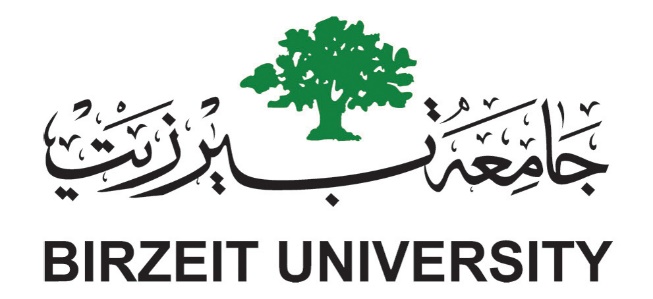
[](http://www.google.ps/url?sa=i&source=images&cd=&cad=rja&docid=TNC2o42g4wGV0M&tbnid=n9SnL2A2yBmR1M:&ved=0CAgQjRwwAA&url=http://sites.birzeit.edu/comp/ArabicOntology/news-events/siera-kick-off-conference-in-the-news/&ei=3wMlUZ6uGPKK4gT864CIAw&psig=AFQjCNG7NukYdzV3_HLvAhGHpdxIJPGq1Q&ust=1361466719460852)

**Department of Computer System Engineering**

**COMPUTER DESIGN LAB**

**ENCS 411**

**Report 3**

**Experiment No. 8**

***Programmable Interval Timer (PIT)***

**Student Name:**

**Ibrahim Abu Farha 1110024**

**Othman Alkhamra 1110017**

**Instructor: Dr. Ahmad Afaneh**

**Teaching Assistant: Eng. Yazan Abu Farha**

**Section: 3**

**Date: 20/05/2014**

# Abstract:

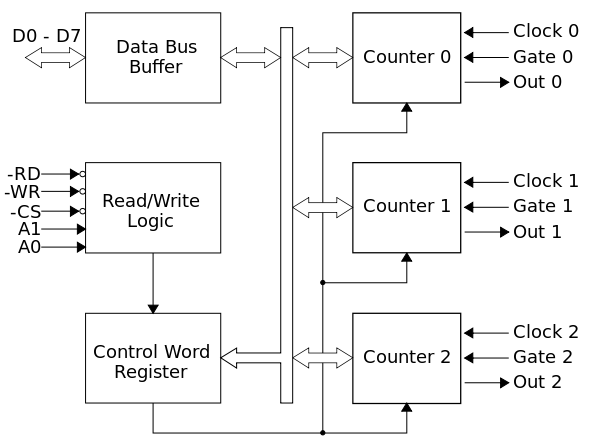
This experiment is all about the Programmable Interval Timer (PIT) 8253/4,   
how to configure it, test it on both the MDA 8086 kit and Personal Computers.

# **Introduction:**

## Programmable Interval Timer (PIT):

The [Intel](http://en.wikipedia.org/wiki/Intel) 8253 and 8254 are [Programmable Interval Timers](http://en.wikipedia.org/wiki/Programmable_Interval_Timer) (PITs), which perform timing and counting functions. In other words, it is counters which generate signal depending on programmable count or using trigger input**.** [1][2]

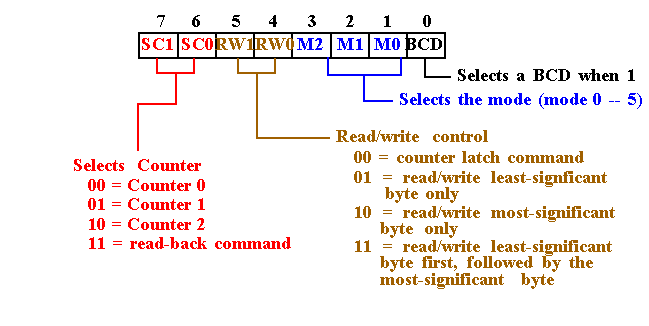
## The Internal Structure of the Programmable Interval Timer:

[](http://upload.wikimedia.org/wikipedia/commons/b/ba/Intel_8253_block_diagram.svg)

**Fig.1.1: 8254 PIT Internal Structure [1]**

As shown in **Fig.1.1**, the programmable interval timer has three counters, called channels. Each channel can be programmed to operate in one of six modes. Once programmed, the channels can perform their tasks independently. All channels are driven by a 1.19MHz oscillator signal. Each “tick” of the PIT generates hardware interrupt request 0. [1][3]

