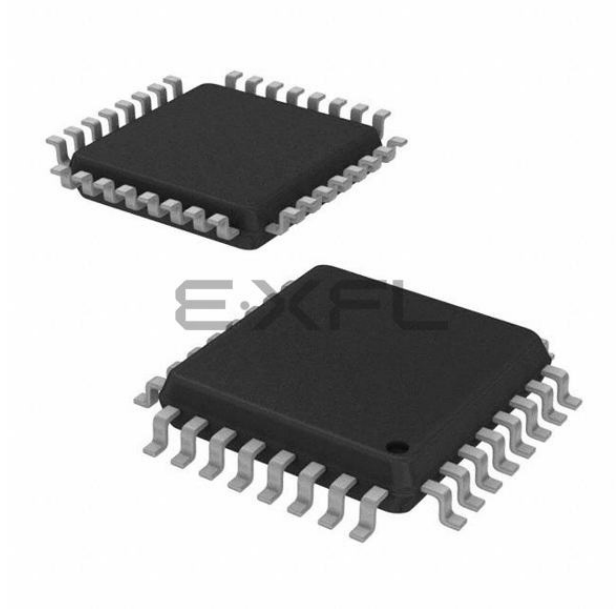


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	ST7
Core Size	8-Bit
Speed	8MHz
Connectivity	SCI, SPI
Peripherals	LVD, POR, PWM, WDT
Number of I/O	24
Program Memory Size	8KB (8K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	384 x 8
Voltage - Supply (Vcc/Vdd)	3.8V ~ 5.5V
Data Converters	A/D 12x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	32-LQFP
Supplier Device Package	-
Purchase URL	https://www.e-xfl.com/product-detail/stmicroelectronics/st72f324bk2ta

List of tables

5.3.1 Accumulator (A)

The Accumulator is an 8-bit general purpose register used to hold operands and the results of the arithmetic and logic calculations and to manipulate data.

5.3.2 Index registers (X and Y)

These 8-bit registers are used to create effective addresses or as temporary storage areas for data manipulation. (The Cross-Assembler generates a precede instruction (PRE) to indicate that the following instruction refers to the Y register.)

The Y register is not affected by the interrupt automatic procedures.

5.3.3 Program counter (PC)

The program counter is a 16-bit register containing the address of the next instruction to be executed by the CPU. It is made of two 8-bit registers PCL (Program Counter Low which is the LSB) and PCH (Program Counter High which is the MSB).

5.3.4 Condition Code register (CC)

The 8-bit Condition Code register contains the interrupt masks and four flags representative of the result of the instruction just executed. This register can also be handled by the PUSH and POP instructions. These bits can be individually tested and/or controlled by specific instructions.

CC Reset value: 111x1xxx

7	6	5	4	3	2	1	0
1	1	I1	H	I0	N	Z	C
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 6. Arithmetic management bits

Bit	Name	Function
4	H	Half carry This bit is set by hardware when a carry occurs between bits 3 and 4 of the ALU during an ADD or ADC instructions. It is reset by hardware during the same instructions. 0: No half carry has occurred. 1: A half carry has occurred. This bit is tested using the JRH or JRNH instruction. The H bit is useful in BCD arithmetic subroutines.
2	N	Negative This bit is set and cleared by hardware. It is representative of the result sign of the last arithmetic, logical or data manipulation. It is a copy of the result 7th bit. 0: The result of the last operation is positive or null. 1: The result of the last operation is negative (that is, the most significant bit is a logic 1. This bit is accessed by the JRMI and JRPL instructions.

5.3.5 Stack Pointer register (SP)

SP — Reset value: 01 FFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The Stack Pointer is a 16-bit register which is always pointing to the next free location in the stack. It is then decremented after data has been pushed onto the stack and incremented before data is popped from the stack (see *Figure 8*).

Since the stack is 256 bytes deep, the 8 most significant bits are forced by hardware. Following an MCU reset, or after a Reset Stack Pointer instruction (RSP), the Stack Pointer contains its reset value (the SP7 to SP0 bits are set) which is the stack higher address.

The least significant byte of the Stack Pointer (called S) can be directly accessed by an LD instruction.

