**CHAPTER 03:**

**I/O PORTS, TIMERS/COUNTERS,INTERRUPTS AND SERIALCOMUNICATION**

**PROGRAMMING**

**Preview:**

In this chapter students will know how to configure the different ports as input or output as P0, P1, P2, and P3. Port structure and how to program different using assembly and ‘C’. Then students can also learn timing and counting programs using assembly and ‘C’. And Interrupt programming using different interrupts that is internal interrupts as well as external interrupts.

**3.1 Port Structure and Simple I/O port programming**

Each port of 8051 has bidirectional capability. Port 0 is called 'true bidirectional port' as it floats (tristated) when configured as input. Port-1, 2, 3 are called 'quasi bidirectional port'.

**3.1.1 Port Structure**

**1] Port-0  Pin Structure:**

Port -0 has 8 pins (P0.0-P0.7).  
The structure of a Port-0 pin is shown in fig

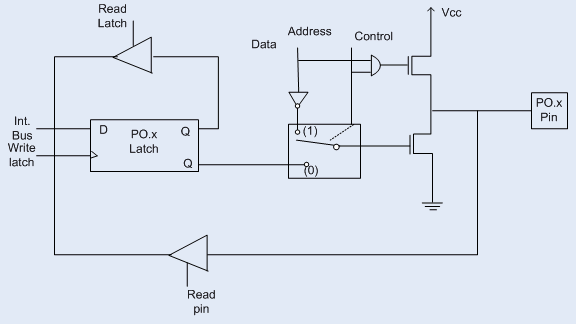


Fig 3.1: Port-0 Structure

1. Port-0 can be configured as a normal bidirectional I/O port or it can be used for address/data interfacing for accessing external memory. When control is '1', the port is used for address/data interfacing. When the control is '0', the port can be used as a normal bidirectional I/O port.
2. Let us assume that control is '0'. When the port is used as an input port, '1' is written to the latch. In this situation both the output MOSFETs are 'off'. Hence the output pin floats. This high impedance pin can be pulled up or low by an external source. When the port is used as an output port, a '1' written to the latch again turns 'off' both the output MOSFETs and causes the output pin to float. An external pull-up is required to output a '1'. But when '0' is written to the latch, the pin is pulled down by the lower MOSFET. Hence the output becomes zero.
3. When the control is '1', address/data bus controls the output driver MOSFETs. If the address/data bus (internal) is '0', the upper MOSFET is 'off' and the lower MOSFET is 'on'. The output becomes '0'.
4. If the address/data bus is '1', the upper transistor is 'on' and the lower transistor is 'off'. Hence the output is '1'. Hence for normal address/data interfacing (for external memory access) no pull-up resistors are required.
5. Port-0 latch is written to with 1's when used for external memory access.
6. When the first 0 is written to a port, it becomes an output
7. To reconfigure it as an input, a 1 must be sent to the port
8. To use any of these ports as an input port, it must be programmed
9. It can be used for input or output, each pin must be connected externally to a 10K ohm pull-up resistor
10. This is due to the fact that P0 is an open drain, unlike P1, P2, and P3
11. Open drain is a term used for MOS chips in the same way that open collector is used for TTL chips

Fig 3.2 Port 0 pins

**2] Port-1 Pin Structure:**

Port-1 has 8 pins (P1.1-P1.7) .The structure of a port-1 pin is shown in fig

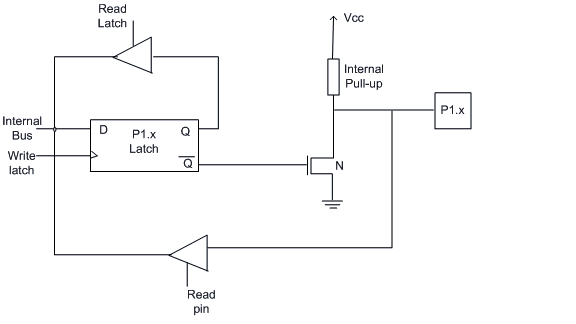
****

Fig3.3 Port 1 Structure

* **Port-1** does not have any alternate function i.e. it is dedicated solely for I/O interfacing.
* When used as output port, the pin is pulled up or down through internal pull-up.
* To use port-1 as input port, '1' has to be written to the latch.
* In this input mode when '1' is written to the pin by the external device then it readfine.
* But when '0' is written to the pin by the external device then the external source must sink current due to internal pull-up. If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading.
* In contrast to port 0, this port does not need any pull-up resistors since it already has pull-up resistors internally
* Upon reset, port 1 is configured as an input port

**3] Port-2 Pin Structure:**

Port-2 has 8-pins (P2.0-P2.7) . The structure of a port-2 pin is shown in fig

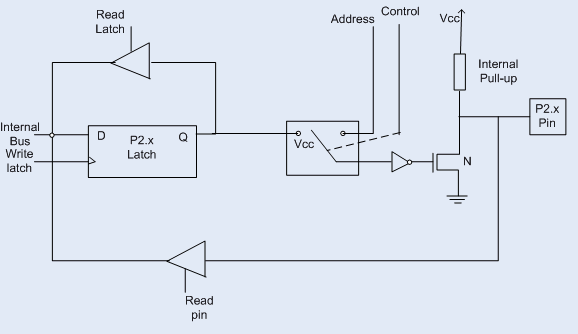
****

Fig3.4: Port 2 Structure

* Port-2 is used for higher external address byte or a normal input/output port. The I/O operation is similar to Port-1.
* Port-2 latch remains stable when Port-2 pin are used for external memory access. Here again due to internal pull-up there is limited current driving capability.
* Just like P1, port 2 does not need any pull up resistors since it already has pull-up

resistors internally

* Upon reset, port 2 is configured as an input port
* To make port 2 an input port, it must be programmed as such by writing 1 to all its bits
* Port 2 is also designated as A8 – A15, indicating its dual function
* Port 0 provides the lower 8 bits via A0 – A7

