Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

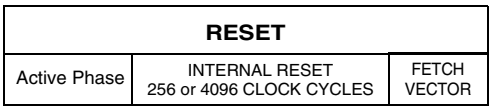
"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" 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	CANbus, LINbusSCI, SPI
Peripherals	LVD, POR, PWM, WDT
Number of I/O	48
Program Memory Size	60KB (60K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	2K x 8
Voltage - Supply (Vcc/Vdd)	3.8V ~ 5.5V
Data Converters	A/D 16x10b
Oscillator Type	External
Operating Temperature	-40°C ~ 125°C (TA)
Mounting Type	Surface Mount
Package / Case	64-LQFP
Supplier Device Package	-
Purchase URL	https://www.e-xfl.com/product-detail/stmicroelectronics/st72f561r9tcs

The reset vector fetch phase duration is two clock cycles.

Figure 12. RESET sequence phases

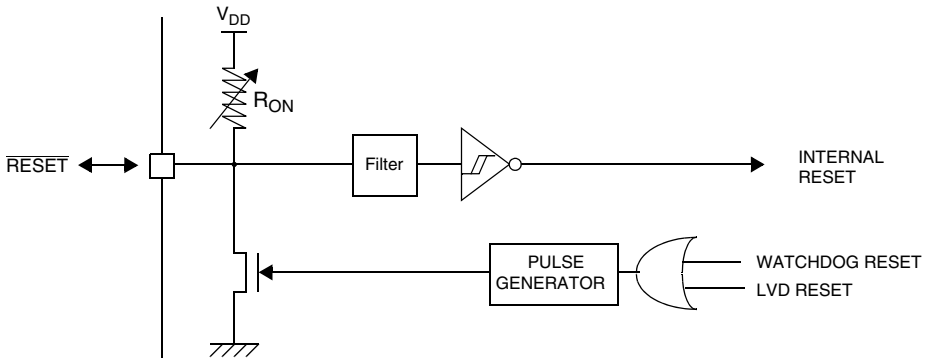


5.5.2 Asynchronous external $\overline{\text{RESET}}$ pin

The $\overline{\text{RESET}}$ pin is both an input and an open-drain output with integrated R_{ON} weak pull-up resistor. This pull-up has no fixed value but varies in accordance with the input voltage. It can be pulled low by external circuitry to reset the device. See *Chapter 20: Electrical characteristics* for more details.

A reset signal originating from an external source must have a duration of at least $t_{h(RSTL)in}$ in order to be recognized (see *Figure 14*). This detection is asynchronous and therefore the MCU can enter reset state even in HALT mode.

Figure 13. Reset block diagram



The $\overline{\text{RESET}}$ pin is an asynchronous signal which plays a major role in EMS performance. In a noisy environment, it is recommended to follow the guidelines mentioned in the electrical characteristics section.

5.5.3 External power-on reset

If the LVD is disabled by option byte, to start up the microcontroller correctly, the user must ensure by means of an external reset circuit that the reset signal is held low until V_{DD} is over the minimum level specified for the selected f_{OSC} frequency.

A proper reset signal for a slow rising V_{DD} supply can generally be provided by an external RC network connected to the $\overline{\text{RESET}}$ pin.

5.5.4 Internal low voltage detector (LVD) reset

Two different reset sequences caused by the internal LVD circuitry can be distinguished:

- Power-on reset
- Voltage drop reset

6.6 External interrupts

6.6.1 I/O port interrupt sensitivity

The external interrupt sensitivity is controlled by the ISxx bits in the EICR register (*Figure 21*). This control allows up to four fully independent external interrupt source sensitivities.

Each external interrupt source can be generated on four (or five) different events on the pin:

- Falling edge
- Rising edge
- Falling and rising edge
- Falling edge and low level

To guarantee correct functionality, the sensitivity bits in the EICR register can be modified only when the I1 and I0 bits of the CC register are both set to 1 (level 3). This means that interrupts must be disabled before changing sensitivity.

The pending interrupts are cleared by writing a different value in the ISx[1:0] of the EICR.

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 level sensitiveness 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).

7.5 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 MCC/RTC interrupt enable flag (OIE bit in MCCSR register) is set and when the AWUEN bit in the AWUCSR register is cleared (*Section 7.6.1: Register description*)

Table 22. MCC/RTC low power mode selection

MCCSR OIE bit	Power saving mode entered when HALT instruction is executed
0	HALT mode
1	ACTIVE HALT mode

The MCU can exit ACTIVE HALT mode on reception of the RTC interrupt and some specific interrupts (see *Table 16*) or a RESET. When exiting ACTIVE HALT mode by means of a RESET a 4096 or 256 CPU cycle delay occurs (depending on the option byte). After the start up delay, the CPU resumes operation by servicing the interrupt or by fetching the reset vector which woke it up (see *Figure 28*).

When entering ACTIVE HALT mode, the I[1:0] bits in the CC register are 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.