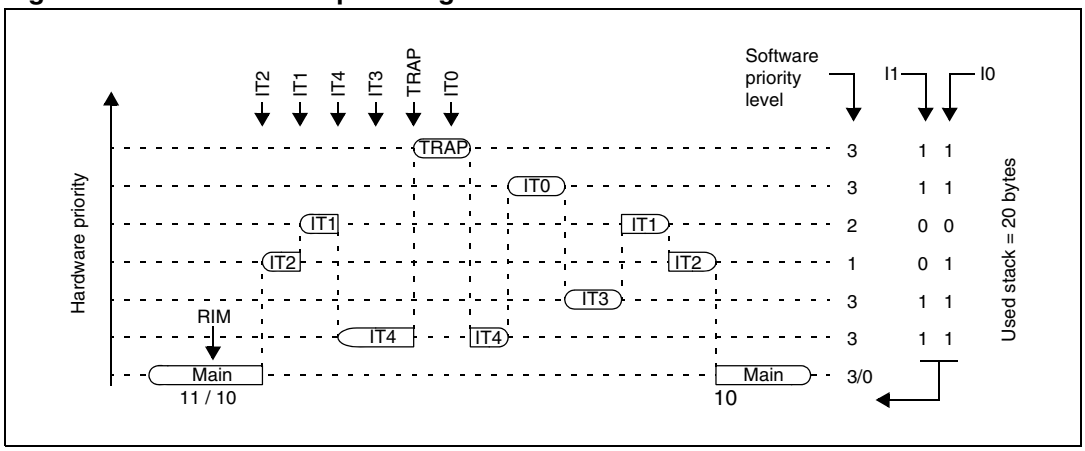
Note: When the lower limit is exceeded, the Stack Pointer wraps around to the stack upper limit, without indicating the stack overflow. The previously stored information is then overwritten and therefore lost. The stack also wraps in case of an underflow.

The stack is used to save the return address during a subroutine call and the CPU context during an interrupt. The user may also directly manipulate the stack by means of the PUSH and POP instructions. In the case of an interrupt, the PCL is stored at the first location pointed to by the SP. Then the other registers are stored in the next locations as shown in *Figure 8*.

- When an interrupt is received, the SP is decremented and the context is pushed on the stack.
- On return from interrupt, the SP is incremented and the context is popped from the stack.

A subroutine call occupies two locations and an interrupt five locations in the stack area.

Figure 19. Nested interrupt management



7.5 Interrupt registers

7.5.1 CPU CC register interrupt bits

CPU CC Reset value: 111x 1010(xAh)

7	6	5	4	3	2	1	0
1	1	**I1**	H	**I0**	N	Z	C
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 15. CPU CC register interrupt bits description

Bit	Name	Function
5	I1	Software Interrupt Priority 1
3	I0	Software Interrupt Priority 0

Table 16. Interrupt software priority levels

Interrupt software priority	Level	I1	I0
Level 0 (main)	Low	1	0
Level 1	↓	0	1
Level 2	↓	0	0
Level 3 (= interrupt disable)(1)	High	1	1

1. TRAP and RESET events can interrupt a level 3 program.

These two bits indicate the current interrupt software priority (see *Table 16*) and are set/cleared by hardware when entering in interrupt. The loaded value is given by the corresponding bits in the interrupt software priority registers (ISPRx).

They can be also set/cleared by software with the RIM, SIM, HALT, WFI, IRET and PUSH/POP instructions (see *Table 18: Dedicated interrupt instruction set*).

7.5.2 Interrupt software priority registers (ISPRx)

ISPRx Reset value: 1111 1111 (FFh)

	7	6	5	4	3	2	1	0
ISPR0	I1_3	I0_3	I1_2	I0_2	I1_1	I0_1	I1_0	I0_0
ISPR1	I1_7	I0_7	I1_6	I0_6	I1_5	I0_5	I1_4	I0_4
ISPR2	I1_11	I0_11	I1_10	I0_10	I1_9	I0_9	I1_8	I0_8
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ISPR3	1	1	1	1	I1_13	I0_13	I1_12	I0_12
	RO	RO	RO	RO	R/W	R/W	R/W	R/W

These four registers contain the interrupt software priority of each interrupt vector.

- Each interrupt vector (except reset and TRAP) has corresponding bits in these registers where its own software priority is stored. This correspondence is shown in the following *Table 17*.

Table 17. ISPRx interrupt vector correspondence

Vector address	ISPRx bits
FFFBh-FFFAh	I1_0 and I0_0 bits
FFF9h-FFF8h	I1_1 and I0_1 bits
...	...
FFE1h-FFE0h	I1_13 and I0_13 bits

- Each I1_x and I0_x bit value in the ISPRx registers has the same meaning as the I1 and I0 bits in the CC register.
- Level 0 cannot be written (I1_x = 1, I0_x = 0). In this case, the previously stored value is kept (for example, previous value = CFh, write = 64h, result = 44h).

The reset, and TRAP vectors have no software priorities. When one is serviced, the I1 and I0 bits of the CC register are both set.

Caution: If the I1_x and I0_x bits are modified while the interrupt x is executed the following behavior has to be considered: If the interrupt x is still pending (new interrupt or flag not cleared) and the new software priority is higher than the previous one, the interrupt x is re-entered. Otherwise, the software priority stays unchanged up to the next interrupt request (after the IRET of the interrupt x).

