Improved PIC16f877 microcontroller timer register to delimit the execution time of an operation

Souleymane MBATHIE[1*], OUESSE Mohamed E. A[1], Baboucar DIATTA[1], Adrien BASSE[1], Ridha BOUALLEGUE[2]

[1]Alioune Diop University of Bambey, Dept. of Information and Communication, Bambey, Senegal

[2]Université de Carthage, Sup'Com, Innov'Com Lab, Tunis, Tunisie

Abstract

We often plan and build applications that carry out various processes requiring precise time management. Time management is attributed to the microcontroller, which can use various techniques such as instruction execution time, which requires delicate control of loops, or the use of functions such as delay, which cause your program to lose time in order to create delays. The problem with this kind of time management is that the time lost is of unknown duration. To alleviate this problem, new microcontrollers almost always offer specialized time-counting and time-management circuits, known as timers. A timer is a kind of clock built into the microcontroller, enabling the duration of an event to be measured. Their use is not complicated, but requires a good understanding of the structure of the various timers on a microcontroller. In this paper, we've chosen to demonstrate the improved and management of timers on Microchip's PIC 16F877 microcontroller using assembly language.

Received on 30 December 2024; accepted on 24 March 2025; published on 14 November 2025

Keywords: PIC16f877 microcontroller, timer register

doi:10.4108/eetiot.10862

1. Introduction

All work, whether it's the design, realization or execution of a project task, requires a set amount of time. The time can be defined downstream or upstream of the project. For some projects, not respecting the timing of tasks may not affect the project's finality, but for other projects in fields such as IT, technology or automation, it may cause the project to become useless. In the new information and communication technologies, time has become an increasingly important factor; more than just counting the time it takes to execute a task, it can offer other additional functionalities; for example, it can trigger alarms when a certain threshold is reached. It can also execute tasks periodically with generated interrupts. To carry out the steps we've listed, the system needs to use a timer, which must be very precise in order to take the necessary action. These timers can be used for a variety of purposes. To date, we have used the delay procedure to apply a delay in the program, which would register up to a specific value, before the activity could continue. The delay procedure had two disadvantages: we couldn't tell exactly how long the delay procedure was in progress; we couldn't perform other steps while the program was running the delay procedure. Now, using timers, we can build a very precise delay that will be based on the system clock and allow us to achieve our desired delay well known in advance. To know how to work with these timers, we need to learn a few things about each of them.

2. Presentation the Timer register

The PIC16F877 is a 40-pin integrated circuit based on RISC architecture with an 8-bit CMOS microprocessor and is part of the Mid-Range family [1]. Mid-Range PICs have a set of 35 instructions and

*Corresponding author. Email: julembathie@gmail.com

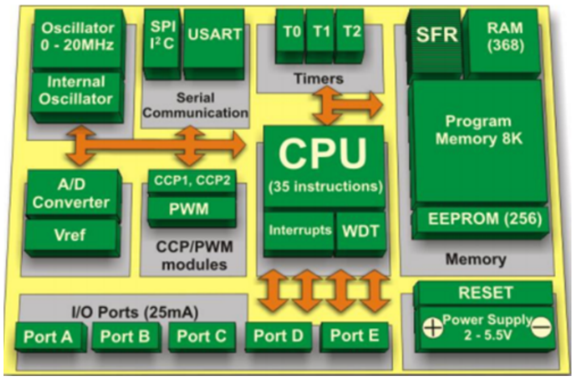


Figure 1. Architecture Pic16F877 [7]

each instruction is stored in a program word [2]. In addition to the microprocessor, the PIC16F877 microcontroller has other integrated elements such as an 8KB 14-bit programmable flash memory, a 256-byte EEPROM, a 512-byte data memory, three timer registers, 14 interrupt sources and five I/O ports [3]. In addition to these elements, the PIC16F877 also has 2 integrated CCP (capture/comparison/PWM) modules, an 8-channel 10-bit ADC and an I2C, SPI and USART serial interface [4], [5] and [6]. It offers a number of advantages, including implementation flexibility, low cost, low power consumption, high speed, small size and highly functional solutions.

