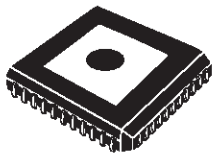
ST52T301/E301

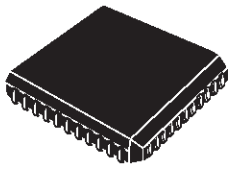
8-Bit OTP/EPROM *DuaLogic*™ MCUs WITH ADC, UART, TIMER, TRIAC & PWM DRIVER

ADVANCED DATA SHEET

- High Speed dedicated structures for Fuzzy Logic (3.5 µs to compute a 4 In x 1 Out rule)
- Capability to perform simple boolean and arithmetic operations
- Up to 4 Input, 2 Output Configurable Variables for each Fuzzy Algorithm and up to 300 Rules
- Up to 16 Triangular and Trapezoidal Membership Functions for each Input variable
- Up to 256 Singleton Membership Functions for all Consequents
- Program and Data EPROM: 2 Kbytes
- 16 general purpose registers available as Register File
- Working Clock Frequencies: 5, 10 and 20MHz
- On-Chip Clock Oscillator driven by Quartz Crystal or Ceramic Resonator
- One external interrupt
- Standard TTL compatible input
- CMOS compatible output
- 4 channel 8 bit Analog to Digital Converter
- Bandgap reference 2.5V
- Digital 8 bit I/O port indepedently programmable with handshake signal
- Serial Communication Interface with asynchronous protocol (UART)
- Programmable Timer with internal Prescaler
- Internal Power Fuzzy Control to drive external Triac (up to 25mA source, 50 mA sink current)
- Internal Fuzzy controlled PWM to drive an external power device
- Software tools and Emulators availability
- Windowed and One Time Programmable (OTP) Memory parts available for prototyping and production phases
- 44 pin Plastic (PLCC44) and Ceramic Windowed Leaded Chip Carrier (CLCC44-W)



CLCC44-W



PLCC44

1.1 GENERAL DESCRIPTION

ST52E301 (1) and ST52T301 (1) devices are members of the W.A.R.P. family of 8-bit *DuaLogic*™ microcontrollers. They are able to perform, in an efficient way, both boolean and fuzzy algorithms, in order to reach the best performances that the two methodologies allow.

The ST52E301 is the erasable EPROM version and the ST52T301 is the OTP version.

The ST52x301 is completely developed and produced by STMicroelectronics using the reliable high performance CMOSM5E (O.7µm) process.

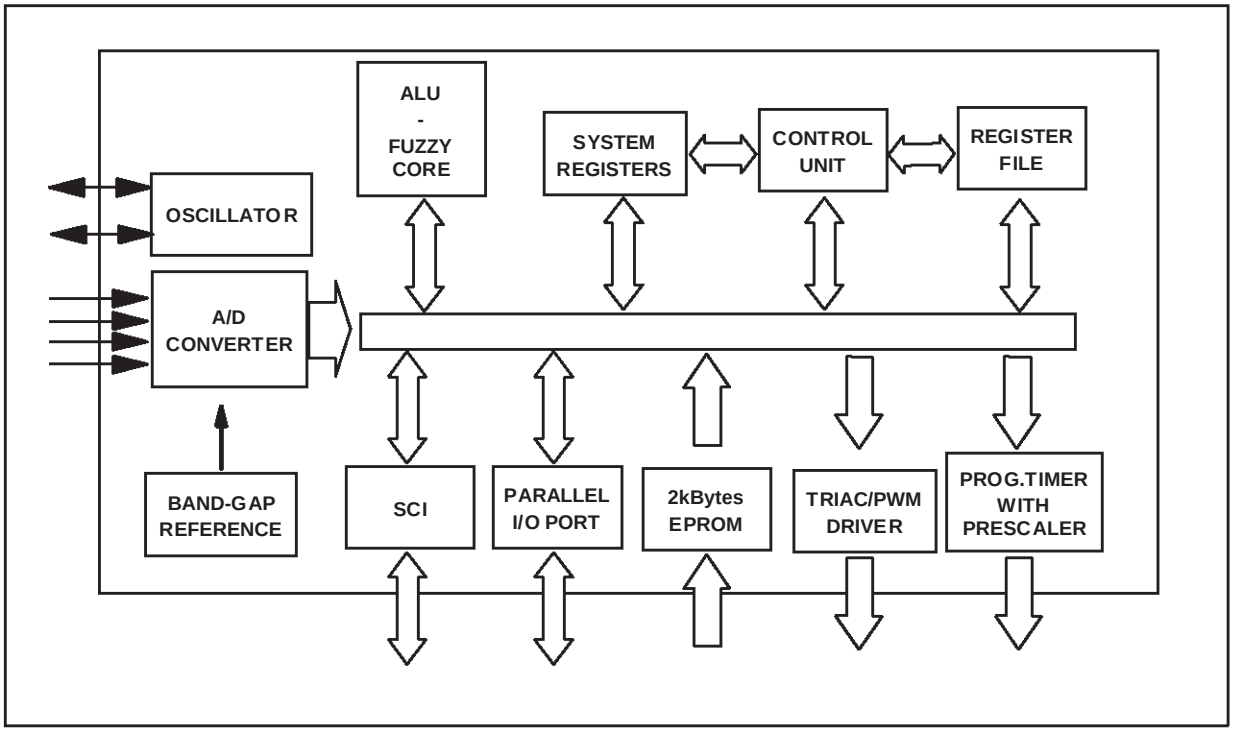
Thanks to Fuzzy Logic, ST52x301 allows to describe a problem using a linguistic model instead of a mathematical model. In this way it is very useful and easy to modelize complex system with very high accuracy.

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.

Note: (1) Formerly W.A.R.P.3TC

Figure 1. ST52x301 Architectural Block Diagram



The flexible I/O configuration of ST52x301 allows to interface with a wide range of external devices, like D/A converters, power control devices (SCRs, TRIACs) and external sensors.

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

The Fuzzy Core includes the fuzzifier (ALPHA calculator), the inference unit and the defuzzifier.

It allows to manage up to 300 Rules (4 Inputs and 1 Output). The rules could be shared in different fuzzy subroutines that can be activated by user defined conditions.

The I/O capabilities, demanded from microcontroller applications, are fulfilled by ST52x301 with 4 Analog Inputs, an asynchronous Peripheral interface (UART) and an 8-bit I/O communication port in order to transfer data from/to the on-chip Register File.

The voltage reference provides biasing to the analog portion of the internal circuitry. The internal reference is a 2.5V Bandgap reference.

The voltage reference can supply up to 0.1 mA of current to power external circuitry.

ST52x301 includes an 8-bit sampling Analog to Digital (A/D) Converter with a 4 analog channel fast multiplexer (32µs conversion time/channel).

It is possible to perform operations on data stored in the Register File (16 bytes), allowing to manage new inputs and feedback outputs.

The TRIAC/PWM Driver peripheral allows to manage directly power devices, implementing three different operating modes: Burst Mode (i.e.Thermal Applications), Phase Angle Partialization (i.e. Motors Control by TRIACs) and high frequency PWM controls.

A programmable Timer with Internal Prescaler, using both internal or external clock, is available.

The microcontroller configuration is stored in the internal EPROM.

A powerful development environment, FUZZYSTUDIO™ 3.0, consisting of a board and software tools, allows an easy configuration and use of ST52x301.

ST52x301 is fully supported by FUZZYSTUDIO3.0 software tools allowing to graphically design a project and obtain an optimized microcode.

ST52x301 exploits a SGS-THOMSON patented strategy to store the MFs in its internal memory.

Figure 2. CLCC44-W Pin Configuration

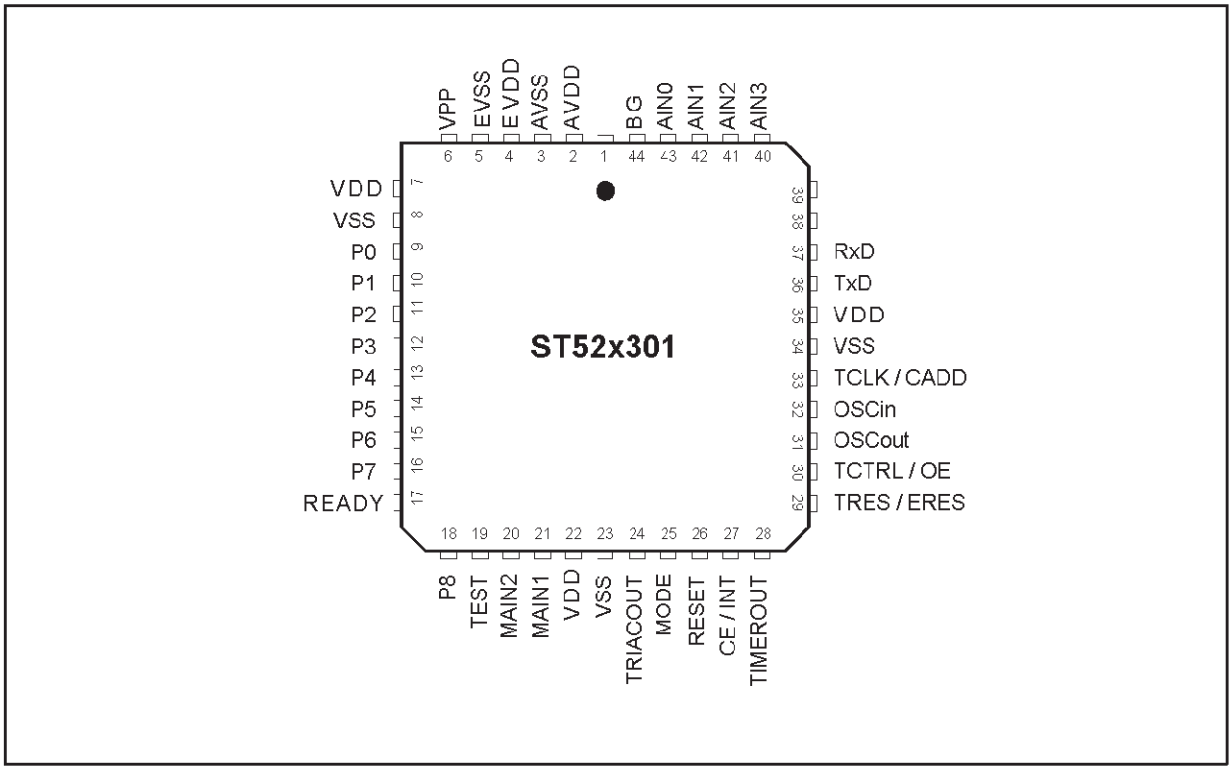


Figure 3. PLCC44 Pin Configuration

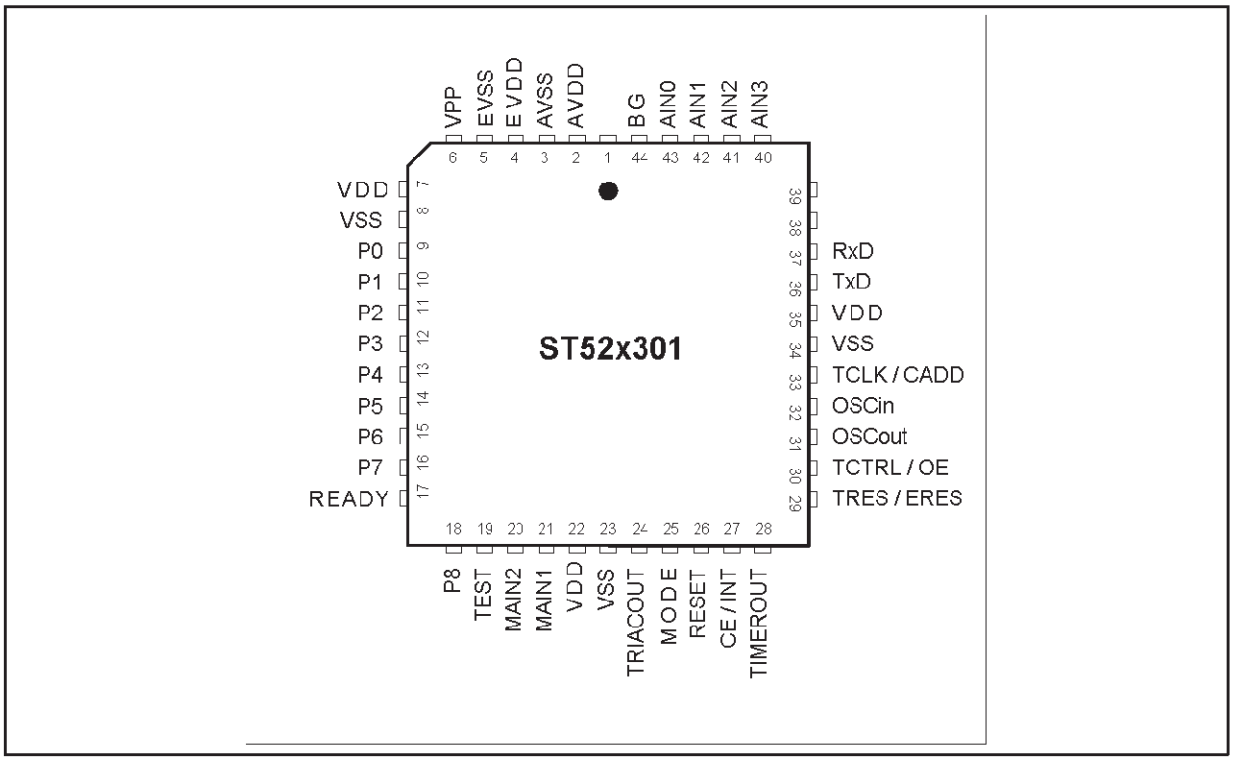


Table 1. PLCC44 and CLCC44-W Pin Configuration

PIN	NAME	TYPE	Programming Phase	Working Phase
1	not connected		-	-
2	AV_{DD}		Analog V_{DD}	Analog V_{DD}
3	AV_{SS}		Analog Ground	Analog Ground
4	EV_{DD}		EPROM Digital Power Supply	EPROM Digital Power Supply
5	EV_{SS}		EPROM Digital Ground	EPROM Digital Ground
6	V_{PP}		EPROM Programming Power supply (12V ±5%)	EPROM V_{DD} (5V ±10%)
7	V_{DD}		Digital Power Supply	Digital Power Supply
8	V_{SS}		Digital Ground	Digital Ground
9	P0	I/O	I/O EPROM Data	Digital I/O
10	P1	I/O	I/O EPROM Data	Digital I/O
11	P2	I/O	I/O EPROM Data	Digital I/O
12	P3	I/O	I/O EPROM Data	Digital I/O
13	P4	I/O	I/O EPROM Data	Digital I/O
14	P5	I/O	I/O EPROM Data	Digital I/O
15	P6	I/O	I/O EPROM Data	Digital I/O
16	P7	I/O	I/O EPROM Data	Digital I/O
17	READY	O		I/O port Handshaking Signal
18	P8	O		Digital Output
19	TEST	I	(must be set to 0)	(must be set to 0)
20	MAIN2	I/O		Zero Crossing/Prescaler Output
21	MAIN1	I		Zero Crossing
22	V_{DD}		Digital Power Supply	Digital Power Supply
23	V_{SS}		Digital Ground	Digital Ground
24	TRIACOUT	O		Triac/PWM Driver Output Pulses
25	MODE	I	Functionment Mode Selector	Functionment Mode Selector
26	RESET	I	General Reset	General Reset
27	CE/INT	I	Chip Enable EPROM	External Interrupt
28	TIMEROUT	O		Output Timer
29	ERES / TRES	I	EPROM Address Counter Reset	External Timer Reset
30	OE / TCTRL	I	EPROM Output Enable	Timer Start/Stop Signal
31	OSCout	I/O	Oscillator Output	Oscillator Output
32	OSCin	I	Oscillator Input	Oscillator Input
33	CADD / TCLK	I	EPROM Change Address Clock	Timer External Clock
34	V_{SS}		Digital Ground	Digital Ground
35	V_{DD}		Digital Power Supply	Digital Power Supply
36	TxD	O		SCI Output
37	RxD	I		SCI Input
38	not connected		-	-
39	not connected		-	-
40	AIN3	Ainp		Analog Input
41	AIN2	Ainp		Analog Input
42	AIN1	Ainp		Analog Input
43	AIN0	Ainp		Analog Input
44	BG	Aout		Band Gap Output

1.2 PIN DESCRIPTION

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

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

V_{SS}. Digital circuit Ground.

EV_{DD}. EPROM Main Power Supply Voltage (5V 10%).

EV_{SS}. EPROM Digital circuit Ground.

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

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

V_{PP}. Main Power Supply for the internal EPROM (12.5V 5%).

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 ST52x301 with various stability/cost trade-offs. An external clock signal can be applied to OSCin, in this case OSCout must be grounded.

RESET. This signal is used to restart ST52x301 at the beginning of its program. It also allows to select the program mode for the EPROM.

INT. External interrupt active on rising or falling edge.

AIN0-AIN3. These 4 lines are connected to the inputs of the analog multiplexer. They allow to acquire 4 analog inputs.

BG. A Voltage equal to 2.5V is available on this pin. It can be used for Analog signal conditioning.

P0-P7. These 8 lines are organized as one I/O port. During the Programming phase such port is used for the EPROM data read/write.

READY. Handshake signal of the parallel port.

P8. Digital output.

TxD. Serial data output of the SCI transmitter block.

RxD. Serial data input of the SCI receiver block.

TRES, **TCLK**, **TCTRL**, **TIMEROUT**. These pins are related with the internal Programmable Timer. The Timer can be reset externally by using TRES. In Working Mode, TRES resets the address counter of the Timer. TRES is active at low level

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

An external Start/Stop signal can be used to control the Timer through the pin TCTRL. The Timer output is available on the pin TIMEROUT.

MAIN1, **MAIN2**, **TRIACOUT**. ST52x301 is able to drive a TRIAC in two different modes: Burst mode or Phase Angle Partialization control mode.

The Burst mode is used for thermal regulation.

MAIN1 and MAIN2 signals are used to detect the zero crossing of the main voltage.

The pulse to drive the TRIAC is given by TRIACOUT pin.

It is possible to use the same pins to implement a PWM Driver. In this case it is possible to fix the period of PWM and to change the duty cycle on fly. The PWM output is given by TRIACOUT pin.

CE, **OE**, **ERES**, **CADD**, **V_{PP}**. These pins are used to manage the EPROM during the Programming phase. During the Programming phase (programming) V_{PP} must be set at 12V. In the Working phase V_{PP} must be equal to **V_{DD}**.

ERES in Programming Mode resets the address counter of the EPROM; it is active at high level.

In the Working phase OE, CE and CADD are used like handshaking signals for the parallel port.

MODE. It selects the functionment mode (Programming or Working mode).

TEST. It enables the testing functionalities; during the Programming and Working phase it must be set to 0.

2 INTERNAL ARCHITECTURE

ST52x301 is made up by the following blocks and peripherals:

- Control Unit
- Fuzzy Core
- ALU
- EPROM
- Clock Oscillator
- Analog Multiplexer and A/D Converter
- Prescaler Timer
- Bandgap
- Triac / PWM Driver
- Digital I/O port
- Serial Communication Interface

ST52x301 Operating Modes

ST52x301 works in two modes, Programming and Working Modes, depending on the control signals level RESET, TEST and MODE.

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

2.1 CONTROL UNIT

The Control Unit (CU) manages: Registers File, Input Registers, Configuration Registers, ALU, Accumulator and Multiplexer inputs. Moreover the CU drives the Fuzzy Core and the peripherals (Triac/PWM Driver and Timer).

The CU reads the stored instructions on the EPROM (Fetch) and decodifies them. If the instructions are arithmetic or logic, the CU runs them directly, sending the control signals to the related blocks. If there is a STOP instruction, the CU transfers the control to the Fuzzy Core.

The Fuzzy Core (FC) will read the next instruction (that must be a fuzzy instruction) from the EPROM. The FC mantains the control of the program until the next STOP instruction. Then the FC transfers the control to the CU.

These characteristics allow to mix fuzzy algorithms with mathematical and logic instructions.

Figure 2.1 shows a flow-chart reasuming the logic behaviour of the instructions management.

Table 2.1. Control Signals setting

Control Signal	Programming	Reset	Working
RESET	0	0	1
TEST	0	0	0
MODE	1	0	0

Figure 2.1. Computation Algorithm Flow Chart

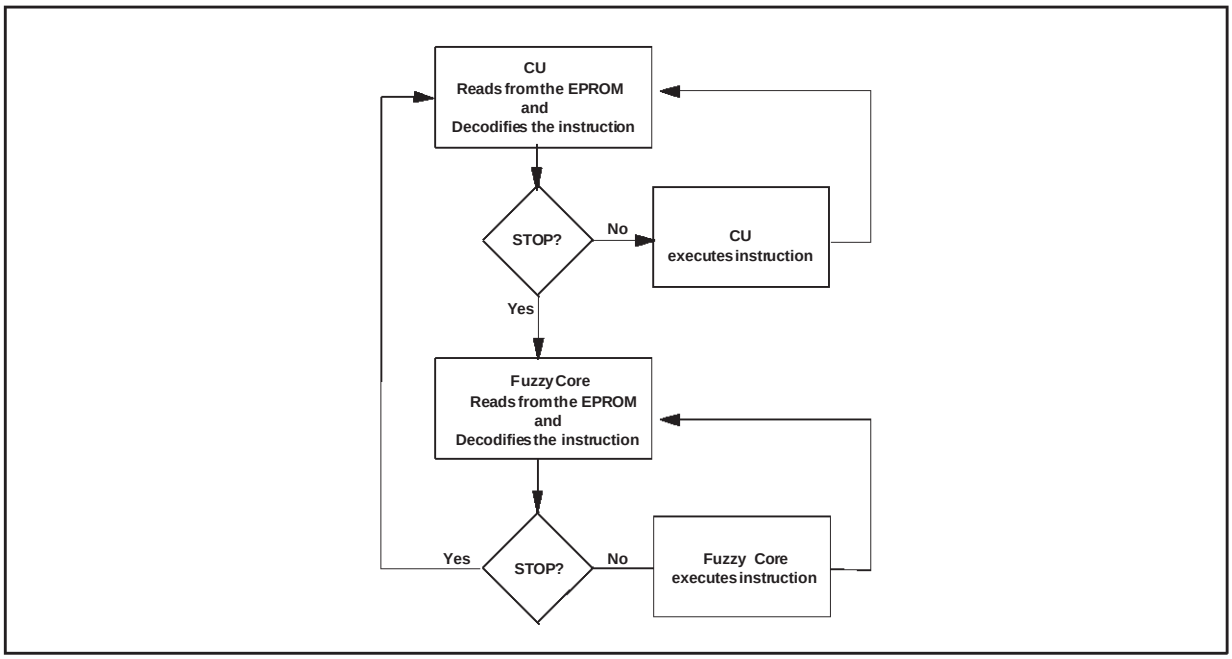
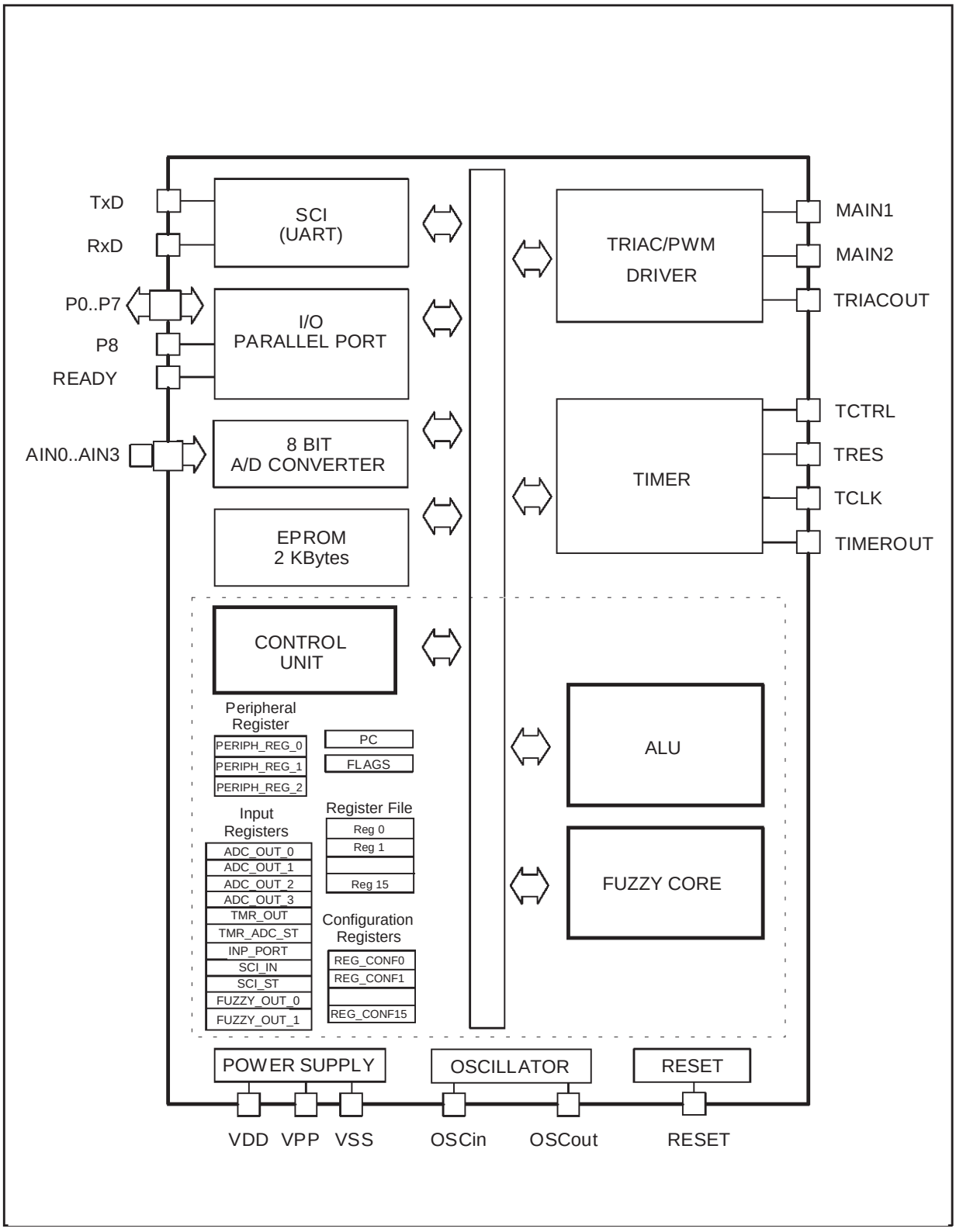


Figure 2.2. ST52x301 Block Diagram



It is not possibile to stop the fuzzy inference before the end of the defuzzification of one output. A set of 26 different arithmetic and logic instructions is available. Each instruction requires from 4 to 7 clock pulses to be performed.

2.1.1 Program Counter

The Program Counter (PC) is a 11-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 11-bit length allows the direct addressing mode of 2048 bytes in the program space.

After having read the current instruction address, the PC value is incremented. To execute relative jumps the PC and the offset are shifted through the Fuzzy Core or the ALU, where they will be added. 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 ST52x301 core includes two pairs of flags that correspond to 2 different modes: normal mode and interrupt mode. Each pair consist of a CARRY flag and a ZERO flag. One pair (CN, ZN) is used during normal operation and one is used during the interrupt mode (CI, ZI).

The ST52x301 core uses the pair of flags that correspond to the actual mode: as soon as an interrupt is generated, the ST52x301 core uses the interrupt flags instead of the normal flags. When the RETI instruction is executed the normal flags are restored if the MCU was in the normal mode before the interrupt. It should be observed that each flag set can only be addressed in its own routine.

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

The Carry flag is set when a carry or a borrow occurs during arithmetic operations, otherwise it is cleared.

The switching between the two sets of flags is automatically performed when an interrupt or a RETI instruction occur.

2.2 ADDRESS SPACES

W.A.R.P3TC has four separate address spaces:

- Register File: 16 8-bit registers
- Input Registers: 11 8-bit registers
- Configuration Registers: 16 8-bit registers
- Peripheral Registers: 3 8-bit registers
- Program memory up to 2K Bytes

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

2.2.1 Register File

The Register File (RF) consists of 16 general purpose 8-bit registers Reg0 to Reg15.

All the registers in the RF can be specified by using a decimal address,

e.g. 0 identify the first register of the RF, called Reg0.

Reg0:3 are directly connected to the FC input. It means that the input values of the fuzzy algorithm must be loaded into these registers by the user.

These registers are used as temporary registers during the macros' computation.