Table 18. Dedicated interrupt instruction set(1)

Instruction	New description	Function/example	I1	H	I0	N	Z	C
HALT	Entering HALT mode		1		0			

Table 18. Dedicated interrupt instruction set(1) (continued)

Instruction	New description	Function/example	I1	H	I0	N	Z	C
IRET	Interrupt routine return	POP CC, A, X, PC	I1	H	I0	N	Z	C
JRM	Jump if I1:0=11 (level 3)	I1:0=11 ?						
JRNM	Jump if I1:0<>11	I1:0<>11 ?						
POP CC	POP CC from the Stack	Mem => CC	I1	H	I0	N	Z	C
RIM	Enable interrupt (level 0 set)	Load 10 in I1:0 of CC	1		0			
SIM	Disable interrupt (level 3 set)	Load 11 in I1:0 of CC	1		1			
TRAP	Software TRAP	Software NMI	1		1			
WFI	WAIT for interrupt		1		0			

1. During the execution of an interrupt routine, the HALT, POP CC, RIM, SIM and WFI instructions change the current software priority up to the next IRET instruction or one of the previously mentioned instructions.

Halt mode recommendations

- Make sure that an external event is available to wake up the microcontroller from Halt mode.
- When using an external interrupt to wake up the microcontroller, reinitialize the corresponding I/O as “Input Pull-up with Interrupt” before executing the HALT instruction. The main reason for this is that the I/O may be wrongly configured due to external interference or by an unforeseen logical condition.
- For the same reason, reinitialize the sensitivity level of each external interrupt as a precautionary measure.
- The opcode for the HALT instruction is 0x8E. To avoid an unexpected HALT instruction due to a program counter failure, it is advised to clear all occurrences of the data value 0x8E from memory. For example, avoid defining a constant in ROM with the value 0x8E.
- As the HALT instruction clears the interrupt mask in the CC register to allow interrupts, the user may choose to clear all pending interrupt bits before executing the HALT instruction. This avoids entering other peripheral interrupt routines after executing the external interrupt routine corresponding to the wake-up event (reset or external interrupt).

External interrupt function

When an I/O is configured as 'Input with Interrupt', an event on this I/O can generate an external interrupt request to the CPU.

Each pin can independently generate an interrupt request. The interrupt sensitivity is independently programmable using the sensitivity bits in the EICR register.

Each external interrupt vector is linked to a dedicated group of I/O port pins (see pinout description and interrupt section). If several input pins are selected simultaneously as interrupt sources, these are first detected according to the sensitivity bits in the EICR register and then logically ORed.

The external interrupts are hardware interrupts, which means that the request latch (not accessible directly by the application) is automatically cleared when the corresponding interrupt vector is fetched. To clear an unwanted pending interrupt by software, the sensitivity bits in the EICR register must be modified.

9.2.2 Output modes

The output configuration is selected by setting the corresponding DDR register bit. In this case, writing the DR register applies this digital value to the I/O pin through the latch. Then reading the DR register returns the previously stored value.

Two different output modes can be selected by software through the OR register: Output push-pull and open-drain.

Table 27. DR register value and output pin status

DR	Push-pull	Open-drain
0	V_{SS}	V_{SS}
1	V_{DD}	Floating

9.2.3 Alternate functions

When an on-chip peripheral is configured to use a pin, the alternate function is automatically selected. This alternate function takes priority over the standard I/O programming.

When the signal is coming from an on-chip peripheral, the I/O pin is automatically configured in output mode (push-pull or open drain according to the peripheral).

When the signal is going to an on-chip peripheral, the I/O pin must be configured in input mode. In this case, the pin state is also digitally readable by addressing the DR register.

Note: Input pull-up configuration can cause unexpected value at the input of the alternate peripheral input. When an on-chip peripheral use a pin as input and output, this pin has to be configured in input floating mode.

