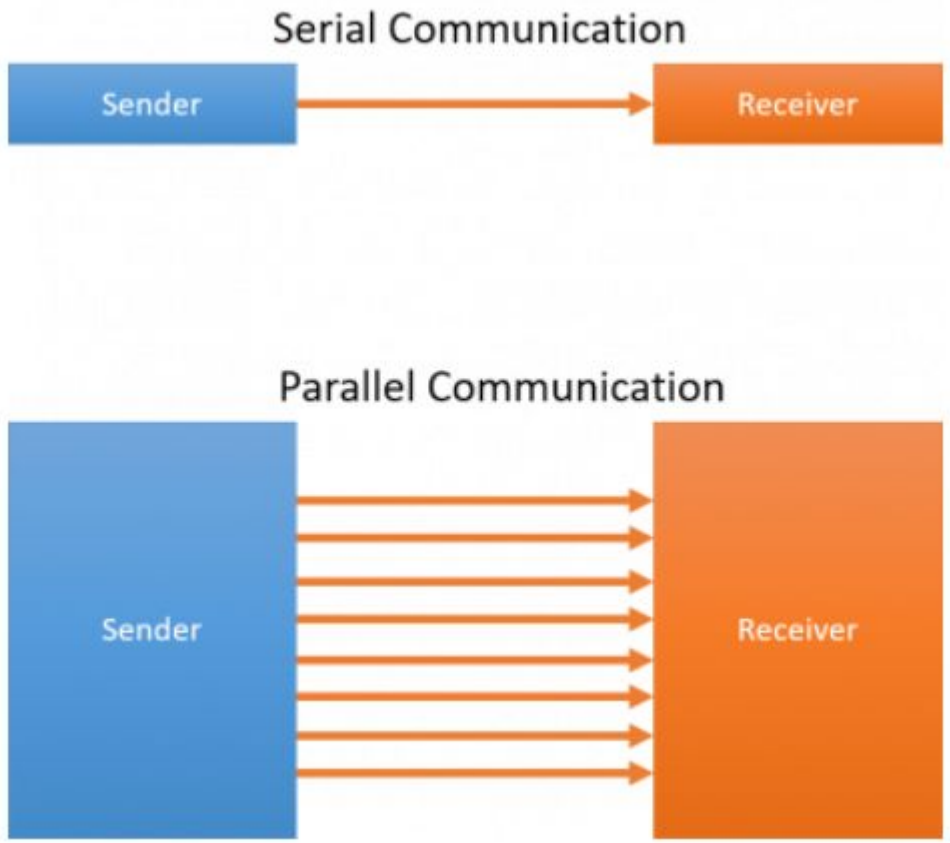
MIT-WPU
School of Electrical and Computer Engineering
T.Y. B. Tech (Semester V)
Microcontroller and Applications
ECE2003B

Serial Port Programming

- Basics of Serial Communication Protocol
- Study of
  - RS 232
  - RS 485
  - UART
  - USB

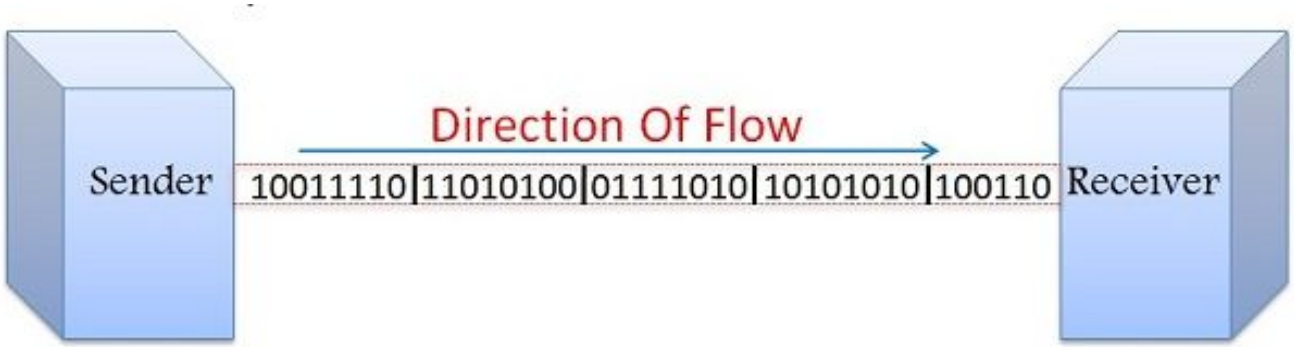
Introduction

- **Parallel communication implies sending a whole byte (or more) of data over multiple parallel wires**
- **Serial communication implies sending data bit by bit over a single wire**



Synchronous Communication

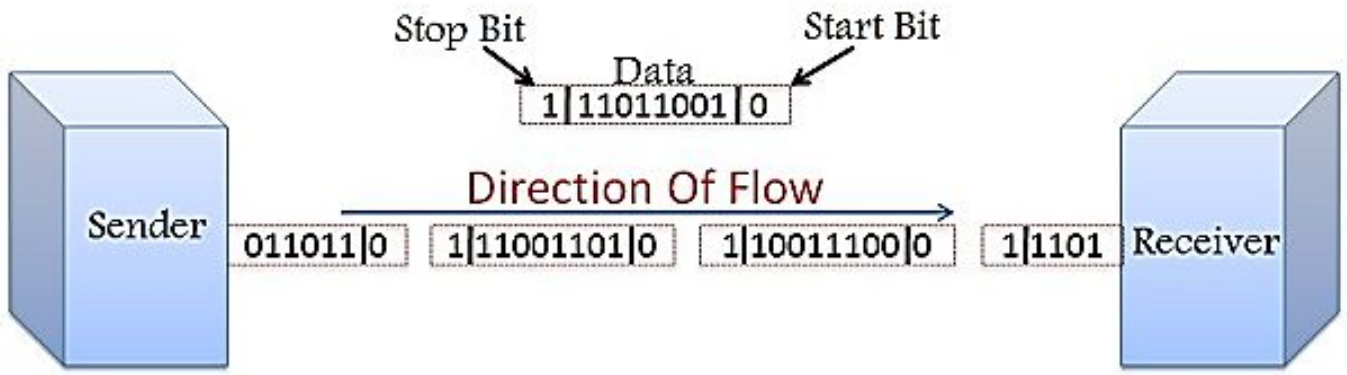
- **In Synchronous Communication, data flows in the form of blocks or frames.**
- **Synchronization between the sender and receiver is necessary so that the sender know where the new byte starts (since there is no gap between the data).**



Asynchronous Communication

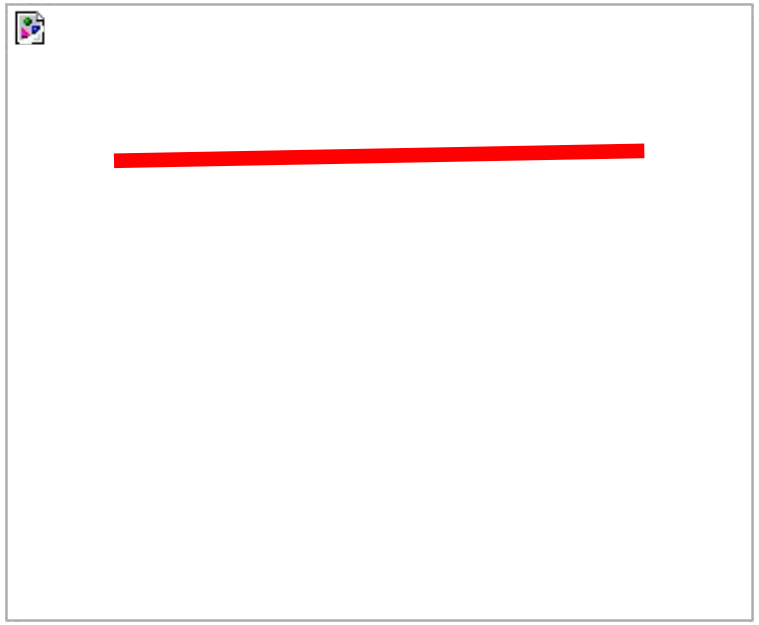
- **In Asynchronous Communication data flows, 1 byte or a character at a time.**
- **It does not require a clock for synchronization; rather it uses the parity bits to tell the receiver how to interpret the data.**
- **The size of a character sent is 8 bits to which a parity bit is added i.e.**

  a start and a stop bit that gives the total of 10 bits.



Modes of Communication

Simplex Mode



Half-Duplex Mode

Full-Duplex Mode

Modes of Communication

Basis for Comparison	Simplex	Half Duplex	Full Duplex
Direction of Communication	Unidirectional	Two-directional, one at a time	Two-directional, simultaneously
Send / Receive	Sender can only send data	Sender can send and receive data, but one a time	Sender can send and receive data simultaneously
Performance	Worst performing mode of transmission	Better than Simplex	Best performing mode of transmission
Example	Keyboard and monitor	Walkie-talkie	Telephone

Serial Communication Protocol

A protocol is a set of rules agreed by both the sender and receiver on

- **How the data is packed**
- **How many bits constitute a character**
- **When the data begins and ends**

Many popular serial communication standards exist—

Some examples are:

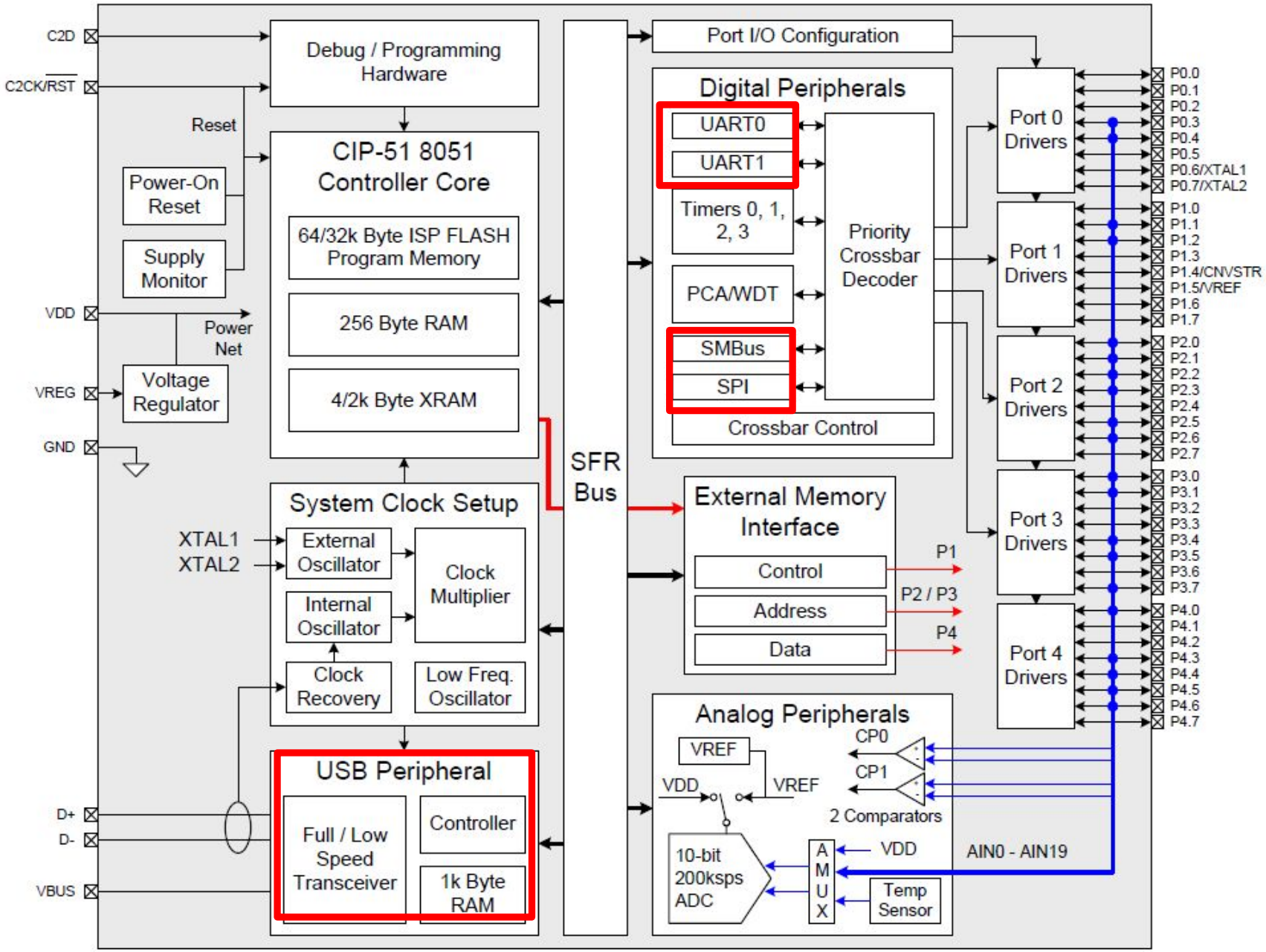
- **RS 485**
- **RS-232 (using UART)**
- **Serial Peripheral interface (SPI)**
- **Inter-Integrated Circuit (I2C)**
- **System Management Bus (SMBus)**
- **USB**

Serial Communication Buses

- The C8051F340 has
  - Two UARTs (UART0 & UART1)
  - one SPI
  - one SMBus
  - one USB hardware peripherals

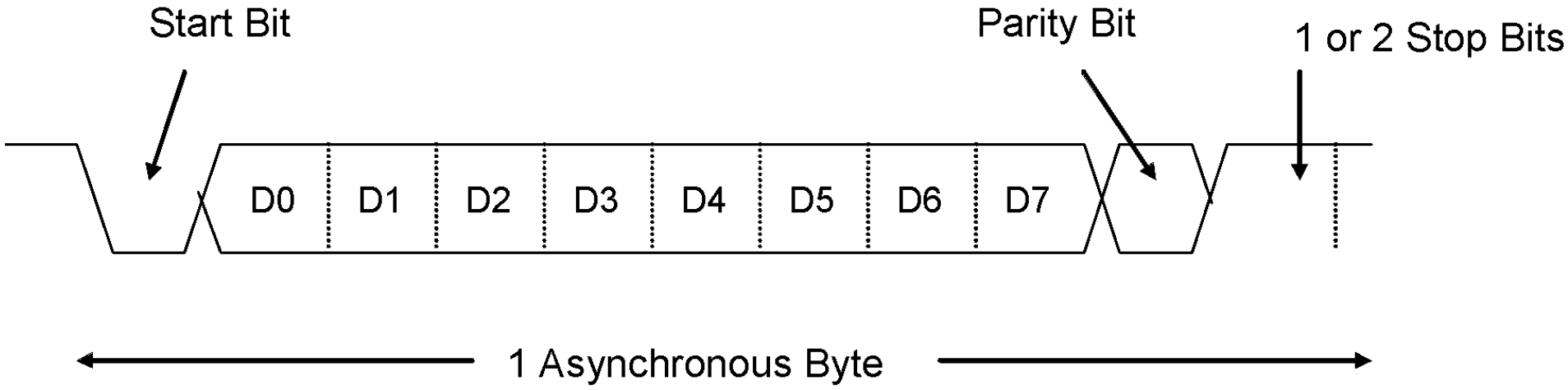
- UART: Universal Asynchronous Receiver/Transmitter

Serial Communication Buses



Asynchronous Serial Communication

- **Start bit—indicates the beginning of the data word (logic ‘0’)**
- **Stop bit—indicates the end of the data word (logic ‘1’)**
- **Parity bit—added for error detection (optional)**
- **Data bits—the actual data to be transmitted (8-bits)**



Asynchronous Serial Communication

- Asynchronous serial data communication is widely used for character-oriented transmissions
- Each character is placed in between start and stop bits, this is called framing
- The start bit is always be one bit, but the stop bit can be one or two bits
- Asynchronous transmission is easy to implement but less efficient as it requires an extra 2-3 control bits for every 8 data bits
- This method is usually used for low volume transmission

Asynchronous Serial Communication

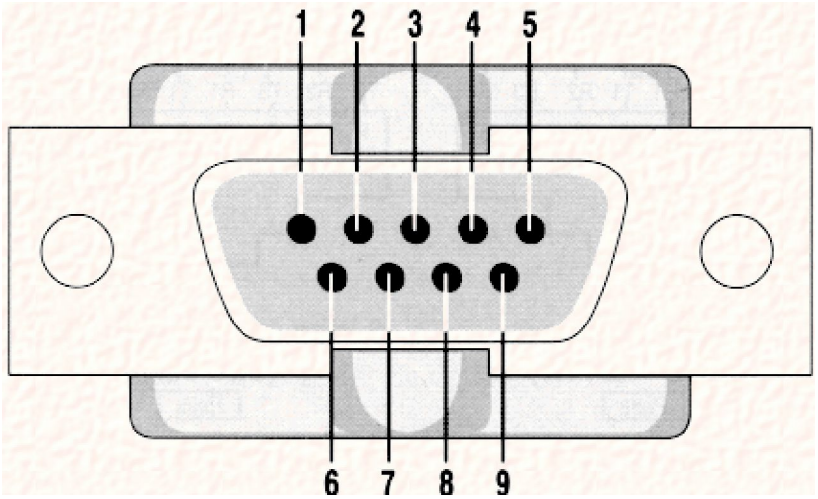
- The rate of data transfer in serial data communication is stated in bps (bits per second)
- Another widely used terminology for bps is baud rate, it is modem terminology and is defined as the number of signal changes per second
- In modems, there are occasions when a single change of signal transfers several bits of data
- As far as the conductor wire is concerned, the baud rate and bps are the same, and we use the terms interchangeably
- Throughput—actual data transmitted per sec (total bits transmitted—overhead)
  - Example: 115200 baud rate
  - If using 8-bit data, 1 start, 1 stop, and no parity bits,
  - Effective throughput is: 115200 * 8 / 10 = 92160 bits/sec

11101001

RS232

- An interfacing standard RS232 was set by the Electronics Industries Association (EIA) in 1960.
- The standard was set long before the advent of the TTL logic family, its input and output voltage levels are not TTL compatible
- In RS232, a Logic '1' is represented by -3V to -25V, while a Logic '0' bit is +3V to +25V, making -3V to +3V undefined
- A line driver such as the MAX232 chip is required to convert RS232 voltage levels to TTL levels, and vice versa
- RS232 is available in two types
    1. DB25
    2. DB9

RS232 DB9 Connector



Pin	Description
1	Data carrier detect (DCD)
2	Received data (RxD)
3	Transmitted data (TxD)
4	Data terminal ready (DTR)
5	Signal ground (GND)
6	Data set ready (DSR)
7	Request to send (RTS)
8	Clear to send (CTS)
9	Ring indicator (RI)

Handshaking Signals in RS232

DTR (data terminal ready)

When terminal is turned on, it sends out signal DTR to indicate that it is ready for communication

DSR (data set ready)

When DCE is turned on and has gone through the self-test, it assert DSR to indicate that it is ready to communicate

RTS (request to send)

When the DTE device has byte to transmit, it assert RTS to signal the modem that it has a byte of data to transmit

CTS (clear to send)

When the modem has room for storing the data it is to receive, it sends out signal CTS to DTE to indicate that it can receive the data now

Handshaking Signals in RS232

DCD (data carrier detect)

