, ω), where the ω is calculated by the Inference Unit as described before.

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

Figure 2.11. MFs Parameters

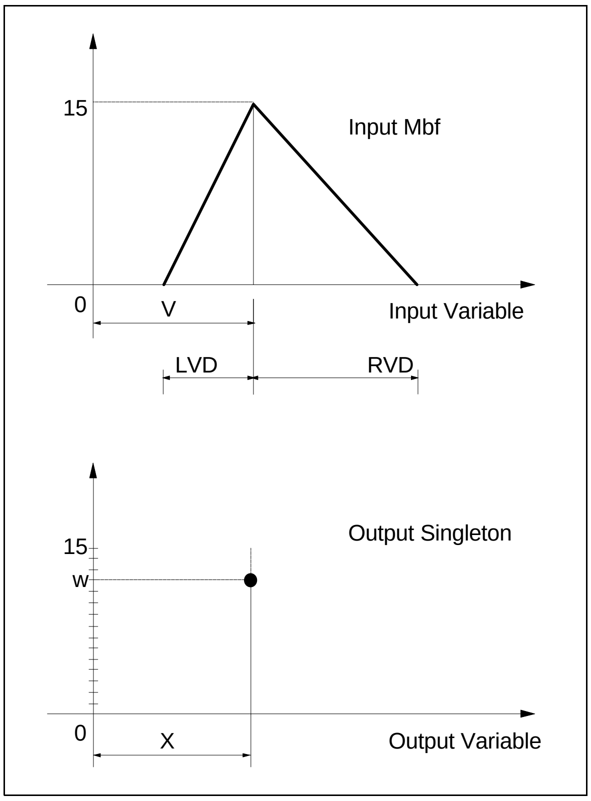
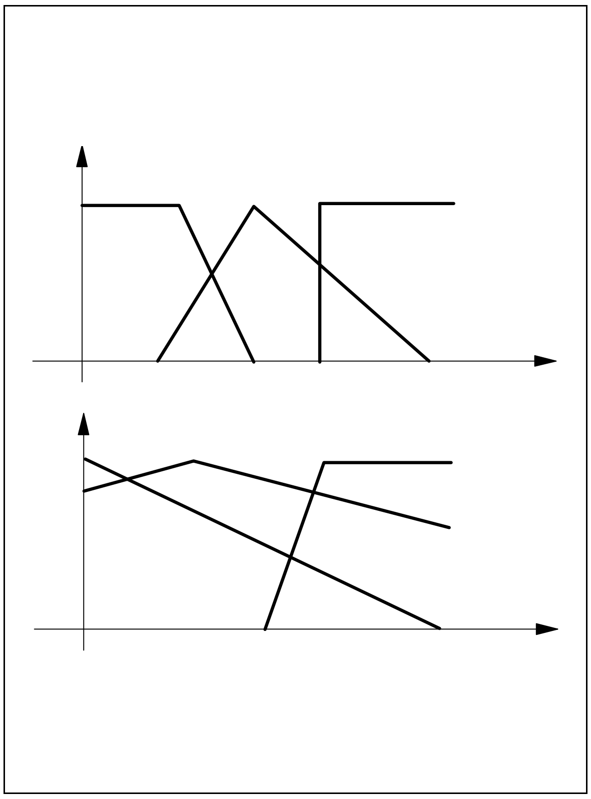


Figure 2.12. Example of valid MFs



2.3.7 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.

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

Table 2.5. Fuzzy Instructions Set

Instruction	Description
MBF n_MF lvd v rvd	Stores the MF n_MF with the shape identified by the parameters lvd, v and rvd.
LDP n m	Fixes the alpha value of the input n with the MF m and stores it in internal registers.
LDN n m	Calculates the negated alpha value of the input n with the MF m and stores 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 the data stack.
SKM	Stores the result of the last fuzzy operation executed in internal registers.
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 single fuzzy output calculation.

The Assembler instructions operate as in the following example:

Let us suppose you have previously defined the MF with MBF instructions, the rule:

IF Inp0 is NOT MF01 AND Inp2 is MF21 OR Inp3 is MF33 THEN CRISP1

is therefore codified as:

LDN 0 1	Loads in the stack the NOT value relative to the first term of the rule (supposing that MF_num for MF01 is 1).
LDP 2 21	Loads in the stack the value relative to the second term of the rule(supposing that MF_num for MF21 is 21) .
FZAND	Calculates the min between two values in the stack.
LDK	Stores the result of the previous operation in the stack.
LDP 3 33	Loads in the stack the value relative to the third term of the rule (supposing that MF_num for MF33 is 33).
FZOR	Calculates the max between the two values in the stack.
CON 58	Performs the product between the values calculated and the value CRISP1 = 58 (consequent calculus)

Let us suppose now you have the following rule:

IF (Inp2 is MF21 AND Inp3 is NOT MF35) OR (Inp0 is MF03 OR Inp1 is NOT MF16) THEN CRISP2

It is codified with the following instructions:

LDP 2 21	Loads in the stack the value relative to the first term of the rule (supposing that MF_num for MF21 is 21).
LDN 3 35	Loads in the stack the NOT value relative to the second term of the rule (supposing that MF_num for MF35 is 35).
FZAND	Calculates the min between the two values in the stack.
SKM	Stores the calculated value on the temporary register.
LDP 0 3	Loads in the stack the value relative to the third term of the rule (supposing that MF_num for MF03 is 3).
LDN 1 16	Loads in the stack the NOT value relative to the fourth term of the rule (supposing that MF_num for MF16 is 16).
FZOR	Calculates the max between the two values in the stack.
LDK	Stores the result of the previous operation in the stack.
LDM	Copies the content of the temporary register in the stack.
FZOR	Calculates the max between the two values in the stack
CON 35	Performs the product between the value calculated and the value CRISP1=35 (Consequent calculus).

After the inference of all the rules relative to an output, you can obtain the output through the instruction:

OUT 0	To calculate the first fuzzy output.

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 per, RAM	3	14	-	-	-
LDRC	LDRC RAM, Const	3	14	-	-	-
LDRE	LDRE RAMi, Eprom	3	16	-	-	-
(LDRE)	LDRE (RAMi), (EPROMj)	3	18	-	-	-
LDRI	LDRI RAM i, inp_reg	3	15	-	-	-
LDRR	LDRR RAM i RAM j	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	7/16	-	-	-
MEGI	MEGI	1	7/16	-	-	-
MDGI	MDGI	1	7/16	-	-	-
RETI	RETI	1	12	-	-	-
RINT	RINT INT	2	8	-	-	-
UDGI	UDGI	1	7/16	-	-	-
UEGI	UEGI	1	7/16	-	-	-
WAITI	WAITI	1	7	-	-	-

Control instructions 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, IS stacked value restored, - not affected

2.4 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 ST52x420 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 ST52x420 is able to perform **multiplication** (MULT) and **division** (DIV), by means of an hardware multiplier and an hardware divider. In this way it is possible to increase the computational throughput and reduce code size requirement for complex algorithms. The multiplication is performed by using 8 bit operands storing the result in 2 registers (16 bit values), see Figure 2.13. 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)

2.5 ST52x420 Assembler Pseudo Instructions

The ST52x420 assembler instructions set includes some pseudo instructions.

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.

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.

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

For more details see the chapter related to the instructions set.

Figure 2.13 Multiplication

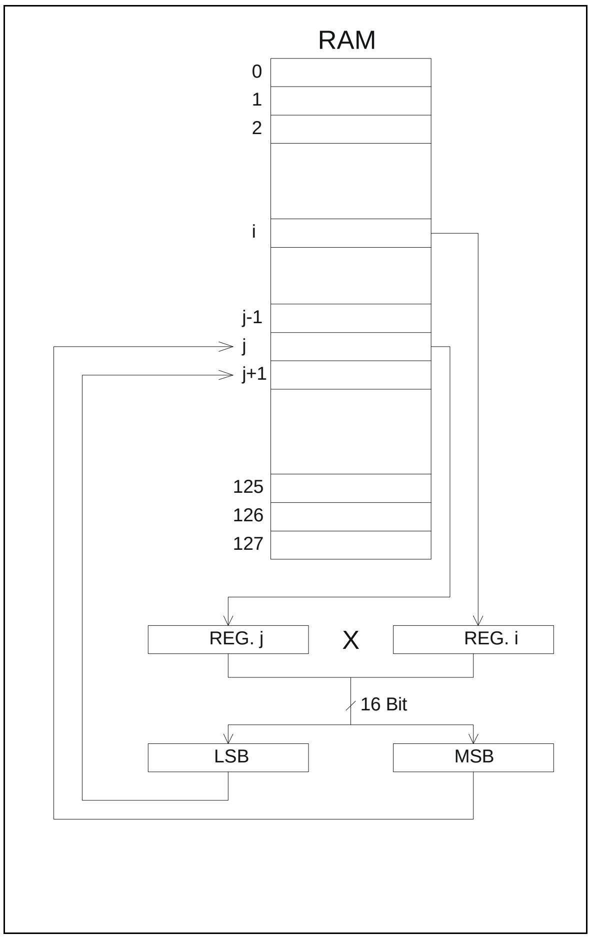
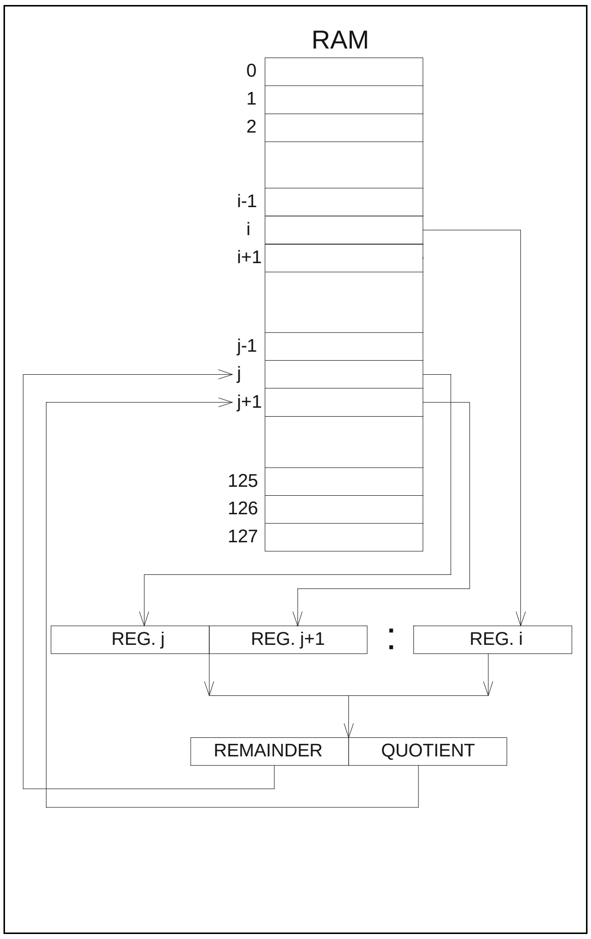


Figure 2.14 Division



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 32 kbits of memory space with an 8-bit internal parallelism (4 kbytes) addressed by an 12-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 ST52x420 EPROM memory is divided into three main blocks (see Figure 3.1):

- *Interrupt Vectors memory block* (3 through 17) contains the jump instructions to the addresses for the interrupt routines. Each interrupt vector is composed of three bytes.
- *Mbfs Setting memory block* (18 through 1023) contains the coordinates of the vertexes of every Mbf defined in the program. The memory space of this block, not used to contain the Mbfs settings, is available for the Program instructions set.
- *The Program Instruction Set memory block* (1024 through 4095) contains the instruction set of the user program.

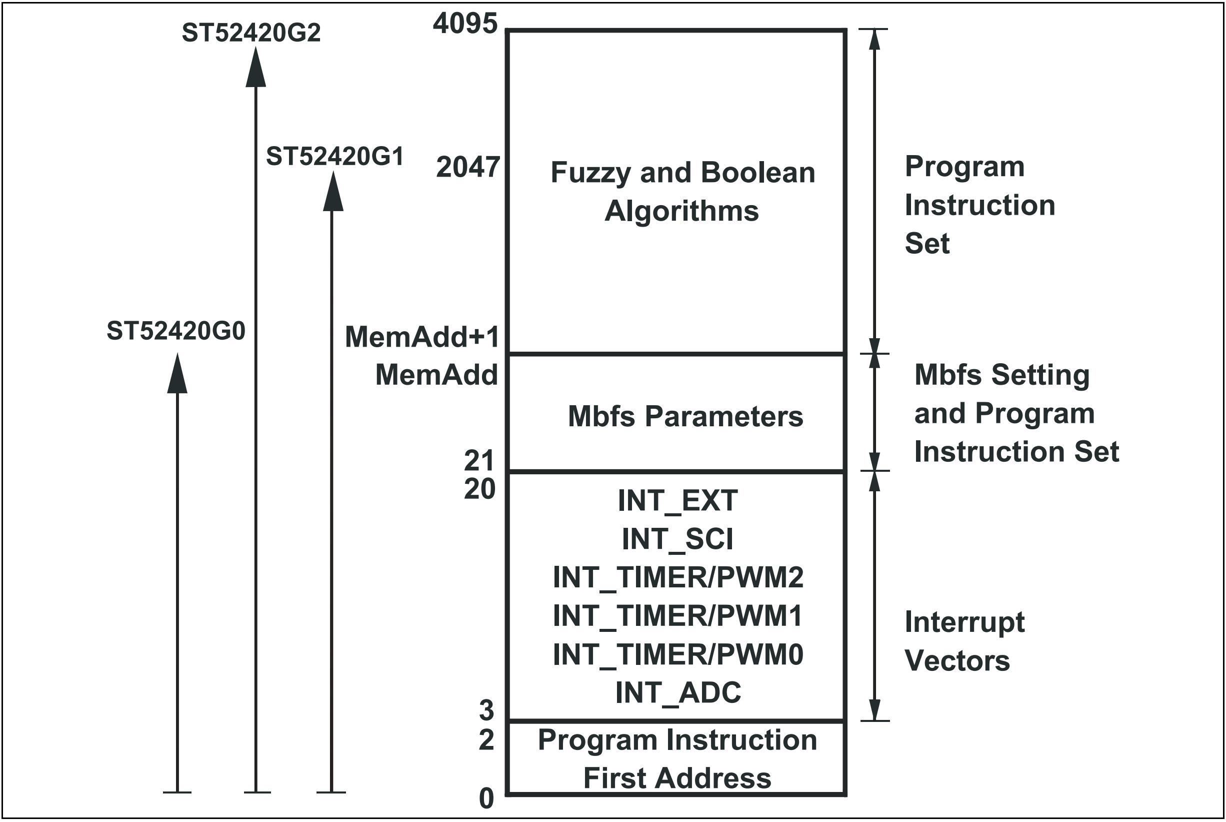
The locations 0, 1 and 2 contain the address of the first microcode instruction.

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

Above operations are managed by using an internal 4-bit configuration register and an EPROM Configuration Register. Depending on the value written in such register, the corresponding operation is performed. The reading phase is executed with V_{PP}= 5V±5%, while the verify 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.

Figure 3.1. Memory Map



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 follows:

RESET (pin 1) =Vss

TEST (pin 4) =Vss

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

RST_ADD (pin 9), INC_ADD (pin 10), RST_CONF (pin 11), INC_CONF (pin 12) and PHASE (pin 5) 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.

Data in/out digital port is PORT A (from pin 18 up to pin 25).

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 means 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.