Figure 28. I/O port general block diagram

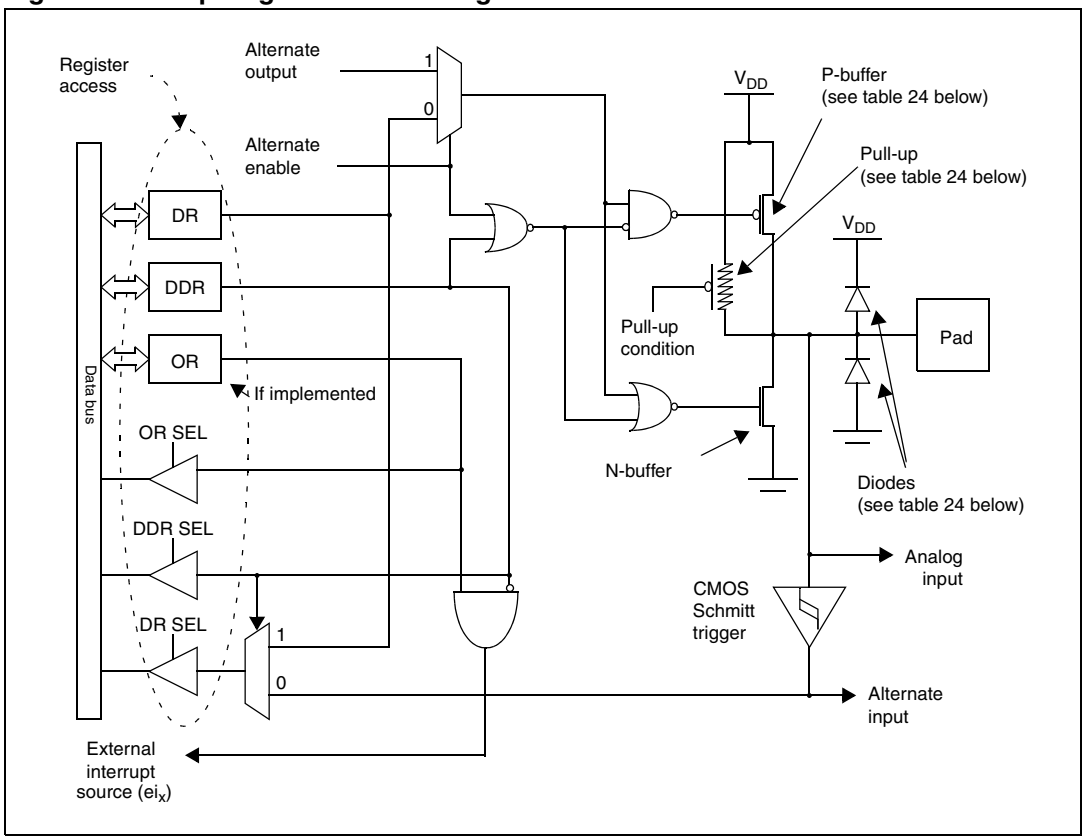
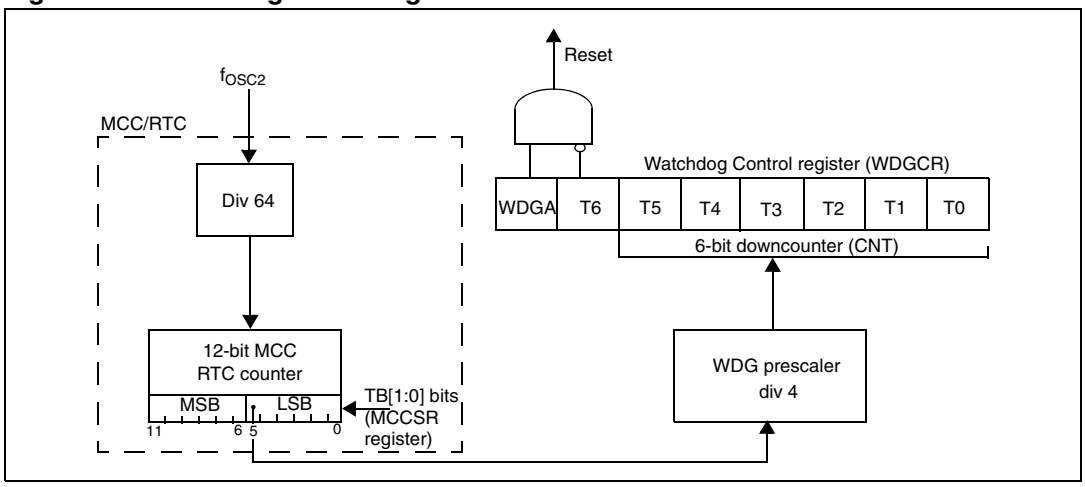


Table 28. I/O port mode options

Configuration mode		Pull-up	P-buffer	Diodes	
				to V_{DD} (1)	to V_{SS} (2)
Input	Floating with/without Interrupt	Off(3)	Off	On	On
	Pull-up with/without Interrupt	On(4)			
Output	Push-pull	Off	On		
	Open drain (logic level)		Off		
	True open drain	NI	NI	NI(5)	

1. The diode to V_{DD} is not implemented in the true open drain pads.
2. A local protection between the pad and V_{SS} is implemented to protect the device against positive stress.
3. Off = implemented not activated.
4. On = implemented and activated.
5. NI = not implemented

