AT89C51 (8051) Microcontroller Wave Implementation

COMP 228 - Concordia University

Zafir Khalid - 40152164

Marwa Khalid - 40155098

Aditya Dua - 40073201

I. Introduction

A microcontroller is an integrated circuit containing the necessary elements of a computer system such as: CPU, memory, and input and output peripherals.[3] Using the 8051 microcontroller, different wave signals were generated to drive an autonomous robot. Ultimately the design of the function generator outputs three types of waves: Rectangular, Sawtooth and Triangular.

The type of function generated depends on the different frequencies inputted. A total of six inputs, four input bits for the frequency of the wave and two digital inputs for the type of wave. The four input bits are: P3.0 to P3.3 and the two digital inputs are P3.4 and P3.5. Note: the frequency range of the wave is 1 kHz to 16 KHz.

The type of wave is determined as follows:

Total of four possibilities:

(1) P3.4 = 0 and P3.5 = 0 resulting in 0 signal
(2) P3.4 = 0 and P3.5 = 1 resulting in Rectangular wave
(3) P3.4 = 1 and P3.5 = 0 resulting in Sawtooth wave
(4) P3.4 = 1 and P3.5 = 1 resulting in Triangular wave

In this study, we aimed to design and implement an assembly code that generates an output function based on the user selection. We will discuss the overall design and assumptions in our implementations along with the code and sample inputs and outputs.

II. Assumptions and parameters

A. *ASSUMPTIONS*

For this implementation, an external crystal is used to act as the main source of the timer. As such, the XTAL oscillator frequency (crystal frequency) is assumed to be 11.0592MHz. This is a common frequency used in 8051 microcontrollers as the timer clock frequency is 1/12th the frequency of the external crystal. As it is the only 1/12th, we have a timer clock frequency of 921.6KHz. This gives a single count the time period of 1.085 microseconds.

Any frequency calculations that yield in decimal values are rounded to the nearest integer for simplicity's sake.

Furthermore, for the runtime calculations, it is assumed that all operations are 100% efficient. Meaning they do not contribute to any time delay other than that generated by the actual delay function.

It is also assumed that the user is trained and aware on how to enter signals into the input pins wherever required.

B. *PARAMETERS*

Throughout the design and implementation of the wave generator, the only input parameters required from the user are at pin P3. Bits 3.0 to 3.3 control the desired frequency in kilohertz, while bits 3.4 and 3.5 specify the type of wave to be generated.

III. Design

A single .a51 assembly code file is used to control the operation of the program. Microchip's 8051 microcontroller was selected for this implementation.

- The microcontroller used houses the following characteristics: 8051-based Fullly Static 24MHz CMOS controller with 32 I/O lines, 2 Timers/Counters, 6 Interrupts/2 Priority Levels, UART, Three level Program Memory Lock, 4K Bytes Flash Memory and 128 Bytes On-chip RAM storage.
- TMOD timer used in mode 2. This 8 bit timer is set to auto reload mode. TF1 stores the overflow, indicating the delay has elapsed. TF1 bit is constantly monitored to count the delay. Once the TH1 8 bit is loaded from the accumulator the timer starts. TL1 is automatically updated and begins counting until an overflow is thrown in TF1. At which point the timer ends.
- A delay look up table stores the count values required to generate frequencies in the range of 1KHz to 16KHz. The calculations for these values are shown in the following section
- A DPTR pointer is used to access the values stored inside the look up table by reading the input from P3.

A. EQUATIONS

Other than the simple arithmetic used to generate the waves, to fill up the delay look up table a more complex equation was required. The value stored in this look up table is precisely calculated to generate the right amount of delay, such that they approximately match the input frequency given in P3 by the user. Note: This formula is based on the assumption made earlier. (ie: time period of a single count is 1.085 microseconds)

Count = Delay in microseconds/machine cycle

Count = Delay in microseconds/1.085 microseconds

Value to add = (255-Count)

Since the max count in mode 2 (8-bit timer) is 255, the value to be loaded into the lookup table is 255 less the count. For example when calculating the value for 16KHz:

Count = (62.5/2)/1.085

Count ≈ 28

Value to add to look up table = 255 - 28 = 227

Value to add to look up table = $E3_{16}$

Using this process each value from 1KHz to 16KHz has a corresponding value in the lookup table.

Below is a table showing each frequency with its corresponding time period and value in the delay table.

TABLE I.

Frequency (KHz)	***Time Period (μs)***	***Value***
1	1000	FF33
2	500	19
3	333.3	66
4	250	8C
5	200	A3
6	166.6	B3
7	142.9	BE
8	125	C6
9	111	CC
10	100	D2
11	90.9	D6
12	83.3	D9
13	76.9	DC
14	71.4	DF
15	66.6	E1
16	62.5	E3

IV. Wave design

The 3 types of waves to be generated all use slightly different logic from one another. Below, each of the three wave designs will be outlined.