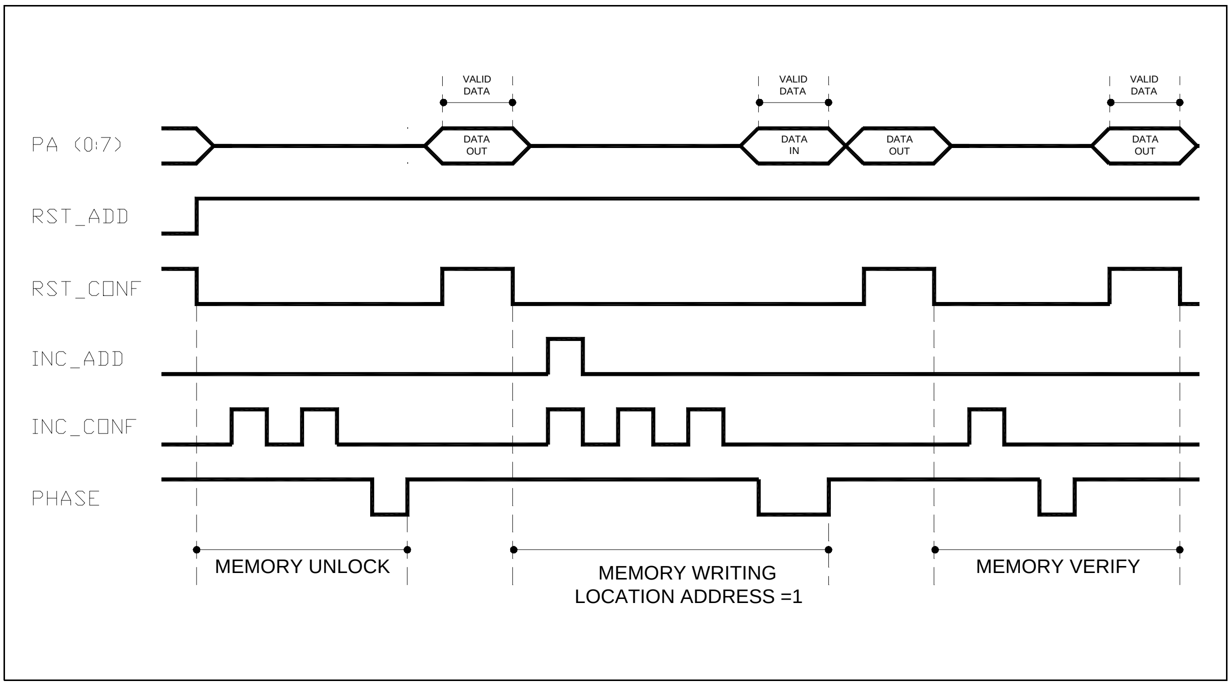
INC_ADD signal increments the memory address register value, that is the address of the EPROM memory cell that will be affected by the operation in progress.

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. When PHASE signal is low valid data have to be present in the PORT A.

Figure 3.2. EPROM Programming Timing



3.1.1 EPROM Operation

To execute one EPROM operation, the corresponding identification value must be loaded in the EPROM Configuration Register (see table 3.1). The signal timing is the following:

- -- RST_ADD= high and PHASE= high,
- -- a positive pulse on RST_CONF signal reset the EPROM Configuration Register,
- -- INC_CONF signal generates a number of positive pulses equal to the value to be loaded in the EPROM Configuration Register.
- -- After this sequence, a negative pulse of the PHASE signal will validate the selected operation. During Writing operations valid data have to be present in the Port A when PHASE signal is low.

The minimum PHASE signal pulse width must be 10 µs for the EPROM Writing Operation and 100 ns for the others.

If the EPROM Configuration Register has not been reseted, every INC_CONF signal impulse increases the existing value.

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.

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

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 (see table 3.1).

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 (see table 3.1),

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

7) valid data have to be present in the PORT A (this step can be performed before),

8) a negative pulse (10 µs) on the PHASE signal validates the Memory Writing operation and the memory location is written.

For sequential Writing operations, it is necessary to repeat the steps since 6 to 7.

3.1.4 EPROM Reading / Verify

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 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 loads 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) pins (Port A) the value stored in the actual memory address and resets the EPROM Configuration Register.

Then, if any error in writing occured, 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 10 mA (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 (see paragraph 3.1.1).

3.2 Eprom Erasure

Thanks to the transparent window available in the CDIP28W 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 2 and 17.

The Interrupt routine is performed as a normal code checking, at the end of each instruction, if an 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 rules, 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 doesn't have to stop a Fuzzy function, that is running in another interrupt routine. For this reason, to use a fuzzy function inside an interrupt routine, the user has to include the fuzzy function between an UDGI (MDGI) instruction and an 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 (GIR) will be generated and Interrupt Service

Figure 4.1. Interrupt Flow

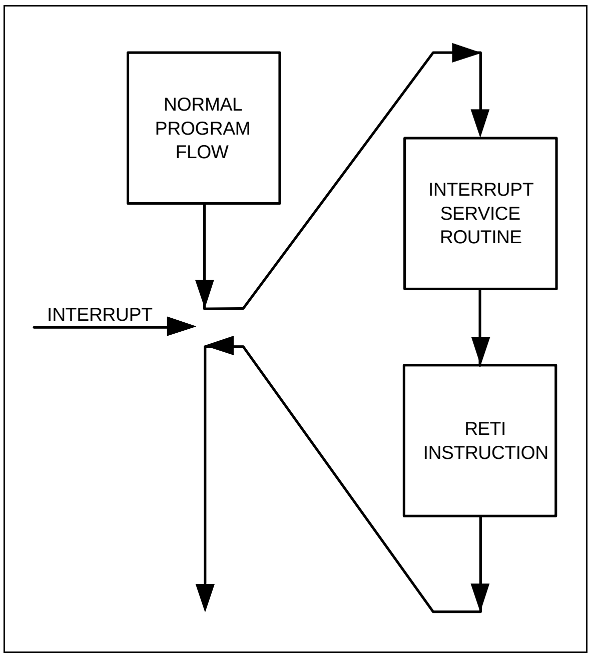


Figure 4.2. Interrupt Vectors Mapping

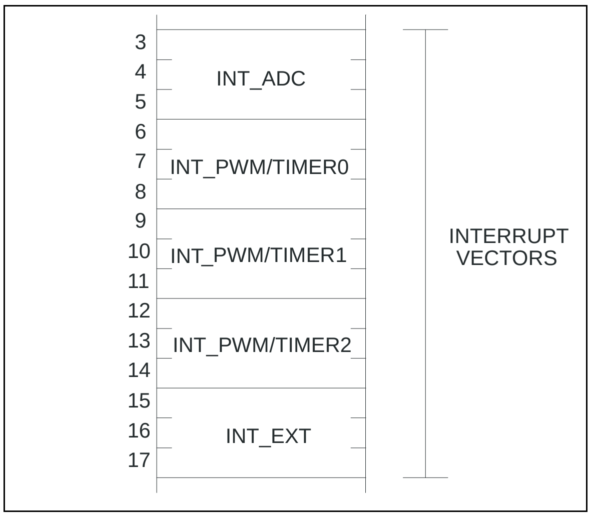
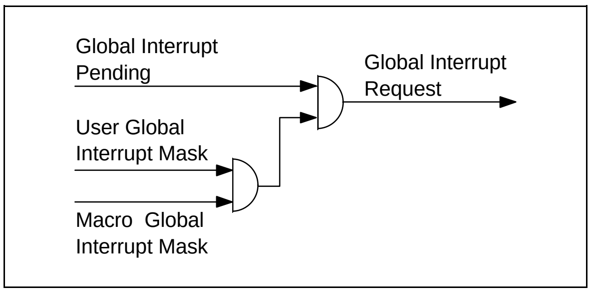


Figure 4.3. Global Interrupt Request



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

ST52x420 manages interrupt signals generated by the internal peripherals (PWM/Timers and Analog to Digital Converter) or coming from the INT/PC0 pin. The External Interrupt is active on high level of INT/PC0 signal.

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 is kept pendent.

For example:

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

```
LDCR 0, 10 // sets 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

Table 4.1. Configuration Register 0

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 0 Interrupt 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	not used	-	
6	not used	-	
7	not used	0	
		1	

Reset Configuration ’00000’

Table 4.2. Interrupts Description

Name	Description		Priority	Peripheral Code	Maskable	EPROM Locations
INT_ADC	ADC	Int	Programmable	00	yes	3-5
INT_PWM/TIMER0	PWM/TIMER 0	Int	Programmable	01	yes	6-8
INT_PWM/TIMER1	PWM/TIMER 1	Int	Programmable	10	yes	9-11
INT_PWM/TIMER2	PWM/TIMER 2	Int	Programmable	11	yes	12-14
INT_EXT	External Interrupt (INT)	Ext	Highest	-	yes	15-17

Figure 4.4. Interrupt Configuration Register 0

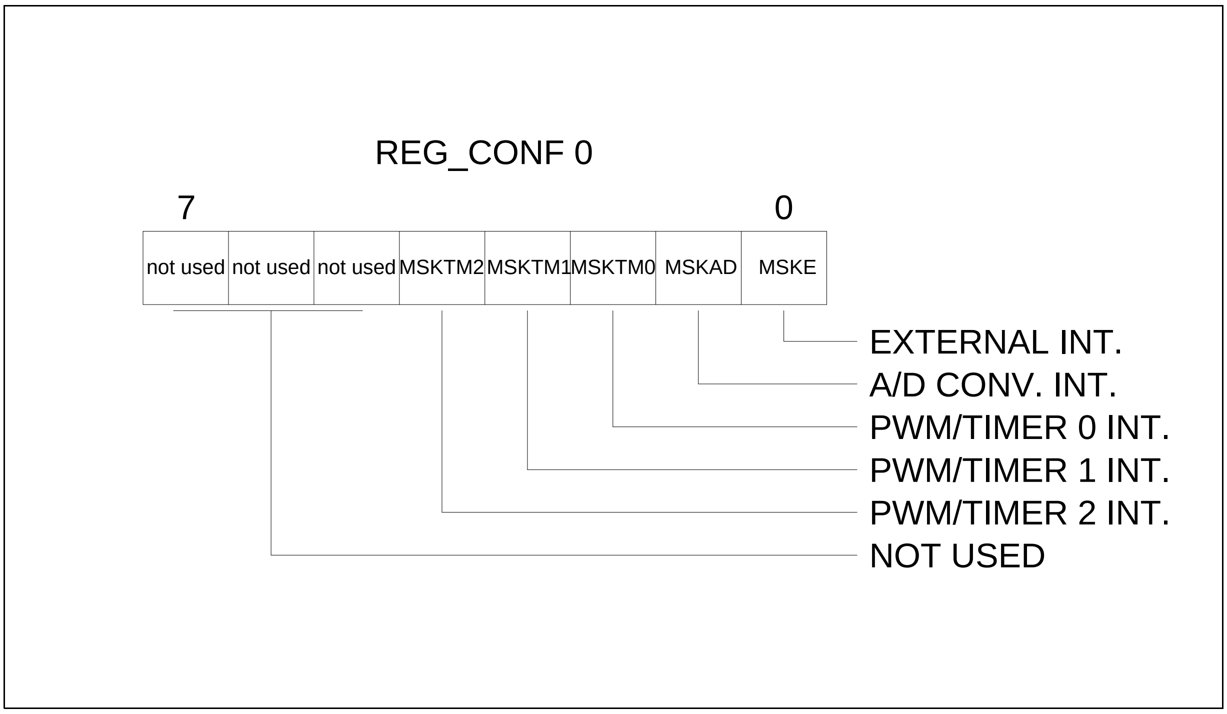
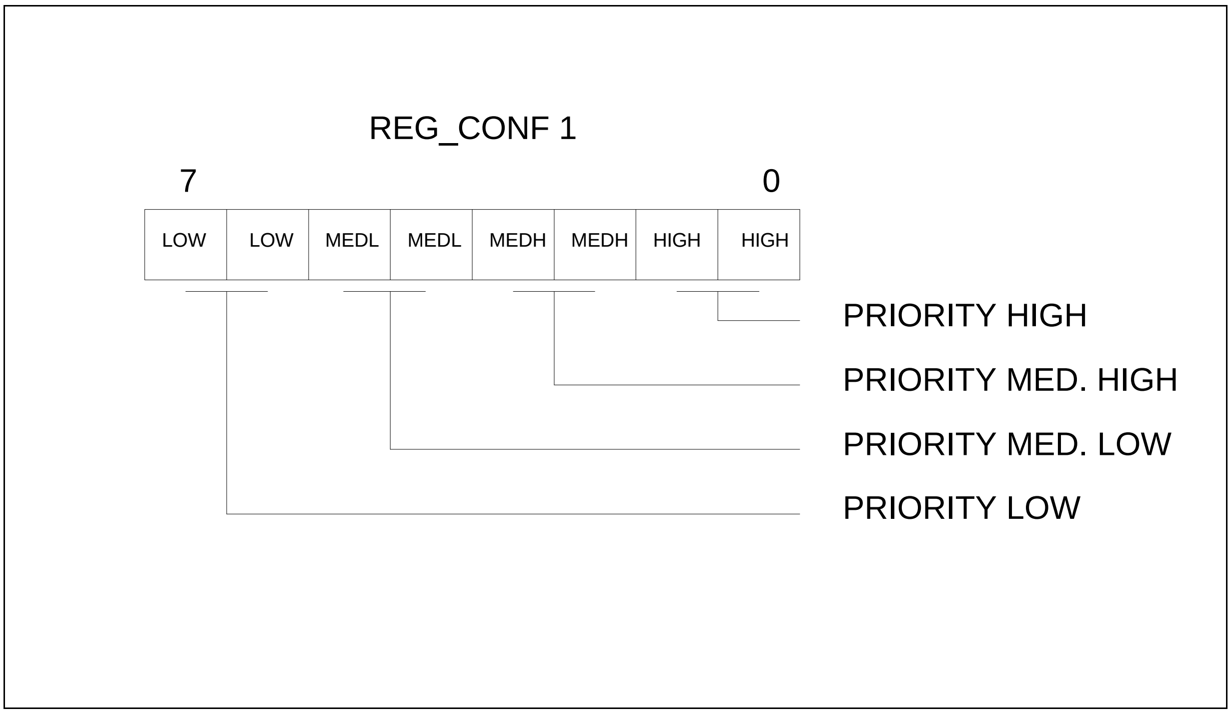


Figure 4.5. Interrupt Configuration Register 1



(INT_ADC) and from the PWM/TIMER 0 (INT_PWM/TIMER0).

4.5 Interrupt Priority

Six priority levels are available: level 5 has the lowest priority, level 0 has the highest priority.

Level 5 is associated to the Main Program, levels 4 to 1 are programmable by means of the priority register called REG_CONF1 (see fig.4.5 and table 4.3); whereas the higher level is related to the external interrupt (INT_EXT).

PWM/Timers and ADC are identified by a two-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 PRI*i* priority level.

i.e.