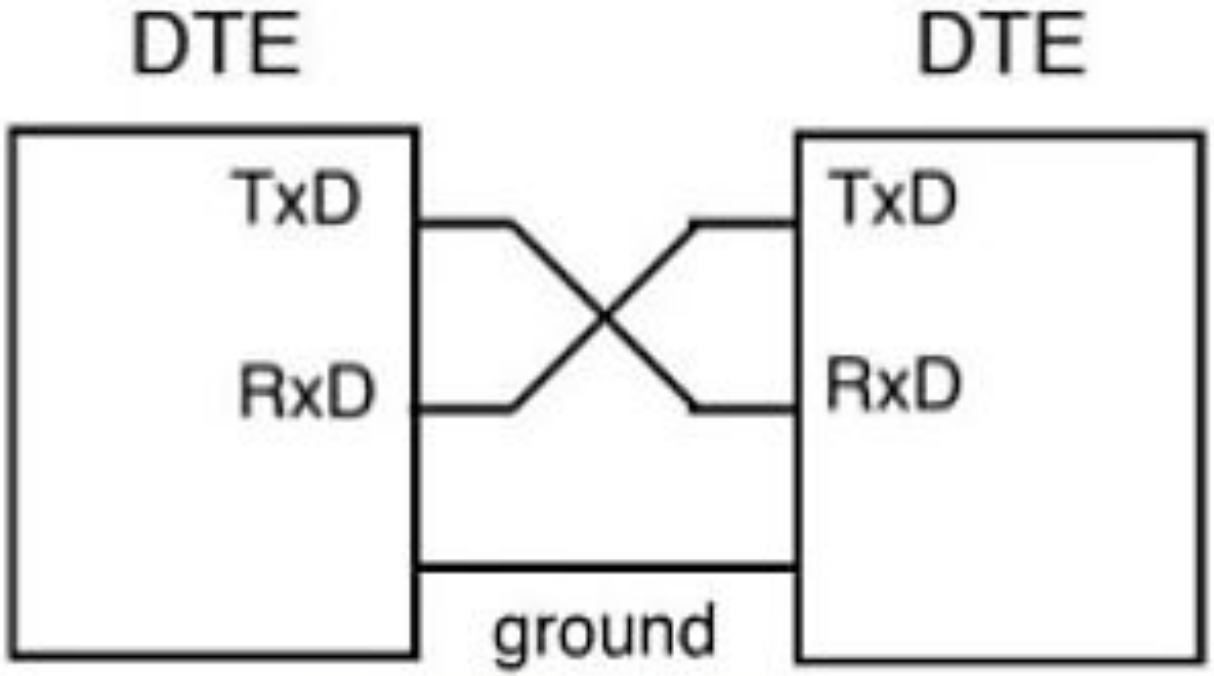
The modem asserts signal DCD to inform the DTE that a valid carrier has been detected and that contact between it and the other modem is established

RI (ring indicator)

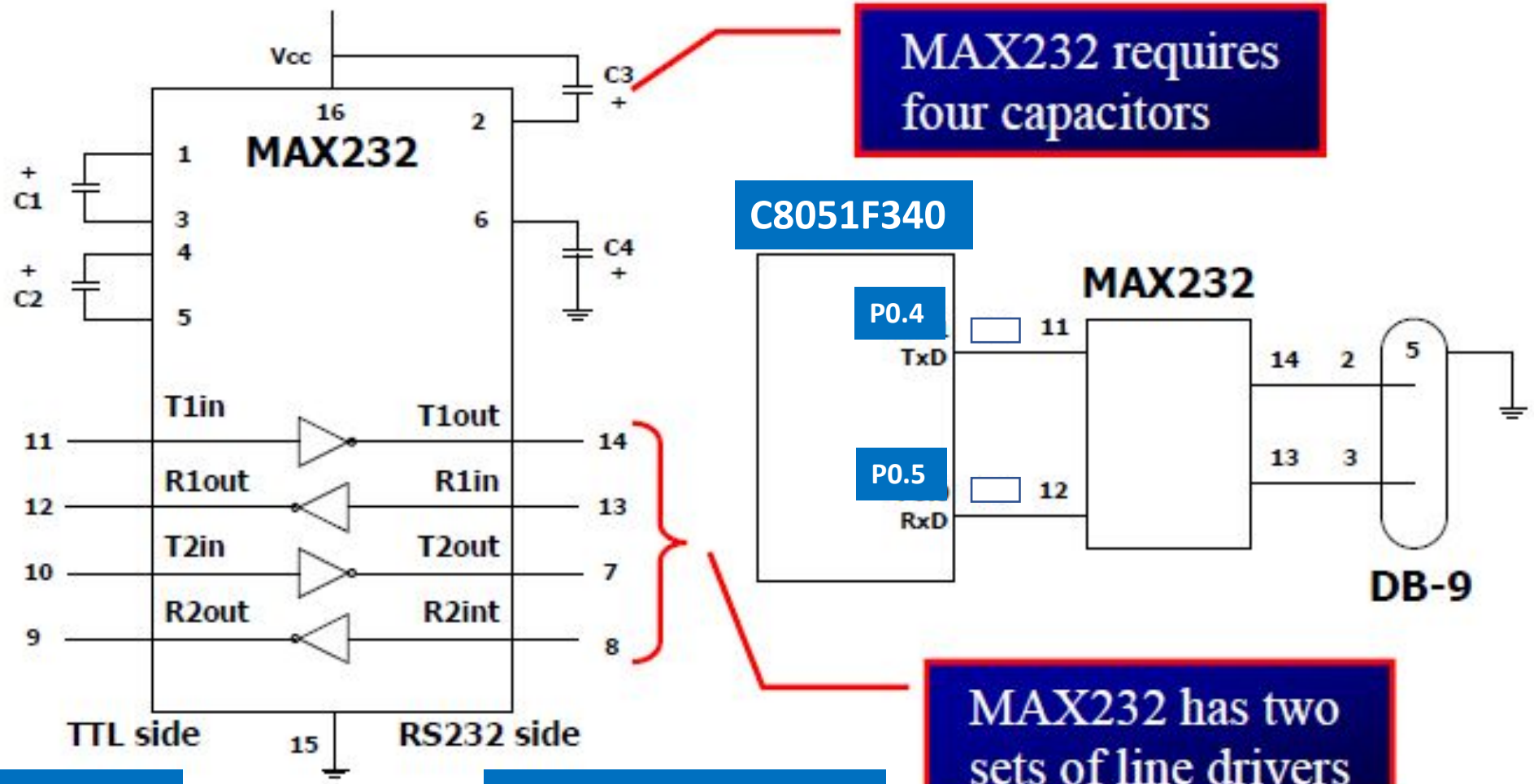
- An output from the modem and an input to a PC indicates that the telephone is ringing
- It goes on and off in synchronous with the ringing sound

Connection between PC and Microcontroller

- The connection between PC and Microcontroller requires only three pins viz. TxD, RxD and Ground



Connection of RS232 using MAX232 with C8051F340



RS-485

- RS 485 was developed to provide high speed data.
- The standard is defined by industry telecommunications bodies and is referred to most commonly as RS485, but references to EIA485 or TIA-485 may also be seen.
- RS 485 is able to provide a headline data rate of 10 Mbps at distances up to 50 feet, but distances can be extended to 4000 feet with a lower speed of 100 kbps.
- Although RS485 was never intended for domestic use, it found many applications where remote data acquisition was require.
- In general,

The RS-485 is for higher speeds over longer ranges.

The RS-232 is best for short-distance low-speed requirements.

Comparison between RS232 and RS485

Characteristics of RS232 and RS485		
Parameter	**RS232**	**RS485**
Cabling	Single-edned	Differential
Numbers of devices	1 transmitter 1 receiver	32 transmitters 32 receivers
Mode of operation	Simplex or full duplex	Simplex or half duplex
Maximum cable length	50 feet	4000 feet
Maximum data rate	20 kbits/s	10 Mbits/s
Signaling	unbalanced	balanced
Typical logic levels	±5 ~ ±15V	±1.5 ~ ±6V
Minimun receiver input impedance	3~7kΩ	12kΩ
Receiver sensitivity	±3V	±200mA

UART0 of C8051F340

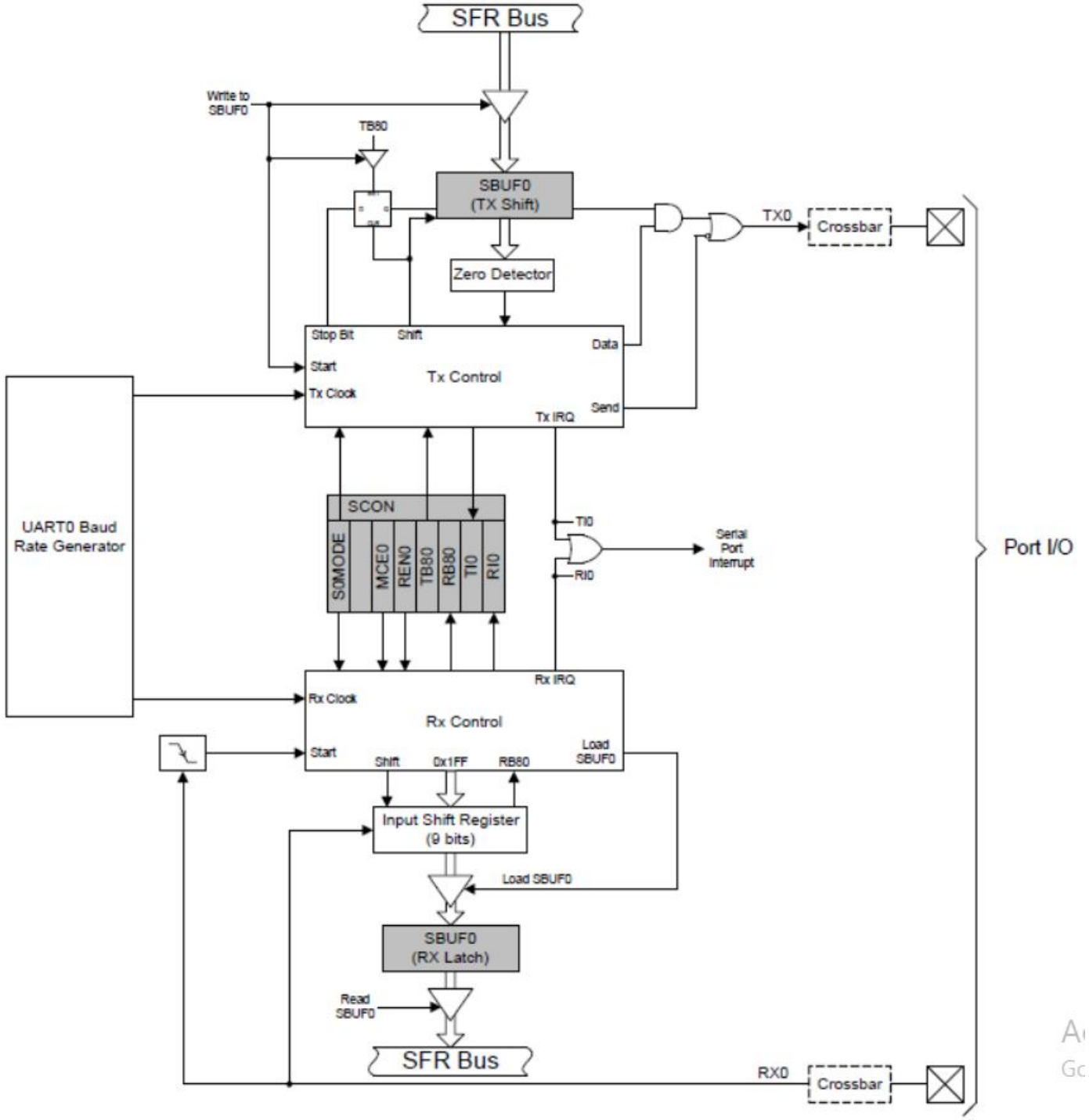
- UART0 is an asynchronous, full duplex serial port offering modes 1 and 3 of the standard 8051 UART.

- UART0 has two associated SFRs:
  1. Serial Control Register 0 (SCON0) and
  2. Serial Data Buffer 0 (SBUF0)

- The single SBUF0 location provides access to both transmit and receive registers.

- Writes to SBUF0 always access the Transmit register. Reads of SBUF0 always access the buffered Receive register; it is not possible to read data from the Transmit register.

UART0 Block Diagram



Registers associated with UART0

- The Serial Port Buffer 0 (SBUF0) :
  - Writes to SBUF0 always access the Transmit register
  - Reads of SBUF0 always access the buffered Receive register
- The Serial Port Control register 0 (SCON0) contains status and control bits
  - The control bits set the operating mode for the serial port, and status bits indicate the end of the character transmission or reception
  - The status bits are tested in software (polling) or programmed to cause an interrupt (TI0 & RI0)

UART Clock Requirements

- A UART needs a clock input for bit timing
- UART baud rates are usually much lower than the MCU system clock, so the system clock cannot be directly used as the UART clock
- Timers are used to generate the UART baud rate by dividing down the system clock

Baud Rate Generation

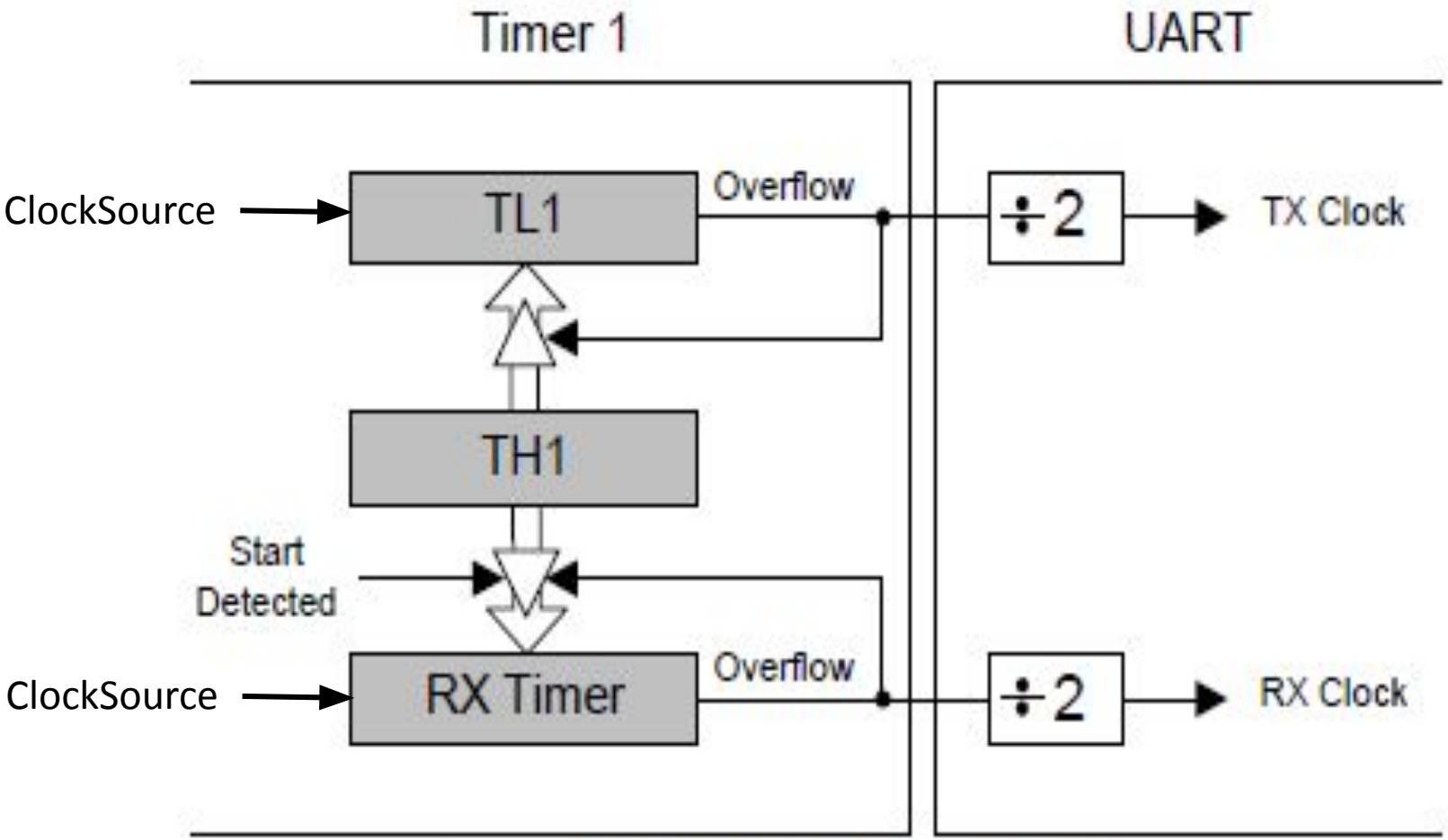


Figure 18.2. UART0 Baud Rate Logic

Baud Rate Generation

- The UART0 baud rate is generated by Timer 1 in Mode 2 i.e. 8-bit auto-reload mode.

- The TX clock is generated by TL1; the RX clock is generated by a copy of TL1 which is not user-accessible.

- Both TX and RX Timer overflows are divided by two to generate the TX and RX baud rates. The RX Timer runs when Timer 1 is enabled, and uses the same reload value (TH1).

- RX Timer reload is forced when a START condition is detected on the RX pin. This allows a receive to begin any time a START is detected, independent of the TX Timer state.

UART Baud Rate generation

$$UartBaudRate = \frac{SystemClock}{(256 - TH1)} \times \frac{1}{2}$$

- **Where SystemClock is the frequency of the clock supplied to Timer 1, and TH1 is the high byte of Timer 1**

- **Timer 1 may be clocked by one of six sources:**
  1. **SYSCLK,**
  2. **SYSCLK / 4,**
  3. **SYSCLK / 12,**
  4. **SYSCLK / 48,**
  5. **External oscillator clock / 8,**
  6. **External input TI.**

- **TH1 = 256 - (SystemClock/UartBaudRate/2)**

UART Baud Rate generation

$$UartBaudRate = \frac{SystemClock}{(256 - TH1)} \times \frac{1}{2}$$

Baud rate = 9600, Consider SystemClock = 12MH

- TH1 = 256 - (SystemClock/UartBaudRate/2)
- TH1 = 256 – 625
- As 625>256 ------ Pre-scalar is required to bring value in the range of 256
- So use pre-scalar as /4
- TH1 = 256 - (625/4)
- TH1 = 256 - 156.25 =99.75 ~ (100)d = (64)hex

UART Baud Rate generation

$$UartBaudRate = \frac{SystemClock}{(256 - TH1)} \times \frac{1}{2}$$

Baud rate = 2400, Consider SystemClock = 12MH

- TH1 = 256 - (SystemClock/UartBaudRate/2)
- TH1 = 256 – 2500
- As 2500>256 ------Pre-scalar is required to bring value in the range of 256
- So use pre-scalar as /12
- TH1 = 256 - (2500/12)
- TH1 = 256 – 208.33 = 47.67 ~ (48)d = (30)hex

UART Baud Rate generation

$$UartBaudRate = \frac{SystemClock}{(256 - TH1)} \times \frac{1}{2}$$

Baud rate = 1200, Consider SystemClock = 12MH

- TH1 = 256 - (SystemClock/UartBaudRate/2)
- TH1 = 256 – 5000
- As 5000>256 ------Pre-scalar is required to bring value in the range of 256
- So use pre-scalar as /48
- TH1 = 256 - (5000/48)
- TH1 = 256 – 104.16 = 151.84 ~ (151)d = (98)hex

Timer Settings for Standard Baud Rates Using the Internal Oscillator

Table 18.1. Timer Settings for Standard Baud Rates Using the Internal Oscillator

	Target Baud Rate (bps)	Actual Baud Rate (bps)	Baud Rate Error	Oscillator Divide Factor	Timer Clock Source	SCA1-SCA0 (pre-scale select*)	T1M*	Timer 1 Reload Value (hex)
SYSCLK = 12 MHz	230400	230769	0.16%	52	SYSCLK	XX	1	0xE6
	115200	115385	0.16%	104	SYSCLK	XX	1	0xCC
	57600	57692	0.16%	208	SYSCLK	XX	1	0x98
	28800	28846	0.16%	416	SYSCLK	XX	1	0x30
	14400	14423	0.16%	832	SYSCLK / 4	01	0	0x98
	9600	9615	0.16%	1248	SYSCLK / 4	01	0	0x64
	2400	2404	0.16%	4992	SYSCLK / 12	00	0	0x30
	1200	1202	0.16%	9984	SYSCLK / 48	10	0	0x98

Operational Modes

- UART0 provides standard asynchronous, full duplex communication.
- The UART mode
  - 8-bit
  - 9-bit
- UART mode is selected by the S0MODE bit (SCON0.7).