Figure 2.3. Address Spaces description

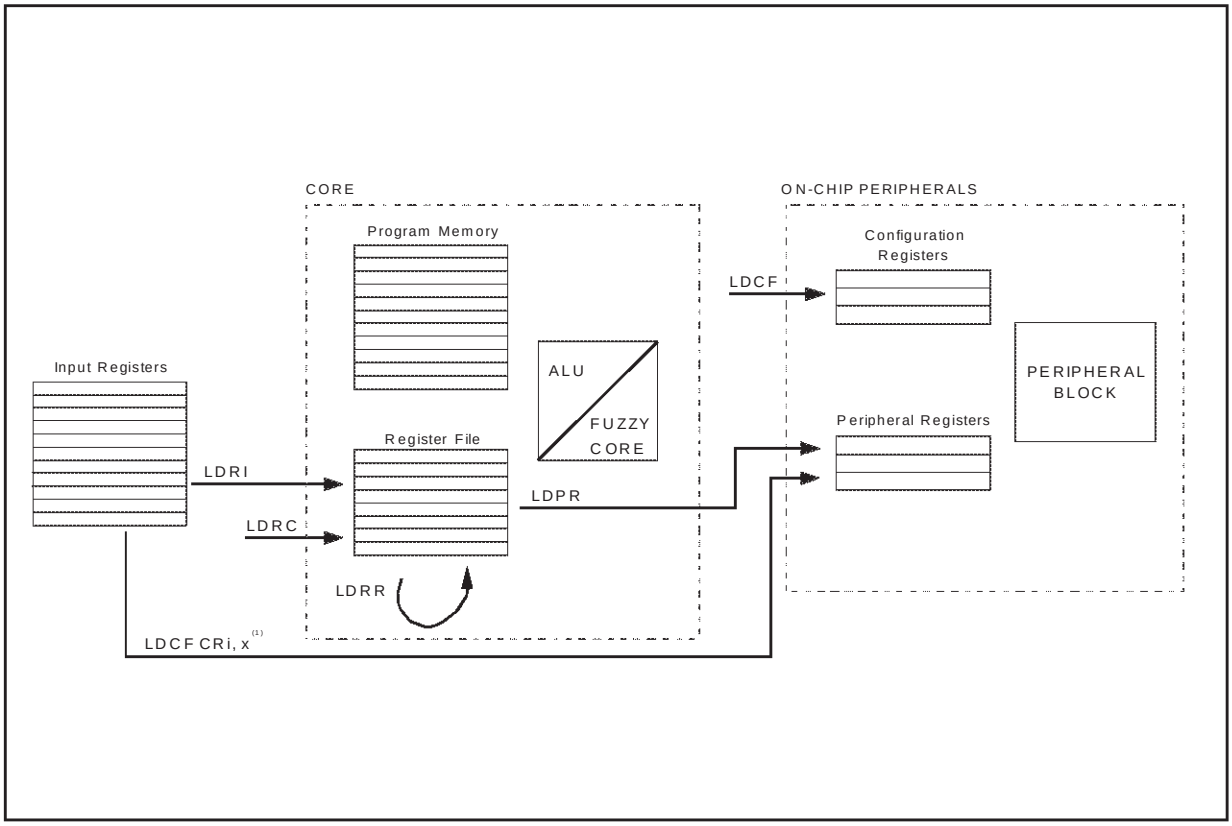
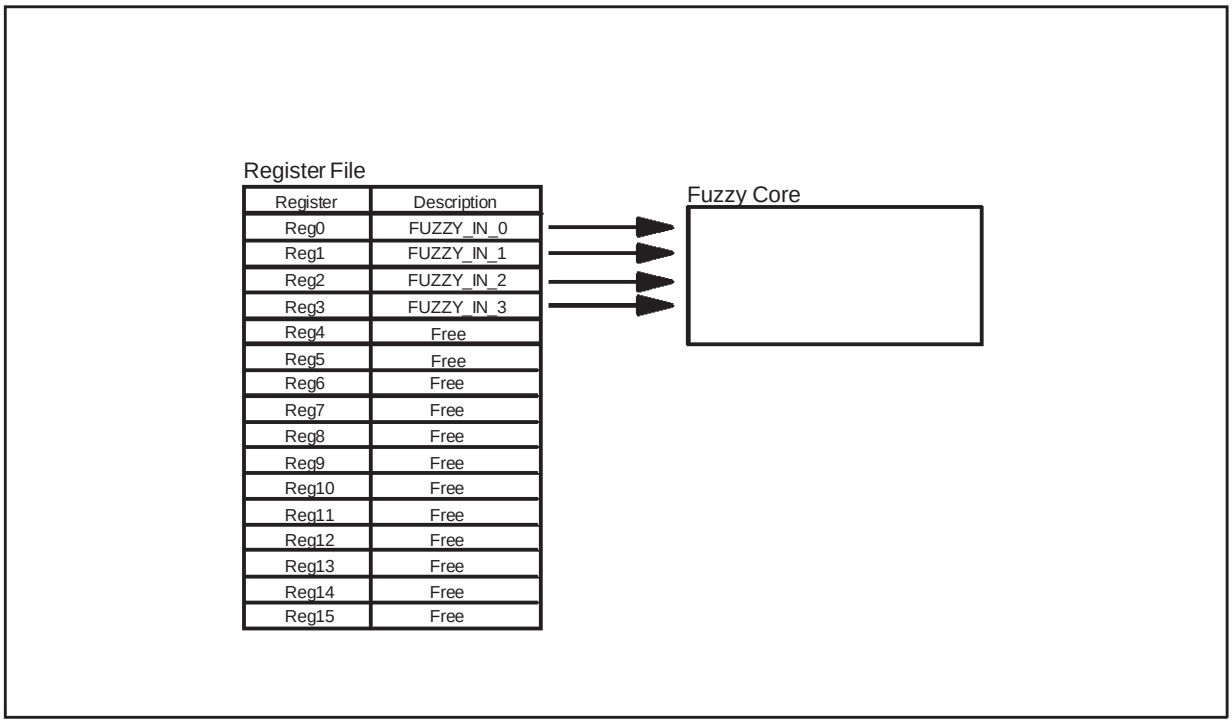


Figure 2.4. Register File description



2.2.2 Input Registers Bench

The Input Registers (IR) bench consists of 11 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 first four registers (ADC_OUT_0:3) of the IR are dedicated to the 4 converted values coming from the ADC.

TMR_OUT registers contains the current counted value by the internal Timer; whereas TMR_ST is the Timer status. For details about TMR_ST, please refer to Timer description.

Data read by the Parallel I/O Port are stored automatically in the 6-th register, INP_PORT.

Data read by the SCI are stored automatically in the 7-th register SCI_IN and SCI status is stored in the SCI_ST register. For details about SCI_ST, please refer to SCI description.

The Fuzzy Core writes the computed output values in the FUZZY_OUT_0:1 registers.

Figure 2.5. Input Registers Bench description

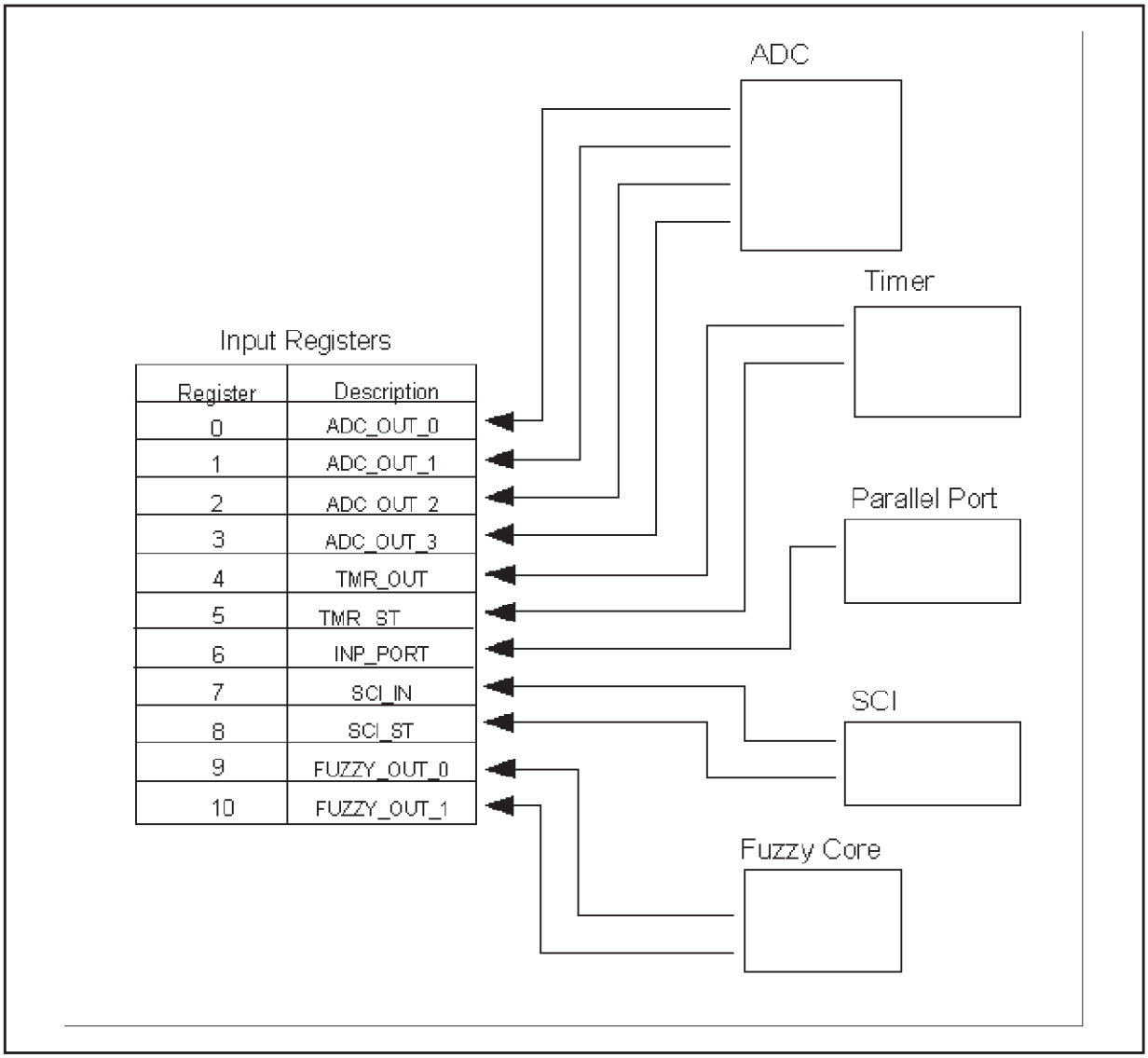


Figure 2.6. TMR_ADC_ST Registers

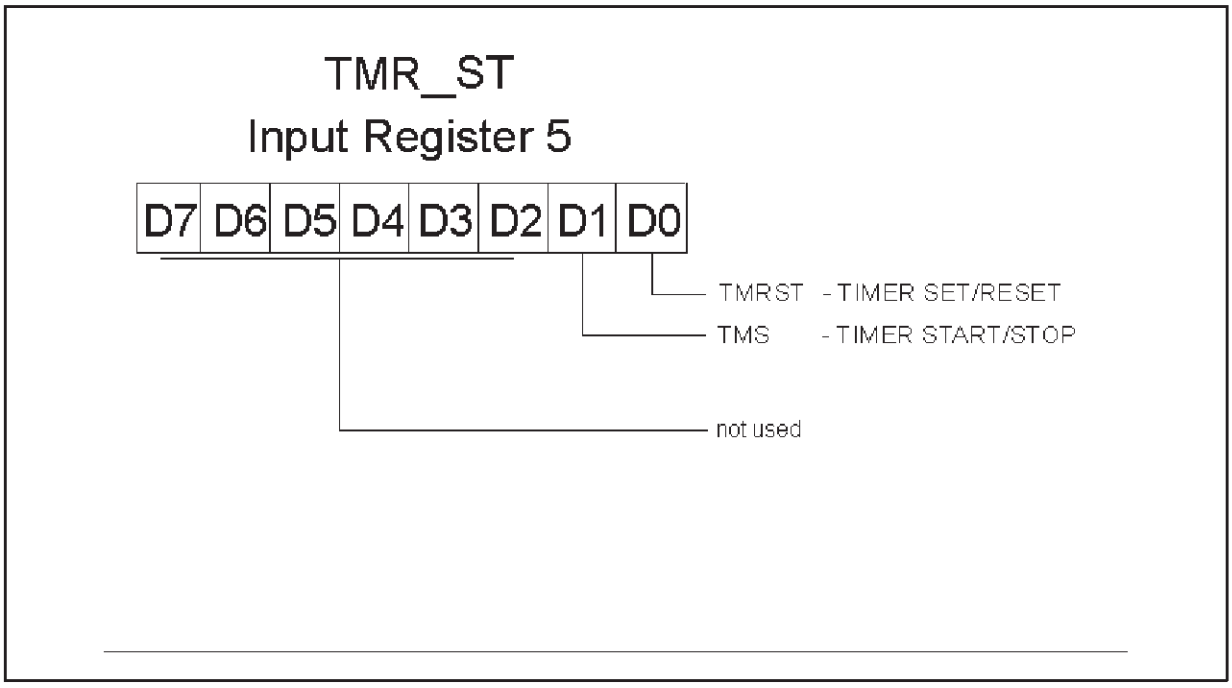
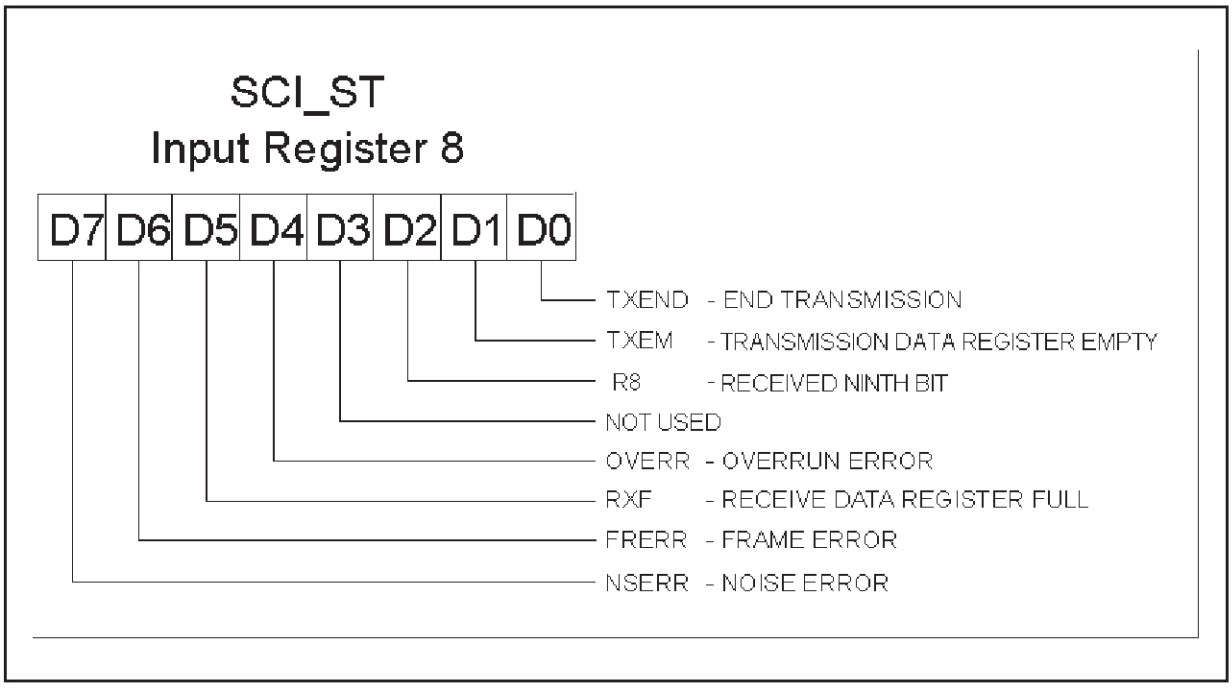


Figure 2.7. SCI_ST Registers



2.2.3 Configuration Registers

The ST52x301 setting permits to configure all blocks. Table 2.2 describes the related block to each bit of the Configuration Registers.

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

Table 2.2. Configuration Registers description

Register	Peripheral	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
REG_CONF0	**PARALLEL PORT**	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0
REG_CONF1	**SCI, CORE, I/O PORT**	RDRF	OVR	BRK	TDRE	TXC	ECKF		P8 OUT
REG_CONF2	**ADC**	not used				IADD		1	ADRST
REG_CONF3	**SCI**	BRSL			T8	M		RE	TE
REG_CONF4	**TIMER**	TMLSB							
REG_CONF5	**TIMER**	TMMSB							
REG_CONF6	**TIMER**	not used	POL	TMS	CKSL	TMEL	IESL	TMST	TMRST
REG_CONF7	**TIMER**	not used			FZSL	INPSL	INTR	INTF	INTSL
REG_CONF8	**TRIAC**	TCLSB							
REG_CONF9	**TRIAC**	TCMSB							
REG_CONF10	**TRIAC**	IOSL	PSF	CKSL		MODE		TCST	TCRST
REG_CONF11	**TRIAC**	INTSL		TCMSK				TCTRS	POL
REG_CONF12	**TRIAC**	FZSL	INPSL	UTPMSB					
REG_CONF13	**TRIAC**	UTPLSB							
REG_CONF14	**INTERRUPT**	EXTI	not used		MSKTC	MSKTM	MSKSCI	MSKAD	MSKE
REG_CONF15	**INTERRUPT**	INT4		INT3		INT2		INT1	

2.2.4 Peripheral Registers

Peripheral Registers contain the initialization values for the Timer, Triac/PWM Driver and Parallel Port.

The peripheral initialization value is kept from a location of the Register File, by using a LDPR instruction, or from FUZZY_OUT_0/1 Input Register according with the related Configuration Registers.

Table 2.3 describes the related peripheral to each Configuration Register.

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

Table 2.3. Peripheral Register description

Peripheral Register	Peripheral
PERIPH_REG_0	Timer
PERIPH_REG_1	Triac/PWM Driver
PERIPH_REG_2	Parallel Port

2.4 FUZZY CORE

ST52x301 Fuzzy Core main features are:

- Up to 4 Inputs with 8-bit resolution
- Up to 16 Membership Functions (Mbfs) for each Input (64 possible Mbfs)
- Up to 2 Outputs with 8-bit resolution
- Possibility to process fuzzy rules with a max. number of 8 antecedents

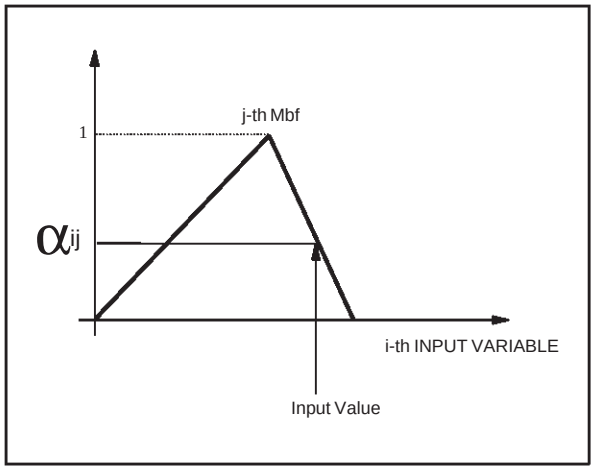
2.4.1 Internal Structure

The block diagram shown in figure 2.9 describes the structure of ST52x301 Fuzzy Core. In this figure we can distinguish different functional blocks: Alpha Calculator, Inference Unit and Defuzzifier. These blocks allow to perform a MAMDANI type fuzzy inference with crisp consequents. It is important to underline that the fuzzy inference is performed by using as inputs the first 4 locations of the Registers File.

2.4.2 Alpha Calculator Unit

This block performs the intersection (alpha weight) between the input values and the related Mbfs (fig. 2.8).

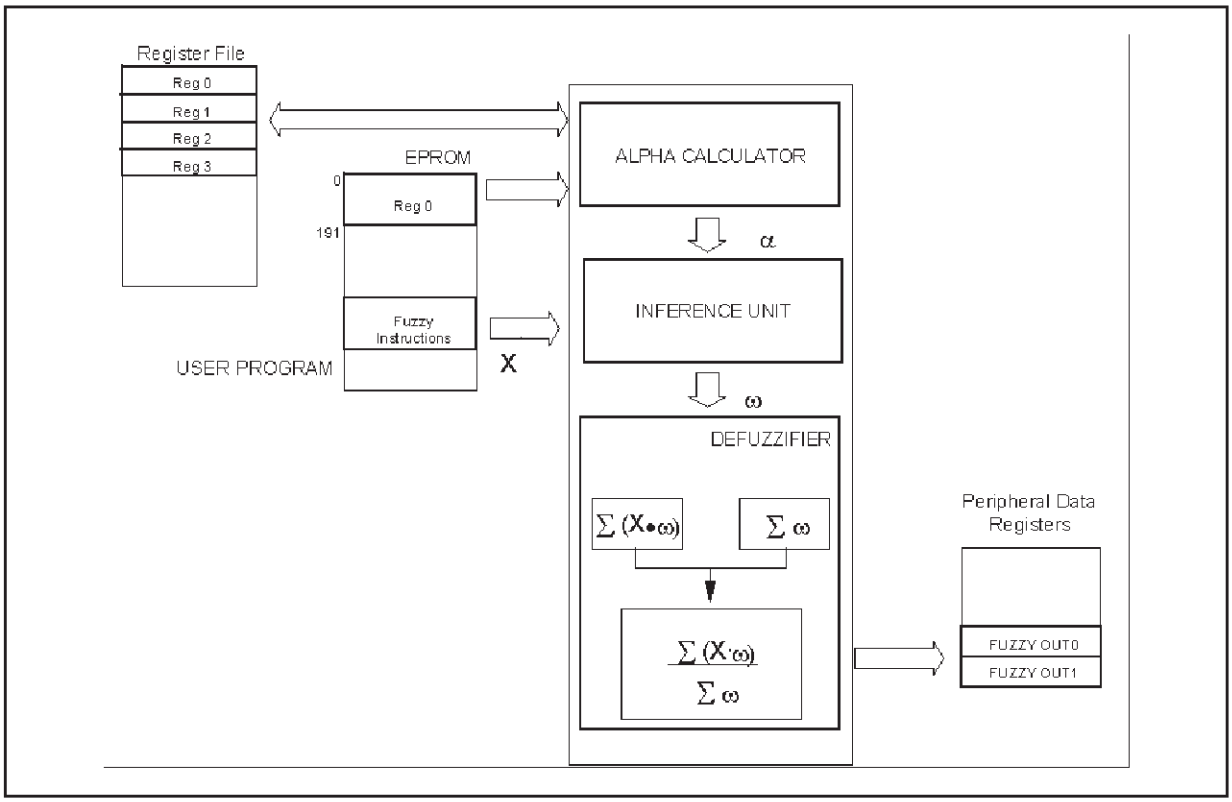
Figure 2.8. Alpha Weigth calculation



Notice that the inputs for this block come from the first four locations of the Register File; it means the user, to evaluate a fuzzy function, must load the input values in these registers.

Alpha Calculator performs what is called **fuzzification**: the input data are transformed in activation level (alpha weight) of the Mbfs.

Figure 2.9. Fuzzy Core Block Diagram



2.4.3 Inference Unit

It manages the alpha weights obtained by the Alpha Calculator Unit 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 up to eight conditions by linguistic connectives AND/OR, NOT operator and brackets.

Each rule can have at maximum 8 alpha weights (however they are connected).

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

Figure 2.10.

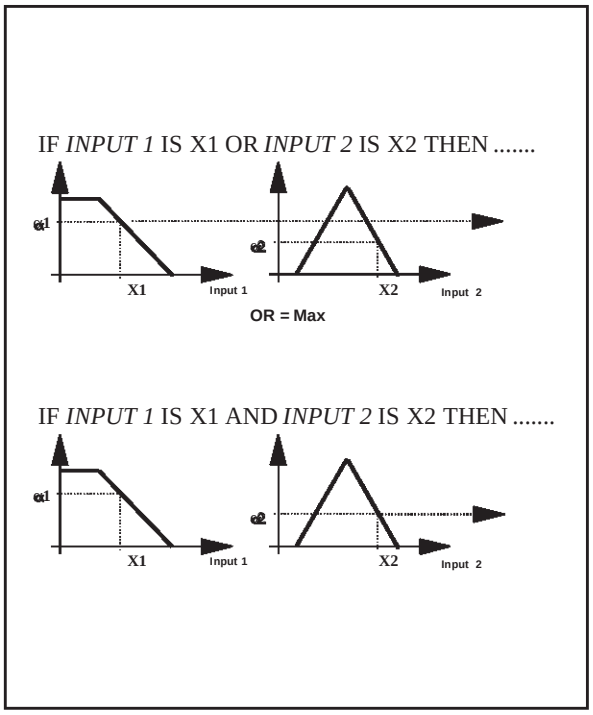
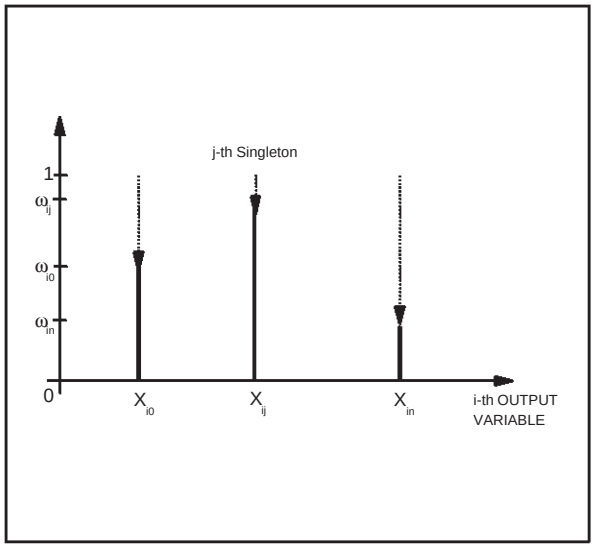


Figure 2.11.



2.4.4 Defuzzifier

This block consists of a Multiplier, two Adders and one Divider. It generates the output crisp values implementing the consequent part of the rules.

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

Each output value (FUZZY_OUT0, FUZZY_OUT1) is deduced from the consequent crisp values (X_i) by using the defuzzification formula:

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

where:

i = 0,1 identifies the current output variable

N = number of the active rules on the current output

ω_{ij} =weigth of the j-th singleton

Xij = abscissa of the j-th singleton

The two fuzzy outputs are stored in the location 9 and 10 of the Input Registers (FUZZY_OUT_0, FUZZY_OUT_1).

2.4.5 Input Membership Function

ST52x301 allows to manage triangular Mbfs. In order to define a Mbf it is necessary to store three different data on the memory:

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

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

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

In order to reduce the dimension 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 Memberships Functions. In the following figure is shown a typical example of Mbfs that can be defined in ST52x301

Each Mbf is then defined storing 3 bytes. To store all the information related with the fuzzy project Mbfs, it is necessary to use 192 bytes of the memory (3 bytes*16 Mbfs*4 Inputs = 192 bytes).

The Mbf is memorized by using the following instruction:

DATA n m lvd v rvd

where

n identifies the input, m identifies the Mbf among the 16 possible Mbfs, lvd, v, rvd are the parameters describing the Mbf's shape.

2.4.6 Output Singleton

ST52x301 uses for the output variables a particular kind of membership function called Singleton. A Singleton has not a shape, like a traditional Mbf, and it is characterized by a single point identified by the couple (X, 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 2.12. Mbfs Parameters

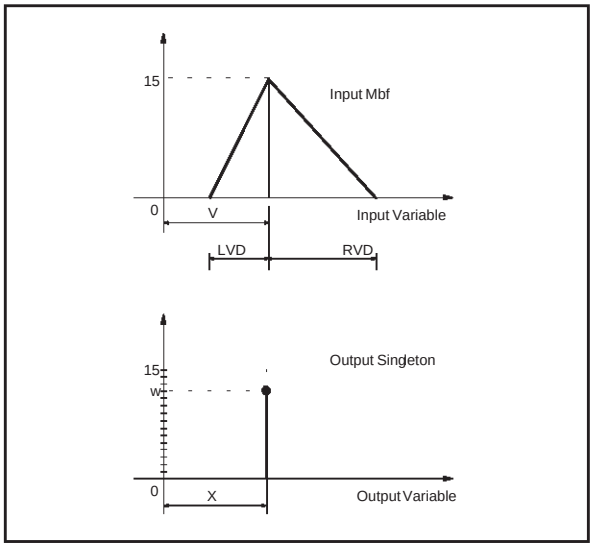
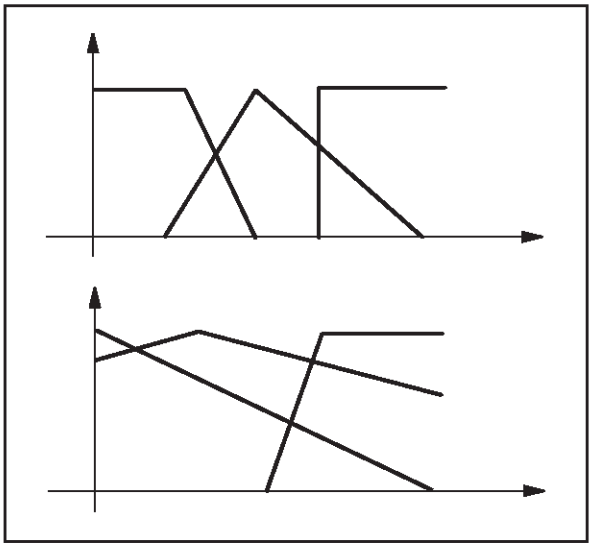


Figure 2.13. Example of valid Mbfs



2.4.7 Fuzzy Rules.

The rules can have the following structures:

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

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

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

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

Each rule is codified by using an istruction 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 are reported in the following table:

Table 2.4. Fuzzy Instructions Set

Instruction	Description
DATA n m lvd v rvd	Stores the Mbf m of the input n with the shape identified by the parameters lvd, v and rvd.
LDP n m	Fixes the alpha value of the input n with the Mbf m and stores it in the data stack.
LDN n m	Calculates the negated alpha value of the input n with the Mbf m and store the result in the data stack.
FZAND	Implements the fuzzy operation AND between the last two values stored in the data stack.
FZOR	Implements the fuzzy operation OR between the last two values stored in the data stack.
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 the memory register M.
LDM	Copies the value of the register M in the data stack.
CON crisp	Multiplies the crisp value with the last ω weight.
OUT n_out	Performs the defuzzification.
STOP	Ends the fuzzy algorithm.

