**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 | I²C, SCI, SPI |
| Peripherals | LVD, POR, PWM, WDT |
| Number of I/O | 48 |
| Program Memory Size | 32KB (32K x 8) |
| Program Memory Type | FLASH |
| EEPROM Size | - |
| RAM Size | 1K x 8 |
| Voltage - Supply (Vcc/Vdd) | 3.8V ~ 5.5V |
| Data Converters | A/D 16x10b |
| Oscillator Type | Internal |
| Operating Temperature | -40°C ~ 85°C (TA) |
| Mounting Type | Surface Mount |
| Package / Case | 64-LQFP |
| Supplier Device Package | - |
| Purchase URL | https://www.e-xfl.com/product-detail/stmicroelectronics/st72f321ar6t6 |

# Table of Contents

**FLASH PROGRAM MEMORY** (Cont'd)

### 4.5 ICP (In-Circuit Programming)

To perform ICP the microcontroller must be switched to ICC (In-Circuit Communication) mode by an external controller or programming tool.

Depending on the ICP code downloaded in RAM, Flash memory programming can be fully customized (number of bytes to program, program locations, or selection serial communication interface for downloading).

When using an STMicroelectronics or third-party programming tool that supports ICP and the specific microcontroller device, the user needs only to implement the ICP hardware interface on the application board (see Figure 6). For more details on the pin locations, refer to the device pinout description.

### 4.6 IAP (In-Application Programming)

This mode uses a BootLoader program previously stored in Sector 0 by the user (in ICP mode or by plugging the device in a programming tool).

This mode is fully controlled by user software. This allows it to be adapted to the user application, (user-defined strategy for entering programming mode, choice of communications protocol used to fetch the data to be stored, etc.). For example, it is possible to download code from the SPI, SCI, USB or CAN interface and program it in the Flash. IAP mode can be used to program any of the Flash sectors except Sector 0, which is write/erase protected to allow recovery in case errors occur during the programming operation.

### 4.7 Related Documentation

For details on Flash programming and ICC protocol, refer to the ST7 Flash Programming Reference Manual and to the ST7 ICC Protocol Reference Manual.

#### 4.7.1 Register Description

**FLASH CONTROL/STATUS REGISTER (FCSR)**

Read/Write

Reset Value: 0000 0000 (00h)

| 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

This register is reserved for use by Programming Tool software. It controls the Flash programming and erasing operations.

**Figure 7. Flash Control/Status Register Address and Reset Value**

| Address (Hex.) | Register Label | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0029h | **FCSR** Reset Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# 5 CENTRAL PROCESSING UNIT

## 5.1 INTRODUCTION

This CPU has a full 8-bit architecture and contains six internal registers allowing efficient 8-bit data manipulation.

## 5.2 MAIN FEATURES

- Enable executing 63 basic instructions
- Fast 8-bit by 8-bit multiply
- 17 main addressing modes (with indirect addressing mode)
- Two 8-bit index registers
- 16-bit stack pointer
- Low power HALT and WAIT modes
- Priority maskable hardware interrupts
- Non-maskable software/hardware interrupts

## 5.3 CPU REGISTERS

The six CPU registers shown in Figure 1 are not present in the memory mapping and are accessed by specific instructions.

**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.

**Index Registers (X and Y)**

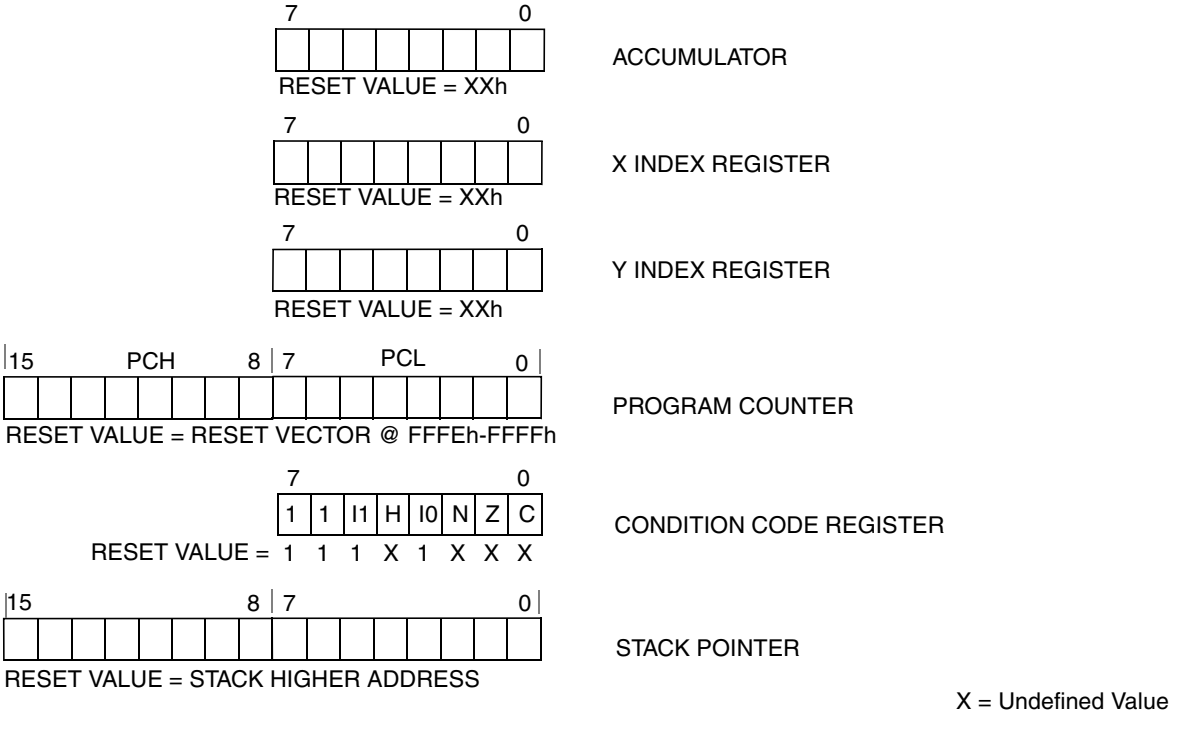
These 8-bit registers are used to create effective addresses or as temporary storage areas for data manipulation. (The Cross-Assembler generates a precede instruction (PRE) to indicate that the following instruction refers to the Y register.)

