German-Russian Institute of Advanced Technologies

TU-Ilmenau (Germany) and KNTRU-KAI (Kazan, Russia)

Guidelines for laboratory work №6 of the subject

«**Computer systems**»

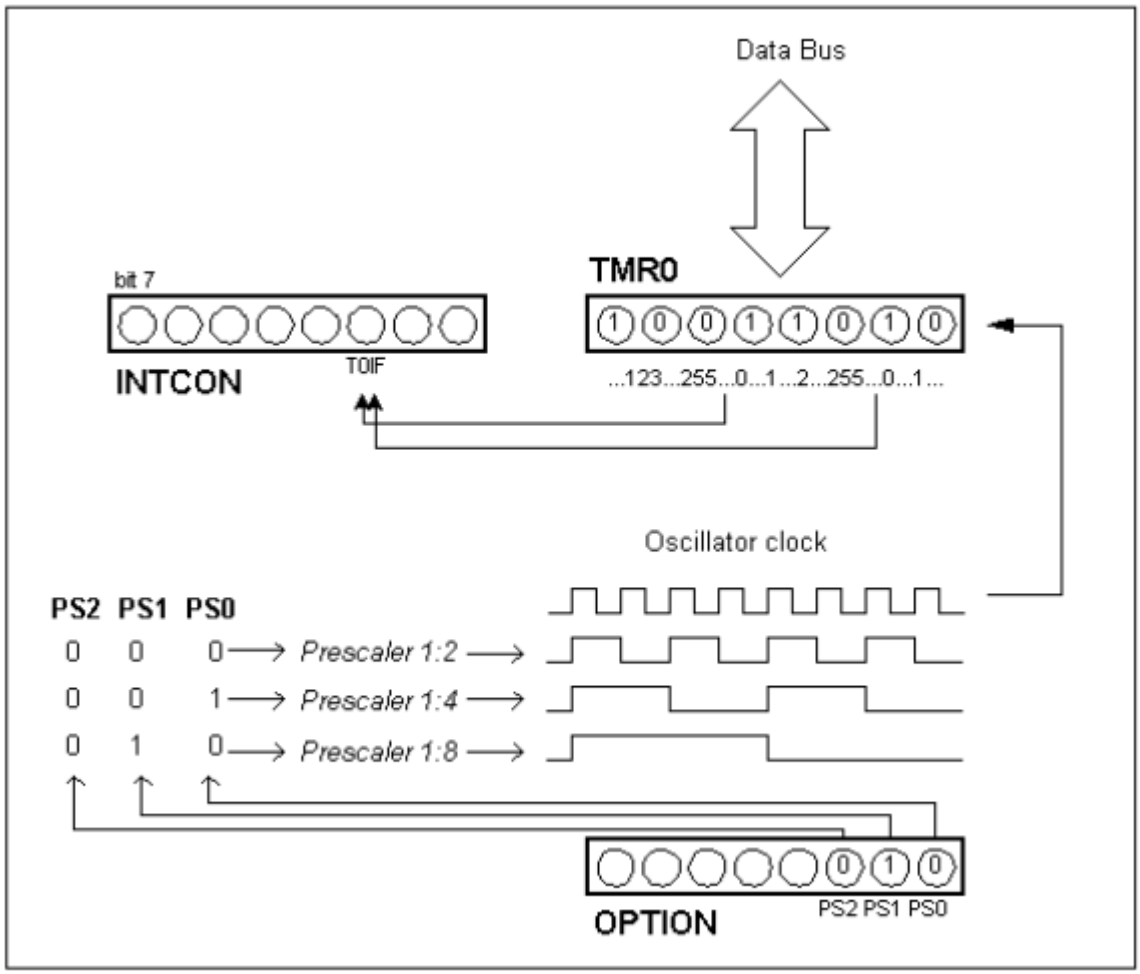
« Proteus Virtual System Modeling (VSM)
PART I. TMR0 Application Counter Using TMR0.
PART II. EEPROM Memory Application»

Kazan 2014

Proteus Virtual System Modeling (VSM)

Part I. TMR0 Application Counter Using TMR0

Timers are usually most complicated parts of a microcontroller, so it is necessary to set aside more time for their explaining. With their application it is possible to create relations between a real dimension such as "time" and a variable which represents status of a timer within a microcontroller. Physically, timer is a register whose value is continually increasing to 255, and then it starts all over again: 0, 1, 2, 3, 4...255....0, 1, 2, 3.....etc. The following figure illustrates this.



