

Experiment 1

Construction of a In System Programmer (ISP) circuit for AT89S52 microcontroller and writing simple codes.

Objective

The objective of this experiment is to connect the ISP programmer circuit for AT89S52 microcontroller and testing the correctness by writing and downloading simple codes.

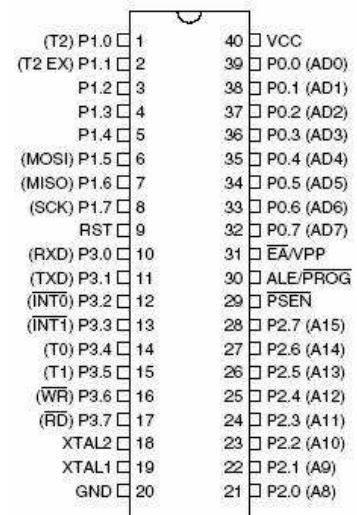
AT89S52 Pin Description

VCC Supply voltage.

GND Ground.

Port 0 Port 0 is an 8-bit open drain bidirectional I/O port. As an output port, each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs. Port 0 can also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode, P0 has internal pull-ups. Port 0 also receives the code bytes during Flash programming and outputs the code bytes during program verification. **External pull-ups are required during program verification.**

Port 1 Port 1 is an 8-bit bidirectional I/O port with internal pull-ups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins, they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL) because of the internal pull-ups. In addition, P1.0 and P1.1 can be configured to be the timer/counter 2 external count input (P1.0/T2) and the timer/counter 2 trigger input (P1.1/T2EX), respectively, as shown in the following table. Port 1 also receives the low-order address bytes during Flash programming and verification.



Port Pin	Alternate Functions
P1.0	T2 (external count input to Timer/Counter 2), clock-out
P1.1	T2EX (Timer/Counter 2 capture/reload trigger and direction control)
P1.5	MOSI (used for In-System Programming)
P1.6	MISO (used for In-System Programming)
P1.7	SCK (used for In-System Programming)

Port 2 Port 2 is an 8-bit bidirectional I/O port with internal pull-ups. The Port 2 output buffers can sink/source four TTL inputs. When 1s are written to Port 2 pins, they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL) because of the internal pull-ups. Port 2 emits the high-order address byte during fetches from external program memory and

during accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application, Port 2 uses strong internal pull-ups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @ RI), Port 2 emits the contents of the P2 Special Function Register. Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

Port 3 Port 3 is an 8-bit bidirectional I/O port with internal pull-ups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to Port 3 pins, they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL) because of the pull-ups. Port 3 receives some control signals for Flash programming and verification. Port 3 also serves the functions of various special features of the AT89S52, as shown in the following table.

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt 0)
P3.3	INT1 (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	WR (external data memory write strobe)
P3.7	RD (external data memory read strobe)

RST Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device. This pin drives high for 98 oscillator periods after the Watchdog times out. The DISRTO bit in SFR AUXR (address 8EH) can be used to disable this feature. In the default state of bit DISRTO, the RESET HIGH out feature is enabled.

ALE/PROG Address Latch Enable (ALE) is an output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input (PROG) during Flash programming. In normal operation, ALE is emitted at a constant rate of 1/6 the oscillator frequency and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external data memory. If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the ALE-disable bit has no effect if the microcontroller is in external execution mode.

PSEN Program Store Enable (PSEN) is the read strobe to external program memory. When the AT89S52 is executing code from external program memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to external data memory.

EA/VPP External Access Enable. EA must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, EA will be internally latched on reset. EA should be strapped to VCC for internal program executions. This pin also receives the 12-volt programming enable voltage (VPP) during Flash programming.

XTAL1 Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2 Output from the inverting oscillator amplifier.

Circuit Diagrams

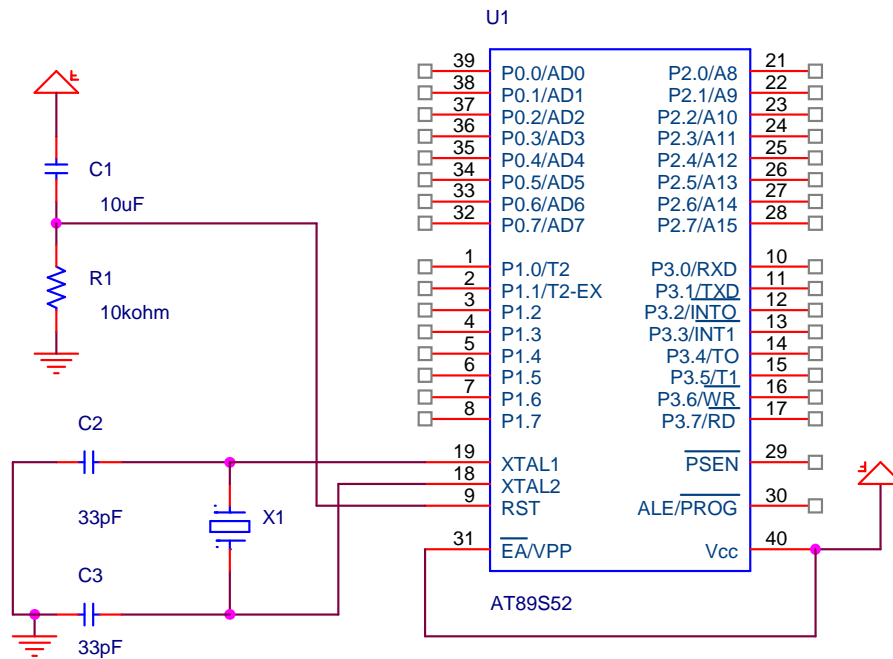


Fig. 1.1 : Basic Connection Diagram for an AT89S52 Microcontroller

Procedure

1. Connect the circuits as shown in figure 1.2 in the bread board. Use 20MHz crystal.
2. Write the following code:

```
#include<reg52x2.h>

void delay(unsigned int i){
    int j;
    for (j=0;j<100;j++){
        while (i--);
    }
}

void main(){
P2_0 = 0;
P2_1 = 0;
while (1){
P2_0 = ~P2_0;
P2_1 = ~P2_0;
delay(1000);
}
```

Now convert it into .hex file and download the code to the microcontroller using ISPv3.

3. Write a simple code in the computer (in Keil or Ride) for blinking two LED's in the sequence: 00, 11, 01, 10. Connect the LED's in P2_0 and P2_1 pins.

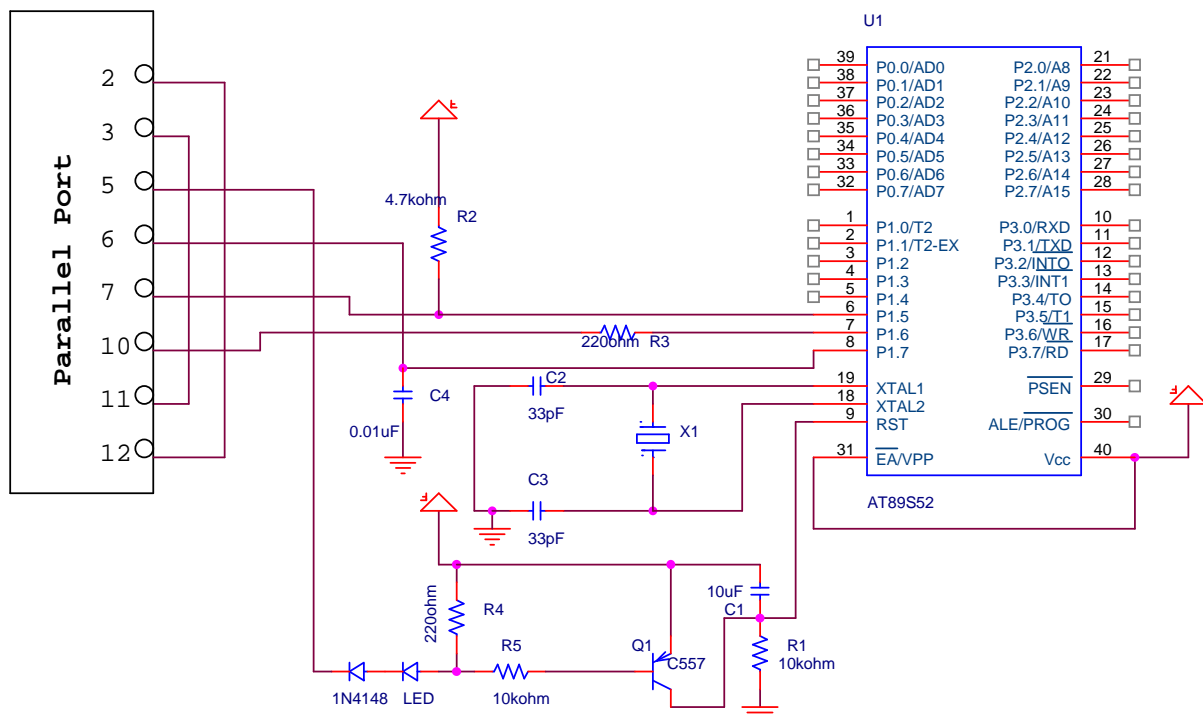


Fig. 1.2: Programmer Circuit for an AT89S52 MicroController

Report

1. Consider two push-to-on switches to pin P1_0 and P1_1. Also consider an LED to pin P2_0. Now write a code to control the LED by the two switches completely independently (Consider that the two switches can not be pressed at a time).
2. Design a system and write a code to drive a seven segment display by Port 1. Add a toggle switch to P2_0 such that the value of the Seven Segment Display increases with every press in the button.

Prepared by: Upal Mahbub

Experiment 3

PWM pattern generation using Timer 0 and Timer 1 of AT89S52.

Objective

The objective of this experiment is to generate PWM pattern with the help of timers available in MCS 51 microcontroller. PWM with variable duty cycle will be generated by Timer 0 and Timer 1 in 8-bit auto reload mode.

Microcontroller Resources to be used

The following resources would be needed to generate PWM waveforms using AT89S52.