```
LDRC 10, 201 //(loads the value
201='11001001' in the RAM Register
10)
```

```
LDCR 1, 10 // sets the REG_CONF1=
'11001001'
```

thus defining the following priority levels:

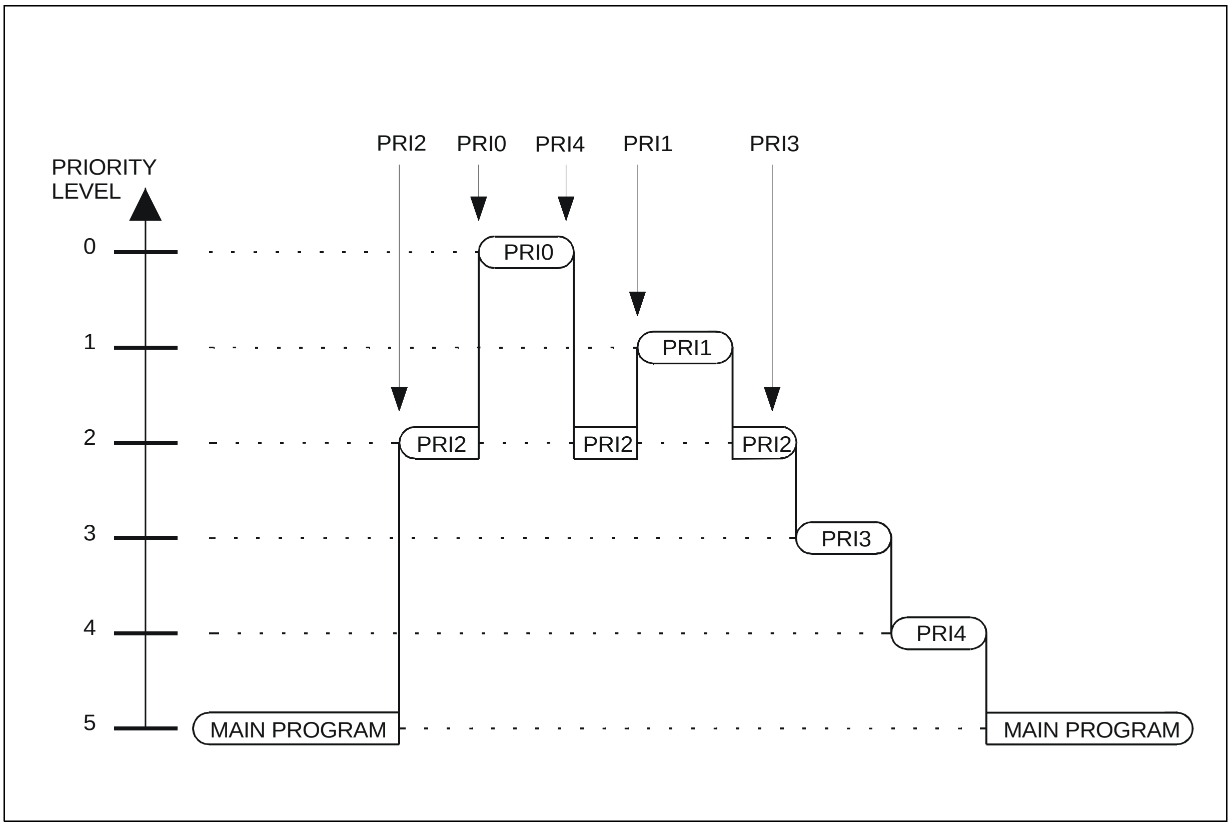
Table 4.3. Configuration Register 1

Bit	Name	Value	Priority Level
0, 1	PRI1	Peripheral Code	High
2, 3	PRI2	Peripheral Code	Medium-High
4, 5	PRI3	Peripheral Code	Medium-Low

- Level 1: INT_PWM/TIMER0(PWM/TIMER 0 Code: 01)
- Level 2: INT_PWM/TIMER0(PWM/TIMER 1 Code: 10)
- Level 3: INT_ADC(ADC Code: 00)
- Level 4: INT_PWM/TIMER0(PWM/TIMER 2 Code: 11)

REMARK: the Interrupt priority must be fixed ad the beginning of the main program, because at the RESET REG_CONF1='00000000', and this

Figure 4.6. Example of a Sequence of Interrupt Requests



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_CONF1 value to modify the interrupt priority

step 3):

reset by mean of RINT instructions all the pending interrupt routines

step 4):

unmask the interrupts by mean 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.

REMARK: when an interrupt is masked, it is excluded from the priority arbitration. This may cause the following side-effect: if an interrupt is masked during the servicing of his own service routine, this may be interrupted by lower priority sources. To avoid any problem you can mask the interrupt just at the end of the routine.

4.6 Interrupts and Low power mode

All not masked interrupts allow the processor to leave the WAIT low power mode. External interrupt and ext. RESET allow the processor to leave the HALT low power mode.

Table 4.4. RINT instruction code

Name	Description	Value
INT_ADC	ADC	0
INT_PWM/TIMER0	PWM/TIMER 0	1
INT_PWM/TIMER1	PWM/TIMER 1	2
INT_PWM/TIMER2	PWM/TIMER 2	3
INT_EXT	External Interrupt	4

4.7 Interrupt RESET

An eventually pending interrupt can be reset with the instruction `RINT j`, which resets the interrupt of the peripheral j according to the following table (see table 4.4)

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 Clock System

The ST52x420 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 ST52x420 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**.

The system clock may be generated by using either a quartz crystal, or a ceramic resonator (CERALOC); or, at least, by means of an external clock.

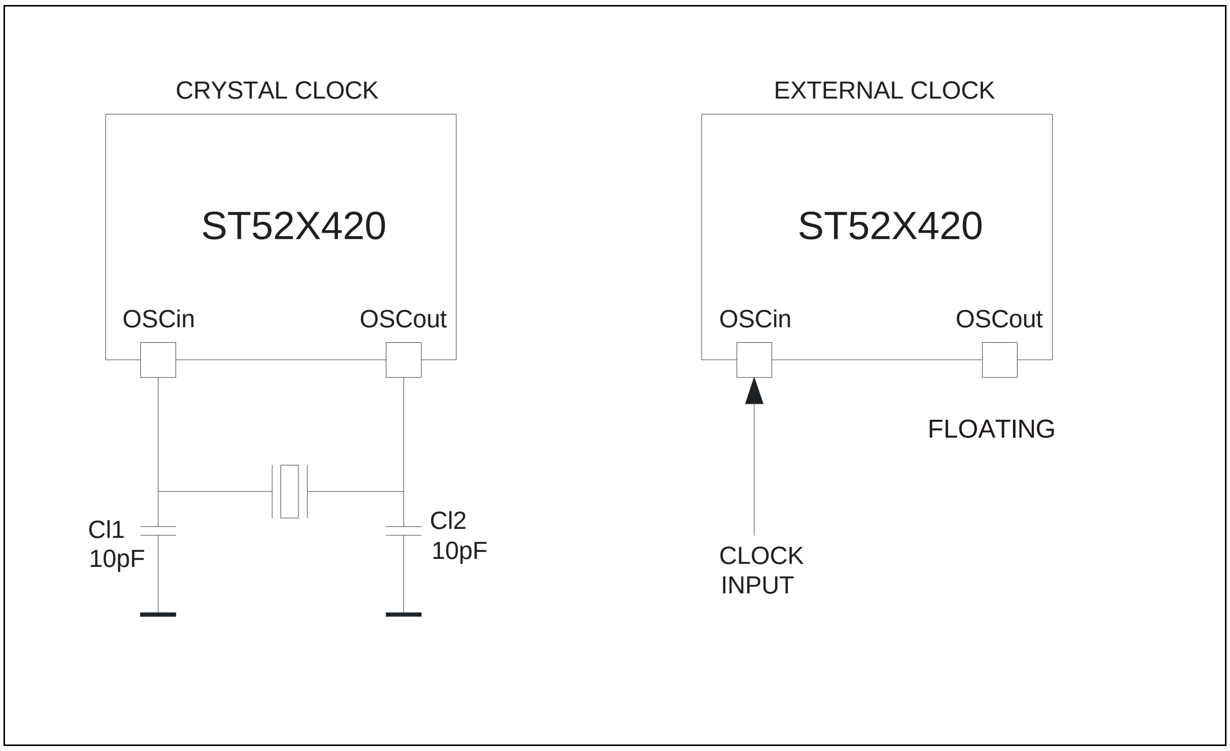
The different clock generator options connection methods are shown in Figure 5.1.

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

The crystal oscillator start-up time is a function of many variables: crystal parameters (especially R_S), oscillator load capacitance (CL), IC parameters, enviroment 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 recognised. 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 V_{DD} (see Table 2.1)

Figure 5.2 Reset Block Diagram

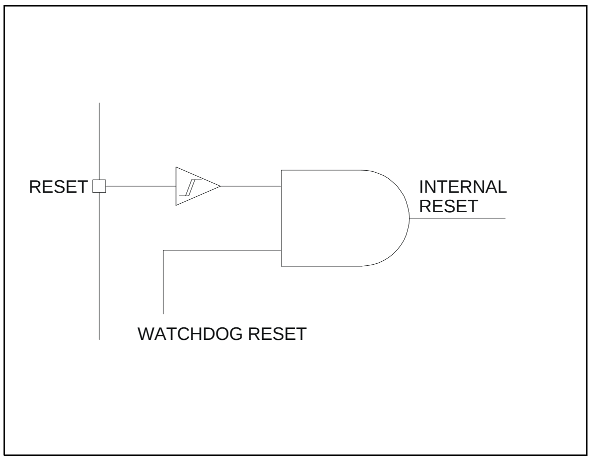
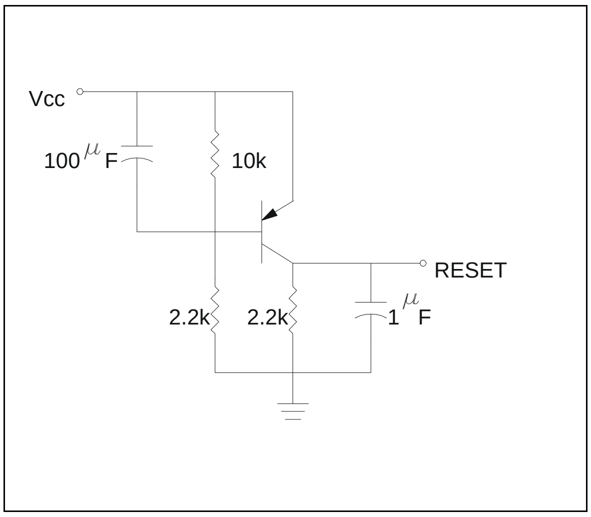


Figure 5.4 Simple Reset Circuit



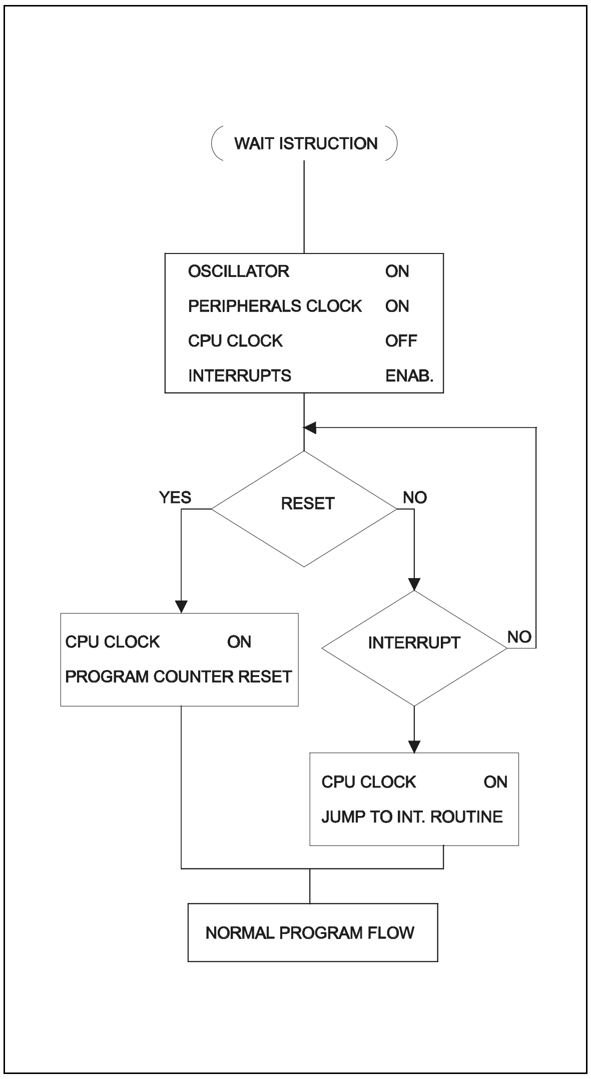
5.3 Power Saving Mode

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 remain active. During the WAIT mode, the unmasked 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.** The HALT instruction will be skipped, if it is executed while the watchdog system is enabled.

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 stabilisation time is provided before restarting the CPU operations. The stabilisation time is 4096 CPU clock cycles.

Wake-Up from HALT mode

After the start up delay, the CPU restarts the operations.

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

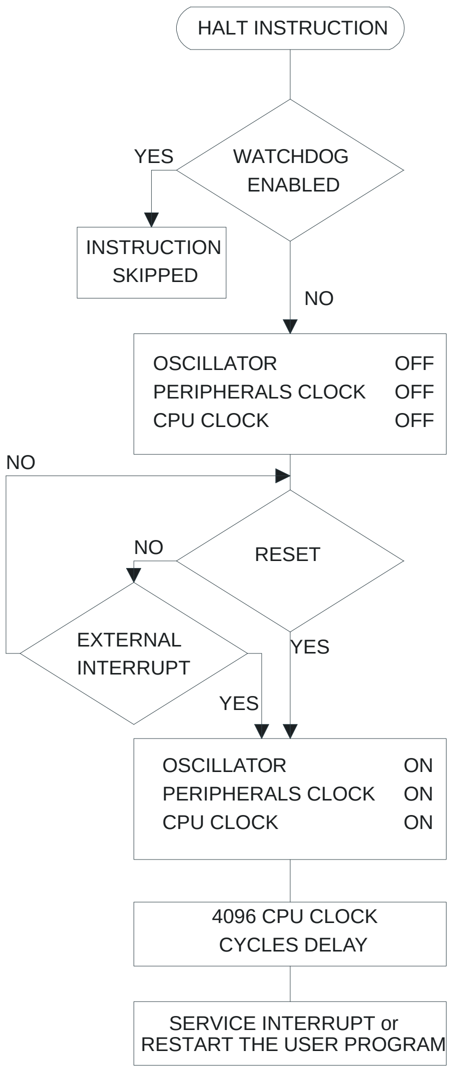
1) External Interrupt

2) External reset fetching the reset vector

Wake-up is regardless of the state of the External Interrupt mask. If a not masked External Interrupt occurs, the device, after the start up delay, continues executing the External Interrupt service routine. If a masked External Interrupt occurs, the device, after the start up delay, continues executing the User Program.

If the External Reset occurs, the device, after the start up delay, restarts the User Program.

Figure 5.5 HALT Flow Chart



6. I/O PORTS

6.1 Introduction

ST52x420 devices feature flexible individually programmable multifunctional input/output lines. Refer to figures 1.3 and 1.4 for specific pin allocations.

19 I/O lines, grouped in 3 different ports, are available on the ST52x420:

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

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

PORT C = 4-bit port. (PC0 - PC3 pins)

The PIN 18 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. Refer to chapter 10 for more details.

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.

REMARK : For a safe use of parallel port as digital I/O, it is recomended to configure the port B and C with all pins in input or output: you can group all the pins in one direction in the port B and the ones in the opposite direction in the port C. You can fully use the port A pins in both directions. Otherwise, you should operate as follow:

1) Reserve two RAM location: one for port B and one for port C.

2) When writing on port, first write to the reserved location and then send it to port.

3) When reading from port, first write the reserved location to port and then read it, otherwise the output pins may be subject to unwanted modification because the bus content is latched also in reading operation.

Be sure to prevent interrupts between the dummy writing intruction and the reading.

Example:

***ldrr 0, 2*0;** RAM 0 is the reserved location for portB

***ldpr 1,*0;** write data to port B

....

udgi; disables interrupt

ldpr 1,0 ; this dummy instruction causes the port B latch to be loaded with the right values of the output pins

ldri 30,10; reads data from port B without modifing the output pins

uegi; enables the interrupt

N.B.: you can use the port pins in alternate function normally.

Figure 6.1 Ports A & C Functional Blocks