**4] PORT 3 Pin Structure**

Port-3 has 8 pin (P3.0-P3.7) . Port-3 pins have alternate functions. The structure of a port-3 pin is shown in fig

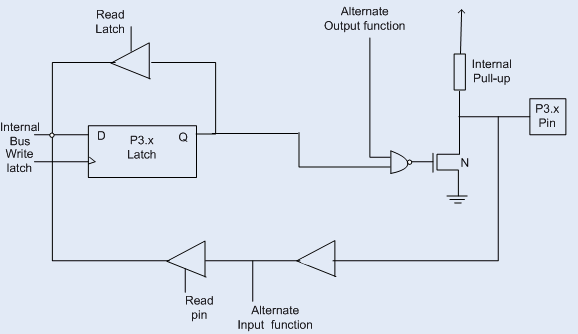


Fig 3.5: Port 3 Structure

* Each pin of Port-3 can be individually programmed for I/O operation or for alternate function.
* The alternate function can be activated only if the corresponding latch has been written to '1'.
* To use the port as input port, '1' should be written to the latch.
* This port also has internal pull-up and limited current driving capability.
* Alternate functions of Port-3 pins are –

|  |  |
| --- | --- |
| P3.0 | R x D |
| P3.1 | T x D |
| P3.2 | http://nptel.ac.in/courses/117104072/micro/lecture6/IMAGES/INTO.gif |
| P3.3 | http://nptel.ac.in/courses/117104072/micro/lecture6/IMAGES/INT1.gif |
| P3.4 | T0 |
| P3.5 | T1 |
| P3.6 | http://nptel.ac.in/courses/117104072/micro/lecture6/IMAGES/WR.gif |
| P3.7 | http://nptel.ac.in/courses/117104072/micro/lecture6/IMAGES/RD.gif |

Fig 3.6 Multifunction of port 3

**Note:**

1. Port 1, 2, 3 each can drive 4 LS TTL inputs.
2. Port-0 can drive 8 LS TTL inputs in address /data mode. For digital output port, it needs external pull-up resistors.
3. Ports-1,2and 3 pins can also be driven by open-collector or open-drain outputs.
4. Each Port 3 bit can be configured either as a normal I/O or as a special function bit.

* **Programs for using port**

1. **The following code will continuously send out to port 0 the alternating value 55H and AAH**

BACK: MOV A,#55H

MOV P0,A

ACALL DELAY

MOV A,#0AAH

MOV P0,A

ACALL DELAY

SJMP BACK

1. **Different ways of Accessing Entire 8 Bits**
   1. **The entire 8 bits of Port 1 are accessed**

BACK: MOV A,#55H

MOV P1,A

ACALL DELAY

MOV A,#0AAH

MOV P1,A

ACALL DELAY

SJMP BACK

* 1. **Rewrite the code in a more efficient manner by accessing the port directly without going through the accumulator**

BACK: MOV P1,#55H

ACALL DELAY

MOV P1,#0AAH

ACALL DELAY

SJMP BACK

* 1. **Another way of doing the same thing**

MOV A,#55H

BACK: MOV P1,A

ACALL DELAY

CPL A

SJMP BACK

* 1. **I/O Ports and Bit Addressability**

BACK: CPL P1.2 ;complement P1.2

ACALL DELAY

SJMP BACK

;another variation of the above program

AGAIN: SETB P1.2 ;set only P1.2

ACALL DELAY

CLR P1.2 ;clear only P1.2

ACALL DELAY

SJMP AGAIN

* **I/O Port programming in ‘C’**

1. **Aim:** Write an 8051 C program to send values 00 – FF to port P1.

**Solution:**

#include <reg51.h>

void main(void)

{

unsigned char z;

for (z=0;z<=255;z++)

P1=z;

}

1. **Aim:** Write an 8051 C program to send hex values for ASCII characters of 0, 1, 2, 3, 4, 5, A, B, C, and D to port P1.

**Solution:**

#include <reg51.h>

void main(void)

{

unsigned char mynum[]=“012345ABCD”;

unsigned char z;

for (z=0;z<=10;z++)

P1=mynum[z];

}

1. **Aim:** Write an 8051 C program to toggle all the bits of P1 continuously.

**Solution:**

//Toggle P1 forever

#include <reg51.h>

void main(void)

{

for (;;)

{

p1=0x55;

p1=0xAA;

}

}

1. **Aim:** Write an 8051 C program to send values of –4 to +4 to port P1.

**Solution:**

//Singed numbers

#include <reg51.h>

void main(void)

{

char mynum[]={+1,-1,+2,-2,+3,-3,+4,-4};

unsigned char z;

for (z=0;z<=8;z++)

P1=mynum[z];

}

1. **Aim:** Write an 8051 C program to toggle bit D0 of the port P1 (P1.0) 50,000 times.

