

UNIVERSITETI I PRISHTINËS
FAKULTETI I INXHINIERISË ELEKTRIKE DHE KOMPJUTERIKE



DOKUMENTACIONI I PROJEKTIT

Lënda: Mikroprocesorë dhe mikrokontrollerë

Tema: Gjeneratori i frekuencave 0-40MHz AD9850 (MODBUS RTU)

Profesori: Mr.Sc Lavdim Kurtaj

Studentët: Ardi Rexhepi
Blerta Krasniqi
Elvisa Gashi
Mirjetë Hajdari
Mrika Govori
Perparim Dubova
Shpresarta Hamza

Prishtinë, 2019

Përmbajtja

1	Hyrje	3
2	Moduli AD9850	4
2.1	Tiparet dhe Perdorimi	5
2.1.1	Tiparet:	5
2.1.2	Aplikimet:	6
2.2	Definimi i pin-ave	6
3	Pllaka Easy8051A.....	7
4	Mikrokontrolleri dhe ndërfaqja	8
5	MODBUS	8
6	Ndërfaqja grafike në PC (GUI)	10
7	Nderfaqja njeri-makine	12

1 Hyrje

Qëllimi i realizimit të këtij projekti është gjenerimi i sinjaleve 0-40MHz me AD9850 (MODBUS RTU). Do të shfrytëzohet moduli **AD9850** i cili përmes komunikimit paralel 8 bitësh do të komunikojë me mikrokontrollerin **AT89S8253** i cili i takon familjes të mikrokontrollerëve 8051.

Modbus shpesh përdoret për të lidhur një kompjuter mbikëqyrës me një njësi terminalesh të largët (RTU) në sistemet e mbikëqyrjes dhe mbledhjes së të dhënave (SCADA). Versionet e protokollit Modbus ekzistojnë për linjat serike (Modbus RTU dhe Modbus ASCII) dhe për Ethernet (Modbus TCP).

Modbus RTU mode është zbatimi më i zakonshëm, qe përdor kodimin binar dhe CRC kontrollin e gabimeve. Mesazhet ASCII Modbus (megjithëse paksa më të lexueshme për shkak se përdorin karaktere ASCII) janë më pak efikase dhe përdorin kontrollin e gabimeve më pak efektiv LRC.

2 Moduli AD9850

AD9850 është një pajisje më e integruar që përdor Teknologjinë DDS të avancuar të shoqëruar me një shpejtësi të brendshme të lartë, konvertues me performancë të lartë D/A dhe krahasues për të formuar një sintetizues të frekuencave të programueshme digjitale dhe funksion të gjeneratorëve të clockut. Kur referohet në një burim të saktë të clockut, AD9850 gjeneron një spekter të pastër, frekuencë/ fazë të programueshme, output analog i vales sinusoidale. Kjo vale sinus mund të përdoret drejtpërdrejt si burim i frekuencës, ose mund të konvertohet në një vale katrore për aplikacionet e gjeneratorëve të agile-clock.

Arkitektura e qarkut AD9850 lejon prodhimin e frekuencave të prodhimit deri në gjysmën e frekuencës së clockut referues (ose 62.5 MHz), dhe frekuenca e daljes mund të ndryshohet në mënyrë digjitale (asinkrone) me një normë deri në 23 milionë frekuenca të reja për sekondë.

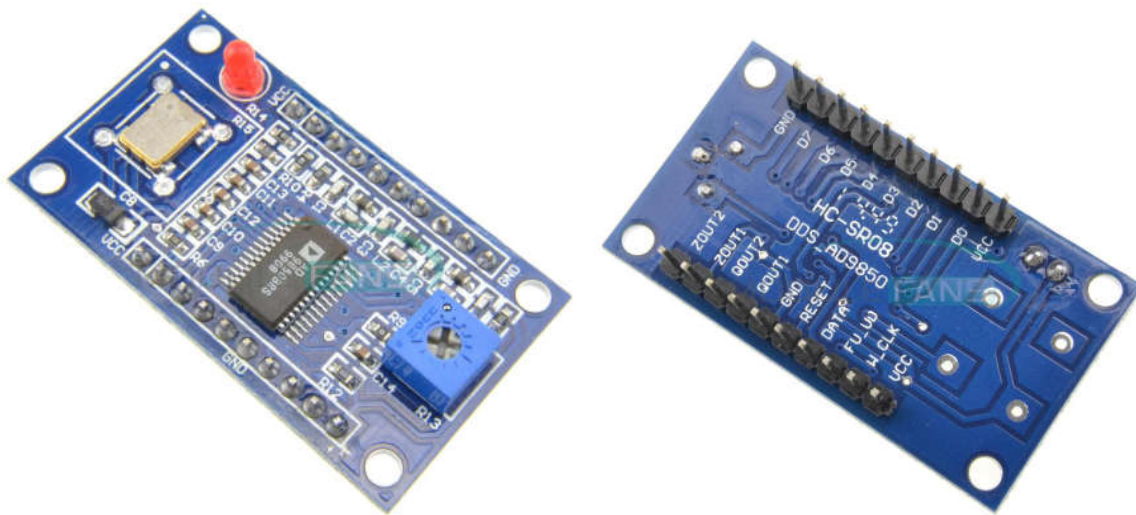


Figura 1. Pamje e modulit AD9850

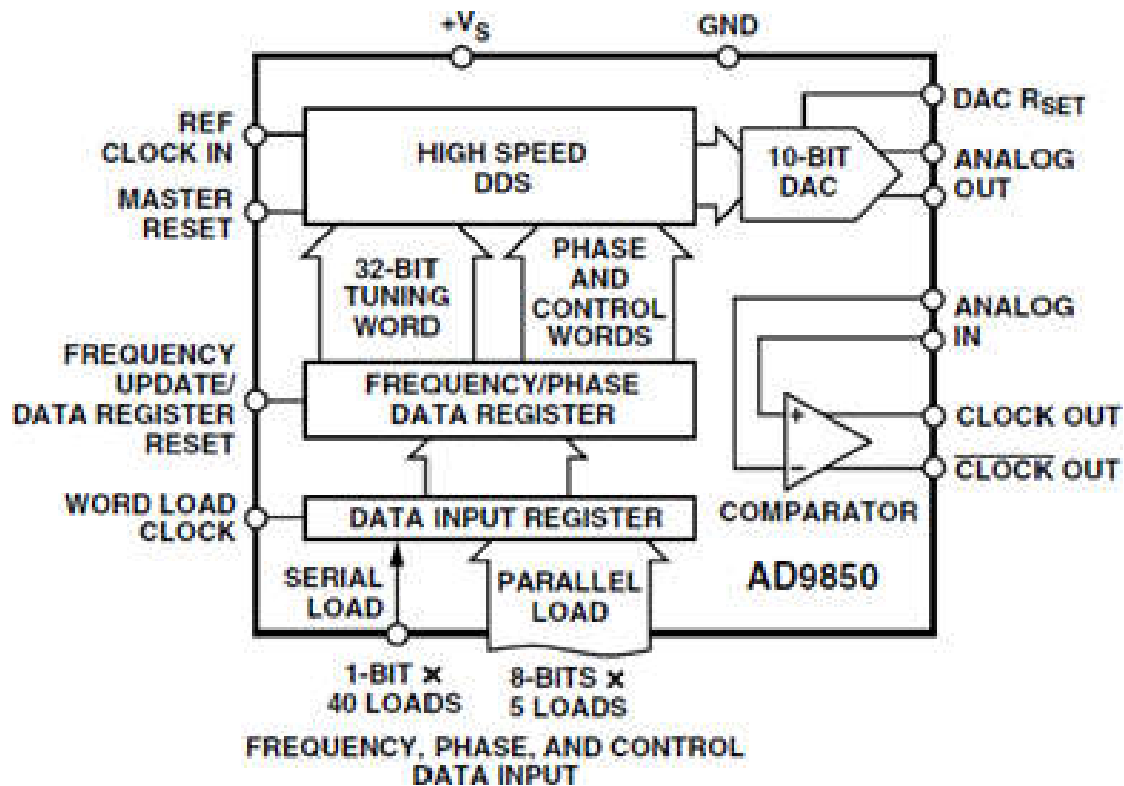


Figura 2.