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

**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).

**Figure 8. CPU Registers**



**INTERRUPTS** (Cont'd)

**Table 7. Dedicated Interrupt Instruction Set**

| Instruction | New Description | Function/Example | I1 | H | I0 | N | Z | C |
|---|---|---|---|---|---|---|---|---|
| HALT | Entering Halt mode | | 1 | | 0 | | | |
| 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 | | | |

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

### 7.7 EXTERNAL INTERRUPT CONTROL REGISTER (EICR)

Read/Write

Reset Value: 0000 0000 (00h)

| 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|
| IS11 | IS10 | IPB | IS21 | IS20 | IPA | TLIS | TLIE |

Bit 7:6 = **IS1[1:0]** *ei2 and ei3 sensitivity*
The interrupt sensitivity, defined using the IS1[1:0] bits, is applied to the following external interrupts:
- ei2 (port B3..0)

| IS11 | IS10 | External Interrupt Sensitivity | |
|---|---|---|---|
| | | IPB bit =0 | IPB bit =1 |
| 0 | 0 | Falling edge & low level | Rising edge & high level |
| 0 | 1 | Rising edge only | Falling edge only |
| 1 | 0 | Falling edge only | Rising edge only |
| 1 | 1 | Rising and falling edge | |

- ei3 (port B7..4)

| IS11 | IS10 | External Interrupt Sensitivity |
|---|---|---|
| 0 | 0 | Falling edge & low level |
| 0 | 1 | Rising edge only |
| 1 | 0 | Falling edge only |
| 1 | 1 | Rising and falling edge |

These 2 bits can be written only when I1 and I0 of the CC register are both set to 1 (level 3).

Bit 5 = **IPB** *Interrupt polarity for port B*
This bit is used to invert the sensitivity of the port B [3:0] external interrupts. It can be set and cleared by software only when I1 and I0 of the CC register are both set to 1 (level 3).
0: No sensitivity inversion
1: Sensitivity inversion

Bit 4:3 = **IS2[1:0]** *ei0 and ei1 sensitivity*
The interrupt sensitivity, defined using the IS2[1:0] bits, is applied to the following external interrupts:

- ei0 (port A3..0)

| IS21 | IS20 | External Interrupt Sensitivity | |
|---|---|---|---|
| | | IPA bit =0 | IPA bit =1 |
| 0 | 0 | Falling edge & low level | Rising edge & high level |
| 0 | 1 | Rising edge only | Falling edge only |
| 1 | 0 | Falling edge only | Rising edge only |
| 1 | 1 | Rising and falling edge | |

- ei1 (port F2..0)

| IS21 | IS20 | External Interrupt Sensitivity |
|---|---|---|
| 0 | 0 | Falling edge & low level |
| 0 | 1 | Rising edge only |
| 1 | 0 | Falling edge only |
| 1 | 1 | Rising and falling edge |

These 2 bits can be written only when I1 and I0 of the CC register are both set to 1 (level 3).

Bit 2 = **IPA** *Interrupt polarity for port A*
This bit is used to invert the sensitivity of the port A [3:0] external interrupts. It can be set and cleared by software only when I1 and I0 of the CC register are both set to 1 (level 3).
0: No sensitivity inversion
1: Sensitivity inversion

Bit 1 = **TLIS** *TLI sensitivity*
This bit allows to toggle the TLI edge sensitivity. It can be set and cleared by software only when TLIE bit is cleared.
0: Falling edge
1: Rising edge

Bit 0 = **TLIE** *TLI enable*
This bit allows to enable or disable the TLI capability on the dedicated pin. It is set and cleared by software.
0: TLI disabled
1: TLI enabled
**Note**: a parasitic interrupt can be generated when clearing the TLIE bit.

**POWER SAVING MODES** (Cont'd)

### 8.4 ACTIVE-HALT AND HALT MODES

ACTIVE-HALT and HALT modes are the two lowest power consumption modes of the MCU. They are both entered by executing the 'HALT' instruction. The decision to enter either in ACTIVE-HALT or HALT mode is given by the MCC/RTC interrupt enable flag (OIE bit in MCCSR register).

| MCCSR OIE bit | Power Saving Mode entered when HALT instruction is executed |
|---|---|
| 0 | HALT mode |
| 1 | ACTIVE-HALT mode |

#### 8.4.1 ACTIVE-HALT MODE

ACTIVE-HALT mode is the lowest power consumption mode of the MCU with a real time clock available. It is entered by executing the 'HALT' instruction when the OIE bit of the Main Clock Controller Status register (MCCSR) is set (see section 10.2 on page 57 for more details on the MCCSR register).

The MCU can exit ACTIVE-HALT mode on reception of an MCC/RTC interrupt or a RESET. When exiting ACTIVE-HALT mode by means of an interrupt, no 256 or 4096 CPU cycle delay occurs. The CPU resumes operation by servicing the interrupt or by fetching the reset vector which woke it up (see Figure 27).

When entering ACTIVE-HALT mode, the I[1:0] bits in the CC register are forced to '10b' to enable interrupts. Therefore, if an interrupt is pending, the MCU wakes up immediately.

In ACTIVE-HALT mode, only the main oscillator and its associated counter (MCC/RTC) are running to keep a wake-up time base. All other peripherals are not clocked except those which get their clock supply from another clock generator (such as external or auxiliary oscillator).

The safeguard against staying locked in ACTIVE-HALT mode is provided by the oscillator interrupt.

**Note:** As soon as the interrupt capability of one of the oscillators is selected (MCCSR.OIE bit set), entering ACTIVE-HALT mode while the Watchdog is active does not generate a RESET.
This means that the device cannot spend more than a defined delay in this power saving mode.

**CAUTION:** When exiting ACTIVE-HALT mode following an MCC/RTC interrupt, OIE bit of MCCSR register must not be cleared before $t_{DELAY}$ after the interrupt occurs ($t_{DELAY}$ = 256 or 4096 $t_{CPU}$ delay depending on option byte). Otherwise, the ST7 enters HALT mode for the remaining $t_{DELAY}$ period.

**Figure 26. ACTIVE-HALT Timing Overview**

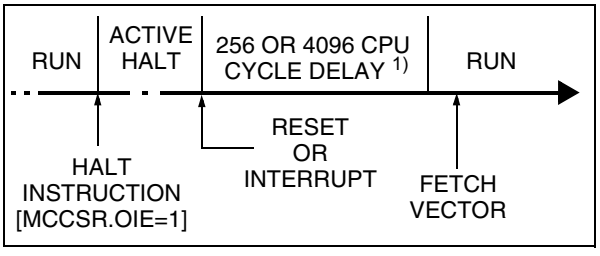
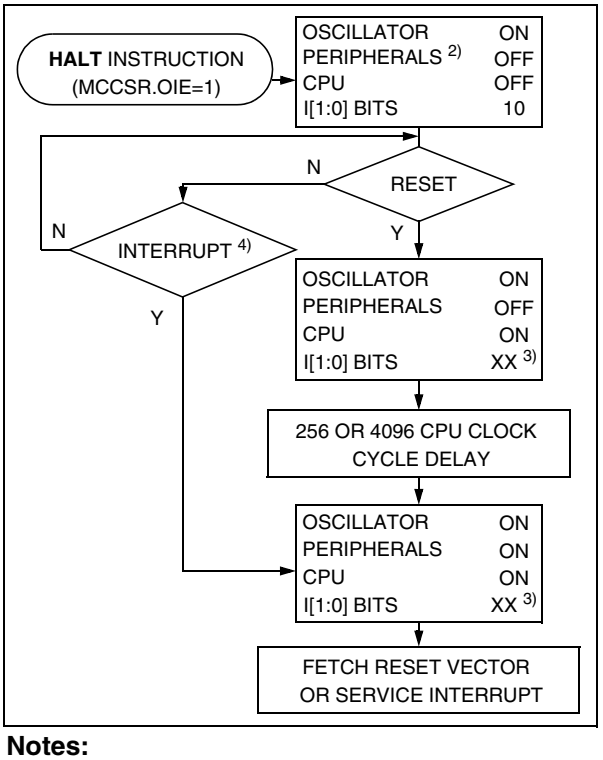