A. Rectangular Wave

The rectangular wave jumps between 0 and 255 repeatedly based on a time interval given by the corresponding input frequency. Register 1 (R1) holds the values 255 and 0 before passing them to P1 (The output pin). For each cycle, the delay method is called twice. Once at 255 and once at 0. This evenly generates the frequency required.

B. Sawtooth Wave

The sawtooth wave makes use of nested loops. Where the outer loop is responsible to keep the wave generating for as long as required - and the inner loop creates its distinct sloping edges. For each cycle of the inner loop, the value stored in P1 (and the accumulator) is decreased by 1 until it reaches 0.

Once at zero, a jump is made back to the outer loop which repeats the process indefinitely.

C. Triangular Wave

The triangular wave is made up of two parts. The upward slope followed by the downward slope. To generate the upward sloping portion a loop is used uptil 255. At this max value the loop for the downward slope is used. Decrementing each time until 0. Encompassing both the increment and decrement loops is another loop which is responsible for keeping the wave generation running indefinitely.

V. Control flow design

As specified earlier the type of wave is specified by the bits at P3.4 and P3.5. To send the flow of control into the right type of wave, a series of JB and JNB operands are used. These operands when arranged together in the manner shown (in the code below) give the result required for the wave selection.

In short, first P3.4 is checked to see whether it is 1 or 0. Following this, the second bit (P3.5) is checked and the cases are handled accordingly. The JB operand jumps when the specified bit is 1. Whereas the JNB operand jumps when the specified bit is 0. The operand jumps to one of the four code bodies: rectangular, sawtooth, triangular or logic zero depending on the value of P3.4 and P3.5.

VI. Code

```
ORG 00                                  ; Instruction for Assembler: Start Code in Memory
MOV P3, #16d                            ; Moves 16d to P3 (USER INPUT)
MOV A, P3                               ; Saves P3 in accumulator
MOV DPTR, #DelayLookup                  ; Set DPTR to DelayLookup
MOVC A, @A+DPTR                         ; Indexes to corresponding frequency entered by user

CLR P3.4                                ; Clear P3.4
CLR P3.5                                ; Clear P3.5

SETB P3.4                               ; Set P3.4 to 1 (USER INPUT)

JNB P3.4, CHECKOTHERFALSE               ; Jump to CHECKOTHERFALSE if P3.4 = 0
JB P3.4, CHECKOTHERTRUE                 ; Jump to CHECKOTHERTRUE if P3.4 = 1

CHECKOTHERFALSE:
        JB P3.5, RECTANGULAR            ; Jump to RECTANGULAR if P3.5 = 1
        JNB P3.5, LOGICZERO             ; Jump to LOGICZERO if P3.5 = 0

CHECKOTHERTRUE:
        JB P3.5, TRIANGULAR             ; Jump to TRIANGULAR if P3.5 = 1
        JNB P3.5, SAWTOOTH              ; Jump to SAWTOOTH if P3.5 = 0

RECTANGULAR:
        CONTINUE:
                MOV R1, #0FFH           ; Moves 0FFH to R1
                MOV P1, R1              ; Saves R1 in P1
                ACALL DELAY             ; Calling Delay
                MOV R1, #00H            ; Move 00H to R1
                MOV P1, R1              ; Saves R1 in P1
                ACALL DELAY             ; Calling DELAY
                SJMP CONTINUE           ; Jump to CONTINUE

SAWTOOTH:
        OUTERLOOP:
                ACALL DELAY             ; Calling DELAY
                MOV A, #0FFH            ; Moves 0FFH to A
                INNERLOOP:
                        MOV P1, A                   ; Saves A in P1
                        DEC A                       ; Decrementing A by 1
                        CJNE A, #00H, INNERLOOP     ; Jump to INNERLOOP if A != 00H
        SJMP OUTERLOOP                  ; Jump to OUTERLOOP

TRIANGULAR:
        MOV A, #00H                     ; Moves 00H to A
        REPEAT:
                INCREMENT:
                        MOV P1, A                   ; Saves A in P1
                        INC A                       ; Incrementing A by 1
                        CJNE A, #0FFH, INCREMENT    ; Jump to INCREMENT if A != 0FFH
                        ACALL DELAY                 ; Calling DELAY
                DECREMENT:
                        MOV P1, A                   ; Saves A in P1
                        DEC A                       ; Decrementing A by 1
                        CJNE A, #00H, DECREMENT     ; Jump to DECREMENT if A != 00H
                        ACALL DELAY                     ;Calling DELAY
        SJMP REPEAT                                 ; Jump to REPEAT

LOGICZERO:
        MOV A, #00H                                 ; Move 00H to A
        MOV P1, A                                   ; Saves A in P1
        SJMP LOGICZERO                              ; Jump to LOGICZERO

DELAY:
        MOV TH1, A                                  ; Timer is Initialized with accumulator value
        MOV TMOD, #00100000b                        ; Timer 1 is set for mode 2
        SETB TR1                                    ; Start Timer 1

LOOP:
        JNB TF1, LOOP                               ; Jump to LOOP if TF1 is 0
        CLR TF1                                     ; Clear Overflow Flag

;Stores all the hexadecimal equivalent delays for various frequencies in the range of 1KHz to 16KHz
DelayLookup:
DB 0, 0xFF31, 0x19, 0x66, 0x8C, 0xA3, 0xB3, 0xBE, 0xC6, 0xCC, 0xD2, 0xD6, 0xD9, 0xDC, 0xDF,
0xE1, 0xE3

END
```