2.1 Tiparet dhe Perdorimi

2.1.1 Tiparet:

- Shpejtësia e prodhimit të sinjalit me frekuencë: 0-40MHz
- 4 Signal outputs
- 2 dalje të valës sinusoidale dhe 2 dalje në valë katrore
- DAC SFDR > 50 dB @ 40 MHz AOUT
- 32-Bit Frekuenca Tuning Word
- Interface i thjeshtuar i kontrollit: Bajta paralel ose format i ngarkimit serial
- Faza e Modulimit të Aftësis (Phase Modulation Capability)
- +3.3 V ose +5 V Operim i Vetëm i Furnizimit
- Low Power: 380 mW @ 125 MHz (+5 V)
- Low Power: 155 mW @ 110 MHz (+3.3 V)
- Funksioni Power-Down
- Size: 42 x 30 x 1.6 mm

2.1.2 Aplikimet:

- Frekuenca / Faza-agile Sintetizimi Sine-Wave
- Rigjenerim i clockut dhe Bllokimi i qarkut për Dixhital
- Komunikimi
- Digitally Controlled ADC Encode Generator
- Agile Local Oscillator Applications

2.2 Definimi i pin-ave

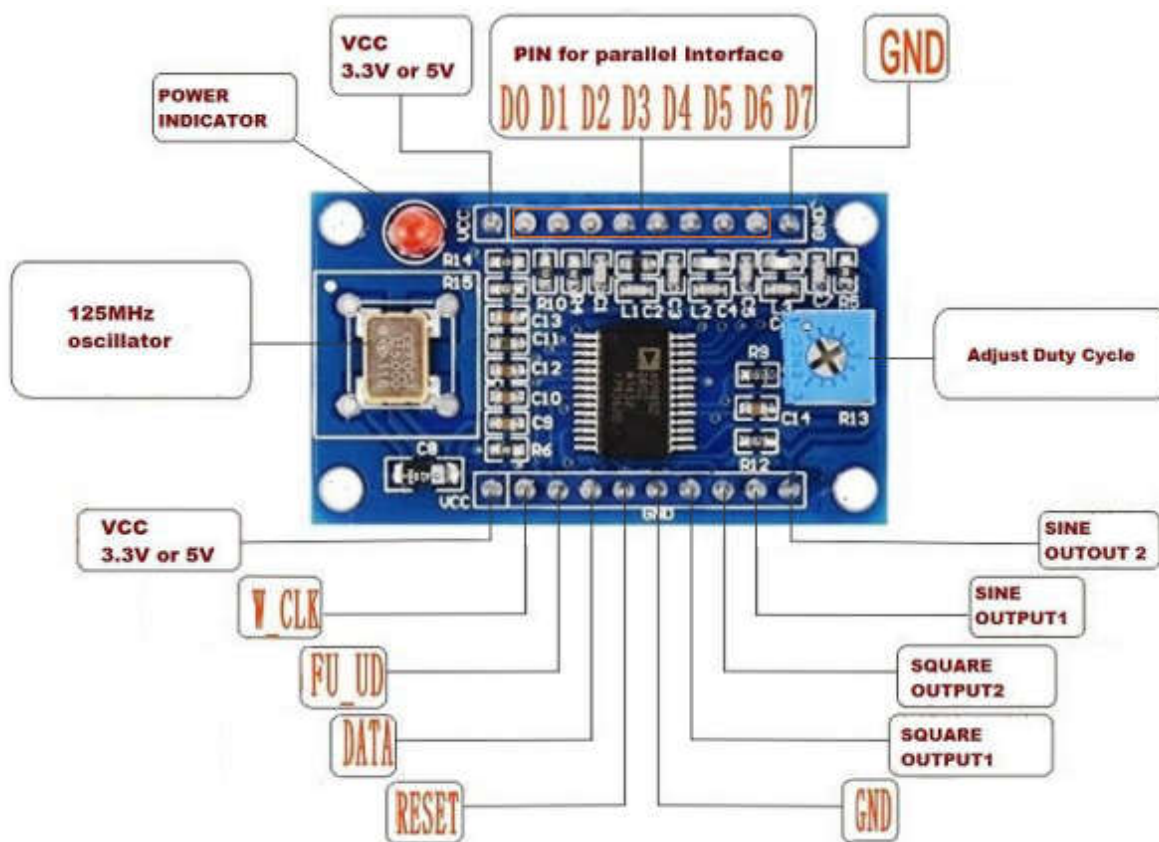


Figura 3. Permbajtja e modulit

Sensori per perdorim ka ne dalje pinat ZOUT2, ZOUT1, QOUT2, QOUT1, GND, RESET, DATA, FU_UD, W_CLK, VCC dhe GND, D7, D6, D5, D4, D3, D2, D1, D0, VCC.

PIN-i	Lidhja
ZOUT2	...

ZOUT1	Oscilloscope
QOUT2	Nuk lidhet
QOUT1	Nuk lidhet
GND	
RESET	P1.6
DATA	
FU_UD	P1.5
W_CLK	P.1.4
VCC	

Porti 0		
PIN-i		Lidhja
GND		GND
D7		P0.7
D6		P0.6
D5		P0.5
D4		P0.4
D3		P0.3
D2		P0.2
D1		P0.1
D0		P0.0
VCC		VCC

3 Pllaka Easy8051A

Pllaka zhvillimore e mikrokontrollerit 8051 është përdorur për kontrollimin e sistemit, leximin e references dhe sensorit, leximin e ndërprerësve kufitar, kontrollin e AD9850 modulit dhe paraqitjen e një interfejsi të thjeshtë me njeriun.