Serial Port 0 Control: SCON

R/W	R	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
S0MODE	-	MCE0	REN0	TB80	RB80	TI0	RI0	01000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit Addressable

SFR Address: 0x98

Bit7: S0MODE: Serial Port 0 Operation Mode.
This bit selects the UART0 Operation Mode.
0: 8-bit UART with Variable Baud Rate.
1: 9-bit UART with Variable Baud Rate.

Bit6: UNUSED. Read = 1b. Write = don't care.

Bit5: MCE0: Multiprocessor Communication Enable.
The function of this bit is dependent on the Serial Port 0 Operation Mode.
S0MODE = 0: Checks for valid stop bit.
0: Logic level of stop bit is ignored.
1: RI0 will only be activated if stop bit is logic level 1.
S0MODE = 1: Multiprocessor Communications Enable.
0: Logic level of ninth bit is ignored.
1: RI0 is set and an interrupt is generated only when the ninth bit is logic 1.

Bit4: REN0: Receive Enable.
This bit enables/disables the UART receiver.
0: UART0 reception disabled.
1: UART0 reception enabled.

Serial Port 0 Control: SCON

Bit3: TB80: Ninth Transmission Bit.
The logic level of this bit will be assigned to the ninth transmission bit in 9-bit UART Mode. It is not used in 8-bit UART Mode. Set or cleared by software as required.

Bit2: RB80: Ninth Receive Bit.
RB80 is assigned the value of the STOP bit in Mode 0; it is assigned the value of the 9th data bit in Mode 1.

Bit1: TI0: Transmit Interrupt Flag.
Set by hardware when a byte of data has been transmitted by UART0 (after the 8th bit in 8-bit UART Mode, or at the beginning of the STOP bit in 9-bit UART Mode). When the UART0 interrupt is enabled, setting this bit causes the CPU to vector to the UART0 interrupt service routine. This bit must be cleared manually by software.

Bit0: RI0: Receive Interrupt Flag.
Set to '1' by hardware when a byte of data has been received by UART0 (set at the STOP bit sampling time). When the UART0 interrupt is enabled, setting this bit to '1' causes the CPU to vector to the UART0 interrupt service routine. This bit must be cleared manually by software.

SBUF0

SFR Definition 18.2. SBUF0: Serial (UART0) Port Data Buffer

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	

SFR Address: 0x99

Bits7–0: SBUF0[7:0]: Serial Data Buffer Bits 7–0 (MSB-LSB)
This SFR accesses two registers; a transmit shift register and a receive latch register. When data is written to SBUF0, it goes to the transmit shift register and is held for serial transmission. Writing a byte to SBUF0 initiates the transmission. A read of SBUF0 returns the contents of the receive latch.

SFR- CKCON: Clock Control

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
T3MH	T3ML	T2MH	T2ML	T1M	T0M	SCA1	SCA0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0x8E

Bit7: T3MH: Timer 3 High Byte Clock Select.
This bit selects the clock supplied to the Timer 3 high byte if Timer 3 is configured in split 8-bit timer mode. T3MH is ignored if Timer 3 is in any other mode.
0: Timer 3 high byte uses the clock defined by the T3XCLK bit in TMR3CN.
1: Timer 3 high byte uses the system clock.

Bit6: T3ML: Timer 3 Low Byte Clock Select.
This bit selects the clock supplied to Timer 3. If Timer 3 is configured in split 8-bit timer mode, this bit selects the clock supplied to the lower 8-bit timer.
0: Timer 3 low byte uses the clock defined by the T3XCLK bit in TMR3CN.
1: Timer 3 low byte uses the system clock.

Bit5: T2MH: Timer 2 High Byte Clock Select.
This bit selects the clock supplied to the Timer 2 high byte if Timer 2 is configured in split 8-bit timer mode. T2MH is ignored if Timer 2 is in any other mode.
0: Timer 2 high byte uses the clock defined by the T2XCLK bit in TMR2CN.
1: Timer 2 high byte uses the system clock.

Bit4: T2ML: Timer 2 Low Byte Clock Select.
This bit selects the clock supplied to Timer 2. If Timer 2 is configured in split 8-bit timer mode, this bit selects the clock supplied to the lower 8-bit timer.
0: Timer 2 low byte uses the clock defined by the T2XCLK bit in TMR2CN.
1: Timer 2 low byte uses the system clock.

SFR- CKCON: Clock Control

Bit3: T1M: Timer 1 Clock Select.
This select the clock source supplied to Timer 1. T1M is ignored when C/T1 is set to logic 1.
0: Timer 1 uses the clock defined by the prescale bits, SCA1-SCA0.
1: Timer 1 uses the system clock.

Bit2: T0M: Timer 0 Clock Select.
This bit selects the clock source supplied to Timer 0. T0M is ignored when C/T0 is set to logic 1.
0: Counter/Timer 0 uses the clock defined by the prescale bits, SCA1-SCA0.
1: Counter/Timer 0 uses the system clock.

Bits1–0: SCA1-SCA0: Timer 0/1 Prescale Bits.
These bits control the division of the clock supplied to Timer 0 and/or Timer 1 if configured to use prescaled clock inputs.

SCA1	SCA0	Prescaled Clock
0	0	System clock divided by 12
0	1	System clock divided by 4
1	0	System clock divided by 48
1	1	External clock divided by 8

Note: External clock divided by 8 is synchronized with the system clock.

XBR0 (Crossbar Register 0)

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
CP1AE	CP1E	CP0AE	CP0E	SYSCKE	SMB0E	SPI0E	URT0E	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address:
								0xE1

Bit7: CP1AE: Comparator1 Asynchronous Output Enable
0: Asynchronous CP1 unavailable at Port pin.
1: Asynchronous CP1 routed to Port pin.

Bit6: CP1E: Comparator1 Output Enable
0: CP1 unavailable at Port pin.
1: CP1 routed to Port pin.

Bit5: CP0AE: Comparator0 Asynchronous Output Enable
0: Asynchronous CP0 unavailable at Port pin.
1: Asynchronous CP0 routed to Port pin.

Bit4: CP0E: Comparator0 Output Enable
0: CP0 unavailable at Port pin.
1: CP0 routed to Port pin.

Bit3: SYSCKE: /SYSCLK Output Enable
0: /SYSCLK unavailable at Port pin.
1: /SYSCLK output routed to Port pin.

Bit2: SMB0E: SMBus I/O Enable
0: SMBus I/O unavailable at Port pins.
1: SMBus I/O routed to Port pins.

Bit1: SPI0E: SPI I/O Enable
0: SPI I/O unavailable at Port pins.
1: SPI I/O routed to Port pins.

Bit0: URT0E: UART0 I/O Output Enable
0: UART0 I/O unavailable at Port pins.
1: UART0 TX0, RX0 routed to Port pins P0.4 and P0.5.

8-Bit UART Transmission

- 8-Bit UART mode uses a total of 10 bits per data byte: one start bit, eight data bits (LSB first), and one stop bit.
- Data in LSB first format is transmitted from the TX0 pin and received at the RX0 pin.
- Data transmission begins when a data byte is written to the SBUF0 register.
- The TI0 Transmit Interrupt Flag (SCON0.1) is set at the end of the transmission (the beginning of the stop-bit time).

8 Bit UART diagram

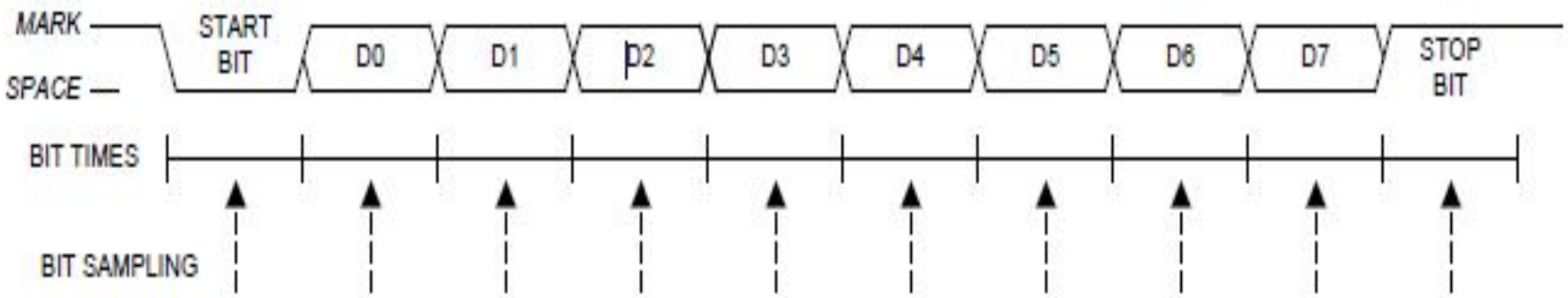
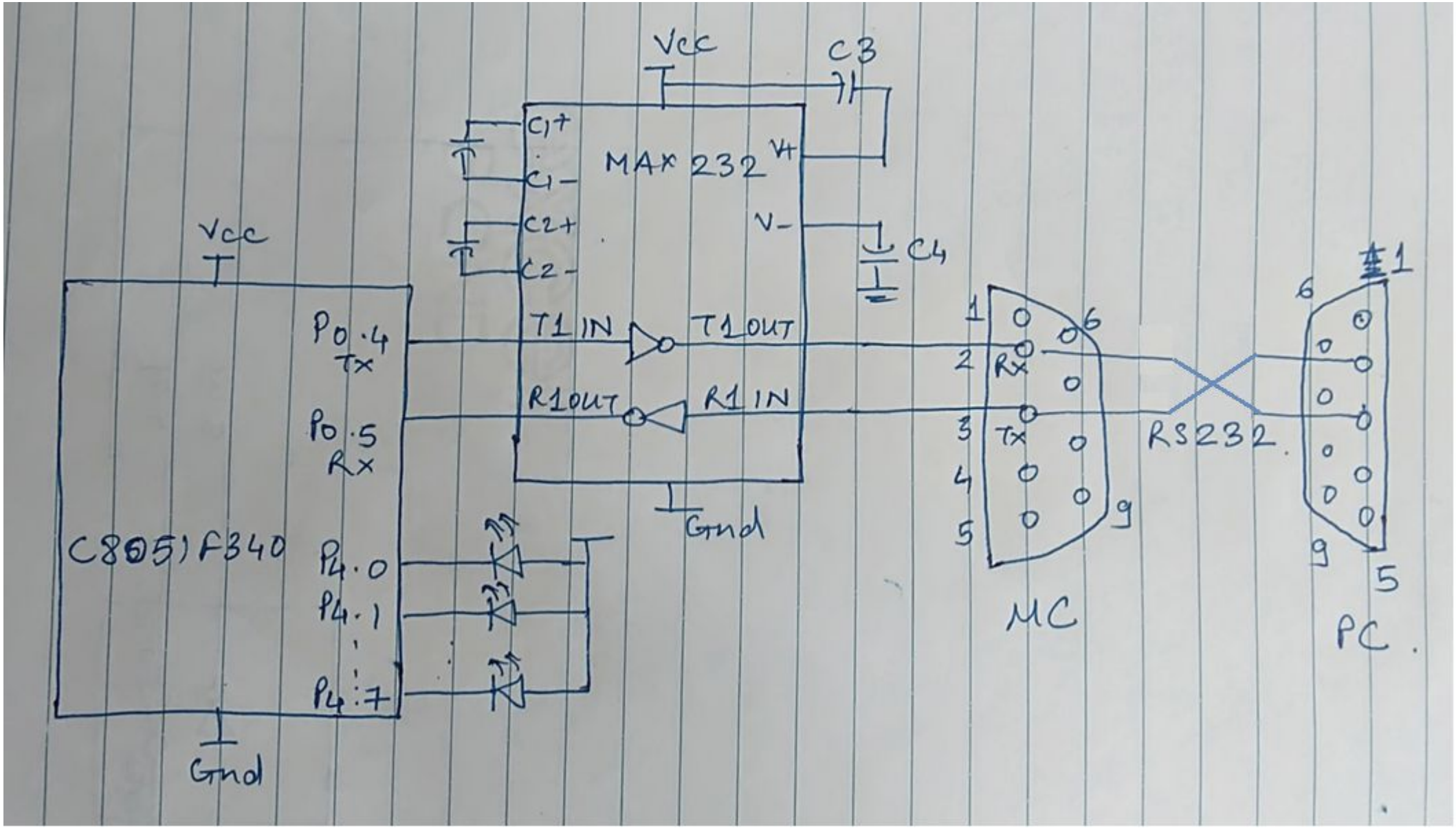


Figure 18.4. 8-Bit UART Timing Diagram

8-Bit UART Reception

- Data reception can begin any time after the REN0 Receive Enable bit (SCON0.4) is set to logic 1.
- After the stop bit is received, the data byte will be loaded into the SBUF0 receive register.
- On receive, the eight data bits are stored in SBUF0 and the stop bit goes into RB80 (SCON0.2).
- The RI0 Receive Interrupt Flag (SCON0.0) is set at the end of the reception (the beginning the stop-bit time).

Interfacing Diagram for UART Communication



UART Transmission Algorithm

- Define SYSCLK 12000000
- Define BR_UART0 9600
- Declare and Initialize character array with string to be transmitted
- Initialize a variable for ‘FOR’ loop
- Configure internal oscillator for its maximum frequency 12MHz (OSCICN)
- Enable UART through crossbar priority decoder (XBR0)
- Enable crossbar priority decoder (XBR1)
- Configure transmit pin as output (P0MDOUT)
- Configure 8-bit transmission and reception disabled (SCON)
- Select pre-scaler for Timer 0/1 (CKCON)

UART Transmission Algorithm (Contd.)

- Load count for desired baud rate in TH1
- Load TH1 value in TL1
- Configure TMOD for Timer 1 in Mode 2
- Start Timer
- For loop
  - Load each character to be sent in SBUF0 register one by one
  - Wait for transmission to be over by polling for TI0 flag
  - Clear TI0 flag for the next iteration
- Stop

UART Transmission Program

```
#include "c8051F340.h"
#define SYSCLK 12000000
#define BR_UART0 9600
void main()
{
 char ch[]={"MIT"};
 int i;
 OSCICN= 0x83;          /*Configure internal oscillator for its maximum frequency 12MHz*/
 XBR0 = 0x01;           /* Route UART0 on P0.4 –TXD& P0.5-RXD - pin */
 XBR1 = 0x40;           /* Enable Crossbar*/
 P0MDOUT = 0x10;        /* P0.4 –TXD - pin configured as Output */
 SCON0=0x00;            /* Select 8 bit UART mode  */
```

UART Transmission Program

```
CKCON = 0x01;                          /* T0/T1 pre-scalar bit SCA1 SCA0=01 to make sysckl/4*/
TH1 = 256-(SYSCLK/BR_UART0/2/4);
                                       /*(SYSCLK/BR_UART0/2/256 < 4) so the prescaler is  by 4 */
                                       /* T1M = 0; SCA1:0 = 01 */
TL1 = TH1;                             /* Init Timer1 */
TMOD = 0x20;                                   /* TMOD: timer 1 in 8-bit auto-reload */
TR1 = 1;                               /* START Timer1 */
while(1)
{
for (i=0;ch[i]!='\0';i++)
      {
            SBUF0 = ch[i];                 /* Load character to be transmiited in SBUF*/
            while(TI0 == 0);
            TI0 = 0;                 /* Clear transmit interrupt flag*/
      }
 }
}
```

UART Reception Algorithm

- Define SYSCLK 12000000
- Define BR_UART0 9600
- Configure internal oscillator for its maximum frequency 12MHz (OSCICN)
- Enable UART through crossbar priority decoder (XBR0)
- Enable crossbar priority decoder (XBR1)
- Configure receive pin as input (P0MDIN)
- Configure Port4 as Digital output
- Configure 8-bit mode and reception enabled (SCON)
- Select prescaler for Timer 1 (CKCON)

UART Reception Algorithm (Contd.)

- Load count for desired baud rate in TH1
- Load TH1 value in TL1
- Configure TMOD for Timer 1 in Mode 2
- Start Timer
- For reception
  - Wait for reception to be over by polling for RI0 flag
  - Send complimented value of SBUF0 register to Port4
  - Clear RI0

UART Reception Program

```
#include "c8051F340.h"
#define SYSCLK 12000000
#define BR_UART0 9600
void main()
{

 OSCICN= 0x83;          /*Configure internal oscillator for its maximum frequency 12MHz*/
 XBR0 = 0x01;           /* Route UART0 on P0.4 –TXD& P0.5-RXD - pin */
 XBR1 = 0x40;           /* Enable Crossbar*/
 P0MDIN = 0x20;         /* P0.5-RXD pin configured as input */
 P4MDOUT =0xFF;         /* P4 as output port to display received data on LEDs*/
 SCON0=0x10;            /*  Enable receiver & select 8 bit UART mode  */
```

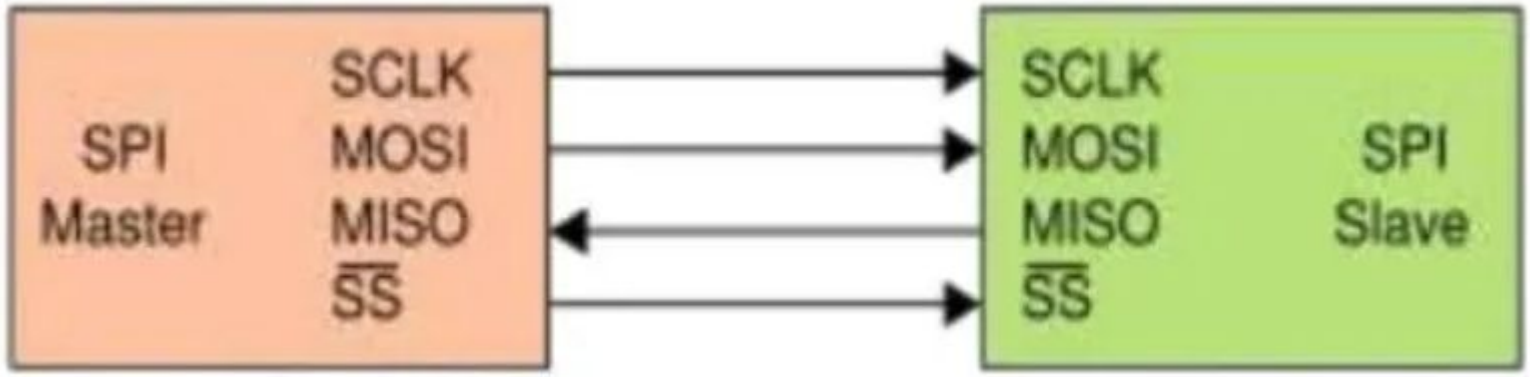
UART Reception Program

```
CKCON = 0x01;                          /* T0/T1 pre-scalar bit SCA1 SCA0=01 to make sysckl/4*/
TH1 =256 -(SYSCLK/BR_UART0/2/4);
                                       /*(SYSCLK/BR_UART0/2/256 < 4) so the prescaler is  by 4 */
                                       /* T1M = 0; SCA1:0 = 01 */


 TL1 = TH1;                            /* Init Timer1*/
 TMOD = 0x20;                          /* TMOD: timer 1 in 8-bit autoreload*/
 TR1 = 1;                              /* START Timer1*/

while(RI0 ==0);
{
        P4=~SBUF0;      /*Display received data on P4 where LED (Common cathode) are connected*/
        RI0 = 0;        /* Clear receive interrupt flag*/
}
}
```

Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI)

- Serial peripheral interface (SPI) is one of the most widely used interfaces between microcontroller and peripheral ICs such as sensors, ADCs, DACs, shift registers, SRAM, EPROM, and others.
- SPI is a synchronous, full duplex Mater Slave based interface.
- The SPI interface can be either 3-wire or 4-wire
- SPI bus consists of a single master and multiple slave devices. But SPI bus can be used in different configurations like a single master and a single slave.



