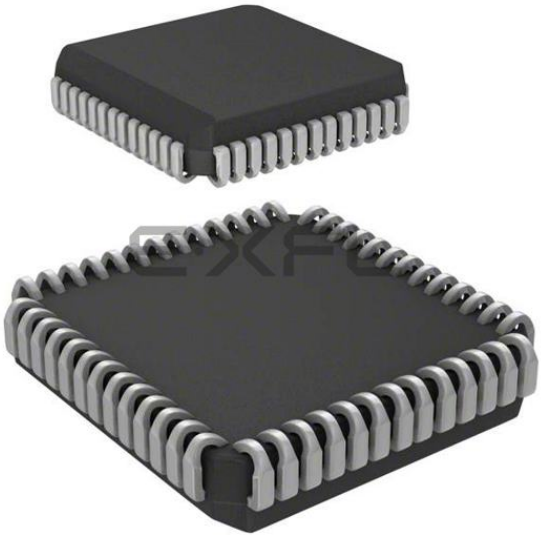


Welcome to E-XFL.COM

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	80C51
Core Size	8-Bit
Speed	40MHz
Connectivity	CANbus, SPI, UART/USART
Peripherals	POR, PWM, WDT
Number of I/O	36
Program Memory Size	64KB (64K x 8)
Program Memory Type	FLASH
EEPROM Size	2K x 8
RAM Size	2.25K x 8
Voltage - Supply (Vcc/Vdd)	3V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	External
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	52-LCC (J-Lead)
Supplier Device Package	52-PLCC (19.15x19.15)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/at89c51cc03c-s3rim

Table 3. CKCON1 Register

CKCON1 (S:9Fh)
Clock Control Register 1

7	6	5	4	3	2	1	0
							SPIX2

Bit Number	Bit Mnemonic	Description
7-1	-	**Reserved** The value read from these bits is indeterminate. Do not set these bits.
0	SPIX2	**SPI clock** (1) Clear to select 6 clock periods per peripheral clock cycle. Set to select 12 clock periods per peripheral clock cycle.

Note: 1. This control bit is validated when the CPU clock bit X2 is set; when X2 is low, this bit has no effect.

Reset Value = 0000 0000b

Examples

```
;*F*****************************************************************************;* NAME: api_rd_eeprom_byte
;* DPTR contain address to read.
;* Acc contain the reading value
;* NOTE: before execute this function, be sure the EEPROM is not BUSY
;*****************************************************************************
api_rd_eeprom_byte:
 MOV   EECON, #02h; map EEPROM in XRAM space
 MOVX A, @DPTR
 MOV   EECON, #00h; unmap EEPROM
ret

;*F*****************************************************************************
;* NAME: api_ld_eeprom_cl
;* DPTR contain address to load
;* Acc contain value to load
;* NOTE: in this example we load only 1 byte, but it is possible upto
;* 128 Bytes.
;* before execute this function, be sure the EEPROM is not BUSY
;*****************************************************************************
api_ld_eeprom_cl:
 MOV   EECON, #02h ; map EEPROM in XRAM space
 MOVX @DPTR, A
 MOVEECON, #00h; unmap EEPROM
ret

;*F*****************************************************************************
;* NAME: api_wr_eeprom
;* NOTE: before execute this function, be sure the EEPROM is not BUSY
;*****************************************************************************
api_wr_eeprom:
 MOV   EECON, #050h
 MOV   EECON, #0A0h
ret
```

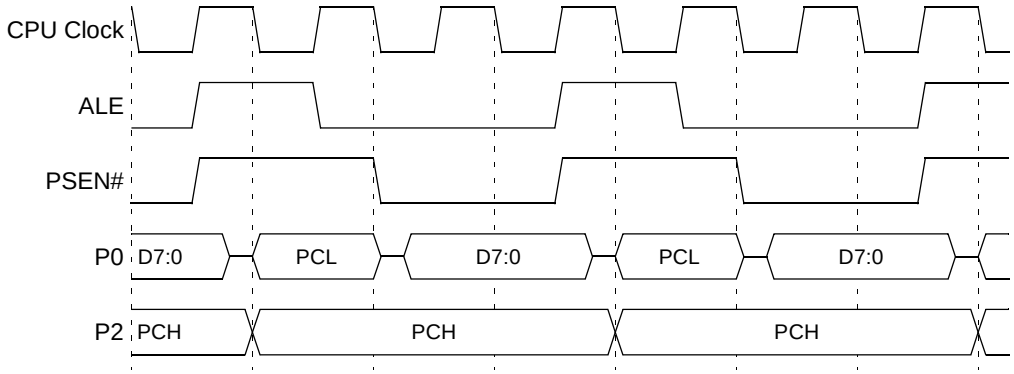
Program/Code Memory

The AT89C51CC03 implement 64K Bytes of on-chip program/code memory. Figure 20 shows the partitioning of internal and external program/code memory spaces depending on the product.

