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, I²C, SCI, 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	Internal
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/st72f521ar9tc

22.1.1 Flash configuration 257
22.1.2 Flash ordering information 260
22.2 ROM device ordering information and transfer of customer code 261
22.3 Development tools 265
22.3.1 Introduction 265
22.3.2 Evaluation tools and starter kits 265
22.3.3 Development and debugging tools 265
22.3.4 Programming tools 265
22.3.5 Socket and emulator adapter information 266

23 Known limitations 267
23.1 All Flash and ROM devices 267
23.1.1 External RC option 267
23.1.2 Safe connection of OSC1/OSC2 pins 267
23.1.3 Reset pin protection with LVD enabled 267
23.1.4 Unexpected reset fetch 267
23.1.5 External interrupt missed 267
23.1.6 Clearing active interrupts outside interrupt routine 271
23.1.7 SCI wrong break duration 272
23.1.8 16-bit timer PWM mode 272
23.1.9 TIMD set simultaneously with OC interrupt 273
23.1.10 CAN cell limitations 273
23.1.11 I^2C multimaster 273
23.2 All Flash devices 274
23.2.1 Internal RC oscillator with LVD 274
23.2.2 I/O behavior during ICC mode entry sequence 274
23.2.3 Readout protection with LVD 274

24 Revision history 275

Software workaround - devices with hardware fix (ST72F521 rev “R”)

To implement a transmission abort under safe conditions, the LOCK bit must not be reset during the critical window (2 bit times). A new function has been implemented in the MCU allowing the application to synchronize the reset of the LOCK bit (abort request) with the reset of the TXRQST bit (internal signal) in the pCAN core.

The synchronization is done using the WKPS bit in the CANCSR register, the function of this bit has been modified and no more Wake-up Pulse (dominant bit) is sent on the CAN_TX signal when the WKPS bit is set. This means the functionality described in the datasheet is no longer applicable (see *Section 17.5.4: WKPS functionality*).

To abort the transmission, the application first sets the WKPS bit and polls it until it is set. The maximum time needed to set this bit is two CAN bit times. Once the application has read the WKPS bit as one, it can reset the LOCK bit to stop the current transmission.

The abort is completed when the LOCK bit is read back as zero by the application. Once the abort has been completed, the application must reset the WKPS bit to be able to transmit again. Of course the transmit buffer must be in LOCK state as usual before any transmission attempt.

The “C” code sequence below shows the software workaround using the WKPS bit.

```
CANCSR |= WKPS;// Set WKPS bit
while(!(CANCSR & WKPS) );// Wait until WKPS bit is set
while( CANBCSR & LOCK )// Wait until abort has been confirmed
{
  CANBCSR &= ~LOCK;
}
CANCSR &= ~WKPS;     // Allow transmission again
CANBCSR |= LOCK;         //Alloc buffer for next transmission
```

Software workaround - devices without hardware fix

To implement a transmission abort under safe conditions, any reset of the LOCK bit during the critical window (2 bit times) must be avoided. Two different cases have to be considered, either the pCAN enters standby mode after the abort, or the abort is performed and pCAN keeps running.

Abort followed by Standby mode (RUN = 0)

In this case, aborting the pending transmissions can safely be done by first entering Standby mode and then releasing the transmit buffers. Standby mode is entered by resetting the RUN bit in the CSR register and once the current transmission attempt, even if it fails due to error or lost arbitration, has been performed, pCAN enters Standby mode (RUN = 0). Once in Standby mode the application can abort all pending transmissions by resetting the corresponding LOCK bit.

Abort while staying in RUN mode (RUN = 1)

Contrary to the STANDBY case described previously, in the RUN case the application has to handle the error or arbitration lost conditions. In case of transmission errors, causing the frame to be transmitted again and again, the application must set the NRTX bit in the CSR register. This will cause pCAN to abort the transmission at the end of the current attempt.

In case of arbitration lost, setting the NRTX bit does not abort the transmission, therefore the application must reset the LOCK bit to abort the transmission. To avoid resetting the LOCK bit during the critical time window, leading to the problem described at the start of this