1. Timer 0 and Timer 1
2. TCON and TMOD registers
3. Interrupts (related to timer 0 and timer 1)
4. Ports
5. On chip clock generator

Timer/Counter Operations

Timer 0

Timer 0 functions as either a timer or event counter in four modes of operation. Timer 0 is controlled by the four lower bits of the TMOD register and bits 0, 1, 4 and 5 of the TCON register. TMOD register selects the method of timer gating (GATE0), timer or counter operation (T/C0#) and mode of operation (M10 and M00). The TCON register provides timer 0 control functions: overflow flag (TF0), run control bit (TR0), interrupt flag (IE0) and interrupt type control bit (IT0).

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

Timer 0 overflow (count rolls over from all 1s to all 0s) sets the TF0 flag, generating an interrupt request. It is important to stop timer/counter before changing mode.

Mode 0 (13-bit Timer)

Mode 0 configures timer 0 as a 13-bit timer which is set up as an 8-bit timer (TH0 register) with a modulo 32 prescaler implemented with the lower five bits of the TL0 register. The upper three bits of TL0 register are indeterminate and should be ignored. Prescaler overflow increments the TH0 register.

As the count rolls over from all 1's to all 0's, it sets the timer interrupt flag TF0. The counted input is enabled to the Timer when TR0 = 1 and either GATE = 0 or INT0 = 1. (Setting GATE = 1 allows the Timer to be controlled by external input INT0, to facilitate pulse width measurements). TR0 is a control bit in the Special Function register TCON. GATE is in TMOD. The 13-bit register consists of all 8 bits of TH0 and the lower 5 bits of TL0. The upper 3 bits of TL0 are indeterminate and should be ignored. Setting the run flag (TR0) does not clear the registers.

Mode 0 operation is the same for Timer 0 as for Timer 1. Substitute TR0, TF0 and INT0 for the corresponding Timer 1 signals. There are two different GATE bits, one for Timer 1 (TMOD.7) and one for Timer 0 (TMOD.3).

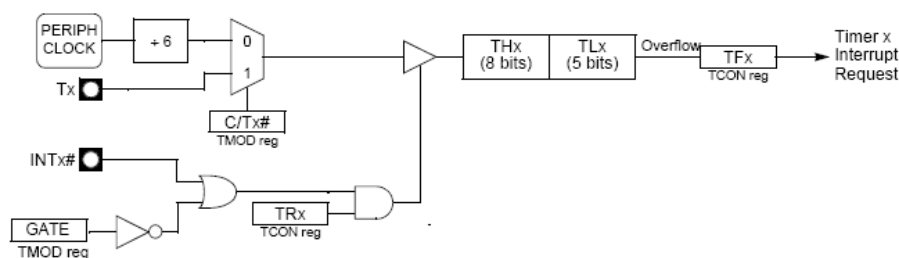


Figure: Timer/Counter x (x = 0 or 1) in Mode 0.

Mode 1 (16-bit Timer)

Mode 1 is the same as Mode 0, except that the Timer register is being run with all 16 bits. Mode 1 configures timer 0 as a 16-bit timer with the TH0 and TL0 registers connected in cascade. The selected input increments the TL0 register.

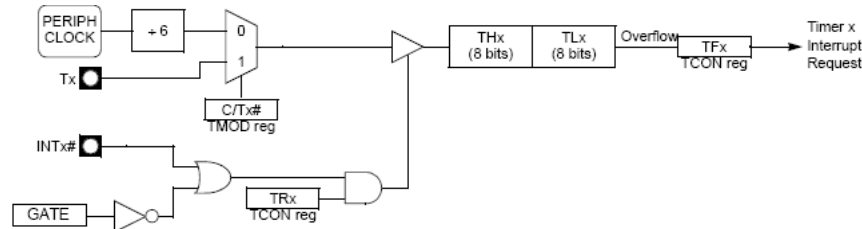


Figure: Timer/Counter x (x = 0 or 1) in Mode 1.

Mode 2 (8-bit Timer with Auto-Reload)

Mode 2 configures timer 0 as an 8-bit timer (TL0 register) that automatically reloads from the TH0 register. TL0 overflow sets TF0 flag in the TCON register and reloads TL0 with the contents of TH0, which is preset by software. When the interrupt request is serviced, hardware clears TF0. The reload leaves TH0 unchanged. The next reload value may be changed at any time by writing it to the TH0 register. Mode 2 operation is the same for Timer/Counter 1.

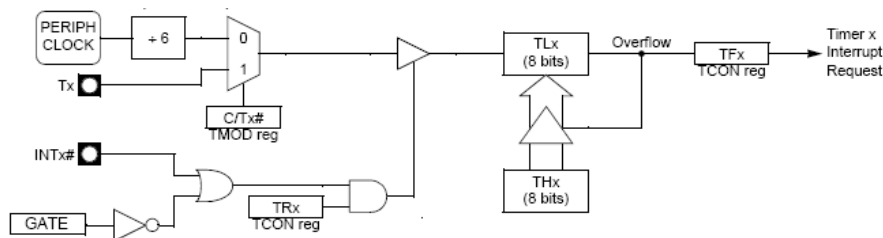


Figure: Timer/Counter x (x = 0 or 1) in Mode 2.

Mode 3 (Two 8-bit Timers)

Mode 3 configures timer 0 so that registers TL0 and TH0 operate as separate 8-bit timers. This mode is provided for applications requiring an additional 8-bit timer or counter. TL0 uses the timer 0 control bits C/T0# and GATE0 in the TMOD register, and TR0 and TF0 in the TCON register in the normal manner. TH0 is locked into a timer function (counting $F_{PER}/6$) and takes over use of the timer 1 interrupt (TF1) and run control (TR1) bits. Thus, operation of timer 1 is restricted when timer 0 is in mode 3.

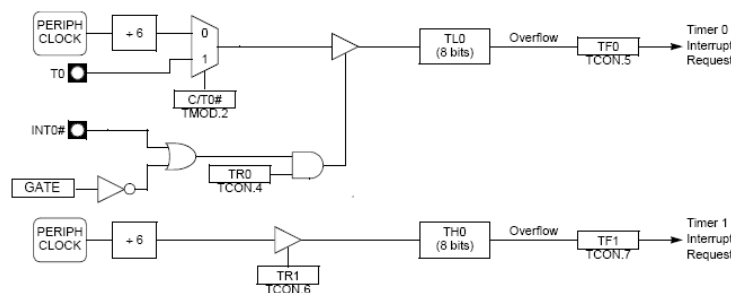


Figure: Timer/Counter 0 in Mode 3: Two 8-bit Counters.

Timer 1

Timer 1 is identical to timer 0, except for mode 3, which is a hold-count mode. The following comments help to understand the differences: • Timer 1 functions as either a timer or event counter in three modes of operation. Timer 1's mode 3 is a hold-count mode.

- Timer 1 is controlled by the four high-order bits of the TMOD register and bits 2, 3, 6 and 7 of the TCON register. The TMOD register selects the method of timer gating (GATE1), timer or counter operation (C/T1#) and mode of operation (M11 and M01). The TCON register provides timer 1 control functions: overflow flag (TF1), run control bit (TR1), interrupt flag (IE1) and interrupt type control bit (IT1).
- Timer 1 can serve as the baud rate generator for the serial port. Mode 2 is best suited for this purpose.
- For normal timer operation (GATE1 = 0), setting TR1 allows TL1 to be incremented by the selected input. Setting GATE1 and TR1 allows external pin INT1# to control timer operation.
- Timer 1 overflow (count rolls over from all 1s to all 0s) sets the TF1 flag generating an interrupt request.
- When timer 0 is in mode 3, it uses timer 1's overflow flag (TF1) and run control bit (TR1). For this situation, use timer 1 only for applications that do not require an interrupt (such as a baud rate generator for the serial port) and switch timer 1 in and out of mode 3 to turn it off and on.
- It is important to stop timer/counter before changing modes.

Mode 0 (13-bit Timer)

Mode 0 configures Timer 1 as a 13-bit timer, which is set up as an 8-bit timer (TH1 register) with a modulo-32 prescaler implemented with the lower 5 bits of the TL1 register (Figure 5-5). The upper 3 bits of the TL1 register are ignored. Prescaler overflow increments the TH1 register.

Mode 1 (16-bit Timer)

Mode 1 configures Timer 1 as a 16-bit timer with the TH1 and TL1 registers connected in cascade. The selected input increments the TL1 register.

Mode 2 (8-bit Timer with Auto Reload)

Mode 2 configures Timer 1 as an 8-bit timer (TL1 register) with automatic reload from the TH1 register on overflow. TL1 overflow sets the TF1 flag in the TCON register and reloads TL1 with the contents of TH1, which is preset by software. The reload leaves TH1 unchanged.

Mode 3 (Halt)

Placing Timer 1 in mode 3 causes it to halt and hold its count. This can be used to halt Timer 1 when TR1 run control bit is not available i.e., when Timer 0 is in mode 3. When Timer 0 is in Mode 3, Timer 1 can be turned on and off by switching it out of and into its own Mode 3, or can still be used by the serial port as a baud rate generator, or in fact, in any application not requiring an interrupt.

Timer Registers**TCON Register - Timer/Counter Control Register**

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Bit Number	Bit Mnemonic	Description
7	TF1	Timer 1 Overflow Flag Cleared by hardware when processor vectors to interrupt routine. Set by hardware on timer/counter overflow, when the timer 1 register overflows.
6	TR1	Timer 1 Run Control Bit Clear to turn off timer/counter 1. Set to turn on timer/counter 1.