**Figure 27. ACTIVE-HALT Mode Flow-chart**




**Notes:**

1. This delay occurs only if the MCU exits ACTIVE-HALT mode by means of a RESET.
2. Peripheral clocked with an external clock source can still be active.
3. Before servicing an interrupt, the CC register is pushed on the stack. The I[1:0] bits of the CC register are set to the current software priority level of the interrupt routine and restored when the CC register is popped.
4. Only the MCC/RTC interrupt can exit the MCU from ACTIVE-HALT mode.

**WATCHDOG TIMER** (Cont'd)

**Figure 34. Exact Timeout Duration ($t_{min}$ and $t_{max}$)**

**WHERE**:

$t_{min0} = (LSB + 128) \times 64 \times t_{OSC2}$

$t_{max0} = 16384 \times t_{OSC2}$

$t_{OSC2} = 125ns$ if $f_{OSC2}=8$ MHz

CNT = Value of T[5:0] bits in the WDGCR register (6 bits)
MSB and LSB are values from the table below depending on the timebase selected by the TB[1:0] bits in the MCCSR register

| TB1 Bit (MCCSR Reg.) | TB0 Bit (MCCSR Reg.) | Selected MCCSR Timebase | MSB | LSB |
|---|---|---|---|---|
| 0 | 0 | 2ms | 4 | 59 |
| 0 | 1 | 4ms | 8 | 53 |
| 1 | 0 | 10ms | 20 | 35 |
| 1 | 1 | 25ms | 49 | 54 |

**To calculate the minimum Watchdog Timeout ($t_{min}$):**

**IF** $CNT < \left[\frac{MSB}{4}\right]$ **THEN** $t_{min} = t_{min0} + 16384 \times CNT \times t_{osc2}$

**ELSE** $t_{min} = t_{min0} + \left[16384 \times \left(CNT - \left[\frac{4CNT}{MSB}\right]\right) + (192 + LSB) \times 64 \times \left[\frac{4CNT}{MSB}\right]\right] \times t_{osc2}$

**To calculate the maximum Watchdog Timeout ($t_{max}$):**

**IF** $CNT \leq \left[\frac{MSB}{4}\right]$ **THEN** $t_{max} = t_{max0} + 16384 \times CNT \times t_{osc2}$

**ELSE** $t_{max} = t_{max0} + \left[16384 \times \left(CNT - \left[\frac{4CNT}{MSB}\right]\right) + (192 + LSB) \times 64 \times \left[\frac{4CNT}{MSB}\right]\right] \times t_{osc2}$

**Note:** In the above formulae, division results must be rounded down to the next integer value.

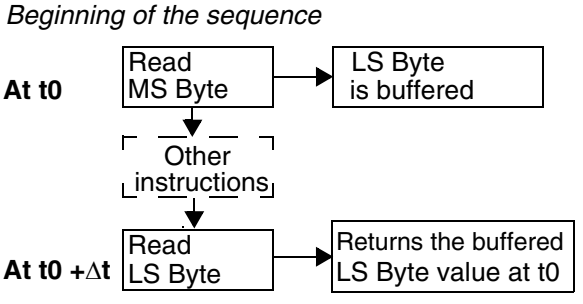
**Example:**

With 2ms timeout selected in MCCSR register

| Value of T[5:0] Bits in WDGCR Register (Hex.) | Min. Watchdog Timeout (ms) $t_{min}$ | Max. Watchdog Timeout (ms) $t_{max}$ |
|---|---|---|
| 00 | 1.496 | 2.048 |
| 3F | 128 | 128.552 |

**16-BIT TIMER** (Cont'd)

**16-bit read sequence:** (from either the Counter Register or the Alternate Counter Register).




The user must read the MS Byte first, then the LS Byte value is buffered automatically.

This buffered value remains unchanged until the 16-bit read sequence is completed, even if the user reads the MS Byte several times.

After a complete reading sequence, if only the CLR register or ACLR register are read, they return the LS Byte of the count value at the time of the read.

Whatever the timer mode used (input capture, output compare, One Pulse mode or PWM mode) an overflow occurs when the counter rolls over from FFFFh to 0000h then:

– The TOF bit of the SR register is set.

– A timer interrupt is generated if:

  – TOIE bit of the CR1 register is set and

  – I bit of the CC register is cleared.

If one of these conditions is false, the interrupt remains pending to be issued as soon as they are both true.

Clearing the overflow interrupt request is done in two steps:

1. Reading the SR register while the TOF bit is set.
2. An access (read or write) to the CLR register.

**Notes:** The TOF bit is not cleared by accesses to ACLR register. The advantage of accessing the ACLR register rather than the CLR register is that it allows simultaneous use of the overflow function and reading the free running counter at random times (for example, to measure elapsed time) without the risk of clearing the TOF bit erroneously.

The timer is not affected by WAIT mode.

In HALT mode, the counter stops counting until the mode is exited. Counting then resumes from the previous count (MCU awakened by an interrupt) or from the reset count (MCU awakened by a Reset).

#### 10.4.3.2 External Clock

The external clock (where available) is selected if CC0 = 1 and CC1 = 1 in the CR2 register.

The status of the EXEDG bit in the CR2 register determines the type of level transition on the external clock pin EXTCLK that will trigger the free running counter.

The counter is synchronized with the falling edge of the internal CPU clock.

A minimum of four falling edges of the CPU clock must occur between two consecutive active edges of the external clock; thus the external clock frequency must be less than a quarter of the CPU clock frequency.