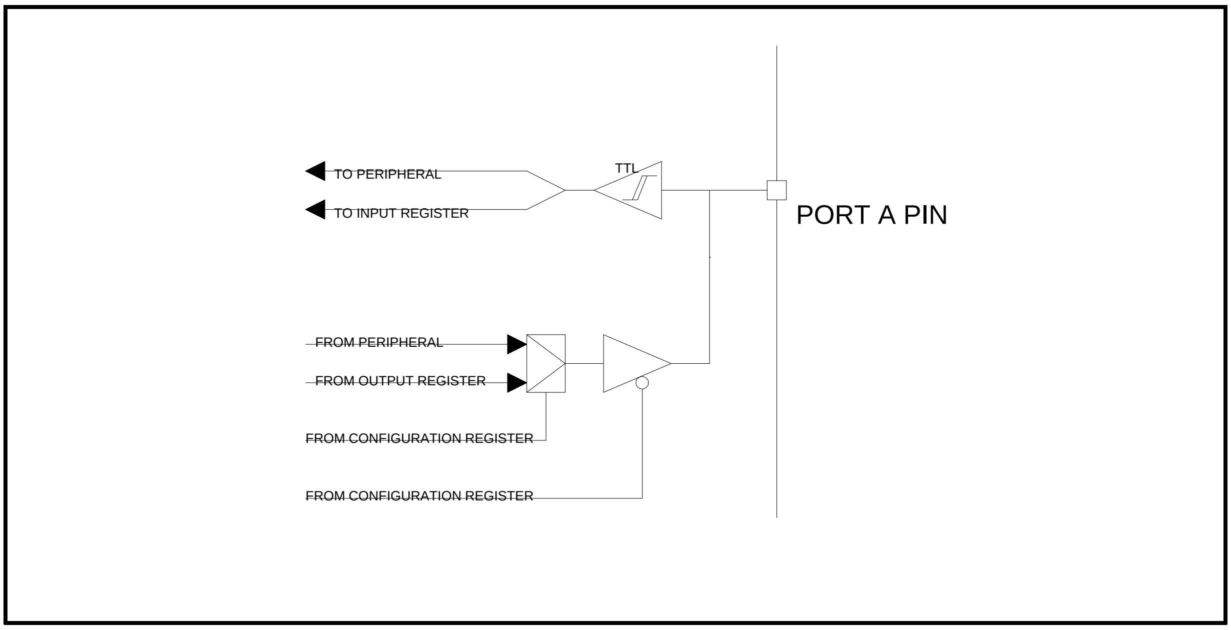
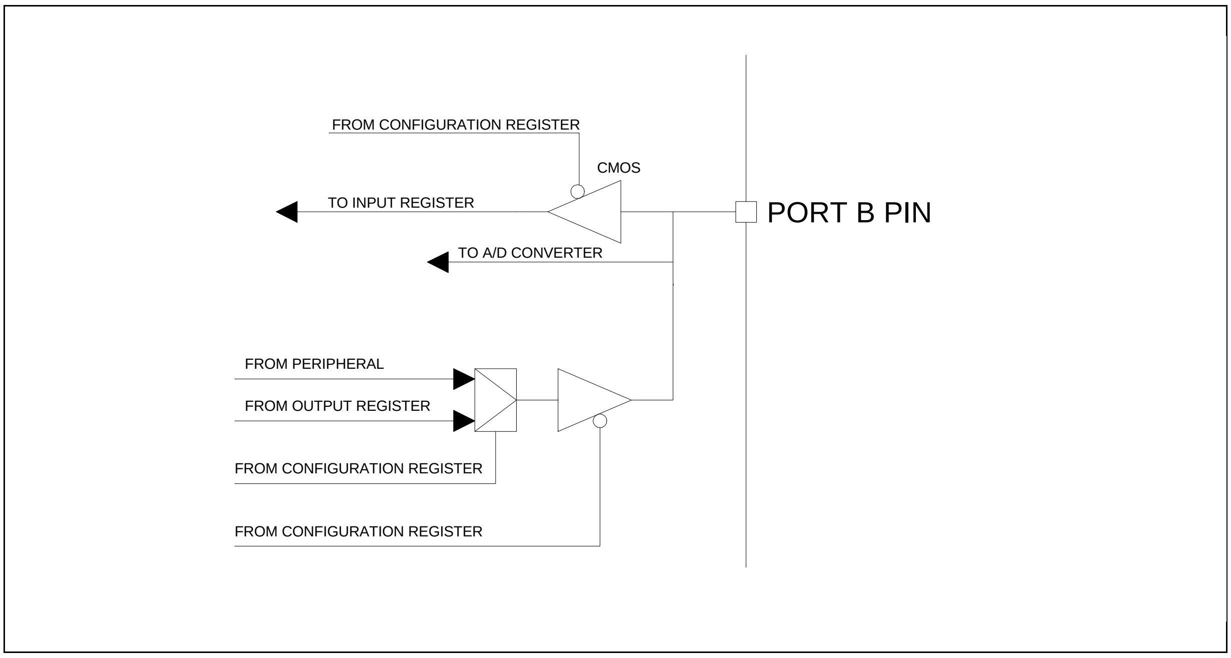


Figure 6.2 Port B Functional Blocks



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

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.

At RESET the Output Registers are '00000000'.

6.4 Alternate Functions.

Several ST52x420 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 20 (PA5/T0CLK) has to be used like external PWM/Timer0 clock, the Reg_Conf4(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.

Table 6.2 Input Register and I/O Ports

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

Table 6.3 Output Register and I/O Ports

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

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 25 in Output Mode
		1	Set the pin 25 in Input Mode
1	D1	0	Set the pin 24 in Output Mode
		1	Set the pin 24 in Input Mode
2	D2	0	Set the pin 23 in Output Mode
		1	Set the pin 23 in Input Mode
3	D3	0	Set the pin 22 in Output Mode
		1	Set the pin 22 in Input Mode
4	D4	0	Set the pin 21 in Output Mode
		1	Set the pin 21 in Input Mode
5	D5	0	Set the pin 20 in Output Mode
		1	Set the pin 20 in Input Mode
6	D6	0	Set the pin 19 in Output Mode
		1	Set the pin 19 in Input Mode
7	D7	0	Set the pin 18 in Output Mode
		1	Set the pin 18 in Input Mode
Reset Configuration ’11111111’			

Table 6.5 Ports B REG_CONF 13

Bit	Name	Value	Description
0	D0	0	Set the pin 9 in Output Mode
		1	Set the pin 9 in Input Mode
1	D1	0	Set the pin 10 in Output Mode
		1	Set the pin 10 in Input Mode
2	D2	0	Set the pin 11 in Output Mode
		1	Set the pin 11 in Input Mode
3	D3	0	Set the pin 12 in Output Mode
		1	Set the pin 12 in Input Mode
4	D4	0	Set the pin 15 in Output Mode
		1	Set the pin 15 in Input Mode
5	D5	0	Set the pin 16 in Output Mode
		1	Set the pin 16 in Input Mode
6	D6	0	Set the pin 17 in Output Mode
		1	Set the pin 17 in Input Mode
7	D7	0	Set the pin 18 in Output Mode
		1	Set the pin 18 in Input Mode
Reset Configuration ’11111111’			

Table 6.6 Port C REG_CONF 15

Bit	Name	Value	Description
0	D0	0	Set the pin 5 in Output Mode
		1	Set the pin 5 in Input Mode
1	D1	0	Set the pin 6 in Output Mode
		1	Set the pin 6 in Input Mode
2	D2	0	Set the pin 7 in Output Mode
		1	Set the pin 7 in Input Mode
3	D3	0	Set the pin 8 in Output Mode
		1	Set the pin 8 in Input Mode
4			Not Used
5			
6			
7			
Reset Configuration ’1111’			

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 9 Digital I/O
		1	pin 9 Analog Input
1	D1	0	pin 10 Digital I/O
		1	pin 10 Analog Input
2	D2	0	pin 11 Digital I/O
		1	pin 11 Analog Input
3	D3	0	pin 12 Digital I/O
		1	pin 12 Analog Input
4	D4	0	pin 15 Digital I/O
		1	pin 15 Analog Input
5	D5	0	pin 16 Digital I/O
		1	pin 16 Analog Input
6	D6	0	pin 17 Digital I/O
		1	pin 17 Analog Input
7	D7	0	pin 18 Digital I/O
		1	pin 18 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 ST52x420. The configuration of these pins is performed by using the Configuration Registers REG_CONF 12 and REG_CONF 16. Anyway the correct I/O mode of the related pin has to be selected by means of REG_CONF 4, 13 and 15.

Table 6.8 PWM/Timers REG_CONF 16

Bit	Name	Value	Description
0	P6SL	1	Pin 6 is configured as Port C Digital I/O
		0	Pin 6 is configured as PWM/Timer 0 output T0OUT
1	P7SL	1	Pin7 is configured as Port C Digital I/O
		0	Pin 7 is configured as PWM/Timer 0 output T1OUT
2	P8SL	1	Pin 8 is configured as Port C Digital I/O
		0	Pin 8 is configured as PWM/Timer 0 output T2OUT
3-7	NC	X	Not Used
Reset Configuration '00000000'			

Table 6.9 PWM/Timers REG_CONF 12

Bit	Name	Value	Description
0	P24SL	1	Pin 24 is configured as PWM/Timer 0 complementary output T0OUT
		0	Pin24 is configured as Port A Digital I/O
1	P23SL	1	Pin 23 is configured as PWM/Timer 1 complementary output T1OUT
		0	Pin23 is configured as Port A Digital I/O
2	P22SL	1	Pin 22 is configured as PWM/Timer 2 complementary output T2OUT
		0	Pin22 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'			

Figure 6.3 REG_CONF 12

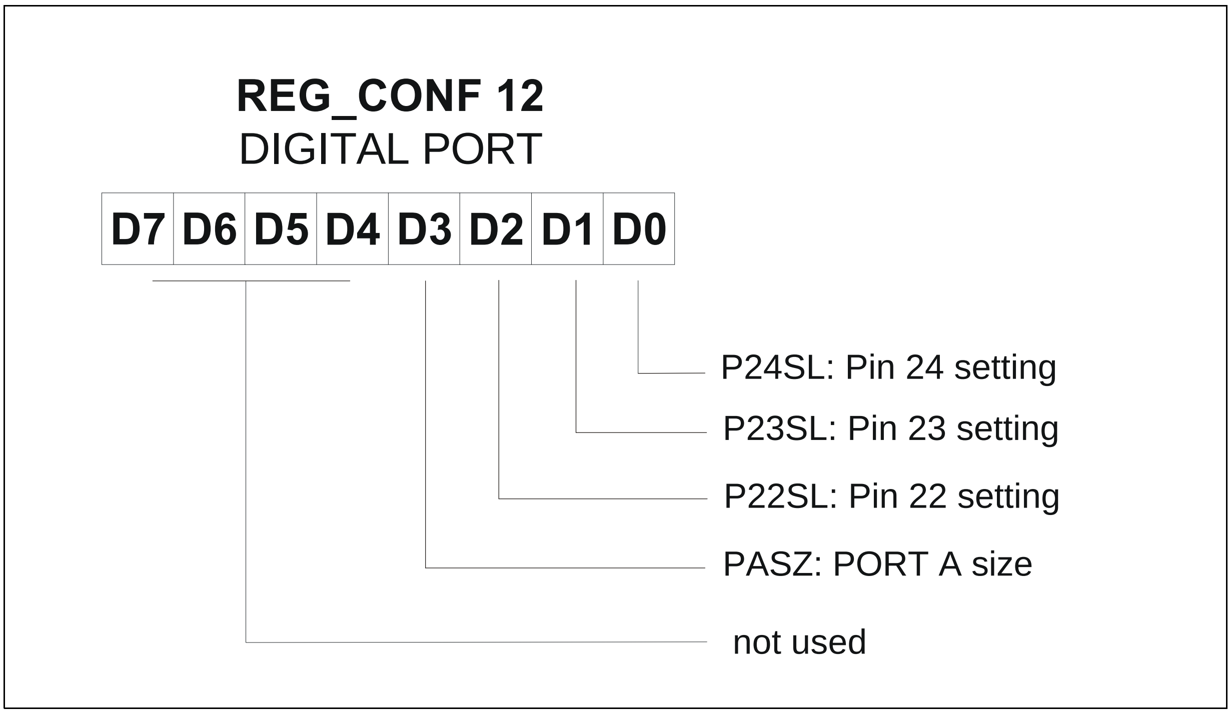
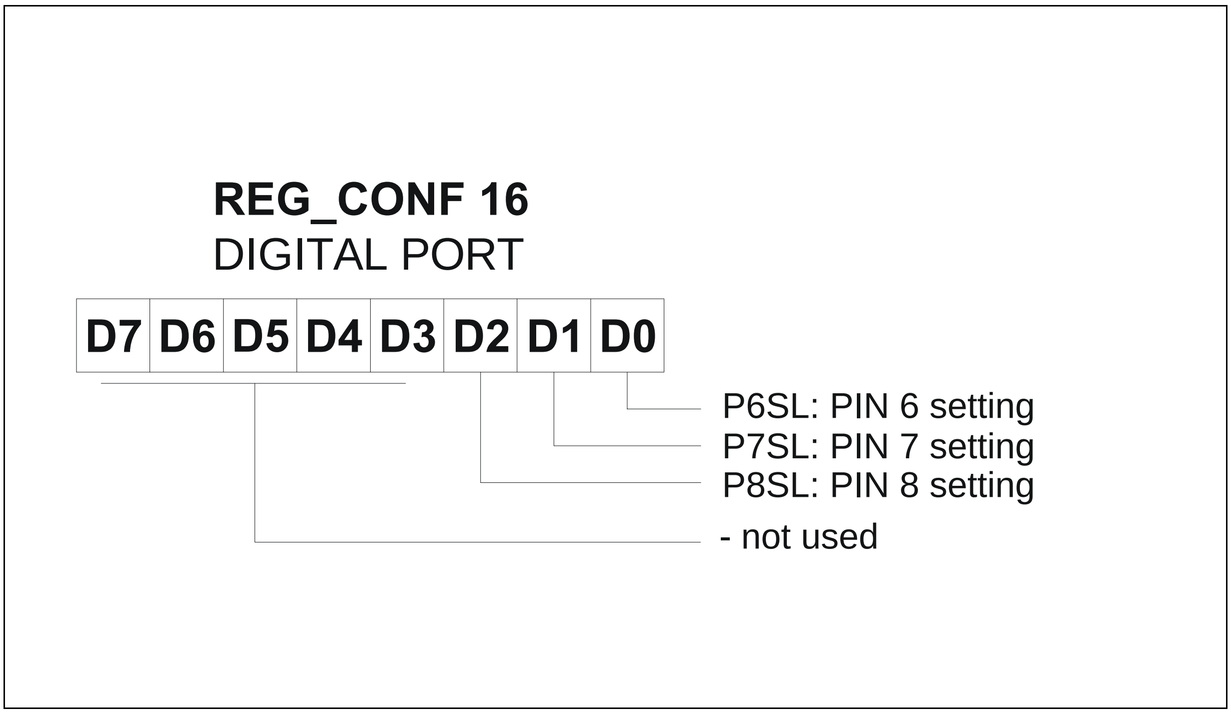


Figure 6.4 REG_CONF 16



7. A/D CONVERTER

7.1 Introduction

The A/D Converter of ST52x420 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 4.1 µs with a 20 MHz clock. This period also includes the time 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.

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

A conversion is performed in 78 A/D clock pulses plus 4 CKM pulses.

The A/D clock is derived from the clock master (CKM) and it is fixed, to CKM or to CKM divided by 2, by means of the bit SCK of the A/D configuration register REG_CONF3 (See table 7.1). The maximum A/D clock frequency has to be 20 MHz.

The conversion range is between the analog Vss and Vdd 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.

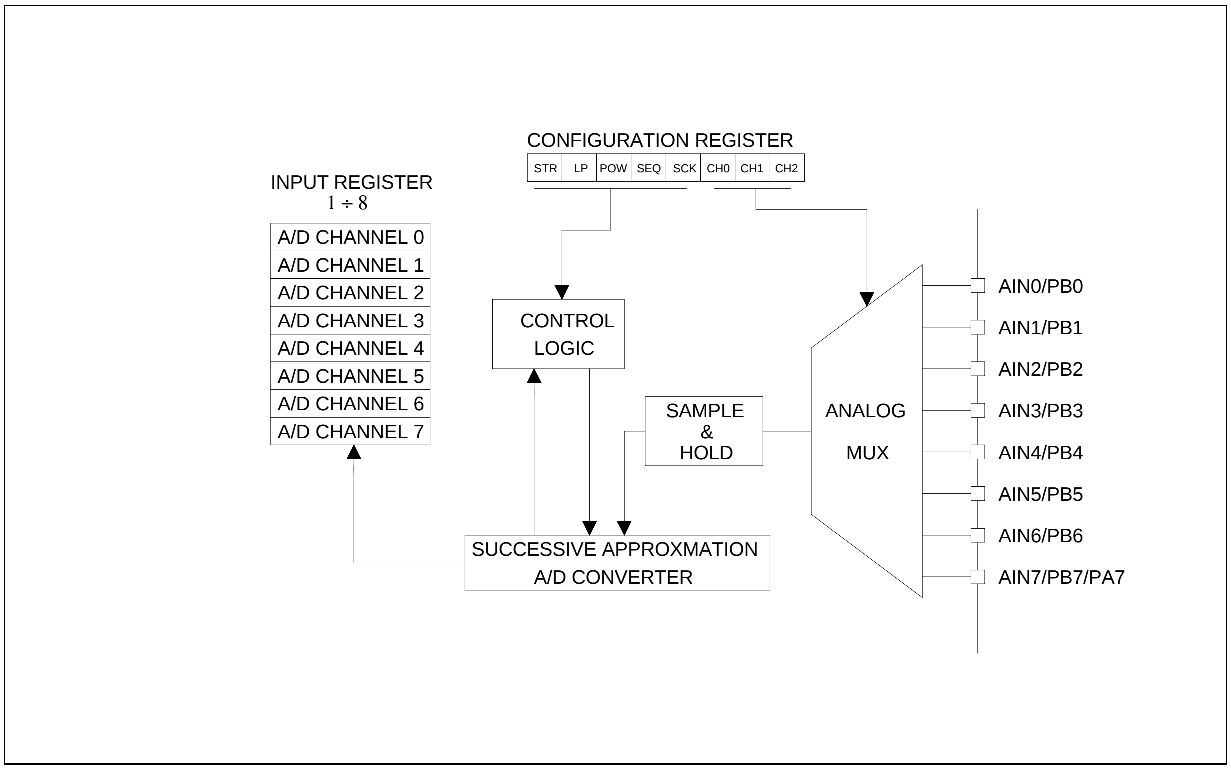
The ST52x420 Interrupt Unit provides one maskable channel for the End of Conversion (EOC)

7.2 Functional Description

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

If input voltage is greater than or equal to Vdda (Voltage Reference high) then result is equal to FFh (full scale) without overflow indication.

Figure 7.1. A/D Converter Structure



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:

$$DigitalResult = 255x\frac{InputVoltage}{ReferenceVoltage}$$

Where Reference Voltage is Vdda - Vss.

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 stabilisation 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.

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.

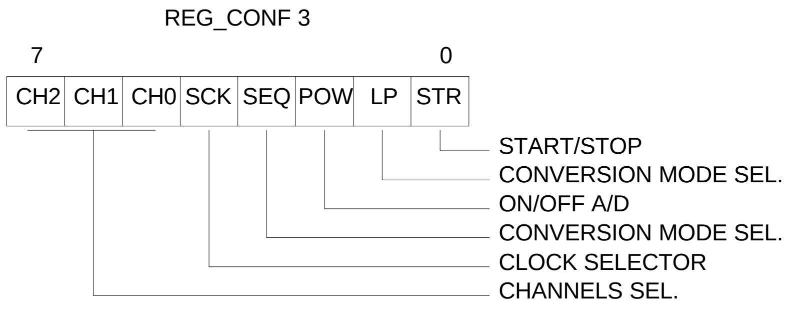
i.e CH(2:0) = ’011’ means continuous conversion of channel 3. At the end of each conversion the relative IR is updated with the last conversion result, while the former value is lost.

To stop the conversion STR has to be set to ’0’. If STR is reseted during a conversion, the ADC will work until the end of the conversion, generating the related Interrupt request, if it was activated.