EEE - 494**Microcontroller and Embedded System Lab Manual**

Microprocessor and Interfacing Lab

Department of Electrical and Electronic Engineering, BUET

5	TF0	Timer 0 Overflow Flag Cleared by hardware when processor vectors to interrupt routine. Set by hardware on timer/counter overflow, when the timer 0 register overflows.
4	TR0	Timer 0 Run Control Bit Clear to turn off timer/counter 0. Set to turn on timer/counter 0.
3	IE1	Interrupt 1 Edge Flag Cleared by hardware when interrupt is processed if edge-triggered (see IT1). Set by hardware when external interrupt is detected on INT1# pin.
2	IT1	Interrupt 1 Type Control Bit Clear to select low level active (level triggered) for external interrupt 1 (INT1#). Set to select falling edge active (edge triggered) for external interrupt 1.
1	IE0	Interrupt 0 Edge Flag Cleared by hardware when interrupt is processed if edge-triggered (see IT0). Set by hardware when external interrupt is detected on INT0# pin.
0	IT0	Interrupt 0 Type Control Bit Clear to select low level active (level triggered) for external interrupt 0 (INT0#). Set to select falling edge active (edge triggered) for external interrupt 0.
Reset Value = 0000 0000b		

TMOD Register - Timer/Counter 0 and 1 Modes

7	6	5	4	3	2	1	0
GATE1	C/T1#	M11	M01	GATE0	C/T0#	M10	M00

Bit Number	Bit Mnemonic	Description		
7	GATE1	Timer 1 Gating Control Bit Clear to enable timer 1 whenever the TR1 bit is set. Set to enable timer 1 only while the INT1# pin is high and TR1 bit is set.		
6	C/T1#	Timer 1 Counter/Timer Select Bit Clear for timer operation: timer 1 counts the divided-down system clock. Set for Counter operation: timer 1 counts negative transitions on external pin T1.		
5	M11	Timer 1 Mode Select Bits		
4	M01	M11	M01	Operating mode
		0	0	Mode 0: 8-bit timer/counter (TH1) with 5-bit pre-scaler (TL1).
		0	1	Mode 1: 16-bit timer/counter.
		1	0	Mode 2: 8-bit auto-reload timer/counter (TL1). Reloaded from TH1 at overflow.
		1	1	Mode 3: timer 1 halted. Retains count.
3	GATE0	Timer 0 Gating Control Bit Clear to enable timer 0 whenever the TR0 bit is set. Set to enable timer/counter 0 only while the INT0# pin is high and the TR0 bit is set.		
2	C/T0#	Timer 0 Counter/Timer Select Bit Clear for timer operation: timer 0 counts the divided-down system clock. Set for counter operation: timer 0 counts negative transitions on external pin T0.		
1	M10	Timer 0 Mode Select Bit		
0	M00	M10	M00	Operating Mode

EEE - 494

Microcontroller and Embedded System Lab Manual

Microprocessor and Interfacing Lab

Department of Electrical and Electronic Engineering, BUET

		0	0	Mode 0: 8-bit timer/counter (TH0) with 5-bit pre-scaler (TL0).
		0	1	Mode 1: 16-bit timer/counter.
		1	0	Mode 2: 8-bit auto-reload timer/counter (TL0). Reloaded from TH0 at overflow.
		1	1	Mode 3: TL0 is an 8-bit timer/counter.
		TH0 is an 8-bit timer using timer 1's TR0 and TF0 bits.		

C Program

```
#include <reg52x2.h>
#include <stdio.h>
#include <math.h>

#define reload_value 0x36 /* reload value example */

/**
 * FUNCTION_PURPOSE: This file set up timer 1 in mode 2 (8 bits auto reload
 * timer) with a software gate.
 * The 8-bits register consist of all 8 bits of TL1 and all 8 bits
 * of TH1 for the reload value. TH1 is load in TL1 at timer1 overflow.
 * FUNCTION_INPUTS: void
 * FUNCTION_OUTPUTS: void
 */

void main(void){
    TMOD &= 0x0F;
    TMOD |= 0x20; /* Gate = 0; C/T1# = 0; Mode 2 */
    TL1 = reload_value; /* Initial MSB value */
    TH1 = reload_value; /* Initial LSB value */
    ET1 = 1; /* Enable timer 1 interrupt */
    EA = 1; /* Interrupt Enable */
    TR1 = 1; /* Timer 1 run */

    while(1); /* Endless */

}

/**
 * FUNCTION_PURPOSE: timer1 interrupt
 * FUNCTION_INPUTS: void
 * FUNCTION_OUTPUTS: P1.0 toggle period = 2 * (256-reload_value) cycles
 */

void it_timer1(void) interrupt 3
{
    TF1 = 0; /* reset interrupt flag (already done by hardware) */
    P2_0 = ~P2_0; /* P2.0 toggle when interrupt. */
}
```

PWM Generation using Timer 1 in Auto-Reload Mode

```
#include <reg52x2.h>
#include <stdio.h>
#include <math.h>

unsigned int RLV=256;
unsigned int N=10;

void main(void){
    TMOD &=0x0F;
    TMOD |=0x20; /* Gate = 0; C/T1#=0; Mode 2 */
    TL1=RLV; /* Initial MSB value */
    TH1=RLV; /* Initial LSB value */
    ET1=1; /* Enable timer 1 interrupt */
    EA=1; /* Interrupt Enable */
    TR1=1; /* Timer 1 run */
    P2_0=0;

    while(1){
        if(P1_0==0)N++;
        if(P1_1==0)N--;
        if(N>250) N=250;
        if(N<10)N=10;
    }
}

void it_timer1(void) interrupt 3
{
    TF1 = 0;
    if(P2_0==0)
    {
        RLV=256-N;
    }
    else
    {
        RLV=N;
    }
    TH1=RLV;
    P2_0=~P2_0;
}
```

Experimental Varification

1. Build the complete circuit along with ISP and +5V power supply to be driven from a 230V/9V, 1A adaptor.
2. Open project using RIDE of Keil Vision.
3. Write the C program and build the Hex file.
4. Download the Hex file into AT89S52 microcontroller program (flash) memory using the ISP programmer and run the program on the microcontroller.
5. Record the PWM waveform frequency using a multi-meter and also an oscilloscope.
6. Record the duty cycle using a multi-meter and also an oscilloscope.
7. Set P1.0 = 0 for a while and monitor the PWM duty cycle.
8. Set P1.0 = 1.
9. Set P1.1 = 0 for a while and monitor the PWM duty cycle.
10. Fill the data readings into the data table I.

11. Modify the program so that the value of N can be adjusted by +20 or -20 over a second by pushing P1.0 or P1.1 to logical zero.
12. Modify the program and set the switching frequency to 1Hz.

DATA TABLE I

Timer 1 in Mode 2 (Auto Reload Mode with 8-bit Timer)					
Record No.	P1.0	P1.1	N	Switching Frequency (kHz)	PWM Duty Cycle (%)
1	1	1			
2	0	1			
3	1	0			

Record No.	P1.0	P1.1	Time (seconds)	N	Switching Frequency (kHz)	PWM Duty Cycle (%)
1	1	1	0			
2	0	1	0			
3	0	1	1			
4	0	1	2			
5	0	1	3			
6	1	0	4			
7	1	0	5			
8	1	0	6			
9	1	0	7			

Report

- (a) Derive equation for the switching frequency and duty cycle of the PWM waveform generated at P2.0.
- (b) Design a two digit seven segment display along with necessary interfaces to display the percent duty in the LED display.
- (c) Write a C program for the task described in (b).

Prepared By: Upal Mahbub, Masum Habib
Supervised By: Dr. Kazi Mujibur Rahman

Experiment 2

Generation of Control Sequence for a Stepper Motor using I/O Ports and Interrupts.

Objective

The objective of this experiment is to use the ports of AT89S52 microcontroller for both input and output. A Stepper Motor is to be controlled implying this knowledge. External Interrupts will also be used to control the motor rotation.

Microcontroller Resources to be used

The following resources would be needed to generate PWM waveforms using AT89S52.

- (a) Ports of the microcontroller
- (b) External Interrupt pins
- (c) IE (Interrupt Enable Register)
- (d) On chip clock generator

Port Structures and Operation

All four ports in the AT89C51/52 are bidirectional. Each consists of a latch (Special Function Registers P0 through P3), an output driver, and an input buffer. The output drivers of Ports 0 and 2, and the input buffers of Port 0, are used in accesses to external memory. In this application, Port 0 outputs the low byte of the external memory address, time-multiplexed with the byte being written or read. Port 2 outputs the high byte of the external memory address when the address is 16 bits wide. Otherwise the Port 2 pins continue to emit the P2 SFR content.

All the Port 3 pins, and two Port 1 pins (in the AT89C52) are multifunctional. They are not only port pins, but also provide the special features listed in the table below.

Table 2-1 Port pin alternate functions

Port Pin	Alternate Function
P1.0 ⁽¹⁾	T2 (Timer/Counter 2 external input)
P1.1 ⁽¹⁾	T2EX (Timer/Counter 2 Capture/Reload trigger)
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt)
P3.3	INT1 (external interrupt)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	WR (external data memory write strobe)
P3.7	RD (external data memory read strobe)

Note: 1. P1.0 and P1.1 serve these alternate functions only on the AT89C52. The alternate functions can only be activated if the corresponding bit latch in the port SFR contains a 1. Otherwise the port pin is stuck at 0.

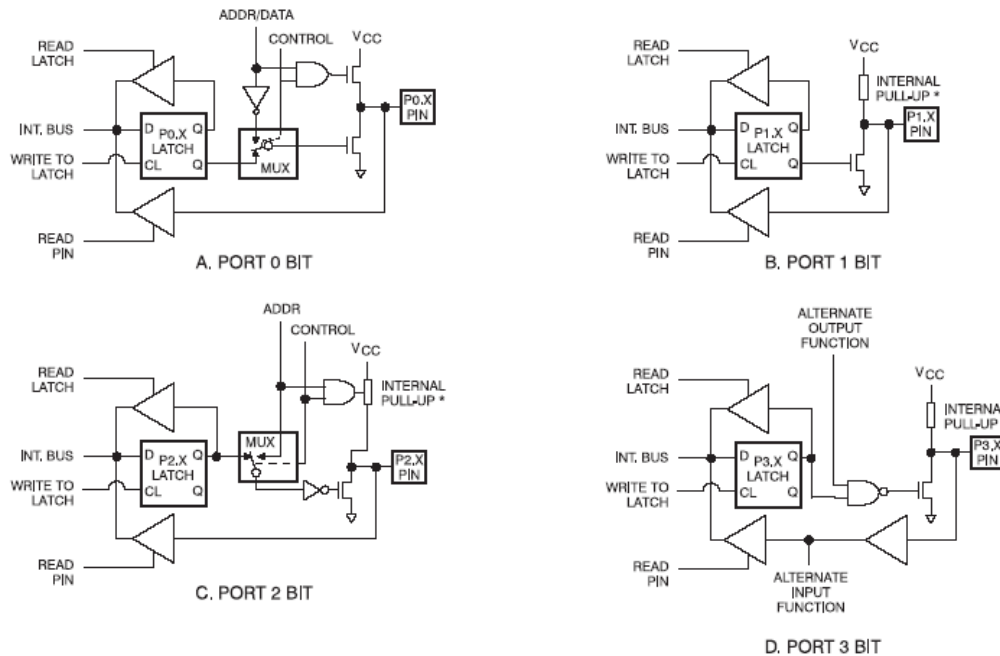
I/O Configurations

Figure 2-1 AT89C51 and AT89C52 Port Bit Latches and I/O Buffers

Interrupts

The AT89C52 has a total of six interrupt vectors: two external interrupts (INT0 and INT1), three timer interrupts (Timers 0, 1, and 2), and the serial port interrupt. These interrupts are all shown in Figure 2-2. Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE. IE also contains a global disable bit, EA, which disables all interrupts at once.