More Information about TMR0

This incrementing is done in the background of everything a microcontroller does. It is up to programmer to "think up a way" how he will take advantage of this characteristic for his needs. One of the ways is increasing some variable on each timer overflow. If we know how much time a timer needs to make one complete

round, then multiplying the value of a variable by that time will yield the total amount of elapsed time.

PIC16F84 has an 8-bit timer. Number of bits determines what value timer counts to before starting to count from zero again. In the case of an 8-bit timer, that number is 256. A simplified scheme of relation between a timer and a prescaler is represented on the previous diagram. Prescaler is a name for the part of a microcontroller which divides oscillator clock before it will reach logic that increases timer status. Number which divides a clock is defined through first three bits in OPTION register. The highest divisor is 256. This actually means that only at every 256th clock, timer value would increase by one. This provides us with the ability to measure longer timer periods.

TMR0 and Reset

After each count up to 255, timer resets its value to zero and starts with a new cycle of

counting to 255. During each transition from 255 to zero, T0IF bit in INTCOM register is set. If interrupts are allowed to occur, this can be taken advantage of in generating interrupts and in processing interrupt routine. It is up to programmer to reset T0IF bit in interrupt routine, so that new interrupt or new overflow could be detected. Beside the internal oscillator clock, timer status can also be increased by the external clock on RA4/TOCKI pin. Choosing one of these two options is done in OPTION register through T0CS bit. If this option of external clock was selected, it would be possible to define the edge of a signal (rising or falling), on which timer would increase its value.

Option Control Register

Bit 0:2 **PS0, PS1, PS2** (Prescaler Rate Select bit)

The subject of a prescaler, and how these bits affect the work of a microcontroller will be covered in section on TMR0.

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{RBPU}$ (1)	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Legend:
R = Readable bit **W** = Writable bit
U = Unimplemented bit, read as '0' -n = Value at POR reset

bit 3 **PSA** (Prescaler Assignment bit)

Bit which assigns prescaler between TMR0 and watchdog timer.

1=prescaler is assigned to watchdog timer.

0=prescaler is assigned to free timer TMR0

bit 4 T0SE (TMR0 Source Edge Select bit)

Bits	TMR0	WDT
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

If trigger TMR0 was enabled with impulses from a RA4/T0CKI pin, this bit would determine whether it would be on the rising or falling edge of a signal.

1=falling edge

0=rising edge

bit 5 **T0CS** (TMR0 Clock Source Select bit)

This pin enables a free-run timer to increment its value either from an internal oscillator, i.e. every 1/4 of oscillator clock, or via external impulses on RA4/T0CKI pin.

1=external impulses

0=1/4 internal clock

bit 6 I**NTEDG** (Interrupt Edge Select bit)

If occurrence of interrupts was enabled, this bit would determine at what edge interrupt on RB0/INT pin would occur.

1= rising edge

0= falling edge

bit 7 **RBPU** (PORTB Pull-up Enable bit)

This bit turns internal pull-up resistors on port B on or off.