Example 1:

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

is codified by the following instructions

LDN 1 1	calculates the NOT α value of $Input_1$ with Mbf_1 and stores the result in the data stack
LDP 4 12	fixes the α value of $Input_4$ with M_{12} and stores the result in the data stack
FZAND	adds the NOT α and α values obtained with the operations LDN1 1 and LDP 4 12
LDK	stores the result of the operation FZAND in the data stack
LDP 3 8	fixes the α value of $Input_3$ with Mbf_8 and stores the result in the data stack
FZOR	implements the operation OR between the results obtained with the operations LDK and LDP
CON crisp$_1$	multiplies the result of the last Ω operation with the crisp value $Crisp_1$

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

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

LDP 3 1	fixes the α value of $Input_3$ with Mbf_1 and stores the result in the data stack
LDN 4 15	calculates the NOT α value of $Input_4$ with Mbf_{15} and stores the result in the data stack
FZAND	adds NOT α and α values obtained with the operations LDP 3 1 and LDN 4 15
SKM	stores the result of the operation FZAND in the memory register M
LDP 1 6	fixes the α value of $Input_1$ with Mbf_6 and stores the result in the data stack
LDN 2 14	calculates the NOT α value of $Input_6$ with Mbf_{14} and stores the result in the data stack
FZOR	implements the operation OR between the α and NOT α values obtained with the two previous operations (LDP 1 6 and LDN 2 14)
LDK	stores the result of the operation OR in the data stack
LDM	copies the value of the memory register M in the data stack
FZOR	implements the operation OR between the last two values stored in the data stack (LDK and LDM)
CON crips$_2$	multiplies the result of the last Ω operation with the crisp value $Crip_2$

At the end of the fuzzy rules set a byte, to identify the output involved in the rules, and the STOP istruction must be inserted.

When the STOP instruction is performed, the control of the algorithm goes back to the CU.

2.5 ARITHMETIC LOGIC UNIT

The 8-bit Arithmetic Logic Unit (ALU) allows to perform arithmetic calculations and logic instructions which can be divided into 4 groups: Load, Arithmetic, Jump and Program Control instructions (refer to the ST52x301 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.

Table 2.5. Arithmetic & Logic Instructions Set

Load Instructions					
Menmonic	**Instruction**	**Bytes**	**Cycles**	**Z**	**S**
LDCF	LDCF conf, const	2	6	-	-
LDRC	LDRC reg, const	2	6	-	-
LDRI	LDRI reg, inp	2	6	-	-
LDPR	LDPR per, reg	1	6	-	-
LDRR	LDRR regi, regj	2	6	-	-

Arithmetic Instructions					
Mnemonic	**Instruction**	**Bytes**	**Cycles**	**Z**	**S**
ADD	ADD regi, regj	2	7	I	I
AND	AND regi, regj	2	7	I	-
SUB	SUB regi, regj	2	7	I	I
SUBO	SUBO regi, regj	2	7	I	I

Jump Instructions					
Mnemonic	**Instruction**	**Bytes**	**Cycles**	**Z**	**S**
JP	JP addr	2	6	-	-
JPNS	JPNS addr	2	6	-	-
JPNZ	JPNZ addr	2	6	-	-
JPS	JPS addr	2	6	-	-
JPZ	JPZ addr	2	6	-	-

SCI Instructions					
Mnemonic	**Instruction**	**Bytes**	**Cycles**	**Z**	**S**
SRX	SRX regi	2	5	-	-
STX	STX regi	2	5	-	-

Notes:
I affected
- not affected

Table 2.6. Arithmetic & Logic Instructions Set (Continue)

Program Control Instructions					
Mnemonic	**Instruction**	**Bytes**	**Cycles**	**Z**	**S**
RETI	RETI	1	5	I	I
RINT	RINT int	1	4	-	-
STOP	STOP	1	4	-	-
WAITI	WAITI	1	4	-	-
UDGI	UDGI	1	4	-	-
UEGI	UEGI	1	4	-	-
MDGI	MDGI	1	4	-	-
MEGI	MEGI	1	4	-	-
IRQ	IRQ int label	2	6	-	-
IRQM	IRQM mask	2	6	-	-
IRQP	IRQP cost	2	6	-	-

Notes:
I affected
- not affected

3 EPROM

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

There are 16K bits of memory space with an 8-bit internal parallelism (2Kbytes) addressed by an 11-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 and 5V±10% during Working Phase. V_{DD} is equal to 5V±10%.

The EPROM memory of ST52x301 is divided in three main blocks (see Figure 3.1):

- Mbfs Setting with (0 through 191) contains the coordinates of the vertexes of every Mbf defined in the program.
- Interrupt Vectors (192 through 201) contain the addresses for the interrupt routines. Each address is composed of two bytes.
- Program Instruction Set (202 through 2048) contains the instruction set of the user program. It can be composed of more Boolean and Fuzzy Algorithms

The operation that can be performed, during Programming Phase, on the EPROM are: Writing, Verify, Writing Inhibit, Standby and Erasing.

Figure 3.2 shows the signals timing in Programming Mode.

3.1 EPROM Programming Phase Procedure

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

RESET: 0, TEST: 0, MODE: 1.

CADD, ERES, OE and CE are the control signals used during the Programming Mode. CADD is active on edge, the others are active on level (OE, CE are active low, ERES is active high).

3.1.1 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:

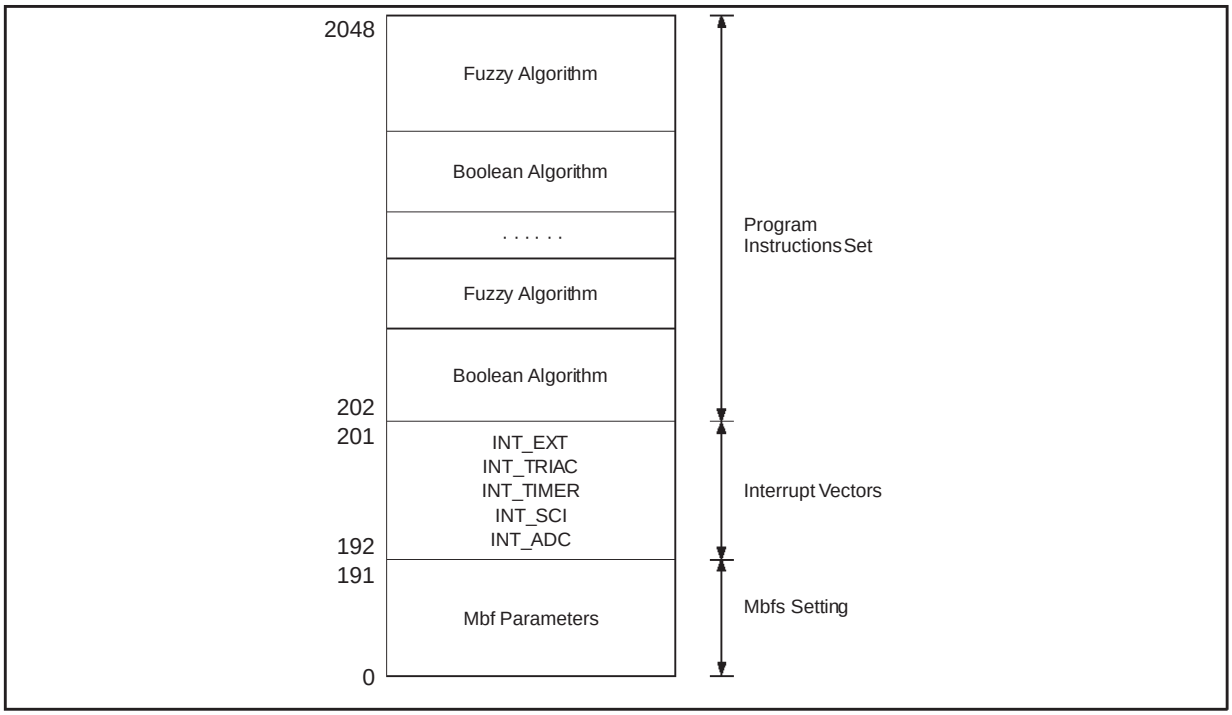
CE = LOW

OE = HIGH

with stable data on the data bus P(0:7).

The total programming pulse width (CE = 0 V) is, typically, 50 µs (by means of 5 pulses of 10 µs), but before activating such pulse, it is suggested to wait for at least 2 µs after V_{PP} rises at 12 V . After the disactivation of the pulse it is suggested to wait for

Figure 3.1. Memory Map



at least 2 µs before updating the data and the address.

The data updating for the next programming is performed, directly by the user, on the data bus P(0:7) while the address is incremented through the pin CADD.

3.1.2 EPROM Verify

A Verify mode is available in order to verify the correctness of the data written. It is possible to activate the Verify mode immediately after the writing of each byte:

CE = HIGH

OE = LOW

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

The data, during this phase, are avalaible on the bus P(0:7)

3.1.3 Writing Inhibit

It occurs between the Writing and Verify Mode:

CE = HIGH

OE = HIGH

3.1.4 Standby Mode

The EPROM has a standby mode which reduces the active current from 10mA (Programming mode) to less than 100 µA. The Memory is placed in standby mode by setting CE at HIGH Logic Level (V_{PP} might be equal to 5 V too). When in standby mode, the outputs are in high impedance state.

3.2 Eprom Erasure

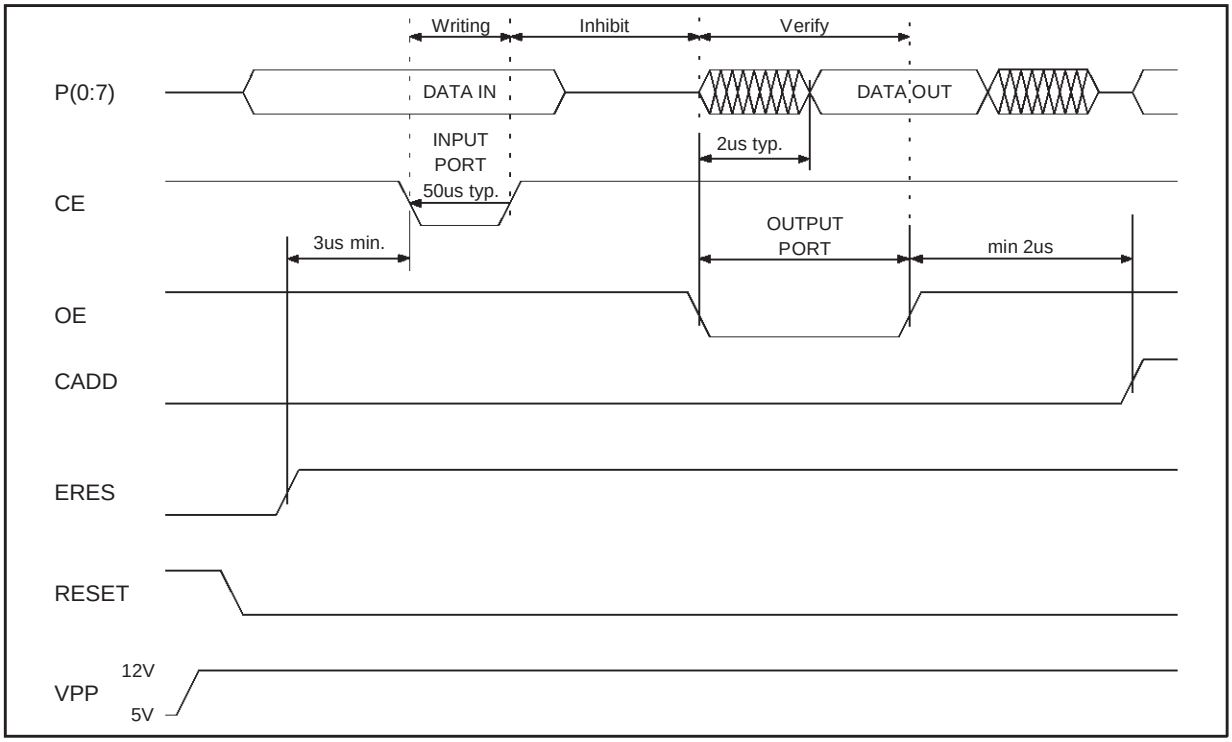
Thanks to the transparent window present in the CLCC44-W package, its memory contents may be erased by exposure to UV light.

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 2537 Å. 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.

Figure 3.2. EPROM Programming Timing



4 INTERRUPTS

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

When such an event occurs, if it 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 one of the EPROM memory location between address 192 and 201.

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

With the instruction RETI (Return from Interrupt) the arithmetic flags and Program Counter (PC) are restored from the top of the stack. This stack, used for the Interrupt priority, is a LIFO queue.

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

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 generate and Interrupt Service Routine associated to the interrupt with higher priority will start.

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

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

MEGI/MDGI instructions set the Macro Global Interrupt Mask in order to assure that the macro will not be broken.

Figure 4.1. Interrupt Flow

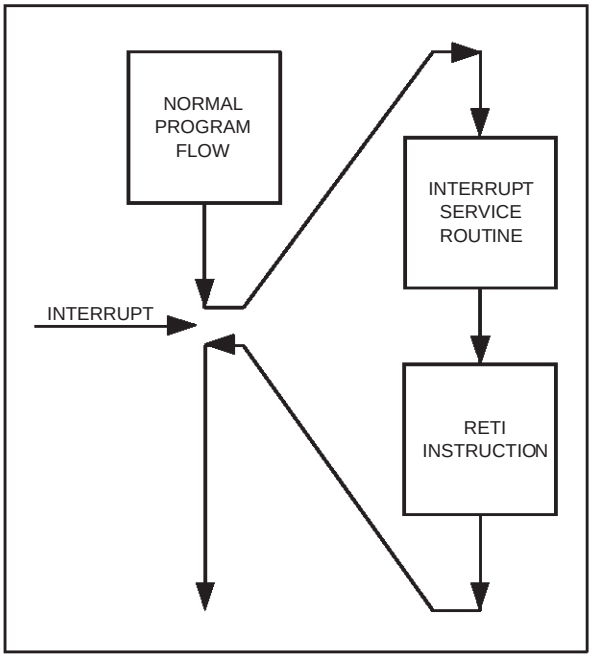


Figure 4.2. Interrupt Vectors Mapping

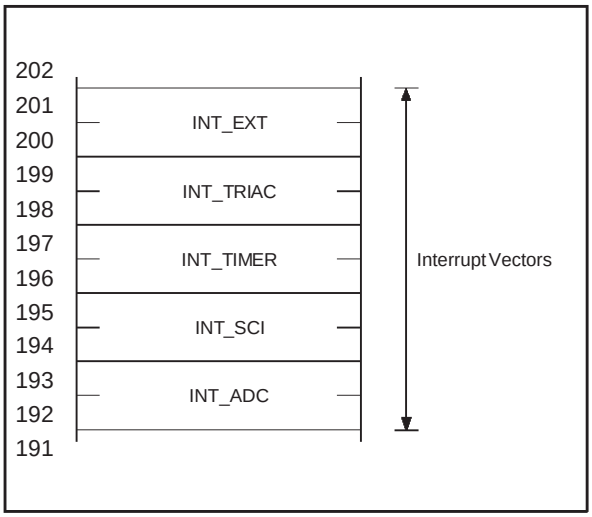
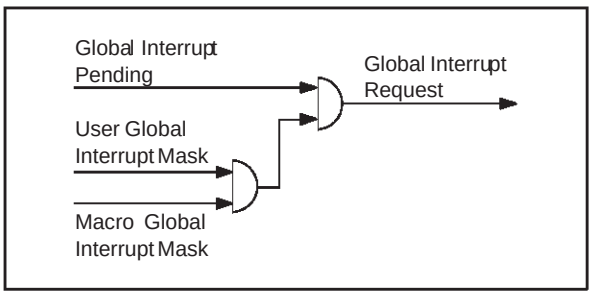


Figure 4.3. Global Interrupt Request generation



4.3 Interrupt Sources

ST52x301 manages interrupt signals generated by the internal peripherals (Timer, Triac/PWM Driver,Analog to Digital Converter and Serial Communication Port) or coming from the INT pin.

The polarity of the External Interrupt is programmed by the EXTI bit of the REG_CONF14 (see Table 4.1 and fig. 4.4). EXTI=0 means that INT_EXT is active on rising edge, otherwise it is active on falling edge.

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

4.4 Interrupt Maskability

The interrupts can be masked by configuring the REG_CONF14. 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 `LDCF 14, 6` (CONF_REG14 =00000110) enables interrupts coming from the ADC (INT_ADC) and from the SCI (INT_SCI).

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_CONF15 (see fig.4.5); whereas the higher level is related to the external interrupt (INT_EXT).

Timer, Triac/PWM Driver, SCI 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 INT*i* priority level.

Table 4.1. Configuration Register 14 Description

Bit	Name	Value	Description
0	MSKE	0	External Interrupt Masked
		1	External Interrupt Not Masked
1	MSKAD	0	A/D Converter Interrupt Masked
		1	A/D Converter Interrupt Not Masked
2	MSKSCI	0	SCI Interrupt Masked
		1	SCI Interrupt Not Masked
3	MSKTM	0	TIMER Interrupt Masked
		1	TIMER Interrupt Not Masked
4	MSKTC	0	TRIAC/ PWM Interrupt Masked
		1	TRIAC/ PWM Interrupt Not Masked
5	not used	-	
6	not used	-	
7	EXTI	0	Active on Rising Edge
		1	Active on Falling Edge

Table 4.2. Interrupts Description

Name	Description		Priority	Peripheral Code	Maskable	EPROM Locations
INT_EXT	External Interrupt (INT)	Ext	Highest	-	yes	200-201
INT_ADC	ADC	Int	Programmable	00	yes	192-193
INT_SCI	SCI	Int	Programmable	01	yes	194-195
INT_TIMER	TIMER	Int	Programmable	10	yes	196-197
INT_TRIAC	TRIAC	Int	Programmable	11	yes	198-199

Figure 4.4. Interrupt Configuration Register 14

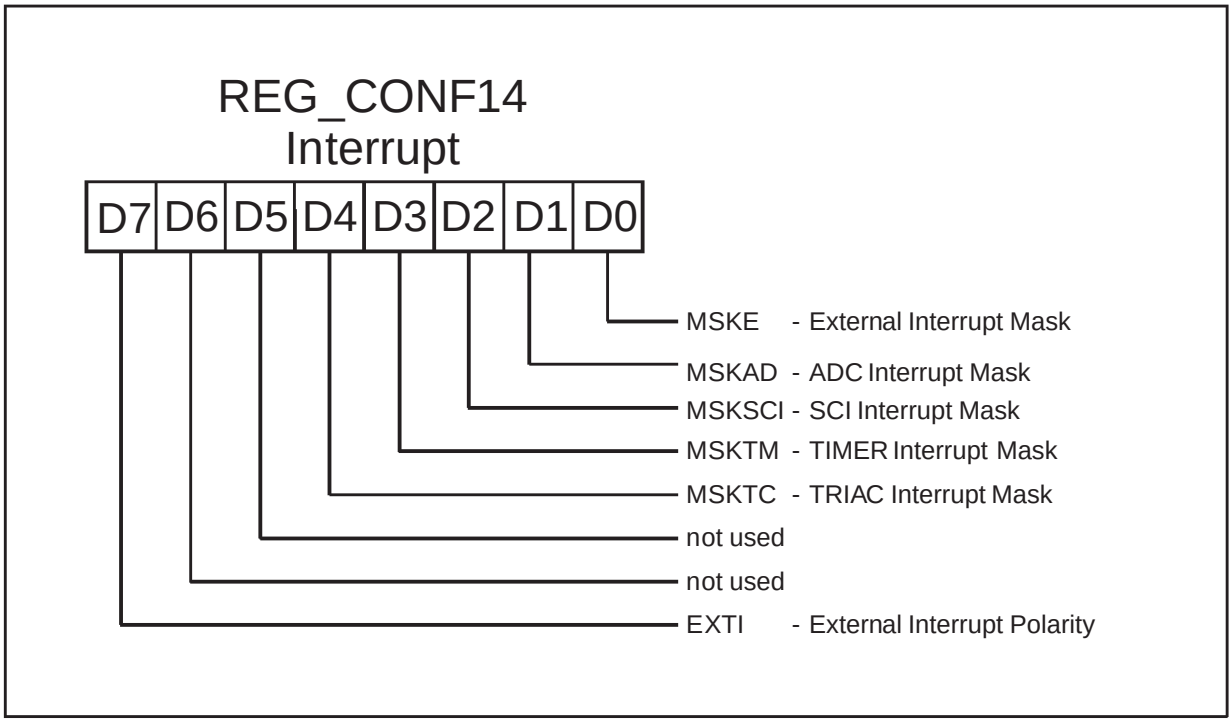
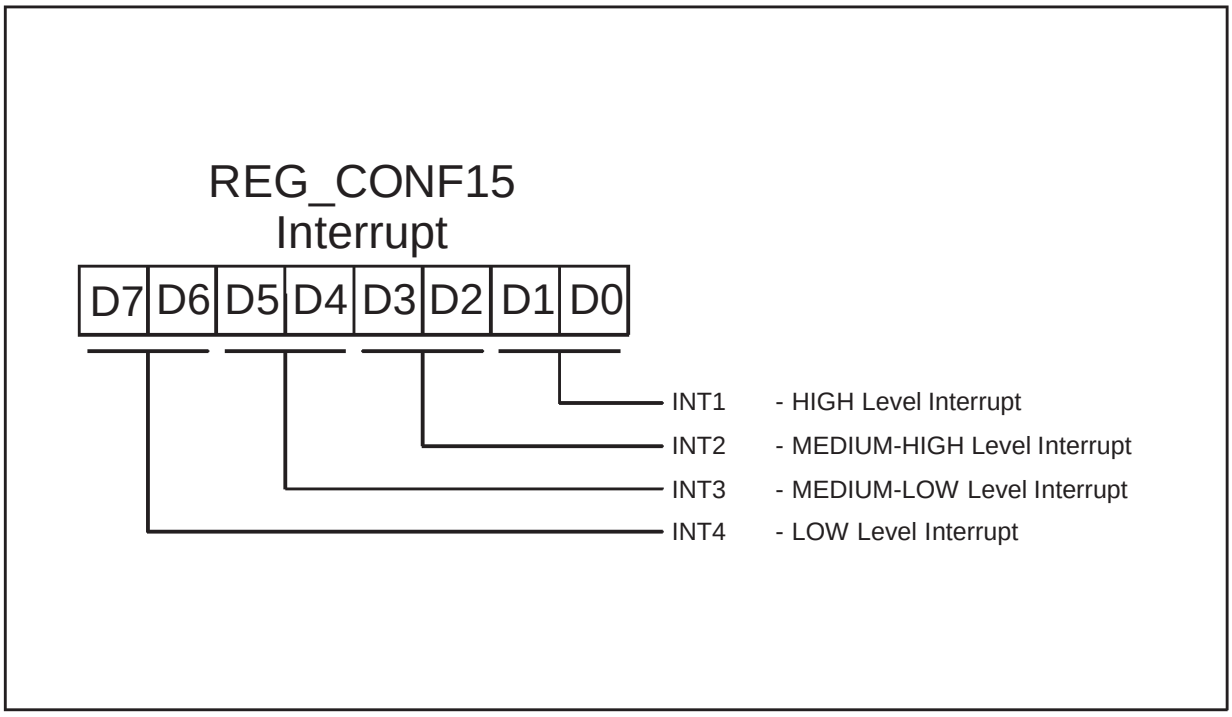


Figure 4.5. Interrupt Configuration Register 15



i.e. `LDCF 15, 201` (REG_CONF15=11001001) define the following priority levels:

- Level 1: INT_SCI(SCI Code: 01)
- Level 2: INT_TIMER(TIMER Code: 10)
- Level 3: INT_ADC(ADC Code: 00)
- Level 4: INT_TRIAC(TRIAC Code: 11)

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 transfering 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.

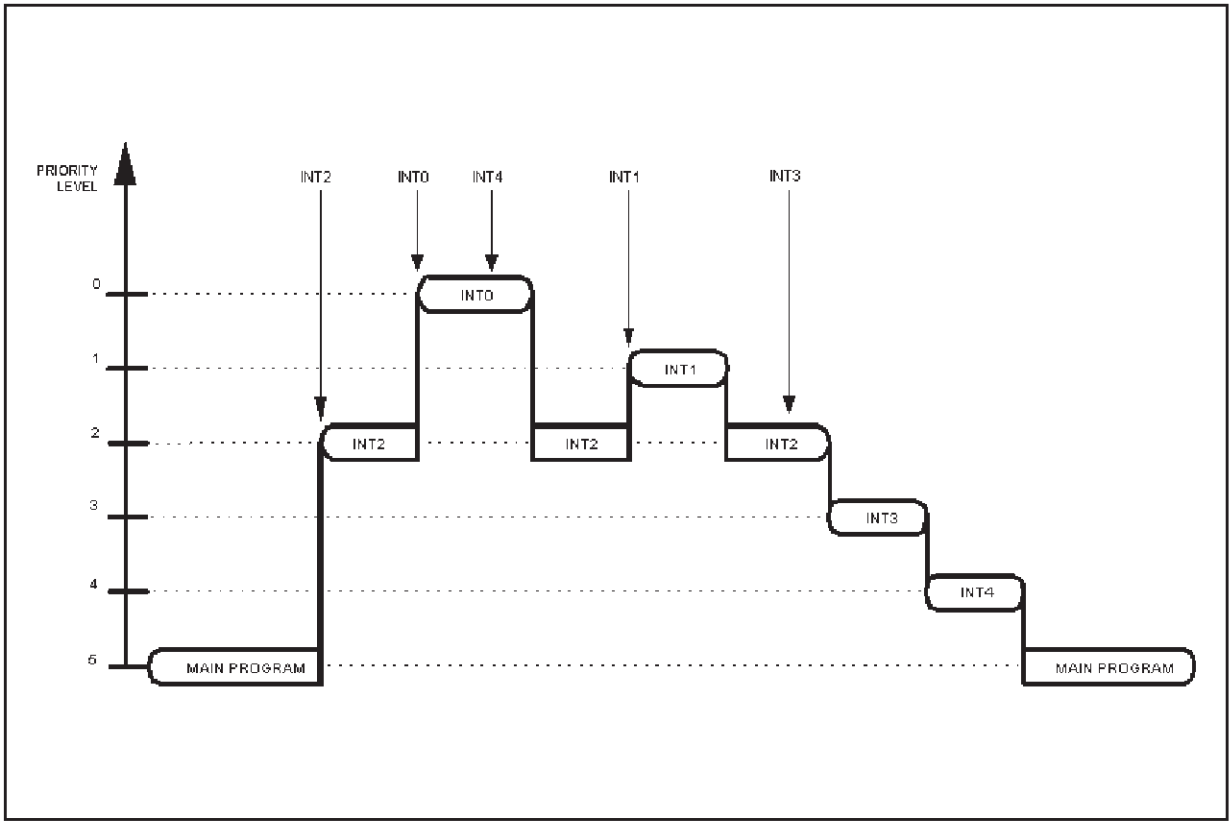
4.6 Interrupt RESET

An eventually pending interrupts can be reset with the instruction `RINT inti` which resets the *i*-th interrupt

Table 4.3. Configuration Register 15 Description

Bit	Name	Value	Level
0, 1	INT1	Peripheral Code	High
2, 3	INT2	Peripheral Code	Medium-High
4, 5	INT3	Peripheral Code	Medium-Low
6, 7	INT4	Peripheral Code	Low

Figure 4.6. Example of a Sequence of Interrupt Requests



5 CLOCK

ST52x301 can work by using a 5, 10 or 20 MHz clock.

The ST52x301 Clock Generator module generates the internal clock for the internal Control Unit, ALU, Fuzzy Core and on-chip peripherals and it is designed to require a minimum of external components.

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.