**16-BIT TIMER** (Cont'd)

**Notes:**

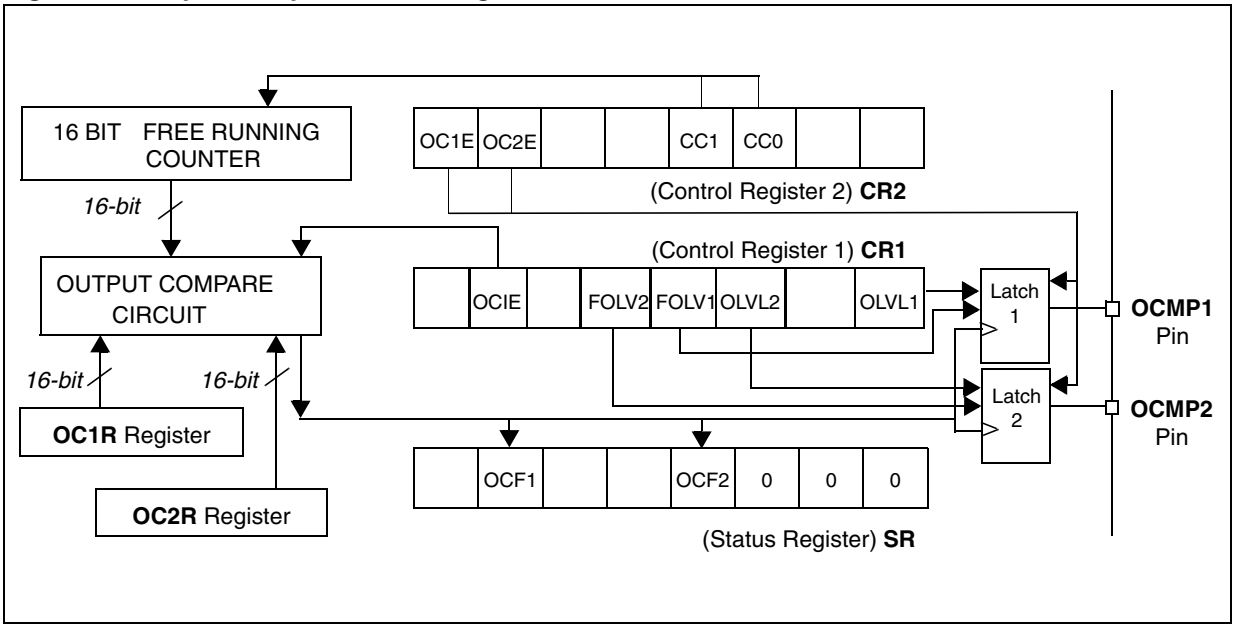
1. After a processor write cycle to the OC*i*HR register, the output compare function is inhibited until the OC*i*LR register is also written.
2. If the OC*i*E bit is not set, the OCMP*i* pin is a general I/O port and the OLVL*i* bit will not appear when a match is found but an interrupt could be generated 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 8 for an example with $f_{CPU}/2$ and Figure 9 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 OCMP*i* pins even if the input capture mode is also used.
5. The value in the 16-bit OC*i*R register and the OLV*i* bit should be changed after each successful comparison in order to control an output waveform or establish a new elapsed timeout.

**Forced Compare Output 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.

**Figure 48. Output Compare Block Diagram**



**16-BIT TIMER** (Cont'd)

### 10.4.4 Low Power Modes

| Mode | Description |
|---|---|
| WAIT | No effect on 16-bit Timer.<br>Timer interrupts cause the device to exit from WAIT mode. |
| HALT | 16-bit Timer registers are frozen.<br>In HALT mode, the counter stops counting until Halt mode is exited. Counting resumes from the previous count when the MCU is woken up by an interrupt with "exit from HALT mode" capability or from the counter reset value when the MCU is woken up by a RESET.<br>If an input capture event occurs on the ICAP*i* pin, the input capture detection circuitry is armed. Consequently, when the MCU is woken up by an interrupt with "exit from HALT mode" capability, the ICF*i* bit is set, and the counter value present when exiting from HALT mode is captured into the IC*i*R register. |

### 10.4.5 Interrupts

| Interrupt Event | Event Flag | Enable Control Bit | Exit from Wait | Exit from Halt |
|---|---|---|---|---|
| Input Capture 1 event/Counter reset in PWM mode | ICF1 | ICIE | Yes | No |
| Input Capture 2 event | ICF2 | | | |
| Output Compare 1 event (not available in PWM mode) | OCF1 | OCIE | | |
| Output Compare 2 event (not available in PWM mode) | OCF2 | | | |
| Timer Overflow event | TOF | TOIE | | |

**Note:** The 16-bit Timer interrupt events are connected to the same interrupt vector (see Interrupts chapter). These events generate an interrupt if the corresponding Enable Control Bit is set and the interrupt mask in the CC register is reset (RIM instruction).

### 10.4.6 Summary of Timer Modes

| MODES | TIMER RESOURCES | | | |
|---|---|---|---|---|
| | Input Capture 1 | Input Capture 2 | Output Compare 1 | Output Compare 2 |
| Input Capture (1 and/or 2) | Yes | Yes | Yes | Yes |
| Output Compare (1 and/or 2) | | | | |
| One Pulse Mode | No | Not Recommended[1)] | No | Partially [2)] |
| PWM Mode | | Not Recommended[3)] | | No |

1) See note 4 in Section 0.1.3.5 One Pulse Mode

2) See note 5 in Section 0.1.3.5 One Pulse Mode

3) See note 4 in Section 0.1.3.6 Pulse Width Modulation Mode

**SERIAL PERIPHERAL INTERFACE** (Cont'd)

– $\overline{SS}$: Slave select:
This input signal acts as a 'chip select' to let the SPI master communicate with slaves individually and to avoid contention on the data lines. Slave $\overline{SS}$ inputs can be driven by standard I/O ports on the master MCU.

#### 10.5.3.1 Functional Description

A basic example of interconnections between a single master and a single slave is illustrated in Figure 54.

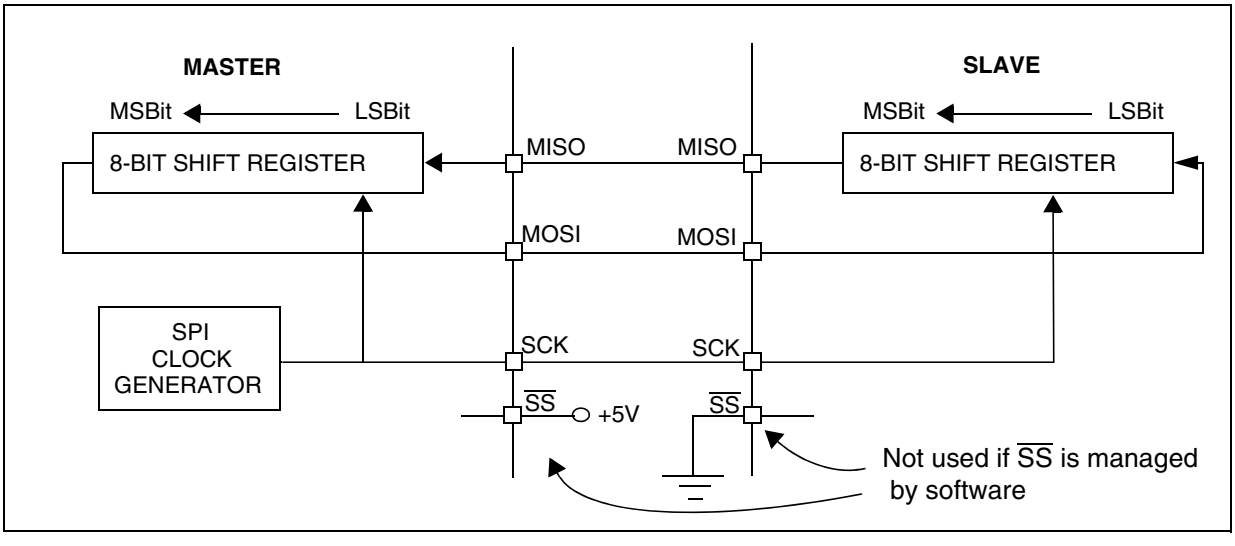
The MOSI pins are connected together and the MISO pins are connected together. In this way data is transferred serially between master and slave (most significant bit first).