**Solution:**

#include <reg51.h>

sbit MYBIT=P1^0;

void main(void)

{

unsigned int z;

for (z=0;z<=50000;z++)

{

MYBIT=0;

MYBIT=1;

}

}

1. **Aim:** Write an 8051 C program to toggle bits of P1 ports continuously with 250 ms.

**Solution:**

#include <reg51.h>

void MSDelay(unsigned int);

void main(void)

{

while (1) //repeat forever

{

p1=0x55;

MSDelay(250);

p1=0xAA;

MSDelay(250);

}

}

void MSDelay(unsigned int itime)

{

unsigned int i,j;

for (i=0;i<itime;i++)

for (j=0;j<1275;j++);

}

1. **Aim:** LEDs are connected to bits P1 and P2. Write an 8051 C program that shows the count from 00H to FFH (0000 0000 to 1111 1111 in binary) on the LEDs.

**Solution:**

#include <reg51.h>

#defind LED P2;

void main(void)

{

P1=00; //clear P1

LED=0; //clear P2

for (;;) //repeat forever

{

P1++; //increment P1

LED++; //increment P2

}

}

1. **Aim:** The data pins of an LCD are connected to P1. The information is latched into the LCD whenever its Enable pin goes from high to low. Write an 8051 C program to send “The Earth is but One Country” to this LCD.

**Solution:**

#include <reg51.h>

#define LCDData P1 //LCDData declaration

sbit En=P2^0; //the enable pin

void main(void)

{

unsigned char message[]

=“The Earth is but One Country”;

unsigned char z;

for (z=0;z<28;z++) //send 28 characters

{

LCDData=message[z];

En=1; //a high-

En=0; //-to-low pulse to latch data

}

}

**3.2 Timer/Counter Programming in assembly and C**

Timers can be used to generate a time delay or as event counters to count events happening outside the microcontroller. Both Timer 0 and Timer 1 are 16 bits wide and can be accessed as low byte and high byte. They can be accessed like any other register

MOV TL0,#4FH

MOV R5,TH0

Both timers 0 and 1 use the same register, called TMOD (timer mode), to set the various timer operation modes that is mode 0, mode1, mode 2, mode3.

**3.2.1 Clock for Timer**

As the every timer requires the clock pulse to count. The timer uses 1/12 of oscillator frequency as a source of clock for its operation.

**Example:** Find the timer’s clock frequency and its period for various 8051-based system, with the crystal frequency 11.0592 MHz when C/T bit of TMOD is 0.

**Solution:**

1/12 × 11.0529 MHz = 921.6 MHz;

T = 1/921.6 kHz = 1.085 us

1. If , it is used as a timer for time delay generation.
2. The clock source for the time delay is the crystal frequency of the 8051



* **Examples**

1. Indicate which mode and which timer are selected for each of the following.

(a) MOV TMOD, #01H (b) MOV TMOD, #20H (c) MOV TMOD, #12H

**Solution:**

We convert the value from hex to binary. From Figure 9-3 we have:

(a) TMOD = 00000001, mode 1 of timer 0 is selected.

(b) TMOD = 00100000, mode 2 of timer 1 is selected.

(c) TMOD = 00010010, mode 2 of timer 0, and mode 1 of timer 1 are selected.

1. Find the value for TMOD if we want to program timer 0 in mode 2, use 8051 XTAL or the clock source, and use instructions to start and stop the timer.

**Solution:**

TMOD = 0000 0010

Timer 0, mode 2

• C/T = 0 to use XTAL clock source

• gate = 0 to use internal (software) start and stop method.

**3.2.2 To generate a time delay using Timer mode 1**

1. Load the TMOD value register indicating which timer (timer 0 or timer 1) is to be

used and which timer mode (0 or 1) is selected

2. Load registers TL and TH with initial count value

3. Start the timer

4. Keep monitoring the timer flag (TF) with the JNB TFx, target instruction to see if it

is raised. Get out of the loop when TF becomes high

5. Stop the timer

6. Clear the TF flag for the next round

7. Go back to Step 2 to load TH and TL again

**Example:** In the following program, we create a square wave of 50% duty cycle (with

equal portions high and low) on the P1.5 bit. Timer 0 is used to generate the

time delay. **Analyze the program**

MOV TMOD,#01 ;Timer 0, mode 1(16-bit mode)

HERE: MOV TL0,#0F2H ;TL0=F2H, the low byte

MOV TH0,#0FFH ;TH0=FFH, the high byte

CPL P1.5 ;toggle P1.5

ACALL DELAY

SJMP HERE

DELAY:

SETB TR0 ;start the timer 0

AGAIN : JNB TF0,AGAIN ;monitor timer flag 0

;until it rolls over

CLR TR0 ;stop timer 0

CLR TF0 ;clear timer 0 flag

RET

* **In the above program notice the following step.**

1. TMOD is loaded.

2. FFF2H is loaded into TH0-TL0.

3. P1.5 is toggled for the high and low portions of the pulse.

4. The DELAY subroutine using the timer is called.

5. In the DELAY subroutine, timer 0 is started by the SETB TR0 instruction.

6. Timer 0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of FFF3, FFF4, FFF5, FFF6, FFF7, FFF8, FFF9, FFFA, FFFB, and so on until it reaches FFFFH. One more clock rolls it to 0, raising the timer flag (TF0=1). At that point, the JNB instruction falls through.