Table 2-2 Interrupt Enable (IE) Register

(MSB)							(LSB)
EA	–	ET2	ES	ET1	EX1	ET0	EX0
Enable Bit = 1 enables the interrupt, Enable Bit = 0 disables the interrupt.							
Symbol	Position	Function					
EA	IE.7	Disables all interrupts. If EA = 0, no interrupt is acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.					
–	IE.6	Reserved.					
ET2	IE.5	Timer 2 interrupt enable bit.					
ES	IE.4	Serial Port interrupt enable bit.					
ET1	IE.3	Timer 1 interrupt enable bit.					
EX1	IE.2	External interrupt 1 enable bit.					
ET0	IE.1	Timer 0 interrupt enable bit.					
EX0	IE.0	External interrupt 0 enable bit.					

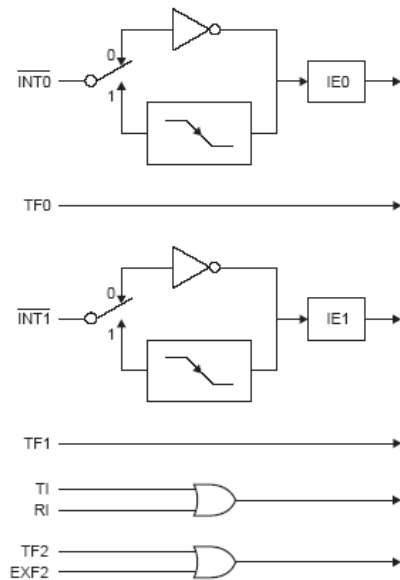


Figure 2-2 Interrupt Sources in AT89S52

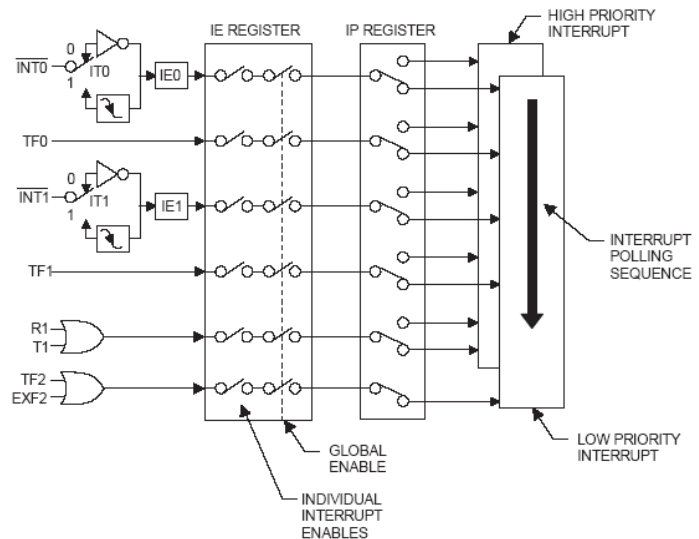


Figure 2-3 AT89S52 Interrupt Control System

Table 2-3 Interrupt Priority (IP) Register

(MSB)							(LSB)
—	—	PT2	PS	PT1	PX1	PT0	PX0
Priority bit = 1 assigns high priority. Priority bit = 0 assigns low priority.							

Symbol	Position	Function
—	IP.7	reserved
—	IP.6	reserved
PT2	IP.5	Timer 2 Interrupt priority bit.
PS	IP.4	Serial Port Interrupt priority bit.
PT1	IP.3	Timer 1 Interrupt priority bit.
PX1	IP.2	External Interrupt 1 priority bit.
PT0	IP.1	Timer 0 Interrupt priority bit.
PX0	IP.0	External Interrupt 0 priority bit.
User software should never write 1s to unimplemented bits, since they may be used in future AT89 Series products.		

Priority Level Structure

	Source	Priority within Level
1.	IE0	(highest)
2.	TF0	
3.	IE1	
4.	TF1	
5.	RI + TI	
6.	TF2 + EXF2	(lowest)

How Interrupts are Handled

The processor acknowledges an interrupt request by executing a hardware-generated LCALL to the appropriate servicing routine. In some cases it also clears the flag that generated the interrupt, and in other cases it does not. It never clears the Serial Port or Timer 2 flags. This must be done in the user's software. The processor clears an external interrupt flag (IE0 or IE1) only if it was transition-activated.

The hardware-generated LCALL pushes the contents of the Program Counter onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt being serviced, as shown in the following table.

Table 2-4 Interrupt sources and vector addresses

Interrupt	Source	Vector Address
External 0	IE0	0003H
Timer 0	TF0	000BH
External 1	IE1	0013H
Timer 1	TF1	001BH
Serial Port	RI or TI	0023H
Timer 2	TF2 or EXF2	002BH
System Reset	RST	0000H

Note: When vectoring to an interrupt the flag that caused the interrupt is automatically cleared by hardware. The exceptions are RI and TI for serial port interrupts, and TF2 and EXF2 for Timer 2 interrupt. Since there are two possible sources for each of these interrupts, it is not practical for the CPU to clear the interrupt flag. These bits must be tested in the ISR to determine the source of the interrupt, and then the interrupting flag is cleared by software.

Table 2-5 Interrupt Flag Bits

Interrupt	Flag	SFR Register and Bit Position
External 0	IE0	TCON.1
External 1	IE1	TCON.3
Timer 1	TF1	TCON.7
Timer 0	TF0	TCON.5
Serial port	TI	SCON.1
Serial port	RI	SCON.0
Timer 2	TF2	T2CON.7 (AT89C52)
Timer 2	EXF2	T2CON.6 (AT89C52)

External Interrupts

The external sources can be programmed to be level-activated or transition-activated by setting or clearing bit IT1 or IT0 in Register TCON. If $ITx = 0$, external interrupt x is triggered by a detected low at the $INTx$ pin. If $ITx = 1$, external interrupt x is edge-triggered. In this mode if successive samples of the $INTx$ pin show a high in one cycle and a low in the next cycle, interrupt request flag IEx in TCON is set.

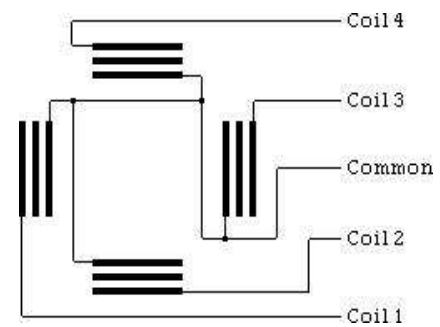
Flag bit IEx then requests the interrupt. Since the external interrupt pins are sampled once each machine cycle, an input high or low should hold for at least 12 oscillator periods to ensure sampling. If the external interrupt is transition-activated, the external source has to hold the request pin high for at least one machine cycle, and then hold it low for at least one machine cycle to ensure that the transition is seen so that interrupt request flag IEx will be set. IEx will be automatically cleared by the CPU when the service routine is called.

If the external interrupt is level-activated, the external source has to hold the request active until the requested interrupt is actually generated. Then the external source must deactivate the request before the interrupt service routine is completed, or else another interrupt will be generated.

Stepper Motor

Four wire coils are found inside a stepper motor, positioned 90 degrees apart, like a clock at 12, 3, 6, and 9. It has a spinning rotor at the middle fitted with permanent magnets around its circumference. In each turn that the rotor spins, the magnets pass, approach, and move away from the four coils. As each magnet passes a wire coil, it causes the flow of electricity through the coil. Alternate electrical currents then flow through the coils in various amounts.

As a multi-phase motor, the stepper motor is ideal for electricity generation projects. Its four-phase motor and allows electricity to continuously flow to its maximum capacity to the next coil, even if electricity fails to flow in one coil. The total electricity generated from the four-phases merged and is rectified to produce a near constant voltage and direct current (DC).



Schematic of a Stepper Motor coils

Systematically use a multimeter to measure the resistance between different pairs of wires. All four coils will have near identical resistances - if they did not the motor would not function properly. Therefore if the pair of wires being measured are both *live*, the resistance measured will be double that measured if one of the wires is a *common*. Why is this? Because two live wires have two coils between them whereas a common and a live have just one coil between them.

C Program

```
#include <reg52x2.h>
#include <stdio.h>
#include <math.h>

void RunStepper(void);

void delay(unsigned int delaysms){
int variabl;
    while(delaysms--){
        variabl = 290;
        while(variabl--);
    }
}

void main(void)
{
    while(1){
        if(P1_1==1){
            RunStepper();
        }
    }
}

void RunStepper(void){
unsigned char i;
unsigned char step[]={1,2,4,8};

    for(i=1;i<=32;i++){
        P2=step[i%4];
        delay(1000);
    }
}
```

Experimental Verification

1. Build the complete circuit along with ISP and +5V power supply to be driven from a 230V/9V, 1A adaptor.
2. Open project using RIDE of Keil Vision.
3. Write the C program and build the Hex file.
4. Download the Hex file into AT89S52 microcontroller program (flash) memory using the ISP programmer and run the program on the microcontroller.
5. Connect 4 LEDs in Port 2's Lower Significant 4 pins.
6. Make P1.1=1 and see if the stepper motor sequence works.
7. Modify the code for rotating the motor more precisely, i.e. the motor should rotate smaller angle per movement.
8. Modify the code so that if P1.0=1 while P1.1=0 then the motor will rotate inversely.