The Flash memory increases EPROM and ROM functionality by in-circuit electrical erasure and programming. Thanks to the internal charge pump, the high voltage needed for programming or erasing Flash cells is generated on-chip using the standard VDD voltage. Thus, the Flash Memory can be programmed using only one voltage and allows In-System Programming commonly known as ISP. Hardware programming mode is also available using specific programming tool.

Figure 20. Program/Code Memory Organization

Figure 22. External Code Fetch Waveforms



Flash Memory Architecture

AT89C51CC03 features two on-chip Flash memories:

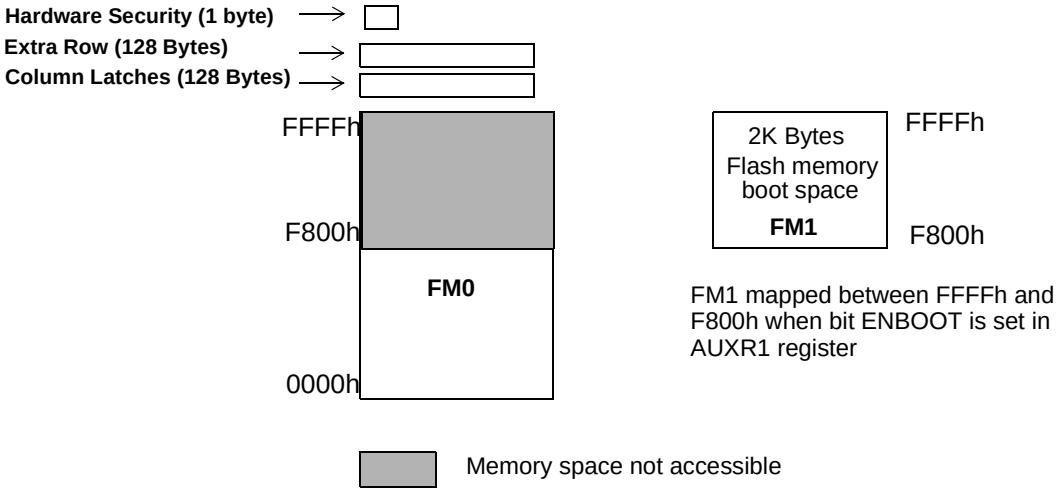
- Flash memory FM0:
  containing 64K Bytes of program memory (user space) organized into 128 byte pages,
- Flash memory FM1:
  2K Bytes for boot loader and Application Programming Interfaces (API).

The FM0 can be program by both parallel programming and Serial In-System Programming (ISP) whereas FM1 supports only parallel programming by programmers. The ISP mode is detailed in the "In-System Programming" section.

All Read/Write access operations on Flash Memory by user application are managed by a set of API described in the "In-System Programming" section.

The bit ENBOOT in AUXR1 register is used to map FM1 from F800h to FFFFh. Figure 23 and Figure 24 show the Flash memory configuration with ENBOOT=1 and ENBOOT=0.

Figure 23. Flash Memory Architecture with ENBOOT=1 (boot mode)



Given Address

Each device has an individual address that is specified in the SADDR register; the SADEN register is a mask byte that contains don't-care bits (defined by zeros) to form the device's given address. The don't-care bits provide the flexibility to address one or more slaves at a time. The following example illustrates how a given address is formed. To address a device by its individual address, the SADEN mask byte must be `1111 1111b`.

For example:

SADDR0101 0110b
SADEN1111 1100b

Given0101 01XXb

Here is an example of how to use given addresses to address different slaves:

Slave A:SADDR1111 0001b
SADEN1111 1010b

Given1111 0X0Xb

Slave B:SADDR1111 0011b
SADEN1111 1001b

Given1111 0XX1b

Slave C:SADDR1111 0011b
SADEN1111 1101b

Given1111 00X1b

The SADEN byte is selected so that each slave may be addressed separately.
For slave A, bit 0 (the LSB) is a don't-care bit; for slaves B and C, bit 0 is a 1. To communicate with slave A only, the master must send an address where bit 0 is clear (e.g. `1111 0000b`).
For slave A, bit 1 is a 0; for slaves B and C, bit 1 is a don't care bit. To communicate with slaves A and B, but not slave C, the master must send an address with bits 0 and 1 both set (e.g. `1111 0011b`).
To communicate with slaves A, B and C, the master must send an address with bit 0 set, bit 1 clear, and bit 2 clear (e.g. `1111 0001b`).