Figure 30. Watchdog block diagram

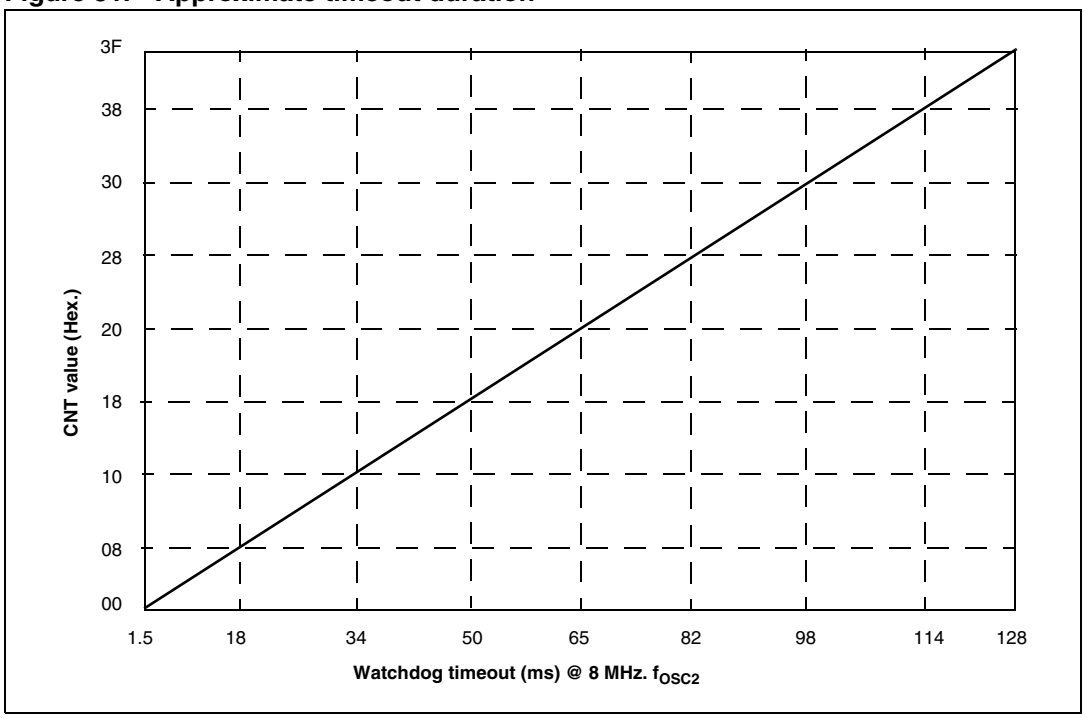


10.1.4 How to program the Watchdog timeout

Figure 31 shows the linear relationship between the 6-bit value to be loaded in the Watchdog Counter (CNT) and the resulting timeout duration in milliseconds. This can be used for a quick calculation without taking the timing variations into account. If more precision is needed, use the formulae in *Figure 32*.

Caution: When writing to the WDGCR register, always write 1 in the T6 bit to avoid generating an immediate reset.

Figure 31. Approximate timeout duration



If the timer clock is an external clock, the formula is:

$$\Delta \text{ OC}i\text{R} = \Delta t * f_{EXT}$$

Where:

Δt = Output compare period (in seconds)
f_{EXT} = External timer clock frequency (in hertz)

Clearing the output compare interrupt request (that is, clearing the OCF*i* bit) is done by:

1. Reading the SR register while the OCF*i* bit is set.
2. An access (read or write) to the OC*i*LR register.

The following procedure is recommended to prevent the OCF*i* bit from being set between the time it is read and the write to the OC*i*R register:

- Write to the OC*i*HR register (further compares are inhibited).
- Read the SR register (first step of the clearance of the OCF*i* bit, which may be already set).
- Write to the OC*i*LR register (enables the output compare function and clears the OCF*i* bit).

Note: 1 After a processor write cycle to the OCiHR register, the output compare function is inhibited until the OCiLR register is also written.

2 If the OCiE bit is not set, the OCMPi pin is a general I/O port and the OLVLi bit will not appear when a match is found but an interrupt could be generated if the OCIE bit is set.

3 In both internal and external clock modes, OCFi and OCMPi are set while the counter value equals the OCiR register value (see Figure 42 on page 83 for an example with $f_{CPU}/2$ and Figure 43 on page 83 for an example with $f_{CPU}/4$). This behavior is the same in OPM or PWM mode.

4 The output compare functions can be used both for generating external events on the OCMPi pins even if the input capture mode is also used.

5 The value in the 16-bit OCiR register and the OLVi bit should be changed after each successful comparison in order to control an output waveform or establish a new elapsed timeout.

Forced output compare capability

When the FOLV*i* bit is set by software, the OLVL*i* bit is copied to the OCMP*i* pin. The OLV*i* bit has to be toggled in order to toggle the OCMP*i* pin when it is enabled (OC*i*E bit = 1). The OCF*i* bit is then not set by hardware, and thus no interrupt request is generated.

The FOLVL*i* bits have no effect in both one pulse mode and PWM mode.

Output Compare 2 Low Register (OC2LR)

This is an 8-bit register that contains the low part of the value to be compared to the CLR register.

OC2LR Reset value: 0000 0000 (00h)

7	6	5	4	3	2	1	0
MSB							LSB
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Counter High Register (CHR)

This is an 8-bit register that contains the high part of the counter value.

CHR Reset value: 1111 1111 (FFh)

7	6	5	4	3	2	1	0
MSB							LSB
RO	RO	RO	RO	RO	RO	RO	RO

Counter Low Register (CLR)

This is an 8-bit register that contains the low part of the counter value. A write to this register resets the counter. An access to this register after accessing the CSR register clears the TOF bit.