Motor Control using Interrupt

```
#include <reg52x2.h>
#include <stdio.h>
#include <math.h>

void delay(unsigned int delaysms){
int variabl;
    while(delaysms--){
        variabl = 290;
        while(variabl--);
    }
}

void main(void)
{
IT0 = 1;
EX0 = 1;
EA = 1;
while(1);
}

void int_ext0(void) interrupt 0
{
    unsigned char i;
    unsigned char step[]={1,2,4,8};

    for(i=1;i<=32;i++){
        P2=step[i%4];
        delay(1000);
    }
}
```

Experimental Verification

1. Write the C program and build the Hex file.
2. Download the Hex file into AT89S52 microcontroller program (flash) memory using the ISP programmer and run the program on the microcontroller.
3. Set P3_2 = 1 to give external interrupt 0 and see if the motor runs.
4. Configure the code so that the motor will run inversely if external interrupt 1 happens.

Report

1. Design a system to control the direction of rotation of a Stepper Motor using P1.0 and P1.1 and the two interrupts. The design should meet the following requirements:
 - i. The motor will start running clockwise if P1.0 or P1.1 is set.
 - ii. If both P1.0 and P1.1 are set, the motor will start rotating counter clockwise.
 - iii. If external interrupt 0 is given the motor will rotate at a higher degree with each pulse.
 - iv. If external interrupt 1 is given the motor will traverse smaller angle with each pulse.
2. Draw the Schematic diagram of the system with the stepper motor and necessary switches. Do you think any of the microcontroller's port can give the motor the proper current supply? If yes, then which one and why? If no, how can you supply the motor proper current?

Prepared By: Upal Mahbub
Supervised By: Dr. Kazi Mujibur Rahman

Experiment 4

PWM pattern generation using Timer 2 of AT89S52.

2.1 Objective

The objective of this experiment is to generate PWM pattern with the help of timers available in MCS 51 microcontroller. PWM with variable duty cycle will be generated by Timer 2 in 16-bit auto reload mode.

2.2 Microcontroller Resources to be used

The following resources would be needed to generate PWM waveforms using AT89S52.

1. Timer 2
2. RCAP2L and RCAP2H registers
3. TMOD and TCON registers
4. Interrupts (related to timer 2)
5. Ports
6. On chip clock generator

2.3 Configuring the Microcontroller

Timer 2 is a 16-bit timer/counter which is present in AT89C52 but not in AT89C51. The count is maintained by two 8-bit timer registers, TH2 and TL2, that are cascade connected. Like Timers 0 and 1, it can operate either as a timer or as an event counter.

It is controlled by the T2CON register (see table 2.2) and the T2MOD register. Timer 2 operation is similar to Timer 0 and Timer 1. C/T2 selects FOSC/6 (timer operation) or external pin T2 (counter operation) as timer register input. Setting TR2 allows TL2 to be incremented by the selected input. It has three operating modes: 'capture', 'autoload' and 'baud rate generator', which are selected by bits in T2CON as shown in Table 2.2.

Table 2.1 Timer 2 Operation Modes

RCLK + TCLK	CP/RL2	TR2	Mode
0	0	1	16-bit auto-reload
0	1	1	16-bit capture
1	X	1	baud rate generator
X	X	0	(off)

A. Auto-reload (Up or Down Counter)

Timer 2 can be programmed to count up or down when configured in its 16-bit auto reload mode. This feature is invoked by the DCEN (Down Counter Enable) bit located in the SFR T2MOD (see Table 4.3). Upon reset, the DCEN bit is set to 0 so that timer 2 will default to count up. When DCEN is set, Timer 2 can count up or down, depending on the value of the T2EX pin.

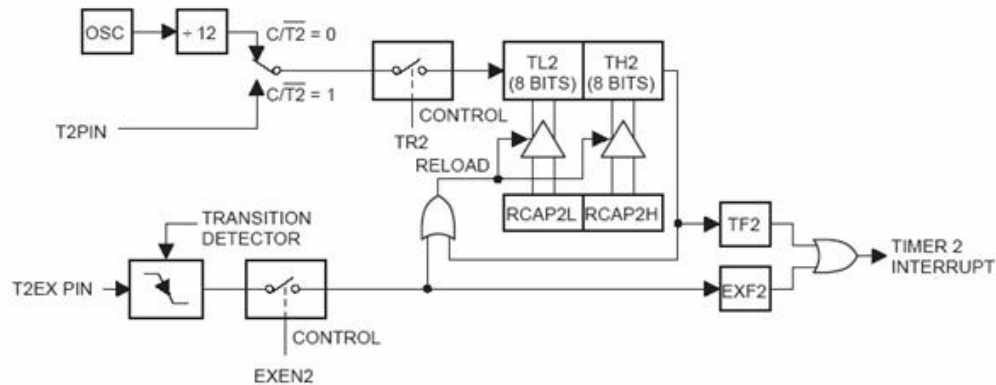


Figure 4.1: Timer 2 Auto Reload Mode (DCEN = 0)

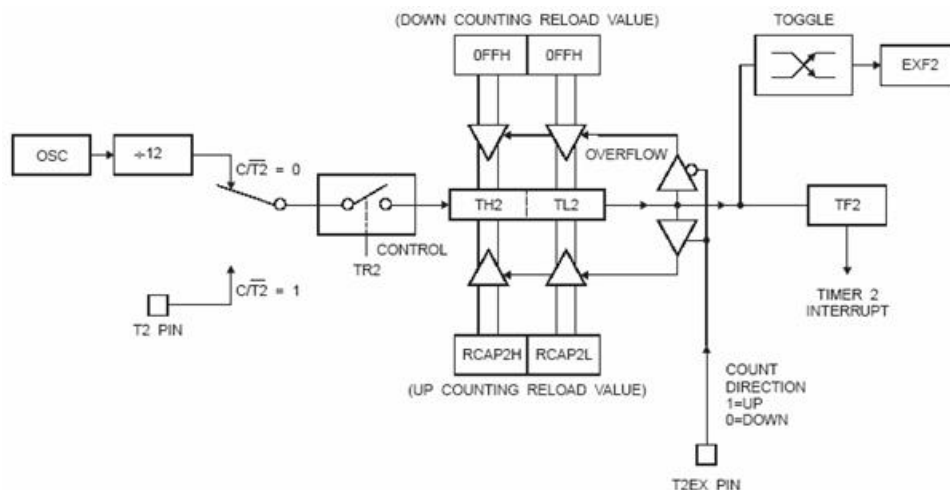


Figure 4.2: Timer 2 Auto Reload Mode (DCEN = 1)

Setting the DCEN bit enables timer 2 to count up or down as shown in Figure 4.1. In this mode the T2EX pin controls the counting direction. When T2EX is high, timer 2 counts up. Timer overflow occurs at FFFFh which sets the TF2 flag and generates an interrupt request. The overflow also causes the 16-bit value in the RCAP2H and RCAP2L registers to be loaded into the timer registers TH2 and TL2.

When T2EX is low, timer 2 counts down. Timer underflow occurs when the count in the timer registers, TH2 and TL2, equals the value stored in the RCAP2H and RCAP2L registers. The underflow sets TF2 flag and reloads FFFFh into the timer registers. The EXF2 bit toggles when timer 2 overflow or underflow occurs, depending on the direction of the count. EXF2 does not generate an interrupt. This bit can be used to provide 17-bit resolution.

Table 4.2 T2CON Register -- Timer 2 Control Register

7	6	5	4	3	2	1	0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2#	CP/RL2#
Bit Number	Bit Mnemonic	Description					
7	TF2	Timer 2 overflow Flag TF2 is not set if RCLK=1 or TCLK = 1. Must be cleared by software. Set by hardware on timer 2 overflow.					
6	EXF2	Timer 2 External Flag Set when a capture or a reload is caused by a negative transition on T2EX pin if EXEN2=1. Set to cause the CPU to vector to timer 2 interrupt routine when timer 2 interrupt is enabled. Must be cleared by software.					
5	RCLK	Receive Clock bit Clear to use timer 1 overflow as receive clock for serial port in mode 1 or 3. Set to use timer 2 overflow as receive clock for serial port in mode 1 or 3.					
4	TCLK	Transmit Clock bit Clear to use timer 1 overflow as transmit clock for serial port in mode 1 or 3. Set to use timer 2 overflow as transmit clock for serial port in mode 1 or 3.					
3	EXEN2	Timer 2 External Enable bit Clear to ignore events on T2EX pin for timer 2 operation. Set to cause a capture or reload when a negative transition on T2EX pin detected, if timer 2 is not used to clock the serial port.					
2	TR2	Timer 2 Run control bit Clear to turn off timer 2. Set to turn on timer 2.					
1	C/T2#	Timer/Counter 2 select bit Clear for timer operation (input from internal clock system: FOSC). Set for counter operation (input from T2 input pin).					
0	CP/RL2#	Timer 2 Capture/Reload bit If RCLK=1 or TCLK=1, CP/RL2# is ignored and timer is forced to auto-reload on timer 2 overflow. Clear to auto-reload on timer 2 overflows or negative transitions on T2EX pin if EXEN2=1. Set to capture on negative transitions on T2EX pin if EXEN2=1.					

Reset Value = 0000 0000b

Table 5-4 T2MOD – Timer 2 Mode Control Register

T2MOD Address = 0C9H

Reset Value = XXXX XX00B

	–	–	–	–	–	–	T2OE	DCEN
Bit	7	6	5	4	3	2	1	0
Symbol				Function				
–				Not implemented, reserved for future				
T2OE				Timer 2 Output Enable bit.				
DCEN				When set, this bit allows Timer 2 to be configured as an up/down counter.				