Why SPI communication is used?

- Embedded system engineers use SPI communication protocol when they want to transfer fast data between two digital devices and a small number of input-output pins are available.

- In embedded systems due to constraints of less number of GPIO pins, we use serial peripheral interface to transfer data between two devices by using the minimum number of GPIO pins.

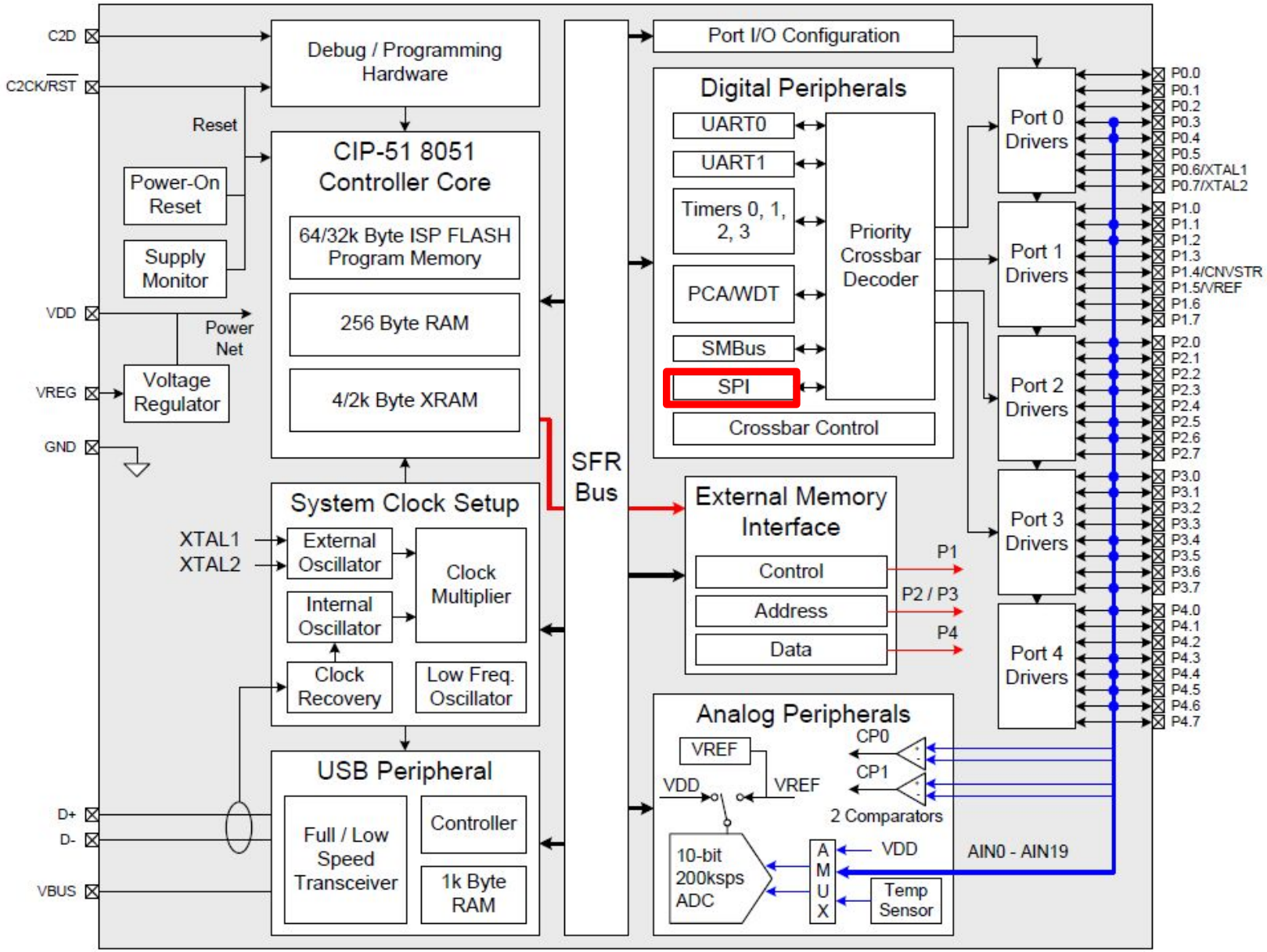
- SPI communication saves us wiring pins and also reduces the cost of hardware.

- This protocol can be implemented very easily and quickly with fewer I/O pins. It has a serial interface and a single master can control multiple slave devices to transfer data between each other.

Serial Communication Buses of C8051F340

- The C8051F340 has
  - Two UARTs (UART0 & UART1)
  - one SPI
  - one SMBus
  - one USB hardware peripherals

SPI of C051F340



Enhanced Serial Peripheral Interface (SPI0)

- The Enhanced Serial Peripheral Interface (SPI0) provides access to a flexible, full-duplex synchronous serial bus.

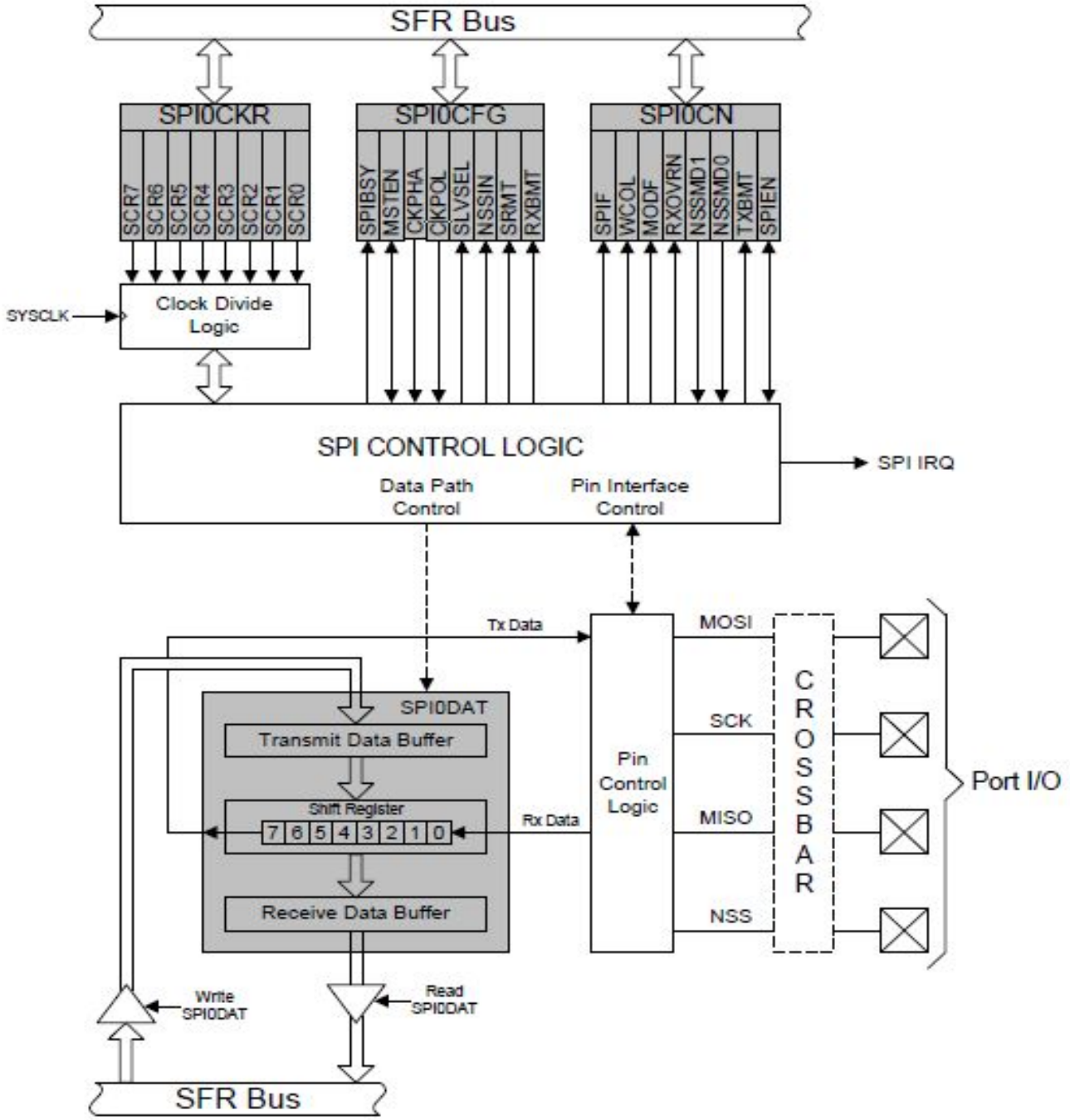
- SPI0 can operate as a master or slave device in both 3-wire or 4-wire modes, and supports multiple masters and slaves on a single SPI bus.

- The slave-select (NSS) signal can be configured as an input to select SPI0 in slave mode, or to disable Master Mode operation in a multi-master environment, avoiding contention on the SPI bus when more than one master attempts simultaneous data transfers.

- NSS can also be configured as a chip-select output in master mode, or disabled for 3-wire operation.

- Additional general purpose port I/O pins can be used to select multiple slave devices in master mode.

SPI Block Diagram



SPI Signal Descriptions

The four signals used by SPI0 are

- **MOSI**
- **MISO**
- **SCK**
- **NSS**

SPI Signal Descriptions

Master Out, Slave In (MOSI)

- The master-out, slave-in (MOSI) signal is an output from a master device and an input to the slave devices.
- It is used to serially transfer data from the master to the slave. This signal is an output when SPI0 is operating as a master and an input when SPI0 is operating as a slave.
- Data is transferred most-significant bit first. When configured as a master, MOSI is driven by the MSB of the shift register in both 3- and 4-wire mode.

Master In, Slave Out (MISO)

- The master-in, slave-out (MISO) signal is an output from a slave device and an input to the master device.
- It is used to serially transfer data from the slave to the master. This signal is an input when SPI0 is operating as a master and an output when SPI0 is operating as a slave.
- Data is transferred most-significant bit first. The MISO pin is placed in a high-impedance state when the SPI module is disabled and when the SPI operates in 4-wire mode as a slave that is not selected.
- When acting as a slave in 3-wire mode, MISO is always driven by the MSB of the shift register.

SPI Signal Descriptions

Serial Clock (SCK)

- The serial clock (SCK) signal is an output from the master device and an input to slave devices.
- It is used to synchronize the transfer of data between the master and slave on the MOSI and MISO lines.
- SPI0 generates this signal when operating as a master. The SCK signal is ignored by a SPI slave when the slave is not selected (NSS = 1) in 4-wire slave mode.

Slave Select (NSS)

The function of the slave-select (NSS) signal is dependent on the setting of the NSSMD1 and NSSMD0 bits in the SPI0CN register. There are three possible modes that can be selected with these bits:

1. **NSSMD[1:0] = 00: 3-Wire Master or 3-Wire Slave Mode:** SPI0 operates in 3-wire mode, and NSS is disabled. When operating as a slave device, SPI0 is always selected in 3-wire mode.

Since no select signal is present, SPI0 must be the only slave on the bus in 3-wire mode. This is intended for point-to-point communication between a master and one slave.

2. **NSSMD[1:0] = 01: 4-Wire Slave or Multi-Master Mode:** SPI0 operates in 4-wire mode, and NSS is enabled as an input. When operating as a slave, NSS selects the SPI0 device. When operating as a master, a 1-to-0 transition of the NSS signal disables the master function of SPI0 so that multiple master devices can be used on the same SPI bus.

3. **NSSMD[1:0] = 1x: 4-Wire Master Mode:** SPI0 operates in 4-wire mode, and NSS is enabled as an output. The setting of NSSMD0 determines what logic level the NSS pin will output. This configuration should only be used when operating SPI0 as a master device.

SPI0 Modes

1. SPI0 Master Mode Operation

A. Multiple-Master Mode
B. 3-Wire Single Master and Slave Mode
C. 4-Wire Single Master Mode and Slave Mode

2. SPI0 Slave Mode Operation

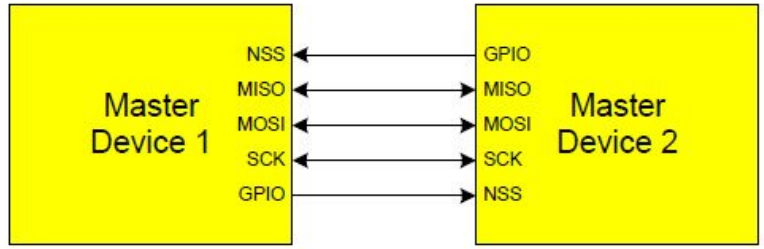


Figure 20.2. Multiple-Master Mode Connection Diagram

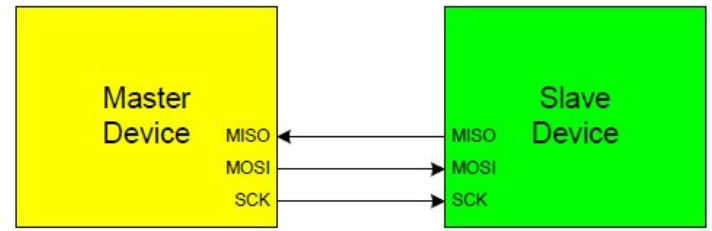


Figure 20.3. 3-Wire Single Master and Slave Mode Connection Diagram

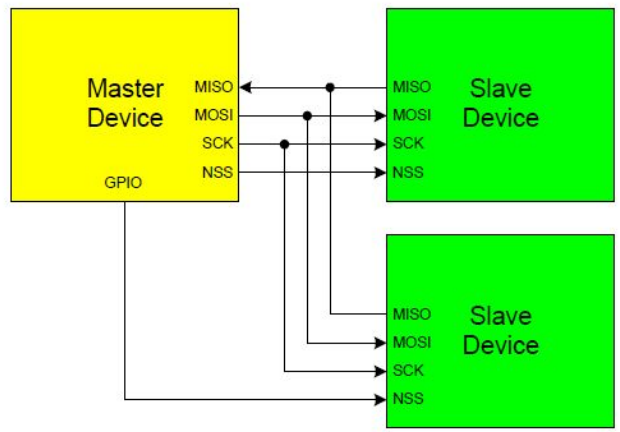
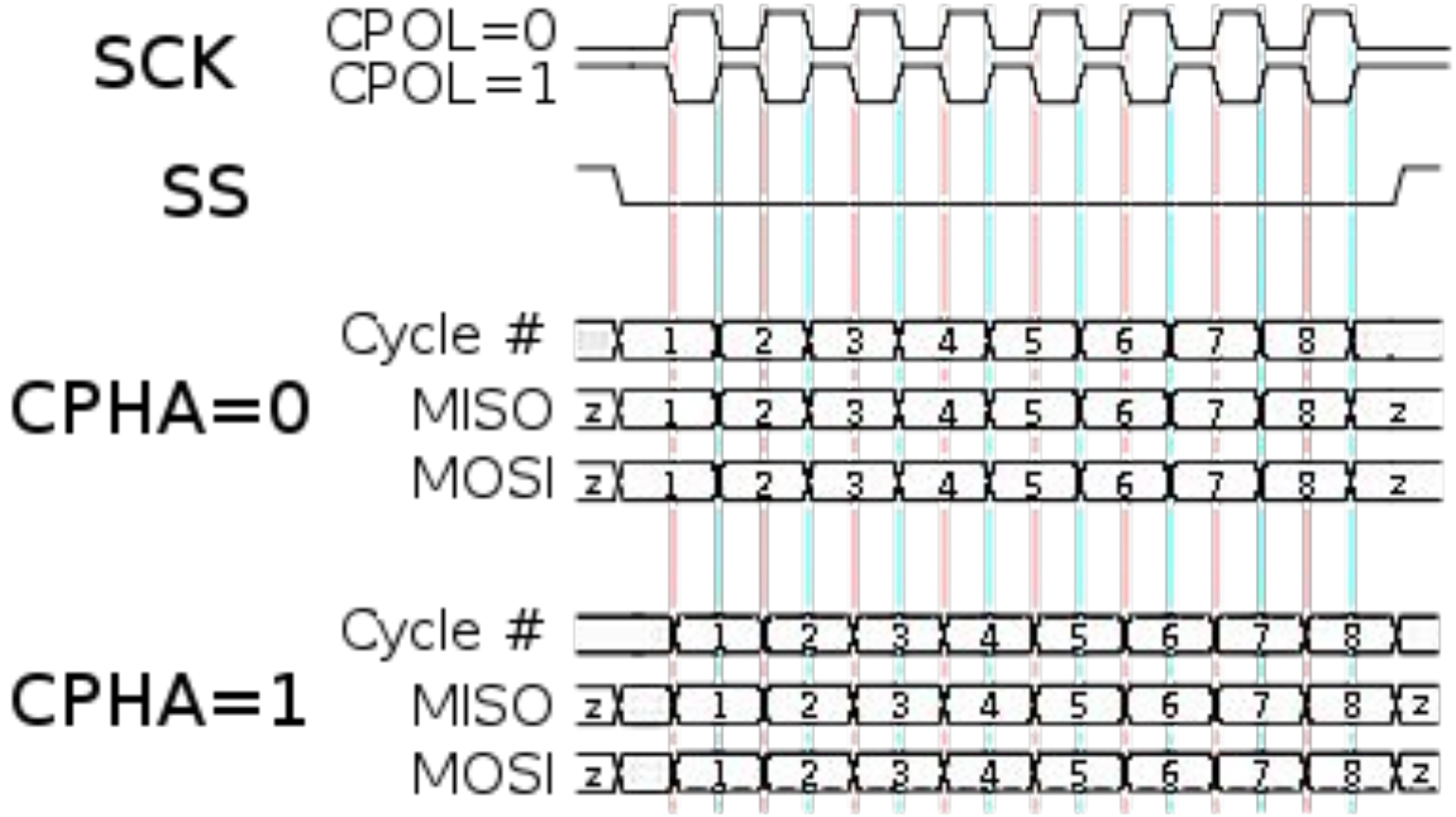


Figure 20.4. 4-Wire Single Master Mode and Slave Mode Connection Diagram

SPI Modes

SPI MODE	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Clock Polarity and Phase



•At CPOL=0 the base value of the clock is zero

For CPHA=0, data are read on the clock's rising edge (low->high transition) and data are changed on a falling edge (high->low transition).