CLR Reset value: 1111 1100 (FCh)

7	6	5	4	3	2	1	0
MSB							LSB
RO	RO	RO	RO	RO	RO	RO	RO

Alternate Counter High Register (ACHR)

This is an 8-bit register that contains the high part of the counter value.

ACHR Reset value: 1111 1111 (FFh)

7	6	5	4	3	2	1	0
MSB							LSB
RO	RO	RO	RO	RO	RO	RO	RO

Changing the conversion channel

The application can change channels during conversion. When software modifies the CH[3:0] bits in the ADCCSR register, the current conversion is stopped, the EOC bit is cleared, and the A/D converter starts converting the newly selected channel.

10.6.4 Low power modes

Note: The A/D converter may be disabled by resetting the ADON bit. This feature allows reduced power consumption when no conversion is needed.

Table 70. Effect of low power modes on ADC

Mode	Description
Wait	No effect on A/D converter
Halt	A/D converter disabled. After wake-up from Halt mode, the A/D converter requires a stabilization time t_{STAB} (see *Section 12: Electrical characteristics*) before accurate conversions can be performed.

10.6.5 Interrupts

None.

10.6.6 ADC registers

ADC Control/Status Register (ADCCSR)

ADCCSR Reset value: 0000 0000 (00h)

7	6	5	4	3	2	1	0
EOC	SPEED	ADON	Reserved	CH[3:0]			
RO	R/W	RW	-	RW			

Table 71. ADCCSR register description

Bit	Name	Function
7	EOC	End of Conversion This bit is set by hardware. It is cleared by hardware when software reads the ADCDRH register or writes to any bit of the ADCCSR register. 0: Conversion is not complete 1: Conversion complete
6	SPEED	ADC clock selection This bit is set and cleared by software. 0: $f_{ADC} = f_{CPU}/4$ 1: $f_{ADC} = f_{CPU}/2$

11.1.5 Indirect (short, long)

The required data byte to do the operation is found by its memory address, located in memory (pointer).

The pointer address follows the opcode. The indirect addressing mode consists of two submodes:

Indirect (short)

The pointer address is a byte, the pointer size is a byte, thus allowing 00 - FF addressing space, and requires 1 byte after the opcode.

Indirect (long)

The pointer address is a byte, the pointer size is a word, thus allowing 64 Kbyte addressing space, and requires 1 byte after the opcode.

11.1.6 Indirect indexed (short, long)

This is a combination of indirect and short indexed addressing modes. The operand is referenced by its memory address, which is defined by the unsigned addition of an index register value (X or Y) with a pointer value located in memory. The pointer address follows the opcode.

The indirect indexed addressing mode consists of two submodes:

Indirect indexed (short)

The pointer address is a byte, the pointer size is a byte, thus allowing 00 - 1FE addressing space, and requires 1 byte after the opcode.

Indirect indexed (long)

The pointer address is a byte, the pointer size is a word, thus allowing 64 Kbyte addressing space, and requires 1 byte after the opcode.

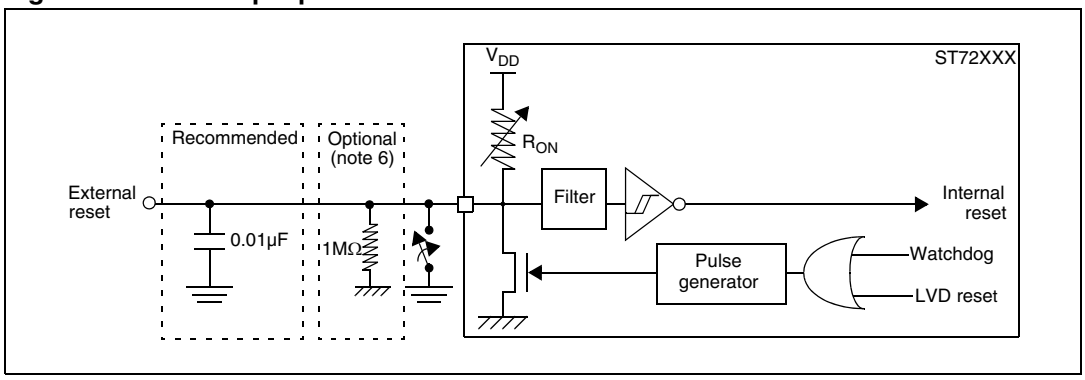
Table 79. Instructions supporting direct, indexed, indirect and indirect indexed addressing modes

Instructions		Function
Long and short	LD	Load
	CP	Compare
	AND, OR, XOR	Logical operations
	ADC, ADD, SUB, SBC	Arithmetic Additions/Subtractions operations
	BCP	Bit Compare

Table 81. Instruction groups (continued)