4.4 C Programs

A. Simple 16-bit Auto Reload with Timer 2

```
#include <reg52x2.h>

#define MSB_reload_value 0x36 /* msb reload value exemple */
#define LSB_reload_value 0x36 /* lsb reload value exemple */

/**
 * FUNCTION_PURPOSE: This file set up timer 2 in mode 0 (16 bits auto-reload
 * up/down counting timer).
 * The 16-bits register consist of all 8 bits of TH2 and all 8 bits
 * of TL2. The EXF2 bit toggles when timer2 overflow or underflow occurs.
 * EXF2 does not generate interrupt. This bit can be used to provide 17-bit
 * resolution
 * FUNCTION_INPUTS: P1.1(T2EX)=0 for down counting or 1 for up counting.
 * FUNCTION_OUTPUTS: void
 */
void main(void)
{
    T2MOD &= 0xFC; /* T2OE=0;DCEN=0; */
    T2MOD |= 0x01; /* DCEN=1; */
    EXF2=0; /* reset flag */
    TCLK=0;RCLK=0; /* disable baud rate generator */
    EXEN2=0; /* ignore events on T2EX */
    TH2=MSB_reload_value; /* Init msb_value */
    TL2=LSB_reload_value; /* Init lsb_value */
    RCAP2H=MSB_reload_value; /* reload msb_value */
    RCAP2L=LSB_reload_value; /* reload lsb_value */
    C_T2=0; /* timer mode */
    CP_RL2=0; /* reload mode */
    EA=1; /* interrupt enable */
    ET2=1; /* enable timer2 interrupt */
    TR2=1; /* timer2 run */
    while(1); /* endless */
}

/**
 * FUNCTION_PURPOSE: timer2 interrupt
 * FUNCTION_INPUTS: void
 * FUNCTION_OUTPUTS: P1.2 toggle period = 2 * (65536-reload_value) cycles
 */
void it_timer2(void) interrupt 5 /* interrupt address is 0x002b */
{
    P1_2 = ~P1_2; /* P1.2 toggle when interrupt. */
    TF2 = 0; /* reset interrupt flag */
}
```

```
#include <reg52x2.h>

unsigned int N = 5000;
unsigned char MSB_reload_value;
unsigned char LSB_reload_value;
unsigned int count = 1;
void main(void)
{
    T2MOD &= 0xFC; /* T2OE=0;DCEN=0; */
```

```
EXF2=0; /* reset flag */
TCLK=0;RCLK=0; /* disable baud rate generator */
EXEN2=0; /* ignore events on T2EX */
TH2=MSB_reload_value; /* Init msb_value */
TL2=LSB_reload_value; /* Init lsb_value */
RCAP2H=MSB_reload_value; /* reload msb_value */
RCAP2L=LSB_reload_value; /* reload lsb_value */
C_T2=0; /* timer mode */
CP_RL2=0; /* reload mode */
EA=1; /* interrupt enable */
ET2=1; /* enable timer2 interrupt */
TR2=1; /* timer2 run */
while(1)
{
    if (P1_0 == 0) N++;
    if (P1_1 == 0) N--;
    if (N>65000) N = 65000; //Ceiling
    if (N<500) N = 500; //Floor Value

    MSB_reload_value = ((65536 - N) & 0xff00) >> 8;
    LSB_reload_value = ((65536 - N) & 0x00ff);
}; /*endless*/
}

void it_timer2(void) interrupt 5 /* interrupt address is 0x002b */
{
    TF2 = 0; /* reset interrupt flag */
    count++;
    if (count == 3) P2_0 = 1;
    if (count == 5)
    {
        P2_0 = 0 ;
        count = 1;
    }
}
```

Experimental Varification

13. Build the complete circuit along with ISP and +5V power supply to be driven from a 230V/9V, 1A adaptor.
14. Open project using RIDE of Keil Vision.
15. Write the C program and build the Hex file.
16. Download the Hex file into AT89S52 microcontroller program (flash) memory using the ISP programmer and run the program on the microcontroller.
17. Record the PWM waveform frequency using a multi-meter and also an oscilloscope.
18. Record the duty cycle using a multi-meter and also an oscilloscope.
19. Set P1.0 = 0 for a while and monitor the PWM duty cycle.
20. Set P1.0 = 1.
21. Set P1.1 = 0 for a while and monitor the PWM duty cycle.
22. Fill the data readings into the data table I.
23. Modify the program so that the value of N can be adjusted by +20 or -20 over a second by pushing P1.0 or P1.1 to logical zero.
24. Modify the program and set the switching frequency to 1Hz.

DATA TABLE I

Timer 1 in Mode 2 (Auto Reload Mode with 8-bit Timer)					
Record No.	P1.0	P1.1	N	Switching Frequency (kHz)	PWM Duty Cycle (%)
1	1	1			
2	0	1			
3	1	0			

Record No.	P1.0	P1.1	Time (seconds)	N	Switching Frequency (kHz)	PWM Duty Cycle (%)
1	1	1	0			
2	0	1	0			
3	0	1	1			
4	0	1	2			
5	0	1	3			
6	1	0	4			
7	1	0	5			
8	1	0	6			
9	1	0	7			

Report

- (d) Derive equation for the switching frequency and duty cycle of the PWM waveform generated at P2.0.
- (e) Design a two digit seven segment display along with necessary interfaces to display the percent duty in the LED display.
- (f) Write a C program for the task described in (b).

Prepared By: Upal Mahbub, Masum Habib
Supervised By: Dr. Kazi Mujibur Rahman

Experiment 5

Serial Communication with AT89S52 Microcontroller's UART feature.

5.1 Objective

The objective of this experiment is to connect two AT89S52 microcontrollers and encode them in proper means to serially communicate with each other using the UART feature.

5.2 Microcontroller Resources to be used

The following resources would be needed for serial communication between two AT89S52.

1. Timer 1
2. SCON and TMOD registers
3. Interrupts (related to serial communication)
4. Ports and TXD, RXD pins
5. On chip clock generator

5.3 Serial Interface

The serial interface provides both synchronous and asynchronous communication modes. It operates as a Universal Asynchronous Receiver and Transmitter (UART) in three full-duplex modes (Modes 1, 2 and 3). Asynchronous transmission and reception can occur simultaneously and at different baud rates.

It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. (However, if the first byte still hasn't been read by the time reception of the second byte is complete, one of the bytes will be lost). The serial port receive and transmit registers are both accessed at Special Function Register SBUF. Writing to SBUF loads the transmit register, and reading SBUF accesses a physically second receive register.

The serial port can operate in 4 modes:

Mode 0: Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

Mode 1: 10 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in Special Function Register SCON. The baud rate is variable.

Mode 2: 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8 in SCON) can be assigned the value of 0 or 1. Or, for example, the parity bit (P, in the PSW) could be moved into TB8. On receive, the 9th data bit goes into RB8 in Special Function register SCON, while the stop bit is ignored. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency.

Mode 3: 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit and a stop bit (1). In fact, Mode 3 is the same as Mode 2 in all respects except the baud rate. The baud rate in Mode 3 is variable.

In all four modes, transmission is initiated in Mode 0 by the condition RI = 0 and REN = 1. Reception is initiated in Mode 0 by the condition RI = 0 and REN = 1. Reception is initiated in the other modes by the incoming start bit if REN = 1.

Serial I/O port includes the following enhancements:

- Framing error detection
- Automatic address recognition

5.4 Serial Port Control Register

The serial port control and status register is the Special Function Register SCON, shown in Table 5-1. This register contains not only the mode selection bits, but also the 9th data bit for transmit and receive (TB8 and RB8), and the serial port interrupts bits (TI and RI).

Table 5-1 SCON: Serial Port Control Register:

FE/SM0	SM1	SM2	REN	TB8	RB8	TI	RI
Bit Position	Symbol	Description					
SCON.7	FE	Framing Error bit (SMOD0=1) Clear to reset the error state, not cleared by a valid stop bit. Set by hardware when an invalid stop bit is detected. SMOD0 must be set to enable access to the FE bit					
	SM0	Serial port Mode bit 0 Refer to SM1 for serial port mode selection. SMOD0 must be cleared to enable access to the SM0 bit					
SCON.6	SM1	Serial port Mode bit 1					
		SM0	SM1	Mode0	Description	Baud Rate	
		0	0	0	Shift Register	FCPU PERIPH/6	
		0	1	1	8-bit UART	Variable (set by timer)	
		1	0	2	9-bit UART	FCPU PERIPH /32 or /16	
		1	1	3	9-bit UART	Variable (set by timer)	
SCON.5	SM2	Serial port Mode 2 bit / Multiprocessor Communication Enable bit Clear to disable multiprocessor communication feature. Set to enable multiprocessor communication feature in mode 2 and 3, and eventually mode 1. This bit should be cleared in mode 0.					
SCON.4	REN	Reception Enable bit Clear to disable serial reception. Set to enable serial reception.					
SCON.3	TB8	Transmitter Bit 8 / Ninth bit to transmit in modes 2 and 3, transmit a logic 0 in the 9th bit. Set to transmit a logic 1 in the 9th bit.					
SCON.2	RB8	Receiver Bit 8 / Ninth bit received in modes 2 and 3 Cleared by hardware if 9th bit received is a logic 0. Set by hardware if 9th bit received is a logic 1. In mode 1, if SM2 = 0, RB8 is the received stop bit. In mode 0 RB8 is not used.					
SCON.1	TI	Transmit Interrupt flag Clear to acknowledge interrupt. Set by hardware at the end of the 8th bit time in mode 0 or at the beginning of the stop bit in the other modes.					
SCON.0	RI	Receive Interrupt flag Clear to acknowledge interrupt. Set by hardware at the end of the 8th bit time in mode 0, see Figure 2-26 and Figure 2-27. in the other modes.					

5.5 Setting the Serial Port Baud Rate

Once the Serial Port Mode has been configured the program must configure the serial ports baud rate. This only applies to Serial Port modes 1 and 3. The Baud Rate is determined based on the oscillator's frequency when in mode 0 and 2. In mode 0, the baud rate is always the oscillator frequency divided by 12. This means if your crystal is

11.059 MHz, mode 0 baud rate will always be 921,583 baud. In mode 2 the baud rate is always the oscillator frequency divided by 64, so an 11.059 MHz crystal speed will yield a baud rate of 172,797.

In modes 1 and 3, the baud rate is determined by how frequently timer 1 overflows. The more frequently timer 1 overflows, the higher the baud rate. There are many ways one can cause timer 1 to overflow at a rate that determines a baud rate, but the most common method is to put timer 1 in 8-bit auto-reload mode (timer mode 2) and set a reload value (TH1) that causes Timer 1 to overflow at a frequency appropriate to generate a baud rate.