7. Timer 0 is stopped by the instruction CLR TR0. The DELAY subroutine ends, and the process is repeated.

**Notice that to repeat the process, we must reload the TL and TH registers, and start the process is repeated …FFF23 FFF4**

1. **Aim:** In above example , calculate the amount of time delay in the DELAY subroutine generated by the timer. Assume XTAL = 11.0592 MHz.

**Solution:**

* The timer works with a clock frequency of 1/12 of the XTAL frequency; therefore, we have 11.0592 MHz / 12 = 921.6 kHz as the timer frequency.
* As a result, each clock has a period of T =1/921.6kHz = 1.085us.

In other words, Timer 0 counts up each 1.085us

resulting in delay = number of counts × 1.085us.

* The number of counts for the roll over is FFFFH – FFF2H = 0DH (13 decimal). However, we add one to 13 because of the extra clock needed when it rolls over from FFFF to 0 and raise the TF flag.

This gives 14 × 1.085us = 15.19us for half the pulse.

For the entire period it is T = 2 × 15.19us = 30.38us as the time delay generated by the timer

* To calculate the values to be loaded into the TL and TH registers, look at the following **e**xampl**e:-**

1. Assume XTAL = 11.0592 MHz, we can use the following steps for finding the TH,

TL registers’ values

1. Divide the desired time delay by 1.085 us
2. Perform 65536 – n, where n is the decimal value we got in Step1
3. Convert the result of Step2 to hex, where
4. yyxx is the initial hex value to be loaded into the timer’s register

Set TL = xx and TH = yy

1. **Aim:** Assume that XTAL = 11.0592 MHz, write a program to generate a square wave of 2 kHz frequency on pin P1.5.

**Solution:**

we must toggle the bit togenerate the square wave. Look at the following steps.

(a) T = 1 / f = 1 / 2 kHz = 500 us the period of square wave.

(b) 1 / 2 of it for the high and low portion of the pulse is 250 us.

(c) 250 us / 1.085 us = 230 and 65536 – 230 = 65306 which in hex is FF1AH.

(d) TL = 1A and TH = FF, all in hex. The program is as follow.

MOV TMOD,#01 ;Timer 0, 16-bitmode

AGAIN: MOV TL1,#1AH ;TL1=1A, low byte of timer

MOV TH1,#0FFH ;TH1=FF, the high byte

SETB TR1 ;Start timer 1

BACK: JNB TF1,BACK ;until timer rolls over

CLR TR1 ;Stop the timer 1

CLR P1.5 ;Clear timer flag 1

CLR TF1 ;Clear timer 1 flag

SJMP AGAIN ;Reload timer

**3.2.3 To generate a time delay Mode 2 Programming**

1. Load the TMOD value register indicating which timer (timer 0 or timer 1) is to be

used, and the timer mode (mode 2) is selected

1. Load the TH registers with the initial count value
2. Start timer
3. Keep monitoring the timer flag (TF) with the JNB TF x, target instruction to see whether it is raised

Get out of the loop when TF goes high

5. Clear the TF flag

6. Go back to Step 4, since mode 2 is auto reload

**Example** Assume XTAL = 11.0592 MHz, find the frequency of the square wave generated on pin P1.0 in the following program

MOV TMOD,#20H ;T1/8-bit/auto reload

MOV TH1,#5 ;TH1 = 5

SETB TR1 ;start the timer 1

BACK: JNB TF1,BACK ;till timer rolls over

CPL P1.0 ;P1.0 to hi, lo

CLR TF1 ;clear Timer 1 flag

SJMP BACK ;mode 2 is auto-reload

**Solution:**

First notice the target address of SJMP. In mode 2 we do not need to reload TH since it is auto-reload.

Now (256 - 05) × 1.085 us =251 × 1.085 us = 272.33 us is the high portion of the pulse. Since it is a 50% duty cycle square wave, the period T is twice that; as a result T = 2 × 272.33 us = 544.67 us and the frequency = 1.83597 kHz

**3.2.4 Counter Programming**

- Timers can also be used as counters counting events happening outside the 8051

- The C/T bit in the TMOD registers decides the source of the clock for the timer

- When C/T = 1, the timer is used as a counter and gets its pulses from outside the 8051

- The counter counts up as pulses are fed from pins 14 and 15, these pins are called

T0(timer 0 input) and T1 (timer 1 input)

**Pin Port Pin Function Description**

15 P3.5 T1 Timer/counter 1 external input

14 P3.4 T0 Timer/counter 0 external input

**Examples:**

1. Assuming that clock pulses are fed into pin T1, write a program for counter 1 in mode 2 to count the pulses and display the state of the TL1 count on P2, which connects to 8 LEDs.

**Solution:**

MOV TM0D,#01100000B ;counter 1, mode 2,

;C/T=1 external pulses

MOV TH1,#0 ;clear TH1

SETB P3.5 ;make T1 input

AGAIN: SETB TR1 ;start the counter

BACK: MOV A,TL1 ;get copy of TL

MOV P2,A ;display it on port 2

JNB TF1,Back ;keep doing, if TF = 0

CLR TR1 ;stop the counter 1

CLR TF1 ;make TF=0

SJMP AGAIN ;keep doing it

Notice in the above program the role of the instruction SETB P3.5. Since ports are set up for output when the 8051 is powered up, we make P3.5 an input port by making it high. In other words, we must configure (set high) the T1 pin (pin P3.5) to allow pulses to be fed into it.