20.3.4 External voltage detector (EVD) thresholds

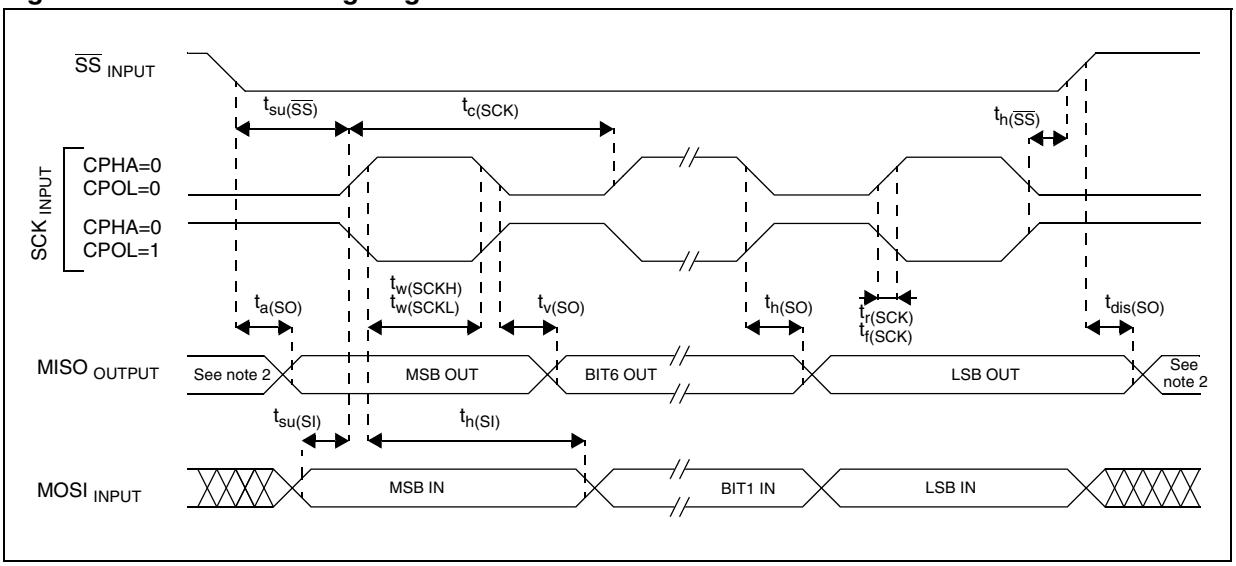
Subject to general operating conditions for V_{DD}, f_{CPU}, and T_A.

Table 130. External voltage detector (EVD) thresholds

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$V_{IT+(EVD)}$	1⇒0 AVDF flag toggle threshold (V_{DD} rise[1]		1.15	1.26	1.35	V
$V_{IT-(EVD)}$	0⇒1 AVDF flag toggle threshold (V_{DD} fall)[1]		1.1	1.2	1.3	
$V_{hys(EVD)}$	EVD voltage threshold hysteresis	$V_{IT+(EVD)}$-$V_{IT-(EVD)}$		200		mV

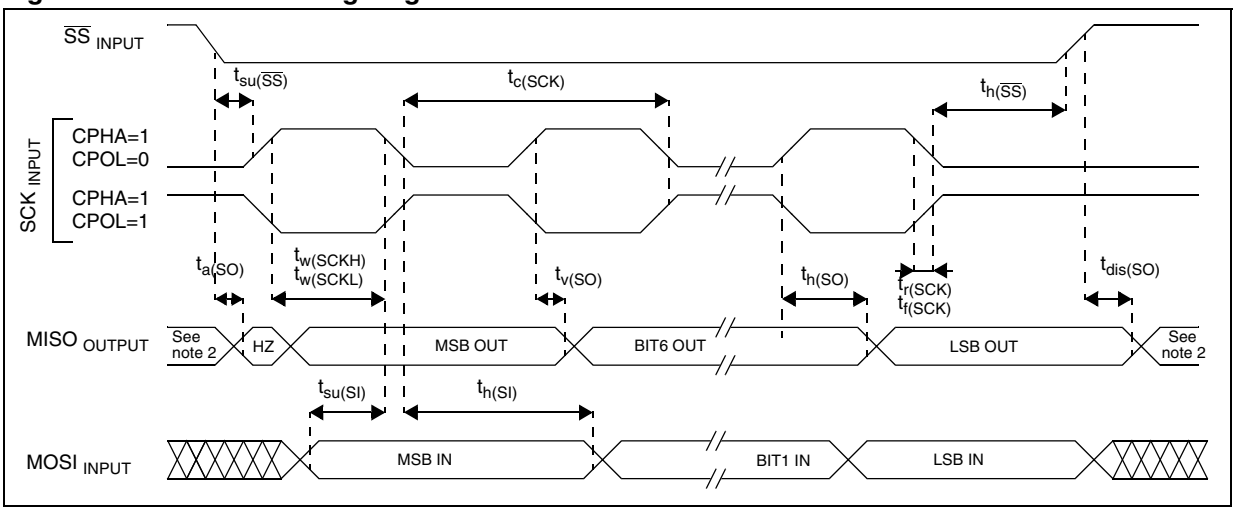
1. Data based on characterization results, not tested in production