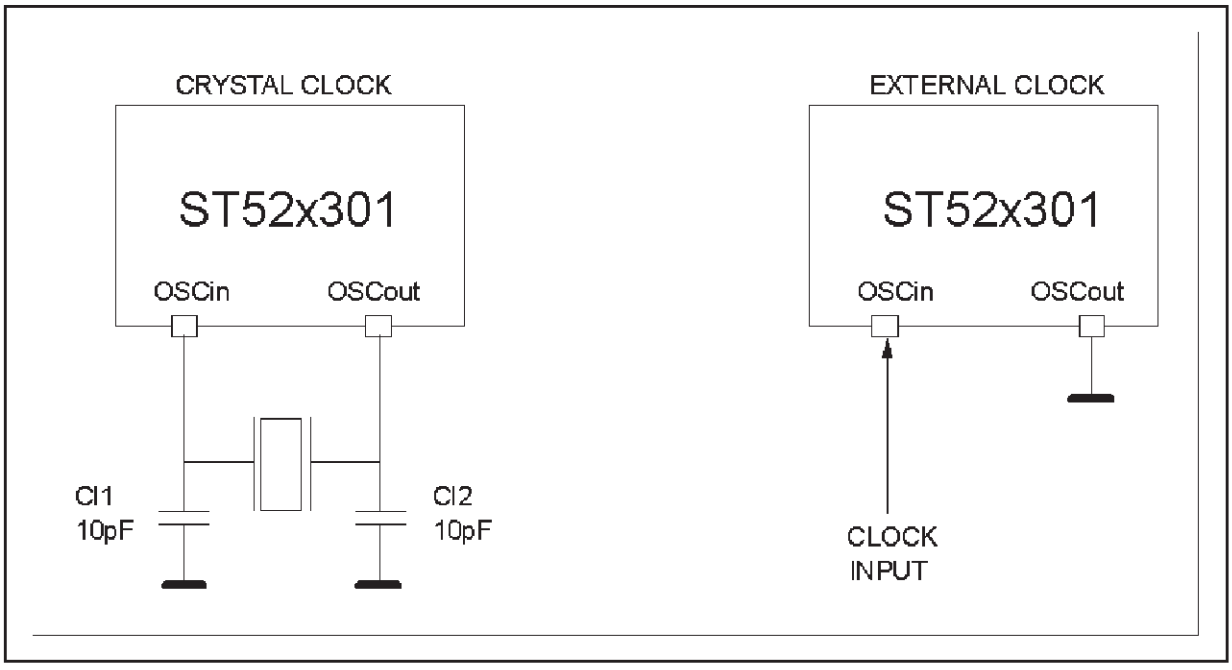
The different clock generator options connection methods are shown in Figure 5.1.

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

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

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

Figure 5.1. Oscillator Connections



6. A/D CONVERTER

The A/D Converter of ST52x301 is an 8-bit analog to digital converter with up to 4 analog inputs offering 8 bit resolution with a total accuracy of 2 LSB and a typical conversion time of 32 µs.

The conversion range is 0 - 2.5 V.

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

The ADC uses 5 registers: one Configuration Register, REG_CONF2, and four Data Registers. These 4 registers are the first 4 Input Registers.

The A/D converter drives the analog Multiplexer in order to sequentially pick up the external inputs to be put in output and stored automatically in 4 8-bit registers.

It is possibile to configure the Multiplexer by means of the register REG_CONF2, in order to select the number of analog inputs to convert.

For example, if the bit 3 and bit 2 of REG_CONF2 are configured at 10, then the Multiplexer will sequentially pick up only the inputs 0,1 and 2.

Table 6.1 shows the convertion sequences according to the possible values of the two bit REG_CONF2 (3:2).

The A/D Converter, at the end of the conversion, will send a signal (end-of-conversion) which can be used like an interrupt signal. The user can select the priority of the A/D interrupt and mask it (see "Interrupt Routine" chapter)

The conversion starts writing "1" on REG_CONF2(0). The A/D is reset by writing "0" in REG_CONF2(0).

The converted data are automatically stored in four 8-bit Input Registers.

By performing an instruction:

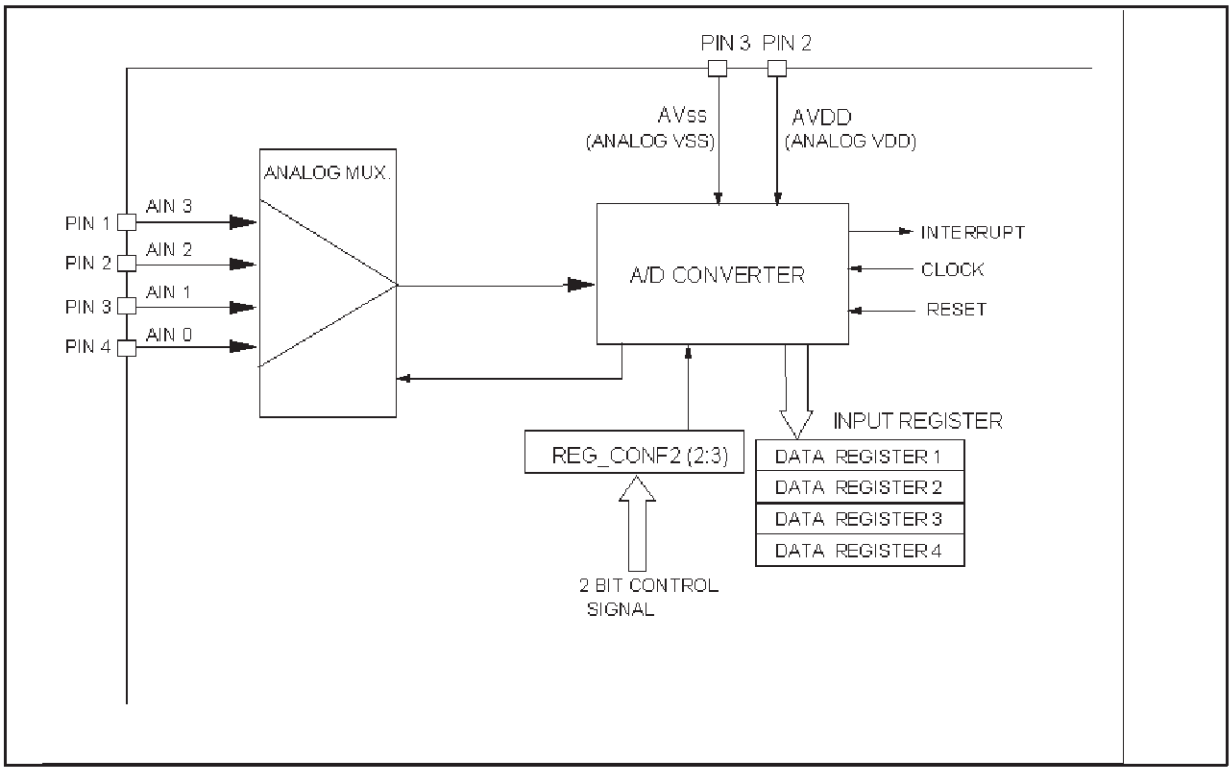
```
LDRI regj ingi
```

the analog input "ingi" is loaded in the register "regj" of the Register File.

Table 6.1.

CONF_REG2 (3:2)	INPUT SEQUENCE
00	Ain0
01	Ain 0, Ain1
10	Ain 0, Ain 1, Ain 2
11	Ain 0, Ain 1, Ain 2, Ain 3

Figure 6.1. A/D Converter Structure



The power consumption of the device can be reduced by turning off the A/D converter,

To switch off the A/D converter the CONF_REG2(0) bit must be reset to ”0”.

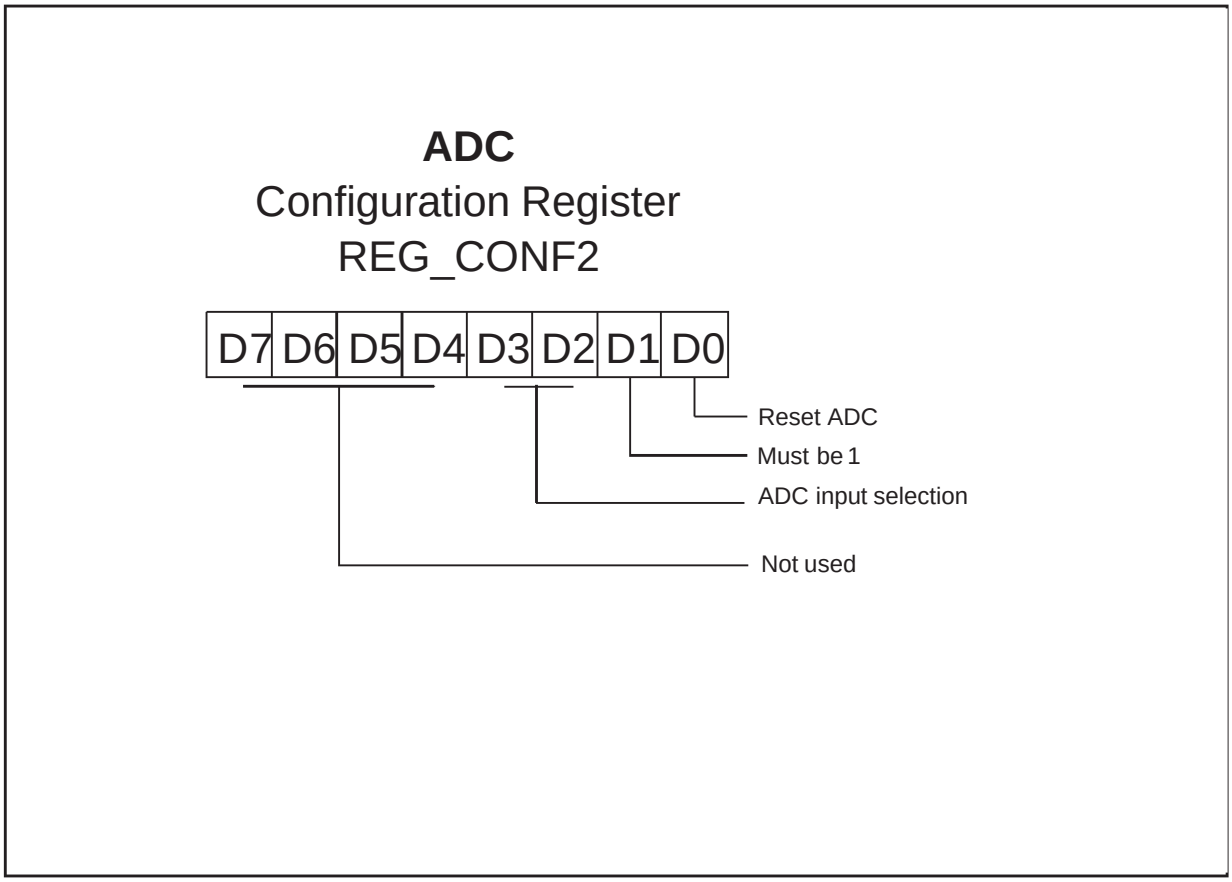
The A/D Converter features a sample and hold.

The input voltage Ain, which has to be converted must be constant, for 12.8 µs.

An internal bandgap reference is available on pin 44, BG. By using this signal as reference for the signal to be converted, the conversion accuracy is not strongly related with the variation of the power supply.

The power supply of the A/D converter (AV_{DD} and AV_{SS}) in order to avoid interferences is mantained separated from the power supply of the digital core.

Figure 6.2. Configuration Register REG_CONF2



7. TIMER

ST52x301 offers one on-chip Timer peripheral. TheTimer consists of an 8-bit counter with a 16-bit programmable prescaler, thus giving a maximum count of 2^{24}, and control logic that allows configuring the functionment and the type of peripheral outputs. Figure 7.2 shows the Timer block diagram and Figure 7.3 shows the internal structure of the Timer.

The content of the 8-bit counter can be read/written and is incremented on the Rising Edge of the 16-bit prescaler output (PRESCOUT). Moreover, it can be read under program control at any instant of the counting phase and loaded in a location of the Register File. The prescaler can be given any value between 0 and FFFFh setting the 4-th (TMLSB) and 5-th (TMMSB) locations of the Configuration Registers Bench.

7.1 Timer Functionment

The Timer requires three signals: TMRCLK, TRST and TSTART (see Figure 7.3). Each of them can be generated internally or externally, this possibility is programmable by the user.

TMRCLK increments the counted value of the Prescaler. It can be, by setting CKSL of REG_CONF6 register, the internal clock signal (CLKM) or the signal provided on the pin TCLK.

TRST resets to zero the content of the 8 bit counter. It is generated by the TRES or RESET external signals or it is forced by TMRST bit of REG_CONF6 register.

TSTART starts/stops the Prescaler counting. It can be given on the pin TCTRL or it is forced by TMST bit of REG_CONF6 register.

The TSTART signal allows to work in two different modes:

LEVEL (Time Counter): If the TSTART signal is high the Timer starts the count. When the TSTART is low the count is stopped and the current value is stored in the TMR_OUT register of the Input register Bench, then it can be transferred to the j-th location of the Registers File by using the instruction:

```
LDRI reg-j 4
```

EDGE(Period Counter): After the reset, when the first edge of the TSTART signal appears, the Timer starts the count, at the next TSTART the Timer is stopped. In this way it is possible to measure the period of an external signal.

The functionment modality is set by the TMEL configuration bit of REG_CONF6 register.

The starting value of the Counter can be either a value contained in the Register File or directly a Fuzzy Output. If INPSL (REG_CONF7(3)) is set to "1" then the value comes from one of the locations of the Register File (`LDRP 0, reg-i);` on the contrary it is generated by the Fuzzy Core. The choice between the two possible fuzzy outputs is set by the FZSL configuration bit of REG_CONF6 register

FZSL=0/1 means the starting value is the loaded from the FUZZY_OUT_0/1.

Figure 7.1. Timer Functionalities

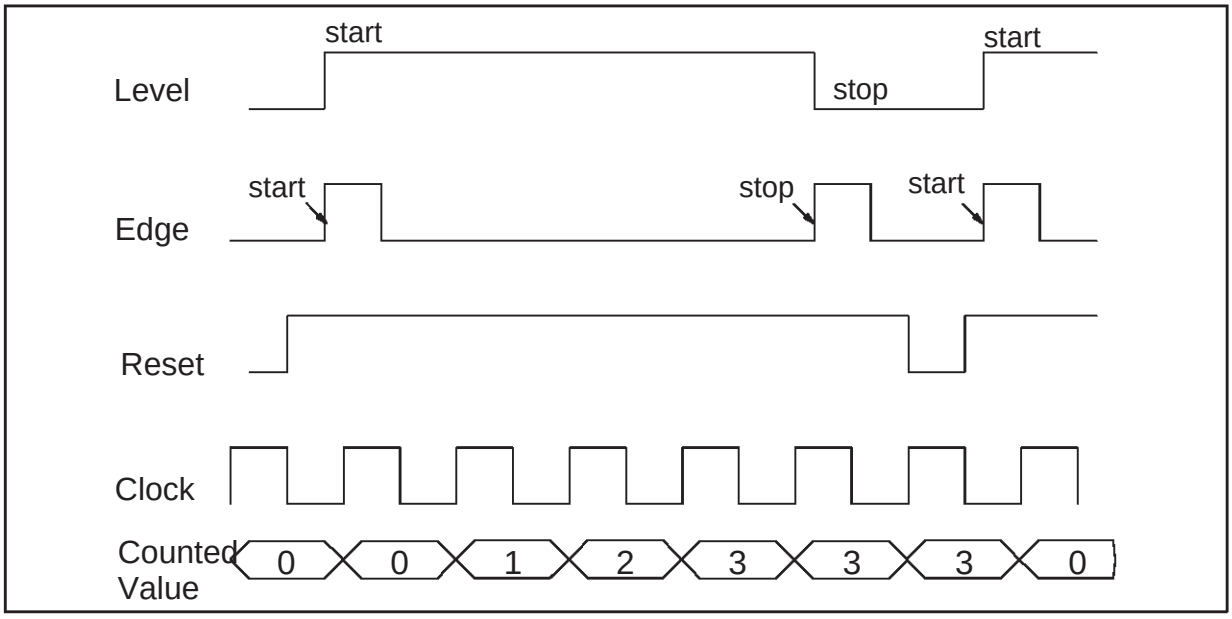


Figure 7.2. Timer Peripheral Block Diagram

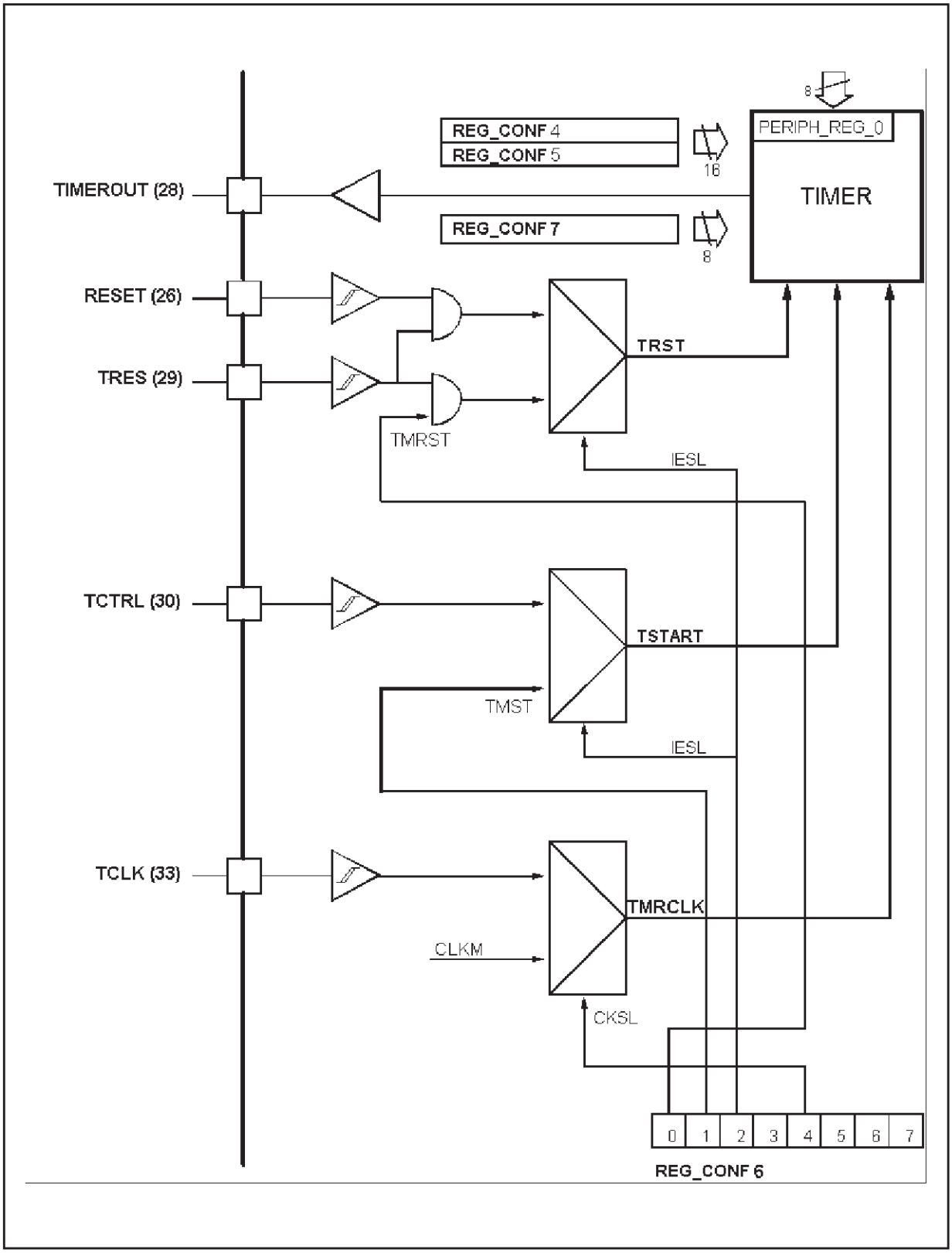
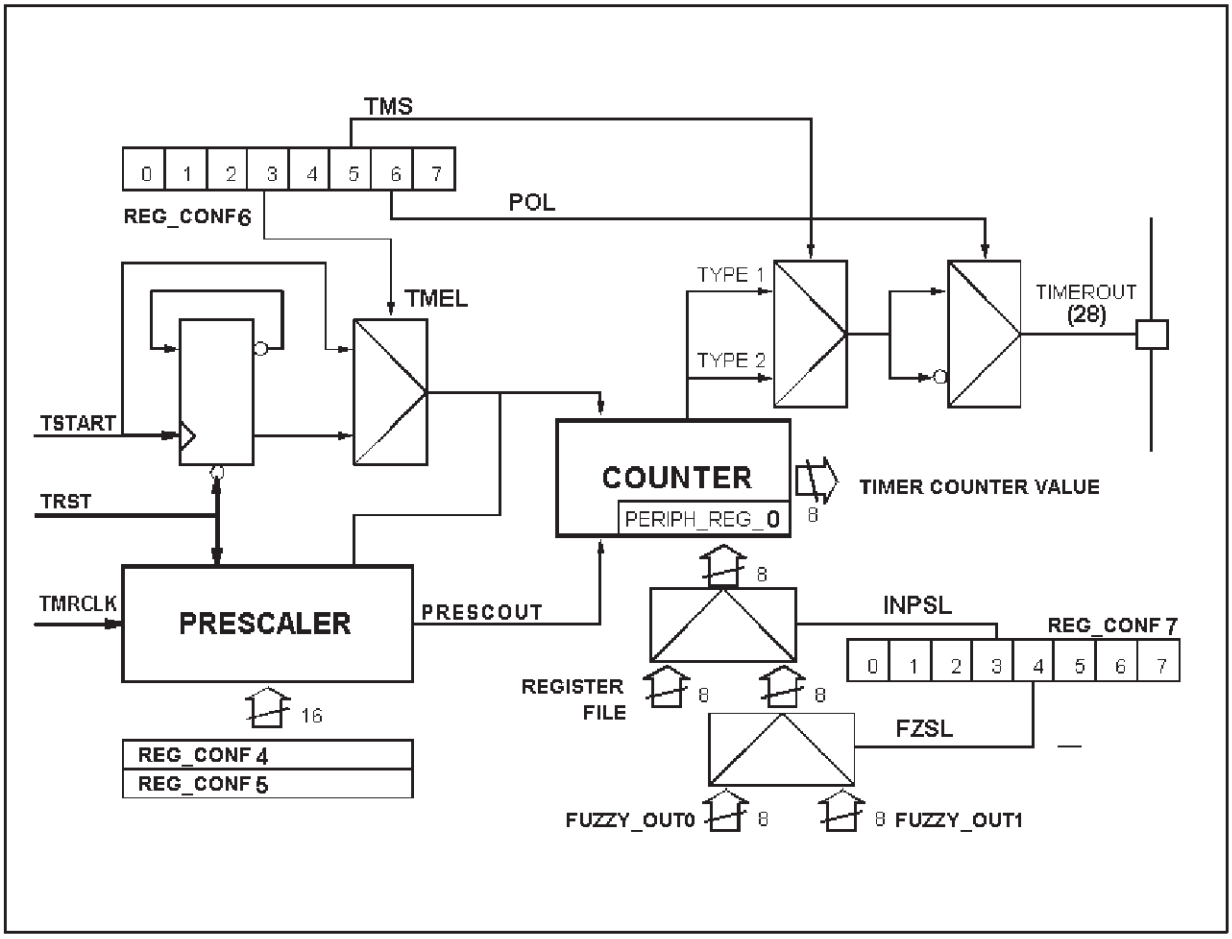


Figure 7.3. Timer Internal Structure



7.2 Timer Interrupt

It is possible to enable the Timer Interrupt by software control. The Timer 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.

In order to program the interrupt mode INTSL, INTF and INTR bits of the REG_CONF7 must be set following the indications shown in the Table 7.1. The Timer interrupt can be used to exit the MCU from the WAIT mode.

Figure 7.4. TIMEROUT Signal Type

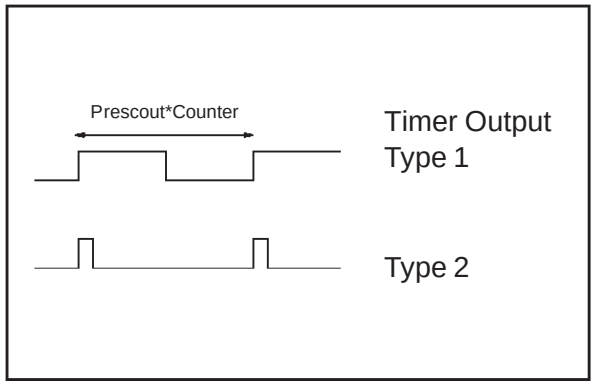


Table 7.1. Timer Interrupt Setting

INTERRUPT MODE	INTSL	INTF	INTR
1	1	X	X
2	0	1	0
3	0	0	1
4	0	1	1

7.3 Timer Configuration

The Timer configuration needs to set 4 registers of the Configuration Register Bench.

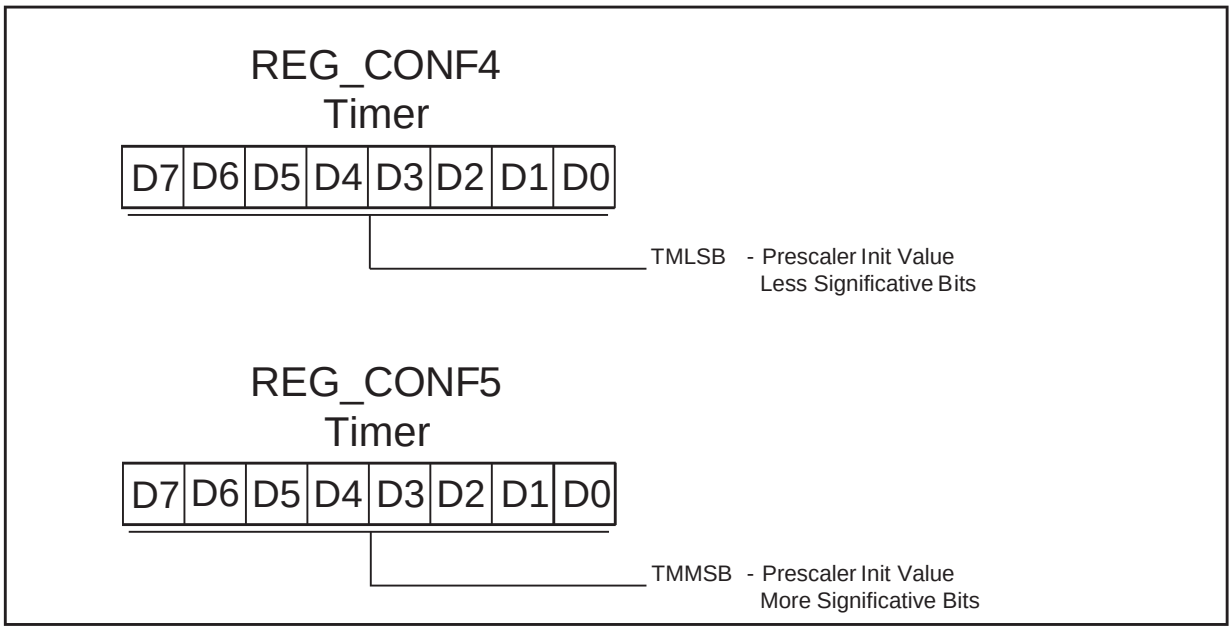
CONF_REG4:

TMLSB contains the less significative bits of the Prescaler starting value.

CONF_REG5:

TMMSB contains the more significative bits of the Prescaler starting value

Figure 7.5. Timer Configuration Register 4 and 5



CONF_REG6:

TMRST sets the internal INR signal.

TMST sets the internal INS signal.

IESL selects the source of the TRES and TSTART signals.

IESL="0" signals are the internal INR and INS.

IESL="1" signals come from the TRES and TCTRL pins.

TMEL selects the TSTART signal allowing to work in **Level Mode** or in **Edge Mode** like previously described.

TMEL="0" means Edge Mode

TMEL="1" means Level Mode.

CKSL selects the source of the TMRCLK (working Timer frequency).

CKSL="0", the TMRCLK is the internal MCLK divided by the Prescaler starting value.

CKSL="1", the TMRCLK is an external clock by TCLK pin.

TMS TIMEROUT is a signal with frequency equal to the working Timer frequency divided by the starting value of the Prescaler (16 bit) and Counter (8 bit). The Timer output can be either a square wave with duty-cycle 50% or a pulse signal (with the pulse duration equal to the Prescaler output signal period).

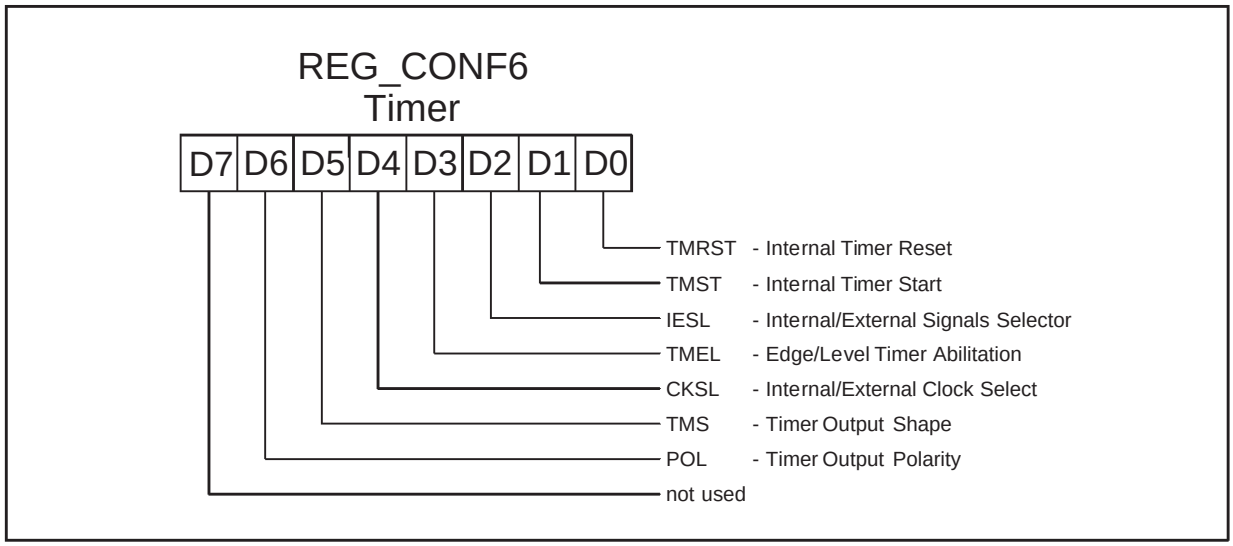
TMS="1", TIMEROUT is a square wave
TMS= "0", TIMEROUT is a pulse signal.