**3.2.5 C Programs for Timers**

1. **Aim:** Write an 8051 C program to toggle all the bits of port P1 continuously with some delay in between. Use Timer 0, 16-bit mode to generate the delay.

**Solution:**

#include <reg51.h>

void T0Delay(void);

void main(void){

while (1) {

P1=0x55;

T0Delay();

P1=0xAA;

T0Delay();

} }

void T0Delay(){

TMOD=0x01;

TL0=0x00;

TH0=0x35;

TR0=1;

while (TF0==0);

TR0=0;

TF0=0;

}

/\* FFFFH – 3500H = CAFFH = 51967 + 1 = 51968

51968 × 1.085 μs = 56.384 ms is the approximate delay \*/

1. **Aim:** A switch is connected to pin P1.2. Write an 8051 C program to monitor SW and create the following frequencies on pin P1.7: SW=0: 500Hz SW=1: 750Hz, use Timer 0, mode 1 for both of them.

**Solution:**

#include <reg51.h>

sbit mybit=P1^5;

sbit SW=P1^7;

void T0M1Delay(unsigned char);

void main(void){

SW=1;

while (1) {

mybit=~mybit;

if (SW==0)

T0M1Delay(0);

else

T0M1Delay(1);

}

}

void T0M1Delay(unsigned char c){

TMOD=0x01;

if (c==0) {

TL0=0x67;

TH0=0xFC;

}

else {

TL0=0x9A;

TH0=0xFD;

}

TR0=1;

while (TF0==0);

TR0=0;

TF0=0;

}

/\* FC67H = 64615

65536 – 64615 = 921

921 × 1.085 μs = 999.285 μs

1 / (999.285 μs × 2) = 500 Hz\*/

1. **Aim:** Write an 8051 C program to toggle only pin P1.5 continuously every250 ms. Use Timer 0, mode 2 (8-bit auto-reload) to create the delay.

**Solution:**

#include <reg51.h>

void T0M2Delay(void);

sbit mybit=P1^5;

void main(void){

unsigned char x,y;

while (1) {

mybit=~mybit;

for (x=0;x<250;x++)

for (y=0;y<36;y++) //we put 36, not 40

T0M2Delay();

} }

void T0M2Delay(void){

TMOD=0x02;

TH0=-23;

/\* 256 – 23 = 233 23 × 1.085 μs = 25 μs and

TR0=1; 25 μs × 250 × 40 = 250 ms\*/

while (TF0==0);

TR0=0;

TF0=0;

}

* + 1. **C Programming of Timers as Counters**

1. **Aim**: Assume that a 1-Hz external clock is being fed into pin T1 (P3.5). Write a C program for counter 1 in mode 2 (8-bit auto reload) to count up and display the state of the TL1 count on P1. Start the count at 0H.

**Solution:**

#include <reg51.h>

sbit T1=P3^5;

void main(void){

T1=1;

TMOD=0x60;

TH1=0;

while (1) {

do {

TR1=1;

P1=TL1;

}

while (TF1==0);

TR1=0;

TF1=0;

}

}

**3.3 Serial Port programming in assembly and C**

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

**Start and Stop Bits**

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
* Block-oriented data transfers use the synchronous method
* The start bit is always one bit, but the stop bit can be one or two bits.
* The start bit is always a 0 (low) and the stop bit(s) is 1 (high) 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
* To allow data transfer between the PC and an 8051 system without any error,

we must make sure that the baud rate of 8051 system matches the baud rate

of the PC’s COM port

|  |
| --- |
| PC Baud Rates19200  9600  4800  2400  1200  600  300  150  110 |

**Example:**

1. **Aim:** With XTAL = 11.0592 MHz, find the TH1 value needed to have the following baud rates.

(a) 9600 (b) 2400 (c) 1200

**Solution:**

The machine cycle frequency of 8051 = 11.0592 / 12 = 921.6 kHz,

and 921.6 kHz / 32 = 28,800 Hz is frequency by UART to timer 1 to

set baud rate.

(a) 28,800 / 3 = 9600 where -3 = FD (hex) is loaded into TH1

(b) 28,800 / 12 = 2400 where -12 = F4 (hex) is loaded into TH1

(c) 28,800 / 24 = 1200 where -24 = E8 (hex) is loaded into TH1

Notice that dividing 1/12 of the crystal frequency by 32 is the default value upon activation of the 8051 RESET pin.



SBUF is an 8-bit register used solely for serial communication. For a byte data to be transferred via the Txd line, it must be placed in the SBUF Register. SBUF holds the byte of data when it is received by 8051 Rxd line

Example: MOV SBUF,#’D’ ;load SBUF=44h, ASCII for ‘D’

MOV SBUF,A ;copy accumulator into SBUF

MOV A,SBUF ;copy SBUF into accumulator

1. **Aim:** Write a program for the 8051 to transfer letter “A” serially at 4800 baud,continuously.