Furthermore, a PIT has a Control Word Register which contains the programmed information which will be sent -by the [microprocessor](http://en.wikipedia.org/wiki/Microprocessor)- to the device. It defines how the PIT logically works. Each access to these ports takes about 1 µs. The control word register contains 8 bits, labeled D7..D0 (D7 is the [MSB](http://en.wikipedia.org/wiki/Most_significant_bit)) see Fig.1.2. [1]



**Fig.1.2: Control Word Register Description [6]**

In Fig.1.1, a Read/Write Logic also appears which has 5 pins, which are listed below:

* : read signal
* : write signal
* : chip select signal
* A0, A1: address lines

Moreover, the Data Bus Buffer block contains the logic to buffer the data bus to / from the microprocessor, and to the internal registers. It has 8 input pins, usually labelled as D7…D0, where D7 is the [MSB](http://en.wikipedia.org/wiki/Most_significant_bit). [1]

## Programmable Interval Timer Modes:

As mentioned before, a PIT has six different operation modes. PITs may be one-shot or periodic.

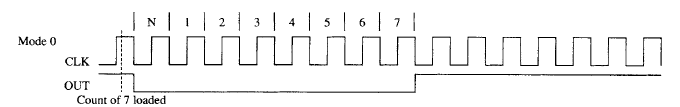
* One-shot timers will signal only once and then stop counting. Periodic timers signal every time they reach a specific value and then restart, thus producing a signal at periodic intervals.
* Periodic timers are typically used to invoke activities that must be performed at regular intervals.

Counters are usually programmed with fixed intervals that determine how long the counter will count before it signals. The interval determines how long the counter will count before it will output a signal. [2]

The six modes of Programming Interval Timer are:

### Mode 0: Interrupt on Terminal Count:

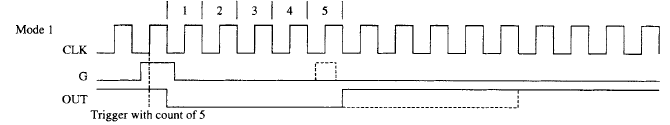
Allows the 8254 counter to be used as an events counter. In this mode, the output becomes a logic 0 when the control word is written and remains there until N plus the number of programmed counts. For example, if a count of 5 is programmed, the output will remain a logic 0 for 6 counts beginning with N. Note that the gate (G) input must be a logic 1 to allow the counter to count. If G becomes a logic 0 in the middle of the count, the counter will stop until G again becomes a logic 1 see **Fig.1.3.** [4]



**Fig.1.3: Mode 0 [4]**

### Mode 1: Hardware Re-triggerable One-Shot:

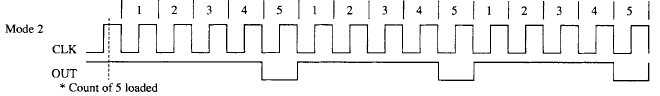
Causes the counter to function as a retriggerable, monostable multivibrator (one-shot). In this mode the G input triggers the counter so that it develops a pulse at the OUT connection that becomes a logic 0 for the duration of the count. If the count is 10, then the OUT connection goes low for 10 clocking periods when triggered. If the G input occurs within the duration of the output pulse, the counter is again reloaded with the count and the OUT connection continues for the total length of the count **see Fig.1.4**.[4]



**Fig.1.4: Mode 1 [4]**

### Mode 2: Rate Generator:

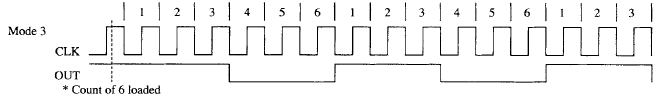
Allows the counter to generate a series of continuous pulses that are one clock pulse wide. The separation between pulses is determined by the count. For example, for a count of 10, the output is a logic 1 for nine clock periods and low for one clock period. This cycle is repeated until the counter is programmed with a new count or until the G pin is placed at a logic 0 level. The G input must be a logic 1 for this mode to generate a continuous series of pulses **see Fig.1.5**. [4]



**Fig.1.5: Mode 2 [4]**

### Mode 3: Square Wave Generator:

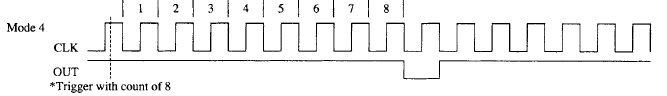
Generates a continuous square wave at the OUT connection, provided that the G pin is a logic 1. If the count is even, the output is high for one half of the count and low for one half of the count. If the count is odd, the output is high for one clocking period longer than it is low. For example, if the counter is programmed for a count of 5, the output is high for three clocks and low for two clocks **see Fig.1.6**. [4]