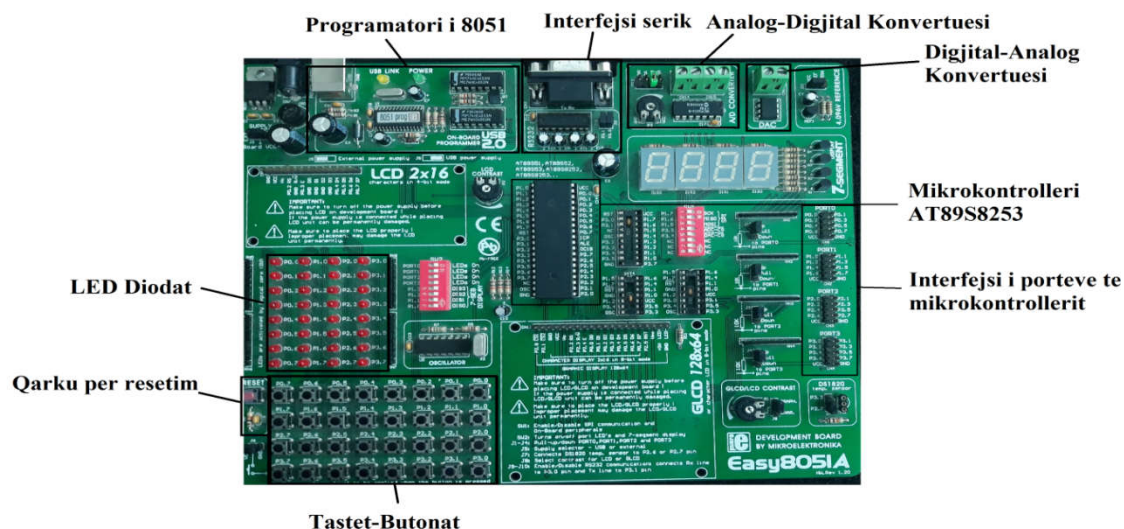


Figura 4. Pllaka Easy8051A dhe pjeset e saj

4 Mikrokontrolleri dhe ndërfaqja

Mikrokontrolleri i cili do të përdoret për realizimin e projektit do të jetë AT89S8253 i cili i takon familjes 8051.

(T2) P1.0	1	40	VCC
(T2 EX) P1.1	2	39	P0.0 (AD0)
P1.2	3	38	P0.1 (AD1)
P1.3	4	37	P0.2 (AD2)
(SS) P1.4	5	36	P0.3 (AD3)
(MOSI) P1.5	6	35	P0.4 (AD4)
(MISO) P1.6	7	34	P0.5 (AD5)
(SCK) P1.7	8	33	P0.6 (AD6)
RST	9	32	P0.7 (AD7)
(RXD) P3.0	10	31	EA/VPP
(TXD) P3.1	11	30	ALE/PROG
(INT0) P3.2	12	29	PSEN
(INT1) P3.3	13	28	P2.7 (A15)
(T0) P3.4	14	27	P2.6 (A14)
(T1) P3.5	15	26	P2.5 (A13)
(WR) P3.6	16	25	P2.4 (A12)
(RD) P3.7	17	24	P2.3 (A11)
XTAL2	18	23	P2.2 (A10)
XTAL1	19	22	P2.1 (A9)
GND	20	21	P2.0 (A8)

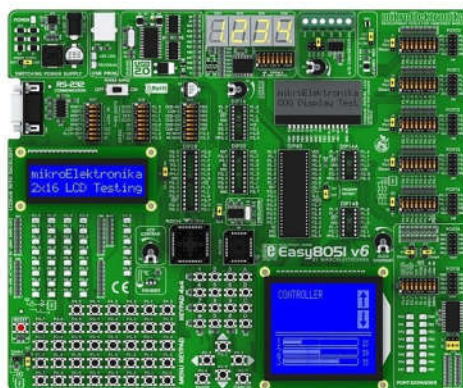


Figura 5. Pinat e 8051 dhe pamje e pllakës Easy8051A

Disa nga specifikat teknike të mikrokontrollerit AT89S8253 janë:

- Kompatibil me produktet e familjes MCS51
- Memorie të brendshme flash 12K Bytes (10,000 cikle lexim/shkrim)
- Memorie për të dhëna EEPROM 2K Bytes
- Memorie RAM 256 x 8 bit
- 32 Linja I/O të programueshme
- Tre kohorë/numërues 16 bit

Mikrokontrolleri do të jetë i instaluar në pllakën zhvillimore **Easy8051A** e cila përmbanë të instaluar edhe disa nga butonat dhe display 7 segmentësh të cilët do të përdorim për të “shfletuar” vlerat e lexuara nga sensori.

5 MODBUS

Modbus është një protokoll i komunikimit që është zhvilluar nga Modicon systems. Përndryshe MODBUS është një metodë që përdoret për transmetim apo bartje të të informacioneve përgjatë linjave serike ndërmjet pajisjeve elektronike.

Protokolli Modbus definohet si një protokoll master / slave, që do të thotë një pajisje që vepron si një master do të drejtojë një ose më shumë pajisje që veprojnë si slaves. Kjo do të thotë që një pajisje slave nuk mund të japë informacion vullnetar; duhet të presë që të kërkohet. Masteri do të

mundet të shkruajë të dhëna në një regjistër të pajisjes slave dhe do të lexojë të dhëna nga regjistrat e një slave.

Forma më e përdorur e protokollit Modbus është RTU.

Modbus RTU është një protokoll serik relativisht i thjeshtë që mund të transmetohet nëpërmjet teknologjisë tradicionale UART. Të dhënat transmetohen në bytes 8-bit, një bit në një kohë, në normat e baud rate që shkojnë nga 1200 bit për sekondë (baud) deri në 115200 bit për sekondë. Shumica e pajisjeve të Modbus RTU mbështesin vetëm shpejtësinë deri në 38400 bit për sekondë.

Një rrjet Modbus RTU ka një Master dhe një ose më shumë slaves. Secili slave ka një adresë unike të pajisjes 8-bit ose numrin e njësisë. Paketat e dërguara nga master përfshijnë adresën e slave-it për të cilin është drejtuar mesazhi. Slave duhet të përgjigjet vetëm nëse adresa e tij është e njohur dhe duhet të përgjigjet brenda një periudhe të caktuar kohore ose masteri do ta quajë si gabim "asnje përgjigje".

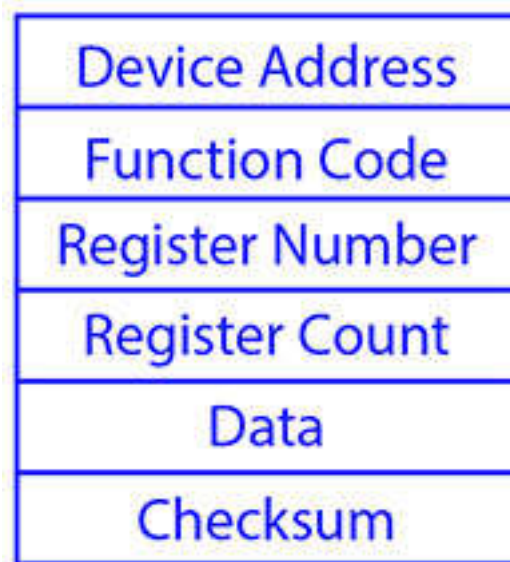


Figura 6 Paketa e Modbus

Çdo shkëmbim i të dhënave përbëhet nga një kërkesë nga masteri, e ndjekur nga një përgjigje nga slave. Çdo pakete e të dhënave, qoftë kërkesë ose përgjigje, fillon me adresën e pajisjes ose adresën e slave, e ndjekur nga kodi i funksionit, i ndjekur nga parametrat që përcaktojnë se çfarë kërkohet apo jepet.

Të dhënat Modbus janë më së shpeshti të lexuara dhe të shkruara si "regjistra" të cilët janë të dhëna 16-bitëshe. Zakonisht, regjistri është ose një numër i plotë signed ose unsigned 16-bit. Nëse kërkohet një numër i plotë 32-bit ose një floating point, këto vlera lexohen në të vërtetë si dy regjistra. Regjistri më i zakonshëm quhet Holding Register, dhe këto të dhëna mund të lexohen ose shkruhen. Lloji tjetër i mundshëm është Input Register, i cili është vetëm i lexueshëm.

Përrjashtimet për regjistrat që janë 16 bit janë coil dhe inputi diskret, të cilat janë vetëm 1 bit. Coil mund të lexohet ose të shkruhet, ndërsa inputet diskrete janë vetëm të lexueshme.