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.

Figure 7.2. A/D Configuration Register (REG_CONF3)



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’. If STR is reseted during a conversion, the ADC will work until the end of the conversion, generating the related Interrupt request, if it was activated.

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 consuption. This state is also the reset configuration and it is forced by hardware when the core is in HALT state (after a HALT istruction execution).

7.3 A/D Registers Description

The result of the conversions of the 8 available channels are loaded in the 8 Input Registers from the decimal addres 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 istruction:

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_CONF3. The figure 7.2 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**: Start 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 ST52x420 is able to work with a clock frequency up to 20 MHz. It is useful to set SCK = ’1’ also when the clock master is lower than 10 MHz and an 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 conversios sequences; a logical level ’0’ stops the conversion. When the A/D is running in the Single Modes (LP=’0’), this bit is hardware reset at the end of a conversion sequence.

Table 7.1 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

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 delay can be selected by using the WDT configuration register.

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

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

To stop the WDT during the user program excution 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 CLKM.

The clock master is then divided by 500 thus obtaining the WDT CLK signal that is used to fix the timeout of the WDT.

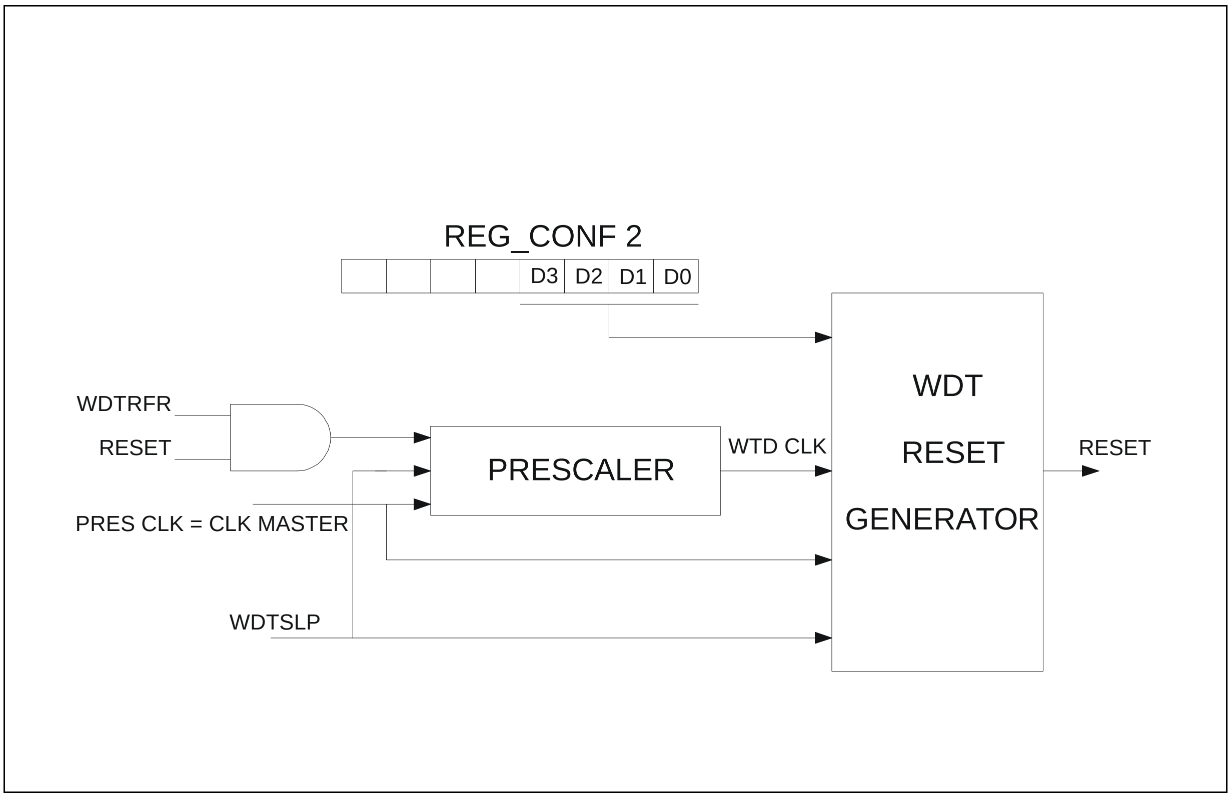
According to the WDT configuration register values it possible to define a WDT delay. Changing the clock master frequency the timeout delay can be calculated according to the configuration register values REG_CONF 2 as described in the following section.

For example, when the clock master is 5 MHz, the delay could be between 0.1 mS and 937.5 mS .

Table 8.1 Watchdog Timing range (CLK=5

	WDT timeout period (mS)
min	1*(500/CLKM)
max	9375*(500/CLKM)

Figure 8.1 Watchdog Block Diagram



8.2 Configuration Register 2 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 multipling 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	16250
		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	1625
		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)
1	2	0.5
4	8	0.125
5	10	0.1
8	16	0.0625
10	20	0.05
20	40	0.025

9. PWM/TIMERS

ST52x420 offers three on-chip PWM/Timer peripherals :TIMER0, TIMER1 and TIMER2.

The ST52x420 timers have the same internal structure, that, basically, consists of an 8-bit counter with a 16-bit programmable prescaler, thus giving a maximum count of 2^{24} (see figure 9.1).

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 complementary signal, on external pins, by setting PxSL 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, TIMER 0 can use also external Timer0 START/STOP signals (Input capture and Output compare), external Timer 0 RESET signal, and external Timer 0 clock: T0STRT, T0RES and 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 9.7)

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

The PWM/Timer x Status can be read from the PWM_x_STATUS Input Register (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

Figure 9.1. Timer Peripheral Block Diagram

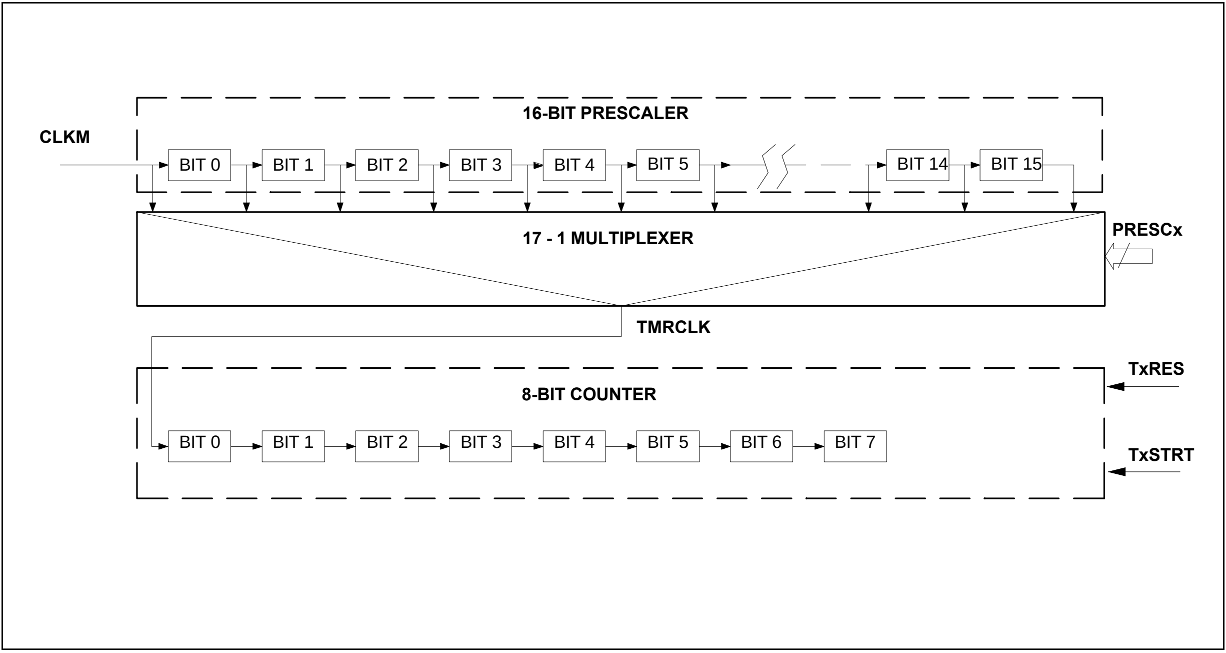
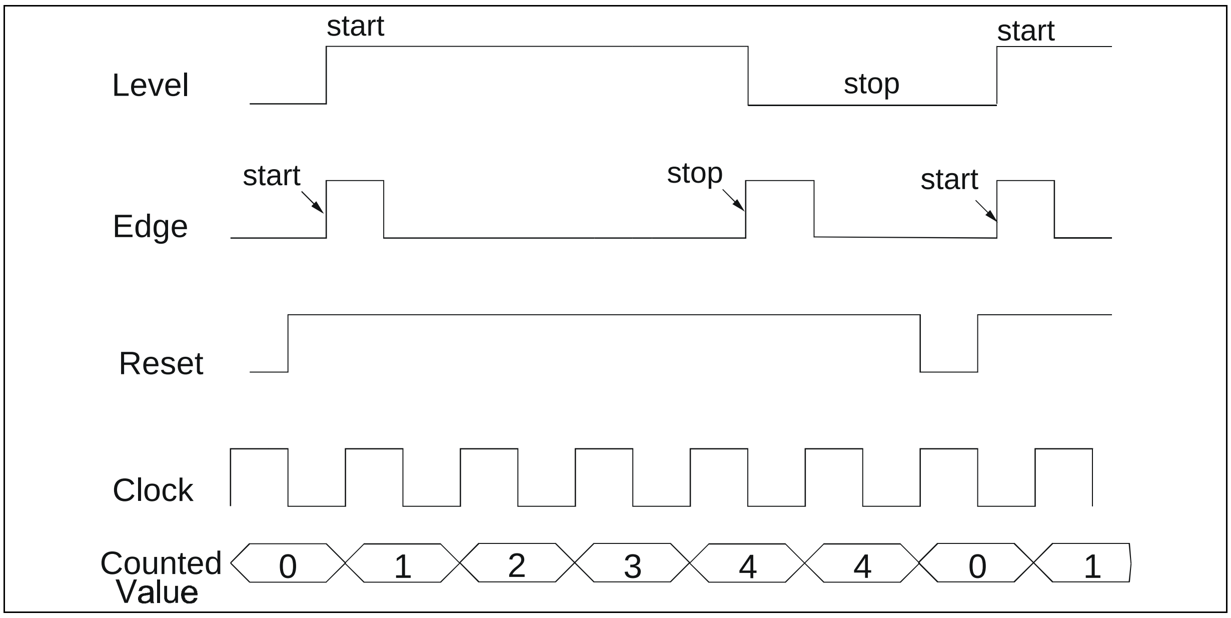


Figure 9.2. Timer 0 External START / STOP Mode



(CLKM) or, only for TIMER0, from the external signal provided on the T0CLK pin.

REMARKS: 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).

TxSTRT signal start/stop the Timer x counting. 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 use 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,

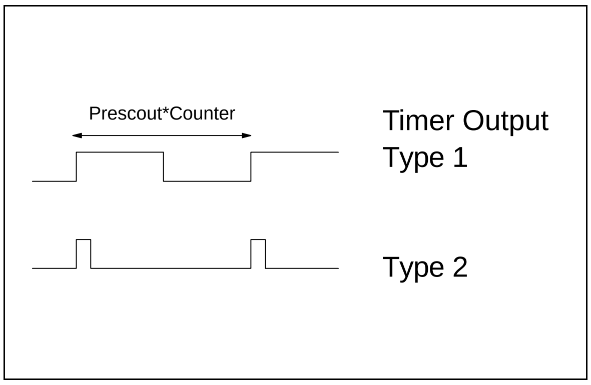
2) TISTR0 = TISTR1 = TISTR2 = 0,

3) T0MSK = T1MSK = T2MSK = 1,

4) TIRST0 = TIRST1 = TIRST2 = 1,

5) TISTR0 = TISTR1 = TISTR2 = 1,

Figure 9.3. TIMEROUT Signal Type



6) T0MSK = T1MSK = T2MSK = 0,

(all timers start simultaneously)

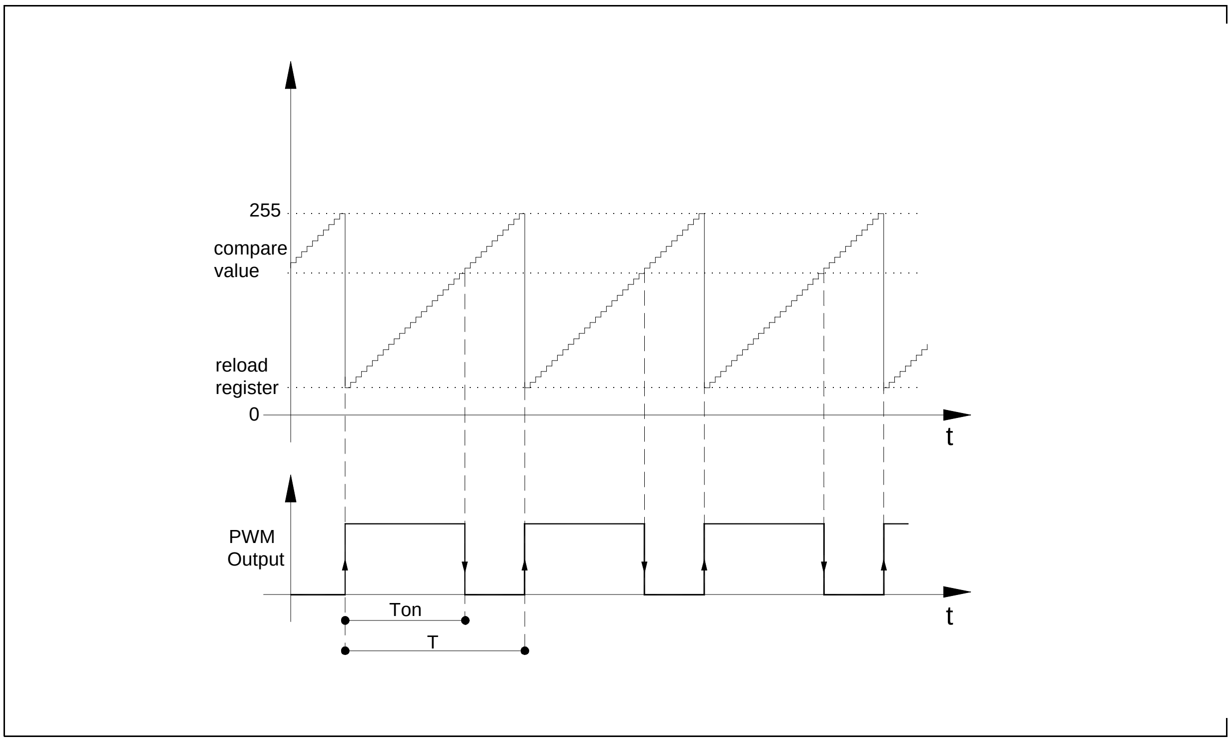
When TxMSK is 1 the TIMER x is reseted.

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.

Fig. 9. PWM Mode with Auto Reload



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 PWM_x_COUNT Output Register value (8 bit) (Output Registers 3, 5 or 7. See table 2.4) + 1, that is the value to count.

If the PWM_x_COUNT Output Register value is zero (default value), the Timer x output signal, TIMERxOUT, is always at low level

REMARKS: The first period of TIMERxOUT signal is shorter than the other periods of a Δt=1/CLKM

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.

REMARKS: The first period of TIMERxOUT signal is shorter than the other periods of a Δt=1/(2*TMRCLKx) - 1/CLKM.

The TIMERxOUT signal can be available on TxOUT pin and the $\overline{\text{TIMERxOUT}}$ inverted signal can be available on 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 PWM_x_RELOAD Output Register, as shown in figure 9.4.

The PWM_x_RELOAD Output Register 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.

REMARKS: 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) *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 PWM_x_COUNT Output Register 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.

REMARKS. If PWM_x_RELOAD value increases the duty cycle resolution decreases. PWM_x_RELOAD=255 is not acceptable.

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.

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 in to 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

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 9.4. Configuration Register 5