Figure 108. SPI slave timing diagram with CPHA = 0(1)



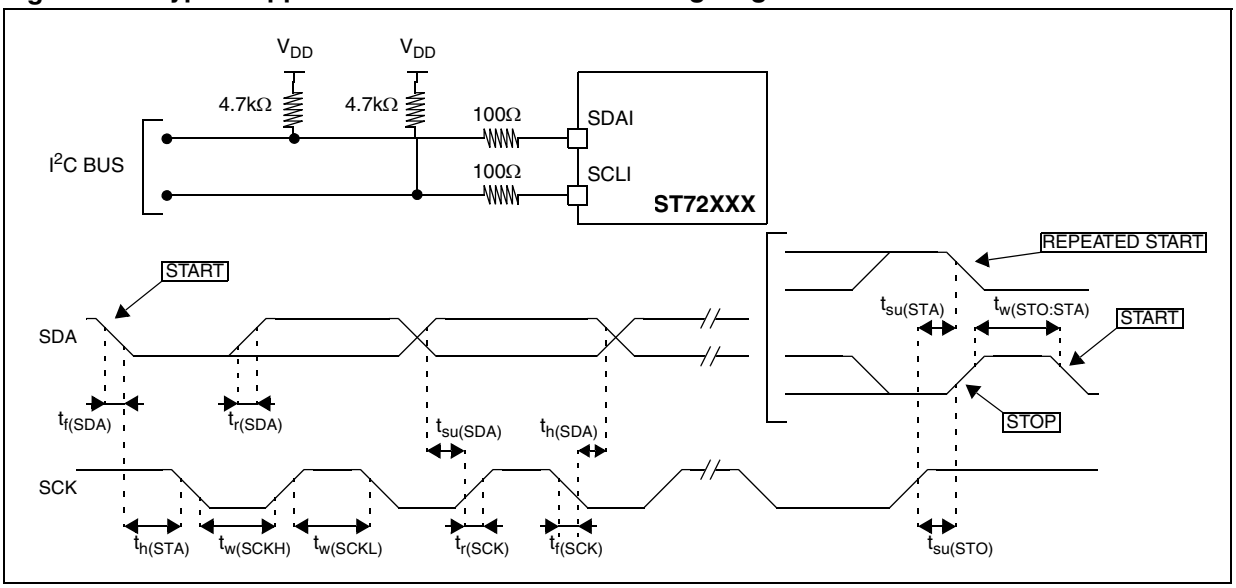
1. Measurement points are done at CMOS levels: $0.3xV_{DD}$ and $0.7xV_{DD}$.
2. When no communication is on-going the data output line of the SPI (MOSI in master mode, MISO in slave mode) has its alternate function capability released. In this case, the pin status depends on the I/O port configuration.

Figure 109. SPI slave timing diagram with CPHA = 1(1)



1. Measurement points are done at CMOS levels: $0.3xV_{DD}$ and $0.7xV_{DD}$.
2. When no communication is on-going the data output line of the SPI (MOSI in master mode, MISO in slave mode) has its alternate function capability released. In this case, the pin status depends of the I/O port configuration.

Figure 111. Typical application with I²C BUS and timing diagram(1)



1. Measurement points are done at CMOS levels: $0.3xV_{DD}$ and $0.7xV_{DD}$.

The following table provides the values to be written in the I2CCCR register to obtain the required I²C SCL line frequency.

Table 154. SCL frequency table

f_{SCL} (kHz)	I2CCCR value							
	f_{CPU} = 4 MHz				f_{CPU} = 8 MHz			
	V_{DD} = 4.1V		V_{DD} = 5V		V_{DD} = 4.1V		V_{DD} = 5V	
	R_P = 3.3kΩ	R_P = 4.7kΩ	R_P = 3.3kΩ	R_P = 4.7kΩ	R_P = 3.3kΩ	R_P = 4.7kΩ	R_P = 3.3kΩ	R_P = 4.7kΩ
400	Not achievable				83h			
300	Not achievable				85h			
200	83h				8Ah	89h	8Ah	
100	10h				24h	23h	24h	23h
50	24h				4Ch			
20	5Fh				FFh			

Legend:

R_P = External pull-up resistance

f_{SCL} = I²C speed

Note: - For speeds around 200 kHz, the achieved speed can have a ±5% tolerance.
- For other speed ranges, the achieved speed can have a ±2% tolerance.

The above variations depend on the accuracy of the external components used.

```
; 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
```