For CPHA=1, data are read on falling edge and data are changed on rising edge.

•At CPOL=1 the base value of the clock is one

For CPHA=0, data are read on falling edge and data are changed on rising edge.

For CPHA=1, data are read on the clock's rising edge and data are changed on a falling edge.

SPI Special Function Registers

SPI0 is accessed and controlled through four special function registers in the system controller:

1. SPI0CN Control Register
2. SPI0DAT Data Register
3. SPI0CFG Configuration Register
4. SPI0CKR Clock Rate Register

SFR- SPI0CFG: SPI0 Configuration

SPI0CFG = _____ //Enable the SPI as a Master
//mode0: CKPHA = '0', CKPOL = '0'

R	R/W	R/W	R/W	R	R	R	R	Reset Value
SPIBSY	MSTEN	CKPHA	CKPOL	SLVSEL	NSSIN	SRMT	RXBMT	00000111
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	

SFR Address: 0xA1

Bit 7: SPIBSY: SPI Busy (read only).
This bit is set to logic 1 when a SPI transfer is in progress (Master or slave Mode).

Bit 6: MSTEN: Master Mode Enable.
0: Disable master mode. Operate in slave mode.
1: Enable master mode. Operate as a master.

Bit 5: CKPHA: SPI0 Clock Phase.
This bit controls the SPI0 clock phase.
0: Data centered on first edge of SCK period.*
1: Data centered on second edge of SCK period.*

Bit 4: CKPOL: SPI0 Clock Polarity.
This bit controls the SPI0 clock polarity.
0: SCK line low in idle state.
1: SCK line high in idle state.

SFR- SPI0CFG: SPI0 Configuration

R	R/W	R/W	R/W	R	R	R	R	Reset Value
SPIBSY	MSTEN	CKPHA	CKPOL	SLVSEL	NSSIN	SRMT	RXBMT	00000111
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	

SFR Address: 0xA1

1: SCK line high in idle state.

Bit 3: SLVSEL: Slave Selected Flag (read only).
This bit is set to logic 1 whenever the NSS pin is low indicating SPI0 is the selected slave. It is cleared to logic 0 when NSS is high (slave not selected). This bit does not indicate the instantaneous value at the NSS pin, but rather a de-glitched version of the pin input.

Bit 2: NSSIN: NSS Instantaneous Pin Input (read only).
This bit mimics the instantaneous value that is present on the NSS port pin at the time that the register is read. This input is not de-glitched.

Bit 1: SRMT: Shift Register Empty (Valid in Slave Mode, read only).
This bit will be set to logic 1 when all data has been transferred in/out of the shift register, and there is no new information available to read from the transmit buffer or write to the receive buffer. It returns to logic 0 when a data byte is transferred to the shift register from the transmit buffer or by a transition on SCK.
NOTE: SRMT = 1 when in Master Mode.

Bit 0: RXBMT: Receive Buffer Empty (Valid in Slave Mode, read only).
This bit will be set to logic 1 when the receive buffer has been read and contains no new information. If there is new information available in the receive buffer that has not been read, this bit will return to logic 0.
NOTE: RXBMT = 1 when in Master Mode.

SFR-SPI0CN: SPI0 Control

SPI0CN = _____; // 4-wire Single Master, SPI enabled

R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	Reset Value
SPIF	WCOL	MODF	RXOVRN	NSSMD1	NSSMD0	TXBMT	SPIEN	00000110
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit Addressable

SFR Address: 0xF8

Bit 7: SPIF: SPI0 Interrupt Flag.
This bit is set to logic 1 by hardware at the end of a data transfer. If interrupts are enabled, setting this bit causes the CPU to vector to the SPI0 interrupt service routine. This bit is not automatically cleared by hardware. It must be cleared by software.

Bit 6: WCOL: Write Collision Flag.
This bit is set to logic 1 if a write to SPI0DAT is attempted when the transmit buffer has not been emptied to the SPI shift register. When this occurs, the write to SPI0DAT will be ignored, and the transmit buffer will not be written. This flag can occur in all SPI0 modes. It must be cleared by software.

Bit 5: MODF: Mode Fault Flag.
This bit is set to logic 1 by hardware (and generates a SPI0 interrupt) when a master mode collision is detected (NSS is low, MSTEN = 1, and NSSMD[1:0] = 01). This bit is not automatically cleared by hardware. It must be cleared by software.

Bit 4: RXOVRN: Receive Overrun Flag (Slave Mode only).
This bit is set to logic 1 by hardware (and generates a SPI0 interrupt) when the receive buffer still holds unread data from a previous transfer and the last bit of the current transfer is shifted into the SPI0 shift register. This bit is not automatically cleared by hardware. It must be cleared by software.

SFR-SPI0CN: SPI0 Control

R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	Reset Value
SPIF	WCOL	MODF	RXOVRN	NSSMD1	NSSMD0	TXBMT	SPIEN	00000110
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit Addressable

SFR Address: 0xF8

be cleared by software.

Bits 3–2: NSSMD1–NSSMD0: Slave Select Mode.
Selects between the following NSS operation modes:
(See **Section "20.2. SPI0 Master Mode Operation" on page 224** and **Section "20.3. SPI0 Slave Mode Operation" on page 226**).
00: 3-Wire Slave or 3-wire Master Mode. NSS signal is not routed to a port pin.
01: 4-Wire Slave or Multi-Master Mode (Default). NSS is always an input to the device.
1x: 4-Wire Single-Master Mode. NSS signal is mapped as an output from the device and will assume the value of NSSMD0.

Bit 1: TXBMT: Transmit Buffer Empty.
This bit will be set to logic 0 when new data has been written to the transmit buffer. When data in the transmit buffer is transferred to the SPI shift register, this bit will be set to logic 1, indicating that it is safe to write a new byte to the transmit buffer.

Bit 0: SPIEN: SPI0 Enable.
This bit enables/disables the SPI.
0: SPI disabled.
1: SPI enabled.

SPI0CKR: SPI0 Clock Rate

SPI0CKR = (SYSCLK/(2*SPI_CLOCK))-1;

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
SCR7	SCR6	SCR5	SCR4	SCR3	SCR2	SCR1	SCR0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	

SFR Address: 0xA2

Bits 7–0: SCR7–SCR0: SPI0 Clock Rate.
These bits determine the frequency of the SCK output when the SPI0 module is configured for master mode operation. The SCK clock frequency is a divided version of the system clock, and is given in the following equation, where *SYSCLK* is the system clock frequency and *SPI0CKR* is the 8-bit value held in the SPI0CKR register.

$$f_{SCK} = \frac{SYSCLK}{2 \times (SPI0CKR + 1)}$$

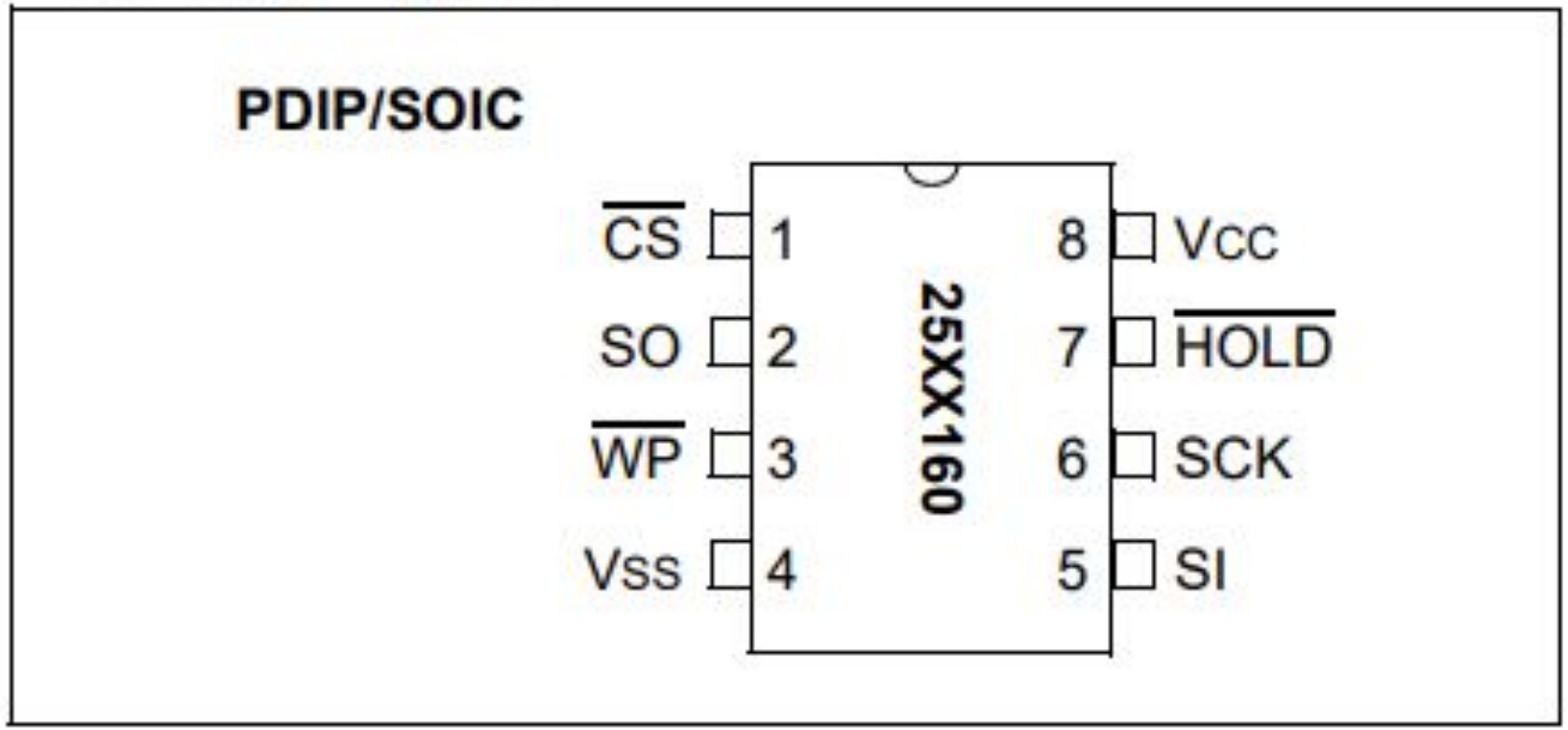
for 0 <= SPI0CKR <= 255

Example: If SYSCLK = 2 MHz and SPI0CKR = 0x04,

$$f_{SCK} = \frac{2000000}{2 \times (4 + 1)}$$

$$f_{SCK} = 200kHz$$

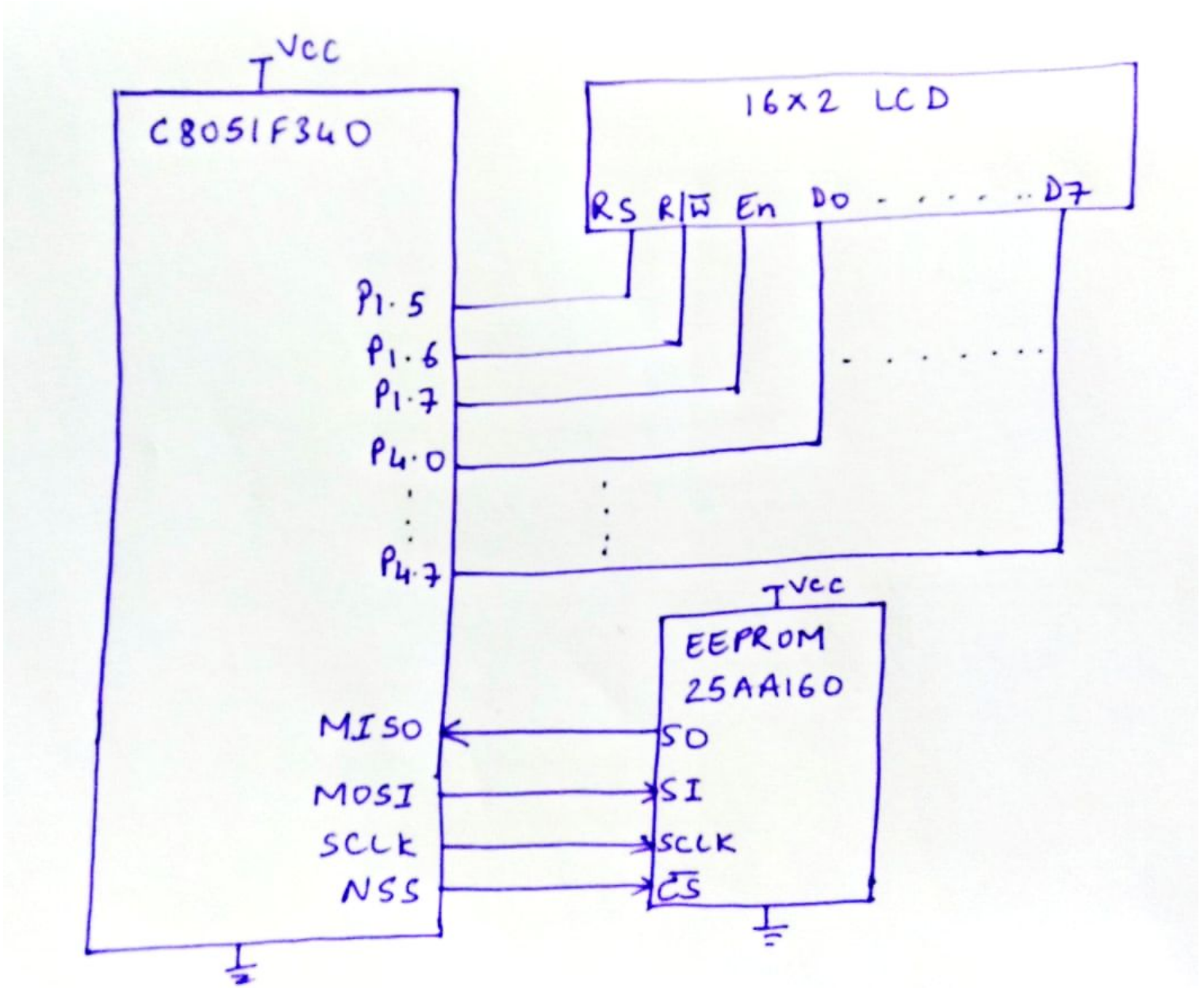
EEPROM 25AA160



Pin Description

Name	PDIP	SOIC	Description
$\overline{CS}$	1	1	Chip Select Input
SO	2	2	Serial Data Output
$\overline{WP}$	3	3	Write-Protect Pin
Vss	4	4	Ground
SI	5	5	Serial Data Input
SCK	6	6	Serial Clock Input
$\overline{HOLD}$	7	7	Hold Input
Vcc	8	8	Supply Voltage

Interfacing Diagram:

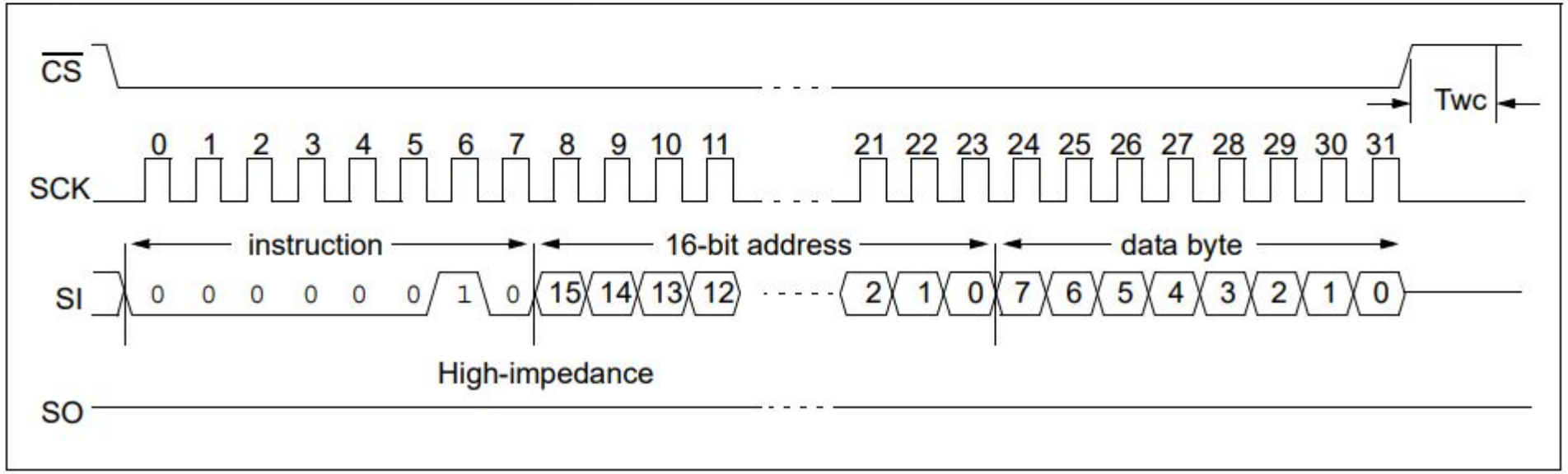


Instruction Set for EEPROM

- #define EEP_WRITE 0x02; /* Write bit of IR   */
- #define EEP_READ   0x03; /* Read bit of IR   */
- #define EEP_WREN   0x06; /* Write Enable Latch */
- #define EEP_WRDI   0x04; /* Write Disable Latch */
- #define EEP_RDSR   0x05; /* Read Status Reg   */
- #define EEP_WRSR   0x01; /* Write Status Reg   */

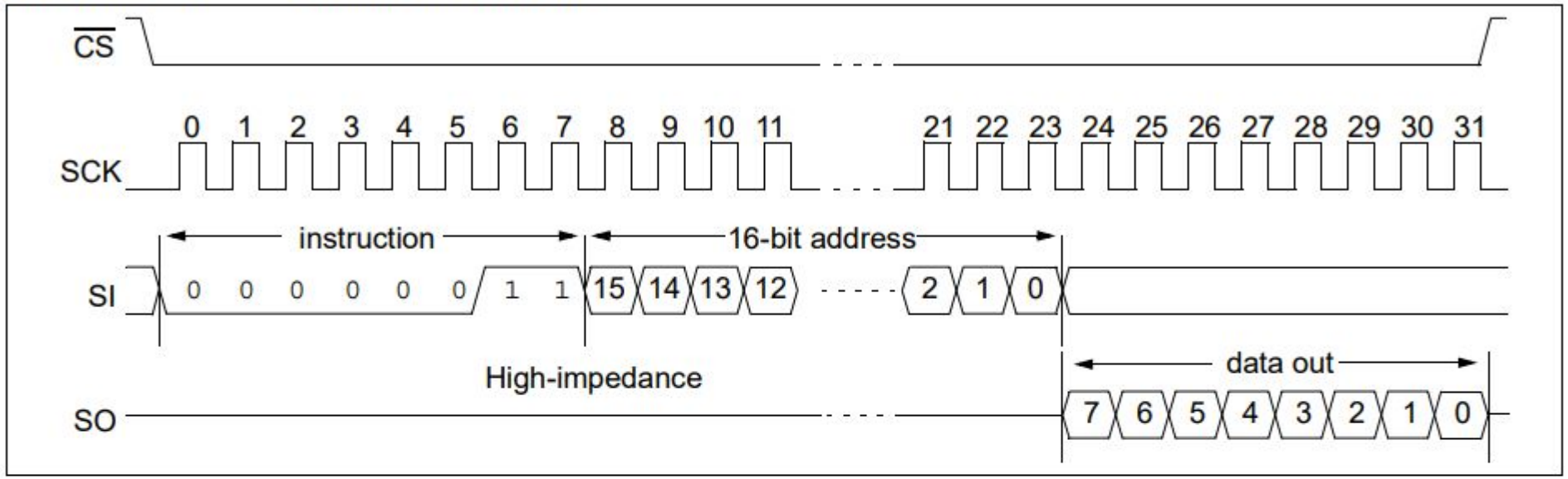
EEPROM Write Sequence

FIGURE 3-2: BYTE WRITE SEQUENCE



EEPROM Read Sequence

FIGURE 3-1: READ SEQUENCE



SPI Initialization

```
void SPI0_Init(void)
{
  SPI0CFG |= 0x40;                  // Enable the SPI as a Master

  SPI0CN  = 0x0D;                   // 4-wire Single Master, SPI enabled

  //SPI clock frequency equation from the datasheet
  SPI0CKR = (SYSCLK/(2*SPI_CLOCK))-1;

  //ESPI0 = 1;                      // Enable SPI interrupts
  SPI0CFG &=~0x30; //mode0: CKPHA = '0', CKPOL = '0'
  //SPI0CFG |= 0x30; //mode3
}
```