Lloji i regjistrit që adresohet nga një kërkesë Modbus përcaktohet nga kodi i funksionit(Function Code). Kodi i funksionit 6 përdoret për të shkruar një regjistër të vetëm mbajtës(Holding Register). Kodi i funksionit 16 përdoret për të shkruar një ose më shumë regjistra mbajtës(Holding Register).

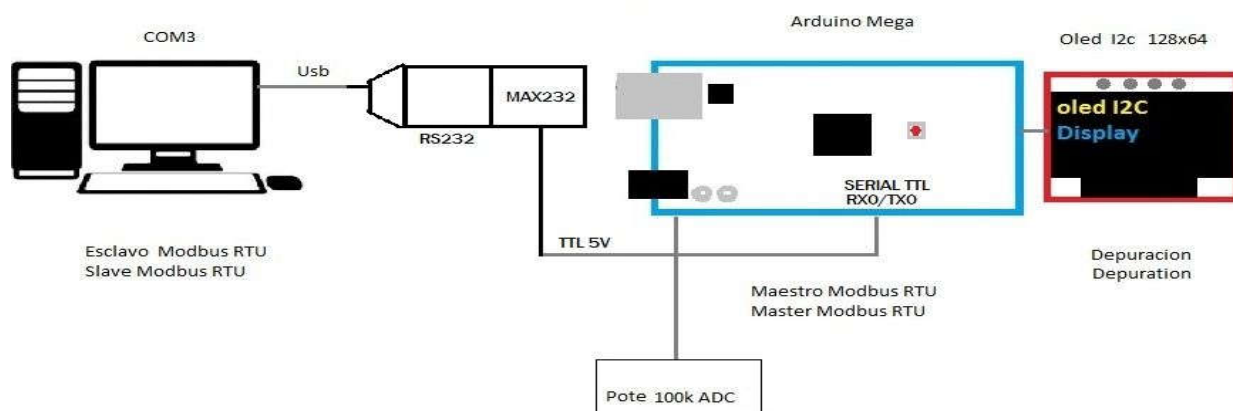


Figure 7. Modbus RTU RS232

Protokolli Modbus përcakton disa funksione kodesh për të hyrë në regjistrat Modbus.

Function Code	Register Type
1	Read Coil
2	Read Discrete Input
3	Read Holding Registers
4	Read Input Registers
5	Write Single Coil
6	Write Single Holding Register
15	Write Multiple Coils
16	Write Multiple Holding Registers

*** Nëse jeni duke përdorur pajisje RS-232, ju mund të keni vetëm dy pajisje gjithsej, pavarësisht se si janë konfiguruar.

6 Ndërfaqja grafike në PC (GUI)

Të dhënat e lexuara nga sensori do të i dërgojmë edhe në PC për analizim. Për këtë arsye është zhvilluar një ndërfaqe grafike në gjuhën programuese C# së bashku me kornizën .NET (WinForms).

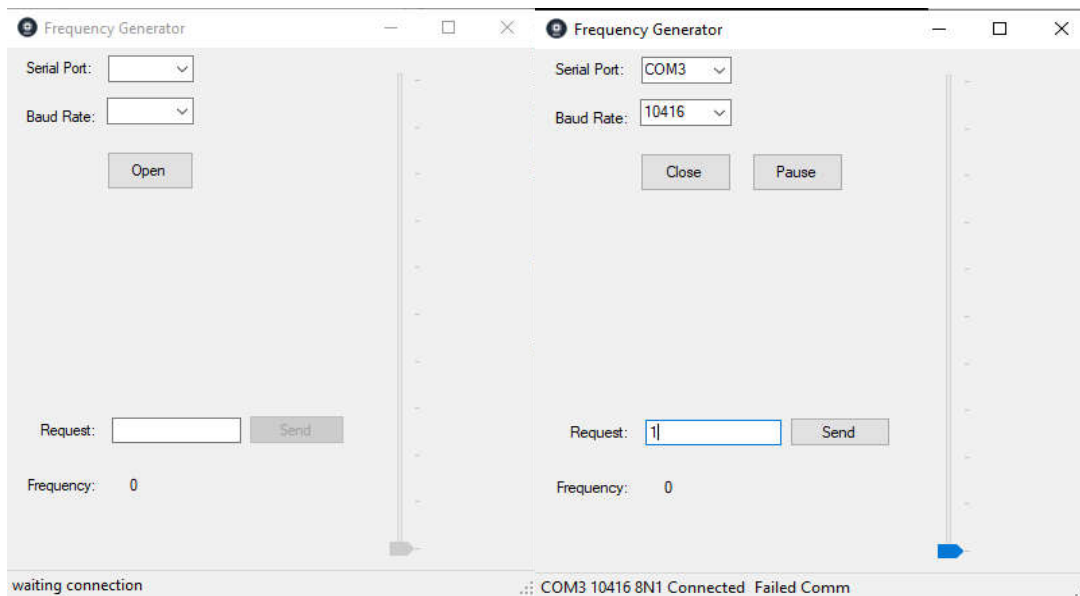


Figura 8. Pamje e faqës së ndërfaqës në PC

Nga këtu shihet se pasi të realizohet lidhja me mikrokontrollerin ,ne mund të zgjedhim portin dhe Baud Rate (10416) dhe pasi të dërgojmë kërkesën në HZ, KHZ ose MHZ rezultati do të shihet në osciloskop.

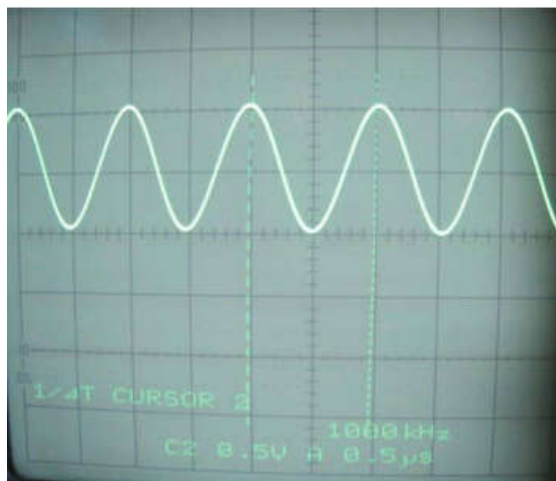
Baud Rate tregon numrin e bitave që porti serik mundet ti transmetojë brenda 1 sekondi.

Për të shfaqur pamjen në GUI duhet të përcjellni shtegun si në vijim:

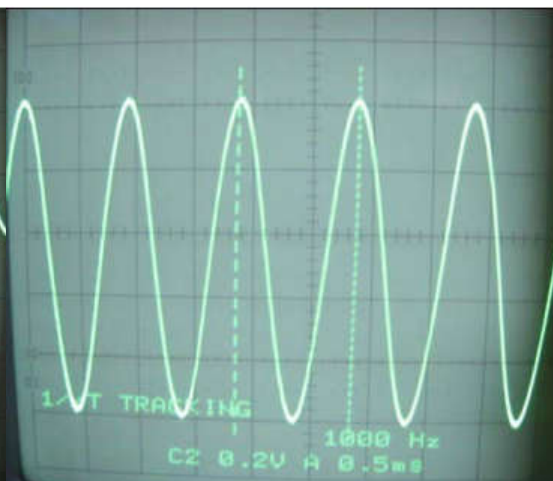
11P_2018-2019/11P AD9850/11P AD9850/obj/Debug/CuurentMeter.