1='pull-up' resistors turned on

0='pull-up' resistors turned off

TASKS

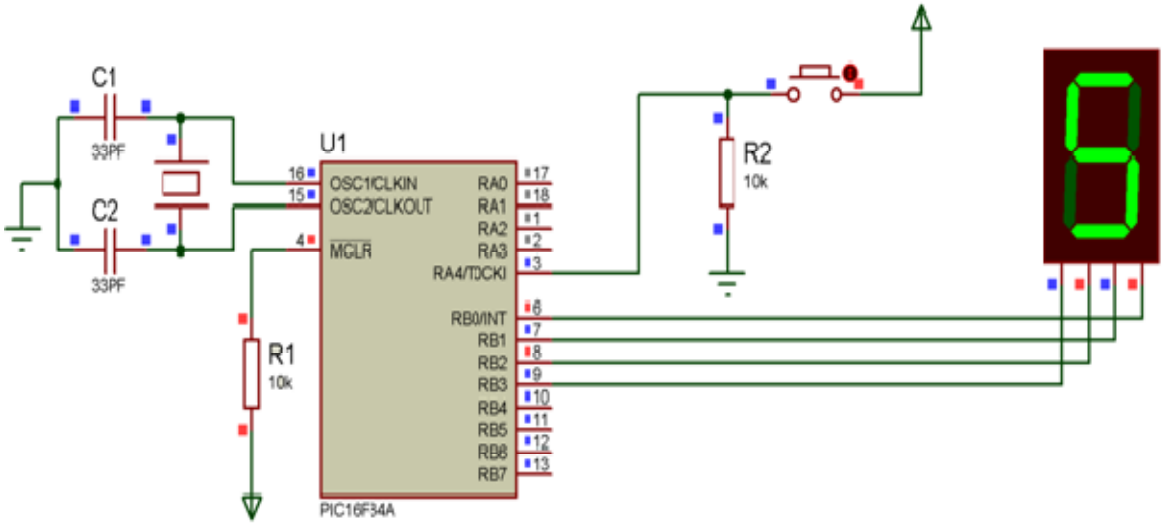
TMR0

• Write an assembly program to make a Counter using TMR0; the counter should increment its value on every 2 pushbuttons on RA4.

• Simulate the program using the circuit shown in figure via Proteus software. Verify it operates properly when simulated

• Program a PIC 16F84A using the QL2006 programmer.

• Build the circuit using the programmed PIC 16F84A and then observe its operation. Demonstrate the circuits operation to the instructor.



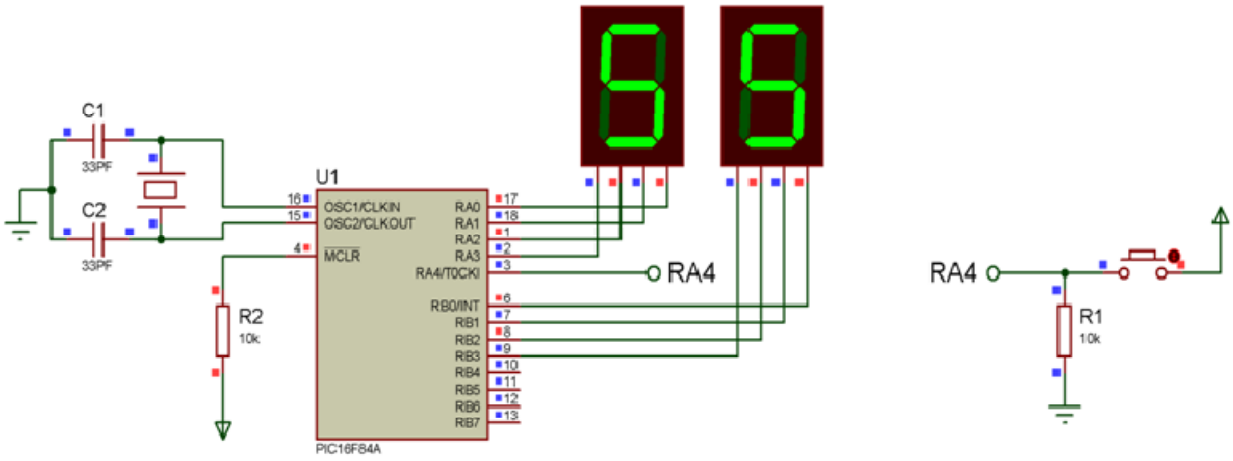
Watchdog Timer (WDT)

• Write an assembly program to make a Counter using WDT; the counter should increment its value on every one single pushbutton on RA4, and also counts from 0 to 99.

• Simulate the program using the circuit shown in figure via Proteus software. Verify it operates properly when simulated

• Program a PIC 16F84A using the QL2006 programmer.