The Pic16F877 microcontroller sets itself apart from other microcontrollers by incorporating timer registers into its architecture.
Timer registers are electronic counters capable of counting the time that elapses very precisely. In fact, they are registers inside a microcontroller that increment each time it receives a clock signal pulse generated internally by the microcontroller. The PIC 16F877 microcontroller has three Timer counters:

- Timer 0: from 0 to 255 (as it is coded on 8 bits);
- Timer 1: from 0 to 65535 (because it is coded on 16 bits);
- Timer 2: 0 to 255 (because it's coded on 8 bits.

Counters progress progressively, and once they reach their peak, they instantly return to zero. However, before counting starts again, it will transmit a signal. This signal is known as a flag. Capturing this signal is crucial, as it will enable us to determine the time needed to measure the duration of an operation. Each timer stores its value in its own register.

3. Experimentation

The experimental scheme is represented using Proteus software and consists of a pic16F877 microcontroller, traffic lights, a start button, resistors and two timers

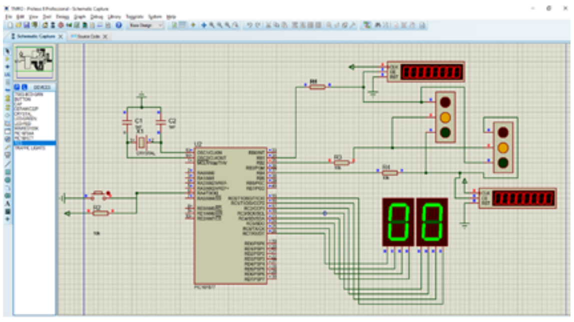


Figure 2. Device containing a pic microcontroller

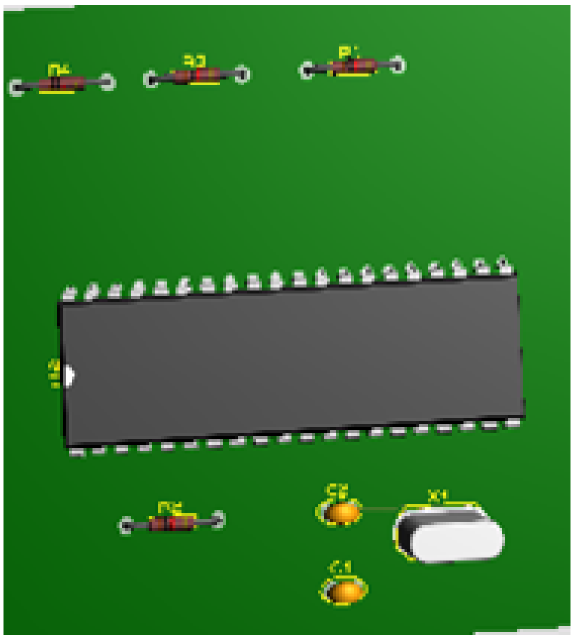


Figure 3. Device containing a pic microcontroller in 3D

to display the time that one of the timer registers will take before triggering flag or flag. The signal light LEDs indicate that the operation time can be counted down without interrupting or impacting the execution of operations. As soon as the device (circuit) is powered up, the lamp in the middle (yellow LED) lights up. Illumination of this lamp indicates that the device is ready to perform the operations assigned to it. If the user presses the start button, the button connected to RA4, the lamp that had been lit goes out, and the RB2 port receives a signal that will light the lamp connected to it. But also the timer register, which was at 0, starts to increment towards its final value.