**Solution:**

MOV TMOD,#20H ;timer 1,mode 2(auto reload)

MOV TH1,#-6 ;4800 baud rate

MOV SCON,#50H ;8-bit, 1 stop, REN enabled

SETB TR1 ;start timer 1

AGAIN: MOV SBUF,#”A” ;letter “A” to transfer

HERE: JNB TI,HERE ;wait for the last bit

CLR TI ;clear TI for next char

SJMP AGAIN ;keep sending A

1. **Aim:** Write a program for the 8051 to transfer “YES” serially at 9600 baud, 8-bit data, 1 stop bit, do this continuously

**Solution:**

MOV TMOD,#20H ;timer 1,mode 2(auto reload)

MOV TH1,#-3 ;9600 baud rate

MOV SCON,#50H ;8-bit, 1 stop, REN enabled

SETB TR1 ;start timer 1

AGAIN: MOV A,#”Y” ;transfer “Y”

ACALL TRANS

MOV A,#”E” ;transfer “E”

ACALL TRANS

MOV A,#”S” ;transfer “S”

ACALL TRANS

SJMP AGAIN ;keep doing it

;serial data transfer subroutine

TRANS: MOV SBUF,A ;load SBUF

HERE: JNB TI,HERE ;wait for the last bit

CLR TI ;get ready for next byte

RET

1. **Aim:** Write a program for the 8051 to receive bytes of data serially, and put them in P1, set the baud rate at 4800, 8-bit data, and 1 stop bit.

**Solution:**

MOV TMOD,#20H ;timer 1,mode 2(auto reload)

MOV TH1,#-6 ;4800 baud rate

MOV SCON,#50H ;8-bit, 1 stop, REN enabled

SETB TR1 ;start timer 1

HERE: JNB RI,HERE ;wait for char to come in

MOV A,SBUF ;saving incoming byte in A

MOV P1,A ;send to port 1

CLR RI ;get ready to receive next ;byte

SJMP HERE ;keep getting data

* + 1. **Doubling Baud Rate**

MOV A,PCON ;place a copy of PCON in ACC

SETB ACC.7 ;make D7=1

MOV PCON,A ;changing any other bits



Baud Rate comparison for SMOD=0 and SMOD=1



**Example:** Find the baud rate if TH1 = -2, SMOD = 1, and XTAL = 11.0592MHz. Is this baud rate supported by IBM compatible PCs?

**Solution:**

With XTAL = 11.0592 and SMOD = 1, we have timer frequency = 57,600 Hz.

The baud rate is 57,600/2 = 28,800. This baud rate is not supported by the BIOS of the PCs; however, the PC can be programmed to do data transfer at such a speed. Also, HyperTerminal in Windows supports this and other baud rates

**3.3.2 Serial Port Programming in ‘C’**

**1. Aim:** Write a C program for 8051 to transfer the letter “A” serially at 4800 baud continuously. Use 8-bit data and 1 stop bit.

**Solution:**

#include <reg51.h>

void main(void){

TMOD=0x20; //use Timer 1, mode 2

TH1=0xFA; //4800 baud rate

SCON=0x50;

TR1=1;

while (1) {

SBUF=‘A’; //place value in buffer

while (TI==0);

TI=0;

} }

1. **Aim:** Write an 8051 C program to transfer the message “YES” serially at 9600 baud, 8-bit data, 1 stop bit. Do this continuously.

**Solution:**

#include <reg51.h>

void Ser T x (unsigned char);

void main(void){

TMOD=0x20; //use Timer 1, mode 2

TH1=0xFD; //9600 baud rate

SCON=0x50;

TR1=1; //start timer

while (1) {

Ser T x (‘Y’);

Ser T x (‘E’);

Ser T x (‘S’);

}

}

void Ser T x (unsigned char x){

SBUF=x; //place value in buffer

while (TI==0); //wait until transmitted

TI=0;

}

1. **Aim:** Program the 8051 in C to receive bytes of data serially and put them in P1. Set the baud rate at 4800, 8-bit data, and 1 stop bit.

**Solution:**

#include <reg51.h>

void main(void){

unsigned char my byte;

TMOD=0x20; //use Timer 1, mode 2

TH1=0xFA; //4800 baud rate

SCON=0x50;

TR1=1; //start timer

while (1) { //repeat forever

while (RI==0); //wait to receive

my byte=SBUF; //save value

P1=my byte; //write value to port

RI=0;

}

}

1. **Aim:** Write a C program to send out the value 44H serially one bit at a time via P1.0. The LSB should go out first.

**Solution:**

#include <reg51.h>

sbit P1b0=P1^0;

sbit regALSB=ACC^0;

void main(void)

{

unsigned char conbyte=0x44;

unsigned char x;

ACC=conbyte;

for (x=0;x<8;x++)

{

P1b0=regALSB;

ACC=ACC>>1;

}

}

1. **Aim:** Write a C program to send out the value 44H serially one bit at a time via P1.0. The MSB should go out first.

**Solution:**

#include <reg51.h>

sbit P1b0=P1^0;

sbit regAMSB=ACC^7;

void main(void)

{

unsigned char conbyte=0x44;

unsigned char x;

ACC=conbyte;

for (x=0;x<8;x++)

{

P1b0=regAMSB;

ACC=ACC<<1;

}

}

1. **Aim:** Write a C program to bring in a byte of data serially one bit at a time via P1.0. The LSB should come in first.