Build the circuit using the programmed PIC 16F84A and then observe its operation. Demonstrate the circuits operation to the instructor. Present your results in a lab report including a copy of the source codes.



PART II. EEPROM Memory Application

PIC16F84 has 64 bytes of EEPROM memory locations on addresses from 00h to 63h those can be written to or read from. The most important characteristic of this memory is that it does not lose its contents during power supply turned off. That practically means that what was written to it will be remaining even if microcontroller is turned off. Data can be retained in EEPROM without power supply for up to 40 years (as manufacturer of PIC16F84 microcontroller states), and up to 10000 cycles of writing can be executed.

In practice, EEPROM memory is used for storing important data or some process parameters.

One such parameter is a given temperature, assigned when setting up a temperature regulator to some process. If that data wasn't retained, it would be necessary to adjust a given temperature after each loss of supply. Since this is very impractical (and even dangerous), manufacturers of microcontrollers have began installing one smaller type of EEPROM memory.

EEPROM memory is placed in a special memory space and can be accessed through special registers. These registers are:

• EEDATA at address 08h, which holds read data or that to be written.

• EEADR at address 09h, which contains an address of EEPROM location being accessed.

• EECON1 at address 88h, which contains control bits.

• EECON2 at address 89h. This register does not exist physically and serves to protect EEPROM from accidental writing.

EECON1 register at address 88h is a control register with five implemented bits. Bits 5, 6 and 7 are not used, and by reading always are zero. Interpretation of EECON1 register bits follows.

U-0	U-0	U-0	R/W-1	R/W-1	R/W-x	R/S-0	R/S-x
—	—	—	EEIF (1)	WRERR	WREN	WR	RD
bit 7							bit 0

Legend:
R = Readable bit **W** = Writable bit
U = Unimplemented bit, read as '0' -n = Value at POR reset

Bit 0 **RD** (Read Control bit)

Setting this bit initializes transfer of data from address defined in EEADR to EEDATA register. Since time is not as essential in reading data as in writing, data from EEDATA can already be used further in the next instruction.

1=initializes reading

0=does not initialize reading

Bit 1 **WR** (Write Control bit)

Setting of this bit initializes writing data from EEDATA register to the address specified trough EEADR register.

1=initializes writing

0=does not initialize writing

Bit 2 **WREN** (EEPROM Write Enable bit) Enables writing to EEPROM If this bit was not set, microcontroller would not allow writing to EEPROM.

1=writing allowed

0=writing disallowed

Bit 3 **WRERR** (Write EEPROM Error Flag) Error during writing to EEPROM This bit was set only in cases when writing to EEPROM had been interrupted by a reset signal or by running out of time in watchdog timer (if it's activated).

1=error occured

0=error did not occur

Bit 4 **EEIF** (EEPROM Write Operation Interrupt Flag bit) Bit used to inform that writing data to EEPROM has ended.

When writing has terminated, this bit would be set automatically. Programmer must clear EEIF bit in his program in order to detect new termination of writing.

1=writing terminated

0=writing not terminated yet, or has not started

Procedure of Writing Operation

1. Put the address which you want to write to in EEADR.
2. Put the data which you want to write in EEDATA.
3. Disable all interrupts (GIE = 0).
4. Set the bin WREN in EECON1.
5. Put the Keys in EECON2 (first 0X55 then 0XAA).
6. Set the bin WR in EECON1.
7. Wait until Writing Operation is finished.

Procedure of Reading Operation

1. Put the address which you want to read from in EEADR.
2. Set the bin RD in EECON1.
3. Move the read data from EEDATA to the work register.

We follow these procedures in all times we want to write to or read from EEPROM Memory, except that the address which we want to write to in writing operation /or read from in reading operation; and the data which we want to write in writing operation/ or read in reading operation differ each time.

TASKS

• Write an assembly program to fill all the EEPROM Memory locations With 7.

Hint *// Build an external Macro called EEPROM_Writing takes two parameters the data and the address to achieve the writing operation; then call it in the main program.*

• Simulate your program using the software PIC Simulator IDE.