At the same time, the timer is started. After the timer register has reached its final value and returned to 0, it triggers an interrupt, so the first timer should stop counting and display the time the TMR0 register has taken to reach its final value. However, we've added a second timer to check the accuracy of the time. The second timer is started when the button is unchecked

Table 1

RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
Bit7	-	-	-	-	-	-	Bit0

Table 2

RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
0	0	0	0	0	1	1	1

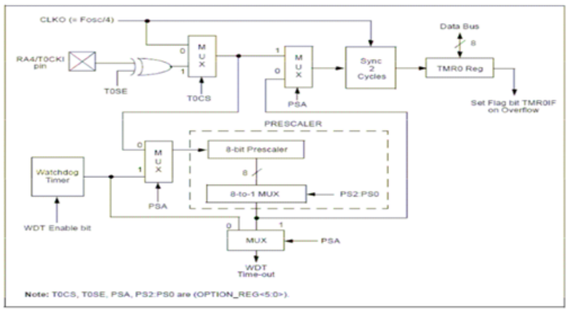
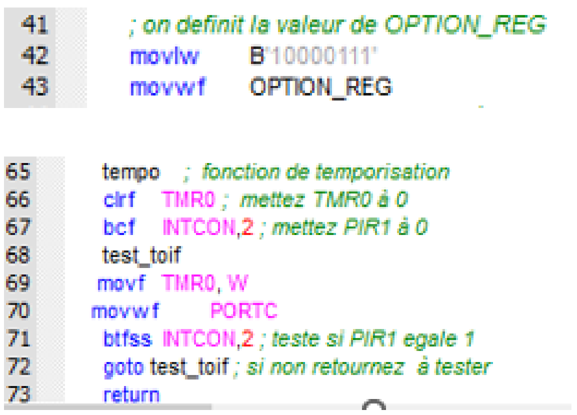


Figure 4. Timer0 schemes [9]



```
     ; on definit la valeur de OPTION_REG
     movlw      B'10000111'
     movwf      OPTION_REG

   tempo    ;  fonction de temporisation
   clrf   TMR0 ;  mettez TMR0 à 0
   bcf    INTCON,2 ; mettez PIR1 à 0
   test_toif
  movf  TMR0, W
 movwf       PORTC
   btfss  INTCON,2 ; teste si PIR1 egale 1
   goto test_toif ; si non retournez  à tester
   return
```

Figure 5. Timer0 code

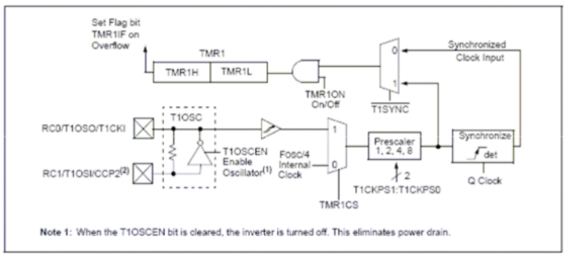


Figure 6. Timer1 schematics [10]

and will stop when the lamp attached to port RB4 is switched off and the timers register has completed its run to 0. This device is designed to measure the time taken for each timer register to reach its final value. To use timers, you need to have a clear understanding of the microcontroller and a clear understanding of how the Timer0, Timer1 and Timer2 timers inside the microcontroller work. In this article we can't go into detail about all the timers, but for the sake of clarity we'll show how to use the Pic16F77's three timer registers using assembly language.

3.1. Timer 0

It has a register called TMR0, which has a size of 8 bits, accessible in bank 0 [8]. At start-up, register TMR0 is initialized to 0 and its value will increment each time it receives a pulse. It can be read at any time and set to a value not exceeding 255 [1] and [3]. Two registers control the operation of this timer: OPTION REG and INTCON. The 8-bit OPTION REG register, accessible in bank1, is used to configure timer 0. These bits are distributed as follows (table 1). Timer 0 has two modes: timer mode and counter mode.

The choice of modes is made via the TOCS bit. To configure timer0 in timer mode, the TOCS bit must be set to 0. In this case, the frequency used by the Pic16f877 is the microcontroller's internal clock frequency divided by 4. The 4 used for division is the prescaler. Each time timer mode is selected, the prescaler is equal to 4.