**Solution:**

#include <reg51.h>

sbit P1b0=P1^0;

sbit ACCMSB=ACC^7;

bit membit;

void main(void)

{

unsigned char x;

for (x=0;x<8;x++)

{

membit=P1b0;

ACC=ACC>>1;

ACCMSB=membit;

}

P2=ACC;

}

1. **Aim:** Write a C program to bring in a byte of data serially one bit at a time via P1.0. The MSB should come in first.

**Solution:**

#include <reg51.h>

sbit P1b0 = P1^0;

sbit regALSB = ACC^0;

bit membit;

void main(void)

{

unsigned char x;

for (x=0; x <8; x ++)

{

membit=P1b0;

ACC=ACC<<1;

regALSB=membit;

}

P2=ACC;

}

**3.4 Interrupt programming in assembly and C**

An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service

Upon activation of an interrupt, the microcontroller goes through the following steps

1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack

2. It also saves the current status of all the interrupts internally (i. e: not on the stack)

3. It jumps to a fixed location in memory, called the interrupt vector table, that holds the address of the ISR the microcontroller gets the address of the ISR from the interrupt vector tableand jumps to it

4.It starts to execute the interrupt service subroutine until it reaches the last instruction

of the subroutine which is RETI (return from interrupt)

5. Upon executing the RETI instruction, the microcontroller returns to the place

where it was interrupted

6. First, it gets the program counter (PC) address from the stack by popping the top

two bytes of the stack into the PC

7. Then it starts to execute from that address

**3.4.1 Enabling and Disabling an Interrupt**

Show the instructions to

(a) enable the serial interrupt, timer 0 interrupt, and external hardware interrupt 1 (EX1),and

(b) disable (mask) the timer 0 interrupt, then

(c) show how to disable all the interrupts with a single instruction.

**Solution:**

(a) MOV IE,#10010110B ;enable serial, ;timer 0, EX1

- Another way to perform the same manipulation is

SETB IE.7 ;EA=1, global enable

SETB IE.4 ;enable serial interrupt

SETB IE.1 ;enable Timer 0 interrupt

SETB IE.2 ;enable EX1

(b) CLR IE.1 ;mask (disable) timer 0

;interrupt only

(c) CLR IE.7 ;disable all interrupts

**3.4.2 TIMER INTERRUPTS**

The timer flag (TF) is raised when the timer rolls over

****

If the timer interrupt in the IE register is enabled, whenever the timer rolls over, TF is raised, and the microcontroller is interrupted in whatever it is doing, and jumps to the interrupt vector table to service the ISR

In this way, the microcontroller can do other until it is notified that the timer has rolled over

**Example:** Write a program that continuously get 8-bit data from P0 and sends it to P1 while simultaneously creating a square wave of 200 μs period on pin P2.1. Use timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz.

**Solution:**

We will use timer 0 in mode 2 (auto reload). TH0 = 100/1.085 us = 92

;--upon wake-up go to main, avoid using

;memory allocated to Interrupt Vector Table

ORG 0000H

LJMP MAIN ;by-pass interrupt vector table

; ;--ISR for timer 0 to generate square wave

ORG 000BH ;Timer 0 interrupt vector table

CPL P2.1 ;toggle P2.1 pin

RETI ;return from ISR

;--The main program for initialization

ORG 0030H ;after vector table space

MAIN: MOV TMOD,#02H ;Timer 0, mode 2

MOV P0,#0FFH ;make P0 an input port

MOV TH0,#-92 ;TH0=A4H for -92

MOV IE,#82H ;IE=10000010 (bin) enable ;Timer 0

SETB TR0 ;Start Timer 0

BACK: MOV A,P0 ;get data from P0

MOV P1,A ;issue it to P1

SJMP BACK ;keep doing it loop ;unless interrupted by TF0

END

**3.4.2 External Hardware Interrupts**

The 8051 has two external hardware interrupts that is on Pin 12 (P3.2) and pin 13 (P3.3) of the 8051, designated as INT0 and INT1 There are two activation levels for the external hardware interrupts

* Level trigged
* Edge trigged

****

****

**Level-Triggered Interrupt**

In the level-triggered mode, INT0 and INT1 pins are normally high

* If a low-level signal is applied to them, it triggers the interrupt Then the microcontroller stops whatever it is doing and jumps to the interrupt vector table to service that interrupt
* The low-level signal at the INT pin must be removed before the execution of the last instruction of the ISR, RETI; otherwise, another interrupt will be generated
* This is called a level-triggered or level activated interrupt and is the default mode upon reset of the 8051

**Example:**

Assume that the INT1 pin is connected to a switch that is normally high. Whenever it goes low, it should turn on an LED. The LED is connected to P1.3 and is normally off. When it is turned on it should stay on for a fraction of a second. As long as the switch is pressed low,

the LED should stay on.