Broadcast Address

A broadcast address is formed from the logical OR of the SADDR and SADEN registers with zeros defined as don't-care bits, e.g.:

SADDR0101 0110b
SADEN1111 1100b

SADDR OR SADEN1111 111Xb

The use of don't-care bits provides flexibility in defining the broadcast address, however in most applications, a broadcast address is FFh. The following is an example of using broadcast addresses:

Slave A:SADDR1111 0001b
SADEN1111 1010b

Given1111 1X11b,

Slave B:SADDR1111 0011b
SADEN1111 1001b

Given1111 1X11B,

Slave C:SADDR=1111 0010b
SADEN1111 1101b

Given1111 1111b

Timers/Counters

The AT89C51CC03 implements two general-purpose, 16-bit Timers/Counters. Such are identified as Timer 0 and Timer 1, and can be independently configured to operate in a variety of modes as a Timer or an event Counter. When operating as a Timer, the Timer/Counter runs for a programmed length of time, then issues an interrupt request. When operating as a Counter, the Timer/Counter counts negative transitions on an external pin. After a preset number of counts, the Counter issues an interrupt request. The various operating modes of each Timer/Counter are described in the following sections.

Timer/Counter Operations

A basic operation is Timer registers THx and TLx (x = 0, 1) connected in cascade to form a 16-bit Timer. Setting the run control bit (TRx) in TCON register (see Figure 30) turns the Timer on by allowing the selected input to increment TLx. When TLx overflows it increments THx; when THx overflows it sets the Timer overflow flag (TFx) in TCON register. Setting the TRx does not clear the THx and TLx Timer registers. Timer registers can be accessed to obtain the current count or to enter preset values. They can be read at any time but TRx bit must be cleared to preset their values, otherwise the behavior of the Timer/Counter is unpredictable.

The C/Tx# control bit selects Timer operation or Counter operation by selecting the divided-down peripheral clock or external pin Tx as the source for the counted signal. TRx bit must be cleared when changing the mode of operation, otherwise the behavior of the Timer/Counter is unpredictable.

For Timer operation (C/Tx# = 0), the Timer register counts the divided-down peripheral clock. The Timer register is incremented once every peripheral cycle (6 peripheral clock periods). The Timer clock rate is $F_{PER}/6$, i.e. $F_{OSC}/12$ in standard mode or $F_{OSC}/6$ in X2 mode.

For Counter operation (C/Tx# = 1), the Timer register counts the negative transitions on the Tx external input pin. The external input is sampled every peripheral cycles. When the sample is high in one cycle and low in the next one, the Counter is incremented. Since it takes 2 cycles (12 peripheral clock periods) to recognize a negative transition, the maximum count rate is $F_{PER}/12$, i.e. $F_{OSC}/24$ in standard mode or $F_{OSC}/12$ in X2 mode. There are no restrictions on the duty cycle of the external input signal, but to ensure that a given level is sampled at least once before it changes, it should be held for at least one full peripheral cycle.

Timer 0

Timer 0 functions as either a Timer or event Counter in four modes of operation. Figure 35 to Figure 38 show the logical configuration of each mode.

Timer 0 is controlled by the four lower bits of TMOD register (see Figure 31) and bits 0, 1, 4 and 5 of TCON register (see Figure 30). TMOD register selects the method of Timer gating (GATE0), Timer or Counter operation (T/C0#) and mode of operation (M10 and M00). TCON register provides Timer 0 control functions: overflow flag (TF0), run control bit (TR0), interrupt flag (IE0) and interrupt type control bit (IT0).

For normal Timer operation (GATE0 = 0), setting TR0 allows TL0 to be incremented by the selected input. Setting GATE0 and TR0 allows external pin INT0# to control Timer operation.

Timer 0 overflow (count rolls over from all 1s to all 0s) sets TF0 flag generating an interrupt request.

It is important to stop Timer/Counter before changing mode.

Table 35. TL1 Register

TL1 (S:8Bh)
Timer 1 Low Byte Register

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

Bit Number	Bit Mnemonic	Description
7:0		**Low Byte of Timer 1.**

Reset Value = 0000 0000b

Bit Shortening

If, on the other hand, the transmitter oscillator is faster than the receiver one, the next falling edge used for resynchronization may be too early. So Phase Segment 2 in bit N is shortened in order to adjust the sample point for bit N+1 and the end of the bit time

Synchronization Jump Width

The limit to the amount of lengthening or shortening of the Phase Segments is set by the Resynchronization Jump Width.

This segment may not be longer than Phase Segment 2.