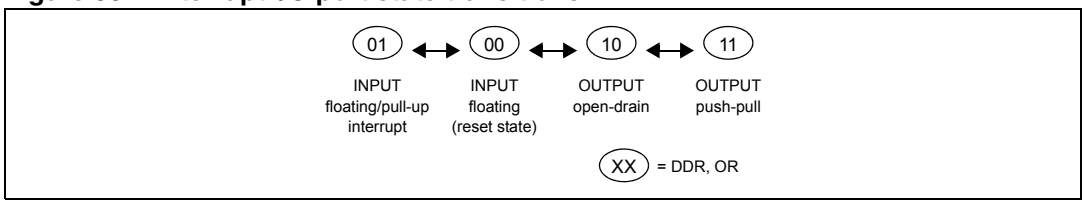
Warning: **The analog input voltage level must be within the limits stated in the absolute maximum ratings.**

8.3 I/O port implementation

The hardware implementation on each I/O port depends on the settings in the DDR and OR registers and specific feature of the I/O port such as ADC Input or true open drain.

Switching these I/O ports from one state to another should be done in a sequence that prevents unwanted side effects. Recommended safe transitions are illustrated in *Figure 33*. Other transitions are potentially risky and should be avoided, since they are likely to present unwanted side-effects such as spurious interrupt generation.

Figure 33. Interrupt I/O port state transitions



8.4 I/O port register configurations

The I/O port register configurations are summarized as follows.

8.4.1 Standard ports

Table 28. Configuration of PB7:6, PC0, PC3, PC7:5, PD3:2, PD5, PE7:0, PF7:0

Mode	DDR	OR
Floating input	0	0
Pull-up input		1
Open drain output	1	0
Push-pull output		1

9.4 Using halt mode with the WDG

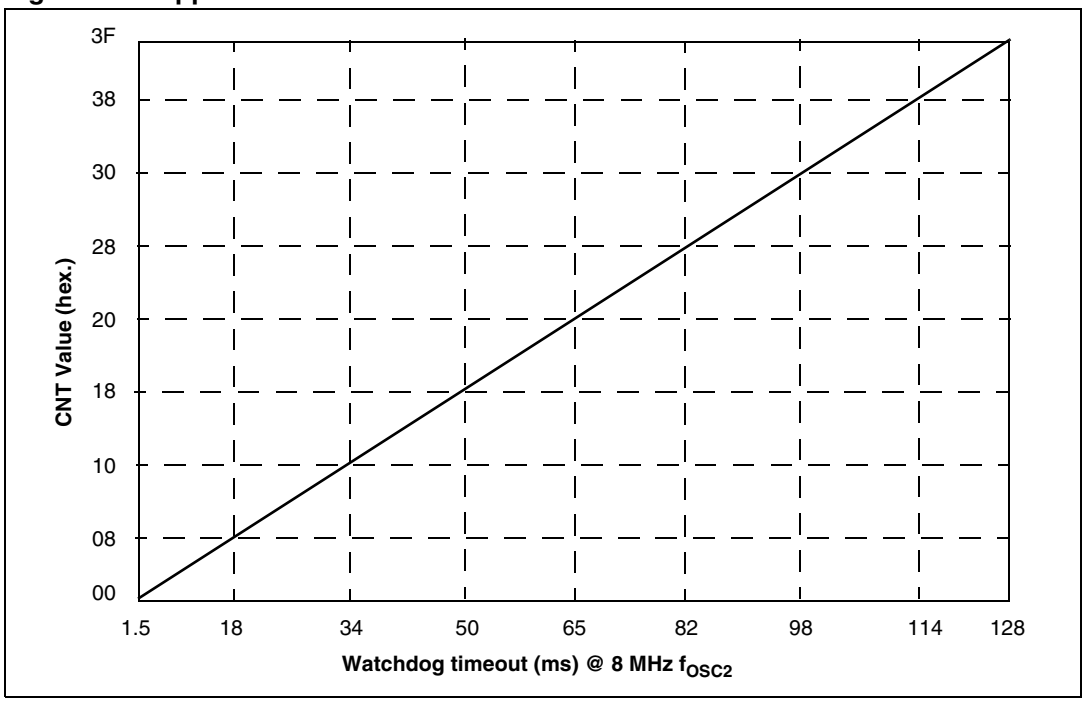
If Halt mode with Watchdog is enabled by option byte (no watchdog reset on HALT instruction), it is recommended before executing the HALT instruction to refresh the WDG counter, to avoid an unexpected WDG reset immediately after waking up the microcontroller.

9.5 How to program the watchdog timeout

Figure 35 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 36*.

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

Figure 35. Approximate timeout duration



13.3.2 Input capture

In this section, the index, *i*, may be 1 or 2 because there are two input capture functions in the 8-bit timer.

The two 8-bit input capture registers (IC1R and IC2R) are used to latch the value of the free running counter after a transition is detected on the ICAP*i* pin (see *Figure 63*).

IC*i*R register is a read-only register.

The active transition is software programmable through the IEDG*i* bit of Control Registers (CR*i*).

Timing resolution is one count of the free running counter (see *Table 55*).

Procedure

To use the input capture function select the following in the CR2 register:

- Select the timer clock (CC[1:0]) (see *Table 55*).
- Select the edge of the active transition on the ICAP2 pin with the IEDG2 bit (the ICAP2 pin must be configured as floating input or input with pull-up without interrupt if this configuration is available).