Write to EEPROM

```
uchar Spi_Eeprom_Write_Byte(uint16 eep_address, uchar byte_data)
{
  uchar SpiData[4],WriteData,WriteStatus =0;

  SpiData[0] = EEP_WRITE;          /* WRITE IR 8bit msb*/
  SpiData[1] = (uchar)(eep_address>>8);   // 1st byte extract
  SpiData[2] = (uchar) eep_address;       // 2nd byte extract
  SpiData[3] = byte_data;                 /* byte data lsb  */

  WriteData = EEP_WREN;

  NSSMD0 = 0;
  Write_Data_Spi(&WriteData,1);   /* Write Enable Latch  */
  NSSMD0 = 1;

  NSSMD0 = 0;                     /* CS low active  */
  WriteStatus = Write_Data_Spi(SpiData,4); /* Pass value*/
  if(WriteStatus)
  {
       NSSMD0 = 1;      /* CS high inactive */
       return(1);
  }
  else
       return(0);
}
```

Read from EEPROM

```
uint16 Spi_Eeprom_Read_Byte(uint16 eep_address)
{
  uchar SpiData[3],eep_data,ReadStatus =0;

  SpiData[0] = EEP_READ;              /* READ IR 8bit msb       */
  SpiData[1] = (uchar)(eep_address>>8);  // 1st byte extract
  SpiData[2] = (uchar)eep_address;      // 2nd byte extract
  NSSMD0 = 0;
  Write_Data_Spi(SpiData,3);          /* Pass address          */
  ReadStatus = Read_Data_Spi(&eep_data,1); /* Store byte in eep_data  */
  if(ReadStatus)
  {
    NSSMD0 = 1;
    return(eep_data);                 /* Return value          */
  }
  else
       return(0);
}
```

Single character write function

```
schar Write_Data_Spi(schar *Data, uchar Length)
{
  while(Length)
  {
    if(!(SPI0CN&0x40))
    {
      SPI0DAT = *Data++;
      while(!SPIF);
      SPIF = 0;
      Length--;
    }
    else
      return(0);
  }
  return(1);
}
```

Single character read function

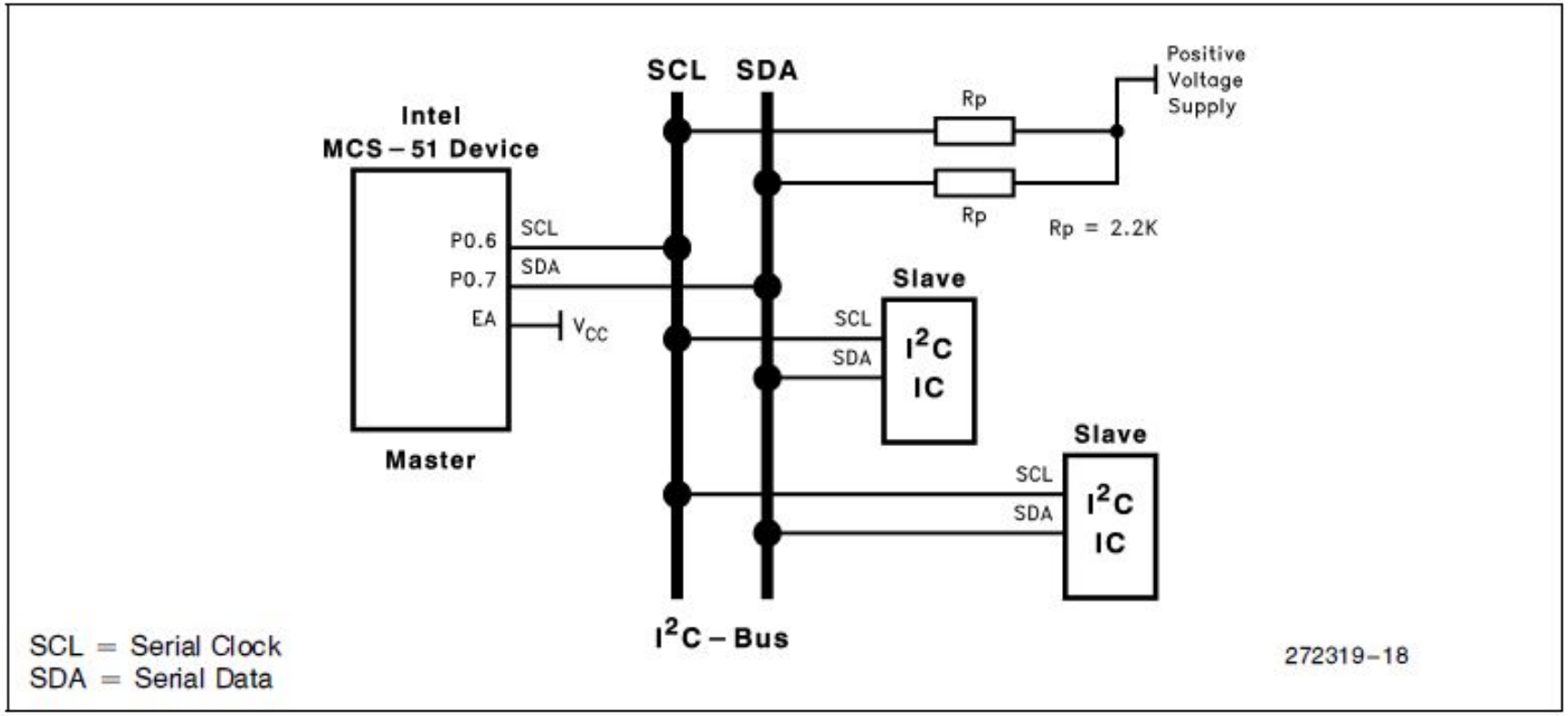
```
schar Read_Data_Spi(schar *Data, uchar Length)
{
  while(Length)
  {
    SPI0DAT = 0xFF;
    while(!SPIF);
    SPIF = 0;
    *Data++ = SPI0DAT;
    Length--;
  }
  return(1);
}
```

I2C Bus Protocol

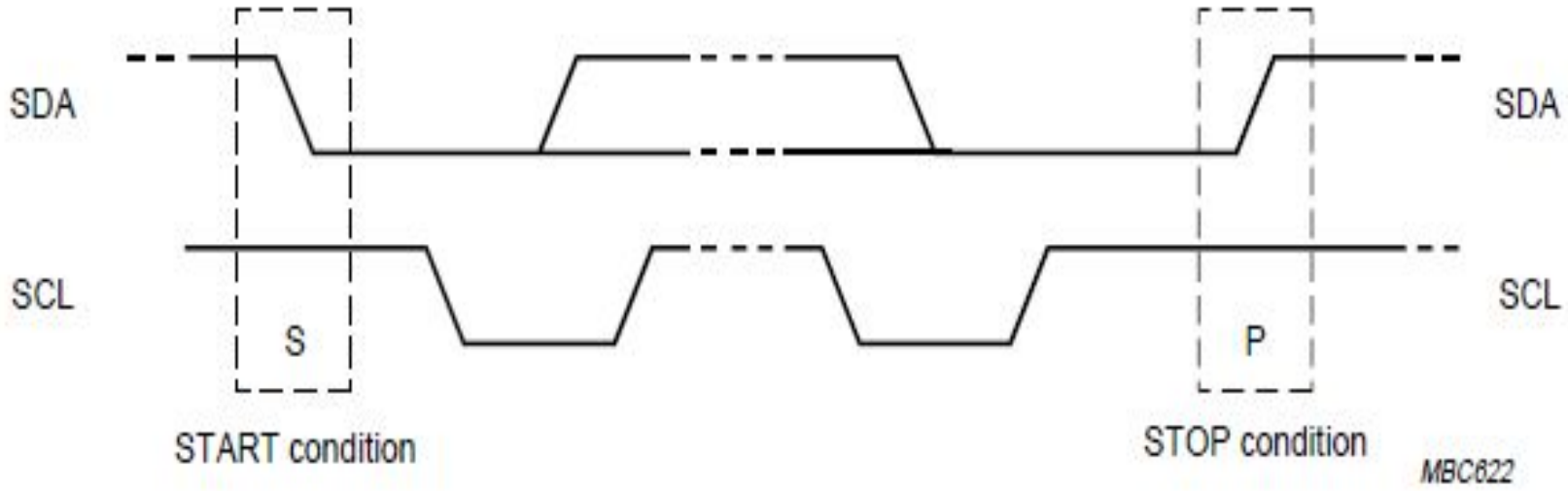
Basic Characteristics

- two-wired bus
- originally to interact within small number of devices
- speeds:
  - 100 kbps (standard mode)
  - 400 kbps (fast mode)
  - 3.4 Mbps (high-speed mode)
- data transfers: serial, 8-bit oriented, bi-directional
- master/slave relationships with multi-master option (arbitration)
- master can operate as transmitter or receiver
- addressing: 7bit or 10bit unique addresses

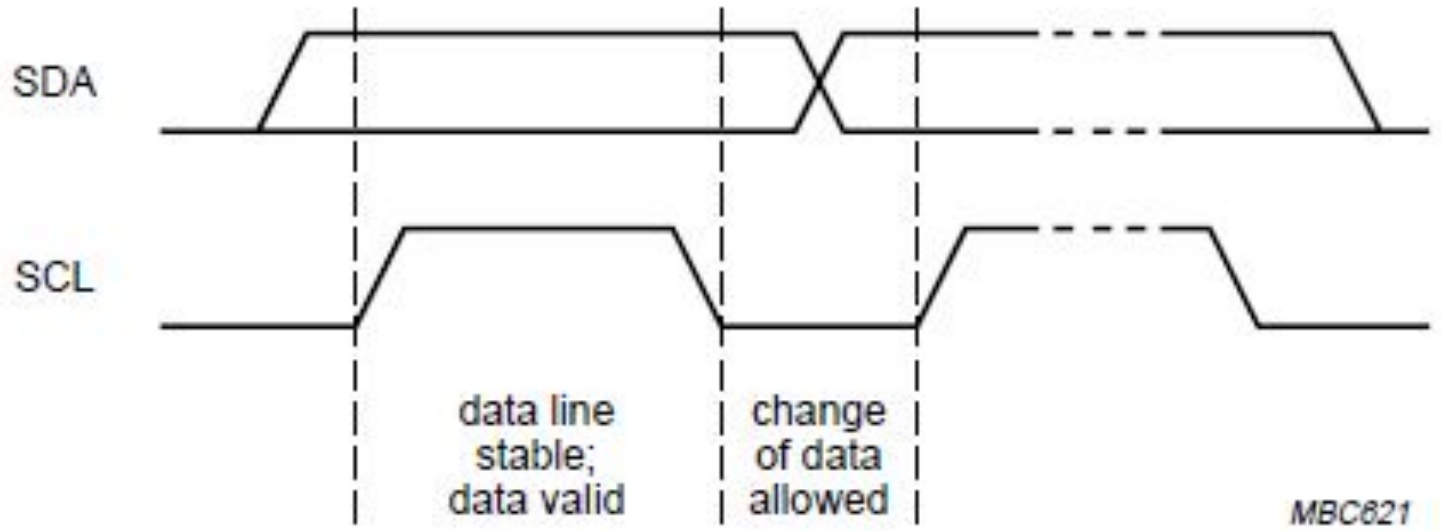
I2C Master/Slave Bus system



Start Stop Condition



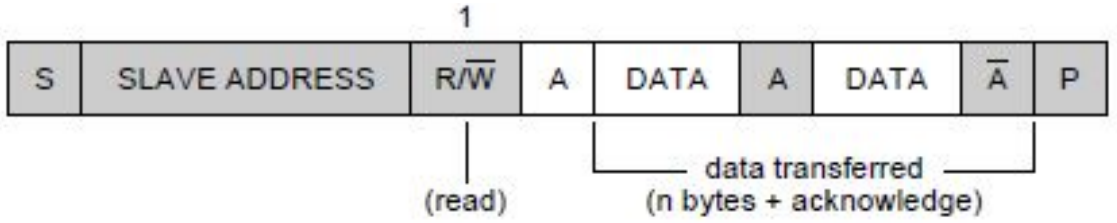
Bit Transfer



Frame Format

master-transmitter

master-receiver (since second byte)



from master to slave

from slave to master

A = acknowledge (SDA LOW)

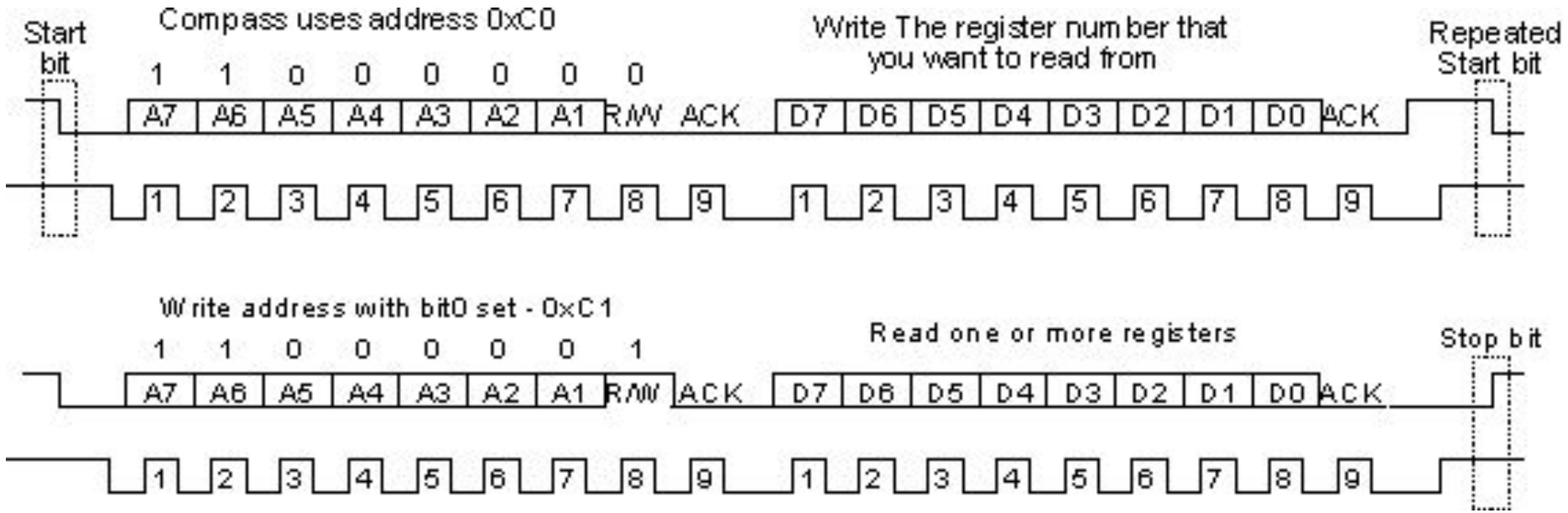
Ā = not acknowledge (SDA HIGH)

S = START condition

P = STOP condition

mbc605

Frame Format



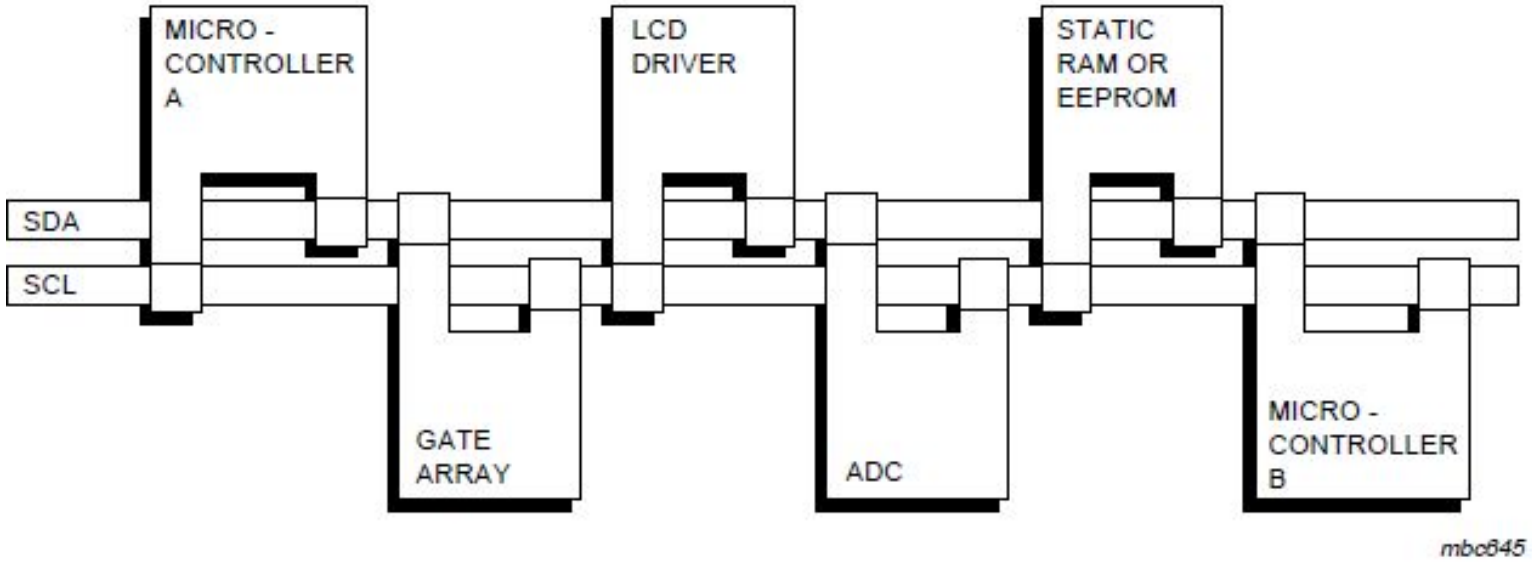
Masters and Slaves

Master device

- controls the SCL
- starts and stops data transfer
- controls addressing of other devices