Programming the Sample Point

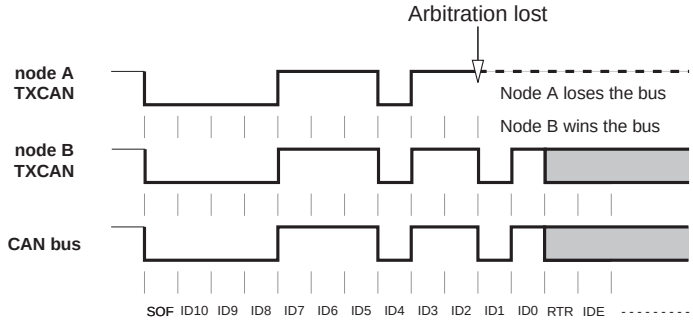
Programming of the sample point allows "tuning" of the characteristics to suit the bus.

Early sampling allows more Time Quanta in the Phase Segment 2 so the Synchronization Jump Width can be programmed to its maximum. This maximum capacity to shorten or lengthen the bit time decreases the sensitivity to node oscillator tolerances, so that lower cost oscillators such as ceramic resonators may be used.

Late sampling allows more Time Quanta in the Propagation Time Segment which allows a poorer bus topology and maximum bus length.

Arbitration

Figure 46. Bus Arbitration



The CAN protocol handles bus accesses according to the concept called “Carrier Sense Multiple Access with Arbitration on Message Priority”.

During transmission, arbitration on the CAN bus can be lost to a competing device with a higher priority CAN Identifier. This arbitration concept avoids collisions of messages whose transmission was started by more than one node simultaneously and makes sure the most important message is sent first without time loss.

The bus access conflict is resolved during the arbitration field mostly over the identifier value. If a data frame and a remote frame with the same identifier are initiated at the same time, the data frame prevails over the remote frame (c.f. RTR bit).

Errors

The CAN protocol signals any errors immediately as they occur. Three error detection mechanisms are implemented at the message level and two at the bit level:

Error at Message Level

- Cyclic Redundancy Check (CRC)
  The CRC safeguards the information in the frame by adding redundant check bits at the transmission end. At the receiver these bits are re-computed and tested against the received bits. If they do not agree there has been a CRC error.
- Frame Check
  This mechanism verifies the structure of the transmitted frame by checking the bit

Acceptance Filter

Upon a reception hit (i.e., a good comparison between the ID+RTR+RB+IDE received and an ID+RTR+RB+IDE specified while taking the comparison mask into account) the ID+RTR+RB+IDE received are written over the ID TAG Registers.

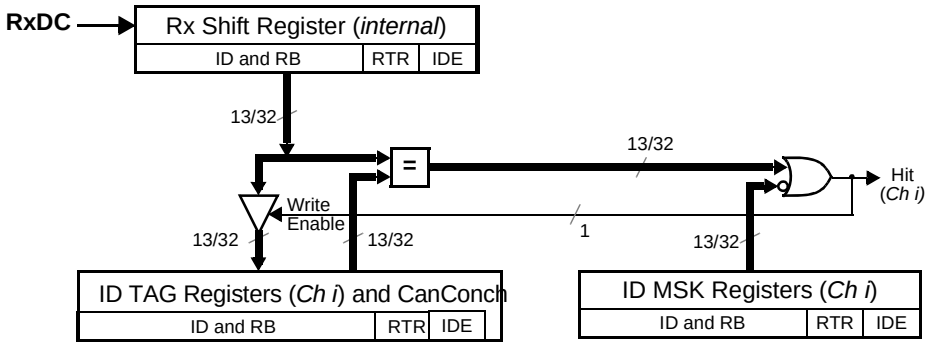
ID => IDT0-29

RTR => RTRTAG

RB => RB0-1TAG

IDE => IDE in CANCONCH register

Figure 54. Acceptance filter block diagram



example:
To accept only ID = 318h in part A.
ID MSK = 111 1111 1111 b
ID TAG = 011 0001 1000 b

CAN SFRs

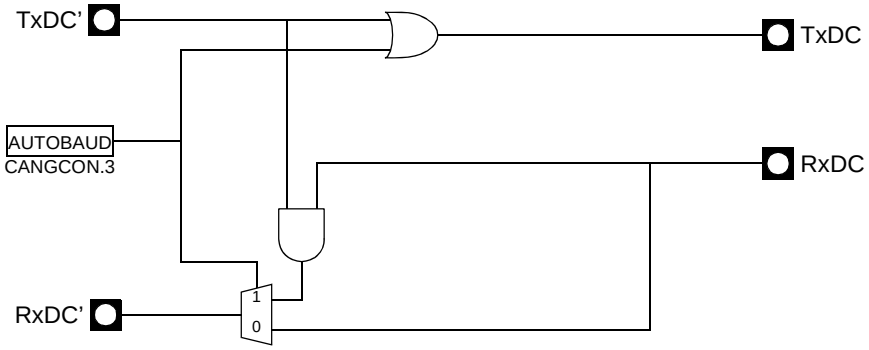
CAN Autobaud and Listening Mode

To activate the Autobaud feature, the AUTOBAUD bit in the CANGCON register must be set. In this mode, the CAN controller is only listening to the line without acknowledging the received messages. It cannot send any message. The error flags are updated. The bit timing can be adjusted until no error occurs (good configuration find).