And select the following in the CR1 register:

- Set the ICIE bit to generate an interrupt after an input capture coming from either the ICAP1 pin or the ICAP2 pin
- Select the edge of the active transition on the ICAP1 pin with the IEDG1 bit (the ICAP1 pin must be configured as floating input or input with pull-up without interrupt if this configuration is available).

When an input capture occurs:

- ICF*i* bit is set.
- The IC*i*R register contains the value of the free running counter on the active transition on the ICAP*i* pin (see *Figure 64*).
- A timer interrupt is generated if the ICIE bit is set and the interrupt mask is cleared in the CC register. Otherwise, the interrupt remains pending until both conditions become true.

Clearing the Input Capture interrupt request (that is, clearing the ICF*i* bit) is done in two steps:

1. Reading the SR register while the ICF*i* bit is set.
2. An access (read or write) to the IC*i*R register.

Note: 1 The ICiR register contains the free running counter value which corresponds to the most recent input capture.

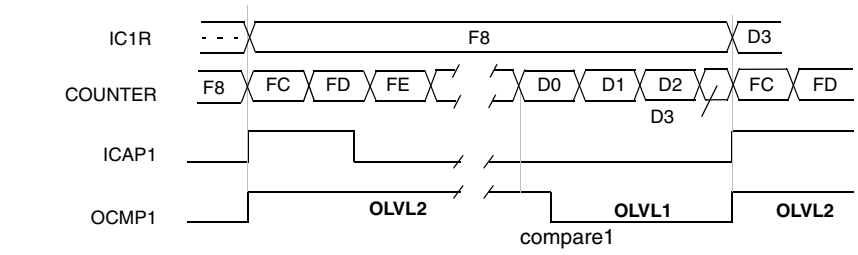
2 The two input capture functions can be used together even if the timer also uses the two output compare functions.

3 Once the ICIE bit is set both input capture features may trigger interrupt requests. If only one is needed in the application, the interrupt routine software needs to discard the unwanted capture interrupt. This can be done by checking the ICF1 and ICF2 flags and resetting them both.

4 In One pulse Mode and PWM mode only Input Capture 2 can be used.

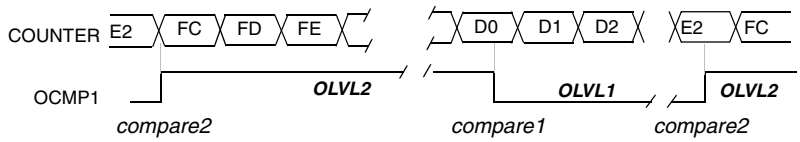
5 The alternate inputs (ICAP1 and ICAP2) are always directly connected to the timer. So any transitions on these pins activates the input capture function.

Figure 68. One pulse mode timing example



Note: IEDG1 = 1, OC1R = D0h, OLVL1 = 0, OLVL2 = 1

Figure 69. Pulse width modulation mode timing example



Note: OC1R = D0h, OC2R = E2, OLVL1 = 0, OLVL2 = 1

13.3.6 Pulse width modulation mode

Pulse Width Modulation (PWM) mode enables the generation of a signal with a frequency and pulse length determined by the value of the OC1R and OC2R registers.

Pulse Width Modulation mode uses the complete Output Compare 1 function plus the OC2R register, and so this functionality can not be used when PWM mode is activated.

In PWM mode, double buffering is implemented on the output compare registers. Any new values written in the OC1R and OC2R registers are taken into account only at the end of the PWM period (OC2) to avoid spikes on the PWM output pin (OCMP1).

Procedure

To use pulse width modulation mode:

set and IC2R can be loaded) but the user must take care that the counter is reset each period and ICF1 can also generates interrupt if ICIE is set.

4 *When the Pulse Width Modulation (PWM) and One Pulse Mode (OPM) bits are both set, the PWM mode is the only active one.*

13.4 Low power modes

Table 52. Effect of low power modes on TIM8

Mode	Description
WAIT	No effect on 8-bit Timer. Timer interrupts cause the device to exit from WAIT mode.
HALT	8-bit Timer registers are frozen. 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. 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.

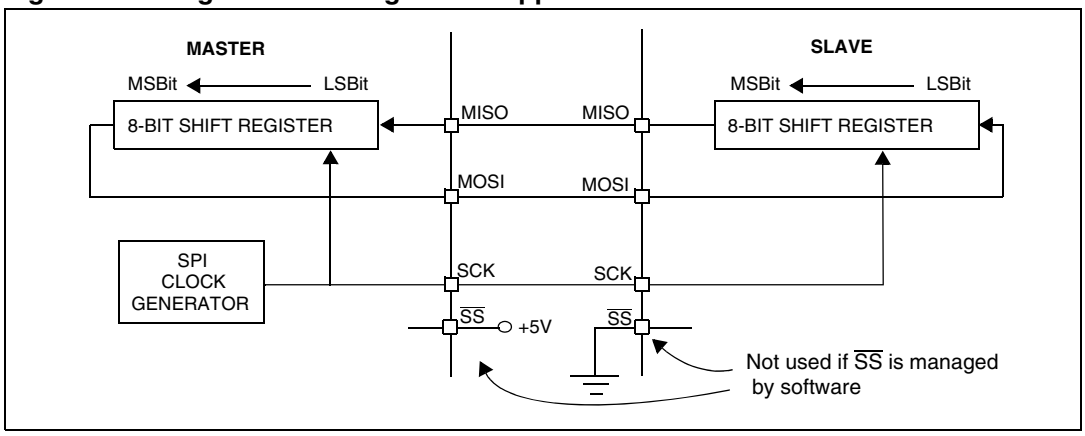
13.5 Interrupts

Table 53. TIM8 interrupt control and wake-up capability

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 8-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).