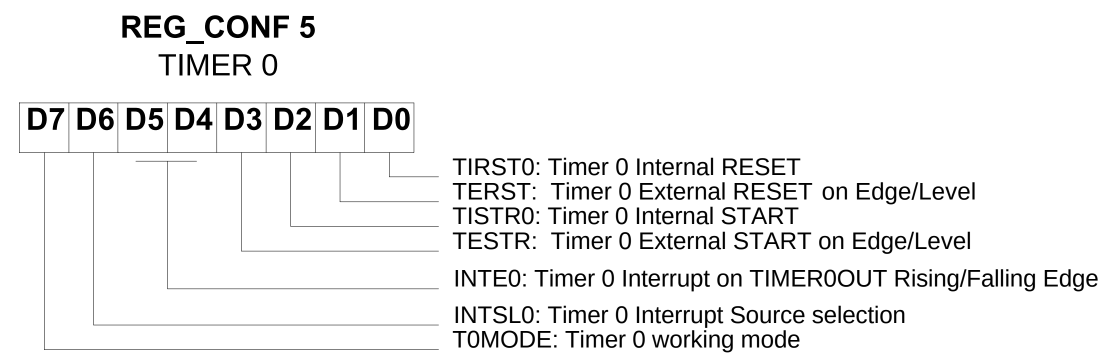

REG_CONF 5
TIMER 0
D7 D6 D5 D4 D3 D2 D1 D0
TIRST0: Timer 0 Internal RESET
TERST: Timer 0 External RESET on Edge/Level
TISTR0: Timer 0 Internal START
TESTR: Timer 0 External START on Edge/Level
INTE0: Timer 0 Interrupt on TIMER0OUT Rising/Falling Edge
INTSL0: Timer 0 Interrupt Source selection
T0MODE: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

Figure 9.5. Configuration Register 6

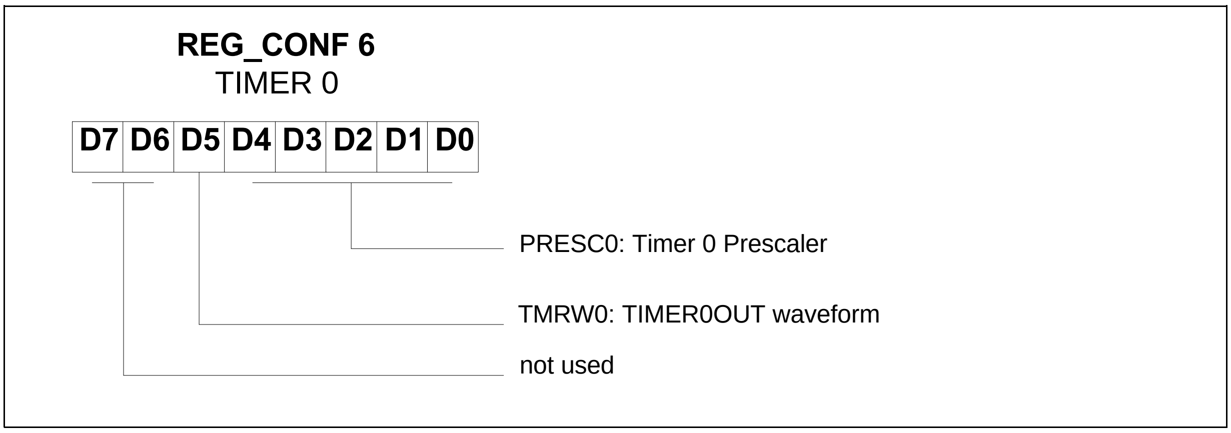


Table 9.3. Configuration Register 7 Description

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

Figure 9.6. Configuration Register 7

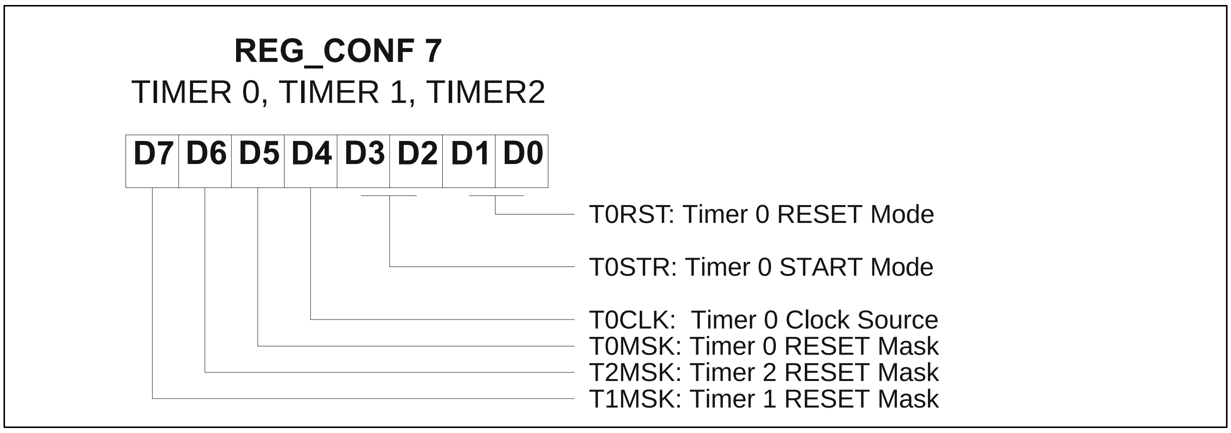

REG_CONF 7
TIMER 0, TIMER 1, TIMER2
D7 D6 D5 D4 D3 D2 D1 D0
T0RST: Timer 0 RESET Mode
T0STR: Timer 0 START Mode
T0CLK: Timer 0 Clock Source
T0MSK: Timer 0 RESET Mask
T2MSK: Timer 2 RESET Mask
T1MSK: Timer 1 RESET Mask

Table 9.4. Configuration 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 9.7. Configuration Register 8

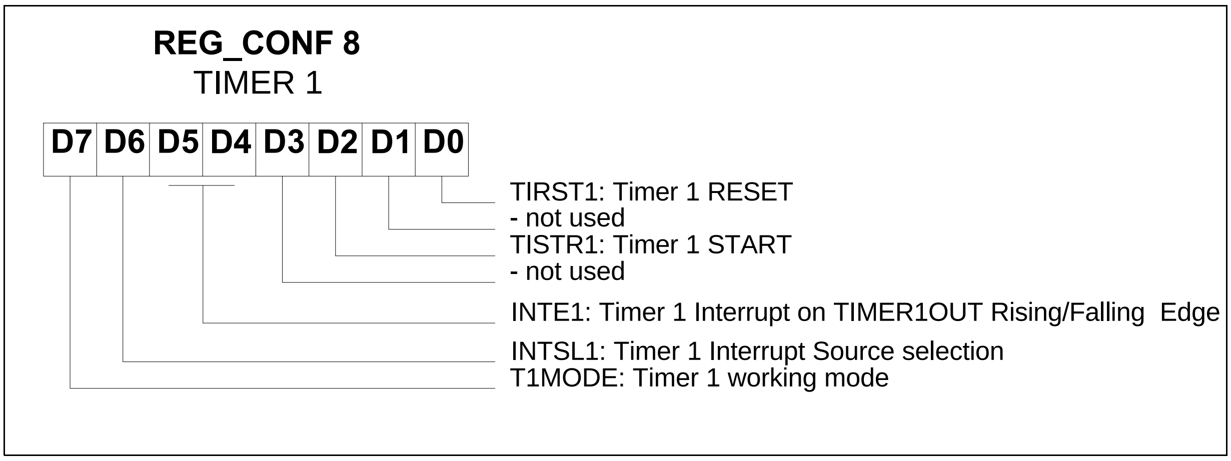

REG_CONF 8
TIMER 1
D7 D6 D5 D4 D3 D2 D1 D0
TIRST1: Timer 1 RESET
- not used
TISTR1: Timer 1 START
- not used
INTE1: Timer 1 Interrupt on TIMER1OUT Rising/Falling Edge
INTSL1: Timer 1 Interrupt Source selection
T1MODE: Timer 1 working mode

Table 9.5. Configuration 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 9.8. Configuration Register 9

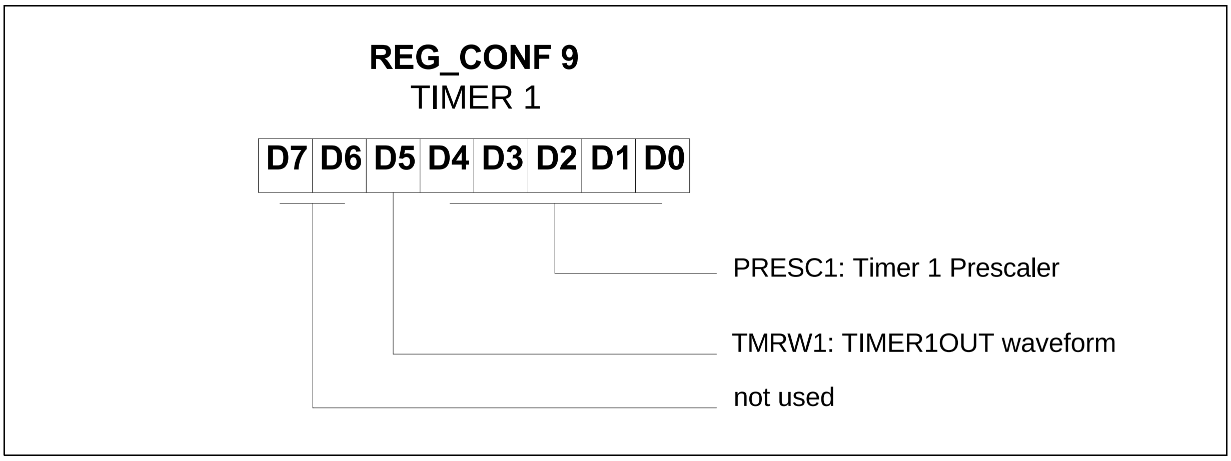


Table 9.6. Configuration Register 10

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 9.9. Configuration Register 10

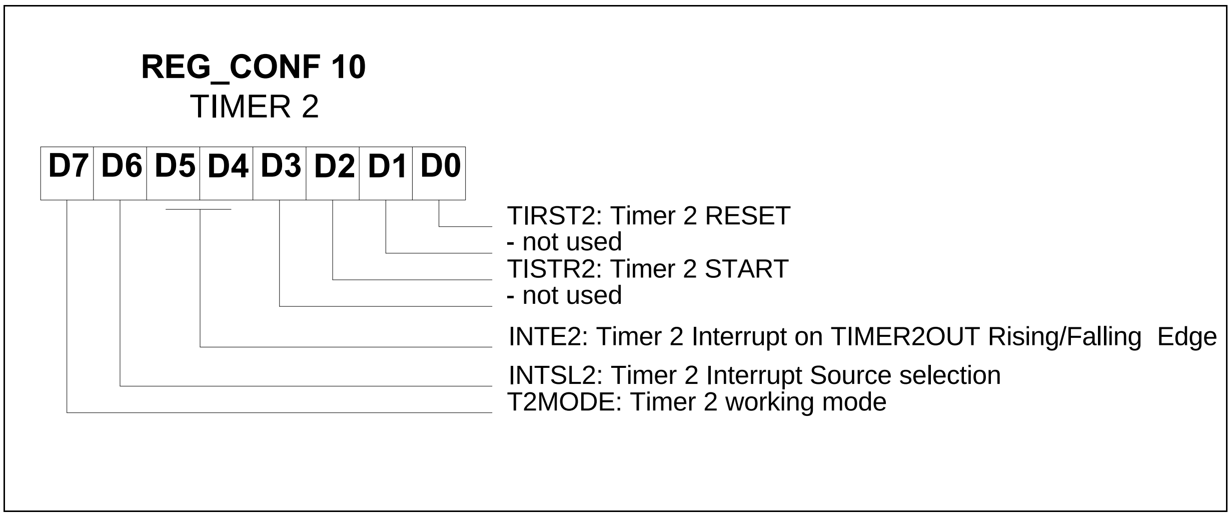

REG_CONF 10
TIMER 2
D7 D6 D5 D4 D3 D2 D1 D0
TIRST2: Timer 2 RESET
- not used
TISTR2: Timer 2 START
- not used
INTE2: Timer 2 Interrupt on TIMER2OUT Rising/Falling Edge
INTSL2: Timer 2 Interrupt Source selection
T2MODE: Timer 2 working mode

Table 9.7. Configuration Register 11

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 9.10. Configuration Register 11

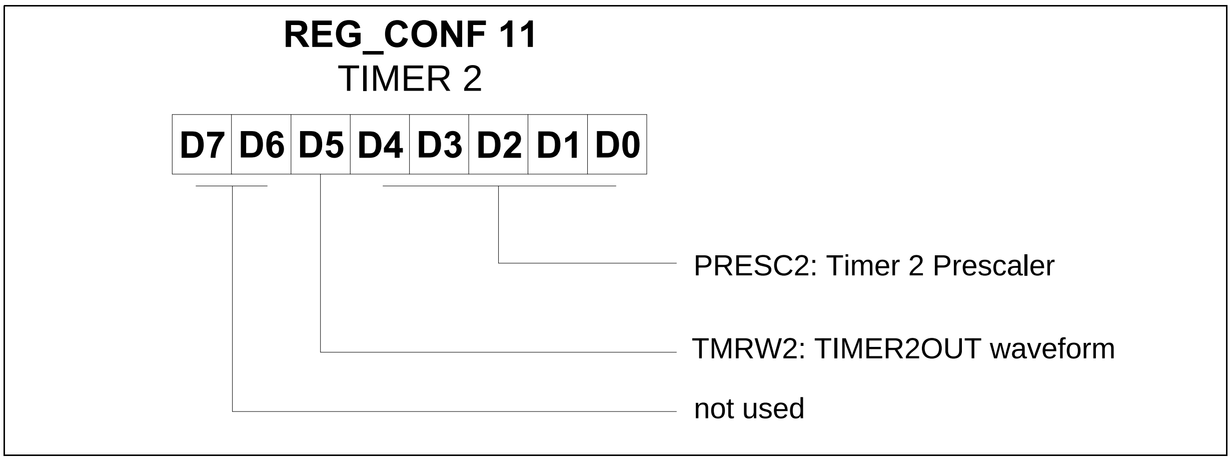


Table 9.8. Configuration Register 12

Bit	Name	Value	Description
0	P24SL	0	Pin 24 equal to PORT A Digital I/O
		1	Pin 24 equal to $\overline{\text{T0OUT}}$
1	P23SL	0	Pin 23 equal to PORT A Digital I/O
		1	Pin 23 equal to $\overline{\text{T1OUT}}$
2	P22SL	0	Pin 22 equal to PORT A Digital I/O
		1	Pin 22 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 9.11. Configuration Register 12

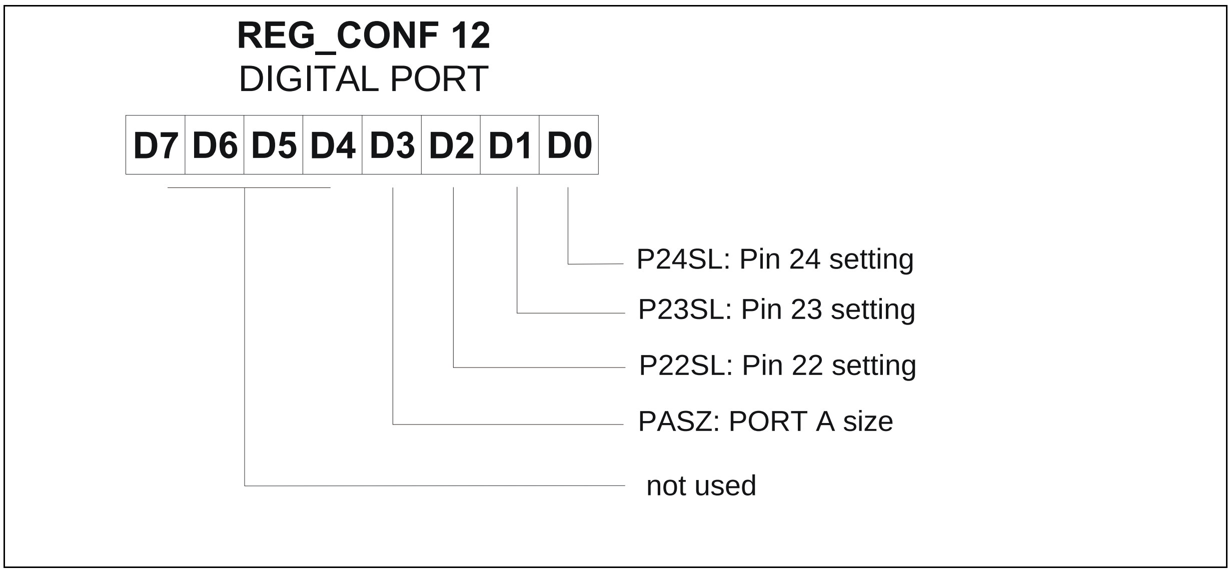


Table 9.9. Configuration Register 16

Bit	Name	Value	Description
0	P6SL	1	Pin 6 equal to PORT C Digital I/O
		0	Pin 6 equal to T0OUT
1	P7SL	1	Pin 7 equal to PORT C Digital I/O
		0	Pin 7 equal to T1OUT
2	P8SL	1	Pin 8 equal to PORT C Digital I/O
		0	Pin 8 equal to T2OUT
3	-	-	- not used
4	-	-	- not used
5	-	-	- not used
6	-	-	- not used
7	-	-	- not used

Figure 9.11. Configuration Register 16

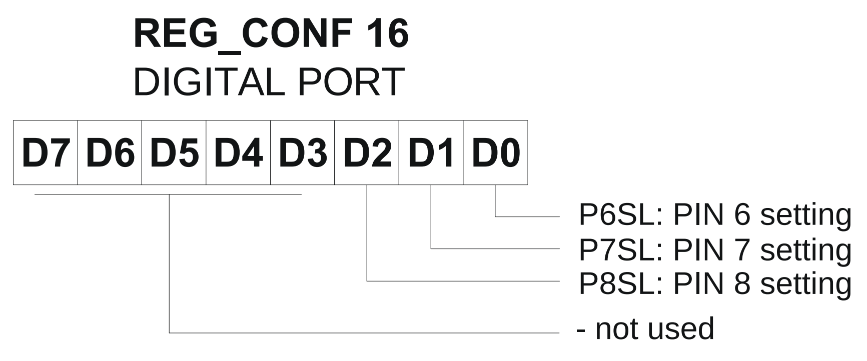


Table 9.10 Input Registers 13. PWM_0_STATUS

Bit	Name	Value	Description
0	STR0ST	0	TIMER 0 is STOP
		1	TIMER 0 is 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 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings

This product contains devices to protect the inputs against damage due to high static voltages, however it is advised to take normal precaution to avoid any voltage higher than maximum rated voltages.

For proper operation it is recommended that V_I and V_O must be higher than V_{SS} and smaller than V_{DD}.

Reliability is enhanced if unused inputs are connected to an appropriated logic voltage level

Table 10.1. Absolute Maximum Ratings

Symbol	Parameter	Value	Unit
V_{DD}	Supply Voltage	-0.5 to 7	V
V_I	Input Voltage	V_{SS}-0.3 to V_{DD}+0.3 (1)	V
V_O	Output Voltage	V_{SS}-0.3 to V_{DD}+0.3 (1)	V
V_{DDA}, V_{SSA}	Analog Supply Voltage	V_{SS}-0.3 to V_{DD}+0.3 (1)	V
V_{PP}	EPROM Programming Voltage	13	V
I_O	Standard Output Source Sink Current	±16	mA
	Operating TemperatureST52x420B /x	-25 to +85	°C
T_{STG}	Storage Temperature	-65 to +150	°C

Note: Stresses above those listed in the Table "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only and operation of the device at these or any other conditions above those indicated in the Operating sections of this specification is not implied. Exposure to Absolute Maximum Rating conditions for extended periods may affect device reliability. Refer also to the STMicroelectronics SURE Program and other relevant quality documents.

1. Within these limits, clamping diodes are garanteed to be not conductive.

(VSS or VDD) RECOMMENDED OPERATING CONDITIONS

(Operating Condition: V_{DD}=5V±5%-T_A= -25 °C to 85 °C, unless otherwise specified)

Table 10.2. Recommended Operation Condition

Symbol	Parameters	Test Conditions	Min	Typ	Max	Unit
V_{DD}	Operating Supply Voltage(1) , (2)	f_{OSC} = 20 MHz	4.75	5.0	5.25	V
V_{PP}	Programming Voltage		11.4	12	12.6	V
V_O	Ouput Voltage		V_{SS}		V_{DD}	V
V_{DDA}, V_{SSA}	Analog Supply Voltage (1)	$Vss \leq V_{SSA} < V_{DDA} \leq V_{DD}$	V_{SS}		V_{DD}	V
f_{OSC}	Oscillator Frequency (3)		1		20	MHz

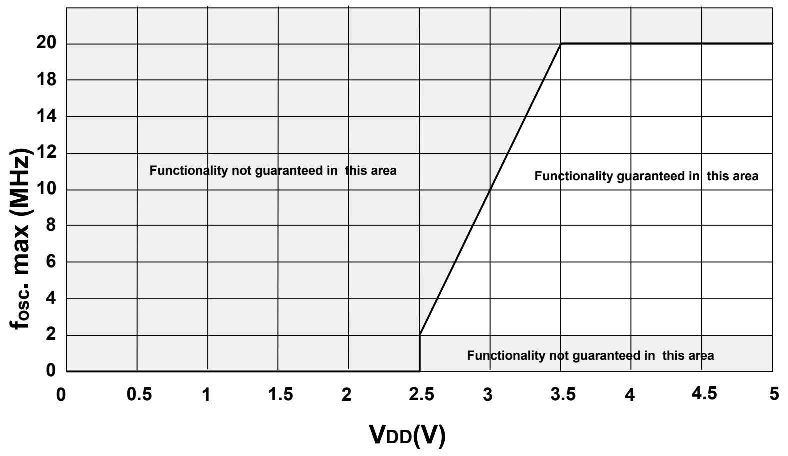
NOTE:

1) The maximum difference between V_{SS} and V_{SSA}, and between V_{DD} and V_{DDA}, must be, in module, less than 0.6 V. The minimum value of V_{DDA} is 3 V.

2) V_{DD} depend on f_{OSC}, see figure 10.1.

3) The f_{OSC} min allowed to use the A/D Converter is 2 MHz

Fig. 10.1 fosc Maximum Operating Fraquency vs VDD supply



NOTE: The data shown in this figure are guaranteed by design and not by testing phase.

DC ELECTRICAL CHARACTERISTICS

(Operating Condition: V_{DD}=5V±5%-T_A= -25 °C to 85 °C, unless otherwise specified)

Table 10.3 DC Electrical Characteristics

Symbol	Parameter	Test Conditions	Min	Typ	Max	Unit
V_{IL}	CMOS type Low Level Input Voltage. Port B pins. see fig.10.7	VDD =5V, T_A=25°C.			2	V
	TTL type Schmitt trig. Low Level Input Voltage. Port A and Port C pins. see fig.10.6				0.8	
	TTL type Schmitt trig. Low Level Input Voltage. RESET pins. see fig.10.6				0.8	
V_{IH}	CMOS High Level Input Voltage. Port B pins. see fig.10.7	V_{DD} =5 V, T_A=25°C.	3.3			V