To reduce the speed of the frequency pulses leaving the prescaler, we activate the prescaler. This is activated by setting the PSA bit of the OPTION_REG register to 0; as soon as the bit is equal to 0, the division ratio is set from 2 to 256 according to the PS2, PS1 and PS0 bits of the same register. In this article, we've used the maximum prescaler because we need the maximum time. In the end, our register will have the following bits (table 2):

To reduce the speed of the frequency pulses leaving the prescaler, we activate the prescaler. This is activated by setting the PSA bit of the OPTION_REG register to 0; as soon as the bit is equal to 0, the division ratio is set from 2 to 256 according to the PS2, PS1 and PS0 bits of the same register. In this article, we've used the maximum prescaler because we need the maximum time. In the end, our register will have the following bits:

To delimit the time TMR0 will take to reach its final value, we need the IT1CON register. In the IT1CON register, when the TMR0 register changes from FFh to 00h, the TMR0IF bit in the IT1CON register changes to 1. We'll scan this bit to authorize a program interrupt.

3.2. Timer 1

Timer1 has a 16-bit register called TMR1[8].This register is split into two 8-bit registers called TMR1H and TMR1L. It increments from 0000 to the maximum value of FFFF in hexadecimal.

Table 3

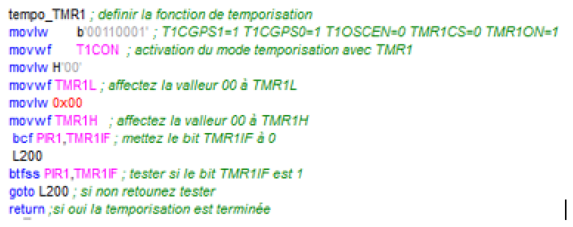
-	-	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
Bit7	-	-	-	-	-	-	Bit0

Table 4

T1CKPS0	T1CKPS0	Predivisor
0	0	1
0	1	2
1	0	4
1	1	8

Table 5

-	-	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
0	0	1	1	0	0	0	1



```
tempo_TMR1 ; definir la fonction de temporisation
movlw      b'00110001' ; T1CGPS1=1 T1CGPS0=1 T1OSCEN=0 TMR1CS=0 TMR1ON=1
movwf      T1CON  ; activation du mode temporisation avec TMR1
movlw H'00'
movwf TMR1L ; affectez la valleur 00 à TMR1L
movlw 0x00
movwf TMR1H   ; affectez la valleur 00 à TMR1H
 bcf PIR1,TMR1IF ; mettez le bit TMR1IF à 0
 L200
btfss PIR1,TMR1IF ; tester si le bit TMR1IF est 1
goto L200 ; si non retounez tester
return ;si oui la temporisation est terminée
```

Figure 7. Timer1 code

For timer1, the configuration is done from the T1CON register, accessible in bank 0 (table 3).

The TMR1ON bit must be set to 1, which activates the timer mode and gives us a prescaler equal to 4.

The T1CKPS0 and T1CKPS1 bits are used for the prescaler (table 4).

The remaining bits are set to 0.
The bit distribution of the register with the maximum pre-divider is as follows (table 5).

The code used to assign bits to the T1CON register and operate timer1 is as follows.

If TMR1 reaches its maximum value, a signal is generated when capacity is exceeded. This signal causes the TMR1IF interrupt flag bit in the PIR1 register to change, marking the end of counting.

3.3. Timer 2

For Timer2 we have 2 count registers: TMR2 and PR2, each consisting of 8 bits. The TMR2 register is used to store the initial count value. The PR2 register is used to store the maximum value we need to reach [11]. Unlike Timer0 and Timer1, which are used with prescaler and prescaler, Timer2 is used with a prescaler and a postscaler.