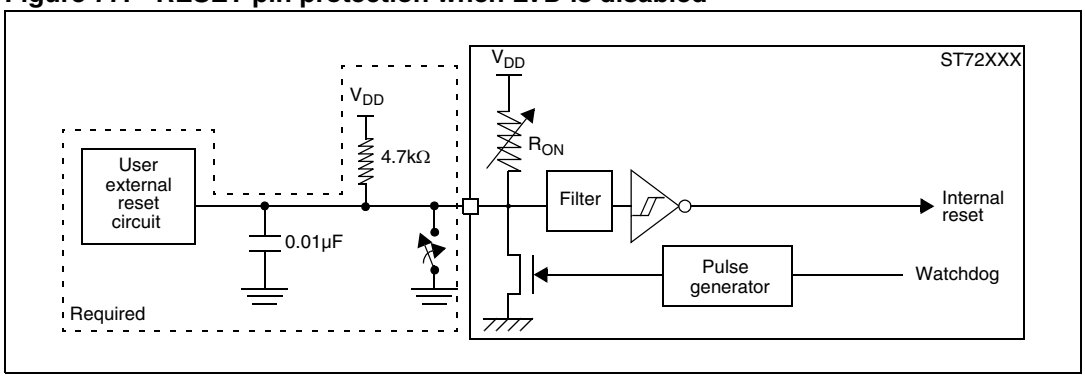
Group	Instructions							
Compare and Tests	CP	TNZ	BCP					
Logical operations	AND	OR	XOR	CPL	NEG			
Bit operation	BSET	BRES						
Conditional Bit Test and Branch	BTJT	BTJF						
Arithmetic operations	ADC	ADD	SUB	SBC	MUL			
Shift and Rotates	SLL	SRL	SRA	RLC	RRC	SWAP	SLA	
Unconditional Jump or Call	JRA	JRT	JRF	JP	CALL	CALLR	NOP	RET
Conditional Branch	JRxx							
Interruption management	TRAP	WFI	HALT	IRET				
Condition Code Flag modification	SIM	RIM	SCF	RCF				

Figure 76. $\overline{\text{RESET}}$ pin protection when LVD is enabled(1)(2)(3)(4)(5)(6)



1. The reset network protects the device against parasitic resets.
2. The output of the external reset circuit must have an open-drain output to drive the ST7 reset pad. Otherwise the device can be damaged when the ST7 generates an internal reset (LVD or watchdog).
3. Whatever the reset source is (internal or external), the user must ensure that the level on the $\overline{\text{RESET}}$ pin can go below the V_{IL} max. level specified in *Section 12.10.1*. Otherwise the reset will not be taken into account internally.
4. Because the reset circuit is designed to allow the internal RESET to be output in the $\overline{\text{RESET}}$ pin, the user must ensure that the current sunk on the $\overline{\text{RESET}}$ pin (by an external pull-up for example) is less than the absolute maximum value specified for $I_{INJ(RESET)}$ in *Section 12.2.2 on page 147*.
5. When the LVD is enabled, it is mandatory not to connect a pull-up resistor. A 10nF pull-down capacitor is recommended to filter noise on the reset line.
6. In case a capacitive power supply is used, it is recommended to connect a 1M ohm pull-down resistor to the $\overline{\text{RESET}}$ pin to discharge any residual voltage induced by this capacitive power supply (this will add 5µA to the power consumption of the MCU).
7. Tips when using the LVD:
   A. Check that all recommendations related to reset circuit have been applied (see notes above)
   B. Check that the power supply is properly decoupled (100nF + 10µF close to the MCU). Refer to AN1709. If this cannot be done, it is recommended to put a 100nF + 1M ohm pull-down on the $\overline{\text{RESET}}$ pin.
   C. The capacitors connected on the RESET pin and also the power supply are key to avoiding any start-up marginality. In most cases, steps 1 and 2 above are sufficient for a robust solution. Otherwise: Replace 10nF pull-down on the $\overline{\text{RESET}}$ pin with a 5µF to 20µF capacitor.

Figure 77. $\overline{\text{RESET}}$ pin protection when LVD is disabled(1)(2)(3)(4)



1. The reset network protects the device against parasitic resets.
2. The output of the external reset circuit must have an open-drain output to drive the ST7 reset pad. Otherwise the device can be damaged when the ST7 generates an internal reset (LVD or watchdog).
3. Whatever the reset source is (internal or external), the user must ensure that the level on the $\overline{\text{RESET}}$ pin can go below the V_{IL} max. level specified in *Section 12.10.1* Otherwise the reset will not be taken into account internally.
4. Because the reset circuit is designed to allow the internal RESET to be output in the $\overline{\text{RESET}}$ pin, the user must ensure that the current sunk on the $\overline{\text{RESET}}$ pin (by an external pull-up for example) is less than the absolute maximum value specified for $I_{INJ(RESET)}$ in *Section 12.2.2*.