Kjo do t'ua mundësojë që të shihni aplikacionin edhe nëse nuk e keni Visual Studio-n të instaluar.

1 MHZ:



1 KHZ:



1MHZ:

1KHZ:

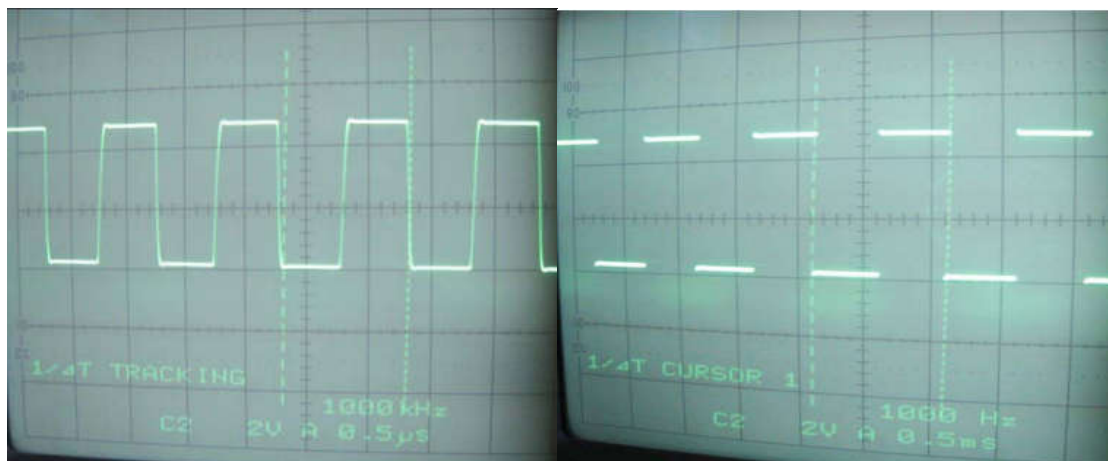


Figura 9. Pamje ne osciloskop

7 Nderfaqja njeri-makine

Ndërfaqja ka 4 butona të cilin e lejon përdoruesin që të zgjedh modin e punës së sistemit, rregullimin e parametrave ose për kërkesë të shfaqjeve. Butonat janë të emëruara ESC, OK, UP dhe DOWN. Butoni OK e dërgon Menyne në një hapë përpara, butoni ESC e kthen një hap mbrapa ndërsa UP dhe DOWN levizin poshtë dhe lartë.

BUTONI	Funksioni
P2.0	UP
P2.1	ESCAPE
P2.2	OK
P2.3	DOWN

8 Referencat

I.Scott MacKenzie and Raphael C. -W. Phan.The 8051 Microcontroller, Fourth Edition.

<https://www.analog.com/media/en/technical-documentation/data-sheets/AD9850.pdf>

<https://www.schneider-electric.com/en/faqs/FA168406/>

https://www.csimn.com/CSI_pages/Modbus101.html?fbclid=IwAR09Ck9Z2COPdotm7-Uy4-qf0wKd2lpoL3cNXe21hFIPmKDWVZTCjBfTrCQ

9 Kodi ne Assembler

```
.....
;DESCRIPTION
;=====DESCRIPTION END=====
;initialization
;-----
;
.....
display0      equ 030h;    register for display 0
display1      equ 031h;    register for display 1
display2      equ 032h;    register for display 2
display3      equ 033h;    register for display 3

nestsamples   equ 034h;    sample time divider
nestednum      equ 5;      sample time constant

nestsamples   equ 035h;    sample time divider
nestednum      equ 40;      sample time constant

proc0_stat     equ 036h;    process state
proc1_stat     equ 037h;    process state

previousbuttonport equ 038h; saving previous scan of keyboards (for negative edge
detection)
shownumH       equ 039h;    data HIGH for displaying
shownumL       equ 3ah;     data LOW for displaying

timer3p5       equ 03bh;    waiting 3.5 character for modbus
start3p5       equ 000h; BIT start 3.5 calculating time

menudone       equ 18h; bit  set when user made all the requests on meny
reqdown        equ 19h; bit  send a request to lower the frequency
requp          equ 1ah; bit  send a request to raise the freq
limup          equ 1bh; bit  limit raising frequency
limdown        equ 1ch; bit  limit lowering frequency

data3          equ 40h;     data sent to device for setting frequency, MSB
data2          equ 41h;     data sent to device for setting frequency, seond byte
data1          equ 42h;     data sent to device for setting frequency, third byte
data0          equ 43h;     data sent to device for setting frequency, LSB

calculateforreading equ 001h; MODBUS parameter to calculate CRC for reading
or writing buffer
writingbuffercount equ 03ch; MODBUS save number of bytes to send (reply)
;Constants
```

```
SlaveID      equ 1;          MODBUS SlaveID
tim3p5       equ 40h;        MODBUS 3.5character constant
```

```
CoilsNumber  equ 64;          MODBUS number of coils
CoilsStartAdd equ 40h;        MODBUS address of first coil
```

```
HoldingRegStartAdd equ 30H; MODBUS start of first register
HoldingRegNumber   equ 10H; MODBUS number of registers
```

```
StartofReadingBuffer equ 128; MODBUS start of readinf buffer
StartofWritingBuffer  equ 192; MODBUS start of writing buffer
```

```
;for device
w_clk      equ p1.4;    DEVICE CLOCK used to load 8bit data
fq_ud      equ p1.5;    DEVICE FREQUENCY UPDATE loaded datas to output
resett      equ p1.6;    DEVICE RESET
```

```
;constants compatible with 7-segment for showing characters
Mhigh equ 00110011b
Mlow  equ 00100111b
Ebig  equ 01111001b
Nbig  equ 00110111b
Xbig  equ 01110110b
Ybig  equ 01100110b
Zbig  equ 01011011b
```

```
=====
Buttonport equ p2
=====INITIALIZATION END=====
```

```
.....
; PROGRAM INTERRUPT ORGANIZATION
;-----
```

```
.....
;reset interup
ljmp main
.....
;timer 0 isr
ljmp t0isr
.....
;serial port isr
ljmp spisr
=====INTERRUPT ORGANIZARION END=====
```

```
.....
;Main Code here
.....
org 030h
```

```
.....  
,,,,,,,,,,,,,,
```

```
.....  
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```

```
=====CLEAN RAM END=====
```

```
.....  
,,,,,,,,,,,,,,
```

```
*****  
,,,,,,,,,,,,,,
```

```
;=====CODE INITIALIZATION END=====
```

:DEVICE RESETING SEQUENCE

```

mov p0, #0
anl p1, #00001111b

```

```

;clr resett
setb resett
clr resett

```

```

;clr w_clk
setb w_clk
clr w_clk

```

```

;clr fq_ud
setb fq_ud
clr fq_ud

```

```

startitall:
jnb menudone, startitall;    wait for user to select meny
mov r2, shownumL;           pass parameter to show on 7seg display low
mov r1, shownumH;           pass parameter to show on 7seg display high
Lcall HextoDisplay;         do the math

jmp startitall;             loop

```

```

HexToDisplay:
lcall HexToBcd;             16-bit (r2-LOW, r1-HIGHT) to r3-ones, r4-tenths, r5-
hundreds, r6-thousands, r7-tenthousands,

```

```

mov a, r3;                  ones
lcall nto7seg;              make ones 7-segment compatible number
mov display0,a;             move data to show at display0

```