The prescaler divides the frequency clock source before the count takes place at the TMR2 register, so the count inside the TMR2 register is performed on the basis of the frequency clock source divided by the prescaler. The TMR2 match output passes through a 4-bit postscaler to generate a TMR2 interrupt that will set the TMR2IF flag bit of the PIR1 register to 1 if it was zero [11] and [12]. The Postscaler divides the frequency, which comes out of the comparator for the last time before being used.

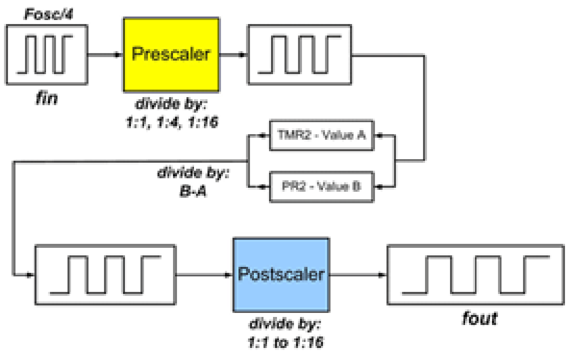


Figure 8. Timer2 schema[11]

Table 6

-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS2
Bit7	-	-	-	-	-	-	Bit0

Table 7

T2CKPS0	T2CKPS0	Prescaler
0	0	1
0	1	4
1	0	16
1	1	16

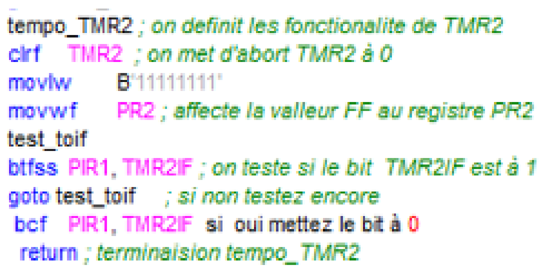
Timer2 is configured from register T2CON, accessible in bank 0 (table 6).

In timer2 we only have the timer mode, and activation will be done by setting the TMR2ON bit in the register to 1.Often The input clock is divided by 4 (FOSC/4), but in timer2 we have another prescaler option selected by the T2CKPS1 and T2CKPS0 control bits (table 7).

The bits from TOUTPS3 to TOUTPS0 are used by a postscaler which gives a scale of 1 /1, 1 /4 or 1/16 selected by the bits TOUTPS0, TOUTPS1, TOUTPS2 and TOUTPS0. The program to determine the maximum time for the device using register TMR2 is written as follows:

4. Result and discussions

For testing purposes, the device is duplicated in three copies, named timer0, timer1 and timer2 respectively,



```
tempo_TMR2 ; on definit les fonctionalite de TMR2
clrf    TMR2  ; on met d'abort TMR2 à 0
movlw       B'11111111'
movwf       PR2 ; affecte la valleur FF au registre PR2
test_toif
btfss  PIR1, TMR2IF ; on teste si le bit  TMR2IF est à 1
goto test_toif     ; si non testez encore
 bcf    PIR1, TMR2IF  si  oui mettez le bit à 0
  return ; terminaision tempo_TMR2
```

Figure 9. Timer2 code

Table 8

Frequency	20MHz	16MHz	8MHz	4MHz	1MHz	100KHz

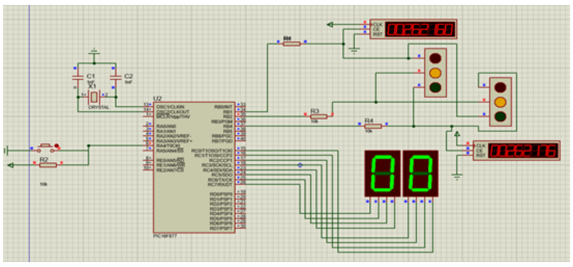


Figure 10. Test Timer0

each of which implements the assembly code it needs to recover the maximum duration that each timer of a pic16F877can do or can work in parallel with the other processes before returning to its final value with different frequencies. The frequencies used are authorized frequencies used in Pic (table 8).