Figure 71. Single master/ single slave application



14.3.2 Slave select management

As an alternative to using the $\overline{SS}$ pin to control the Slave Select signal, the application can choose to manage the Slave Select signal by software. This is configured by the SSM bit in the SPICSR register (see *Figure 73*).

In software management, the external $\overline{SS}$ pin is free for other application uses and the internal $\overline{SS}$ signal level is driven by writing to the SSI bit in the SPICSR register.

In Master mode:

- $\overline{SS}$ internal must be held high continuously

In Slave Mode:

There are two cases depending on the data/clock timing relationship (see *Figure 72*):

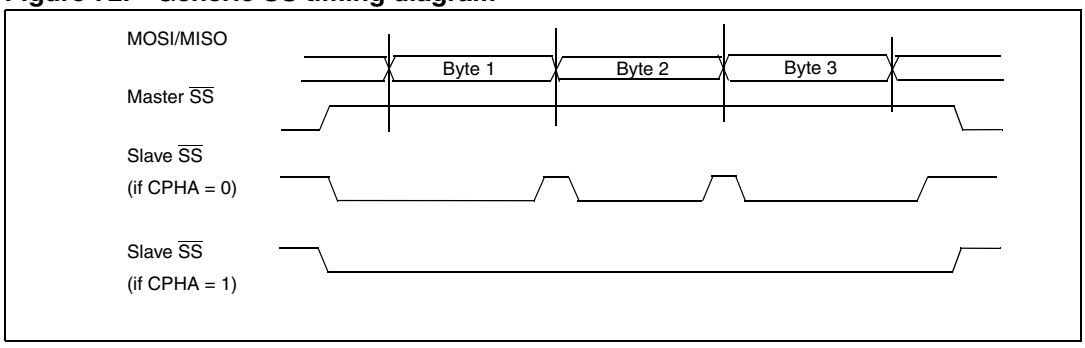
If CPHA = 1 (data latched on second clock edge):

- $\overline{SS}$ internal must be held low during the entire transmission. This implies that in single slave applications the $\overline{SS}$ pin either can be tied to V_{SS}, or made free for standard I/O by managing the $\overline{SS}$ function by software (SSM = 1 and SSI = 0 in the in the SPICSR register)

If CPHA = 0 (data latched on first clock edge):

- $\overline{SS}$ internal must be held low during byte transmission and pulled high between each byte to allow the slave to write to the shift register. If $\overline{SS}$ is not pulled high, a Write Collision error will occur when the slave writes to the shift register (see *Write collision error (WCOL)*).

Figure 72. Generic $\overline{SS}$ timing diagram



15.10.4 Control register 3 (SCICR3)

Read/ write

Reset value: 0000 0000 (00h)

7							0
LDUM	LINE	LSLV	LASE	LHDM	LHIE	LHDF	LSF

Bit 7 = **LDUM** *LIN Divider Update Method.*
This bit is set and cleared by software and is also cleared by hardware (when RDRF = 1). It is only used in LIN Slave mode. It determines how the LIN Divider can be updated by software.

0: LDIV is updated as soon as LPR is written (if no auto synchronization update occurs at the same time).
1: LDIV is updated at the next received character (when RDRF = 1) after a write to the LPR register

Note: If no write to LPR is performed between the setting of LDUM bit and the reception of the next character, LDIV will be updated with the old value.

After LDUM has been set, it is possible to reset the LDUM bit by software. In this case, LDIV can be modified by writing into LPR / LPFR registers.

Bits 6:5 = **LINE, LSLV** *LIN Mode Enable Bits.*
These bits configure the LIN mode:

Table 66. LIN mode configuration

LINE	LSLV	Meaning
0	x	LIN mode disabled
1	0	LIN Master Mode
	1	LIN Slave Mode

The LIN master configuration enables:

The capability to send LIN synch breaks (13 low bits) using the SBK bit in the SCICR2 register.

The LIN slave configuration enables:

- The LIN slave baud rate generator. The LIN Divider (LDIV) is then represented by the LPR and LPFR registers. The LPR and LPFR registers are read/write accessible at the address of the SCIBRR register and the address of the SCIETPR register
- Management of LIN headers.
- LIN synch break detection (11-bit dominant).
- LIN wake-up method (see LHDM bit) instead of the normal SCI Wake-Up method.
- Inhibition of break transmission capability (SBK has no effect)
- LIN parity checking (in conjunction with the PCE bit)

17 beCAN controller (beCAN)

The beCAN controller (Basic Enhanced CAN), interfaces the CAN network and supports the CAN protocol version 2.0A and B. It has been designed to manage high number of incoming messages efficiently with a minimum CPU load. It also meets the priority requirements for transmit messages.