POL defines the polarity of the Timer output signal (TIMEROUT).

Table 7.2. Configuration Register 6 Description

Bit	Name	Value	Description
0	TMRST	0	Stop
		1	Start
1	TMST	0	Stop
		1	Start
2	IESL	0	Internal Signals
		1	External Signals
3	TMEL	0	on Edge
		1	on Level
4	CKSL	0	Internal Timer Clock
		1	External Timer Clock
5	TMS	0	Pulse Wave (Type 2)
		1	Square Wave (Type 1)
6	POL	0	Positive Polarity
		1	Negative Polarity
7	not used	-	

Figure 7.6. Timer Configuration Register 6



CONF_REG7:

INTSL It allows to select the interrupt mode for the Timer.

INTSL="0" Interrupt is generated on the falling edge of the Counter Stop.

INTSL="1" the interrupt is generated on the edges of TIMEROUT.

INTF

INTR

INPSL selects the source of the value of the Counter between a location of the Register File and the Fuzzy Core.

INPSL="0", Counter value coming from the FC.

INPSL="1", Counter value coming from the RF.

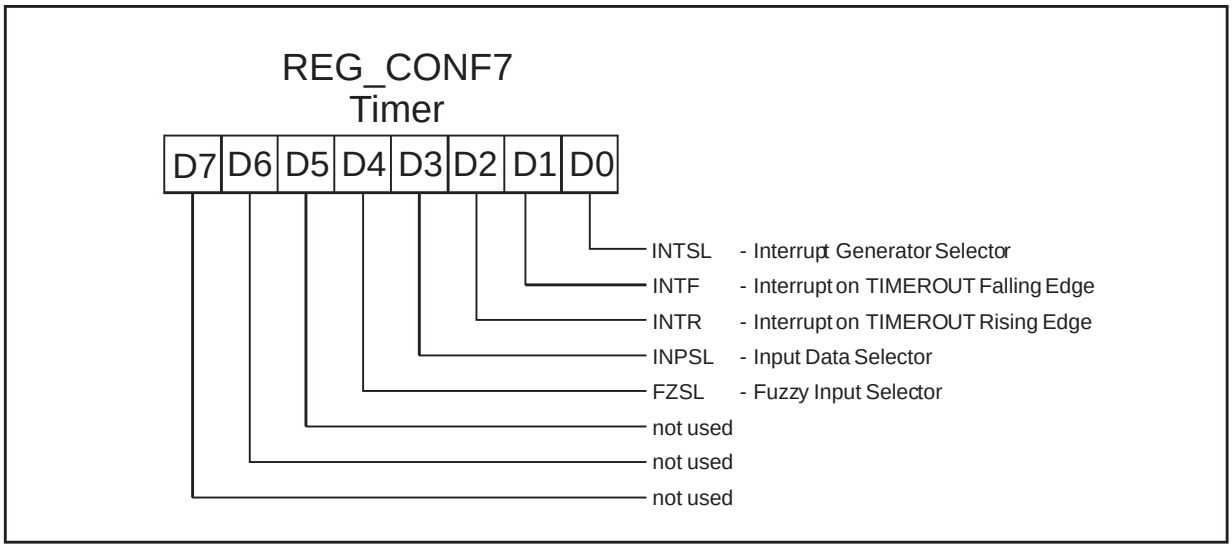
FZSL FZSL="0", the value of the Timer Counter is equal to FUZZY_OUT_0

FZSL="1", the value of the Timer Counter is equal to FUZZY_OUT_1

Table 7.3. Configuration Register 7 Description

Bit	Name	Value	Description
0	INTSL	0	INT_TMR on Falling Edge of Counter Stop
		1	INT_TMR on Edges of TIMEROUT
1	INTF	0	NO INT_TMR on Falling Edge of TIMEROUT
		1	INT_TMR on Falling Edge of TIMEROUT
2	INTR	0	NO INT_TMR on Rising Edge of TIMEROUT
		1	INT_TMR on Rising Edge of TIMEROUT
3	INPSL	0	Timer Data Input coming from the Fuzzy Core
		1	Timer Data Input coming from a Register File location
4	FZSL	0	Timer Data Input coming from FUZZY_OUT_0
		1	Timer Data Input coming from FUZZY_OUT_1
5	not used	-	
6	not used	-	
7	not used	-	

Figure 7.7. Timer Configuration Register 7



8 I/O PORT

ST52x301 is provided with dedicated lines for input/output. These lines, grouped into an 8-bit I/O Port P(0:7), can be programmed to provide parallel input/output with a handshake line (READY) to carry data in/out.

The I/O Port is not able to perform operations on the single bit, and the communication cannot be performed at the same time in input and output.

It is possible to program the parallel port direction by using the register REG_CONF0 in order to set which bits are in input and which are in output.

The port has an internal register (PERIPH_REG_2) dedicated to hold output data coming from the Register File through an LDPR instruction.

Input data are automatically stored in the IN_PORT register, 6-th location of the Input Register.

P8 pin is a digital output line available directly connected to the OUT bit of the REG_CONF1; then it can be set by using a LDCF instruction.

(see table 8.2 and Figure 8.8)

Figure 8.1.

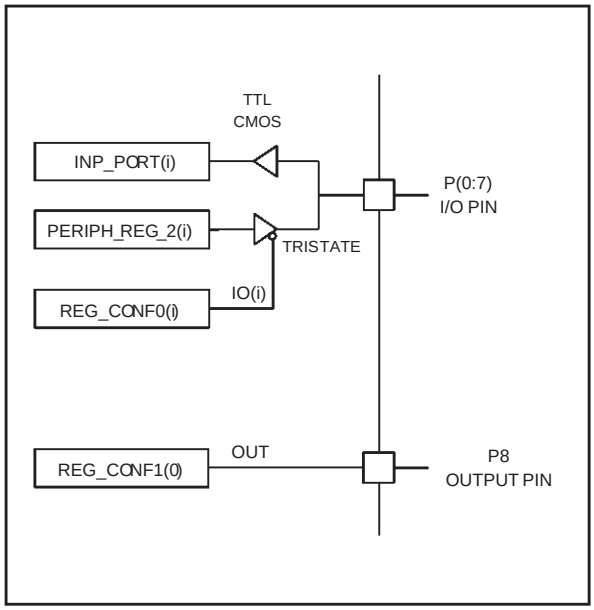
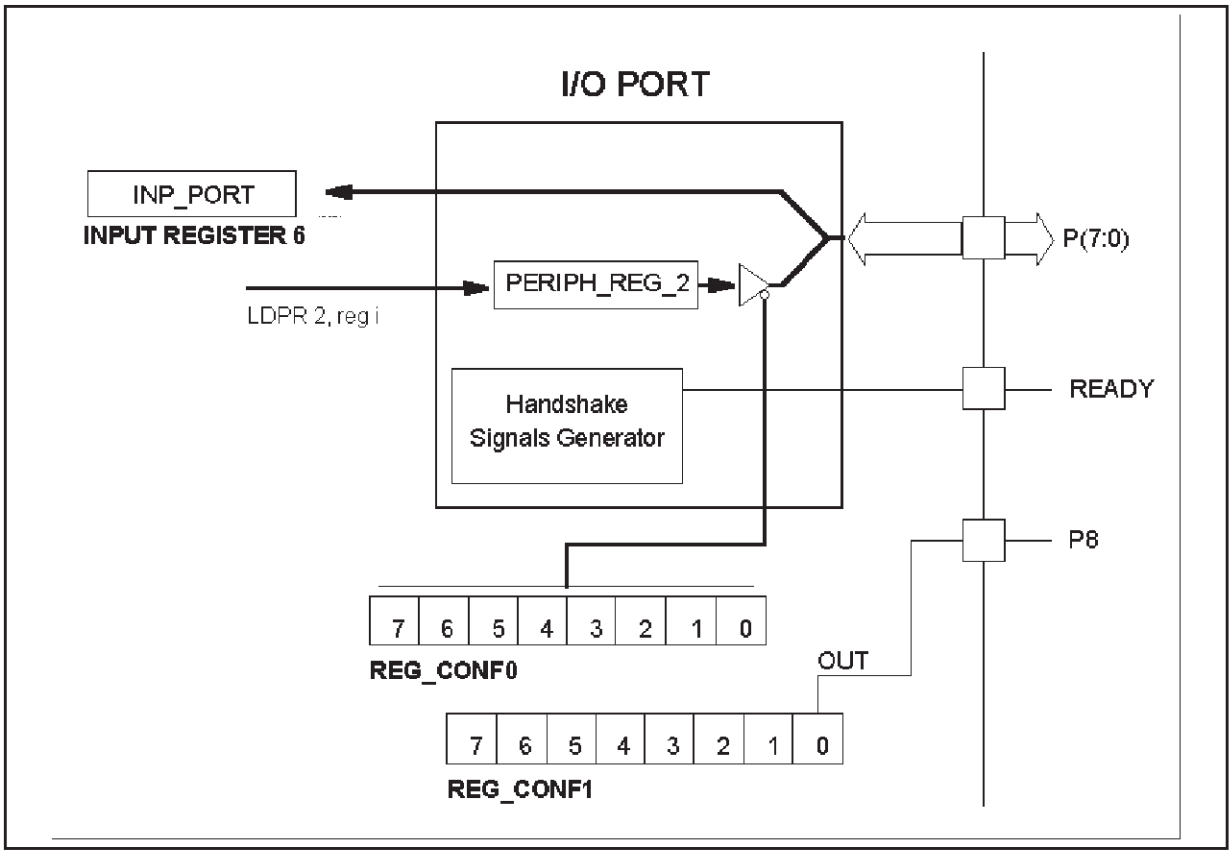


Figure 8.2.



8.1 I/O PORT CONFIGURATION

REG_CONF0 allows dynamic change in I/O Port configuration during program execution setting the communication direction of each bit.

IOi setting equal to "0" configures the i-th bit of the P(0:7) I/O Port in input. Data coming from external digital devices are stored in the 6-th location (INP_PORT) of the Input register bench.

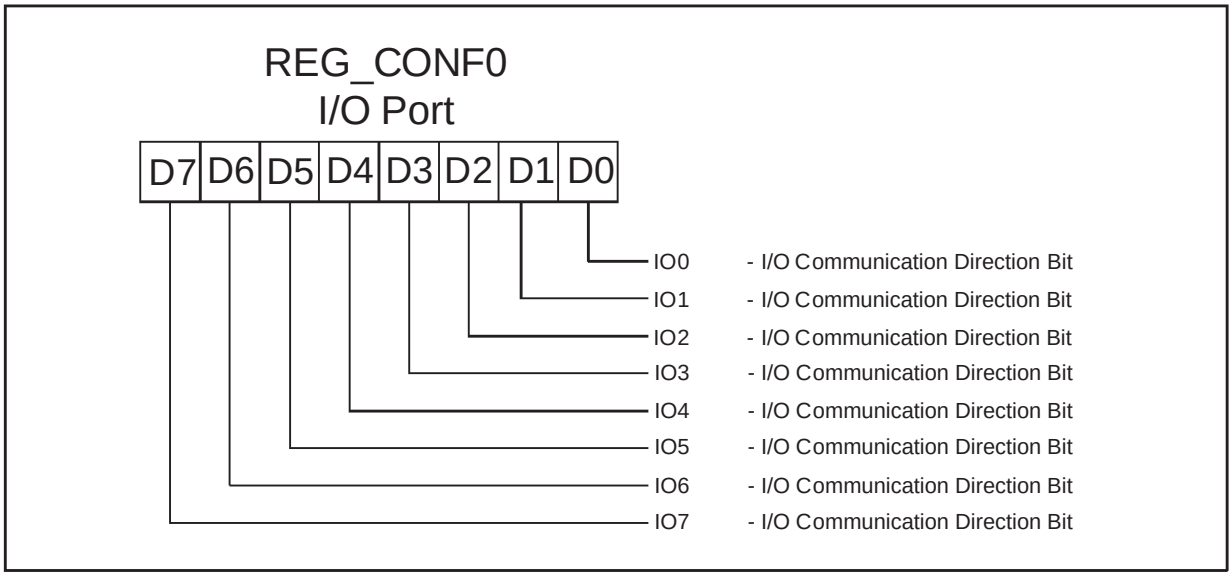
IOi="1" sets the i-th bit of the port in output. Data stored in the i-th location of the Register File is written on the port by using the instruction:

```
LDPR 2, regi
```

Table 8.1. Configuration Register 0 Setting

Bit	Name	Value	Description
0	IO0	0	Input Pin
		1	Output Pin
1	IO1	0	Input Pin
		1	Output Pin
2	IO2	0	Input Pin
		1	Output Pin
3	IO3	0	Input Pin
		1	Output Pin
4	IO4	0	Input Pin
		1	Output Pin
5	IO5	0	Input Pin
		1	Output Pin
6	IO6	0	Input Pin
		1	Output Pin
7	IO7	0	Input Pin
		1	Output Pin

Figure 8.3. Configuration Register 0



8.2 INPUT HANDSHAKE

Figure 8.5 illustrates the timing associated with the READY Handshake signal, when the instruction LDRI reg 6 is performed.

When the LDRI instruction is executed to read the port, ST52x301 resets the READY signal to indicate that it is not possible to change the input data during this phase of reading.

To synchronize the transmission with READY signal will prevent the INP_PORT data from changing while ST52x301 is reading the port.

READ PORT signal represented in figure 8.5 is an ST52x301 internal signal.

Input data on the port are continuously sampled and are strobed into the port only when READY is set.

Figure 8.4. One Line Input Handshake

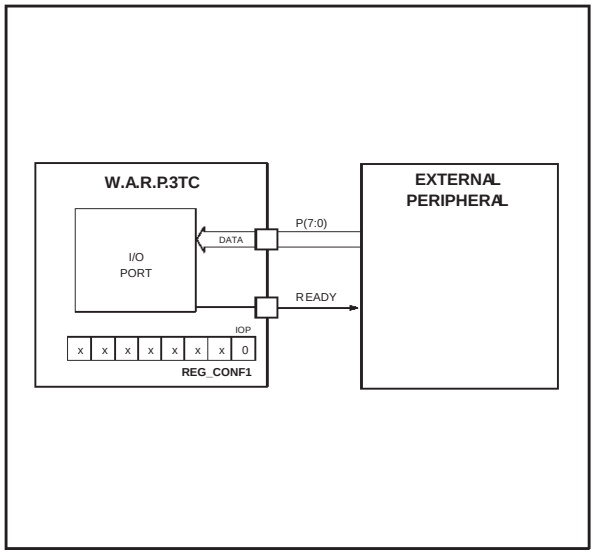
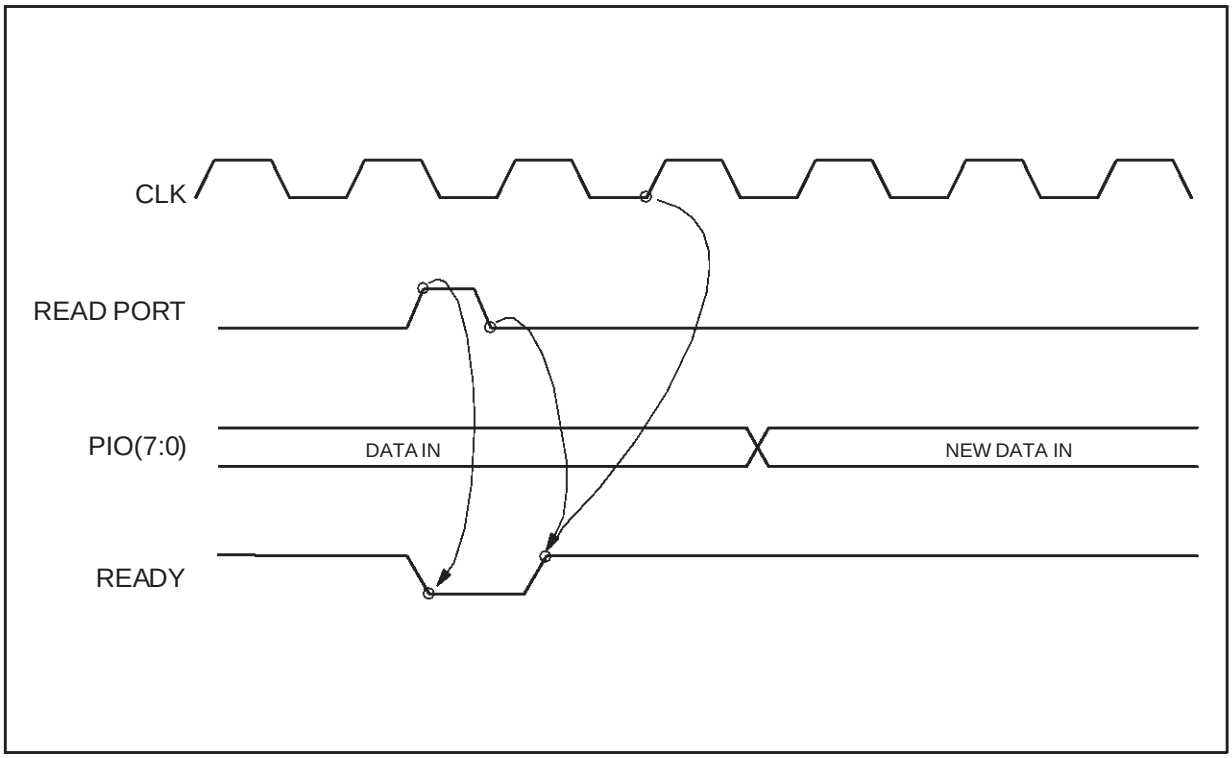


Figure 8.5. One Line Input Handshake Timing



8.3 OUTPUT HANDSHAKE

Figure 8.7 illustrates the timing associated with the READY Handshake signal, when the instruction LDPR 2 reg is performed.

When READY is reset no significant data are on the output port pins, because ST52x301 is writing into the PERIPH_REG_2.

When the data is ready in PERIPH_REG_2, READY signal is set.

The rising edge of READY signal can be used as a latching signal.

No peripheral acknowledge is waited for.

If the signal READY is high, it means that the data out is still not read. In this case, the following LDPR instruction is stored in a one register peripheral stack.

If the READY is maintained high, the following LDPR instructions store the data coming from the Registers File on the same register stack.

Figure 8.6. One Line Output Handshake

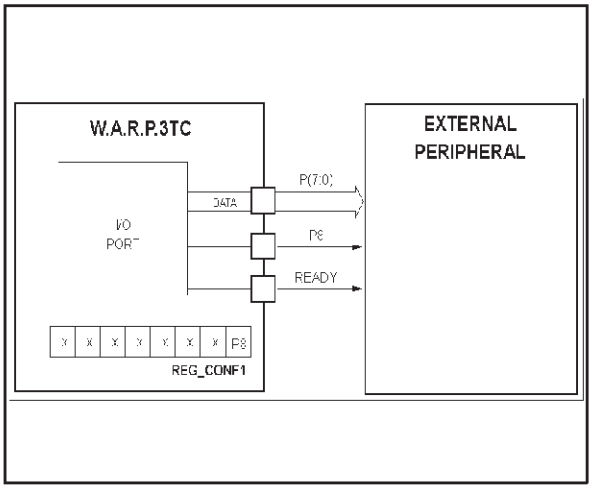
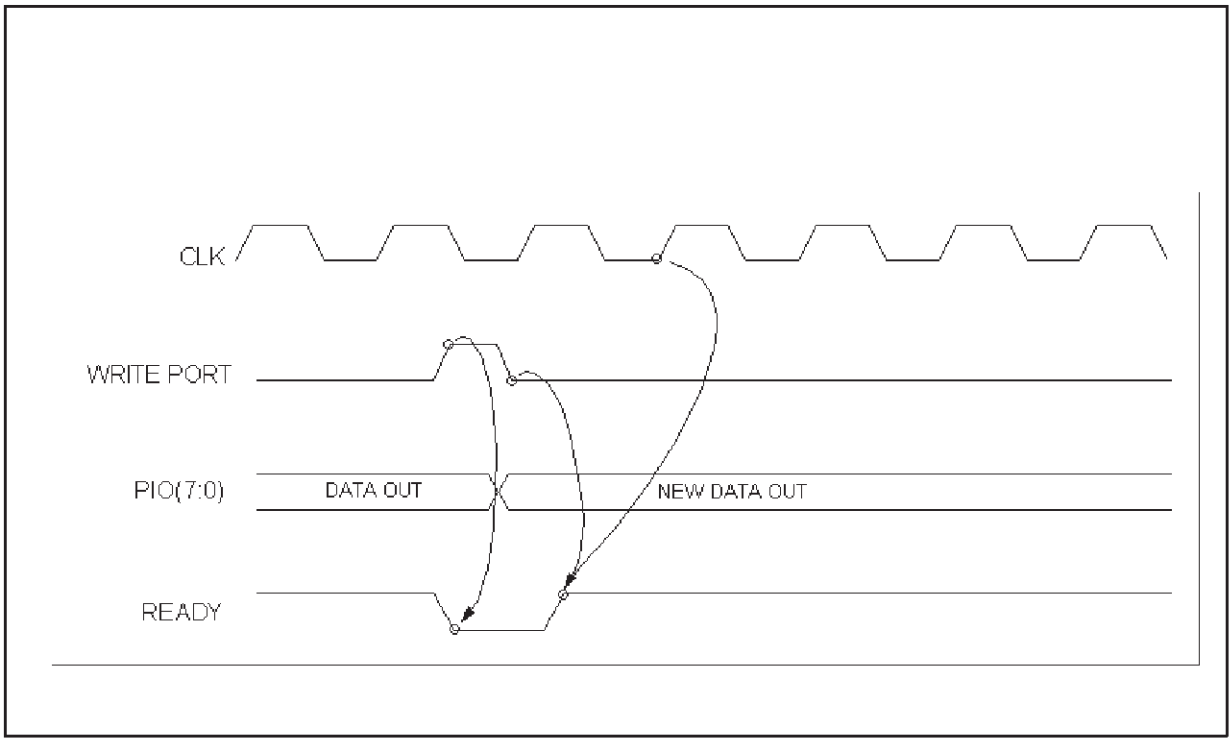


Figure 8.7. One Line Output Handshake Timing

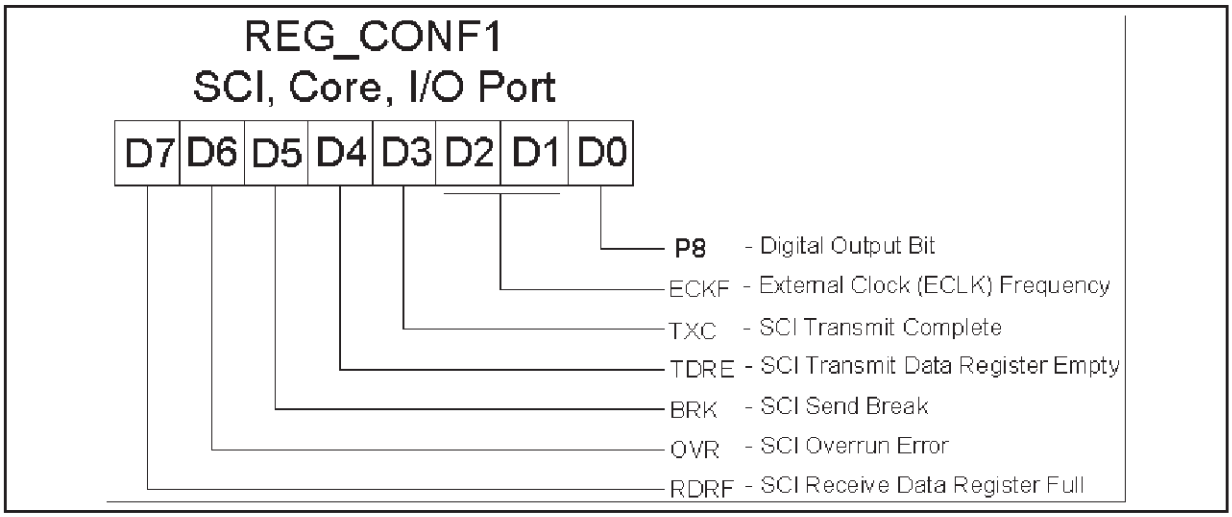


It means that each LDPR instruction deletes the old value contained in the parallel port stack register and rewrite a new value on the same stack register. Only the last LDPR instruction is executed if the READY signal is maintained high during several LDRP instructions.

Table 8.2 Configuration Register 1 Setting

Bit	Name	Value	Description
0	P8	-	Digital Output Bit
1	ECKF	00	5 MHz
		01	10 MHz
2		10	20 MHz
		11	20 MHz
3	TXC	0	SCI End Transmission Interrupt Disabled
		1	SCI End Transmission Interrupt Enabled
4	TDRE	0	SCI Transmission Data Register Empty Interrupt Disabled
		1	SCI Transmission Data Register Empty Interrupt Enabled
5	BRK	0	SCI Break Error Interrupt Disabled
		1	SCI Break Error Interrupt Enabled
6	OVR	0	SCI Overrun Error Interrupt Disabled
		1	SCI Overrun Error Interrupt Enabled
7	RDRF	0	SCI Received Data Register Full Interrupt Disabled
		1	SCI Received Data Register FullInterrupt Enabled

Figure 8.8.



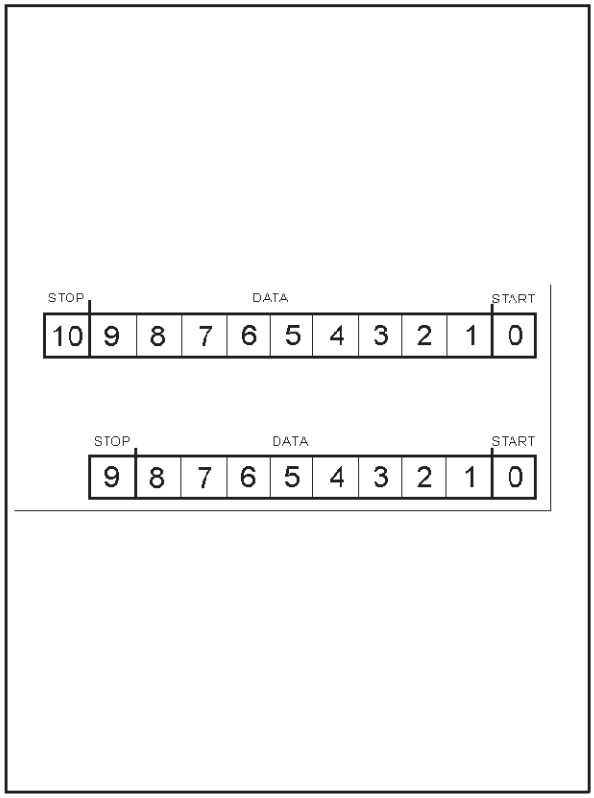
9 SERIAL COMMUNICATION INTERFACE

The Serial Communication Interface (SCI) integrated into the fuzzy processor ST52x301 provides a general purpose shift register peripheral, that allows to link several widely distributed MCUs, through their SCI subsystem. The SCI gives a serial interface providing communication with common baud rates, up to 38400 Hz, and flexible character format.

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

The SCI is composed of three modules: Receiver, Transmitter and Baud-Rate Generator and it is configured by means of Configuration Registers 3 and 1.

Figure 9.1. SCI transmitted word structures



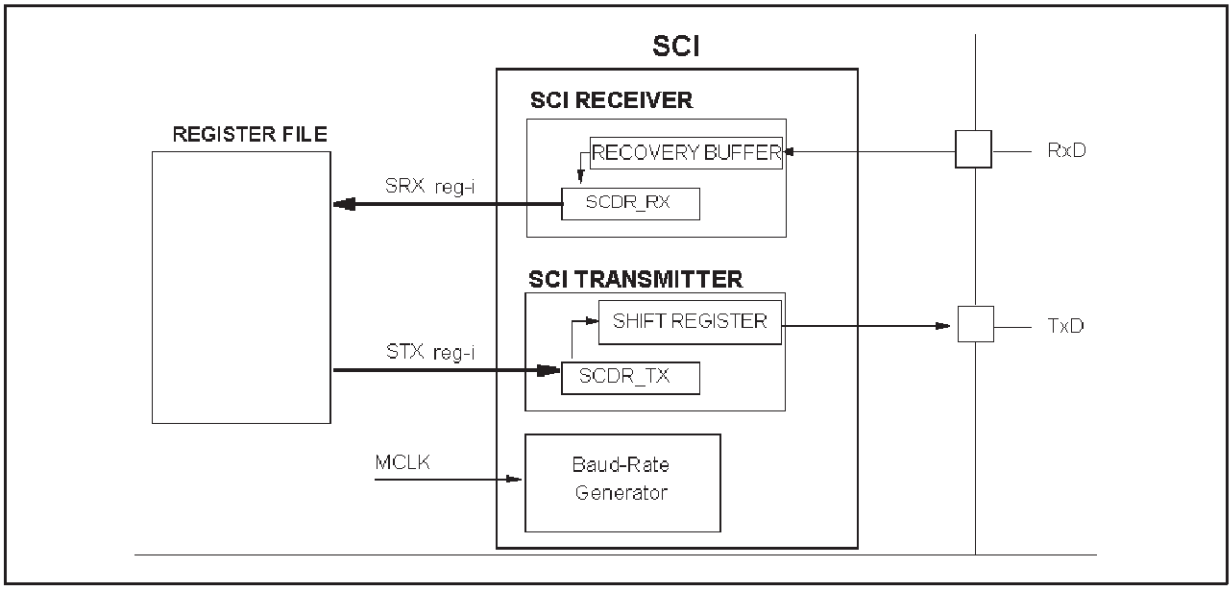
9.1 SCI RECEIVER BLOCK

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

The RE configuration bit set to ”1” (Configuration Register 3) enables the SCI Receiver.