The Pic16F877 uses a maximum frequency of 20 MHz; it can also operate with the above frequencies. With Timer0, to obtain the maximum time with any frequency, use the maximum prescaler and initialize the TMR0 value to 0. If TMR0 is set to 0, the counter will have to make 256 turns before returning to 0, so we have:

$$T_{Max} = \frac{1}{Frequency} * prescaler * maximumpredivisor * tours(256 - ValueTMR0) \quad (1)$$

Example: for f = 1MHz; prescaler = 4 ; maximum predivisor = 256; tMR0=0 we have TMax= 1/1000 000 * 4 * 256 * 256 = 0.262144second = 262ms.

After checking the device, we find the same result for the frequencies.

After calculating each of the frequencies below using Timer0 we have (table 9):

For Timer1, the operating principle is similar to that of Timer0. Since we have 16 bits here, the number of pulses the microcontroller needs to make to close the time register is 65536. The maximum prescaler for timer1 is 8, so the formula for calculating the timer duration is :

$$T_{Max} = \frac{1}{Frequency} * prescaler * predivisor * 65536 \quad (2)$$

Example: for f = 1MHz we have: TMax = 1/100000 * 4 * 8 * 65536 = 2.097152 Second, Same result found in the device.

For each of the frequencies below we have (table 10):

For Timer2, if we want the maximum delay time, the postdivider and predivider values must be set to 16; the PR2 value must be initialized to 255 and the TMR2 value must be set to 0. The formula for calculating the delay time is therefore:

$$T_{Max} = \frac{1}{Frequency} * prescaler * postdivisorvalue * predivisor * (ValuePR2 + 1) \quad (3)$$

if we take f = 1MHz, TMax = 1/1000000 * 4 * 16 * 16 * (255 + 1) = 0.262144 seconds = 262ms. We can see that we have the same results as Timer0 (table 11).

If we draw up a timer graph, we can see that the maximum dwell time for timer0 and timer2 are the same for the same frequencies, whereas for timer1, as

Table 9

Frequency	20MHz	16MHz	8MHz	4MHz	1MHz	100KHz
$T_{Max(seconde)}$	0,0131072	0,016384	0,032768	0,065536	0,262144	2,62144

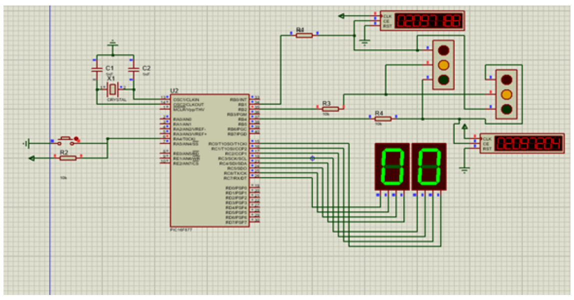


Figure 11. Test Timer1

Table 10

Frequency	20MHz	16MHz	8MHz	4MHz	1MHz	100KHz
$T_{Max(seconde)}$	0,1048576	0,131072	0,262144	0,524288	2,097152	20,97152

Table 11

Frequency	20MHz	16MHz	8MHz	4MHz	1MHz	100KHz
$T_{Max(seconde)}$	0,0131072	0,016384	0,032768	0,065536	0,262144	2,62144

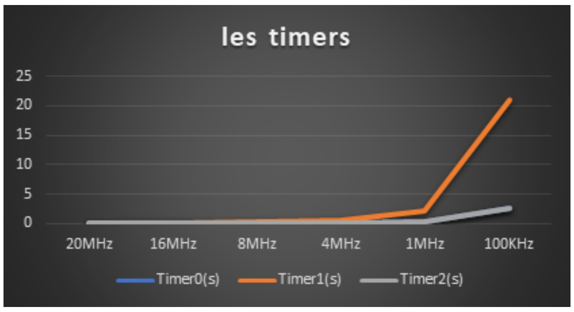


Figure 12. Comparison of timers

soon as the frequency is equal to or less than 8MHz, it starts to detach from the other timers.