17.1 Main features

- Supports CAN protocol version 2.0 A, B Active
- Bit rates up to 1Mbit/s

Transmission

- 2 transmit mailboxes
- Configurable **transmit priority**

Reception

- 1 receive FIFO with three stages
- 6 scalable filter banks
- Identifier list feature
- Configurable FIFO overrun

Management

- Maskable interrupts
- Software-efficient mailbox mapping at a unique address space

17.2 General description

In today's CAN applications, the number of nodes in a network is increasing and often several networks are linked together via gateways. Typically the number of messages in the system (and thus to be handled by each node) has significantly increased. In addition to the application messages, network management and diagnostic messages have been introduced.

- An enhanced filtering mechanism is required to handle each type of message.

  Furthermore, application tasks require more CPU time, therefore real-time constraints caused by message reception have to be reduced.
- A receive FIFO scheme allows the CPU to be dedicated to application tasks for a long time period without losing messages.

  The standard HLP (Higher Layer Protocol) based on standard CAN drivers requires an efficient interface to the CAN controller.
- All mailboxes and registers are organized in 16-byte pages mapped at the same address and selected via a page select register.

Normal mode. Before entering Normal mode beCAN always has to synchronize on the CAN bus. To synchronize, beCAN waits until the CAN bus is idle, this means 11 consecutive recessive bits have been monitored on CANRX.

Initialization mode

The software initialization can be done while the hardware is in Initialization mode. To enter this mode the software sets the INRQ bit in the CMCR register and waits until the hardware has confirmed the request by setting the INAK bit in the CMSR register.

To leave initialization mode, the software clears the INQR bit. beCAN has left initialization mode once the INAK bit has been cleared by hardware.

While in Initialization mode, all message transfers to and from the CAN bus are stopped and the status of the CAN bus output CANTX is recessive (high).

Entering initialization mode does not change any of the configuration registers.

To initialize the CAN controller, software has to set up the bit timing registers and the filter banks. If a filter bank is not used, it is recommended to leave it non active (leave the corresponding FACT bit cleared).

Normal mode

Once the initialization has been done, the software must request the hardware to enter Normal mode, to synchronize on the CAN bus and start reception and transmission. Entering normal mode is done by clearing the INRQ bit in the CMCR register and waiting until the hardware has confirmed the request by clearing the INAK bit in the CMSR register. Afterwards, the beCAN synchronizes with the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (≡ bus idle) before it can take part in bus activities and start message transfer.

The initialization of the filter values is independent from Initialization mode but must be done while the filter bank is not active (corresponding FACTx bit cleared). The filter bank scale and mode configuration must be configured in initialization mode.

Low power mode (Sleep)

To reduce power consumption, beCAN has a low power mode called Sleep mode. This mode is entered on software request by setting the SLEEP bit in the CMCR register. In this mode, the beCAN clock is stopped. Consequently, software can still access the beCAN registers and mailboxes but the beCAN will not update the status bits.

Example 1: If software requests entry to initialization mode by setting the INRQ bit while beCAN is in sleep mode, it will not be acknowledged by the hardware, INAK stays cleared.

beCAN can be woken up (exit Sleep mode) either by software clearing the SLEEP bit or on detection of CAN bus activity.

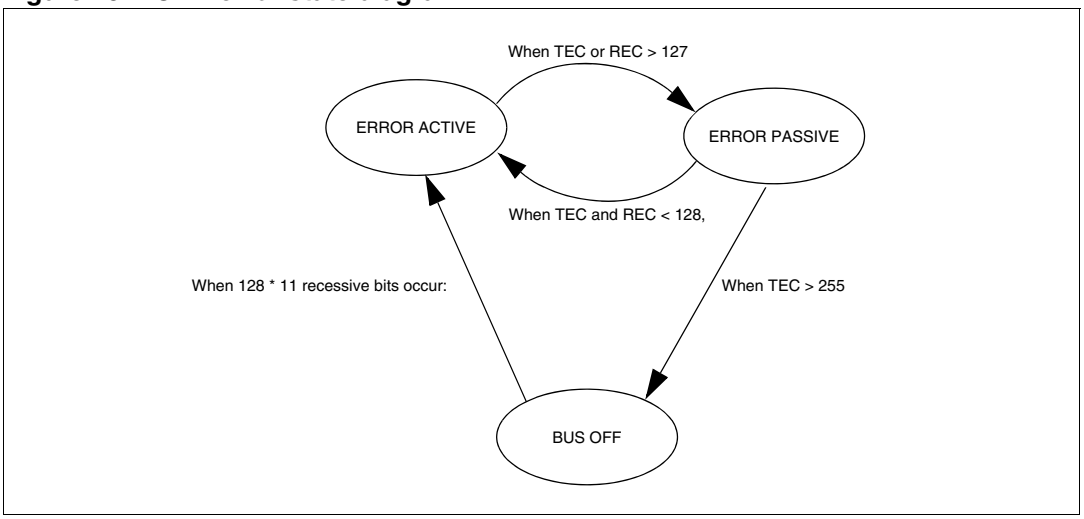
On CAN bus activity detection, hardware automatically performs the wake-up sequence by clearing the SLEEP bit if the AWUM bit in the CMCR register is set. If the AWUM bit is cleared, software has to clear the SLEEP bit when a wake-up interrupt occurs, in order to exit from sleep mode.