```

mov a, r4;                  tenths
lcall nto7seg;              make tens 7-segment compatible number
mov display1,a;             move data to show at display1

```

```

mov a, r5;                  hundreds
lcall nto7seg;              make hundreds 7-segment compatible number
mov display2,a;             move data to show at display2

```

```

mov a, r6;                  thousands
lcall nto7seg;              make thousands 7-segment compatible number
mov display3,a;             move data to show at display3

```

```

ret

```

```

;=====MAIN CODE END=====

```

```

.....
; timer 0 interrupt service routine
;-----

```


[illegible]

```

        clr ri;
        mov @r0, sbuf;           save data read at serial port buffer to reading buffer
        inc r0;                  increment pointer
        setb start3p5;           start 3.5character timer
        pop psw;                 replace Bank previously activated
        reti

transmit:
        clr ti;
        djnz writingbuffercount, continue; count if not end of sending frame
        jmp terminate
continue:
        mov sbuf, @r1;           write data at serial port buffer for transmission
        inc r1;                  increment buffer for next time
terminate:
        pop psw
        reti

;=====SERIAL COMM INTERRUPT END=====

;-----
;DIAPLY
;-----
;source- WUS script-Prof. Lavdim Kurtaj
;Routine that displays on 4 7-segments
;the datas on ram meories 30h - 33h
;-----
display:
        push acc;                save accumulator

        mov a,proc0_stat;        check process state
        rl a
        rl a
        mov dptr,#tabproc0 ;    lookup table address
        anl p1,#11000000b;      turnoff all displays
        jmp @a+dptr
tabproc0:
        ljmp p0_s00
        nop
        ljmp p0_s01
        nop
        ljmp p0_s02
        nop
        ljmp p0_s03
        nop

p0_s00::
        mov p0,display0
        setb p1.0
        mov proc0_stat,#1
        ljmp p0_end

```

```

p0_s01;;          display at 1 and set next time to process display 2
    mov p0,display1
    setb p1.1
    mov proc0_stat,#2
    ljmp p0_end

```

```

p0_s02;;          display at 2 and set next time to process display 3
    mov p0,display2
    setb p1.2
    mov proc0_stat,#3
    ljmp p0_end

```

```

p0_s03;;          display at 3 and set next time to process display 0
    mov p0,display3
    setb p1.3
    mov proc0_stat,#0
    ljmp p0_end

```

```

p0_end:
    pop acc; return accumulator
    ret

```

```

;===== END=====

```

```

.....
;WriteDevice

```

```

;routine that sends data3-0 to AD9850
;uses P0 as 8bit parallel data transmission mode
; and P1.4, P1.5 P1.6 as signalling pins
;-----

```

```

writedevice:
    mov p1, #0;      turn all displays off and device lines

    mov p0, #0;      send phase 0 and turnon sequence
    setb w_clk;      clock
    clr w_clk

    mov p0, data3;    send LSB
    setb w_clk;      clock
    clr w_clk

    mov p0, data2;    send second byte
    setb w_clk;      clock
    clr w_clk

    mov p0, data1;    send third byte
    setb w_clk;      clock
    clr w_clk

```

```

mov p0, data0;    send MSB
setb w_clk;       clock
clr w_clk

setb fq_ud;       update frequency (output it)
clr fq_ud

ret

```

```

.....
;HEXIMAL TO BCD
;-----
;source
;http://www.iuma.ulpgc.es/~nunez/clases-sed-mai-8051/programas/16bit-t0-bcd.asm
;value in registers R1 and R2 will be
;turned to binary-coded-decimal in
;register R3 throughg R7
.....

```

HextoBCD:

push acc

```

MOV    R3,#00D
MOV    R4,#00D
MOV    R5,#00D
MOV    R6,#00D
MOV    R7,#00D

    MOV    B,#10D
MOV    A,R2
DIV    AB
MOV    R3,B        ;Resto en R3
MOV    B,#10        ; R7,R6,R5,R4,R3
DIV    AB
MOV    R4,B        ;Resto en R4
MOV    R5,A
CJNE   R1,#0H,HIGH_BYTE ; CHECK FOR HIGH BYTE
SJMP   ENDD

```

HIGH_BYTE:

```

    MOV    A,#6
ADD    A,R3
    MOV    B,#10
DIV    AB
MOV    R3,B
    ADD    A,#5
ADD    A,R4
    MOV    B,#10
DIV    AB
MOV    R4,B
    ADD    A,#2

```


push acc

mov a, buttonport; save buttons state
cpl a; complement, easy8051a uses click to low
push acc; save data
;cpl a
anl a, #00001111b ; Assign only lower half for neg edg
xrl a, previousbuttonport;check which ones have changed
pop b; return buttons state
anl a, b; accept only the ones that have negative edge
mov previousbuttonport, b;save the data for next scan
jz ended

jnb acc.0, nochangein0
;change in pin 0 code UP
jb menudone, requpp
mov a , proc1_stat
inc a
cjne a, #4, gottogo
jmp ended
gottogo:
inc proc1_stat
jmp ended
requpp:
setb requp
jmp ended

nochangein0:

jnb acc.1, nochangein1
;change in pin 1 code ESC

jb menudone, justplain
mov proc1_stat, #0
justplain:
clr limup
clr menudone
mov data0, #0
mov data1, #0
mov data2, #0
mov data3, #0
mov shownumL, #0
mov shownumh, #0
jmp ended

nochangein1:

jnb acc.2, nochangein2
;change in pin 2 code OK

jb menudone, ended
mov a , proc1_stat
jz makeitone

```

        setb menudone
        jmp ended
makeitone:
        mov proc1_stat, #1
        jmp ended
nochangein2:
        jnb acc.3, nochangein3
        ;change in pin 3 code DOWN
        jb menudone, reqdownn
        mov a , proc1_stat
        dec a
        jz ended
        mov proc1_stat, a
        jmp ended
reqdownn:
        setb reqdown
        jmp ended
nochangein3:
        jnb acc.4, nochangein4
        ;change in pin 4 code

        jmp ended
nochangein4:
        jnb acc.5, nochangein5
        ;change in pin 5 code

        jmp ended
nochangein5:
        jnb acc.6, nochangein6
        ;change in pin 6 code

        jmp ended
nochangein6:
        jnb acc.7, ended
        ;change in pin 7 code

        jmp ended

ended:
        pop acc
        ret
;===== END=====
;-----
;
;
;
;
;
;
;
;
;
```

MODBUS:

```
mov psw, #00001000b
```

```
mov a, @r0
cjne a, 14, failedCheckk; Check CRCHigh
inc r0
mov a, @r0
cjne a, 15, failedCheckk; Check CRCLow
```

```

mov r0, #StartOfReadingBuffer
MOV R1, #StartOfWritingBuffer
mov a, @r0
inc r0
cjne a, #slaveID, failedcheckk;Check slaveID
mov @r1, a
inc r1
;This device is addressed by modbus
mov a, @r0;GetFunctionCode
inc r0
cjne a, #15, noFC15WriteMultipleCoils
.....
1111111111111111

```

```
cjne a, #16, noFC16WriteMultipleRegisters
```

```

jc AddressPassFC16
mov r1, #StartOfWritingBuffer+1
mov @r1, #90H
inc r1
mov @r1, #02
inc r1
ljmp Preparetransmit

```


AddressPassFC16:

```
mov r1, #STARTOFWRITINGBUFFER+ 1
mov a, #16
MOV @r1, a
inc r1
mov a, #0
mov @r1, a
inc r1
```