	TTL type Schmitt trig. High Level Input Voltage. Port A and Port C pins. see fig.10.6		2.2			
	TTL type Schmitt trig. High Level Input Voltage. RESET pins. see fig.10.6		2.2			
V_{OL}	Standard Low Level Output Voltage	I_{OL} =8mA			0.4	V
V_{OH}	Standard High Level Output Voltage	I_{OL} =-8mA	V_{DD}-0.5			V
V_{Hys}	TTL type Schmitt trig. Hysteresis Voltage	see fig. 10.6		1.4		V
I_{IL}	Low Level Leakage Input Current	V_I=V_{SS}		-1		μA
I_{IH}	High Level Leakage Input Current	V_I=V_{DD}		+4		μA
I_{DD}	Supply Current in RUN mode	VPP connected with $V_{ss;}$ F_{OSC}= 10 MHz		10		mA
	Supply Current in HALT mode (1)			1	10	μA
I_{DDA}	Analog Supply Current in RUN mode	V_{PP} connected with $V_{ss;}$ F_{OSC}= 10 MHz		0.34		mA
	Analog Supply Current in HALT mode (1)			0.2		μA

Note:

The Supply Currents in WAIT mode (I_{DD} and I_{DDA}) depend on the active peripherals.

1) These values are guaranteed by design and not by testing

AC ELECTRICAL CHARACTERISTICS

(Operating Condition: V_{DD}=5V±5%-T_A= -25 °C to 85 °C, unless otherwise specified)

Table 10.4. AC Electrical Characteristics

Symbol	Parameter	Test Conditions	Min	Typ	Max	Unit
R_S	Input protection Resistor	All Input Pins		1		kΩ
C_S	Pin Capacitance	All Input Pins		5		pF

Table 10.5. Timing Parameters

Symbol	Parameters	Test Conditions	Min	Typ	Max	Unit
f_{OSC}	Oscillator Frequency		1		20	MHz
t_{CLH}	Clock High		25		500	ns
t_{CLL}	Clock Low		25		500	ns
t_{SET}	Setup	see fig 10.2		5		ns
t_{HLD}	Hold	see fig. 10.2		5		ns
t_{WRESET}	Minimum Reset Pulse Width	F_{OSC}=20 MHz	100			ns
t_{WINT}	Minimum External Interrupt Pulse Width	FOSC=20 MHz	100			ns
t_{IR}	Input Rise Time	see fig.10.3			15	ns
t_{IF}	Input Fall Time	see fig.10.3			15	ns
t_{OR}	Output Rise Time	C_{LOAD}=10 pF see fig.10.2		10		ns
t_{OF}	Output Fall	CLOAD=10 pF see fig.10.3		10		ns

Figure 10.2. Data Input Timing

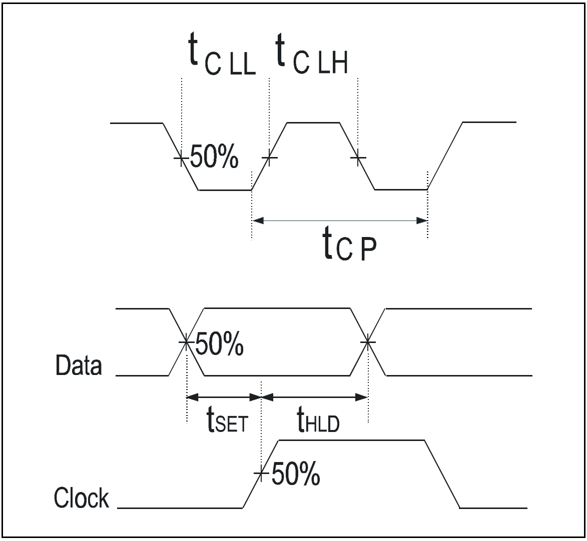


Figure 10.3. I/O Rise and Fall Timing

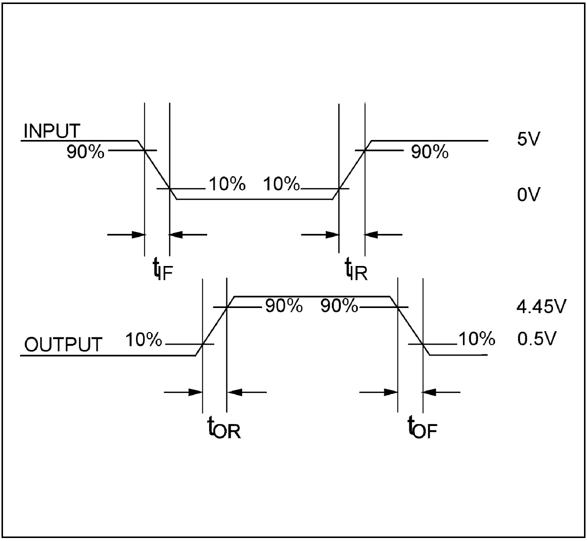


Figure 10.4. PORT A and PORTC Pin Equivalent Circuit

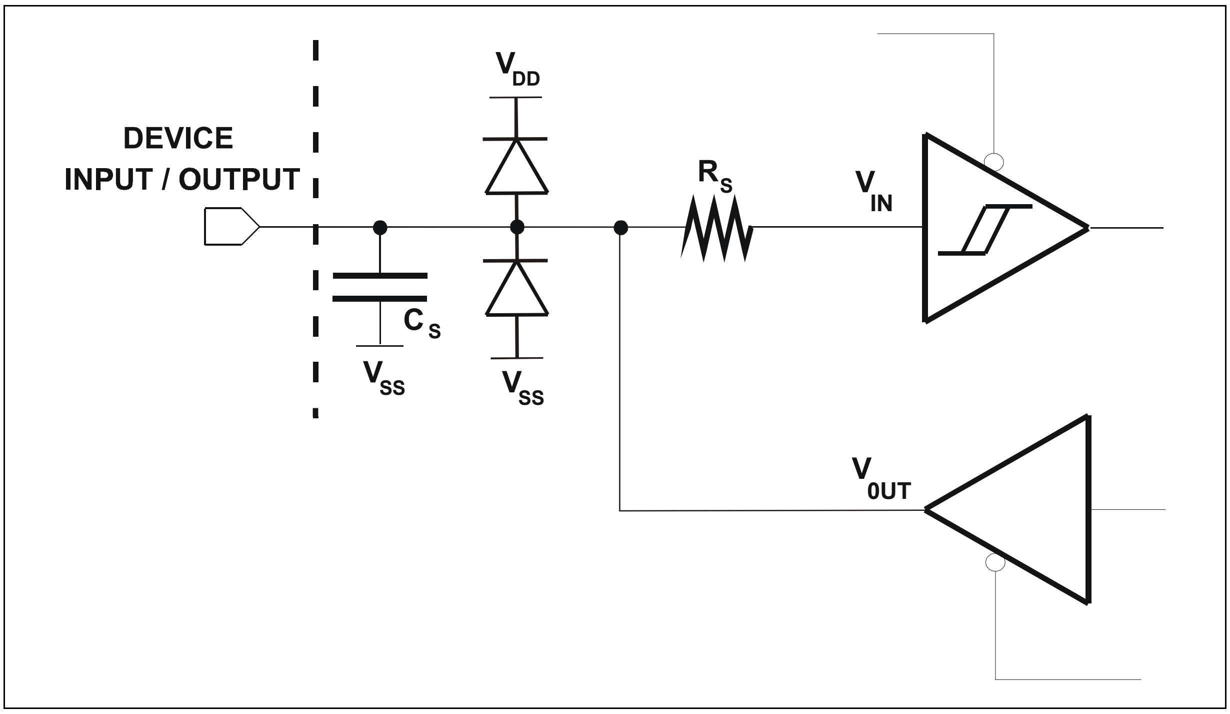


Figure 10.5. PORT B Pin Equivalent Circuit

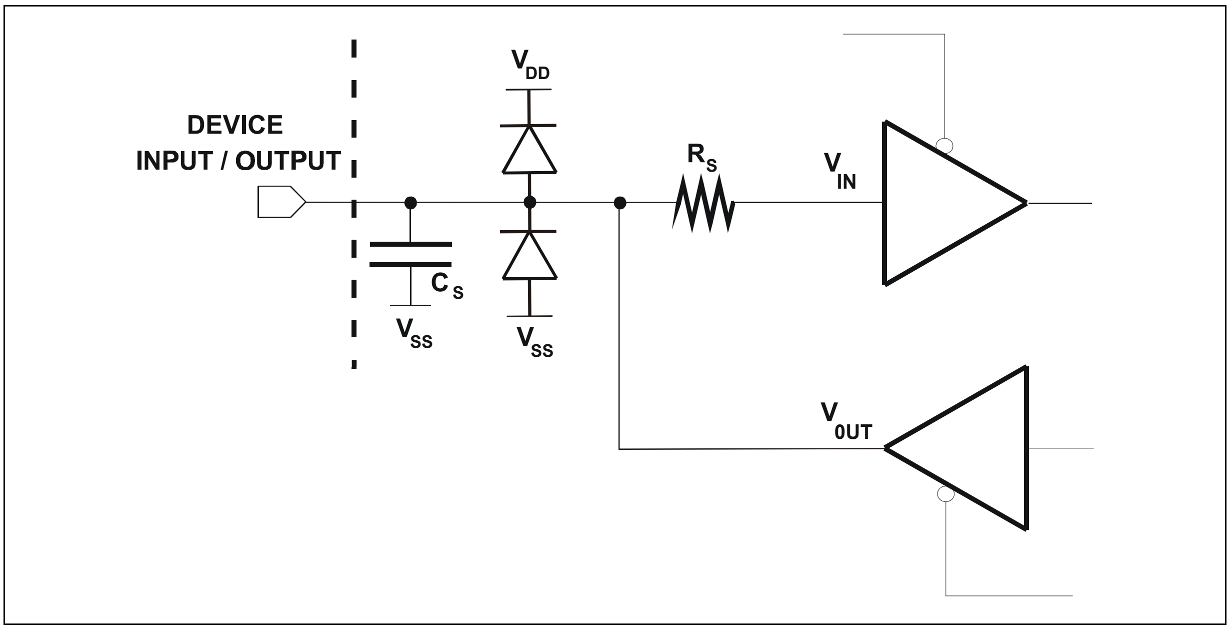


Figure 10.6. TTL-level Input Schmitt Trigger

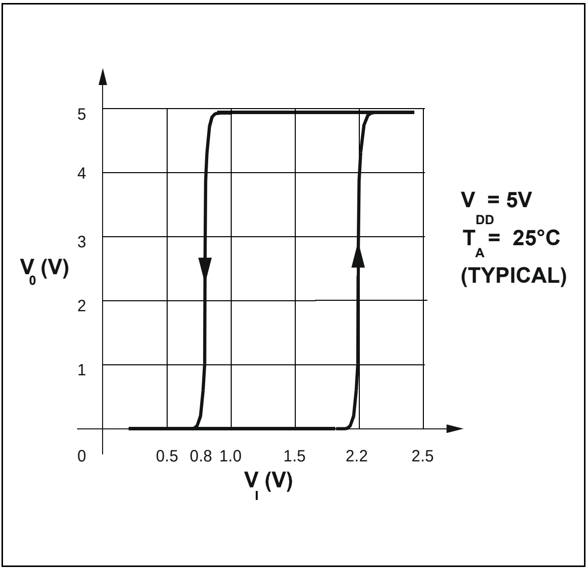
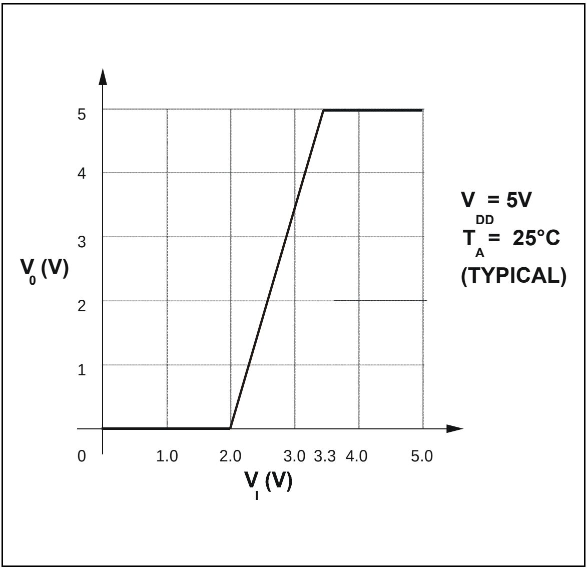


Figure 10.7 PORT B pins CMOS-level Input



TIMER CHARACTERISTICS

(Operating Condition: V_{DD}=5V±5%-T_A=-25 °C to 85 °C, unless otherwise specified)

Table 10.7. Timer Characteristics

Symbol	Parameter	Min	Typ	Max	Unit
t_{RES}	Resolution	$1/F_{OSC}$			µs
f_{IN}	External Input Frequency on TIMER0 Internal Input Frequency on timer			10 20	MHz
t_W	Pulse Width on TIMEROUT pin	$1/F_{OSC}$			µs

A/D CONVERTER CHARACTERISTICS

(Operating Condition: V_{DD}=5V±5%- T_A=-25 °C to 85 °C, unless otherwise specified)

Table 10.8. A/D Converter Characteristics

Symbol	Parameter	Test Conditions	Min	Typ	Max	Unit
Res	Resolution			8		bit
A_{TOT}	Total Accuracy (1)	F_{OSC} > 5 MHz F_{OSC} > 10 MHz F_{OSC} = 20 MHz		±1		LSB
t_C	Conversion Time	F_{osc} measured in MHz	82/F_{osc}		160/F_{osc}	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
AR_I	Analog Input Resistance	f_{OSC} = 20 MHz			1	k Ω
AC_{IN}	Analog Input Capacitance				25	pF

Notes: 1. Noise at 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:

Flags:	Z	set if result is zero, cleared otherwise.
	C	set 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	set if result is zero, cleared otherwise.
C	set if overflow, cleared otherwise.
S	set 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⇐ 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	set 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 ⇐ dst (7)** **dst (0) ⇐ 0** **dst (n+1) ⇐ 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	set if result is zero, cleared otherwise.
C	set if MSB was 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	set if result is zero, cleared otherwise.
C	not affected.
S	set if LSB was 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	set if result is zero, cleared otherwise.
	C	not affected.
	S	set 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 (MSB) and dst+1 (LSB) locations) is divided by the source. The LSB of the destination location pair (dst+1) is loaded with the result, the MSB (dst) is loaded with the remainder. In case of overflow the MSB and the LSB are loaded both with 255.

Flags:

Z	set if result is zero, cleared otherwise.
C	set if overflow, cleared otherwise.
S	set 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: **5**

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 - 16

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	set if result is zero, cleared otherwise.
C	set 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 C 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 labelled 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 labelled 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 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 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 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 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 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 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 $\Leftarrow$ 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 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 (000 01001b 11000100b) is labelled 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 ⇐ 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 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 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          00011011    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 ⇐ (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 RAM 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 (both 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          0010011     00011000            00001010
```