Note: If the wake-up interrupt is enabled (WKUIE bit set in CIER register) a wake-up interrupt will be generated on detection of CAN bus activity, even if the beCAN automatically performs the wake-up sequence.

Table 83. Receive mailbox mapping (continued)

Offset to receive mailbox base address (bytes)	Register Name
10	MDAR4
11	MDAR5
12	MDAR6
13	MDAR7
14	Reserved
15	Reserved

Figure 104. CAN error state diagram



17.4.5 Error management

The error management as described in the CAN protocol is handled entirely by hardware using a Transmit Error Counter (TECR register) and a Receive Error Counter (RECR register), which get incremented or decremented according to the error condition. For detailed information about TEC and REC management, please refer to the CAN standard.

Both of them may be read by software to determine the stability of the network. Furthermore, the CAN hardware provides detailed information on the current error status in CESR register. By means of CEIER register and ERRIE bit in CIER register, the software can configure the interrupt generation on error detection in a very flexible way.

Bus-off recovery

The Bus-Off state is reached when TECR is greater then 255, this state is indicated by BOFF bit in CESR register. In Bus-Off state, the beCAN acts as disconnected from the CAN bus, hence it is no longer able to transmit and receive messages.

Depending on the ABOM bit in the CMCR register beCAN will recover from Bus-Off (become error active again) either automatically or on software request. But in both cases the beCAN has to wait at least for the recovery sequence specified in the CAN standard (128 x 11 consecutive recessive bits monitored on CANRX).

critical period. But the application may lose additional time waiting in the while loop as we are no longer able to guarantee a maximum of 6 CAN bit times spent in the workaround.

In this particular case the time the application can spend in the workaround may increase up to a full CAN frame, depending of the frame contents. This case is very rare but happens when a specific sequence is present on in the CAN frame.

The example in *Figure 113* shows reception at maximum CAN baud rate: In this case t_{CAN} is $8/f_{CPU}$ and the sampling time is $10/f_{CPU}$.

If the application is using the maximum baud rate and the possible delay caused by the workaround is not acceptable, there is another workaround which reduces the Rx pin sampling time.

Workaround 2 (see *Figure 114*) first tests that FMP = 2 and the CAN cell is receiving, if not the FIFO can be released immediately. If yes, the program goes through a sequence of test instructions on the RX pin that last longer than the time between the acknowledge dominant bit and the critical time slot. If the Rx pin is in recessive state for more than 8 CAN bit times, it means we are now after the acknowledge and the critical slot. If a dominant bit is read on the bus, we can release the FIFO immediately. This workaround has to be written in assembly language to avoid the compiler optimizing the test sequence.

The implementation shown here is for the CAN bus maximum speed (1 Mbaud @ 8 MHz CPU clock).

Figure 113. Reception at maximum CAN baud rate

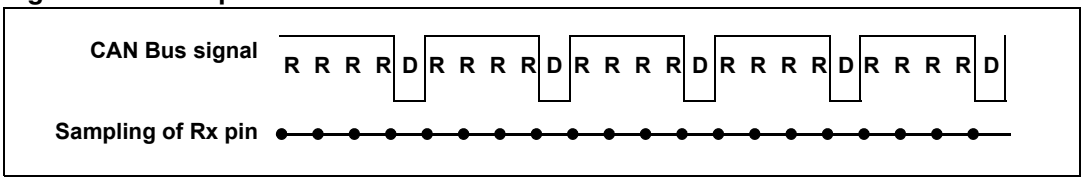


Figure 114. Workaround 2

```
Ld          a, CRFR
And         a,#3
Cp          a,#2            ; test FMP=2 ?
Jrne        _RELEASE        ; if not release


Btjf        CMSR,#5,_RELEASE ; test if reception on going.
                             ; if not release

Btjf        CDGR,#3,_RELEASE ; sample RX pin for 8 CAN bit time
Btjf        CDGR,#3,_RELEASE
Btjf        CDGR,#3,_RELEASE
btjf        CDGR,#3,_RELEASE
btjf        CDGR,#3,_RELEASE
btjf        CDGR,#3,_RELEASE
btjf        CDGR,#3,_RELEASE
btjf        CDGR,#3,_RELEASE
btjf        CDGR,#3,_RELEASE
btjf        CDGR,#3,_RELEASE
btjf        CDGR,#3,_RELEASE
btjf        CDGR,#3,_RELEASE
btjf        CDGR,#3,_RELEASE
btjf        CDGR,#3,_RELEASE

_RELEASE:
bset        CRFR,#5
```

MIDR1

7							0
STID5	STID4	STID3	STID2	STID1	STID0	EXID17	EXID16

Bits 7:2 = **STID[5:0]** *Standard Identifier*
6 least significant bits of the standard part of the identifier.

Bits 1:0 = **EXID[17:16]** *Extended Identifier*
2 most significant bits of the extended part of the identifier.

MIDR2

7							0
EXID15	EXID14	EXID13	EXID12	EXID11	EXID10	EXID9	EXID8

Bits 7:0 = **EXID[15:8]** *Extended Identifier*

Bits 15 to 8 of the extended part of the identifier.