The communication is always initiated by the master. When the master device transmits data to a slave device via MOSI pin, the slave device responds by sending data to the master device via the MISO pin. This implies full duplex communication with both data out and data in synchronized with the same clock signal (which is provided by the master device via the SCK pin).

To use a single data line, the MISO and MOSI pins must be connected at each node ( in this case only simplex communication is possible).

Four possible data/clock timing relationships may be chosen (see Figure 57) but master and slave must be programmed with the same timing mode.

**Figure 54. Single Master/ Single Slave Application**



**SERIAL PERIPHERAL INTERFACE** (Cont'd)

**CONTROL/STATUS REGISTER (SPICSR)**
Read/Write (some bits Read Only)
Reset Value: 0000 0000 (00h)

| 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|
| SPIF | WCOL | OVR | MODF | - | SOD | SSM | SSI |

Bit 7 = **SPIF** *Serial Peripheral Data Transfer Flag (Read only).*
This bit is set by hardware when a transfer has been completed. An interrupt is generated if SPIE=1 in the SPICR register. It is cleared by a software sequence (an access to the SPICSR register followed by a write or a read to the SPIDR register).
0: Data transfer is in progress or the flag has been cleared.
1: Data transfer between the device and an external device has been completed.

**Note:** While the SPIF bit is set, all writes to the SPIDR register are inhibited until the SPICSR register is read.

Bit 6 = **WCOL** *Write Collision status (Read only).*
This bit is set by hardware when a write to the SPIDR register is done during a transmit sequence. It is cleared by a software sequence (see Figure 58).
0: No write collision occurred
1: A write collision has been detected

Bit 5 = **OVR** S*PI Overrun error (Read only).*
This bit is set by hardware when the byte currently being received in the shift register is ready to be transferred into the SPIDR register while SPIF = 1 (See Section 10.5.5.2). An interrupt is generated if SPIE = 1 in SPICR register. The OVR bit is cleared by software reading the SPICSR register.
0: No overrun error
1: Overrun error detected

Bit 4 = **MODF** *Mode Fault flag (Read only).*
This bit is set by hardware when the $\overline{SS}$ pin is pulled low in master mode (see Section 10.5.5.1 Master Mode Fault (MODF)). An SPI interrupt can be generated if SPIE=1 in the SPICSR register. This bit is cleared by a software sequence (An access to the SPICR register while MODF=1 followed by a write to the SPICR register).
0: No master mode fault detected
1: A fault in master mode has been detected

Bit 3 = Reserved, must be kept cleared.

Bit 2 = **SOD** *SPI Output Disable.*
This bit is set and cleared by software. When set, it disables the alternate function of the SPI output (MOSI in master mode / MISO in slave mode)
0: SPI output enabled (if SPE=1)
1: SPI output disabled

Bit 1 = **SSM** $\overline{SS}$ *Management.*
This bit is set and cleared by software. When set, it disables the alternate function of the SPI $\overline{SS}$ pin and uses the SSI bit value instead. See Section 10.5.3.2 Slave Select Management.
0: Hardware management ($\overline{SS}$ managed by external pin)
1: Software management (internal $\overline{SS}$ signal controlled by SSI bit. External $\overline{SS}$ pin free for general-purpose I/O)

Bit 0 = **SSI** $\overline{SS}$ *Internal Mode.*
This bit is set and cleared by software. It acts as a 'chip select' by controlling the level of the $\overline{SS}$ slave select signal when the SSM bit is set.
0 : Slave selected
1 : Slave deselected

**DATA I/O REGISTER (SPIDR)**
Read/Write
Reset Value: Undefined

| 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

The SPIDR register is used to transmit and receive data on the serial bus. In a master device, a write to this register will initiate transmission/reception of another byte.

**Notes:** During the last clock cycle the SPIF bit is set, a copy of the received data byte in the shift register is moved to a buffer. When the user reads the serial peripheral data I/O register, the buffer is actually being read.

While the SPIF bit is set, all writes to the SPIDR register are inhibited until the SPICSR register is read.

**Warning:** A write to the SPIDR register places data directly into the shift register for transmission.

A read to the SPIDR register returns the value located in the buffer and not the content of the shift register (see Figure 53).

**SERIAL COMMUNICATIONS INTERFACE** (Cont'd)

**CONTROL REGISTER 1 (SCICR1)**

Read/Write

Reset Value: x000 0000 (x0h)

| 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|
| R8 | T8 | SCID | M | WAKE | PCE | PS | PIE |

Bit 7 = **R8** *Receive data bit 8.*
This bit is used to store the 9th bit of the received word when M = 1.

Bit 6 = **T8** *Transmit data bit 8.*
This bit is used to store the 9th bit of the transmitted word when M = 1.

Bit 5 = **SCID** *Disabled for low power consumption*
When this bit is set the SCI prescalers and outputs are stopped and the end of the current byte transfer in order to reduce power consumption.This bit is set and cleared by software.
0: SCI enabled
1: SCI prescaler and outputs disabled

Bit 4 = **M** *Word length.*
This bit determines the word length. It is set or cleared by software.
0: 1 Start bit, 8 Data bits, 1 Stop bit
1: 1 Start bit, 9 Data bits, 1 Stop bit

**Note**: The M bit must not be modified during a data transfer (both transmission and reception).

Bit 3 = **WAKE** *Wake-Up method.*
This bit determines the SCI Wake-Up method, it is set or cleared by software.
0: Idle Line
1: Address Mark

Bit 2 = **PCE** *Parity control enable.*
This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9th bit if M = 1; 8th bit if M = 0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCE is active after the current byte (in reception and in transmission).
0: Parity control disabled
1: Parity control enabled

Bit 1 = **PS** *Parity selection.*
This bit selects the odd or even parity when the parity generation/detection is enabled (PCE bit set). It is set and cleared by software. The parity is selected after the current byte.
0: Even parity
1: Odd parity

Bit 0 = **PIE** *Parity interrupt enable.*
This bit enables the interrupt capability of the hardware parity control when a parity error is detected (PE bit set). It is set and cleared by software.
0: Parity error interrupt disabled
1: Parity error interrupt enabled.