In this mode, the error counters are frozen.

To go back to the standard mode, the AUTOBAUD bit must be cleared.

Figure 56. Autobaud Mode



Routines Examples

1. Init of CAN macro

```
// Reset the CAN macro
 CANGCON = 01h;
// Disable CAN interrupts
 ECAN   = 0;
 ETIM   = 0;
// Init the Mailbox
 for num_page =0; num_page <15; num_page++
{
   CANPAGE = num_channel << 4;
   CANCONCH = 00h
   CANSTCH = 00h;
   CANIDT1 = 00h;
   CANIDT2 = 00h;
   CANIDT3 = 00h;
   CANIDT4 = 00h;
   CANIDM1 = 00h;
   CANIDM2 = 00h;
   CANIDM3 = 00h;
   CANIDM4 = 00h;
   for num_data =0; num_data <8; num_data++)
    {
    CANMSG = 00h;
        }
}
// Configure the bit timing
 CANBT1 = xxh
 CANBT2 = xxh
 CANBT3 = xxh
```

Table 69. CANIDT4 Register for V2.0 part A

CANIDT4 for V2.0 part A (S:BFh)
CAN Identifier Tag Registers 4

7	6	5	4	3	2	1	0
-	-	-	-	-	RTRTAG	-	RB0TAG

Bit Number	Bit Mnemonic	Description
7-3	-	**Reserved** The values read from these bits are indeterminate. Do not set these bits.
2	RTRTAG	**Remote Transmission Request Tag Value**.
1	-	**Reserved** The values read from this bit are indeterminate. Do not set these bit.
0	RB0TAG	**Reserved Bit 0 Tag Value.**

No default value after reset.

Table 70. CANIDT4 Register for V2.0 part A

CANIDT1 for V2.0 part B (S:BCh)
CAN Identifier Tag Registers 1

7	6	5	4	3	2	1	0
IDT 28	IDT 27	IDT 26	IDT 25	IDT 24	IDT 23	IDT 22	IDT 21

Bit Number	Bit Mnemonic	Description
7-0	IDT28:21	**IDentifier Tag Value** See Figure 54.

No default value after reset.

Table 71. CANIDT2 Register for V2.0 part B

CANIDT2 for V2.0 part B (S:BDh)
CAN Identifier Tag Registers 2

7	6	5	4	3	2	1	0
IDT 20	IDT 19	IDT 18	IDT 17	IDT 16	IDT 15	IDT 14	IDT 13

Bit Number	Bit Mnemonic	Description
7-0	IDT20:13	**IDentifier Tag Value** See Figure 54.

No default value after reset.

Table 79. CANIDM2 Register for V2.0 part B

CANIDM2 for V2.0 part B (S:C5h)
CAN Identifier Mask Registers 2

7	6	5	4	3	2	1	0
IDMSK 20	IDMSK 19	IDMSK 18	IDMSK 17	IDMSK 16	IDMSK 15	IDMSK 14	IDMSK 13

Bit Number	Bit Mnemonic	Description
7-0	IDMSK20:13	**IDentifier Mask Value** 0 - comparison true forced. 1 - bit comparison enabled. See Figure 54.

Note: The ID Mask is only used for reception.

No default value after reset.

Table 80. CANIDM3 Register for V2.0 part B

CANIDM3 for V2.0 part B (S:C6h)
CAN Identifier Mask Registers 3

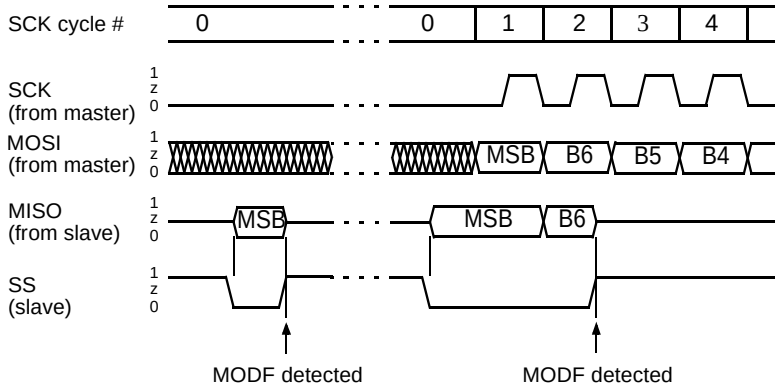
7	6	5	4	3	2	1	0
IDMSK 12	IDMSK 11	IDMSK 10	IDMSK 9	IDMSK 8	IDMSK 7	IDMSK 6	IDMSK 5

Bit Number	Bit Mnemonic	Description
7-0	IDMSK12:5	**IDentifier Mask Value** 0 - comparison true forced. 1 - bit comparison enabled. See Figure 54.