5. Conclusion

A timer is a counter register that increments with each pulse of a microcontroller clock without interrupting or pausing other programs. When the timer reaches a certain value, it overflows, and when it overflows a flag is set to 1 in a control register. By monitoring this flag, it is possible to count the time that elapses until a given duration is reached. Timers are becoming increasingly present in microcontrollers. This increase in the use of timers in microcontrollers is due to the fact that timers have better management of task execution time than the processor. In addition, timers enable the CPU to be faster and the time defined will also be precise in time, but the use of these timers requires mastery of all the registers associated with the timer. This article provides details of how the timer registers work and also compares the three timers available in PIC16F877. From this comparison we can see that Timer0 and Timer2 have the same maximum duration for each frequency, whereas for Timer1, as soon as the frequency exceeds 8 MHz, the maximum duration that can be used to delimit a task is greater than the maximum duration of Timer0 and Timer2.

References

[1] Khairurrijal, M. Abdullah, et M. Budiman:Home-made PIC 16F877 microcontroller-based temperature control system for learning automatic control, Comput. Appl. Eng. Educ., vol. 19, no 1, p. 10-17, mars 2011, doi: 10.1002/cae.20283.

[2] G. Ilangovan, M. R. Ezilarasan, et M. Thanjaivadivel: A Review on Piezo-Electric Power Harvesting Using Switch Harvesting on Inductor, Int. J. Mech. Eng. Technol., vol. 8, no 10, 2017.

[3] S. Rustemli et M. Ates: Measurement and simulation of power factor using pic16f877, Przeglad Elektrotechniczny Electr. Rev., vol. 88, no 6, p. 290-294, 2012.

[4] I. Çolak et R. Bayindir:PIC 16F877 ile DA Motor Hiz Kontrolü, Pamukkale Üniversitesi Mühendis. Bilim. Derg., vol. 11, no 2, p. 277-285, 2011.

[5] A. Isik, O. Karakaya, P. A. Öner, et M. K. Eker: PMDC motor speed control with fuzzy logic algorithm using PIC16F877 micro controller and plotting data on monitor, in 2009 Fifth International Conference on Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control, Ieee, p. 1-4, 2009,.

[6] S. Yilmaz, B. TOMBALOGLU, K. KARABULUTLU, Y. GUMUS, et H. DiNCER: Temperature Control Applications by means of a pic16f877 Microcontroller, Univ. Kocaeli Electron. Commun. Res. Appl. Cent.-EHSAM Turk., 2001.

[7] Christian Tavernier:Les Microcontrôleurs PIC: Description et mise en oeuvre, Dunod - Paris, 2ème édition, p.224, 2002

[8] K. S. Adarsh, A. Dinesh et D. Jyothy Elizebeth:E-Uniform For Soldier's Who Work At Extreme Temperature Regions, Int. J. Eng. Res. Gen. Sci., vol. 3, no 3, p. 993-998, 2015.

[9] Osolinskyi Oleksandr, Agnieszka Molga, Volodymyr Kochan, Anatoliy Sachenko: Method of Ensuring the Noise Immunity at Measurement of Single-Board Microcontroller Average Energy Within IoT Environment, 2020 IEEE 40th International Conference on Electronics and Nanotechnology (ELNANO), 22-24 April, Kyiv, Ukraine, 2020

[10] Christian Tavernier: Microcontrôleurs PIC (10, 12, 16): Description et mise en oeuvre, Dunod - Paris, 3ème édition, p.344, 2007.

[11] Christian Tavernier: Microcontrôleurs PIC : programmation en Basic, Dunod - Paris, p.265, 2006

[12] Hafedh Sakka: Le microcontrôleur PIC 16F877 de la description à l'application, Centre de Publication Universitaire - Tunisie, p. 220, 2015.