VII. Sample input and output

INPUT:

```
SETB P3.4
```

OUTPUT:

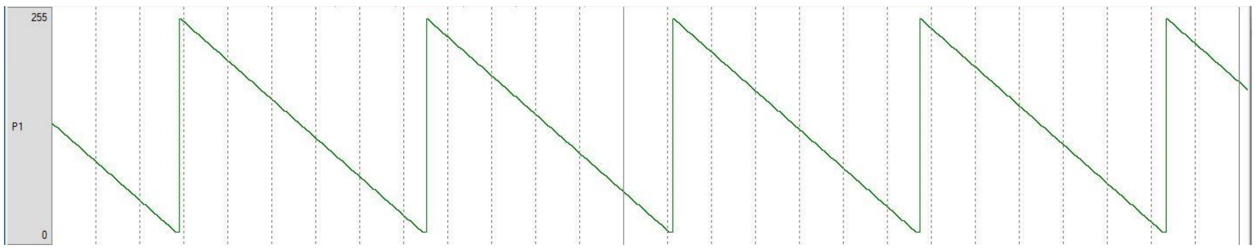

FIG 2.0

INPUT:

```
SETB P3.5
```

OUTPUT:

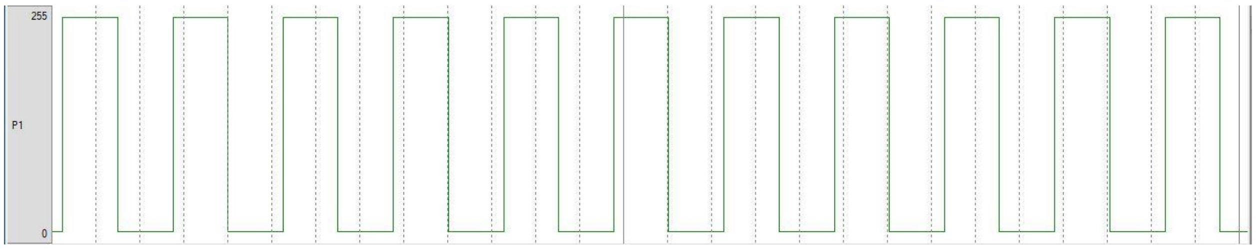

FIG 2.1

INPUT:

```
SETB P3.4
SETB P3.5
```

OUTPUT:

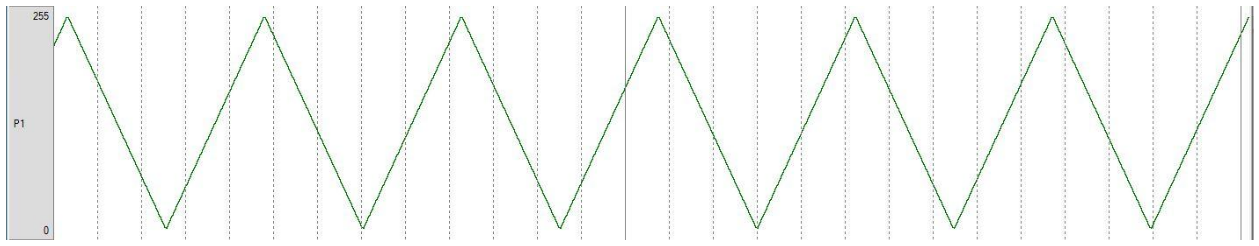

FIG 2.3

VIII. Conclusion

The 8051 microcontroller is taking input from the user at pin P3. The bits from 3.0 to 3.3 are controlling the desired frequency that is required in kilohertz, whereas bits 3.4 and 3.5 are responsible for the type of wave generated. It is being used to drive an autonomous robot by generating three types of waves:

1) Rectangular waves → They were generated when only the 3.5 bit is being used.
2) Sawtooth → They were generated when only the 3.4 bit is being used.
3) Triangular → They were generated when both the 3.4 and 3.5 bits are being used.

The single .a51 assembly code is being used over here to control the operation from the microchip(8051 based CMOS or NMOS Microcontroller with 32 I/O lines, 2 Timers/Counters, 5 Interrupts/2 Priority Levels, 4K Bytes ROM, 128 Bytes on-chip RAM), which in turn controls the autonomous robot.

From the assumptions, there was a 100% efficiency to be assumed other than the actual delay function. Which was calculated for each value respectively.