MIDR3

7							0
EXID7	EXID6	EXID5	EXID4	EXID3	EXID2	EXID1	EXID0

Bits 7:1 = **EXID[6:0]** *Extended Identifier*
6 least significant bits of the extended part of the identifier.

Mailbox data length control register (MDLC)

Read / write

Reset value: xxxx xxxx (xxh)

7							0
0	0	0	0	DLC3	DLC2	DLC1	DLC0

Note: All bits of this register is write protected when the mailbox is not in empty state.

Bit 7 = Reserved, must be kept cleared.

Bits 6:4 = Reserved, forced to 0 by hardware.

Bits 3:0 = **DLC[3:0]** *Data Length Code*
This field defines the number of data bytes a data frame contains or a remote frame request.

Mailbox data registers (MDAR[7:0])

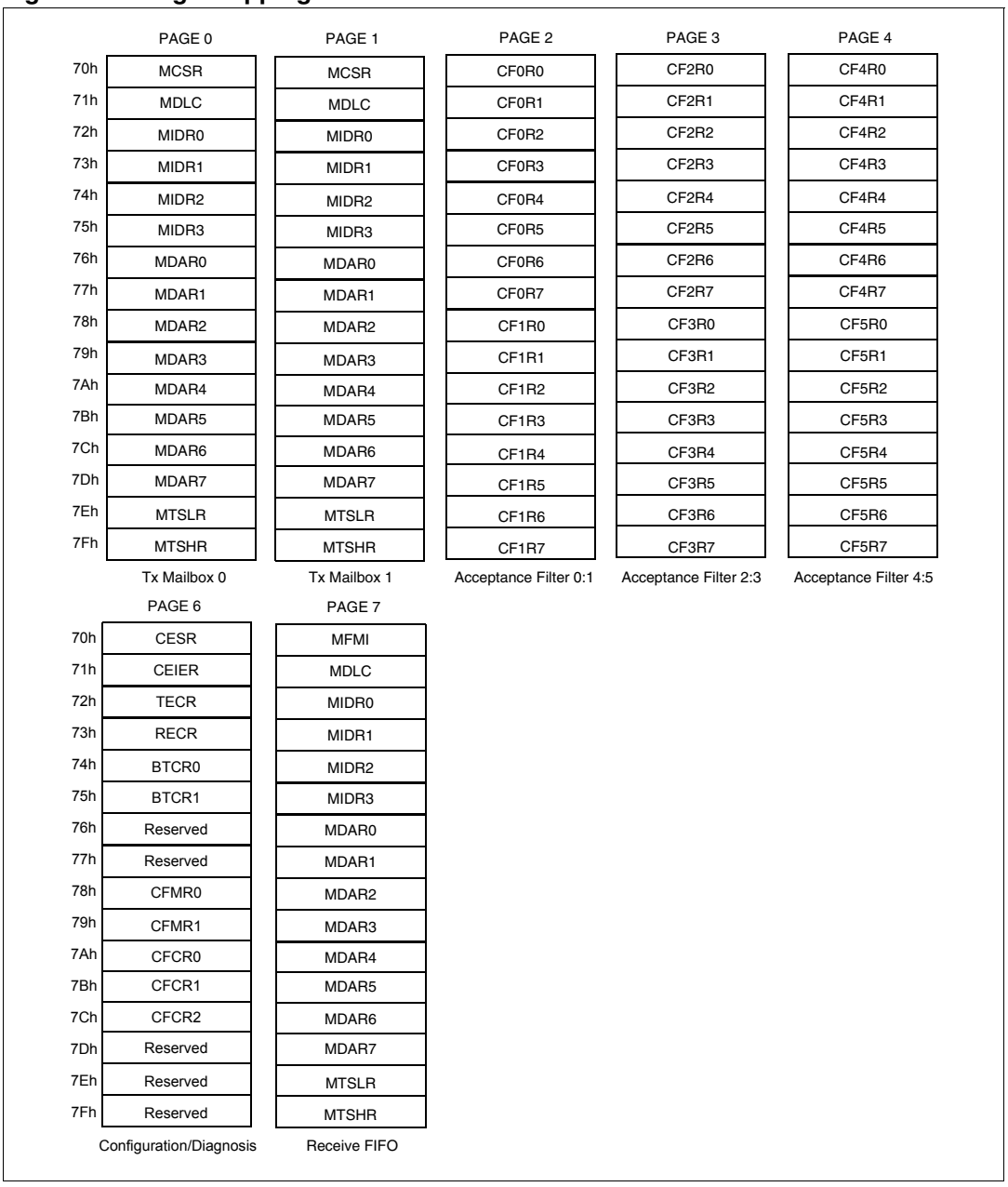
Read / Write

Reset value: Undefined

7							0
DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0

Note: All bits of this register are write protected when the mailbox is not in empty state.