Note: The ID Mask is only used for reception.

No default value after reset.

Figure 65. Mode Fault Conditions in Slave Mode



Note: when SS is discarded (SS disabled) it is not possible to detect a MODF error in slave mode because the SPI is internally selected. Also the SS pin becomes a general purpose I/O.

OverRun Condition

This error mean that the speed is not adapted for the running application:

An OverRun condition occurs when a byte has been received whereas the previous one has not been read by the application yet.
The last byte (which generate the overrun error) does not overwrite the unread data so that it can still be read. Therefore, an overrun error always indicates the loss of data.

Interrupts

Three SPI status flags can generate a CPU interrupt requests:

Table 91. SPI Interrupts

Flag	Request
SPIF (SPI data transfer)	SPI Transmitter Interrupt Request
MODF (Mode Fault)	SPI mode-fault Interrupt Request
SPTE (Transmit register empty)	SPI transmit register empty Interrupt Request

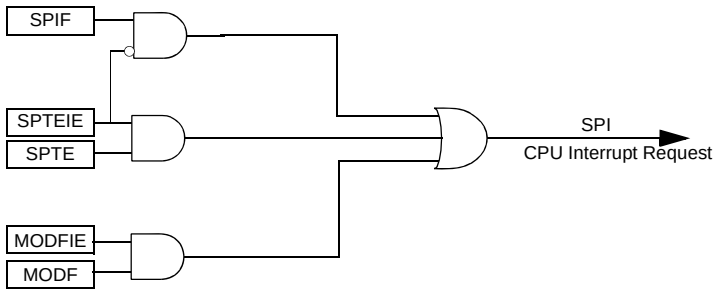
Serial Peripheral data transfer flag, SPIF: This bit is set by hardware when a transfer has been completed. SPIF bit generates transmitter CPU interrupt request only when SPTEIE is disabled.

Mode Fault flag, MODF: This bit is set to indicate that the level on the $\overline{SS}$ is inconsistent with the mode of the SPI (in both master and slave modes).

Serial Peripheral Transmit Register empty flag, SPTE: This bit is set when the transmit buffer is empty (other data can be loaded is SPDAT). SPTE bit generates transmitter CPU interrupt request only when SPTEIE is enabled.

Note: While using SPTE interruption for “burst mode” transfers (SPTEIE=’1’), the user software application should take care to clear SPTEIE, during the last but one data reception (to be able to generate an interrupt on SPIF flag at the end of the last data reception).

Figure 66. SPI Interrupt Requests Generation



Registers

Three registers in the SPI module provide control, status and data storage functions. These registers are describe in the following paragraphs.

Serial Peripheral Control Register (SPCON)

- The Serial Peripheral Control Register does the following:
- Selects one of the Master clock rates
- Configure the SPI Module as Master or Slave
- Selects serial clock polarity and phase
- Enables the SPI Module
- Frees the SS pin for a general-purpose

Table 92 describes this register and explains the use of each bit

Table 92. SPCON Register

SPCON - Serial Peripheral Control Register (0D4H)

7	6	5	4	3	2	1	0
SPR2	**SPEN**	**SSDIS**	**MSTR**	**CPOL**	**CPHA**	**SPR1**	**SPR0**

Bit Number	Bit Mnemonic	Description
7	SPR2	**Serial Peripheral Rate 2** Bit with SPR1 and SPR0 define the clock rate (See bits SPR1 and SPR0 for detail).
6	SPEN	**Serial Peripheral Enable** Cleared to disable the SPI interface (internal reset of the SPI). Set to enable the SPI interface.
5	SSDIS	**$\overline{SS}$ Disable** Cleared to enable $\overline{SS}$ in both Master and Slave modes. Set to disable $\overline{SS}$ in both Master and Slave modes. In Slave mode, this bit has no effect if CPHA ='0'. When SSDIS is set, no MODF interrupt request is generated.
4	MSTR	**Serial Peripheral Master** Cleared to configure the SPI as a Slave. Set to configure the SPI as a Master.

SPCON, SPSTA and SPDAT registers may be read and written at any time while there is no on-going exchange. However, special care should be taken when writing to them while a transmission is on-going:

- Do not change SPR2, SPR1 and SPR0
- Do not change CPHA and CPOL
- Do not change MSTR
- Clearing SPEN would immediately disable the peripheral
- Writing to the SPDAT will cause an overflow.