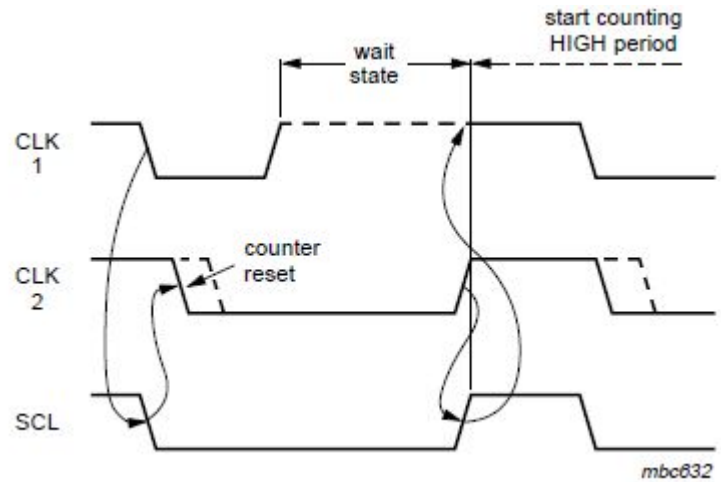
- Slave device
  - device addressed by master
- Transmitter/Receiver
  - master or slave
  - master-transmitter sends data to slave-receiver
  - master-receiver requires data from slave-transmitter

Multiple Masters

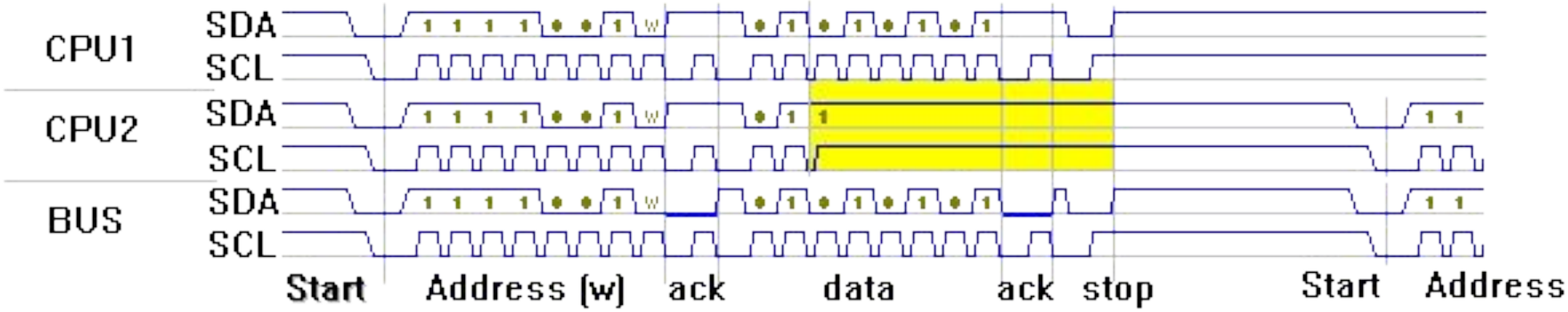


- more bus
- several masters can start frame at once
- synchronization needed on SCL
- arbitration needed on SDA

Clock Stretching



I2C Bus Arbitration



Applications of I^2C

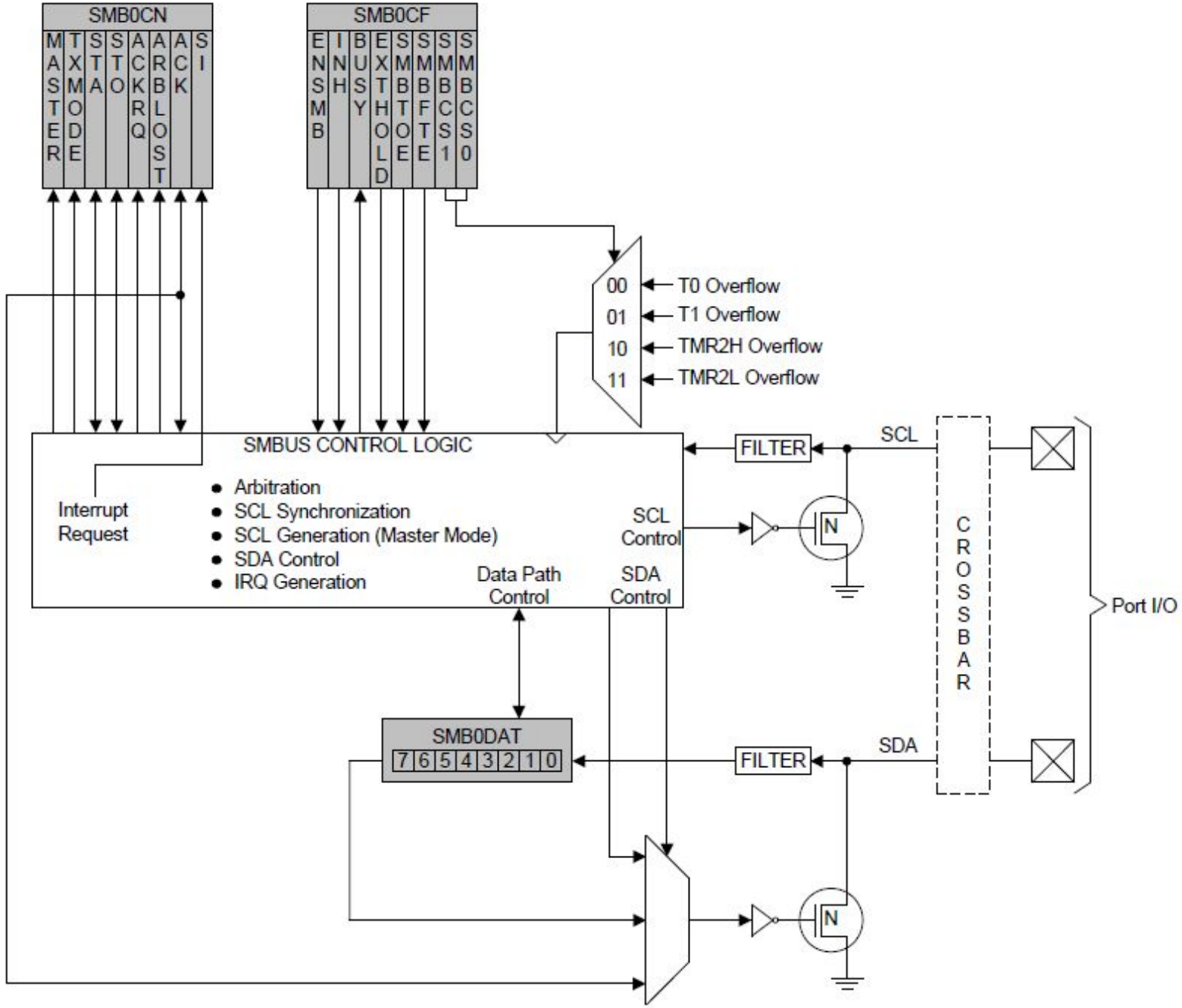
I^2C is appropriate for peripherals where simplicity and low manufacturing cost are more important than speed. Common applications of the I^2C bus are:

- **Reading configuration data from SPD EEPROMs on SDRAM, DDR SDRAM, DDR2 SDRAM memory sticks (DIMM) and other stacked PC boards.**
- **Supporting systems management for PCI cards.**
- **Accessing NVRAM chips that keep user settings.**
- **Accessing low speed DACs and ADCs.**
- **Changing contrast, hue, and color balance settings in monitors (Display Data Channel).**
- **Changing sound volume in intelligent speakers.**
- **Controlling OLED/LCD displays, like in a cellphone.**
- **Reading hardware monitors and diagnostic sensors, like a CPU thermostat and fan speed.**
- **Reading real time clocks.**
- **Turning on and turning off the power supply of system components.**

SMBus in C8051F340

- The SMBus I/O interface is a two-wire, bi-directional serial bus.
- The SMBus is compliant with the System Management Bus Specification, version 1.1, and compatible with the I2C serial bus.
- The SMBus interface may operate as a master and/or slave, and may function on a bus with multiple masters.
- The SMBus provides control of SDA (serial data), SCL (serial clock) generation and synchronization, arbitration logic, and START/STOP control and generation.
- Three SFRs are associated with the SMBus:
  - SMB0CF configures the SMBus; SMB0CN controls the status of the SMBus; and SMB0DAT is the data register, used for both transmitting and receiving SMBus data and slave addresses.

SMBus Block Diagram



SMB0CF: SMBus Clock/Configuration

R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	Reset Value
ENSMB	INH	BUSY	EXTHOLD	SMBTOE	SMBFTE	SMBCS1	SMBCS0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	

SFR Address: 0xC1

Bit7: ENSMB: SMBus Enable.
This bit enables/disables the SMBus interface. When enabled, the interface constantly monitors the SDA and SCL pins.
0: SMBus interface disabled.
1: SMBus interface enabled.

Bit6: INH: SMBus Slave Inhibit.
When this bit is set to logic 1, the SMBus does not generate an interrupt when slave events occur. This effectively removes the SMBus slave from the bus. Master Mode interrupts are not affected.
0: SMBus Slave Mode enabled.
1: SMBus Slave Mode inhibited.

Bit5: BUSY: SMBus Busy Indicator.
This bit is set to logic 1 by hardware when a transfer is in progress. It is cleared to logic 0 when a STOP or free-timeout is sensed.

Bit4: EXTHOLD: SMBus Setup and Hold Time Extension Enable.
This bit controls the SDA setup and hold times according to.
0: SDA Extended Setup and Hold Times disabled.
1: SDA Extended Setup and Hold Times enabled.

SMB0CF: SMBus Clock/Configuration

Bit3: SMBTOE: SMBus SCL Timeout Detection Enable.
This bit enables SCL low timeout detection. If set to logic 1, the SMBus forces Timer 3 to reload while SCL is high and allows Timer 3 to count when SCL goes low. Timer 3 should be programmed to generate interrupts at 25 ms, and the Timer 3 interrupt service routine should reset SMBus communication.

Bit2: SMBFTE: SMBus Free Timeout Detection Enable.
When this bit is set to logic 1, the bus will be considered free if SCL and SDA remain high for more than 10 SMBus clock source periods.

Bits1–0: SMBCS1-SMBCS0: SMBus Clock Source Selection.
These two bits select the SMBus clock source, which is used to generate the SMBus bit rate. The selected device should be configured according to Equation 17.1.

SMBCS1	SMBCS0	SMBus Clock Source
0	0	Timer 0 Overflow
0	1	Timer 1 Overflow
1	0	Timer 2 High Byte Overflow
1	1	Timer 2 Low Byte Overflow

SMB0CN Control Register

R	R	R/W	R/W	R	R	R/W	R/W	Reset Value
MASTER	TXMODE	STA	STO	ACKRQ	ARBLOST	ACK	SI	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit Addressable

SFR Address: 0xC0

Bit7: MASTER: SMBus Master/Slave Indicator.
This read-only bit indicates when the SMBus is operating as a master.
0: SMBus operating in Slave Mode.
1: SMBus operating in Master Mode.

Bit6: TXMODE: SMBus Transmit Mode Indicator.
This read-only bit indicates when the SMBus is operating as a transmitter.
0: SMBus in Receiver Mode.
1: SMBus in Transmitter Mode.

Bit5: STA: SMBus Start Flag.
Write:
0: No Start generated.
1: When operating as a master, a START condition is transmitted if the bus is free (If the bus is not free, the START is transmitted after a STOP is received or a timeout is detected). If STA is set by software as an active Master, a repeated START will be generated after the next ACK cycle.
Read:
0: No Start or repeated Start detected.
1: Start or repeated Start detected.

SMB0CN Control Register

Bit3:	ACKRQ: SMBus Acknowledge Request This read-only bit is set to logic 1 when the SMBus has received a byte and needs the ACK bit to be written with the correct ACK response value.
Bit2:	ARBLOST: SMBus Arbitration Lost Indicator. This read-only bit is set to logic 1 when the SMBus loses arbitration while operating as a transmitter. A lost arbitration while a slave indicates a bus error condition.
Bit1:	ACK: SMBus Acknowledge Flag. This bit defines the out-going ACK level and records incoming ACK levels. It should be written each time a byte is received (when ACKRQ=1), or read after each byte is transmitted. 0: A "not acknowledge" has been received (if in Transmitter Mode) OR will be transmitted (if in Receiver Mode). 1: An "acknowledge" has been received (if in Transmitter Mode) OR will be transmitted (if in Receiver Mode).
Bit0:	SI: SMBus Interrupt Flag. This bit is set by hardware under the conditions listed in Table 17.3. SI must be cleared by software. While SI is set, SCL is held low and the SMBus is stalled.

Comparison of Serial Interfaces

	Synchronous		Asynchronous
Peripheral	**SPI**	**I²C**	**UART**
Max Bit Rate	10Mbit/s	1Mbit/s	500kbit/s
Max Bus Size	Limited by no. of pins	128 devices	Point to point (RS232) 256 devices (RS485)
Number of pins	3 + n	2	2
Pros	Simple, low cost, high speed	Small pin count, allows multiple masters	Longer distance, improved noise immunity (requires transceivers)
Cons	Single master, short distance	Slowest, short distance	Requires accurate clock frequency
Typical Application	Direct connection to ASICs and other peripherals on same PCB	Bus connection with peripherals on same PCB	Interface with terminals, personal computers and other data acquisition systems
Examples	Serial EEPROMs (25CXXX series), MCP320X A/D converter, ENC28J60 Ethernet controller, MCP251X CAN controller...	Serial EEPROMs (24CXXX series), MCP98XX temperature sensors, MCP322x A/D converters...	RS232, RS422, RS485, LIN bus, MCP2550 IrDA interface...

Comparison of various 8-bit microcontrollers

Parameter	8951	PIC	AVR	ARM
Bus Width	8bit	8/16/32bit	8/32	32/64 bit
Communication protocol	UART,SPI, I2C	UART,SPI, I2C, CAN, Ethernet	UART,SPI, I2C, CAN, Ethernet, USB	UART,SPI, I2C, CAN, Ethernet, USB, IrDA, SAI (Serial Audio Interface
Speed	12 Clock/ Instruction cycle	4 Clock/ Instruction cycle	1 Clock/ Instruction cycle	1 Clock/ Instruction cycle
Architecture	CISC	Some features of RISC	RISC	RISC
Memory Architecture	Harvard	Harvard	Modified Harvard	Modified Harvard
Families	8051 variants	PIC16, PIC17, PIC18,PIC24, PIC32	Tiny, Mega, XMaga, Special purpose AVR	ARM4, 5, 6, 7, 9 etc.
Power Consumption	Average	Low	Low	Low

- Case Study-
  - Data Acquisition System
  - Traffic Light Controller

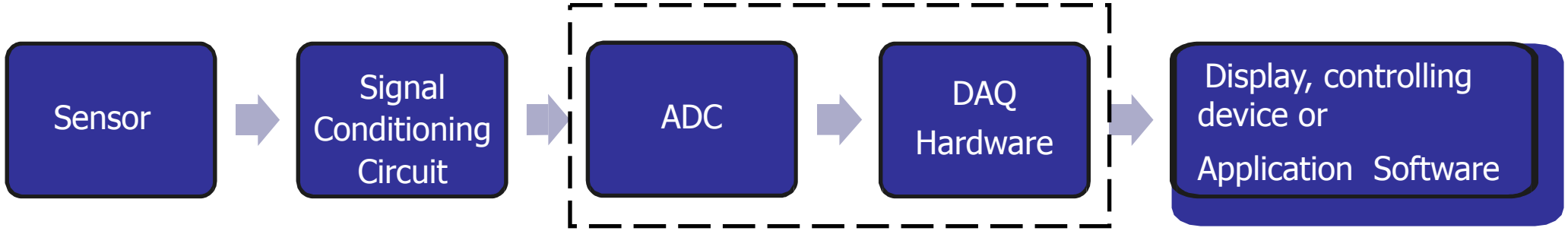
Data Acquisition System

- Data acquisition systems, as the name implies, are products or processes used to collect information to document or analyze some physical phenomenon.

- The purpose of any DAQ system is to gather useful measurement data for characterization, monitoring or control.

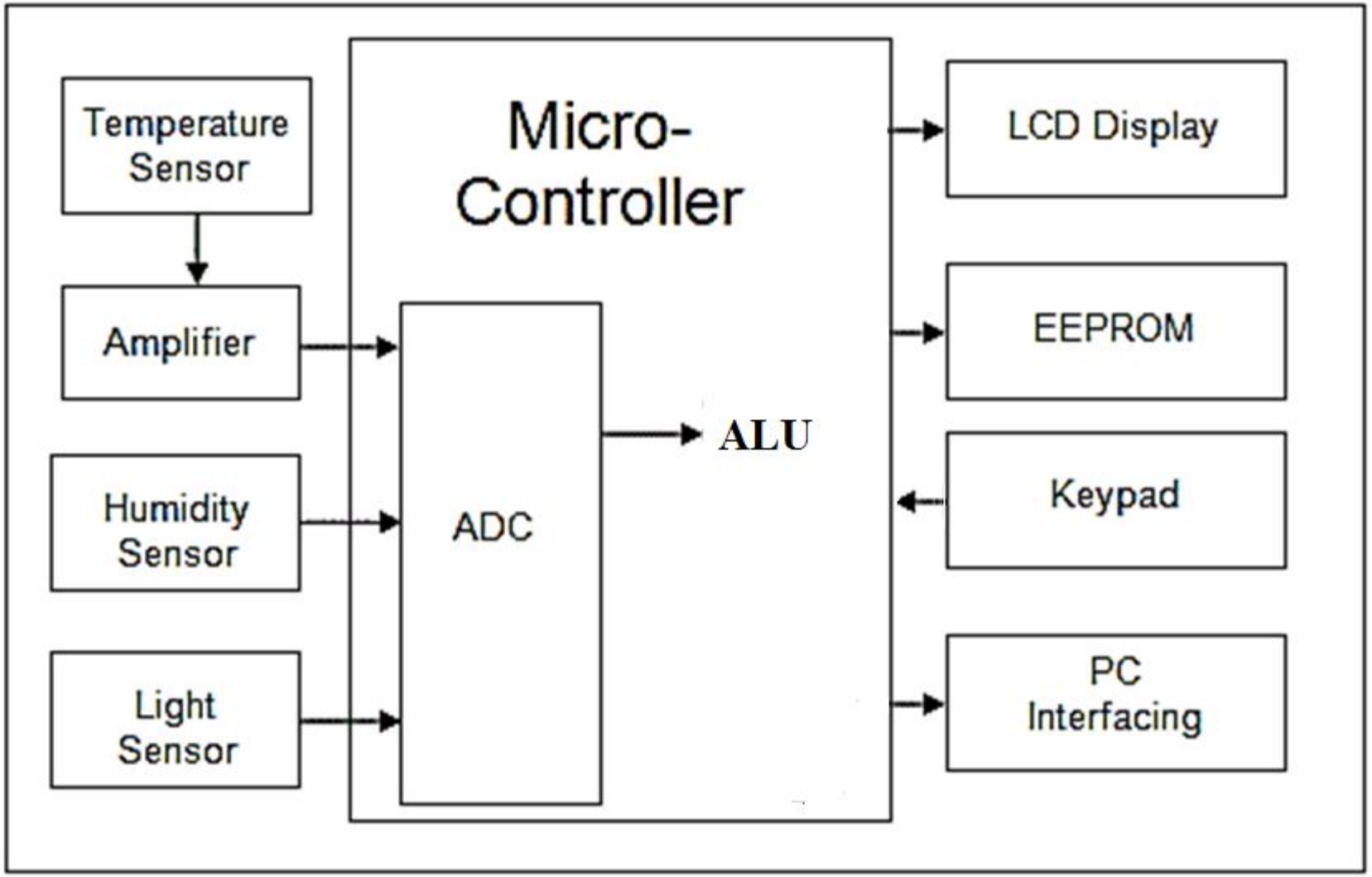
- Data acquisition systems typically involve the conversion of analog input into digital values for processing by microcontroller.