To determine the value that must be placed in TH1 to generate a given baud rate, we may use the following equation (assuming PCON.7 is clear).

$$TH1 = 256 - ((Crystal / 384) / Baud)$$

If PCON.7 is set then the baud rate is effectively doubled, thus the equation becomes:

$$TH1 = 256 - ((Crystal / 192) / Baud)$$

For example, if we have an 11.059 MHz crystal and we want to configure the serial port to 19,200 baud we try plugging it in the first equation:

$$\begin{aligned} TH1 &= 256 - ((Crystal / 384) / Baud) \\ TH1 &= 256 - ((11059000 / 384) / 19200) \\ TH1 &= 256 - ((28,799) / 19200) \\ TH1 &= 256 - 1.5 = 254.5 \end{aligned}$$

As you can see, to obtain 19,200 baud on a 11.059Mhz crystal we have to set TH1 to 254.5. If we set it to 254 we will have achieved 14,400 baud and if we set it to 255 we will have achieved 28,800 baud. Thus this will not work properly. One way to achieve 19,200 baud is to set PCON.7 (SMOD). When we do this we double the baud rate and utilize the second equation mentioned above. Thus we have:

$$\begin{aligned} TH1 &= 256 - ((Crystal / 192) / Baud) \\ TH1 &= 256 - ((11059000 / 192) / 19200) \\ TH1 &= 256 - ((57699) / 19200) \\ TH1 &= 256 - 3 = 253 \end{aligned}$$

Here we are able to calculate a nice, even TH1 value. Therefore, to obtain 19,200 baud with an 11.059MHz crystal we must:

1. Configure Serial Port mode 1 or 3.
2. Configure Timer 1 to timer mode 2 (8-bit auto-reload).
3. Set TH1 to 253 to reflect the correct frequency for 19,200 baud.
4. Set PCON.7 (SMOD) to double the baud rate.

5.6 Using Timer 1 to Generate Baud Rates

When Timer 1 is used as the baud rate generator, the baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate and the value of SMOD as follows:

The Timer 1 interrupt should be disabled in this application. The Timer itself can be configured for either “timer” or “counter” operation, and in any of its 3 running modes. In the most typical applications, it is configured for “timer” operation, in the auto-reload mode (high nibble of TMOD = 0010B). In that case, the baud rate is given by the formula One can achieve very low baud rates with Timer 1 by leaving the Timer 1 interrupt enabled, and configuring the Timer to run as a 16-bit timer (high nibble of TMOD = 0001B), and using the Timer 1 interrupt to do a 16-bit software reload. Table 5-2 lists various commonly used baud rates and how they can be obtained from Timer 1.

Table 5-2 Timer 1 Generated Commonly Used Baud Rates

Fosc (MHz)	11.0592	12	14.7456	16	20	SMOD
Baudrate						
150	40h	30h	00h			0
300	A0h	98h	80h	75h	52h	0
600	D0h	CCh	C0h	BBh	A9h	0
1200	E8h	E6h	E0h	DEh	D5h	0
2400	F4h	F3h	F0h	EFh	EAh	0
4800		F3h	EFh	EFh		1
4800	FAh		F8h		F5h	0
9600	FDh		FCh			0
9600					F5h	1
19200	FDh		FCh			1
38400			FEh			
76800			FFh			

5.7 Lab Work

Code Example

```

/**
 * @file $RCSfile: uart_t1.c,v $
 * Copyright (c) 2004 Atmel.
 * Please read file license.txt for copyright notice.
 * @brief This file is an example to use uart with timer1.
 * UART will echo a received data.
 * This file can be parsed by Doxygen for automatic documentation
 * generation.
 * Put here the functional description of this file within the software
 * architecture of your program.
 * @version $Revision: 1.0 $ $Name: $
 */
/* @section I N C L U D E S */

#include "reg_c51.h"

char uart_data;

/**
 * FUNCTION_PURPOSE: This file set up uart in mode 1 (8 bits uart) with
 * timer 1 in mode 2 (8 bits auto reload timer).
 * FUNCTION_INPUTS: void
 * FUNCTION_OUTPUTS: void
 */
void main (void)
{

```

```

SCON = 0x50; /* uart in mode 1 (8 bit), REN=1 */
TMOD = TMOD | 0x20; /* Timer 1 in mode 2 */
TH1 = 0xFD; /* 9600 Bds at 11.059MHz */
TL1 = 0xFD; /* 9600 Bds at 11.059MHz */
ES = 1; /* Enable serial interrupt */
EA = 1; /* Enable global interrupt */
TR1 = 1; /* Timer 1 run */
while(1) /* endless */
{
}
/**
 * FUNCTION_PURPOSE: serial interrupt, echo received data.
 * FUNCTION_INPUTS: P3.0(RXD) serial input
 * FUNCTION_OUTPUTS: P3.1(TXD) serial output
 */

void serial_IT(void) interrupt 4
{
    if (RI == 1)
    { /* if reception occur */
        RI = 0; /* clear reception flag for next reception */
        uart_data = SBUF; /* Read receive data */
        SBUF = uart_data; /* Send back same data on uart */
    }
    else TI = 0; /* if emission occur */
} /* clear emission flag for next emission */

```

The above given code is a generalized serial communication program code for UART. The microcontroller containing the above code will receive data using RXD and then reflect it to the sending side through TXD. The code is written for 9600 Boudrate at 11.059 MHz crystal. Complete the following tasks:

1. Make necessary changes in the code to make it work for 4800 Boudrate at 20 MHz.
2. Change the given code so that the received data will be shown in P2.
3. Write a code for the sender microcontroller. The sender should send the value of P2 through TXD. It should only transmit when P1_0 is 0.
4. Verify the code in Keil/Ride and also in a real time simulation software (Proteus or Circuit Maker).
5. Download the codes in two different microcontrollers.
6. Connect the basic circuit to each microcontroller and connect the TXD to RXD of one another.
7. Verify if the code works.

5.8 Report

Design a system for controlling two LEDs connected to a Microcontroller by three switches connected to another Microcontroller. The switches should control the LEDs in the following pattern:

SW0	SW1	SW2	Action
0	0	0	Both LEDs are off.
0	0	1	LED 2 is on.
0	1	0	LED 1 is on.
0	1	1	Both LEDs are on.
1	X	X	Hold the previous states.

Prepared By: Upal Mahbub
Supervised By: Dr. Kazi Mujibur Rahman

Experiment 6

Interfacing Liquid Crystal Display (LCD) module with AT89S52 microcontroller.

6.1 Objective

The objective of this experiment is interface an 2 X 16 line LCD with At89s52 and exploring the entry mode, display shift, character generation and some other operations.

6.2 Microcontroller Resources to be used

The following resources would be needed for serial communication between two AT89S52.

1. LCD module
2. Ports
3. On chip clock generator

6.3 LCD : Basic Concepts

Liquid Crystal Display also called as LCD is very helpful in providing user interface as well as for debugging purpose. The most common type of LCD controller is HITACHI 44780 which provides a simple interface between the controller & an LCD. These LCD's are very simple to interface with the controller as well as are cost effective.



The most commonly used *ALPHANUMERIC* displays are *1x16* (Single Line & 16 characters), *2x16* (Double Line & 16 character per line) & *4x20* (four lines & Twenty characters per line).

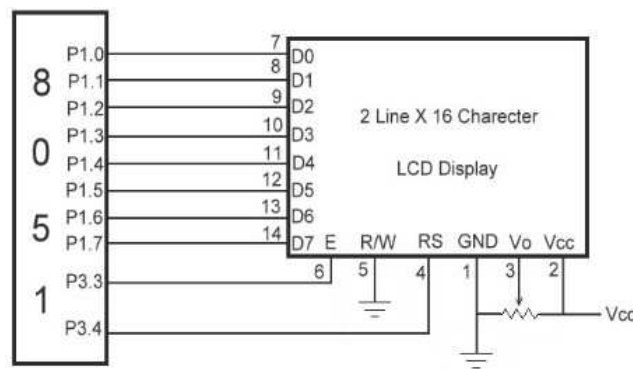
The LCD requires 3 control lines (RS, R/W & EN) & 8 (or 4) data lines. The number on data lines depends on the mode of operation. If operated in 8-bit mode then 8 data lines + 3 control lines i.e. total 11 lines are required. And if operated in 4-bit mode then 4 data lines + 3 control lines i.e. 7 lines are required. How do we decide which mode to use? It's simple if you have sufficient data lines you can go for 8 bit mode & if there is a time constrain i.e. display should be faster then we have to use 8-bit mode because basically 4-bit mode takes twice as more time as compared to 8-bit mode.

Pin	Symbol	Function
1	Vss	Ground
2	Vdd	Supply Voltage
3	Vo	Contrast Setting
4	RS	Register Select
5	R/W	Read/Write Select
6	En	Chip Enable Signal
7-14	DB0-DB7	Data Lines
15	A/Vee	Gnd for the backlight
16	K	Vcc for backlight

When **RS** is low (0), the data is to be treated as a command. When RS is high (1), the data being sent is considered as text data which should be displayed on the screen.

When **R/W** is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively reading from the LCD. Most of the times there is no need to read from the LCD so this line can directly be connected to Gnd thus saving one controller line.

The **ENABLE** pin is used to latch the data present on the data pins. A HIGH - LOW signal is required to latch the data. The LCD interprets and executes our command at the instant the EN line is brought low. If you never bring EN low, your instruction will never be executed.



For Contrast setting a 10K pot should be used as shown in the figure.

Display Data RAM (DDRAM)

Display data RAM (DDRAM) is where you send the characters (ASCII code) you want to see on the LCD screen. It stores display data represented in 8-bit character codes. Its capacity is 80 characters (bytes). Below you see DD RAM address layout of a 2*16 LCD.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	+ Character position (dec.)									
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	+ Row0 DDRAM address (hex)									
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67	+ Row1 DDRAM address (hex)									

In the above memory map, the area shaded in black is the visible display (For 16x2 display).

For first line addresses for first 15 characters is from 00h to 0Fh. But for second line address of first character is 40h and so on up to 4Fh for the 16th character.