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

LDRR

Load RAM, RAM

Format:	**ldrr** dst, src
Operation:	**dst ⇐ 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, 16 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, 16 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	set 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	set 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]

WARNING: the page pointer is modified by the jump instructions (JP, CALL, JPC, etc.) so the PGSET instruction should be specified again before accessing an EPROM location if a jump instruction have been used after the last PGSET.

Flags:

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

Bytes: 1
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: 12

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)** **flag ⇐ 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 to stack value.
C	restored to stack value.
S	restored to stack value.

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 $\Leftarrow$ 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: 1
Cycles: 8

Example: If the interrupt 3 source (TIMER2) has generated an interrupt request remaining pending (being the interrupt masked or globally disabled), after the instruction

```
rint 3     00110001          00000011
```

the TIMER2 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	set 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 ⇐ 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	set if result is zero, cleared otherwise.
C	set if overflow, cleared otherwise.
S	set 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:	7 if GI already disabled, 16 otherwise	

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 Compiler 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, 16 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**

Example: The instruction:

waiti 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 $\Leftarrow$ 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 $\Leftarrow$ 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(0) ⇐ 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(0) ⇐ 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

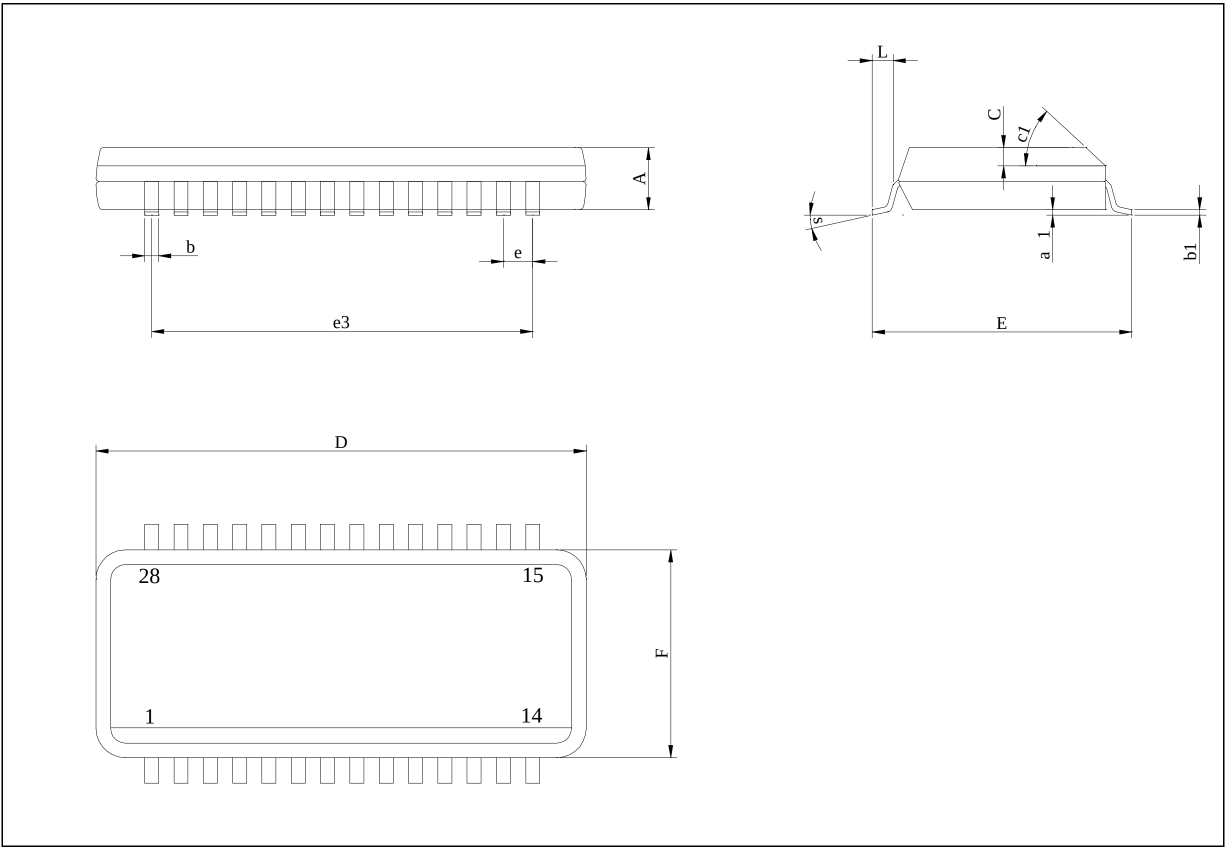
Format: **skm**

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.

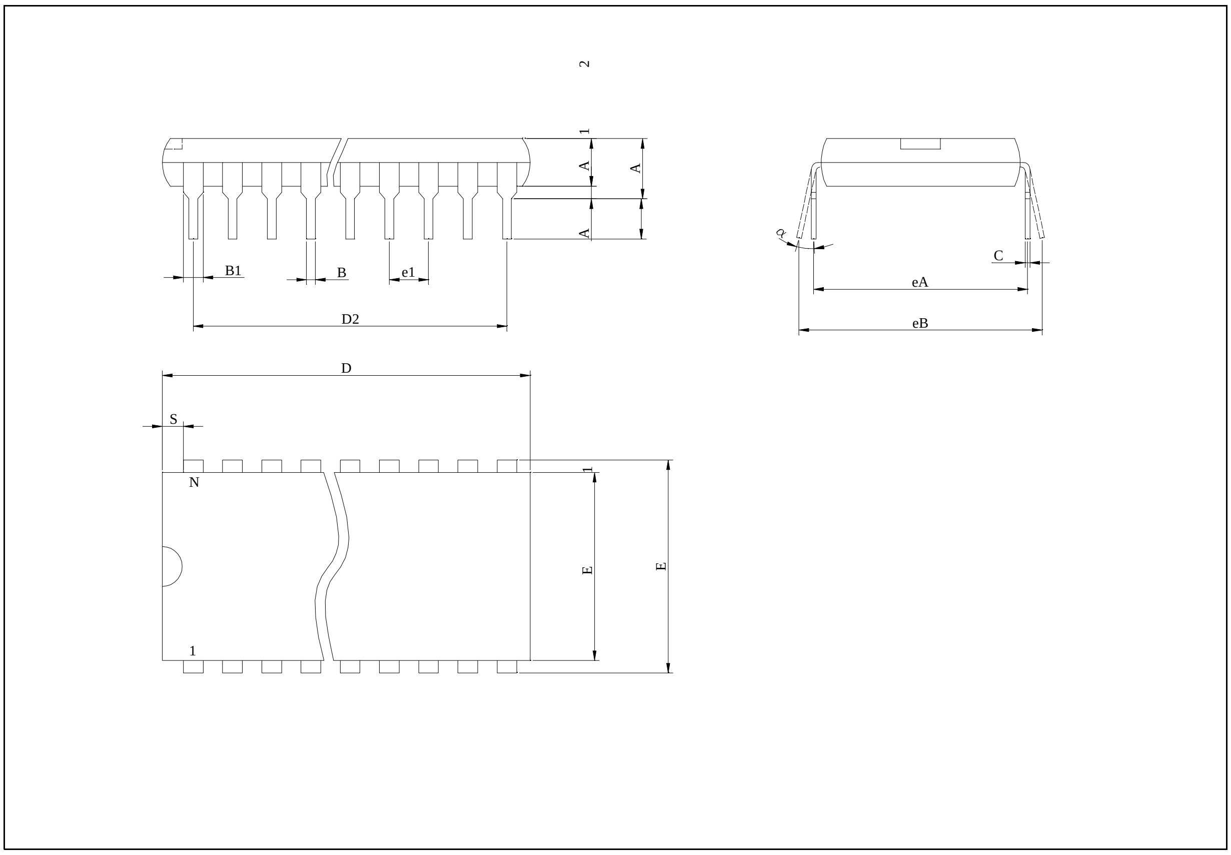
PSO28 PACKAGE MECHANICAL DATA

DIM	mm			inch.		
	MIN	TYP	MAX	MIN	TYP	MAX
A			2.65			0.104
a1	0.1		0.3	0.004		0.012
b	0.35		0.49	0.014		0.019
b1	0.23		0.32	0.009		0.013
C		0.5			0.020	
c1	45^o (typ.)					
D	17.7		18.1	0.697		0.713
E	10		10.65	0.394		0.419
e		1.27			0.050	
e3		16.51			0.65	
F	7.4		7.6	0.291		0.299
L	0.4		1.27	0.016		0.050
S	8^o (max)					



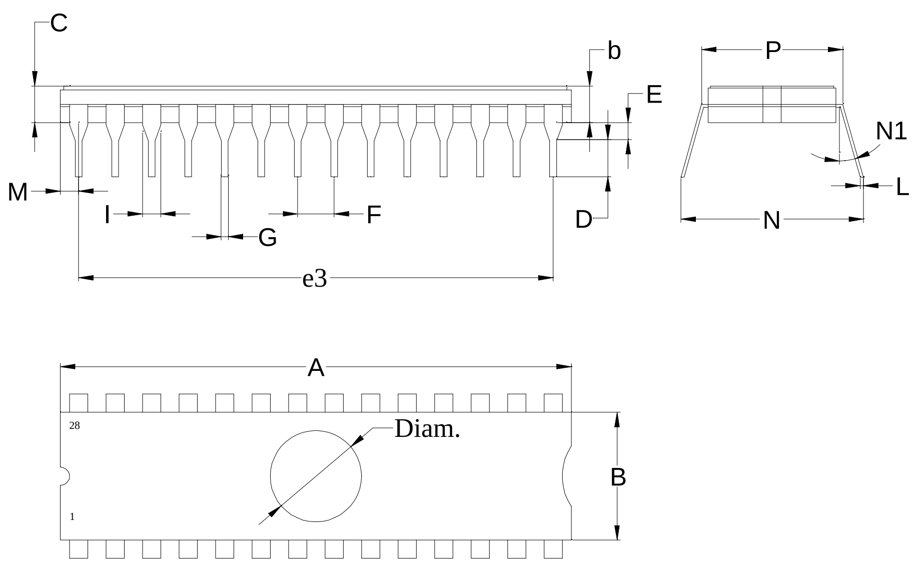
PLASTIC DIP28 PACKAGE MECHANICAL DATA

DIM	mm			inch.		
	MIN	TYP	MAX	MIN	TYP	MAX
A			5.08			0.200
A1		0.38			0.015	
A2		3.56	4.06		0.140	0.160
B		0.38	0.51		0.015	0.020
B1	1.52			0.060		
C		0.20	0.30		0.008	0.012
D		36.83	37.34		1.450	1.470
D2	33.02			1.300		
E	15.24			0.600		
E1		13.59	13.84		0.535	0.545
e1	2.54			0.100		
eA	14.99			0.590		
eB		15.24	17.78		0.600	0.700
L		3.18	3.43		0.125	0.135
S		1.78	2.08		0.070	0.082
α		0^O	10^O		0^O	10^0
N	28			28		



CERAMIC DIP28 WINDOWED PACKAGE MECHANICAL DATA

DIM	mm			inch		
	MIN	TYP	MAX	**MIN**	**TYP**	MAX
A			38.10			1.469
B	13.05		13.36	0.514		0.526
C	3.90		5.08	0.153		0.177
D	3.18			0.125		
E	0.50		1.78	0.020		0.070
e3		33.02			1.300	
F	2.29		2.79	0.90		0.110
G	0.40		0.55	0.18		0.22
I	1.17		1.42	0.48		0.58
L	0.22		0.31	0.010		0.012
M	1.52		2.49	0.060		0.098
N	16.17		18.32	0.637		0.721
N1	4d		15d			
P	15.40		15.80	0.606		0.616
Q			5.71			0.225
Diam.	6.86		7.36	0.275		0.285



ORDERING INFORMATION

PART NUMBER	MEMORY SIZE	TEMPERATURE RANGE	PACKAGE
ST52T420G0M6	1 kb	-25 to +85°C	PSO28
ST52T420G1M6	2 kb	-25 to +85°C	
ST52T420G2M6 ST52T420B/M	4 kb	-25 to +85°C	
ST52E420B/D	4 kb	-25 to +85°C	CDIP28W
ST52T420G0B6	1 kb	-25 to +85°C	PDIP28
ST52T420G1B6	2 kb	-25 to +85°C	
ST52T420G2B6 ST52T420B/B	4 kb	-25 to +85°C	
ST52x420/KIT			DEVELOPMENT KIT