```
mov r0, #StartOfreadingBuffer+3
;mov r0, a
mov a, @r0
mov @r1, a
inc r1
```

```
rl a
add a, #HoldingRegStartAdd
;mov r0, a; Start of registers requested to WRITE
push acc
```

```
mov a, #0
mov @r1, a
inc r1
```

```
mov r0, #StartOfreadingBuffer+5
mov a, @r0
mov @r1, a
inc r1
```

```
mov r0, #startofreadingbuffer+6
mov a, @r0
mov r2, a
MOV R1, #STARTOFREADINGBUFFER+7
```

```
pop 08
```

LoopitFC16:

```
mov a, @r1
mov @r0, a
inc r0
inc r1
djnz 10, LoopitFC16
MOV R1, #STARTOFwritingBUFFER+6
ljmp Preparetransmit
```

noFC16WriteMultipleRegisters:

```
dec a
rl a
rl a
mov dptr, #tabFC
jmp @a+dptr
```

tabFC:

```
ljmp FC01_ReadCoils
nop
ljmp FC02_ReadDiscreteInputs
nop
ljmp FC03_ReadHoldingRegisters
nop
ljmp FC04_ReadInputRegisters
nop
ljmp FC05_WriteSingleCoil
nop
ljmp FC06_WriteSingleRegister
nop
```

```
failedcheckk:
ljmp failedcheck
```

FC01_ReadCoils:

```
cjne @r0, #0, sendexpectioncode
inc r0
inc r0
cjne @r0, #0, sendexpectioncode
dec r0
mov a, @r0
inc r0
inc r0
cjne a,#CoilsNumber, notCoilatend; Check address
```

```
cjne @r0, #1, sendExpectioncode; Reading Last Bit and only one
ljmp allrightnow
```

notcoilatend:

```
jc lessthanendcoil
```

```
sendexpectioncode::Addres above allowed, and/or number of coils above allowed
mov @r1, #81h
inc r1
mov @r1, #2
inc r1
ljmp preparetransmit
```

lessthanendcoil:

```
dec r0
dec r0
mov a, @r0
inc r0
inc r0
add a, @r0
cjne a, #CoilsNumber, NotalltheCoils; make sure doent overflow the address
```

```

sjmp allrightnow
notAllthecoils:
jnc sendexpectationcode
allrightnow:
dec r0
dec r0
mov a, @r0
mov b, #8
div ab
add a, #CoilsStartAdd/8; we got the addres of first register to send
add a, #20
push acc;save the addres of first reg
mov r0, #StartOfReadingBuffer + 4; number of bits
mov a, @r0
mov b, #8
div ab

```

```

mov r2, a; Loop number of registers
inc b; mask; number of bits at the end

```

```

mov a, #0ffh
shiftloop:
dijnz b, shiftit
sjmp endshifting
Shiftit:
clr c
rrc a
sjmp shiftloop
endshifting:
mov r3, a;save mask
mov r1, #StartofWritingBuffer + 1;response
mov @r1, #1;response fc code
inc r1
inc r2
mov a, r2
mov @r1, a
inc r1
pop 08

```

```

loopwriting:

```

```

mov a, @r0
inc r0
mov @r1, a
inc r1
dijnz r2, loopwriting

```

```

dec r1

```

```

mov a, @r0
anl a, r3

```

```
mov @r1, a
inc r1
```

```
ljmp Preparetransmit
```

FC02_ReadDiscreteInputs:

```
ljmp Preparetransmit
```

FC03_ReadHoldingRegisters:

```
lcall AddressLavidation
```

```
jc AddressPass
mov r1, #StartOfWritingBuffer+1
mov @r1, #83h
inc r1
mov @r1, #02
inc r1
ljmp Preparetransmit
```

```
AddressPass:
mov r0, #StartOfreadingBuffer+3
;mov r0, a
mov a, @r0
rl a
add a, #HoldingRegStartAdd
;mov r0, a; Start of registers requested to read
push acc
```

```
mov r0, #StartOfreadingBuffer+5
mov a, @r0
rl a
mov r2, a
```

```
mov r1, #StartOfWritingBuffer+1
mov @r1, #03
inc r1
mov @r1, a
inc r1
```

```
pop 08
```

LoopitFC3:

```
mov a, @r0
mov @r1, a
inc r0
inc r1
djnz 10, LoopitFC3
```

```
ljmp Preparetransmit
```

FC04_ReadInputRegisters:

ljmp Preparetransmit

FC05_WriteSingleCoil:

```
mov r1, #StartOfWritingBuffer+1
mov @r1, #85h
inc r1
mov @r1, #01
inc r1
ljmp Preparetransmit
```

FC06_WriteSingleRegister:

```
mov r0, #StartOfReadingBuffer+2
cjne @r0, #0, sendexpectationFC6
sjmp addresspassFC6
sendexpectationFC6:
mov r1, #StartOfWritingBuffer+1
mov @r1, #86h
inc r1
mov @r1, #02
inc r1
ljmp Preparetransmit
```

addresspassFC6:

cjne @r0, #HoldingRegNumber, notthelastregFC6

notthelastregFC6:

```
jnc sendexpectationFC6
mov r1, #StartOfWritingBuffer+1
mov @r1, #06h
inc r1
mov @r1, #0
inc r1
mov r0, #StartOfreadingBuffer+3
;mov r0, a
mov a, @r0

mov @r1, a
inc r1

rl a
add a, #HoldingRegStartAdd
;mov r0, a; Start of registers requested to read
push acc

mov r0, #StartOfreadingBuffer+4
mov a, @r0
```

```

mov @r1, a
inc r1

pop 08
mov @r0, a

inc 08
push 08

mov r0, #StartOfreadingBuffer+5
mov a, @r0

mov @r1, a
inc r1

pop 08
mov @r0, a

ljmp Preparetransmit

```

Preparetransmit:

```

push 09
mov r2, #startofwritingbuffer
mov r3, 09
clr calculateforreading
lcall crc16
pop 09
mov a, r6
mov @ r1, a
inc r1
mov a, r7
mov @r1, a
inc r1
mov a, r1
inc a
clr c
subb a, #Startofwritingbuffer
mov writingbuffercount, a

```

```

setb ti

```

failedcheck:

```

mov r1, #startofwritingbuffer
mov r0, #startofreadingbuffer
pop psw
pop acc
ret

```

```

;===== END =====

```

```

.....
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,

```

```

;CRC16

```

```

;-----

```

CRC16:

```
jnb calculateforreading, forwriting
mov r0, #startofreadingbuffer
sjmp calculateCRC
forwriting:
mov r0, #startofwritingbuffer
calculateCRC:
mov r7, #0ffh
mov r6, #0ffh
```

```

Innerloop:
    clr c

    mov a, r7
    rrc a
    mov r7, a

    mov a, r6
    rrc a
    mov r6, a

    jnc dontpol

```

```

mov a, r7
xrl a, #0a0h
mov r7, a

```

```

mov a, r6
xrl a, #01
mov r6, a

```

```

dontpol:
djnz r4, innerloop
djnz r5, outterloop
ret

```

```

;=====

```

```

.....
;Adress Validation

```

```

;-----

```

```

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

```

```

.....

```

```

AddressLavidation:

```

```

mov r0, #StartOfReadingBuffer+2
cjne @r0, #0 ,failedaddress
inc r0
inc r0
cjne @r0, #0 ,failedaddress
dec r0
cjne @r0, #HoldingREGNumber, notLastAddress
inc r0
inc r0
cjne @r0, #1, failedaddress
CheckGoodaddress:
setb c
ret

```