Analog-to-Digital Converter (ADC)

This section describes the on-chip 10 bit analog-to-digital converter of the AT89C51CC03. Eight ADC channels are available for sampling of the external sources AN0 to AN7. An analog multiplexer allows the single ADC converter to select one from the 8 ADC channels as ADC input voltage (ADCIN). ADCIN is converted by the 10-bit cascaded potentiometric ADC.

Two kinds of conversion are available:
- Standard conversion (8 bits).
- Precision conversion (10 bits) (Up to 85°C only).

For the precision conversion, set bit PSIDLE in ADCON register and start conversion. The device is in a pseudo-idle mode, the CPU does not run but the peripherals are always running. This mode allows digital noise to be as low as possible, to ensure high precision conversion.

For this mode it is necessary to work with end of conversion interrupt, which is the only way to wake the device up.

If another interrupt occurs during the precision conversion, it will be treated only after this conversion is ended.

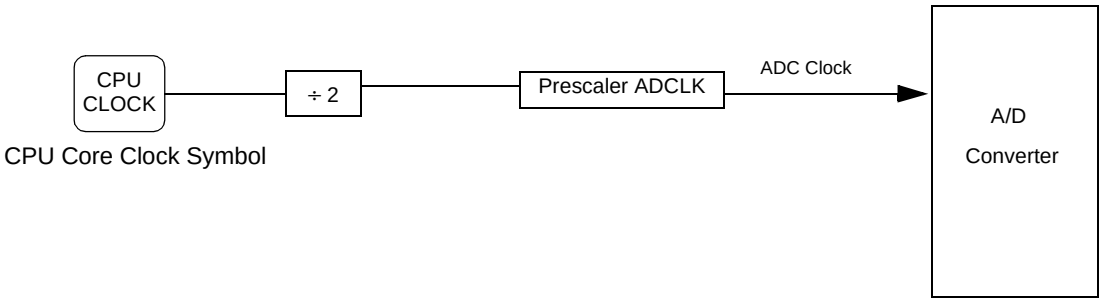
Features

- 8 channels with multiplexed inputs
- 10-bit cascaded potentiometric ADC
- Conversion time 16 micro-seconds (typ.)
- Zero Error (offset) ± 2 LSB max
- Positive External Reference Voltage Range (VREF) 2.4 to 3.0Volt (typ.)
- ADCIN Range 0 to 3Volt
- Integral non-linearity typical 1 LSB, max. 2 LSB
- Differential non-linearity typical 0.5 LSB, max. 1 LSB
- Conversion Complete Flag or Conversion Complete Interrupt
- Selectable ADC Clock

ADC Port1 I/O Functions

Port 1 pins are general I/O that are shared with the ADC channels. The channel select bit in ADCF register define which ADC channel/port1 pin will be used as ADCIN. The remaining ADC channels/port1 pins can be used as general-purpose I/O or as the alternate function that is available.

Figure 75. A/D Converter Clock



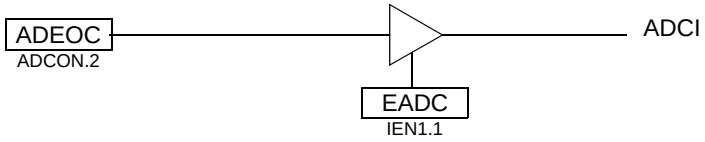
ADC Standby Mode

When the ADC is not used, it is possible to set it in standby mode by clearing bit ADEN in ADCON register. In this mode its power dissipation is about 1 µW.

IT ADC Management

An interrupt end-of-conversion will occurs when the bit ADEOC is activated and the bit EADC is set. For re-arming the interrupt the bit ADEOC must be cleared by software.

Figure 76. ADC Interrupt Structure



Routines examples

1. Configure P1.2 and P1.3 in ADC channels

```
// configure channel P1.2 and P1.3 for ADC
 ADCF = 0Ch

// Enable the ADC
 ADCON = 20h
```

2. Start a standard conversion

```
// The variable "channel" contains the channel to convert
// The variable "value_converted" is an unsigned int
// Clear the field SCH[2:0]
 ADCON and = F8h
// Select channel
 ADCON | = channel
// Start conversion in standard mode
 ADCON | = 08h
// Wait flag End of conversion
 while((ADCON and 01h)! = 01h)
// Clear the End of conversion flag
 ADCON and = EFh
// read the value
 value_converted = (ADDH << 2)+(ADDL)
```

3. Start a precision conversion (need interrupt ADC)

```
// The variable "channel" contains the channel to convert
// Enable ADC
```