• Write an assembly program to take the data existed on PORT A and display it on PORT B; first, the data must be taken from PORT A and stored in the EEPROM address location 0X10, and then be taken again from EEPROM and be displayed on PORT B.

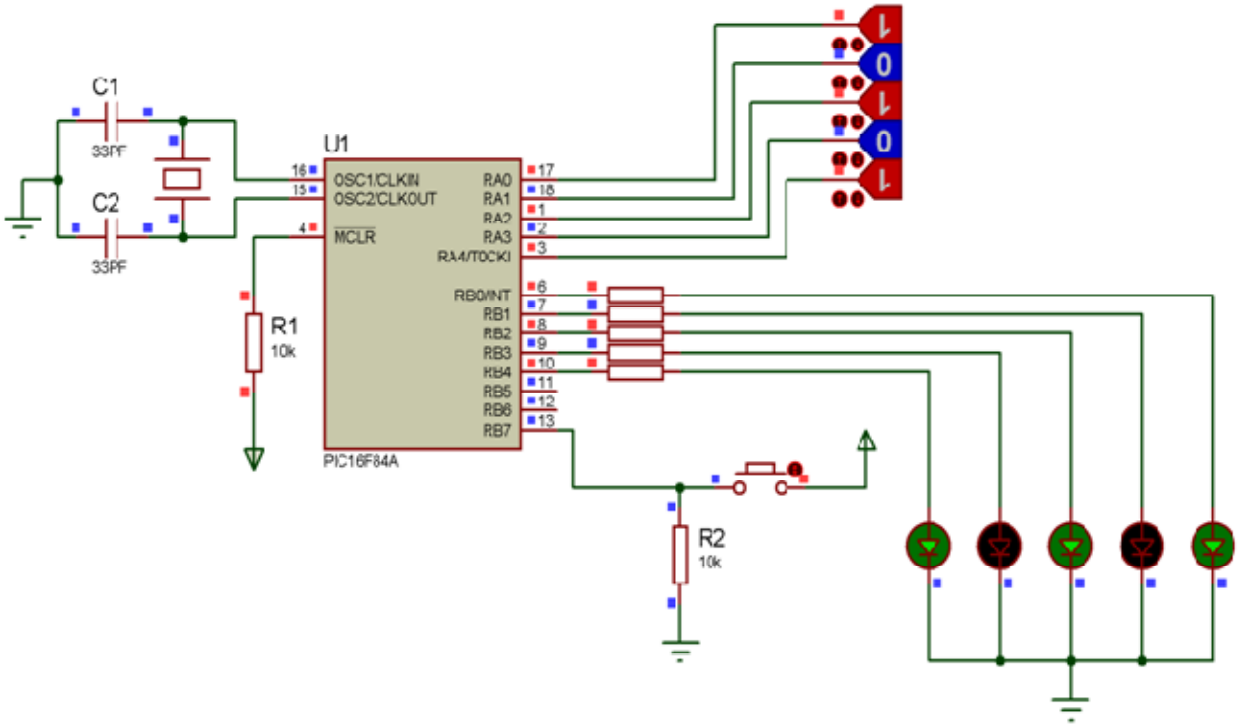
• You should use the EEPROM_Writing Macro from the previous part for writing operation, and also build a nother macro EEPROM_Reading for reading operation.

• Simulate your program using the software PIC Simulator IDE.

• Simulate the program using the circuit shown in figure via Proteus software. Verify it operates properly when simulated

• Program a PIC 16F84A using the QL2006 programmer.

• Build the circuit using the programmed PIC 16F84A and then observe its operation. Demonstrate the circuits operation to the instructor. Present your results in a lab report including a copy of the source codes.



CONTROL QUESTIONS

1. What is timer?
2. Why do we need timers?
3. What is Option Control Register?
4. What is EEPROM memory?

REPORT FORM

German-Russian Institute of Advanced Technologies
TU-Ilmenau (Germany) and KNTRU-KAI (Kazan, Russia)

Laboratory work 6

« Proteus Virtual System Modeling (VSM)
PART I. TMR0 Application Counter Using TMR0.
PART II. EEPROM Memory Application»

Student: _____

Teacher: _____

Kazan 2014