**SERIAL COMMUNICATIONS INTERFACE** (Cont'd)

**CONTROL REGISTER 2 (SCICR2)**

Read/Write

Reset Value: 0000 0000 (00h)

| 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|
| TIE | TCIE | RIE | ILIE | TE | RE | RWU | SBK |

Bit 7 = **TIE** *Transmitter interrupt enable*.
This bit is set and cleared by software.
0: Interrupt is inhibited
1: An SCI interrupt is generated whenever TDRE=1 in the SCISR register

Bit 6 = TCIE *Transmission complete interrupt enable*
This bit is set and cleared by software.
0: Interrupt is inhibited
1: An SCI interrupt is generated whenever TC=1 in the SCISR register

Bit 5 = **RIE** *Receiver interrupt enable*.
This bit is set and cleared by software.
0: Interrupt is inhibited
1: An SCI interrupt is generated whenever OR=1 or RDRF=1 in the SCISR register

Bit 4 = **ILIE** *Idle line interrupt enable.*
This bit is set and cleared by software.
0: Interrupt is inhibited
1: An SCI interrupt is generated whenever IDLE=1 in the SCISR register.

Bit 3 = **TE** *Transmitter enable.*
This bit enables the transmitter. It is set and cleared by software.
0: Transmitter is disabled
1: Transmitter is enabled

**Notes:**

– During transmission, a "0" pulse on the TE bit ("0" followed by "1") sends a preamble (idle line) after the current word.

– When TE is set there is a 1 bit-time delay before the transmission starts.

**CAUTION:** The TDO pin is free for general purpose I/O only when the TE and RE bits are both cleared (or if TE is never set).

Bit 2 = **RE** *Receiver enable.*
This bit enables the receiver. It is set and cleared by software.
0: Receiver is disabled
1: Receiver is enabled and begins searching for a start bit

Bit 1 = **RWU** *Receiver wake-up.*
This bit determines if the SCI is in mute mode or not. It is set and cleared by software and can be cleared by hardware when a wake-up sequence is recognized.
0: Receiver in Active mode
1: Receiver in Mute mode

**Note:** Before selecting Mute mode (setting the RWU bit), the SCI must receive some data first, otherwise it cannot function in Mute mode with wake-up by idle line detection.

Bit 0 = **SBK** *Send break.*
This bit set is used to send break characters. It is set and cleared by software.
0: No break character is transmitted
1: Break characters are transmitted

**Note:** If the SBK bit is set to "1" and then to "0", the transmitter sends a BREAK word at the end of the current word.

**INSTRUCTION SET OVERVIEW** (Cont'd)

| Mnemo | Description | Function/Example | Dst | Src | | I1 | H | I0 | N | Z | C |
|---|---|---|---|---|---|---|---|---|---|---|---|
| JRULE | Jump if (C + Z = 1) | Unsigned <= | | | | | | | | | |
| LD | Load | dst <= src | reg, M | M, reg | | | | | N | Z | |
| MUL | Multiply | X,A = X * A | A, X, Y | X, Y, A | | | 0 | | | | 0 |
| NEG | Negate (2's compl) | neg $10 | reg, M | | | | | | N | Z | C |
| NOP | No Operation | | | | | | | | | | |
| OR | OR operation | A = A + M | A | M | | | | | N | Z | |
| POP | Pop from the Stack | pop reg | reg | M | | | | | | | |
| | | pop CC | CC | M | | I1 | H | I0 | N | Z | C |
| PUSH | Push onto the Stack | push Y | M | reg, CC | | | | | | | |
| RCF | Reset carry flag | C = 0 | | | | | | | | | 0 |
| RET | Subroutine Return | | | | | | | | | | |
| RIM | Enable Interrupts | I1:0 = 10 (level 0) | | | | 1 | | 0 | | | |
| RLC | Rotate left true C | C <= A <= C | reg, M | | | | | | N | Z | C |
| RRC | Rotate right true C | C => A => C | reg, M | | | | | | N | Z | C |
| RSP | Reset Stack Pointer | S = Max allowed | | | | | | | | | |
| SBC | Substract with Carry | A = A - M - C | A | M | | | | | N | Z | C |
| SCF | Set carry flag | C = 1 | | | | | | | | | 1 |
| SIM | Disable Interrupts | I1:0 = 11 (level 3) | | | | 1 | | 1 | | | |
| SLA | Shift left Arithmetic | C <= A <= 0 | reg, M | | | | | | N | Z | C |
| SLL | Shift left Logic | C <= A <= 0 | reg, M | | | | | | N | Z | C |
| SRL | Shift right Logic | 0 => A => C | reg, M | | | | | | 0 | Z | C |
| SRA | Shift right Arithmetic | A7 => A => C | reg, M | | | | | | N | Z | C |
| SUB | Substraction | A = A - M | A | M | | | | | N | Z | C |
| SWAP | SWAP nibbles | A7-A4 <=> A3-A0 | reg, M | | | | | | N | Z | |
| TNZ | Test for Neg & Zero | tnz lbl1 | | | | | | | N | Z | |
| TRAP | S/W trap | S/W interrupt | | | | 1 | | 1 | | | |
| WFI | Wait for Interrupt | | | | | 1 | | 0 | | | |
| XOR | Exclusive OR | A = A XOR M | A | M | | | | | N | Z | |

## 12.6 MEMORY CHARACTERISTICS

### 12.6.1 RAM and Hardware Registers

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $V_{RM}$ | Data retention mode [1] | HALT mode (or RESET) | 1.6 | | | V |

### 12.6.2 FLASH Memory

| DUAL VOLTAGE HDFLASH MEMORY | | | | | | |
|---|---|---|---|---|---|---|
| **Symbol** | **Parameter** | **Conditions** | **Min [2]** | **Typ** | **Max [2]** | **Unit** |
| $f_{CPU}$ | Operating frequency | Read mode | 0 | | 8 | MHz |
| | | Write / Erase mode | 1 | | 8 | |
| $V_{PP}$ | Programming voltage [3] | $4.5V \leq V_{DD} \leq 5.5V$ | 11.4 | | 12.6 | V |
| $I_{DD}$ | Supply current[4] | RUN mode ($f_{CPU}$ = 4MHz) | | | 3 | mA |
| | | Write / Erase | | 0 | | |
| | | Power down mode / HALT | | 1 | 10 | µA |
| $I_{PP}$ | $V_{PP}$ current[4] | Read ($V_{PP}$=12V) | | | 200 | |
| | | Write / Erase | | | 30 | mA |
| $t_{VPP}$ | Internal $V_{PP}$ stabilization time | | | 10 | | µs |
| $t_{RET}$ | Data retention | $T_A$=85°C | 40 | | | years |
| | | $T_A$=105°C | 15 | | | |
| | | $T_A$=125°C | 7 | | | |
| $N_{RW}$ | Write erase cycles | $T_A$= 55°C | 1000 | | | cycles |
| | | $T_A$= 85°C | 100 | | | cycles |
| $T_{PROG}$ $T_{ERASE}$ | Programming or erasing temperature range | | -40 | 25 | 85 | °C |

**Notes:**

1. Minimum $V_{DD}$ supply voltage without losing data stored in RAM (in HALT mode or under RESET) or in hardware registers (only in HALT mode). Not tested in production.
2. Data based on characterization results, not tested in production.
3. $V_{PP}$ must be applied only during the programming or erasing operation and not permanently for reliability reasons.
4. Data based on simulation results, not tested in production.