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

Figure 9.2. SCI Block Diagram



the START condition, the Noise error and the Frame error.

When the SCI Receiver is in IDLE status, it is waiting for the START condition, that is obtained with a logic level 0, consecutive to a logic level 1. This condition is detected, if, with the fixed sampling time, three logic levels 0 are sampled after three logic levels 1.

The recognition of the START bit forces the SCI Receiver Block to enter in an data acquisition sequence, according to serial mode.

The 2 bits, M, of the Configuration Register 3 allow to define the serial mode with the convention shown in table 9.2.

The bit, T8, in case of $M = 10$ is used to set the parity check to perform, as indicated in the previous table 9.2.

The recognition of STOP condition allows to transfer the received data, from Recovery Buffer to SCDR_RX buffer, adding the eventual ninth data bit, according to the meaning shown in the previous table 9.2. After this operation, RXF flag of SCI Status Input Register 8 (fig. 9.3) is set to logic level 1. The Control Unit reads the data from SCDR_RX buffer (in read-only mode) with SRX instruction and provides a reset at logic level 0 to RDRF flag.

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

Recovery Buffer Block

This block is structured as a synchronised finite state machine on the CLOCK_RX signal falling edge.

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

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

Table 9.1 Configuration Register 3 Setting

Bit	Name	Value	Description
0	TE	0	Transmission DISABLED
		1	Transmission ENABLED
1	RE	0	Receiver DISABLED
		1	Receiver ENABLED
2	M	00	8, No Parity, 1 bit stop
		01	8, No Parity, 2 bit stop
3		10	8, Parity, 1 bit stop
		11	9, No Parity, 1 bit stop
4	T8	0	Parity Odd, if Parity is selected (M = 10); otherwise 9th Data bit
		1	Parity Even, if Parity is selected (M = 10); otherwise 9th Data bit
5	BRSL	000	600 Hz
		001	1200 Hz
		010	2400 Hz
6		011	4800 Hz
		100	9600 Hz
		101	19200 Hz
7		110	38400 Hz
		111	External Clock

The analysis of RxD input signal is carried out looking three samples for each bits received.0

If these three samples are not equal, then the noise error flag, NSERR, of Input Register 8 is set to 1 and the received data value will be the one assumed by the majority of the samples.

By means of the procedure described above, to avoid SCI becomes IDLE, because of a limited noise due to an erroneous sampling, the transmission is recognized as correct and the noise flag error is set.

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

At the end of data reception, Recovery Buffer Block, will supply information on eventual frame errors by setting to 1 FRERR flag bit of Input Register 8.

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

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

SCDR_RX block

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

The SCDR_RX block waits the signal of complete reception, from the Recovery Buffer, to load the word received. Moreover, the SCDR_RCX block loads the values of FRERR and NSERR flag bits (Input Register 8), and sets the RXF flag to 1.

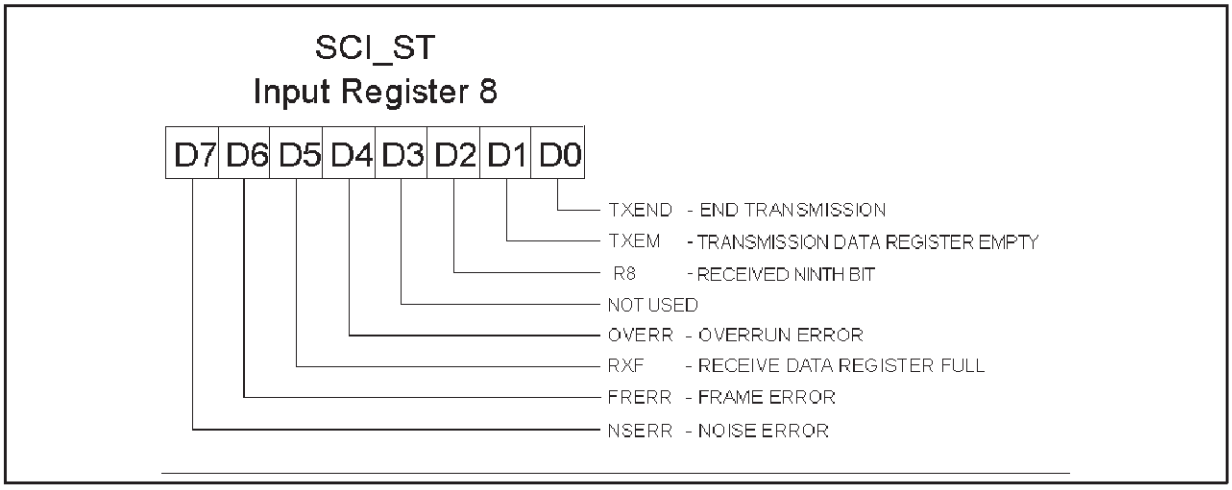
Using SRX instruction the data are transferred to Register File and RXF flag is reset to 0, to indicate SCDR_RX block is empty.

If a new data arrives before the previous one has been transferred to Register File, an overrun error occurres and OVERR flag, of Input Register 8, is set to 1.

Table 9.2 Configuration Register 1 Setting

Bit	Name	Value	Description
0	P8	-	Digital Output Bit
1	ECKF	00	5 MHz
		01	10 MHz
2		10	20 MHz
		11	20 MHz
3	TXC	0	SCI End Transmission Interrupt Disabled
		1	SCI End Transmission Interrupt Enabled
4	TDRE	0	SCI Transmission Data Register Empty Interrupt Disabled
		1	SCI Transmission Data Register Empty Interrupt Enabled
5	BRK	0	SCI Break Error Interrupt Disabled
		1	SCI Break Error Interrupt Enabled
6	OVR	0	SCI Overrun Error Interrupt Disabled
		1	SCI Overrun Error Interrupt Enabled
7	RDRF	0	SCI Received Data Register Full Interrupt Disabled
		1	SCI Received Data Register FullInterrupt Enabled

Figure 9.3. SCI Status Input Register



9.2 SCI TRANSMITTER BLOCK

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

The whole block receives through Configuration Register 3 (M bits) the settings for the following transmission modes (see table 9.1):

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

In case of 9 bit frame transmission, the most significative bit arrives through T8 of the Configuration Register 3.

In an 8-bit transmission, instead, T8 is used to configure the SCI, according to information contained in M (see table 9.1): in particular to chose the polarity control (even or odds) to implement the parity check.

After a RESET signal, RST, the SCDR_TX block is in IDLE state until it receives enabling signal, TE=1, of Configuration Register 3.

If TE=1, using STX instruction the data, to be transmitted, are transferred from Register File to SCDR_TX block and the flag of Input Register 8, TXEM, is reset to 0, to indicate SCDR_TX block is full.

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

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

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

If ST52x301 core resets TE to 0, the transmission is interrupted, but the SCI Transmitter block completes the transmission in progress before to reset.

9.3 Baud Rate Generator Block

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

Table 9.1. shows the set of frequencies selected by means of BRSL (Configuration Register 3).

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

If BRSL is equal to 111, CLOCK_RX and CLOCK_TX signals coincide with clock master, CKM.

10 TRIAC/PWM DRIVER

ST52x301 offers a peripheral able to generate a signal on pin 24, TRIACOUT, to drive an external device, like a TRIAC, a IGBT or a Power Mos. Triac/PWM driver can perform 3 different working modes according to REG_CONF10 bits, MODE (see Table 10.4):

MODE = "01": PWM

MODE = "10": Burst Mode Triac Control (Thermal Regulations)

Note: i**n this case CKSL of REG_CONF10 must be set to "1x". (see Table 10.4)**

MODE = "11": Phase Angle Partialization Triac Control (Motor Control)

The Triac/PWM Driver can be initialized by using a value fixed by a control algorithm, that can be either the output of a fuzzy inference or the result of an arithmetic calculus stored in the Register File.

In the latter case, by using the `LDPR 1,reg-i` instruction, the value, contained in the i-th register of Register File, is stored in the Triac Driver/PWM peripheral register PERIPH_REG_1.

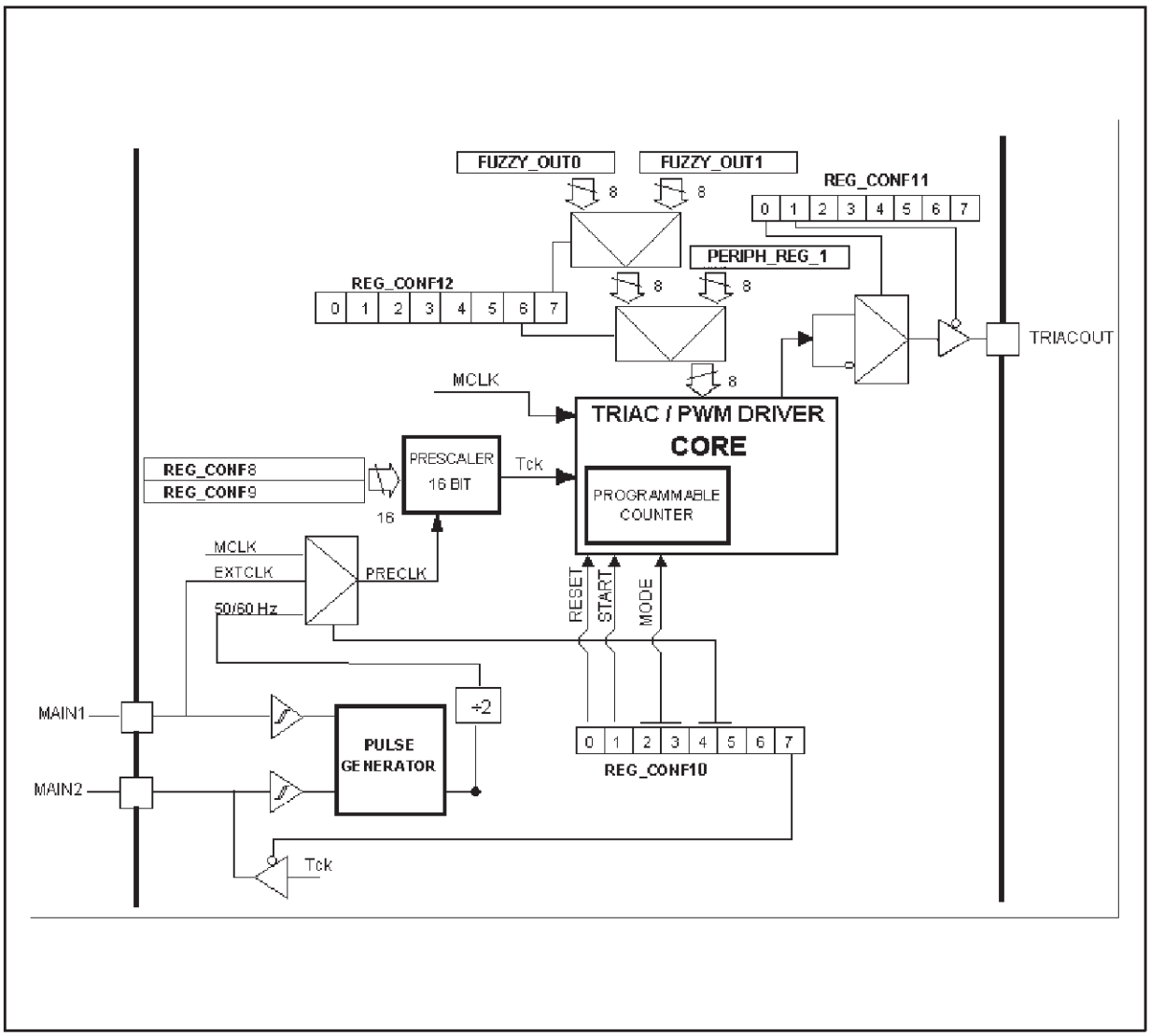
Figure 10.1 shows the internal structure of Triac/PWM Driver.

PWM Mode

The PWM working mode is obtained by setting REG_CONF10 bits, MODE, at "01" value.

It consists of a signal, with fixed period , whose duty cycle can be modified.

Figure 10.1. TRIAC/PWM Driver Simplified Block Diagram



The PWM period can be generated, internally, by dividing the master clock or, externally, by using an external clock signal.

In both cases, the clock signal is divided by a 16-bit Prescaler, managed by REG_CONF8 and REG_CONF9 (see Figure 10.2).

The duty cycle is fixed by a value, that can be either the output of a fuzzy inference or the result of an arithmetic calculus. In the first case, it can be loaded directly in the register of the peripheral, otherwise it can be stored in one location of the Register File for further manipulations and then used for the control of the PWM.

Burst Mode

It is based on turning on and off the TRIAC, for a fixed integer number of main voltage periods, in order to control the power transferred to the load.

For this reason a Burst Mode TRIAC control consists of a signal, with a period, T, containing an integer number of the main voltage periods, whose duty cycle is proportional to the number of periods in which the TRIAC is ON (Duty Cycle). This kind of Triac control is mainly used for thermal regulation.

The duty cycle is fixed by a value that can be directly the output of a fuzzy inference or the result of an arithmetic calculus.

In order to work in Burst mode, it is necessary to detect the pre-post zero-crossing of main voltage, by using an external inserting circuitry.

The user can define the period T, by means of the internal 16-bit prescaler, setting REG_CONF8 and REG_CONF9 (see Figure 10.2). T is proportional to the main voltage period, it is in the range 0 to 21.8 sec (if the main frequency is 50Hz).

The width and the polarity of the pulses can be programmed according to the Triac and the circuit characteristics.

Phase Angle Partialization Mode

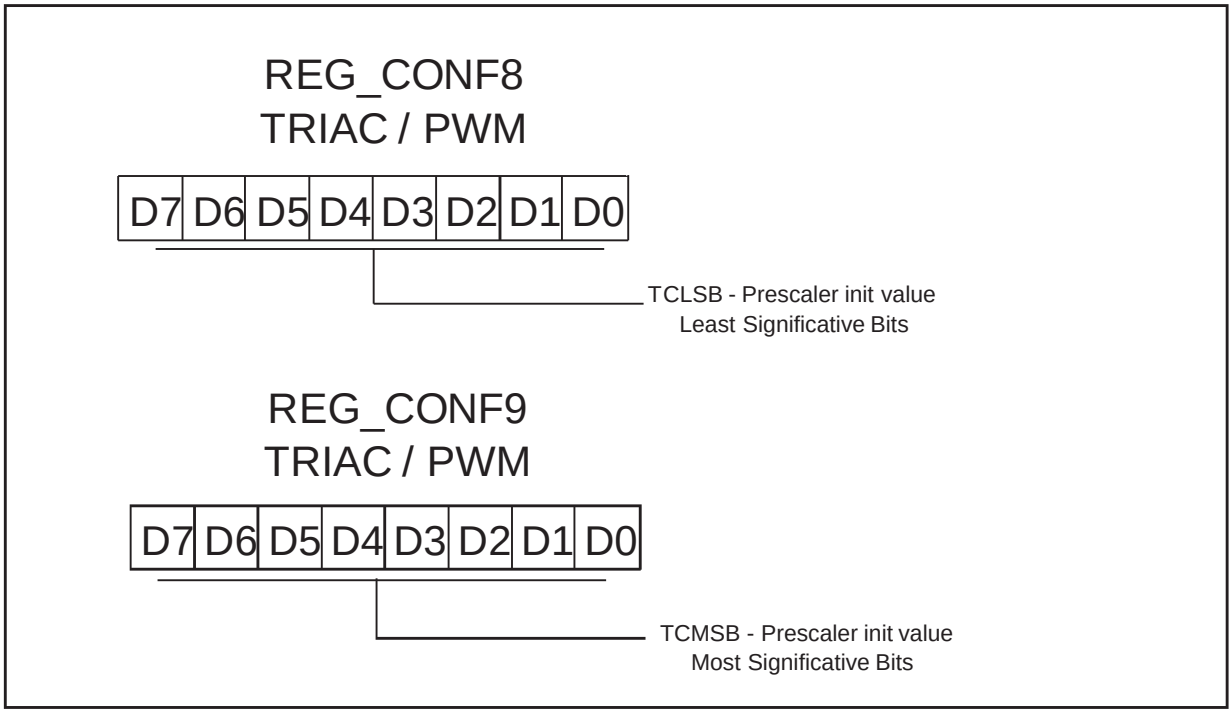
This method is based on turning on the TRIAC only for a part (phase angle) of each main voltage period. When the phase angle is large the energy (power) supplied to the load is low, viceversa, when the phase angle is small the energy supplied to the load is high.

The phase angle can be fixed by a fuzzy algorithm or by a value stored in the Register File.

The phase angle is an 8-bit value.

The peripheral is programmable in order to work with a main voltage frequency of 50 or 60 Hz.

Figure 10.2. TRIAC/PWM Configuration Register 8 and 9



10.1 PWM GENERATOR WORKING MODE

When REG_CONF10 (3:2) bits, MODE, are ”01”, the peripheral is programmed to work in PWM Mode.

By using the 16-bit prescaler, the PWM period can be generated by dividing the internal master clock, or an external clock signal applied on the pin MAIN1, or the main voltage frequency, by using the circuit shown in Figure 10.6.

NOTE: The external clock signal, applied on MAIN1 pin, must have a frequency at least three time smaller than the internal master clock.

The clock source can be selected by using REG_CONF10(5:4) bits, CKSL (see Table 10.4 and Figure 10.9). If the clock source selected is not the main voltage frequency (CKSL=1x), MAIN2 pin can be configured as input or output, by using REG_CONF10(7) bit, IOSL (see Table 10.4).

If MAIN2 is an output, on this pin it is possible to get the prescaler output signal Tck.

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

T=256*Tck

where Tck is the output of the 16-bit prescaler managed by REG_CONF8 and REG_CONF9 (see Figure 10.2).

NOTE. In PWM working mode, the value N, stored in the 16-bit prescaler, must be in the range from 2 to 2^{16}-1

By using a 20 MHz clock master it is possible to obtain a PWM frequency in the range 1.2 Hz to 26.04 KHz.

The value Ton is proportional to a value, INIT_VALUE, that can be a fuzzy output or a value coming from Register File, according with the INPSL and FZSL configuration bits of REG_CONF12 (see Table 10.6 and Figure 10.12).

The Ton is equal to:

Ton= INIT_VALUE*Tck.

It means the Ton can be fixed by the control algorithm that can be either the output of a fuzzy inference or the result of an arithmetic calculus. In the second case, the data, stored in the i-th location of the Register File, can be loaded by using the instruction:

`LDPR 1, reg-i`.

If the INIT_VALUE is 255 the Toff is equal to Tck.

Table 10.1. MODE - Triac/PWM Working Mode Settings

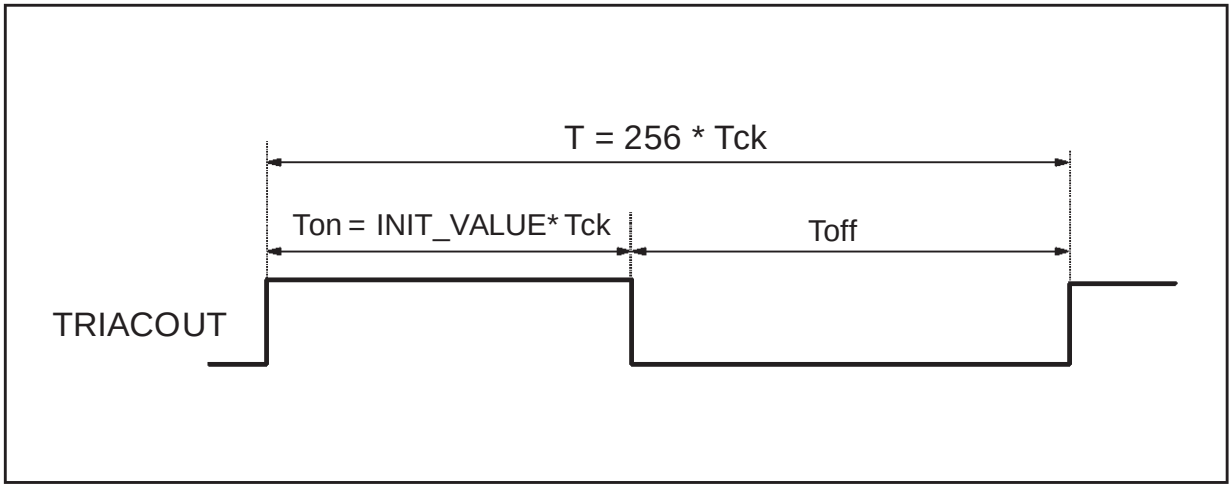
Value	Description
01	PWM Driver
10	Burst Mode Control (1)
11	Phase Angle Control

Note: (1) REG_CONF10(5) must be set to ”1”

Table 10.2. PWM Frequencies

MCLK Frequency	1/T	
	min	max
5 MHz	0.3Hz	6.51 kHz
10 MHz	0.6 Hz	13.02 kHz
20 MHz	1.2 Hz	26.04 kHz

Figure 10.3. PWM Functionament



10.2 BURST MODE

When REG_CONF10 (3:2) bits, MODE, are ”10” the peripheral is programmed to work in BURST MODE.

Notice that when you are working in Burst mode **CKSL must be set to ”1x”**. (see Table 10.4)

A square wave, Tb, is generated with a duty cycle proportional to the power the user intends to transfer on the load. A pulse is generated for each zero crossing of the main voltage included in the Ton of the fixed period. Figure 10.4 shows the typical Burst Control working mode. The period T of the signal Tb (see Figure 10.4) is equal to 256*Tck.

The signal Tck is generated programming the 16-bit Prescaler, by REG_CONF8 and REG_CONF9 (see Figure 10.2). Tck is equal to the main voltage frequency (50 or 60 Hz) divided by N+1, where N value is from 0 to 2^{16}-1.

The value Ton is proportional to a value, INIT_VALUE, that can be a fuzzy output or a value coming from Register File, according with the INPSL and FZSL configuration bits of REG_CONF12 (see Table 10.6 and Figure 10.12).

On TRIACOUT pin is generated a sequence of pulses, programmed, by using REG_CONF11(0) bit, POL (see Table 10.5), in order to be positive or negative, to drive the Triac in different quadrants. The number of generated pulses, N_PULSES, is:

N_PULSES = 2 [(N+1)*INIT_VALUE - N]

where N is the value stored in the 16-bit pescaler.

Then Ton = (N_PULSES / 2)* $T_{POWER\ LINE}$

The first pulse is obtained during the first zero crossing of the main voltage and the last one is generated after INIT_VALUE*Tck clock pulses, where Tck is the Prescaler output, generated by using the main voltage frequency applied to MAIN1 and MAIN2 pins.

The peripheral can be programmed in order to work with 50 or 60 Hz main voltage frequency, by setting the REG_CONF10(6) bit, PSF (see Table 10.4).

Ranges of the Tb signal period depend on the power line frequency (see Table 10.3).

In order to drive a Triac in Burst Mode it is required to generate a sequence of pulse, that must be centred on the zero crossing of the power line as shown in the Figure 10.7. For this reason, the pre zero crossing and the post zero crossing of the power line must be detected.

To detect the zero-crossing and get also the main voltage frequency, the user must generate MAIN1 and MAIN2 signals, by using the circuit shown in Figure 10.6.

MAIN1 and MAIN2 signals are used in the block called PULSE GENERATOR of the peripheral (see Figure 10.1).

In particular the pulses are generated by using the rise edge of the signal MAIN1 and the falling edge of the signal MAIN2.

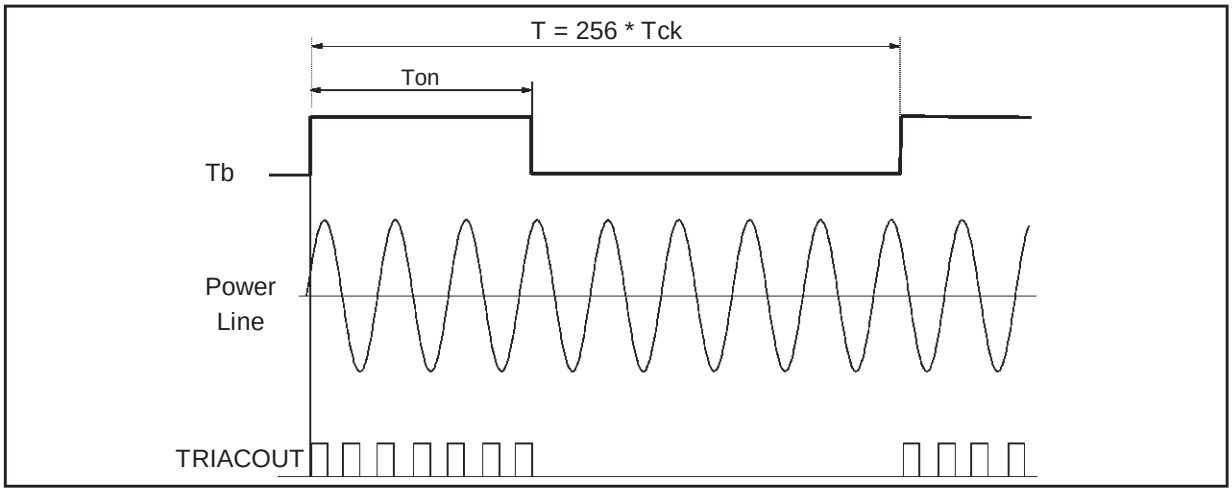
Figure 10.5 shows the generation of the Triac pulses Tp .

The first firing pulse for the Triac is generated on the zero crossing of the power line, while the next pulses are centred on the zero crossing.

Table 10.3. TRIACOUT Signal Period

Power Line Frequency	T	
	min	max
50 Hz	5.12 s	335544 s
60 Hz	4.26 s	279620 s

Figure 10.4. Burst Working Mode



Normally the Triac firing pulses start 1/3 Tp before the zero crossing and the length of the pulses is Tp, see Figure 10.5.

The length Tp of the pulses is programmable by using UTP value, that is a 14-bits number, obtained with REG_CONF12(5:0) bits, UTPMSB, and REG_CONF13, UTPLSB (see figure 10.12 and table 10.6):

UTP(13:0) = [UTPMSB(5:0) UTPLSB(7:0)]

$T_P = T_{MCLK} * UTP$

The value Tp is in the range 0 to 4.9 ms when the clock master is 20 MHz.

According to REG_CONF11(0) configuration register bit, POL, it is possible to set the firing pulses polarity; in order to obtain positive or negative gate Triac currents, allowing to work respectively in I and IV quadrants, or in the II and III quadrants (see Figures 10.5 and 10.12).