**Fig.1.6: Mode 3 [4]**

### Mode 4: Software Triggered Strobe

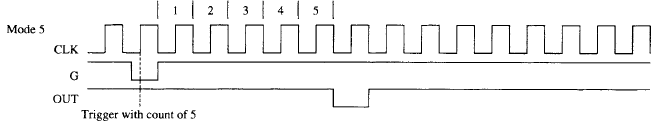
Allows the counter to produce a single pulse at the output. If the count is programmed as a 10, the output is high for 10 clocking periods and low for one clocking period. The cycle does not begin until the counter is loaded with its complete count. This mode operates as a software triggered oneshot. As with modes 2 and 3, this mode also uses the G input to enable the counter. The G input must be a logic 1 for the counter to operate for these three modes **see Fig.1.7**. [4]



**Fig.1.7: Mode 4 [4]**

### Mode 5: Hardware Re-triggerable Strobe

A hardware triggered one-shot that functions as mode 4, except that it is started by a trigger pulse on the G pin instead of by software. This mode is also similar to mode 1 because it is retriggerable **see Fig.1.8**. [4]



**Fig.1.8: Mode 5 [4]**

# **Procedure and Discussion:**

## 2.1 Task 1: Configuring PIT on MDA-8086 Kit

In this task, we would like to write a program that is used to control the LEDs on the kit using the PIT as a timer, this program will change the lightning LED after certain duration controlled by the PIT. Here is the code of the program.

CODE SEGMENT   
ASSUME CS: CODE, DS: CODE, ES: CODE, SS: CODE   
PPIC\_C EQU 1FH   
PPIC EQU 1DH   
PPIB EQU 1BH   
PPIA EQU 19H   
CTC1 EQU 0BH ; counter 1  
CTCC EQU 0FH ; control register ;   
INTA EQU 10H ; address of the PIC  
INTA2 EQU INTA+2 ; PIC address   
INT\_V EQU 40H\*4   
  
ORG 1000H   
 XOR BX, BX  
 MOV ES, Bx  
   
 MOV AX, OFFSET INT\_SER   
 MOV BX, INT\_V   
 MOV WORD PTR ES: [BX], AX   
 XOR AX, AX   
 MOV WORD PTR ES: [BX+2], AX   
 ; The segment is zero here   
 CALL INIT   
 CALL P\_INIT   
; 8255 Initialization   
 MOV AL, 10000000B   
 OUT PPIC\_C, AL   
 MOV AL, 11111111B   
 OUT PPIA, AL   
 MOV AL, 00000000B   
 OUT PPIC, AL   
 MOV AH, 11110001B  
 MOV AL, AH   
 OUT PPIB, AL   
 STI   
L2: NOP   
 JMP L2   
 INT 3   
 ; The Interrupt Service Routine   
INT\_SER:   
 SHL AH, 1   
 TEST AH, 00010000B   
 JNZ L1   
 OR AH, 11110000B   
 JMP L3   
 ; LED out   
L1: MOV AH, 11110001B   
L3: MOV AL, AH   
 OUT PPIB, AL   
 PUSH AX   
 MOV AX, 0ffFFH   
 OUT CTC1, AL   
 MOV AL, AH   
 OUT CTC1, AL   
 POP AX   
 ; EOI command   
 MOV AL, 00100000B   
 OUT INTA, AL   
 STI   
 IRET   
 ; 8253 Initialization   
P\_INIT PROC NEAR   
 PUSH AX   
 MOV AL,01110000B   
 OUT CTCC, AL   
 MOV AX, 0fFFFH   
 OUT CTC1, AL   
 MOV AL, AH   
 OUT CTC1,AL   
 POP AX   
 RET   
P\_INIT ENDP   
 ; 8259 Initialization   
INIT PROC NEAR   
 ; ICW1   
 MOV AL, 00010011B   
 OUT INTA, AL   
 ; ICW2 interrupt vector   
 MOV AL, 40H; the address of IR0  
 OUT INTA2, AL   
 ; ICW4   
 MOV AL, 00000001B;   
 OUT INTA2, AL   
 ; interrupt mask   
 MOV AL, 11111110B; mask all bits except IR)  
 OUT INTA2, AL   
 RET   
INIT ENDP   
CODE ENDS   
 END