**Warning:** Do not connect 12V to $V_{PP}$ before $V_{DD}$ is powered on, as this may damage the device.

## 12.7 EMC CHARACTERISTICS

Susceptibility tests are performed on a sample basis during product characterization.

### 12.7.1 Functional EMS (Electro Magnetic Susceptibility)

Based on a simple running application on the product (toggling 2 LEDs through I/O ports), the product is stressed by two electro magnetic events until a failure occurs (indicated by the LEDs).

- **ESD**: Electro-Static Discharge (positive and negative) is applied on all pins of the device until a functional disturbance occurs. This test conforms with the IEC 1000-4-2 standard.
- **FTB**: A Burst of Fast Transient voltage (positive and negative) is applied to $V_{DD}$ and $V_{SS}$ through a 100pF capacitor, until a functional disturbance occurs. This test conforms with the IEC 1000-4-4 standard.

A device reset allows normal operations to be resumed. The test results are given in the table below based on the EMS levels and classes defined in application note AN1709.

#### 12.7.1.1 Designing hardened software to avoid noise problems

EMC characterization and optimization are performed at component level with a typical application environment and simplified MCU software. It should be noted that good EMC performance is highly dependent on the user application and the software in particular.

Therefore it is recommended that the user applies EMC software optimization and prequalification tests in relation with the EMC level requested for his application.

**Software recommendations:**

The software flowchart must include the management of runaway conditions such as:

– Corrupted program counter

– Unexpected reset

– Critical Data corruption (control registers...)

**Prequalification trials:**

Most of the common failures (unexpected reset and program counter corruption) can be reproduced by manually forcing a low state on the RESET pin or the Oscillator pins for 1 second.

To complete these trials, ESD stress can be applied directly on the device, over the range of specification values. When unexpected behaviour is detected, the software can be hardened to prevent unrecoverable errors occurring (see application note AN1015).

| Symbol | Parameter | Conditions | Level/ Class |
|---|---|---|---|
| $V_{FESD}$ | Voltage limits to be applied on any I/O pin to induce a functional disturbance | Flash device: $V_{DD}$=5V, $T_A$=+25°C, $f_{OSC}$=8MHz conforms to IEC 1000-4-2 | 4B |
| | | ROM device: $V_{DD}$=5V, $T_A$=+25°C, $f_{OSC}$=8 MHz, conforms to IEC 1000-4-2 | 4A |
| $V_{FFTB}$ | Fast transient voltage burst limits to be applied through 100pF on $V_{DD}$ and $V_{DD}$ pins to induce a functional disturbance | $V_{DD}$=5V, $T_A$=+25°C, $f_{OSC}$=8 MHz, conforms to IEC 1000-4-4 | 3B |
| $V_{FFTB}$ | Fast transient voltage burst limits to be applied through 100pF on $V_{DD}$ and $V_{DD}$ pins to induce a functional disturbance | $V_{DD}$=5V, $T_A$=+25°C, $f_{OSC}$=8 MHz, conforms to IEC 1000-4-4 | 3B |

**COMMUNICATION INTERFACE CHARACTERISTICS** (Cont'd)

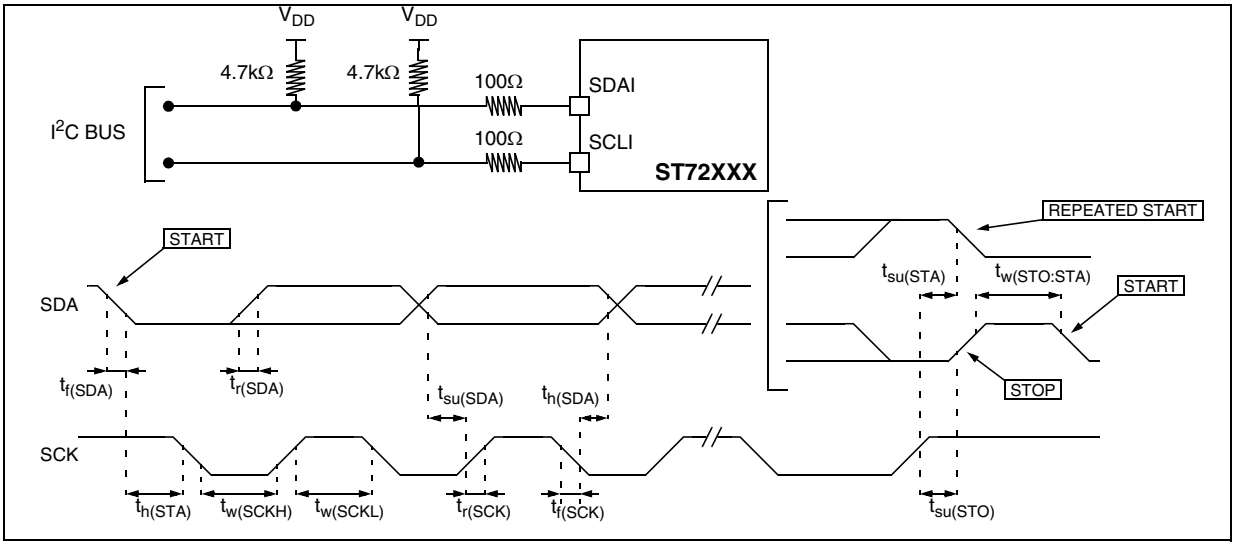
### 12.11.2 $I^2C$ - Inter IC Control Interface

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

Refer to I/O port characteristics for more details on the input/output alternate function characteristics (SDAI and SCLI). The ST7 $I^2C$ interface meets the requirements of the Standard $I^2C$ communication protocol described in the following table.