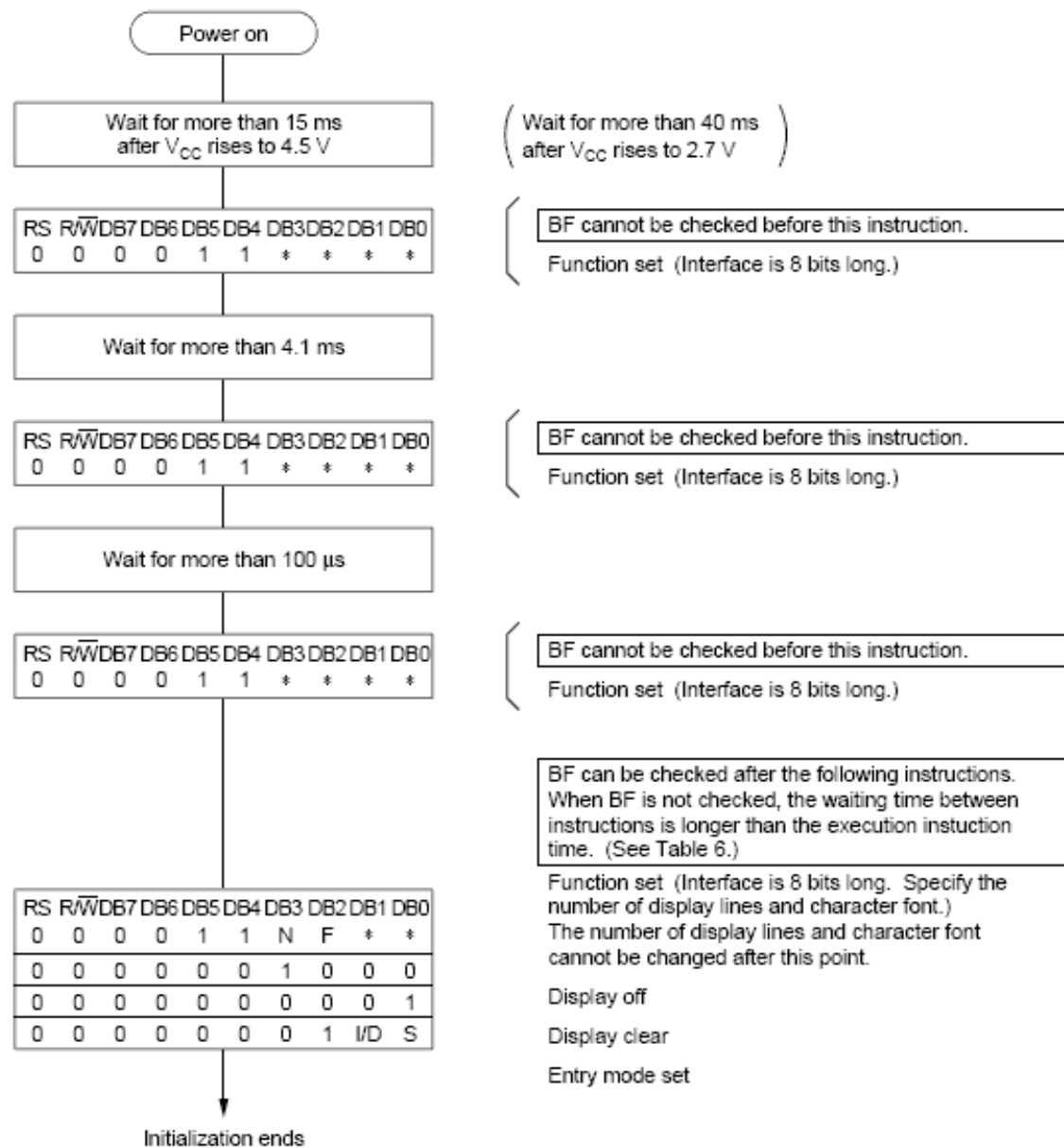
So if you want to display the text at specific positions of LCD, we require to manipulate address and then to set cursor position accordingly.

Character Generator RAM (CGRAM)-User defined character RAM

In the character generator RAM, we can define our own character patterns by program. CG RAM is 64 bytes, allowing for eight 5*8 pixel, character patterns to be defined. However how to define this and use it is out of scope of this tutorial. So I will not talk any more about CGRAM

6.4 LCD Initialization

Initializing LCD with instructions is really simple. Given below is a flowchart that describes the step to follow, to initialize the LCD.



6.5 LCD Command and Instruction Set

Command	Code										Description	Execution Time
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears the display and returns the cursor to the home position (address 0).	82μs~1.64ms
Return Home	0	0	0	0	0	0	0	0	1	*	Returns the cursor to the home position (address 0). Also returns a shifted display to the home position. DD RAM contents remain unchanged.	40μs~1.64ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets the cursor move direction and enables/disables the display.	40μs
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Turns the display ON/OFF (D), or the cursor ON/OFF (C), and blink of the character at the cursor position (B).	40μs
Cursor & Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves the cursor and shifts the display without changing the DD RAM contents.	40μs
Function Set	0	0	0	0	1	DL	N\$	F	*	#	Sets the data width (DL), the number of lines in the display (L), and the character font (F).	40μs
Set CG RAM Address	0	0	0	1	A _{CG}						Sets the CG RAM address. CG RAM data can be read or altered after making this setting.	40μs
Set DD RAM Address	0	0	1	A _{DD}						Sets the DD RAM address. Data may be written or read after making this setting.	40μs	
Read Busy Flag & Address	0	1	BF	AC						Reads the BUSY flag (BF) indicating that an internal operation is being performed and reads the address counter contents.	1μs	
Write Data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	46μs
Read Data from CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	46μs
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift. S/C = 0: cursor move S/C = 1: Display shift S/C = 0: cursor move R/L = 1: Shift to the right. R/L = 0: Shift to the left. DL = 1: 8 bits DL = 0: 4 bits N = 1: 2 lines N = 0: 1 line F = 1: 5x10 dots F = 0: 5 x 7 dots BF = 1: Busy BF = 0: Can accept data # Set to 1 on 24x4 modules \$ With KS0072 is Address Mode.										DD RAM: Display data RAM CG RAM: Character generator RAM A _{CG} : CG RAM Address A _{DD} : DD RAM Address Corresponds to cursor address. AC: Address counter Used for both DD and CG RAM address.	Execution times are typical. If transfers are timed by software and the busy flag is not used, add 10% to the above times.

Table: Command and Instruction set for LCD type HD44780

Although looking at the table you can make your own commands and test them. Below is a brief list of useful commands which are used frequently while working on the LCD.

No.	Instruction	Hex	Decimal
1	Function Set: 8-bit, 1 Line, 5x7 Dots	0x30	48
2	Function Set: 8-bit, 2 Line, 5x7 Dots	0x38	56
3	Function Set: 4-bit, 1 Line, 5x7 Dots	0x20	32
4	Function Set: 4-bit, 2 Line, 5x7 Dots	0x28	40
5	Entry Mode	0x06	6
6	Display off Cursor off (clearing display without clearing DDRAM content)	0x08	8
7	Display on Cursor on	0x0E	14
8	Display on Cursor off	0x0C	12
9	Display on Cursor blinking	0x0F	15
10	Shift entire display left	0x18	24
12	Shift entire display right	0x1C	30
13	Move cursor left by one character	0x10	16
14	Move cursor right by one character	0x14	20
15	Clear Display (also clear DDRAM content)	0x01	1
16	Set DDRAM address or cursor position on display	0x80+add*	128+add*
17	Set CGRAM address or set pointer to CGRAM location	0x40+add**	64+add**

Table: Frequently used commands and instructions for LCD

6.6 User Defined Character Generation

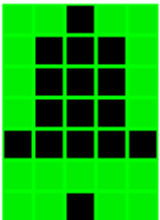
all character based LCD of type HD44780 has CGRAM area to create user defined patterns. For making custom patterns we need to write values to the CGRAM area defining which pixel to glow. These values are to be written in the CGRAM address starting from 0x40. If you are wondering why it starts from 0x40? Then the answer is given below.

Command	Code										Description	Execution Time
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Set CG RAM Address	0	0	0	1	A _{CG}						Sets the CG RAM address. CG RAM data can be read or altered after making this setting.	40μs

Bit 7 is 0 and Bit 6 is 1, due to which the CGRAM address command starts from 0x40, where the address of CGRAM (A_{CG}) starts from 0x00. CGRAM has a total of 64 Bytes. When you are using LCD as 5x8 dots in function set then you can define a total of 8 user defined patterns (1 Byte for each row and 8 rows for each pattern), where as when LCD is working in 5x10 dots, you can define 4 user defined patterns.

Lets take an of building a custom pattern. All we have to do is make a pixel-map of 7x5 and get the hex or decimal value or hex value for each row, bit value is 1 if pixel is glowing and bit value is 0 if pixel is off. The final 7 values are loaded to the CGRAM one by one. As i said there are 8 rows for each pattern, so last row is usually left blank (0x00) for the cursor. If you are not using cursor then you can make use of that 8th row also. so you get a bigger pattern.

To explain the above explanation in a better way. I am going to take an example. Lets make a "Bell" pattern as shown below.

Custom Pattern	Decimal	Hex
	Row 1: 4 Row 2: 14 Row 3: 14 Row 4: 14 Row 5: 31 Row 6: 0 Row 7: 4	0x04 0x0E 0x0E 0x0E 0x1F 0x00 0x04

Now we get the values for each row as shown.

Bit: 4 3 2 1 0 - Hex

Row1: 0 0 1 0 0 - 0x04

Row2: 0 1 1 1 0 - 0x0E

Row3: 0 1 1 1 0 - 0x0E

Row4: 0 1 1 1 0 - 0x0E

Row5: 1 1 1 1 1 - 0x1F

Row6: 0 0 0 0 0 - 0x00

Row7: 0 0 1 0 0 - 0x04

Row8: 0 0 0 0 0 - 0x00

We are not using row 8 as in our pattern it is not required. if you are using cursor then it is recommended not to use the 8th row. Now as we have got the values. We just need to put these values in the CGRAM. You can decided which place you want to store in. Following is the memory map for custom patterns in CGRAM.

Memory Map	
Pattern No.	CGRAM Address (Acg)
1	0x00 - 0x07
2	0x08 - 0x0F
3	0x10 - 0x17
4	0x18 - 0x1F
5	0x20 - 0x27
6	0x28 - 0x2F
7	0x30 - 0x37
8	0x38 - 0x3F

We can point the cursor to CGRAM address by sending command, which is 0x40 + CGRAM address.

Report

Write a header file for LCD control with options for Display Shift, Cursor Shift, Entry Mode Set, Clear Display, Cursor Home and New Character Generation options.