15 Known limitations

15.1 All Flash and ROM devices

15.1.1 Safe connection of OSC1/OSC2 pins

The OSC1 and/or OSC2 pins must not be left unconnected, otherwise the ST7 main oscillator may start and, in this configuration, could generate an f_{OSC} clock frequency in excess of the allowed maximum (> 16 MHz), putting the ST7 in an unsafe/undefined state. Refer to *Section 6.3 on page 32*.

15.1.2 External interrupt missed

To avoid any risk of generating a parasitic interrupt, the edge detector is automatically disabled for one clock cycle during an access to either DDR and OR. Any input signal edge during this period will not be detected and will not generate an interrupt.

This case can typically occur if the application refreshes the port configuration registers at intervals during runtime.

Workaround

The workaround is based on software checking the level on the interrupt pin before and after writing to the PxOR or PxDDR registers. If there is a level change (depending on the sensitivity programmed for this pin) the interrupt routine is invoked using the call instruction with three extra PUSH instructions before executing the interrupt routine (this is to make the call compatible with the IRET instruction at the end of the interrupt service routine).

But detection of the level change does not make sure that edge occurs during the critical one cycle duration and the interrupt has been missed. This may lead to occurrence of same interrupt twice (one hardware and another with software call).

To avoid this, a semaphore is set to ‘1’ before checking the level change. The semaphore is changed to level '0' inside the interrupt routine. When a level change is detected, the semaphore status is checked and if it is ‘1’ this means that the last interrupt has been missed. In this case, the interrupt routine is invoked with the call instruction.

There is another possible case that is, if writing to PxOR or PxDDR is done with global interrupts disabled (interrupt mask bit set). In this case, the semaphore is changed to ‘1’ when the level change is detected. Detecting a missed interrupt is done after the global interrupts are enabled (interrupt mask bit reset) and by checking the status of the semaphore. If it is ‘1’ this means that the last interrupt was missed and the interrupt routine is invoked with the call instruction.

To implement the workaround, the following software sequence is to be followed for writing into the PxOR/PxDDR registers. The example is for Port PF1 with falling edge interrupt sensitivity. The software sequence is given for both cases (global interrupt disabled/enabled).

Table 122. Port A and F configuration

PLL	PA3	PF4	PF1	PF0	Clock disturbance
Off	0	1	0	Toggling	Maximum 2 clock cycles lost at each rising or falling edge of PF0
On	0	1	0	1	Maximum 1 clock cycle lost out of every 16

As a consequence, for cycle-accurate operations, these configurations are prohibited in either input or output mode.

Workaround

To avoid this from occurring, it is recommended to connect one of these pins to GND (PF4 or PF0) or V_{DD} (PA3 or PF1).

16 Revision history

Table 123. Document revision history

Date	Revision	Changes
23-May-2007	1	Initial release
23-Jul-2007	2	Replaced ST72324B-Auto with ST72324Bxx-Auto in document title on cover page. *1 analog peripheral (low current coupling) on page 1*: Replaced '12 robust input ports' with '12 input ports' *Table 1: Device summary on page 1*: Corrected order of listed packages Added *Section 1.2: Differences between ST72324B-Auto and ST72324B datasheets on page 16* *Figure 2: 44-pin LQFP package pinout on page 17*: Displayed port numbers for pins 18 and 20 (port numbers were hidden due to formatting error) *Table 2: Device pin description on page 18*: - replaced V_{DDA} with V_{REF} in *Note 1* - modified *Note 2* *Section 5.3.4: Condition Code register (CC) on page 29*: Replaced IxSPR with ISPRx *Section 9.5.1: I/O port implementation on page 65*: Removed following tables: - Standard ports PA5:4, PC7:0, PD5:0, PE1:0, PF7:6, 4 - Interrupt ports PB4, PB2:0, PF2:0 (with pull-up) - Interrupt ports PA3, PB3 (without pull-up) - *True open drain ports PA7:6* (configurations in these four tables already exist in *Table 32: Port configuration*) *Section 12.6.3: Crystal and ceramic resonator oscillators*: Replaced two tables *Crystal and ceramic resonator oscillators (8/16K Flash and ROM devices)* and *Crystal and ceramic resonator oscillators (32 Kbyte Flash and ROM devices)* with single *Table 95: Crystal and ceramic resonator oscillators on page 156* *Table 96: OSCRANGE selection for typical resonators on page 157*: Deleted footnote detailing SMD and LEAD which was linked to 'Reference' column header *Table 102: EMI emissions on page 161*: - added LQFP32 package to all listed devices - changed values for 32 Kbyte ROM devices *Table 105: General characteristics on page 163*: - modified *Note 5* - modified *Note 6* *Figure 76: RESET pin protection when LVD is enabled(1)(2)(3)(4)(5)(6) on page 168*: Replaced 'MW' with 'M ohm' in footnotes to correct formatting error *Table 111: 10-bit ADC characteristics on page 172*: Modified input current leakage parameter and added *Note 2* *Table 112: ADC accuracy on page 175*: - added conditions to total unadjusted error, to offset error and to gain error - modified *Note 2*