**Solution:**

ORG 0000H

LJMP MAIN ;by-pass interrupt ;vector table

;--ISR for INT1 to turn on LED

ORG 0013H ;INT1 ISR

SETB P1.3 ;turn on LED

MOV R3,#255

BACK : DJNZ R3,BACK ;keep LED on for a while

CLR P1. ;turn off the LED

RETI ;r eturn from ISR

;--MAIN program for initialization

ORG 30H

MAIN: MOV IE,#10000100B ;enable external INT 1

HERE: SJMP HERE ;stay here until get interrupted

END

**Edge-Triggered Interrupt**

To make INT0 and INT1 edge triggered interrupts, we must programthe bits of the TCON register

* The TCON register holds, among other bits, the IT0 and IT1 flag bits that

determine level- or edge-triggered mode of the hardware interrupt

* IT0 and IT1 are bits D0 and D2 of the TCON register
* They are also referred to as TCON.0 and TCON.2 since the TCON register is
* Bit addressable

**Example:**

1. Assume that pin 3.3 (INT1) is connected to a pulse generator, write a program in which the falling edge of the pulse will send a high to P1.3, which is connected to an LED (or buzzer). In other words, the LED is turned on and off at the same rate as the pulses are applied to the INT1 pin.

**Solution:**

ORG 0000H

LJMP MAIN

;--ISR for hardware interrupt INT1 to turn on LED

ORG 0013H ;INT1 ISR

SETB P1.3 ;turn on LED

MOV R3,#255

BACK: DJNZ R3,BACK ;keep the buzzer on for a while

CLR P1.3 ;turn off the buzzer

RETI ;return from ISR

;------MAIN program for initialization

ORG 30H

MAIN: SETB TCON.2 ;make INT1 edge-triggered int.

MOV IE,#10000100B ;enable External INT 1

HERE: SJMP HERE ;stay here until get interrupted

END

2. Write a C program that continuously gets a single bit of data from P1.7 and sends it to P1.0, while simultaneously creating a square wave of 200 μs period on pin P2.5. Use Timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz**.**

**Solution:**

We will use timer 0 mode 2 (auto-reload). One half of the period is

100 μs. 100/1.085 μs = 92, and TH0 = 256 - 92 = 164 or A4H

#include <reg51.h>

sbit SW =P1^7;

sbit IND =P1^0;

sbit WAVE =P2^5;

void timer0(void) interrupt 1 {

WAVE=~WAVE; //toggle pin

}

void main() {

SW=1; //make switch input

TMOD=0x02;

TH0=0xA4; //TH0=-92

IE=0x82 ; //enable interrupt for timer 0

while (1) {

IND=SW; //send switch to LED

}

}

**3.4.3 Write a C program using interrupts to do the following:**

(a) Receive data serially and send it to P0

(b) Read port P1, transmit data serially, and give a copy to P2

(c) Make timer 0 generate a square wave of 5 kHz frequency on P0.1

Assume that XTAL = 11.0592 MHz. Set the baud rate at 4800.

**Solution:**

#include <reg51.h>

sbit WAVE =P0^1;

void timer0() interrupt 1 {

WAVE=~WAVE ; //toggle pin

}

void serial0() interrupt 4 {

if (TI==1) {

TI=0 ; //clear interrupt

}

else {

P0=SBUF; //put value on pins

RI=0; //clear interrupt

}

}

**QUESTION BANK**

**2 M Questions**

1. State the condition of RS, RW & E pins while sending data on LCD.

**4 Marks question**

1. What do you mean by open drain or open collector output? What is the advantage of it? Why we require external pull up registers for port 1 of 8051 microcontroller? [SUMMER 12]
2. Write an ALP to generate two square waves on P1.3 & P2.3 of 50 kHz & 25 kHz. Assume X’tal =12 MHz. [SUMMER 13]
3. Write an ALP to receive bytes of data serially & put them in P2.1. Set the baud rate at 9600, 8-bit data & 1-stop bit. [ Winter 12]
4. Observe the following program & find

1. Frequency of square wave generated on P1.0

2. Smallest frequency achievable & TH value for the same.

MOV TMOD, #20H

MOV TH1, #05

SETB TR1

BACK: JNB TF1, BACK

CPL P1.0

CLR TF1

SJMP BACK

1. Write a program that continuously gets 8 bit data from port 0 & sends it to port 1, while simultaneously creating square wave of 200 μsec on pin P2.1. Use timer 0 to create square wave. Assume X’tal=11.0592 MHz.
2. Draw the port ‘3’ structure of 8051 with neat label. Also list any two alternate functions of port 3.
3. Write ‘C’ or assembly language program for 8051 to transfer

letter “M” serially at 4800 baud rate. Summer 12

1. Write assembly on ‘C’ language program to generate a square wave of 50Hz frequency on Pin Pl-2. Use interrupts for timer. Assume crystal frequency 11.0592 MHz.[winter 12]

**6 Marks Questions**

1. Generate the following waveform on P1.2. Assume X’tal=22MHz, Timer 1 in mode1.[ winter 12]



1. . Take data in through ports 0, 1 & 2, one after the other & transfer this data

serially, continuously at baud rate 4800, timer 1 in mode 2, 8 bit data & 1 stop bit.

1. Two switches are connected to pins P3.2 & P3.3. When a switch is pressed, the

corresponding line goes low. Write a program to –

i. Send message “YES” serially at baud rate of 9600.

ii. Send message “NO” serially at the baud rate of 9600.

1. Draw “DB-9” RS232 connector and describe any four signal.

**8 M Questions**

1. State the interrupts in 8051 microcontroller. At what part pins are external interrupts located? State the role of the two bits of TCON.0 and TCON.2 play in execution of the external interrupts. [ summer 12]