To increase the immunity of the peripheral against the electrical noise of the main voltage, a programmable masking time, by using REG_CONF11(5:2) bits, TCMSK (see Table 10.5)

MCLK Frequency	T_P	
	min	max
5 MHz	0.0012 ms	19.6608 ms
10 MHz	0.0006 ms	9.8304 ms
20 MHz	0.0003 ms	4.9152 ms

Figure 10.5. Burst Mode pulse polarity

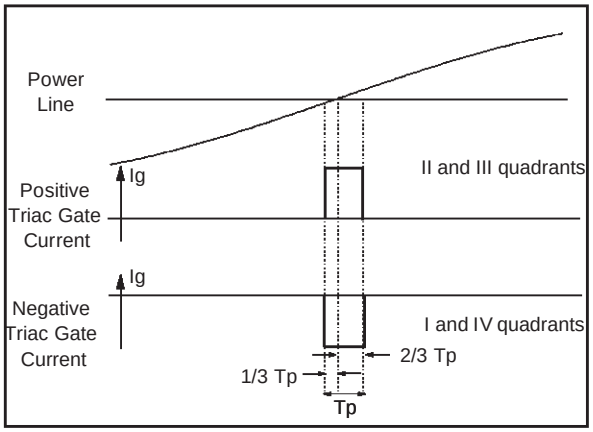


Figure 10.7 Burst Mode Zero Crossing

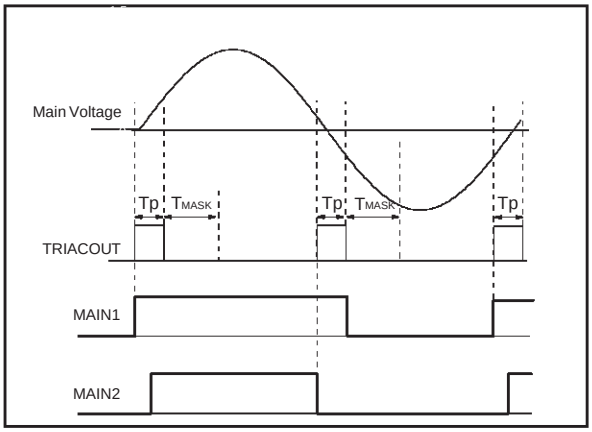
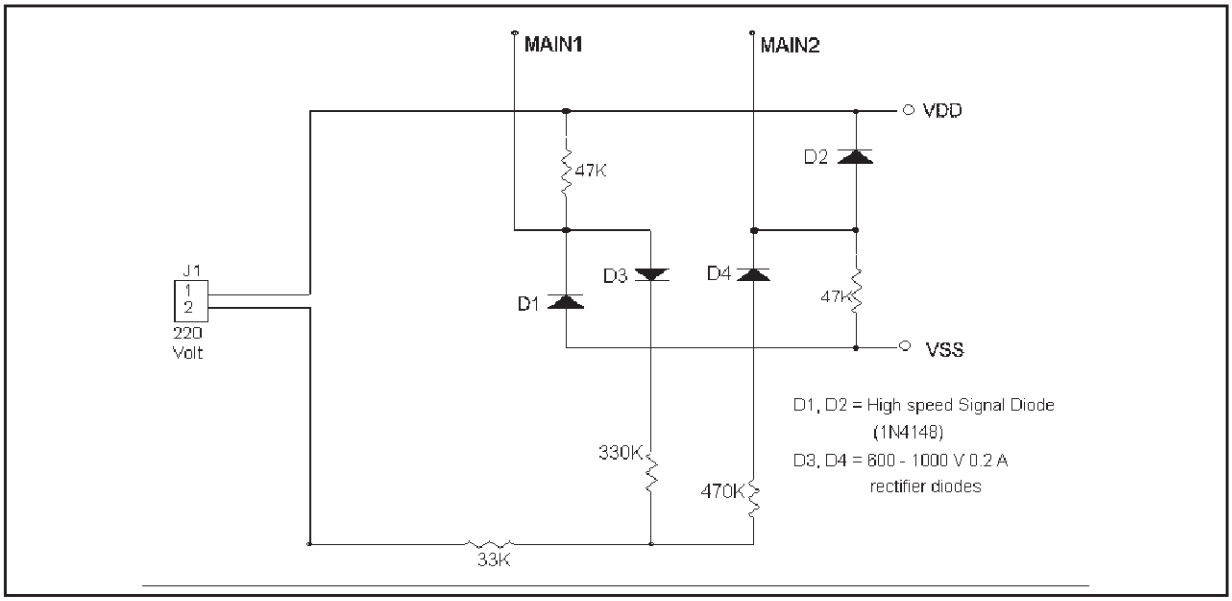


Figure 10.6 Burst Mode Zero Crossing Circuit



and Figure 10.11), is introduced after each firing pulse (see Figure 10.7):

Masking time =(2^TCMSK*200 +100) nS.

If TCMSK is 0 then Masking time is 0.

In fact, to avoid the detection of electrical noise, during the masking time no signal, coming from MAIN1 and MAIN2, is taken into account.

Working in the II and III quadrant the peripheral implements the following procedure:

1) The firing pulse is set to ”1” on the rising edge of MAIN1.

2) The firing pulse is reset to ”0” after the time Tp fixed by program.

3) The firing pulse is reset to ”0” for a time equal to the fixed masking time.

4) On the falling edge of MAIN2 the firing pulse is set to ”1”

5) The firing pulse is reset to ”0” after the time Tp fixed by program.

6) The firing pulse is reset to ”0” for a time equal to the fixed masking time.

Following this approach it is possible to filter electrical noise and oscillations on the signal MAIN1 and MAIN2.

It is possible to generate a programmable Interrupt in four different ways:

1) No Interrupt;

2) Interrupt on the rising edge of the signal Tb.

3) Interrupt on the falling edge of the signal Tb.

4) Interrupt on both the edge of the signal Tb.

The Interrupt is programmable by using the register REG_CONF11(7:6), INTSL (see Table 10.5).

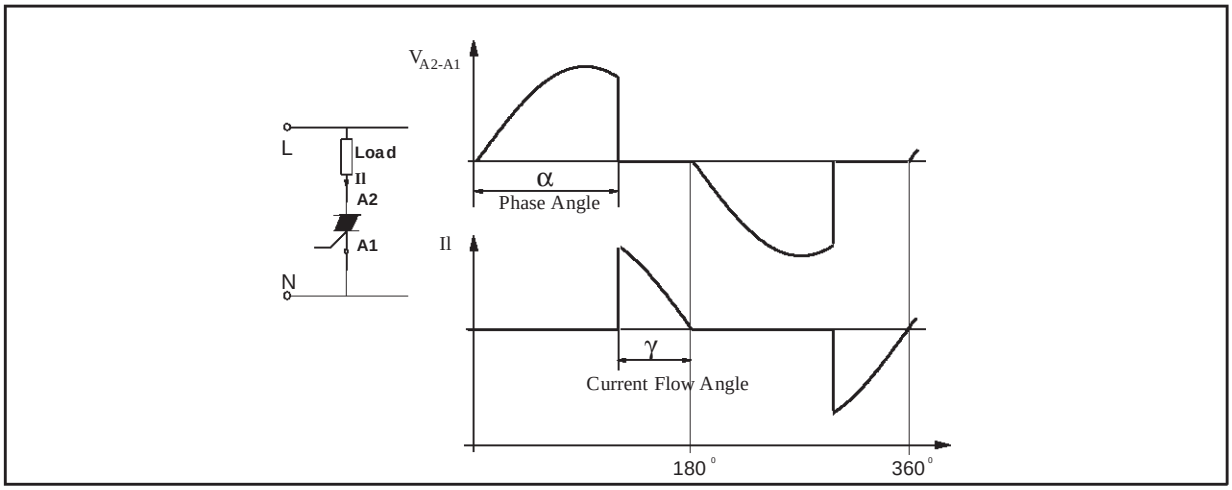
TCMSK	Masking Time
0000	0 µs
0001	0.5 µs
0010	0.9 µs
0011	1.7 µs
0100	3.3 µs
0101	6.5 µs
0110	12.9 µs
0111	25.7 µs
1000	51.3 µs
1001	102.5 µs
1010	204.9 µs
1011	409.7 µs
1100	819.3 µs
1101	1638.9 µs
1110	3276.9 µs
1111	6553.7 µs

10.3 PHASE ANGLE PARTIALIZATION WORKING MODE

When REG_CONF10 (3:2) bits, MODE, are ”11” the peripheral is programmed to work in PHASE ANGLE PARTIALIZATION mode.

In this mode Triac is controlled each period of the main voltage. The power transferred to the load is proportional to the CURRENT FLOW ANGLE γ. This kind of Triac control is suitable to drive the Triac

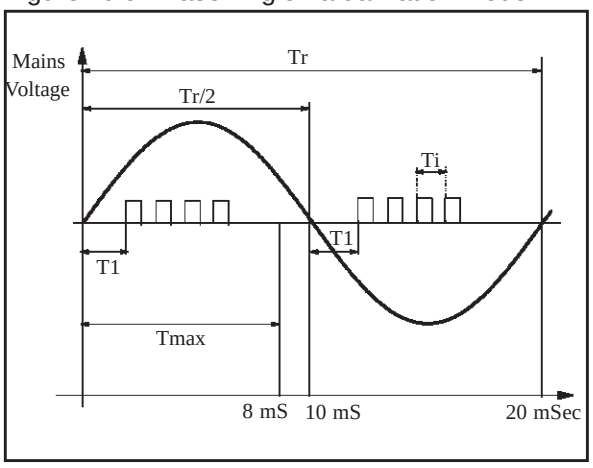
Figure 10.8. Phase Angle Partialization Mode



with inductive load (i.e. universal or monophase motors). In the figure 10.8 is shown the relation between the Phase Angle α and the Current flow angle γ. The peripheral allows to control the Phase Angle or equivalently the time T1 (see Figure 10.9). It is possible to change Time T1 setting the contents of the peripheral register PERIPH_REG_1. This value could be directly loaded by using one of the two fuzzy outputs or by using a value coming from the Registers File, according with INPSL and FZSL configuration bits of REG_CONF12 (see Table 10.6 and Figure 10.13).

In order to synchronize the peripheral with the zero crossing of the main voltage the two pins MAIN1 and MAIN2 must be connected together if the external circuit is the one shown in the Figure 10.10. It is possible to use different circuits for the zero crossing detection, but the MAIN1 signal rising edge must be synchronized with a main voltage zero crossing and the MAIN2 signal falling edge must be synchronized with the following main voltage zero crossing, always.

Figure 10.9 Phase Angle Partialization mode



The peripheral can be programmed in order to work with 50 or 60 Hz main voltage frequency by setting the REG_CONF10(6) bit, PSF (see Table 10.4).

If main voltage frequency is equal to 50 Hz, then Tr, see figure 10.9, is equal to 20 mSec and T1 is:

T1 = PERIPH_REG_1(0:7)*(1/25.5) ms.

The length of the semiperiod Ti/2 is programmable by using the registers REG_CONF12(0:5) and REG_CONF13, (see figure 10.12). By using a clock master equal to 20 MHz the pulse width is in the range from 0.2 to 250 µs. The duty cycle of Ti is always 50 %.

In order to avoid problems for the Triac firing when the load is inductive 8 different pulses are generated by the peripheral.

If the time T1 is bigger than a fixed time Tmax then no pulses are generated and the Triac is maintained off. This feature was implemented in order to avoid the firing of the Triac in the second half period of the main voltage. The firing pulses are generated when the contents of the PERIPH_REG_1 is less or equal to 204, otherwise they are not generated.

When the frequency of the main voltage is 50 Hz, T1max is equal to 8 mSec.

It is possible to generate a programmable interrupt in four different ways:

1) no Interrupt;

2) Interrupt on the rising edge of the signal MAIN1

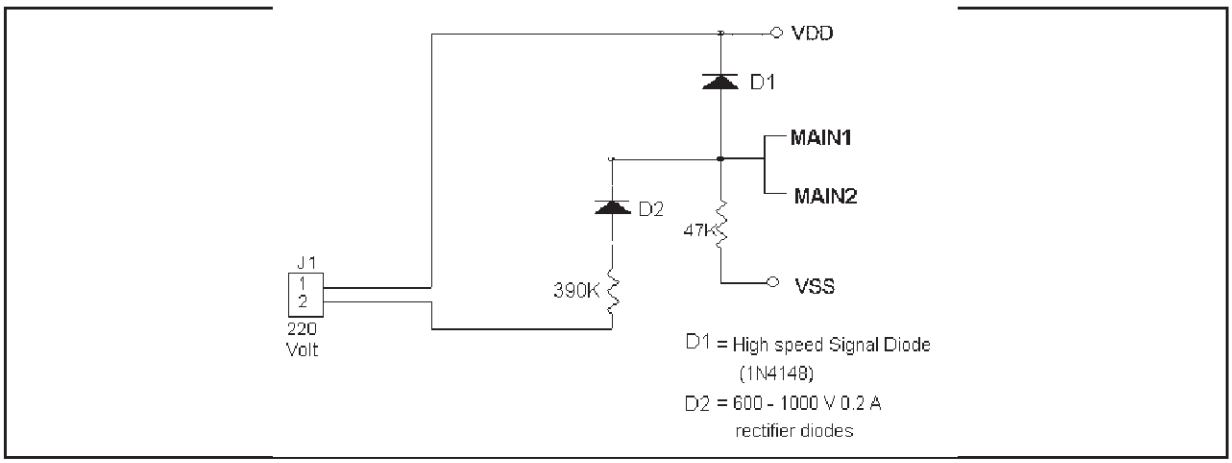
3) Interrupt on the falling edge of the signal MAIN2

4) Interrupt on both the edges of the signal MAIN1.

The Interrupt is programmable by using the register REG_CONF11(7:6), INTSL

10.4. TRIAC/PWM DRIVER PROGRAMMING

Figure 10.10 Phase Angle Partialization Zero Crossing



It is possible SET or RESET the TRIAC/PWM Peripheral by using the REG_CONF10(0) bit, TCRST (see Table 10.4).

If TRIAC/PWM Peripheral is SET, It is possible START or STOP it, by using the REG_CONF10(1) bit, TCST (see Table 10.4), to start or stop the internal counter without resetting it.

It is possible to enable the TRIACOUT, by using the REG_CONF11(0) bit, TCTRS (see Table 10.5 and Figure 10.11).

IF TCTRS is 0 the TRIAC/PWM Peripheral output is in tristate status.

Table 10.4 Configuration Register 10 Description

Bit	Name	Value	Description
0	TCRST	0	Triac Reset
		1	Triac Set
1	TCST	0	Triac Stop
		1	Triac Start
2	MODE	00	not used
		01	PWM signal Generator
3		10	Burst Mode (1)
		11	Phase Partialization
4	CKSL	00	Clock Master
		01	External Clock on MAIN1
5		1x	Main Voltage Frequency
6	PSF	0	Main Power at 50 Hz
		1	Main Power at 60 Hz
7	IOSL	0	MAIN2 Input pin
		1	MAIN2 Output pin

Note: (1) CKSL must be set to "1x"

Table 10.5 Configuration Register 11 Description

Bit	Name	Value	Description
0	POL	0	Output pulse Polarity = positive
		1	Output pulse Polarity = negative
1	TCTRS	0	TRIACOUT status = Tristate
		1	TRIACOUT status = Enabled
2	TCMSK		Masking time =(2^TCMSK*200 +100) nS. TCMSK=0 →Masking time=0
3			
4			
5			
6	INTSL	00	No Interrupt source selected
		01	Interrupt on falling edge of the TRIAC/PWM signal, or of the Main Voltage
7		10	Interrupt on rising edge of the TRIAC/PWM signal, or of the Main Voltage
		11	Interrupt on both of edges of the TRIAC/PWM signa,l or of the Main Voltage

Table 10.6. Configuration Register 12 Description

Bit	Name	Value	Description
0 ÷ 5	UTPMSB		Output Impulse Width most significative bits
6	INPSL	0	TRIAC/PWM Input from Fuzzy Output
		1	TRIAC/PWM Input from Register File
7	FZSL	0	TRIAC/PWM Input from Fuzzy Output 1
		1	TRIAC/PWM Input from Fuzzy Output 2

Figure 10.11. TRIAC/PWM Configuration Register 10

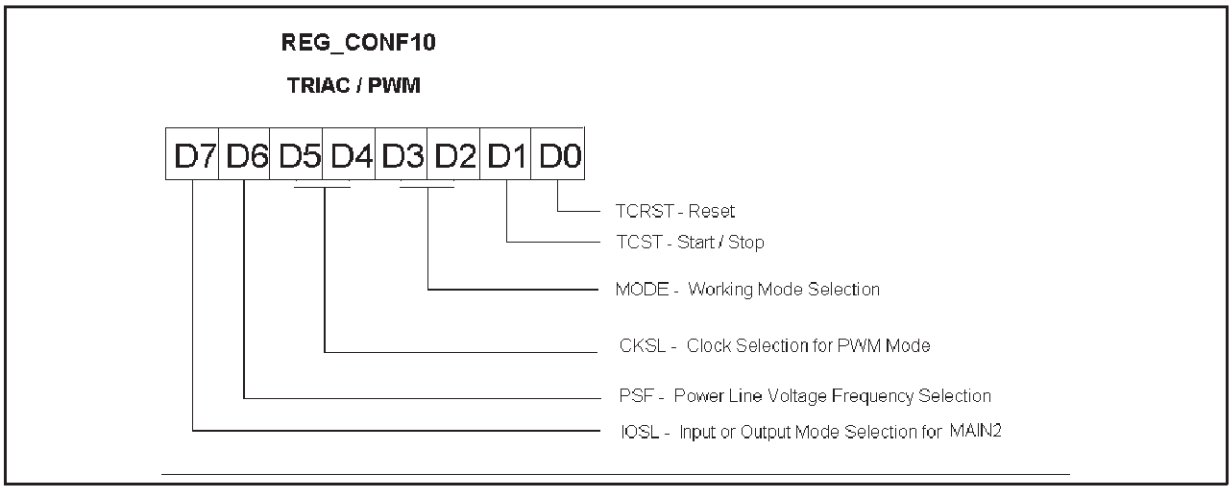


Figure 10.12 TRIAC/PWM Configuration Register 11

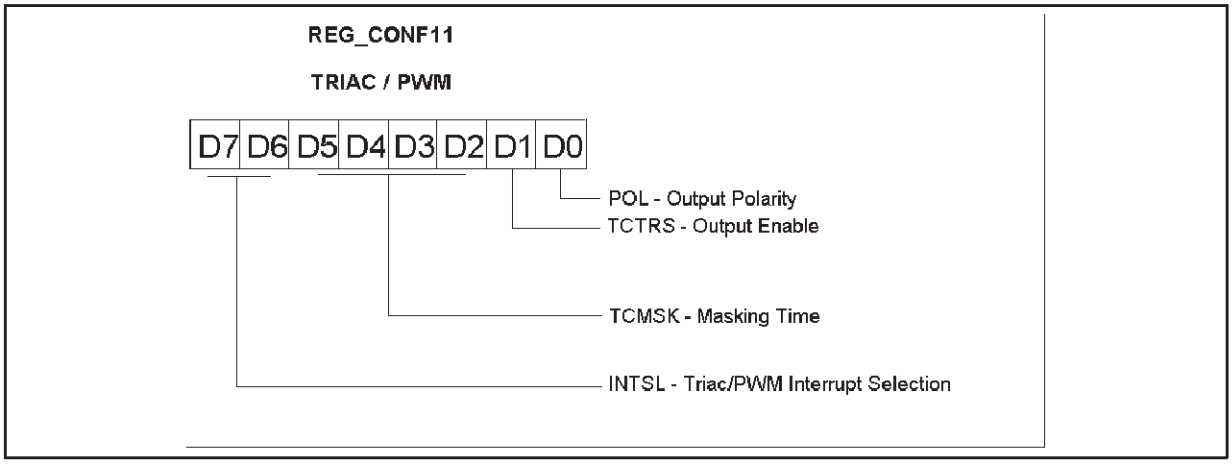
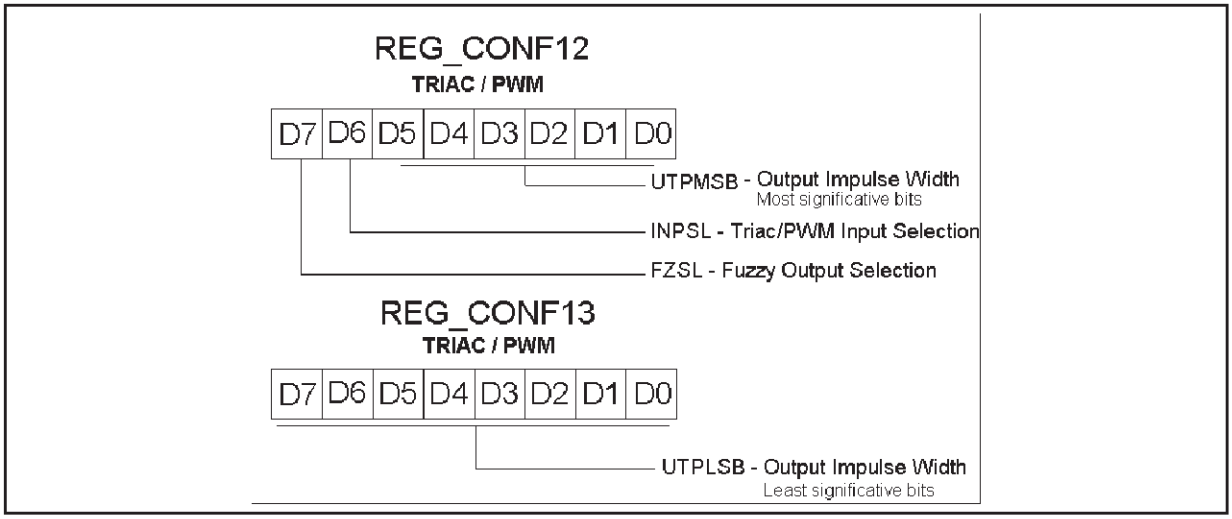


Figure 10.13 TRIAC/PWM Configuration Registers 12 and 13



11 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 voltagees.

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 11.1. 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 (2)	±20	mA
	TRIACOUT Output Source Sink Current	±80 (3)	mA
T_{OPT}	Operating Temperature	0 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 SGS-THOMSON SURE Program and other relevant quality documents.

1. Within these limits, clamping diodes are garanteed to be not conductive.
2. All except TRIACOUT pin
3. For not more than 1 sec.

(VSS or VDD) RECOMMENDED OPERATING CONDITIONS

(Operating Condition: V_{DD}=5V±5%-T_A=0 °C to 85 °C, unless otherwise specified)

Table 11.2. Recommended Operation Condition

Symbol	Parameters	Test Conditions	Min	Typ	Max	Unit
V_{DD}	Operating Supply Voltage		4.75	5.0	5.25	V
V_{PP}	Programming Voltage		11.4	12	12.6	V
V_O	Ouput Voltage		V_{SS}		VDD	V
V_{DDA}, V_{SSA}	Analog Supply Voltage	$Vss \leq V_{SSA} < V_{DDA} \leq V_{DD}$	VSS		V_{DD}	V
f_{OSC}	Oscillator Frequency (1)		5	10	20	MHz

Notes: 1. For correct behaviour of some peripherals, it is possible to work only with one of the 5 - 10 - 20 MHz frequencies.

DC ELECTRICAL CHARACTERISTICS

(Operating Condition: V_{DD}=5V±5%-T_A=0 °C to 85 °C, unless otherwise specified)

Table 11.3 DC Electrical Characteristics

Symbol	Parameter	Test Conditions	Min	Typ	Max	Unit
V_{IL}	TTL type Schmitt trig. Low Level Input Voltage	VDD =4.75 V see fig.11.6			0.7	V
	CMOS type Schmitt trig. Low Level Input Voltage	V_{DD} =4.75 V see fig.11.7			1.2	V
V_{IH}	TTL type Schmitt trig. High Level Input Voltage	V_{DD} =5.25 V see fig.11.6	2			V
	CMOS type Schmitt trig. High Level Input Voltage	V_{DD} =5.25 V see fig.11.7	3.5			V
V_{OL}	Standard Low Level Output Voltage	I_{OL} =4mA			0.4	V
	TRIACOUT Low Level Output Voltage	I_{OL} =50mA		2		V
V_{OH}	Standard High Level Output Voltage(1)	I_{OL} =-4mA	V_{DD}-0.5			V
	TRIACOUT High Level Output Voltage(1)	I_{OL} =50mA		V_{DD}-2		V
V_{Hys}	TTL type Schmitt trig. Hysteresis Voltage	see fig. 11.6		1.2		V
	CMOS type Schmitt trig. Hysteresis Voltage	see fig. 11.7		2.0		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_{OL}	Tri-State Output Leakage Current	V_O=V_{SS} or V_{DD}		±10		mA
I_{DD}	Supply Current in RESET mode	V_{PP} connected with V_{DD}; V_{RESET} =V_{SS} F_{OSC}= 10 MHz		11		mA
	Supply Current in RUN mode	V_{PP} connected with V_{DD}; F_{OSC}= 10 MHz		11		mA
I_{DDA}	Analog Supply Current in RESET mode	V_{PP} connected with V_{DD}; V_{RESET} =V_{SS} F_{OSC}= 10 MHz		3		mA
	Analog Supply Current in RUN mode	V_{PP} connected with V_{DD}; F_{OSC}= 10 MHz		10		mA

AC ELECTRICAL CHARACTERISTICS

(Operating Condition: V_{DD}=5V±5%-T_A=0 °C to 85 °C, unless otherwise specified)

Table 11.4. AC Electrical Characteristics

Symbol	Parameter	Test Conditions	Min	Typ	Max	Unit
R_S	Input protection Resistor	All Input Pins		1		kΩ
C_{IN}	Input Capacitance	All Input Pins		10		pF
C_{OUT}	Output Capacitance	All Ouput Pins		10		pF

Table 11.5. Timing Parameters

Symbol	Parameters	Test Conditions	Min	Typ	Max	Unit
f_{OSC}	Oscillator Frequency				20	MHz
t_{CLH}	Clock High		25			ns
t_{CLL}	Clock Low		25			ns
t_{SET}	Setup	see fig 11.1		5		ns
t_{HLD}	Hold	see fig. 11.1		5		ns
t_{WRESET}	Minimum Reset Pulse Width		100			ns
t_{WINT}	Minimum External Interrupt Pulse Width		100			ns
t_{IR}	Input Rise Time	see fig.11.2			15	ns
t_{IF}	Input Fall Time	see fig.11.2			15	ns
t_{OR}	Output Rise Time	C_{LOAD}=10 pF see fig.11.2		10		ns
t_{OF}	Output Fall	CLOAD=10 pF see fig.11.2		10		ns