```

notlastaddress:
jnc failedaddress
mov r0, #StartOfReadingBuffer+3
mov a, @r0
inc r0
inc r0
add a, @r0
cjne a, #HoldingRegNumber, TestOverflow
sjmp CheckGoodAddress

```



```

TestOverflow:
jc CheckGoodAddress
failedaddress:
clr c
ret

```

```

;=====

```

```

LoopP:
    mov a, proc1_stat ;           process state
    rl a
    rl a
    mov dptr,#tabproc1s ;       lookup table address

    jmp @a+dptr
tabproc1s:
    ljmp p1_s00s
    nop
    ljmp p1_s01s
    nop
    ljmp p1_s02s
    nop
    ljmp p1_s03s
    nop

p1_s00s: ;move datas to 7seg to show MENU
    mov display3, #MHigh
    mov display2, #Mlow
    mov display1, #Ebig
    mov display0, #Nbig
    ljmp p1_ends

p1_s01s: ;move datas to 7seg to show HZ
    mov display3, #0
    mov display2, #Xbig
    mov display1, #Zbig
    mov display0, #0
    ljmp p1_ends

p1_s02s: ;move datas to 7seg to show KHZ
    mov display3, #01110100b
    mov display2, #00100000b
    mov display1, #Xbig
    mov display0, #Zbig
    ljmp p1_ends

p1_s03s: ;move datas to 7seg to show MHZ
    mov display3, #Mhigh
    mov display2, #Mlow
    mov display1, #Xbig

```

```

mov display0, #Zbig
ljmp p1_ends

```

```

p1_ends:
    ret

```

```

.....
request: ;process request to increase/decrease frequency

```

```

;UInt32.MAX=0xFFFFFFFF=4294967295
;4294967295->125.000.000Hz frequency;
;34359738=0x20C49BA    ->1.000.000Hz; (1MHz constant)| 02h (MSB) |0Ch (third byte)| 49h
(second byte)| BAh (LSB)
;34359=0x8637          ->1.000Hz; (1KHz constant)|86h (MSB)| 37h (LSB)
;34=0x22              ->1HZ; (1HZ constant)| 22h
    jnb menudone, p1_s00

```

```

    mov a, proc1_stat
    rl a
    rl a
    mov dptr,#tabproc1

```

```

    jmp @a+dptr
tabproc1:
    ljmp p1_s00
    nop
    ljmp p1_s01
    nop
    ljmp p1_s02
    nop
    ljmp p1_s03
    nop

```

```

p1_s00:
    ljmp p1_end1; meny state doesnt use request

```

```

p1_s01: ; HZ state
    jnb requp, notrequp; pass if theres no increase request, look for decrease request
    clr requp;          clear parameters for next use

    inc shownuml;        increase data to show at display
    mov a, shownuml;
    jnz contt;           if overflow
    inc shownumh;        increase MSB too
    contt:

    clr c
    mov a, data0;
    add a, #022h;        add to data0 constant for 1HZ
    mov data0,a;        save it
    jnc notreqdown;     if no overflow pass

```

```

inc data1;           else increase second register
mov a, data1
jnz notreqdown;     case overflow

inc data2;           increase third register
mov a, data2
jnz notreqdown

```

notrequp:

```

jnb reqdown, notreqdown; check for request to decrease frequency
clr reqdown;           clear parameter for next use

dec shownuml;          lower the data to show at 7seg
mov a, shownuml
cjne a, #255, contin; case of underflow
dec shownumh;          lower the high part too
mov a, shownumH
cjne a, #255, contin; case of 2 underflows (high and low) then set to 0
mov shownuml, #0
mov shownumh, #0
jmp notreqdown;       and dont touch device data

```

contin:

```

clr c
mov a, data0;
subb a, #022h;         subbstract LSB the 1Hz constant
mov data0, a
jnc notreqdown;       if no underflow, pass

dec data1;            else subbstract form second register
mov a, data1
cjne a, #255, notreqdown; if underflow

dec data2;            decrease third register too
mov a, data2
cjne a, #255, notreqdown; if 3 underflows then set frequency to 0
    mov data0, #0
    mov data1, #0
    mov data2, #0
notreqdown:
ljmp p1_end1

```

p1_s02: ;REQUEST for KHZ

```

jnb requp, notrequp1; check increase request
clr requp

inc shownuml
mov a, shownuml
jnz contt1

```

inc shownumh
contt1:

clr c
mov a, data0
add a, #37h; increase 1KHz constant LOW
mov data0,a

mov a, data1
addc a, #086h; increase 1KHz constant HIGH
mov data1,a

mov a, data2
addc a, #00h
mov data2,a

jnc notreqdown1

inc data3

notrequp1:
jnb reqdown, notreqdown1;check for decrease request
clr reqdown

dec shownuml;decrease show number
mov a, shownuml
cjne a, #255, contin1
dec shownumh
mov a, shownumH
cjne a, #255, contin1
mov shownuml, #0
mov shownumh, #0
;jmp notreqdown1

contin1:

clr c
mov a, data0
subb a, #37h; decrease 1KHz constant low
mov data0, a

mov a, data1
subb a, #086h; decrease 1KHz constant HIGH
mov data1, a

mov a, data2
subb a, #00h
mov data2, a

jnc notreqdown1

```

dec data3
mov a, data3
cjne a, #255, notreqdown1
    mov data0, #0
    mov data1, #0
    mov data2, #0
    mov data3, #0
notreqdown1:

```

```

ljmp p1_end1

```

p1_s03: ;REQUESTS for 1MHz

```

jnb reup, notrequp2
clr reup
jb limup, notrequp2
inc shownuml
mov a, shownuml
jnz contt2
inc shownumh
contt2:

```

```

clr c
mov a, data0
add a, #0bah;          add 1KHz constant LOW byte
mov data0,a

```

```

    mov a, data1
    addc a, #049h; add 1KHz constant second byte
    mov data1,a

```

```

    mov a, data2
    addc a, #0ch; add 1KHz constant third byte
    mov data2,a

```

```

    mov a, data3
    addc a, #02h; add 1KHz constant MSB
    mov data3,a
    cjne a, #0ffh, notreqdown2

```

```

    setb limup
    mov data3, #0ffh
    mov data2, #0ffh
    mov data1, #0ffh
    mov data0, #0ffh

```

```

notrequp2:
jnb reqdown, notreqdown2
clr reqdown
clr limup

```

```

dec shownuml
mov a, shownuml
cjne a, #255, contin2
dec shownumh
mov a, shownumh
cjne a, #255, contin2
mov shownuml, #0
mov shownumh, #0
;jmp notreqdown2

```

contin2:

```

clr c
mov a, data0
subb a, #0bah;      Subbstract 1MHz constant LSB
mov data0, a

```

```

mov a, data1
subb a, #049h;      Subbstract 1MHz constant second byte
mov data1, a

```

```

mov a, data2
subb a, #0ch; Subbstract 1MHz constant third byte
mov data2, a

```

```

mov a, data3
subb a, #02h; Subbstract 1MHz constant MSB
mov data3, a

```

```

jnc notreqdown2
mov data0, #0
mov data1, #0
mov data2, #0
mov data3, #0
notreqdown2:

```

ljmp p1_end1

```

p1_end1:
clr requp
clr reqdown
ret

```

END