In the program above, we use the PIT in order to control the LEDs. In this task we use Counter 1 in the PIT in order to control the LEDs and when the light would be moved. We use the aforementioned counter in mode 0, in which the output goes low for a duration that is the count+1 and then it goes high and also causes and interrupt. At first we send the control word to the PIT in order to program it in the previous configuration the control word is (01110000B), which means that counter 1 will work in mode 0 and it will receive the count as LSB followed by MSB and it will count in binary. In this task, the control of the movement of the LEDs is controlled through the interrupt caused by the counter. The main code is an infinite loop and when the counter causes an interrupt it will be handled by the ISR, here we use INT 40H so we install the interrupt in the interrupt vector table and load the values of the address of the ISR. When this interrupt happens the ISR sends a new value to the PPI port (port A) that is used to interface the LEDs and it also reloads the count to the counter .Since we use hardware interrupts we need to use the PIC in order to interface them in a proper way, so we program it as needed and the value of interrupt vector that we use here is INT 40H, because we have installed this interrupt in the interrupt vector table and the output of the counter is connected to IR0.  
Since the counter is in mode 0, the gate must be kept high in order to let it count. So if we press the gate button this will connect the gate pin to the ground and the counter will stop its operation till it goes high again and this will stop the movement of the lightning LED. In the previous program the value of the count we load to the counter is 0FFFFH which is 65535 in decimal and since the counter in mode 0 the output will be low for 65536 (65535+1) clock cycles. The input clock to the counter is 2.5MHz so the delay of the counter will be (65533\*1/(2.5MHz)) which equals 26 ms.

## 2.2 Task 2: Configuring PIT on MDA-8086 Kit (with more delay)

This task is the same as the previous one, but here we want more delay between each movement of the lightning LED. In the previous task we have used the maximum delay we could have from the counter (counter 1) and we used (0FFFFH) as the value of the count which produces a 26 ms delay. Since the count (divisor) register is 16-bit it is impossible to get more delay from that counter. In order to achieve more delay we need to do some changes in the circuit and use two counters (counter 1 and counter 2). We have connected the output of the counter one as an input clock to the counter two; we also have connected the output of counter 2 to IR1 in the PIC. Due to the previous modifications we have modified the code of the programs, since counter 1 will act as the clock generator for counter 2, counter 1 must be programmed to work in mode 3 which produces a clock, counter 2 is programmed in mode 0.here is the programming of the counters.

MOV AL,01110110B   
 OUT CTCC, AL   
 MOV AX, 0FFFFH   
 OUT CTC1, AL   
 MOV AL, AH   
 OUT CTC1,AL   
MOV AL,10110110b  
OUT CTCC, AL   
MOV AX, 5  
OUT 0DH,AL  
 MOV AL, AH   
OUT 0DH,AL

In the previous piece of code counter 1 will produce a clock with a clock cycle duration of 26 ms. Since counter 2 is programmed in mode 0 it will wait 6 clock cycles to change its output value from low to high, so the total delay will be (6\*26ms) which equals 156 ms. Moreover, because the output of the counter 2 is connected to IR1 in the PIC we changed the interrupt used from INT 40H to INT 41H. In addition to that, both gate 1 and gate 2 must be kept high and if we press any of them this will cause the gate pin of the counter to be connected to the ground and the counter will stop its operations till it goes back.

# **Conclusion:**

In this experiment, we have used many 8086 peripherals like the PIT, PIC and the PPI. We have used all of them at the same time in order to get our program running as we wanted. The concentration here was on how to use the Programmable Interval Timer (PIT), we learned its different modes of operation, how to program it and use it in many functionalities such as generation of waveforms. Furthermore, we have used the dot matrix as an extra task and it worked successfully.

# **References:**

[1] <http://en.wikipedia.org/wiki/Intel_8253>

**Access Date: 15 – 05 – 2014**

[2] <http://en.wikipedia.org/wiki/Programmable_interval_timer>

**Access Date: 15 – 05 – 2014**

[3] Programmable Interval Timer MDA-8086 Kit – PPI Application, Experiment 8.

**Access Date: 15 – 05 – 2014**

[4] Intel Microprocessor Architecture, Programming and Interfacing 8th Edition, Barry B. Brey

**Access Date: 15 – 05 – 2014**

[5] <http://wiki.osdev.org/Programmable_Interval_Timer>

**Access Date: 16 – 05 – 2014**