Table 112. IPL0 Register

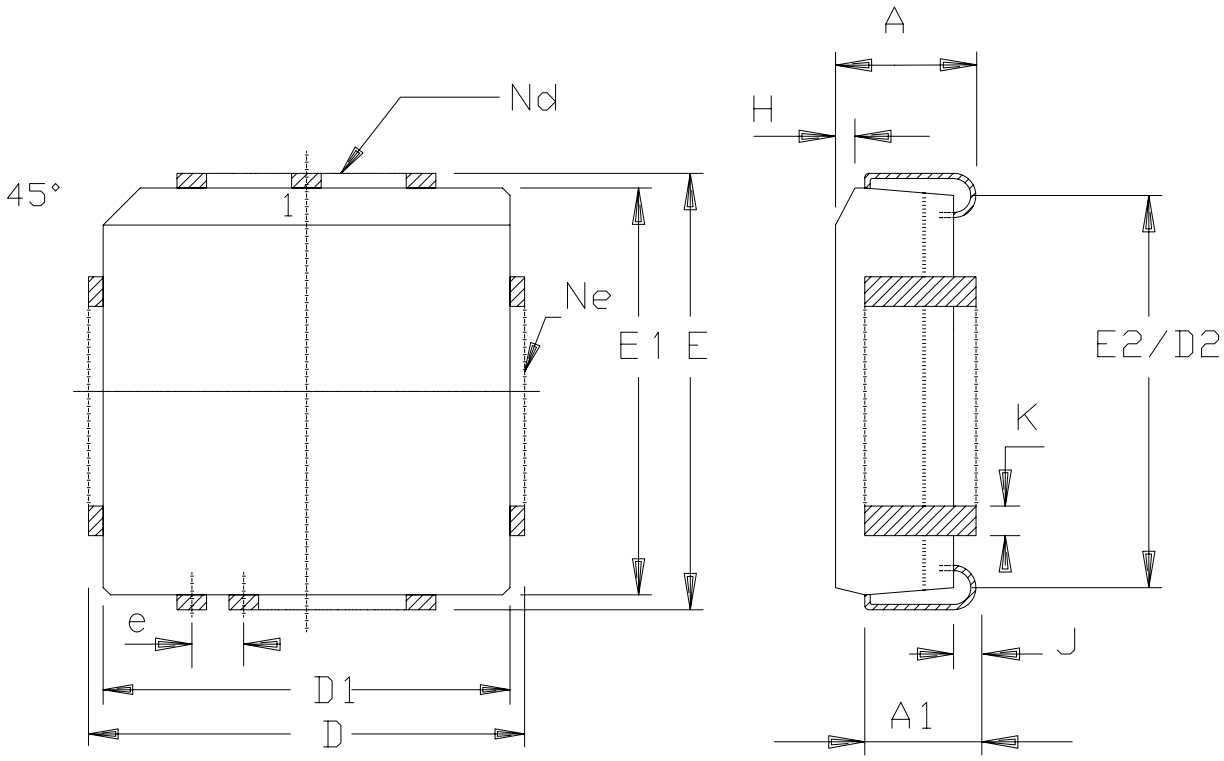
IPL0 (S:B8h)
Interrupt Enable Register

7	6	5	4	3	2	1	0
-	PPC	PT2	PS	PT1	PX1	PT0	PX0

Bit Number	Bit Mnemonic	Description
7	-	**Reserved** The value read from this bit is indeterminate. Do not set this bit.
6	PPC	**PCA Interrupt Priority bit** Refer to PPCH for priority level
5	PT2	**Timer 2 Overflow Interrupt Priority bit** Refer to PT2H for priority level.
4	PS	**Serial Port Priority bit** Refer to PSH for priority level.
3	PT1	**Timer 1 Overflow Interrupt Priority bit** Refer to PT1H for priority level.
2	PX1	**External Interrupt 1 Priority bit** Refer to PX1H for priority level.
1	PT0	**Timer 0 Overflow Interrupt Priority bit** Refer to PT0H for priority level.
0	PX0	**External Interrupt 0 Priority bit** Refer to PX0H for priority level.

Reset Value = X000 0000b
bit addressable

PLCC44



	MM		INCH	
A	4.20	4.57	.165	.180
A1	2.29	3.04	.090	.120
D	17.40	17.65	.685	.695
D1	16.44	16.66	.647	.656
D2	14.99	16.00	.590	.630
E	17.40	17.65	.685	.695
E1	16.44	16.66	.647	.656
E2	14.99	16.00	.590	.630
e	1.27	BSC	.050	BSC
H	1.07	1.42	.042	.056
J	0.51	-	.020	-
K	0.33	0.53	.013	.021
Nd	11		11	
Ne	11		11	
PKG STD	00			