Figure 116. Page mapping for CAN



20.5.1 Crystal and ceramic resonator oscillators

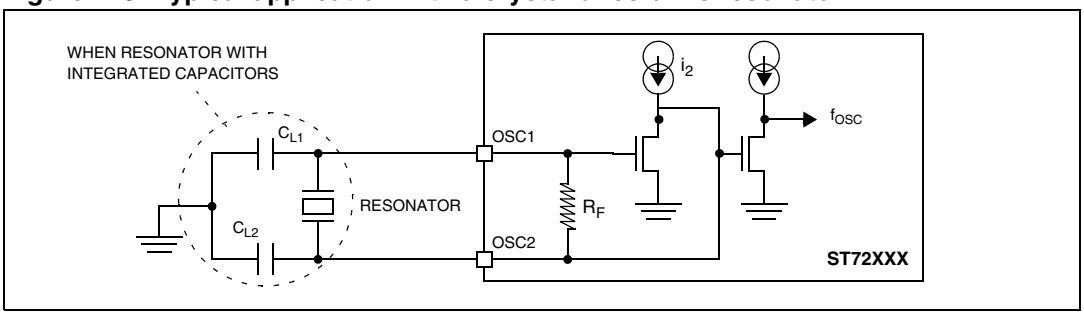
The ST7 internal clock can be supplied with four different crystal/ ceramic resonator oscillators. All the information given in this paragraph is based on characterization results with specified typical external components. In the application, the resonator and the load capacitors have to be placed as close as possible to the oscillator pins in order to minimize output distortion and start-up stabilization time. Refer to the crystal/ceramic resonator manufacturer for more details (frequency, package, accuracy...)[(a)(b)].

Table 104. Oscillator characteristics

Symbol	Parameter	Conditions	Min	Max	Unit
f_{OSC}	Oscillator Frequency[(1)]	LP: Low power oscillator MP: Medium power oscillator MS: Medium speed oscillator HS: High speed oscillator	1 >2 >4 >8	2 4 8 16	MHz
R_F	Feedback resistor		20	40	kΩ
C_{L1} C_{L2}	Recommended load capacitance versus equivalent serial resistance of the crystal or ceramic resonator (R_S)	R_S = 200Ω LP oscillator R_S = 200Ω MP oscillator R_S = 200Ω MS oscillator R_S = 100Ω HS oscillator	22 22 18 15	56 46 33 33	pF
i_2	OSC2 driving current	V_{DD} = 5V LP oscillator V_{IN} = V_{SS} MP oscillator MS oscillator HS oscillator	80 160 310 610	150 250 460 910	µA

1. The oscillator selection can be optimized in terms of supply current using an high quality resonator with small R_S value. Refer to crystal/ceramic resonator manufacturer for more details.

Figure 123. Typical application with a crystal or ceramic resonator



a. Resonator characteristics given by the crystal/ceramic resonator manufacturer.

b. $t_{SU(OSC)}$ is the typical oscillator start-up time measured between V_{DD} = 2.8V and the fetch of the first instruction (with a quick V_{DD} ramp-up from 0 to 5V (< 50µs).

20.5.2 PLL characteristics

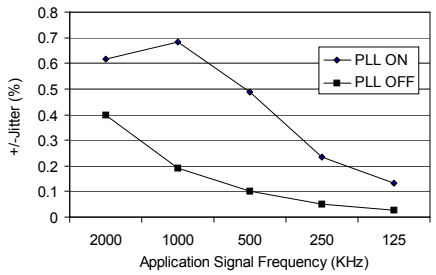
Operating conditions: V_{DD} 3.8 to 5.5V @ T_A 0 to 70°C[a] or V_{DD} 4.5 to 5.5V @ T_A -40 to 125°C

Table 105. PLL characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$V_{DD(PLL)}$	PLL Voltage Range	T_A = 0 to +70°C	3.8		5.5	
		T_A = -40 to +125°C	4.5			
f_{OSC}	PLL input frequency range		2		4	MHz
$\Delta f_{CPU}/f_{CPU}$	PLL jitter(1)	f_{OSC} = 4 MHz, V_{DD} = 4.5 to 5.5V		See note(2)		%
		f_{OSC} = 2 MHz, V_{DD} = 4.5 to 5.5V				

1. Data characterized but not tested.
2. Under characterization.

Figure 124. PLL jitter vs signal frequency(1)



1. Measurement conditions: f_{CPU} = 4 MHz, T_A = 25°C

The user must take the PLL jitter into account in the application (for example in serial communication or sampling of high frequency signals). The PLL jitter is a periodic effect, which is integrated over several CPU cycles. Therefore, the longer the period of the application signal, the less it is impacted by the PLL jitter.

Figure 124 shows the PLL jitter integrated on application signals in the range 125 kHz to 2 MHz. At frequencies of less than 125 kHz, the jitter is negligible.

a. Data characterized but not tested

22.3 Transfer of customer code

Customer code is made up of the ROM/FASTROM contents and the list of the selected options (if any). The ROM/FASTROM contents are to be sent on diskette, or by electronic means, with the S19 hexadecimal file generated by the development tool. All unused bytes must be set to FFh.

The selected options are communicated to STMicroelectronics using the correctly completed OPTION LIST appended.

Refer to application note AN1635 for information on the counter listing returned by ST after code has been transferred.

The STMicroelectronics sales organization is pleased to provide detailed information on contractual points.

The following *Figure 153* serves as a guide for ordering.

Figure 153. ST72P561xxx-Auto FastROM commercial product structure