| Symbol | Parameter | Standard mode $I^2C$ | | Fast mode $I^2C$ [5] | | Unit |
|---|---|---|---|---|---|---|
| | | Min [1] | Max [1] | Min [1] | Max [1] | |
| $t_{w(SCLL)}$ | SCL clock low time | 4.7 | | 1.3 | | µs |
| $t_{w(SCLH)}$ | SCL clock high time | 4.0 | | 0.6 | | µs |
| $t_{su(SDA)}$ | SDA setup time | 250 | | 100 | | ns |
| $t_{h(SDA)}$ | SDA data hold time | 0 [3] | | 0 [2] | 900 [3] | ns |
| $t_{r(SDA)}$ $t_{r(SCL)}$ | SDA and SCL rise time | | 1000 | $20+0.1C_b$ | 300 | ns |
| $t_{f(SDA)}$ $t_{f(SCL)}$ | SDA and SCL fall time | | 300 | $20+0.1C_b$ | 300 | ns |
| $t_{h(STA)}$ | START condition hold time | 4.0 | | 0.6 | | µs |
| $t_{su(STA)}$ | Repeated START condition setup time | 4.7 | | 0.6 | | µs |
| $t_{su(STO)}$ | STOP condition setup time | 4.0 | | 0.6 | | µs |
| $t_{w(STO:STA)}$ | STOP to START condition time (bus free) | 4.7 | | 1.3 | | µs |
| $C_b$ | Capacitive load for each bus line | | 400 | | 400 | pF |

**Figure 95. Typical Application with $I^2C$ Bus and Timing Diagram** [4]




**Notes:**

1. Data based on standard $I^2C$ protocol requirement, not tested in production.
2. The device must internally provide a hold time of at least 300ns for the SDA signal in order to bridge the undefined region of the falling edge of SCL.
3. The maximum hold time of the START condition has only to be met if the interface does not stretch the low period of SCL signal.
4. Measurement points are done at CMOS levels: $0.3xV_{DD}$ and $0.7xV_{DD}$.
5. At 4MHz $f_{CPU}$, max.$I^2C$ speed (400kHz) is not achievable. In this case, max. $I^2C$ speed will be approximately 260KHz.

**Table 31. ST7 Application Notes**

| IDENTIFICATION | DESCRIPTION |
|---|---|
| AN1071 | HALF DUPLEX USB-TO-SERIAL BRIDGE USING THE ST72611 USB MICROCONTROLLER |
| AN1106 | TRANSLATING ASSEMBLY CODE FROM HC05 TO ST7 |
| AN1179 | PROGRAMMING ST7 FLASH MICROCONTROLLERS IN REMOTE ISP MODE (IN-SITU PROGRAMMING) |
| AN1446 | USING THE ST72521 EMULATOR TO DEBUG AN ST72324 TARGET APPLICATION |
| AN1477 | EMULATED DATA EEPROM WITH XFLASH MEMORY |
| AN1527 | DEVELOPING A USB SMARTCARD READER WITH ST7SCR |
| AN1575 | ON-BOARD PROGRAMMING METHODS FOR XFLASH AND HDFLASH ST7 MCUS |
| AN1576 | IN-APPLICATION PROGRAMMING (IAP) DRIVERS FOR ST7 HDFLASH OR XFLASH MCUS |
| AN1577 | DEVICE FIRMWARE UPGRADE (DFU) IMPLEMENTATION FOR ST7 USB APPLICATIONS |
| AN1601 | SOFTWARE IMPLEMENTATION FOR ST7DALI-EVAL |
| AN1603 | USING THE ST7 USB DEVICE FIRMWARE UPGRADE DEVELOPMENT KIT (DFU-DK) |
| AN1635 | ST7 CUSTOMER ROM CODE RELEASE INFORMATION |
| AN1754 | DATA LOGGING PROGRAM FOR TESTING ST7 APPLICATIONS VIA ICC |
| AN1796 | FIELD UPDATES FOR FLASH BASED ST7 APPLICATIONS USING A PC COMM PORT |
| AN1900 | HARDWARE IMPLEMENTATION FOR ST7DALI-EVAL |
| AN1904 | ST7MC THREE-PHASE AC INDUCTION MOTOR CONTROL SOFTWARE LIBRARY |
| AN1905 | ST7MC THREE-PHASE BLDC MOTOR CONTROL SOFTWARE LIBRARY |
| **SYSTEM OPTIMIZATION** | |
| AN1711 | SOFTWARE TECHNIQUES FOR COMPENSATING ST7 ADC ERRORS |
| AN1827 | IMPLEMENTATION OF SIGMA-DELTA ADC WITH ST7FLITE05/09 |
| AN2009 | PWM MANAGEMENT FOR 3-PHASE BLDC MOTOR DRIVES USING THE ST7FMC |
| AN2030 | BACK EMF DETECTION DURING PWM ON TIME BY ST7MC |

```
TNZ Y
jrne OUT
LD A,sema     ; check the semaphore status if edge is detected
CP A,#01
jrne OUT
call call_routine; call the interrupt routine
OUT:LD A,#00
LD sema,A
.call_routine     ; entry to call_routine
PUSH A
PUSH X
PUSH CC
.ext1_rt          ; entry to interrupt routine
LD A,#00
LD sema,A
IRET
```

**Case 2:** Writing to PxOR or PxDDR with Global Interrupts Disabled:

```
SIM               ; set the interrupt mask
LD A,PFDR
AND A,#$02
LD X,A       ; store the level before writing to PxOR/PxDDR
LD A,#$90
LD PFDDR,A; Write into PFDDR
LD A,#$ff
LD PFOR,A          ; Write to PFOR
LD A,PFDR
AND A,#$02
LD Y,A       ; store the level after writing to PxOR/PxDDR
LD A,X       ; check for falling edge
cp A,#$02
jrne OUT
TNZ Y
jrne OUT
LD A,#$01
LD sema,A  ; set the semaphore to '1' if edge is detected
RIM                ; reset the interrupt mask
LD A,sema  ; check the semaphore status
CP A,#$01
jrne OUT
call call_routine; call the interrupt routine
RIM
OUT:          RIM
JP while_loop
.call_routine ; entry to call_routine
PUSH A
PUSH X
PUSH CC
.ext1_rt      ; entry to interrupt routine
LD A,#$00
LD sema,A
IRET
```

### 15.1.6 Clearing active interrupts outside interrupt routine

When an active interrupt request occurs at the same time as the related flag is being cleared, an unwanted reset may occur.

**Note:** clearing the related interrupt mask will not generate an unwanted reset

**Concurrent interrupt context**

The symptom does not occur when the interrupts are handled normally, i.e.

when:

– The interrupt flag is cleared within its own interrupt routine

– The interrupt flag is cleared within any interrupt routine

– The interrupt flag is cleared in any part of the code while this interrupt is disabled

If these conditions are not met, the symptom can be avoided by implementing the following sequence:

Perform SIM and RIM operation before and after resetting an active interrupt request.

Example:

```
SIM
reset interrupt flag
RIM
```

**Nested interrupt context:**

The symptom does not occur when the interrupts are handled normally, i.e.

when:

– The interrupt flag is cleared within its own interrupt routine