Block Diagram of Data Acquisition System



- Key Components
  - Sensor
  - Signal Conditioning Circuit
  - ADC
  - DAQ Hardware
    - Microcontroller / PCI Cards / Processors
  - Display
    - LCD Display / Seven Segment / TFT etc.
  - Application Software on PC : LabView / MATLAB etc.

Typical Data Acquisition System



Sensor and Transducer

- A sensor is a device that detects changes and events in a physical stimulus and provides a corresponding output signal that can be measured and/or recorded. Here, the output signal can be any measurable signal and is generally an electrical quantity.

- Transducers are the devices that convert energy in one form into another form. Generally the energy is in the form of a signal. Transducer is a term alternatively used for the sensor.

Sensors

Mechanical Sensors

- Pressure sensor – measures pressure
- Barometer – measures atmospheric pressure
- Altimeter – measures the altitude of an object above a fixed level
- Liquid flow sensor – measures liquid flow rate
- Gas flow sensor – measures velocity, direction, and/or flow rate of a gas
- Accelerometer – measures acceleration
- Thermal Sensors
- Thermometer – measures absolute temperature
- Calorimeter – measures the heat of chemical reactions or physical changes and heat capacity

Physical Phenomena & Sensors

- Temperature: Thermocouple, Thermistor, RTD, Fibre optics
- Strain: Strain Gauge, Load Cell
- Sound: Condenser, Microphone, Piezoelectric
- Vibration: Accelerometer, Proximity Probe, LVDT, Variable Reluctance
- Position and Displacement: LVDT, Optical Encoders, Hall Effect Sensor, Potentiometers
- Pressure: Absolute, Gauge, Vaccume
- Force: Beam and S type strain gauge

Sensor Selection Criteria

- **Type of Sensing:** The parameter that is being sensed like temperature, pressure, humidity, distance, etc.
- **Operating Principle:** The principle of operation of the sensor.
- **Power Consumption:** The power consumed by the sensor will play an important role in defining the total power of the system.
- **Accuracy:** The accuracy of the sensor is a key factor in selecting a sensor.
- **Environmental Conditions:** The conditions in which the sensor is being used will be a factor in choosing the quality of a sensor.
- **Cost:** Depending on the cost of application, a low cost sensor or high cost sensor can be used.
- **Resolution and Range:** The smallest value that can be sensed and the limit of measurement are important.
- **Calibration and Repeatability:** Change of values with time and ability to repeat measurements under similar conditions.

ADC Selection Consideration (If external ADC interfaced)

PRIMARY

- What is the required level of system accuracy?
- How many bits of resolution are required?
- What is the nature of the analog input signal?
- How fast must the converter operate (conversion speed)?
- What are the environmental conditions?
- Is a track-and-hold circuit required?

ADC Selection Consideration (If external ADC interfaced)

SECONDARY

- Does the system have multiple channels?
- Should the reference be internal or external?
- What are the drive amplifier requirements?
- What are the digital interface requirements?
- What type of digital output format is required?
- What are the timing conditions?

Need of Signal Conditioner

- The electrical signals generated by the transducers/sensors must be optimized for the input range of the ADC.
- Signal conditioning is used to amplify, attenuate, shape, or isolate signals from transducers/ sensor before they are sent to the measurement hardware i.e. ADC.
- Signal conditioning converts the signal to a form that is better measured by the system, or in some cases, makes it possible to measure the signal at all.

Kinds of Signal Conditioning

- Amplification
  - Instrumentation Amplifiers
- Conversion (Sensor Output to Voltage)
  - I to V, F to V etc.
- Filtering
  - Attenuation
- Isolation
- Linearization
- Multiplexing
- Excitation
- Circuit Protection

Selection of Microcontroller

- CPU Architecture
  - Harvard or Princeton
- ALU : 8-bit, 16-bit, 32-bit
- Clock Speed
- Instruction Set : RISC or CISC
- Memory
  - External Memory Interface capability
  - Internal Flash Memory
- Cache Memory
- Memory Management Unit

Selection of Microcontroller

- I/O Ports
- Serial Communication Interface
  - UART / SPI / I2C / 1-wire
  - Harvard or Princeton
- On-chip ADC
  - SAR or Sigma Delta
- On-chip Timer/ Counter
- PWM Module
- Advanced Communication : USB,Ethernet,CAN
- Debugging Facilities : JTAG, SWD, ICE

Selection of Microcontroller

- Easily availability of Development tools
  - IDE
  - Compiler
  - Programmer
  - Debugger
- Cost and Availability of Microcontrollers
- Ease of integration
- Major hardware blocks required
- Easily availability of Testing Facilities

Application Areas of Data Acquisition System

- Process Monitoring
- Wireless Sensor Network
- Internet of Things
- Test and Measurement
- Furnace Control
- Green House/poly house
- Resource Management
- Automotive Electronics
- Robotic Applications

Performance Parameters of Data Acquisition System

- Resolution
- Accuracy
- Number of Channels supported
- Communication **Protocols**
- Communication Speed
- Portability
- Flexibility / Scalability

Data Acquisition System - Example : To measure room temperature

Sensor selection: LM35

Temperature range: 5 degree to 50degree

Specifications: Refer datasheet

LM35 Regulator Features:

- Can measure temperature ranging from -55°C to 150°C
- Output voltage is directly proportional (Linear) to temperature (i.e.) there will be a rise of 10mV (0.01V) for every 1°C rise in temperature.
- ±0.5°C Accuracy
- Drain current is less than 60uA
- Low cost temperature sensor
- Small and hence suitable for remote applications
- Available in TO-92, TO-220, TO-CAN and SOIC package

Calculations

LM35 resolution is 10 mV per 1°C

So, sensor output range: 50mV to 500mV , considering the temperature range to be sensed as 5°C to 50°C

Output of sensor = Input to ADC

So, Input range to ADC 50mV to 500mV

If Vref of ADC is 5V then 500mV has to be amplified to 5V.

So, Amplification factor will be 10

Need to design amplifier with gain of 10

Now, for example for 1°C output of sensor = 1*10 =10mV

So, input to ADC = 10mV*10 = 100mV=0.1V

ADC0H:ADC0L= (Vin / Vref)*1024 = 0.1/5*1024=20.48=20.5

Now, for example for 20°C output of sensor = 20*10mV =200mV

So, input to ADC = 200mV*10 = 2V

ADC0H:ADC0L= ____

ADC Calculations

If we select C8051F340 and program on chip 10 bit ADC for this application, we need to read the temperature.

So read Dout from ADC and convert it in temperature.

ADC Output Register (Dout),

ADC0H:ADC0L= (Vin / Vref)*1024 = 00000001:10011010

Convert this value to Dout value as,

$$Dout = [((B1 \text{ of } ADC0H)*2^9)+(B0 \text{ of } ADC0H)*2^8))+ (ADC0L)]_{Hex}$$

For Example:

$$Dout = [(100) + (9A)] = (19A)_{Hex}$$

$$Dout = (HexToDecimal(Dout)) = (410)_{Dec}$$

$$Dout = Dout/20.5 = (20)_{Dec} = 20°C$$

Finally display it on LCD by representing this value as char and by separating the digits and adding the unit.

ADC Calculations

If we select C8051F340 and program on chip 10 bit ADC for this application, we need to read the temperature _____

So read Dout from ADC and convert it in temperature.

ADC Output Register (Dout),

ADC0H:ADC0L= (Vin / Vref)*1024 = 00000010:01100110

Convert this value to Dout value as,

Dout =

For Example:

Dout =

Dout =

Dout =

Finally display it on LCD by representing this value as char and by separating the digits and adding the unit.

ADC Calculations

If we select C8051F340 and program on chip 10 bit ADC for this application, we need to read the temperature 30°C

So read Dout from ADC and convert it in temperature.

ADC Output Register (Dout),

ADC0H:ADC0L= (Vin / Vref)*1024 = 00000010:01100110

Convert this value to Dout value as,

$$Dout = [((B1 \text{ of } ADC0H)*2^9)+(B0 \text{ of } ADC0H)*2^8))+ (ADC0L)]_{Hex}$$

For Example:

$$Dout = [200+66] =(266)_{Hex}$$

$$Dout = (HexToDecimal(Dout)) = (614)_{Dec}$$

$$Dout = Dout/20.5 = (30)_{Dec} = 30°C$$

Finally display it on LCD by representing this value as char and by separating the digits and adding the unit.

ADC Calculations

If we select C8051F340 and program on chip 10 bit ADC for this application, we need to read the temperature _____

So read Dout from ADC and convert it in temperature.

ADC Output Register (Dout),

ADC0H:ADC0L= (Vin / Vref)*1024 = 00000010:11001101

Convert this value to Dout value as,

Dout =

For Example:

Dout =

Dout =

Dout =

Finally display it on LCD by representing this value as char and by separating the digits and adding the unit.

ADC Calculations

If we select C8051F340 and program on chip 10 bit ADC for this application, we need to read the temperature 35°C

So read Dout from ADC and convert it in temperature.

ADC Output Register (Dout),

ADC0H:ADC0L= (Vin / Vref)*1024 = 00000010:11001101

Convert this value to Dout value as,

$$Dout = [((B1 \text{ of } ADC0H)*2^9)+(B0 \text{ of } ADC0H)*2^8))+ (ADC0L)]_{Hex}$$

For Example:

$$Dout = [200+CD]=(2CD)_{Hex}$$

$$Dout = (HexToDecimal(Dout)) = (\ 717)_{Dec}$$

$$Dout = Dout/20.5 = (35)_{Dec} = 35°C$$

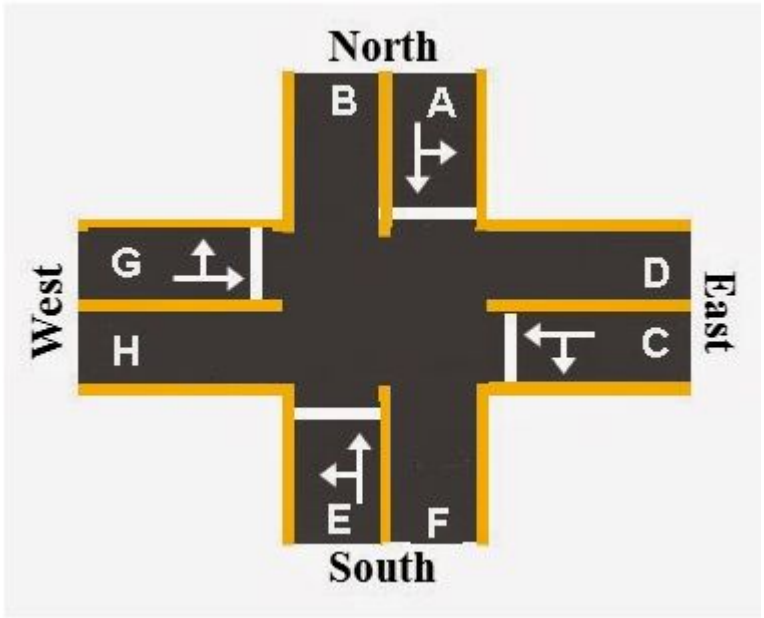
Finally display it on LCD by representing this value as char and by separating the digits and adding the unit.

Traffic Light Controller

- Traffic lights are the devices that are placed on the intersection points and employed to control the flow of traffic on the road.
- Several attempts have been made to make traffic light's sequence dynamic so that these traffic lights operate according to the current volume of the traffic (traffic density based)
- Challenges:
  - traffic density
  - improving the safety
  - improving efficiency of vehicles
  - the limitation in altering the transportation infrastructure
  - treats a emergency vehicles means priority to ambulance, fire brigade or V.I.P vehicles.

Case Study - Traffic Light Controller

Paper design for roadways and junction



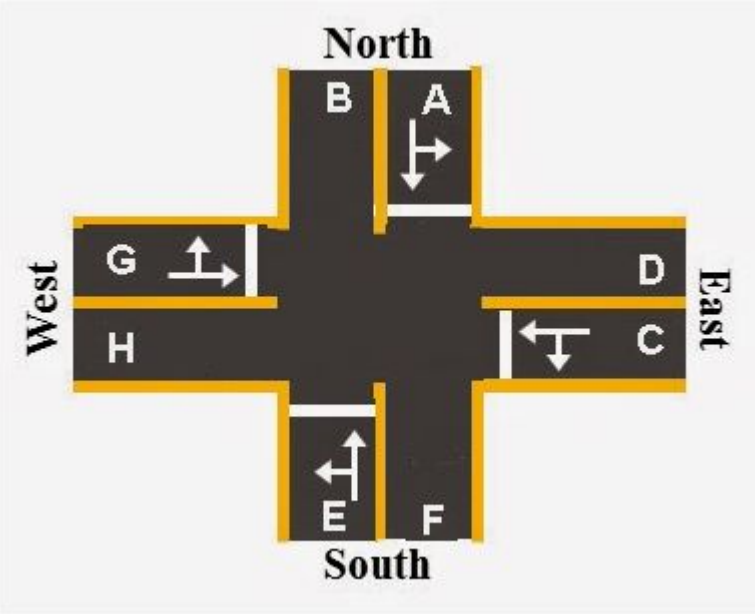
PHASE I-

- Initially Vehicle from A needs to travel to F and from E to B roads.
- So in the first Phase forward green signal in A and E permits vehicles to pass through while East and west roads are stopped by red signal.

PHASE II-

- Phase II permits the vehicle to pass from G to D and from C to H roads.
- Traffic flow from rest of the two roads North and south was stopped by means of Red signal.

Paper design for roadways and junction



PHASE III-

- Phase three permits traffic flow in the left directions from A to D and from E to H.
- Traffic flow in East and west are stopped by means of red signal.

PHASE IV-

- Phase four permits traffic flow from C to F and from G to B.
- Traffic flow in the North and south are stopped by means of red signal.
- The cycle repeats again from Phase I to Phase IV and thus the traffic is regulated.

Approaches

- Following a predetermined timing circuit (independently on timers)
- Design sensor based signaling (sensors placed on, over, near, or even under, the roadway)
- A microcontroller/computer-controlled system
- Image Processing

Sensors

- IR Transmitter and Receiver
- Weight sensors
- Wireless sensors
- Camera

Microcontroller Programming

- PORT
- Timer
- ADC
- Communication Protocols

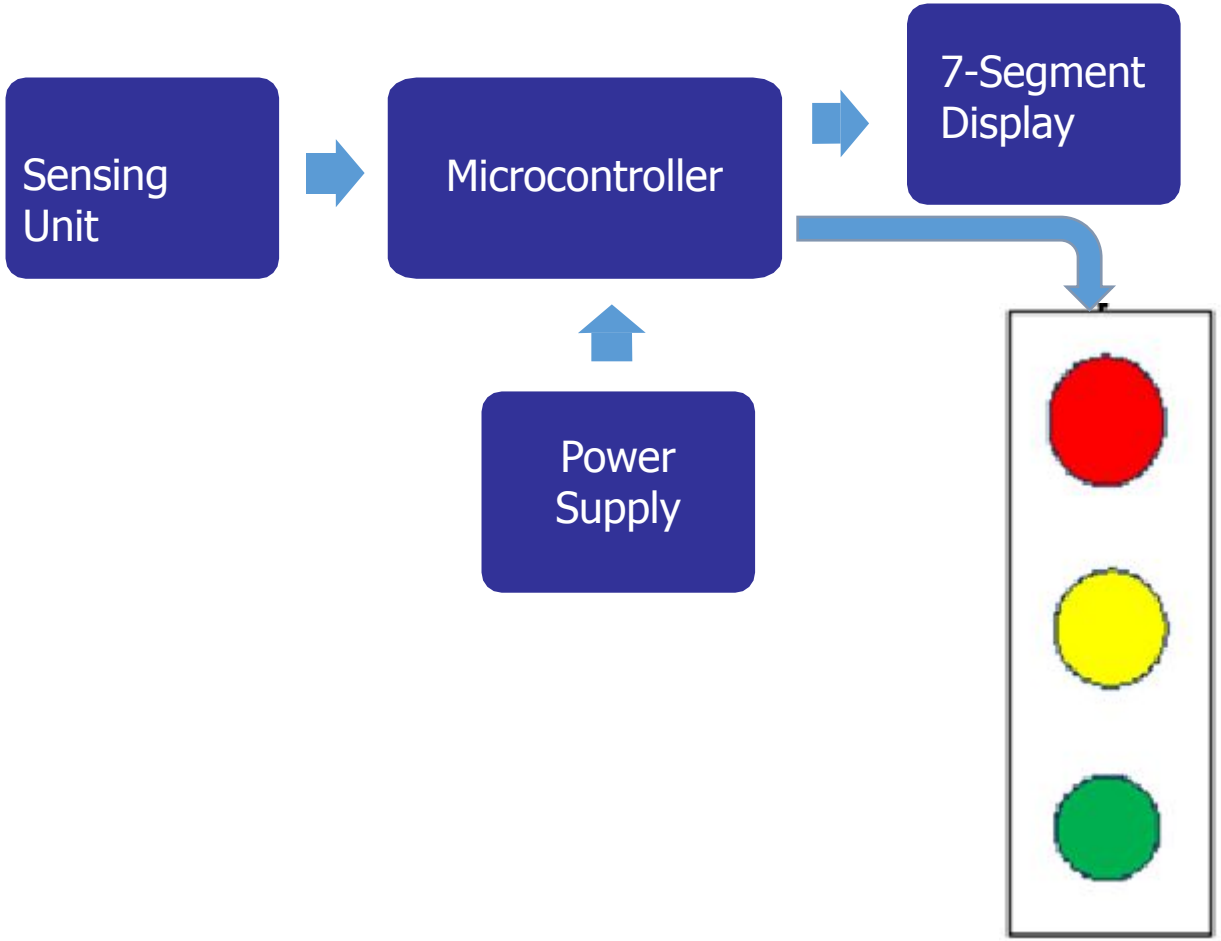
Display

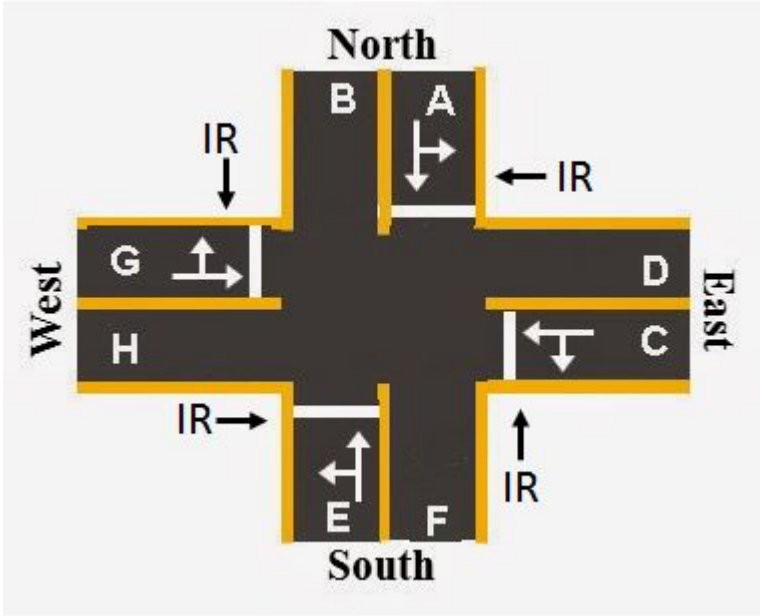
- 7-Segment Display
- LEDs

Note: Selection of sensors and microcontroller is depend on the design and the best reference is the datasheet.

Microcontroller Based Traffic Light Controller System

Hardware Design Consideration





Software Design Consideration

- Algorithm
- Programming
- IDE
- Testing