Figure 11.1. Data Input Timing

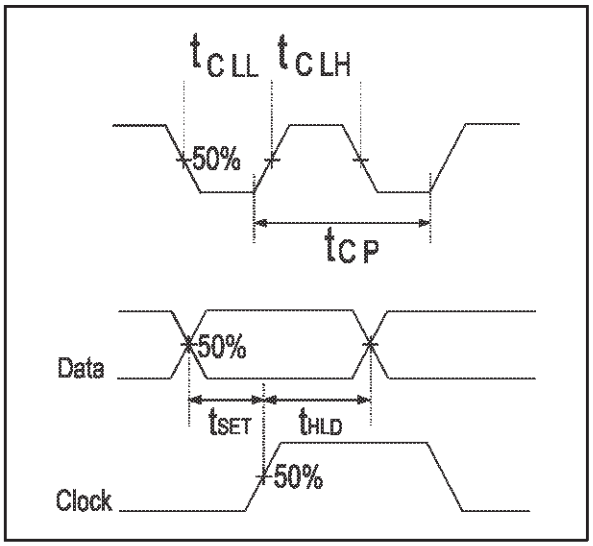


Figure 11.2. I/O Rise and Fall Timing

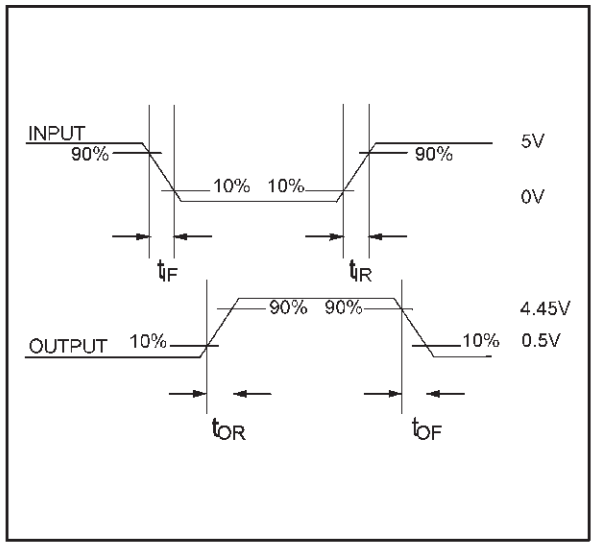


Figure 11.3. Input Pin Equivalent Circuit

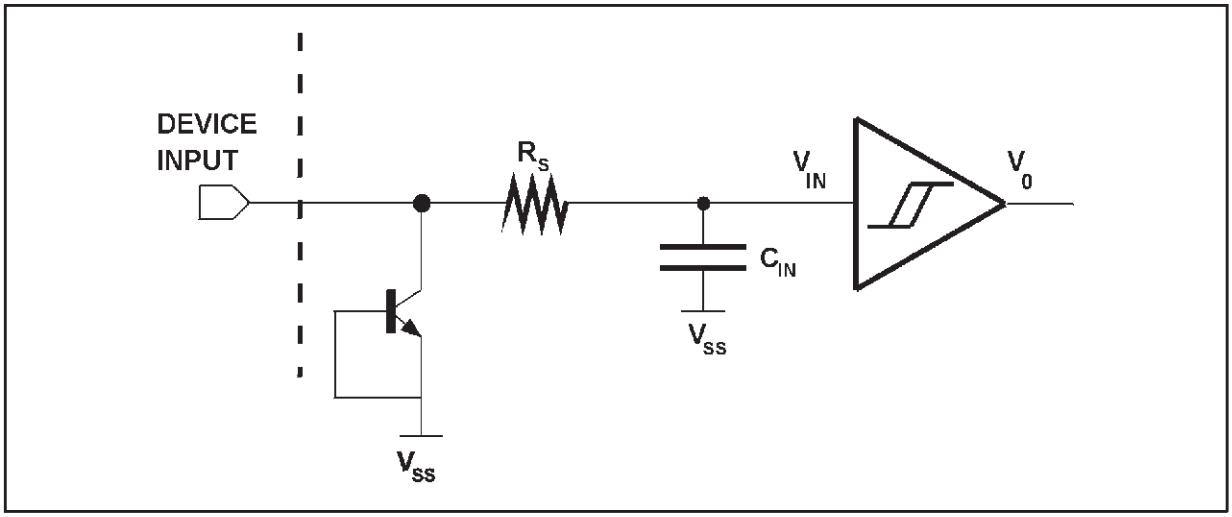


Figure 11.4. Equivalent Tristate Output Circuit

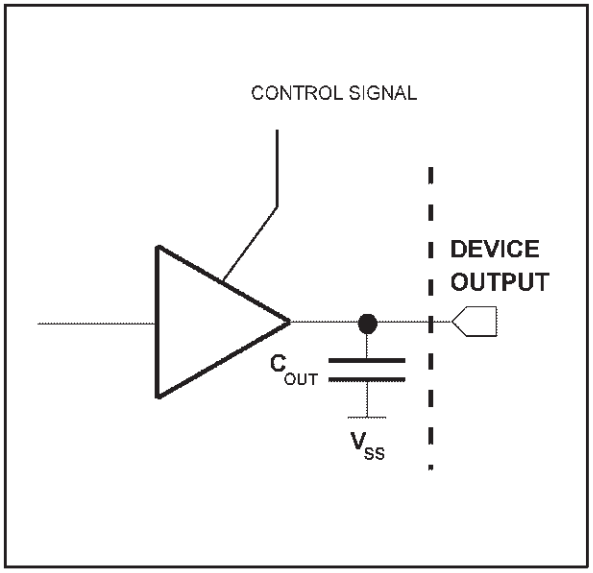


Figure 11.5. Equivalent Output Circuit

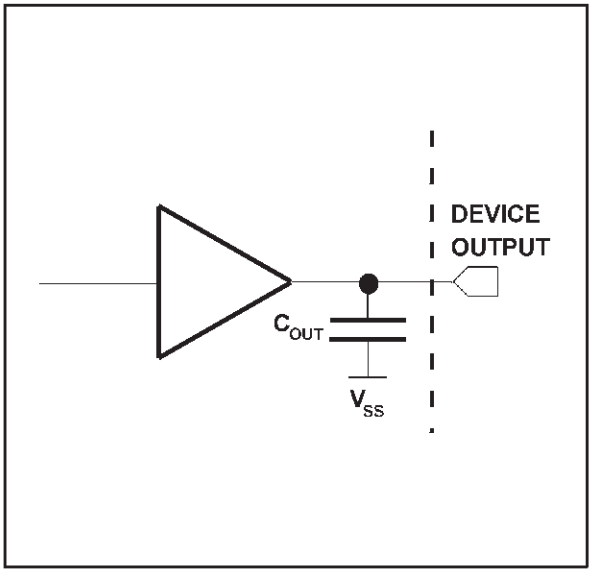


Figure 11.6. TTL-level Input Schmitt Trigger

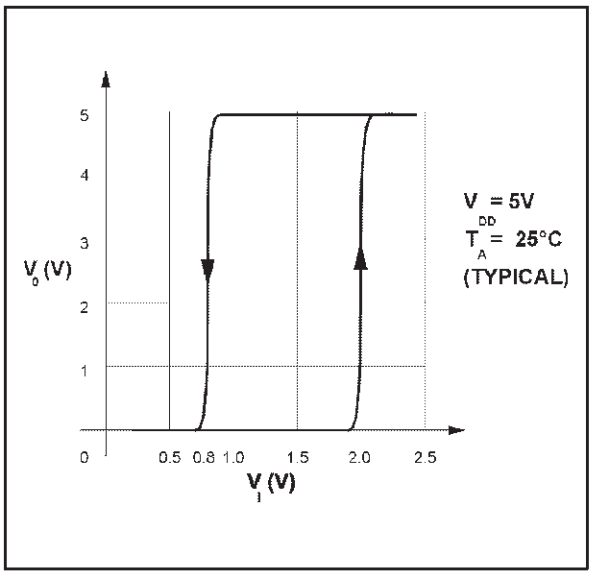
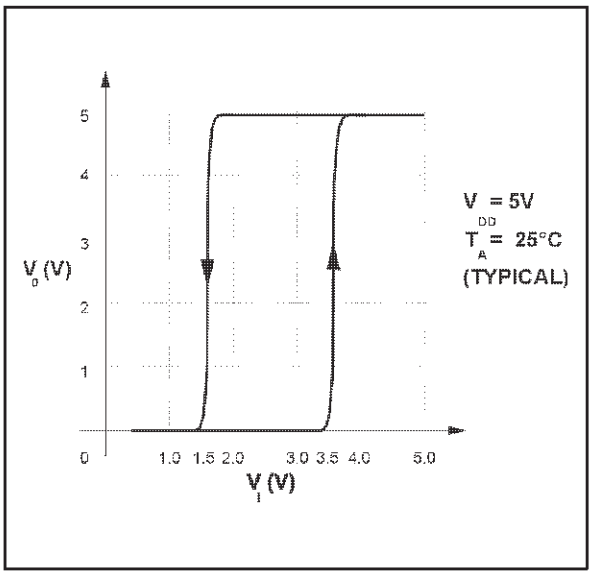


Figure 11.7. CMOS-level Input Schmitt Trigger



Note: Only for RETE1 and RETEIO signals

TIMER CHARACTERISTICS

(Operating Condition: V_{DD}=5V±5%-T_A=0 °C to 85 °C, unless otherwise specified)

Table 11.7. Timer Characteristics

Symbol	Parameter	Min	Typ	Max	Unit
t_{RES}	Resolution	$1/F_{OSC}$			µs
f_{IN}	External Input Frequency on timer Internal Input Frequency on timer			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=0 °C to 85 °C, unless otherwise specified)

Table 11.8. 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		±2		LSB
t_C	Conversion Time	F_{OSC} =5 - 10 - 20 MHz		32		µs
V_{AN}	Conversion Range		V_{SSA}		2.5	V
V_{ZI}	Zero Scale Voltage	Conversion result= 00 Hex		V_{SSA}		V
V_{FS}	Full Scale Voltage (bandgap)	Conversion result= FF Hex		2.474		V
ΔV_{FS}%	Full Scale Voltage (bandgap) precision v/s V_{DDA} variation	VDDA=5V±5%			1	%
AD_I	Analog Input Current during Conversion	f_{OSC} = 20 MHz		2		µA
AC_{IN} (2)	Analog Input Capacitance				2	pF
ASI	Analog Source Impedance				1	kΩ
ORI	Output Reference Impedance		100			Ω
ORL	Output Reference Load				0.1	mA
ORLC	Analog Reference Load Capacitance				10	pF

Source-Off and Drain-Off Leakage Currents are in the range of nA.

Notes: 1. Noise at V_{DDA}, V_{SSA} <40 mV

2. Excluding Pad Capacitance.

INSTRUCTION SET

ADD
Addition

Format: **ADD regi, regj**

Operation: **regi <- regi + regj**

Description: The contents of the Register File j-th register specified as source is added to the destination i-th register, leaving the result in the destination register. The result is 255 if overflow occurs.

Flags: Z set if result is zero, cleared otherwise.
S set if overflow, cleared otherwise.

Bytes: 2

Cycles: 7

Example: If the register 4 contains the value 45 and the register 11 contains the value 15, then the instruction

ADD 4,11 1001000 0100|1011

causes the register 4 of the Register File to be loaded with the value 60.

If the register 4 contains the value 200 and the register 11 contains the value 100, the instruction causes the register 4 to be loaded with the value 44 (result-256) and the S flag to be set

AND
Logical AND

Format: **AND regi, regj**

Operation: **regi <- regi AND regj**

Description: The instruction logically ANDs the contents of the Register File j-th register specified as source and the destination i-th register in the Register File, leaving the result in the destination register.

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

Bytes: 2

Cycles: 7

Example: If the register 4 contains the value 10011100 and the register 12 contains the value 01010101, then the instruction

AND 4,12 10010001 0100|1100

causes the register 4 of the Register File to be loaded with the value 00010100.

CON

Consequent

Format: CON cost

Operation: **Dividend Register <- Dividend Register + cost*teta**
Divisor <- Divisor + teta

Description: This intruction computes the values to add in the defuzzification 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

Membership Functions data

Format: **DATA var mbf lvd vtx rvd**

Operation:

ADM location 16*var+mbf	<- lvd
ADM location 16*var+mbf+64	<- vtx
ADM location 16*var+mbf+128	<- rvd

Description: This instruction is a pseudo instruction (it does not correspond to any operation executed by the processor) that allows to store membership functions data in the ADM (Antecedent Data Memory). The var and the mbf data identify the membership function. The lvd data is the left semibase distance of the M.F., the vtx data is the position of the vertex and rvd is the right semibase distance.

FZAND
Fuzzy AND

Format: **FZAND**

Operation: **K <- stack0 AND stack1**

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

FZOR
Fuzzy OR

Format: FZOR

Operation: K <- stack0 OR stack1

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

IRQ
Interrupt Vector

Format: **IRQ int label**

Operation: **interrupt vector <- label** (PC = Program Counter)

Description: This instruction allows to specify the interrupt int service routine start address at label location.

Flags: Z,S not affected.

Bytes: 2

Cycles: 6

Example: The instruction:

IRQ 1 IntRout1

determinates that if interrupt 1 is serviced, the program counter (PC) is loaded with the memory address value labelled with IntRout1.

Remark: The instruction IRQ is a dummy instruction used to store data in the chip memory. It is neither stored in memory nor executed. A series of IRQ instructions must be ended by a dummy end operation.

IRQM
Mask Interrupt

Format: **IRQM mask**

Operation: **interrupt mask register <- mask**

Description: The interrupts are masked with the specified mask.

Flags: Z,S not affected.

Bytes: 2

Cycles: 6

Example: The instruction:
IRQM 10 1011|1110 00001010
enables the interrupts 1 and 3 and disables all the others.

IRQP
Interrupt Priority

Format: **IRQP cost**

Operation: **interrupt priority register <- cost**

Description: The interrupts priority is set according the specified values.

Flags: Z,S not affected.

Bytes: 2

Cycles: 6

Example: The instruction:

IRQP 198 **1011|1111 11|00|01|10**

determines the interrupt 2 to have highest priority, interrupt 1 medium priority and interrupt 0 lower priority.

Remark: each couples of bits must have different values, that is interrupts must have different priority level.enables the interrupts 1 and 3 and disables all the others.

JP
Unconditional Jump

Format: **JP addr**

Operation: **PC <- addr** (PC = Program Counter)

Description: The instruction replaces the PC value with the specified value causing an unconditional jump to another location in the program memory.

Flags: Z,S not affected.

Bytes: 2

Cycles: 6

Example: The instruction:
JP 1123 1010|0100 01100011
causes the PC to be loaded with the value 1123 and the program to continue from that location.

JPNS
Jump on Non Sign Flag

Format: **JPNS addr**

Operation: **if S=0, PC <- addr** (PC = Program Counter)

Description: If the S flag is cleared then the PC value is replaced with the specified value, causing a jump to another location in the program memory.

Flags: Z,S not affected.

Bytes: 2

Cycles: 6

Example: If the S flag is cleared then the instruction:

JPNS 1123 1111|0100 01100011

causes the PC to be loaded with the value 1123 and the program to continue from that location.

JPNZ

Jump on Non Zero Flag

Format: **JPNZ addr**

Operation: **if Z=0, PC <- addr** (PC = Program Counter)

Description: If the Z flag is cleared then the PC value is replaced with the specified value, causing a jump to another location in the program memory.

Flags: Z,S not affected.

Bytes: 2

Cycles: 6

Example: If the Z flag is cleared then the instruction:

JPNZ 1123 1101|0100 01100011

causes the PC to be loaded with the value 1123 and the program to continue from that location.

JPS
Jump on Sign Flag

Format: **JPS addr**

Operation: **if S=1, PC <- addr** (PC = Program Counter)

Description: If the S flag is set then the PC value is replaced with the specified value, causing a jump to another location in the program memory.

Flags: Z,S not affected.

Bytes: 2

Cycles: 6

Example: If the S flag is set then the instruction:

JPS 1123 1110|0100 01100011

causes the PC to be loaded with the value 1123 and the program to continue from that location.

JPZ
Jump on Zero Flag

Format: **JPZ addr**

Operation: **if Z=1, PC <- addr** (PC = Program Counter)

Description: If the Z flag is set then the PC value is replaced with the specified value, causing a jump to another location in the program memory.

Flags: Z,S not affected.

Bytes: 2

Cycles: 6

Example: If the Z flag is set then the instruction:

JPZ 1123 1110|0100 01100011

causes the PC to be loaded with the value 1123 and the program to continue from that location.

LDCF

Load Constant into Configuration Register

Format: **LDCF conf, const**

Operation: **conf <- const**

Description: The immediate constant value (const) specified as source is loaded into the destination peripheral configuration register (conf).

Flags: Z,S not affected.

Bytes: 2

Cycles: 6

Example: The instruction:

LDCF 5,43 1011|0101 00101011

causes the peripheral configuration register 5 to be loaded with the value 43.

LDK
Load Stack with K register

Format: LDK

Operation: stack0 <- K

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

LDM

Load Stack with M register

Format: LDM

Operation: stack0 <- M

Description: This instruction loads in the stack the value temporarily stored in the register M with a 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: This instruction performs the fuzzyfication and loads in the stack the negated alpha value of the M.F. mbf of 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: This instruction performs the fuzzyfication and loads in the stack the alpha value of the M.F. mbf of var Variable.

LDPR

Load Register into Peripheral Register

Format: **LDPR per, reg**

Operation: **per <- re**g

Description: The contents register specified as source (reg) is loaded into the destination peripheral register (per).

Flags: Z, S not affected.

Bytes: 1

Cycles: 5 (6 if parallel port with H/S is addressed)

Example: If the register 7 of the Register File contains the value 25 then the instruction:

LDPR 2,7 01|10|0111

causes the register 2 of the Peripheral Register (i.e. parallel port) to be loaded with the value 25.

LDRC

Load constant into Register

Format: **LDRC reg, const**

Operation: **reg <- const**

Description: The immediate constant value specified as source is loaded into the destination register in the Register File.

Flags: Z,S not affected.

Bytes: 2

Cycles: 6

Example: The instruction:

LDRC 5,43 1000|0101 00101011

causes the register 5 of the Register File to be loaded with the value 43.

LDRI
Load Input register into Register file

Format: **LDRI reg, inp**

Operation: **reg <- inp**

Description: The contents of a input register specified as source (inp) is loaded into the destination register in the Register File (reg).

Flags: Z,S not affected.

Bytes: 2

Cycles: 6

Example: If the register 2 of the A/D converter contains the value 25 then the instruction:

LDRI 5,2 000|xxxxx 0101|0010 x = don't care

causes the register 5 of the Register file to be loaded with the value 25.

LDRR
Load Register into Register

Format: **LDRR regi, regj**

Operation: **regj <- regi**

Description: The contents of the Register File j-th register specified as source is loaded into the destination i-th register.

Flags: Z,S not affected.

Bytes: 2

Cycles: 6

Example: If the register 2 of the Register File contains the value 25 then the instruction:

LDRR 5,2 100101100101|0010

causes the register 5 of the Register file to be loaded with the value 25.

MDGI
Macro Disable Global Interrupt

Format: MDGI

Operation: **MGI bit <- 0**

Description: All the interrupts are disabled by this instruction. This instruction is used by FUZZYSTUDIO™ 3.0 Compiler macros to disable interrupt during macro execution.

Flags: Z,S not affected.

Bytes: 1

Cycles: 4

Example: After the instruction:

MDGI **10011010**

all the interrupts cannot be acknowledged and remain pending

MEGI

Macro Enable Global Interrupt

Format: MDGI

Operation: **MGI bit <- 1**

Description: The not masked interrupts are enabled by this instruction only if a UDGI instruction has not specified before, not followed by a UEGI instruction. This instruction is used by FUZZYSTUDIO™ 3.0 Compiler macros to disable interrupt during macro execution.

Flags: Z,S not affected.

Bytes: 1

Cycles: 4

Example: After the instruction:

MEGI **10011011**

not masked interrupts can be acknowledged if the interrupts are not globally disabled by the UDGI instruction

OUT
Output computation

Format: OUT const

Operation: Output Register const <- defuzzyfication result of the const output

Description: This instruction performs the defuzzyfication of the specified output (const can assume only the values 0 or 1) and loads in the correspondent Fuzzy Output Register the result.

RETI
Return from Interrupt

Format: **RETI**

Operation: **PC <- stack**
Z <- stack
S <- stack

Description: This instruction resumes the program execution exactly at the point it was left when an interrupt occurred. Z and S flag are set to the status they had when the interrupt service routine was started.

Flags: Z,S restored to the original setting before an interrupt occured.

Bytes: 1

Cycles: 5

Example: If the PC stack contains the value 1123 and the program is processing an interrupt service routine, then the instruction

RETI 10010101

causes the PC to be loaded with the value 1123 and the flags to be restored to the status before the interrupt occurred.

RINT
Reset Interrupt

Format: **RINT int**

Operation: **cancel pending interrupt n. int**

Description: The specified pending interrupt is cancelled if not currently in service.

Flags: Z,S not affected.

Bytes: 1

Cycles: 4

Example: The instruction:

RINT	**2**	0001|x|010	x = don't care

causes the bit 2 of the interrupt pending register to be cleared so that the interrupt 2 is not acknowledged.

Remark: The use of RINT istruction has no effect if a specified interrupt has already been aknowledged and related service routine has not been completed.

SKM

Store K register in M register

Format: SKM

Operation: M <- K

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

SRX
SCI Reception

Format: **SRX** regi

Operation: **regi <- SCDR_RX**

Description: The contents of the SCDR_RX block of the SCI receiver block, is transferred in the Register File i-th register specified as destination.

Flags: Z,S not affected.

Bytes: 2

Cycles: 5

Example: If the SCDR_RX block of the SCI receiver block contains the value 45, then the instruction

SRX 4 00101101

causes the register 4 of the Register File to be loaded with the value 45.

STOP
Stop Program Execution

Format: **STOP**

Operation: **Stop section**

Description: This instruction separates arithmetic instructions and fuzzy instructions. Also it ends a IRQ specification section.

Flags: Z,S not affected.

Bytes: 1

Cycles: 4

Example: The instruction:

STOP 10010111

if put after arithmetic instructions, it allows to start a block of fuzzy instruction and vice versa.

STX
SCI Transmission

Format: **STX** regi

Operation: **SCDR_TX <- regi**

Description: The contents of the Register File i-th register specified as source is transferred in the SCDR_TX block of the SCI transmitter block, to be transmitted.

Flags: Z,S not affected.

Bytes: 2

Cycles: 5

Example: If the register 4 contains the value 45, then the instruction

STX 4 00101101

causes the serial transmission of 45.

SUB
Subtraction

Format: **SUB regi, regj**

Operation: **regi <- regi -regj**

Description: The contents of the Register File j-th register specified as source is subtracted from the destination i-th register, leaving the result in the destination register.

Flags: Z set if result is zero, cleared otherwise.
S set if underflow, cleared otherwise.

Bytes: 2

Cycles: 7

Example: If the register 4 contains the value 45 and the register 11 contains the value 15, then the instruction

SUB 4,11 10010010 0100|1011
causes the register 4 of the Register File to be loaded with the value 30.

If the register 4 contains the value 100 and the register 11 contains the value 200, the instruction causes the register 4 to be loaded with the value 156 (result+256) and the S flag to be set.

SUBO

Subtraction with Offset

Format: **SUBO regi, regj**

Operation: **regi <- regi - regj +128**

Description: The contents of the Register File register specified as source are subtracted from the destination register in the Register File, the value 128 is added to the result that is stored in the destination register. 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 or if overflow occurs, cleared otherwise.
S set if underflow or overflow, cleared otherwise.

Bytes: 2

Cycles: 7

Example: If the register 4 contains the value 45 and the register 11 contains the value 15, then the instruction

SUBO 4,11 10010011 0100|1011

causes the register 4 of the Register File to be loaded with the value 158. The value 45 corresponds to -83, the value 11 corresponds to -113; so the operation is equivalent to perform -83 - (-113) = 30. As a matter of fact the result 158 corresponds to the value 30.

If the register 4 contains the value 50 and the register 11 contains the value 200, the instruction causes the register 4 to be loaded with the value 234 (result+256) and the S flag to be set.

If the register 4 contains the value 200 and the register 11 contains the value 50, the instruction causes the register 4 to be loaded with the value 22 (result-256) and the S and Z flags to be set.

UDGI

User Disable Global Interrupt

Format: **UDGI**

Operation: **UGI bit <- 0**

Description: All the interrupt are disabled by this instruction.

Flags: Z,S not affected.

Bytes: 1

Cycles: 4

Example: After the instruction:

UDGI **10011000**

all the interrupts cannot be acknowledged and remain pending.

UEGI
User Enable Global Interrupt

Format: **UEGI**

Operation: **UGI bit <- 1**

Description: All the interrupts are enabled by this instruction.

Flags: Z,S not affected.

Bytes: 1

Cycles: 4

Example: After the instruction:

UEGI **10011001**

not masked interrupts can be acknowledged if the interrupt are not globally disabled by the MDGI instruction.

WAITI

Wait for interrupt

Format: **WAITI**

Operation: **Wait state**

Description: This instruction causes the program to stop, without halting the peripherals, until an interrupt occurs**..**

Flags: Z,S not affected.

Bytes: 1

Cycles: 4

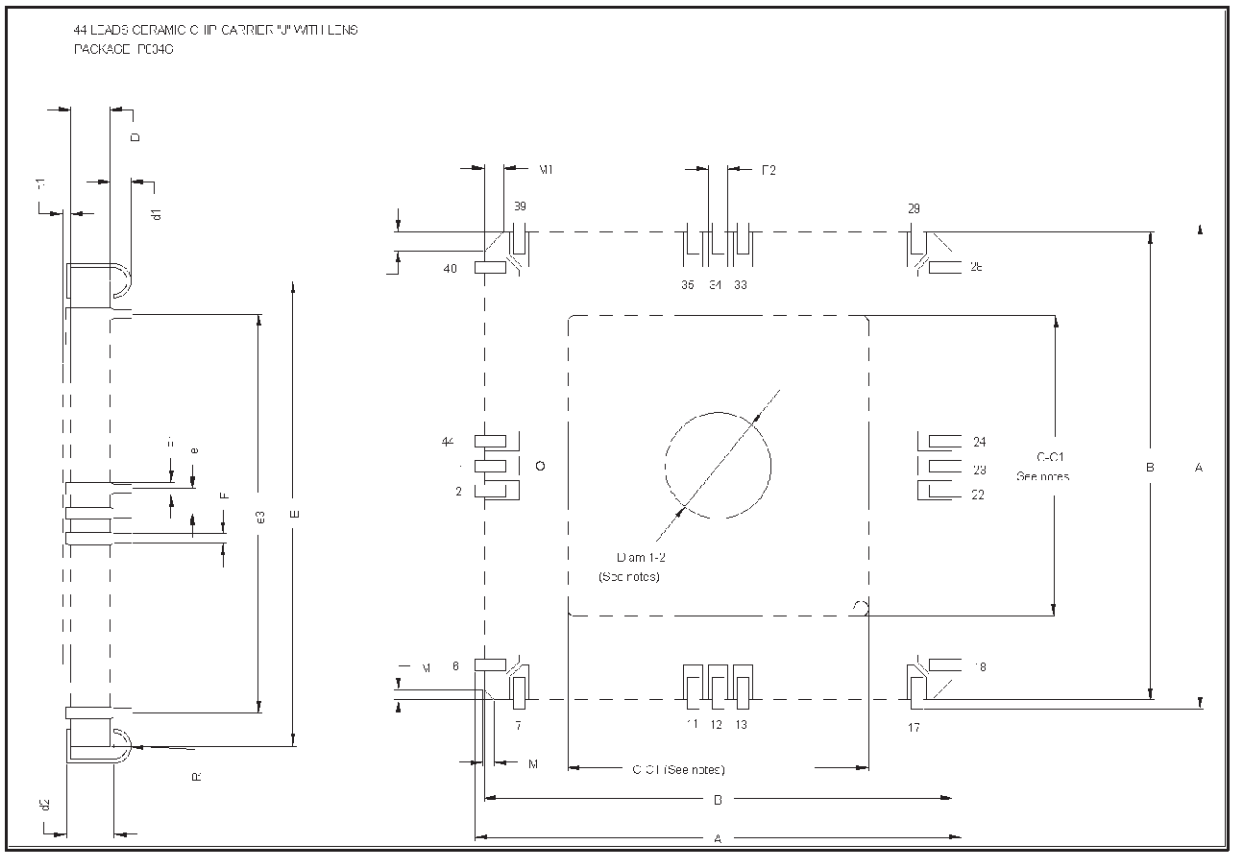
Example: The instruction:

WAITI 10010100

halts the program execution leaving the peripherals running on, until an interrupt occurs.

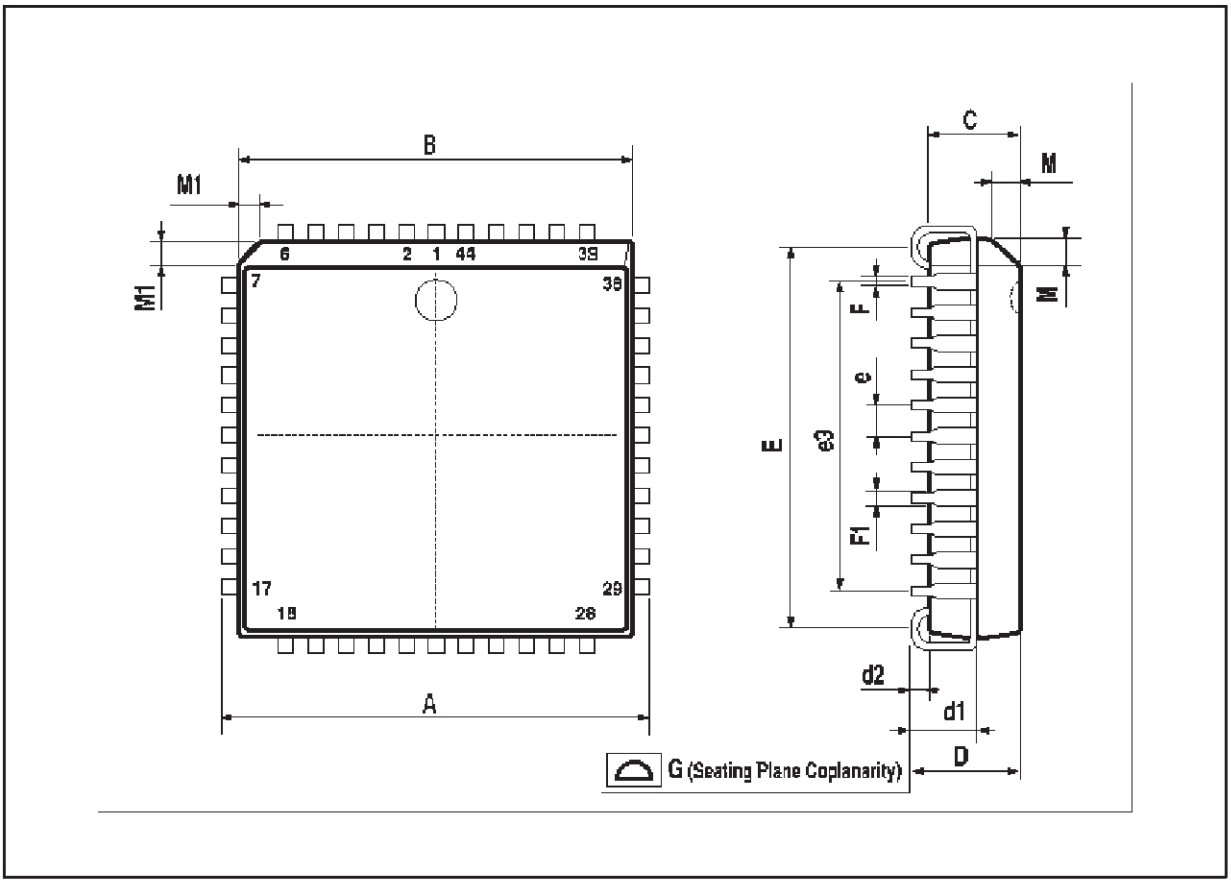
CLCC44 PACKAGE MECHANICAL DATA

DIM	mm			inch.		
	MIN	TYP	MAX	MIN	TYP	MAX
A	17.27		17.78	.680		.662
B	16.33		16.81	.643		.662
C		12.01			.475	
C1		13.03			.513	
c1		1.30			0.52	
D	1.82		2.23	0.72		.088
d1		0.889			.035	
d2		2.362			.093	
E	16.26		16.76	.640		.660
e		1.27			.050	
e3		12.50			.500	
F		0.431			.017	
F1		0.762			.030	
F2		0.965			.038	
M		0.508			.020	
M1		1.016			.040	
R		0.762			.030	



PLCC44 PACKAGE MECHANICAL DATA

DIM	mm			inch.		
	MIN	TYP	MAX	MIN	TYP	MAX
A	17.4		17.65	0.685		0.695
B	16.51		16.65	0.650		0.656
C	3.65		3.7	0.144		1.146
D	4.2		4.57	0.165		0.180
d1	2.59		2.74	0.102		0.108
d2		0.68			0.027	
E	14.99		16	0.590		0.630
e		1.27			0.050	
e3		1.27			0.500	
F		0.46			0.018	
F1		0.71			0.028	
G			0.101			0.004
M		1.16			0.046	
M1		1.14			0.045	



ORDERING INFORMATION

PART NUMBER	PACKAGE
ST52E301/C	CLCC44-W
ST52T301/P	PLCC44

Full Product Information at http://www.st.com