Caiet de Practică realizat în cadrul Companiei Vitesco

Student : Mucedu Georgeta-Iuliana

Grupa : 1306A

Perioada de desfășurare : 28.06.2021 - 16.07.2021

Total nr. ore : 90

Nota finală : 8 (există un mail cu situația finală la secretariat)

Săptămâna 1

Ziua 1 (28.06.2021 - luni) - Prezentare companie + traineri
- Recapitulare noțiuni fundamentale C
- Recapitulare noțiuni microcontroller

Ziua 2 (29.06.2021 - marți) - Instalare Microchip (Atmel Studio)
- Instalare Proteus
- Prezentare interfețe programe
- Realizarea unui schematic în Proteus

Ziua 3 (30.06.2021 - miercuri) - Prezentarea plăcii (atmega 32)
- Introducere Porturi
- Introducere Leduri
- Introducere Butoane
- Exerciții

Ziua 4 (1.07.2021 - joi) - Discutare soluții propuse pentru exerciții
- Continuare cu exerciții pentru aprofundare

Ziua 5 (2.07.2021 - vineri) - Verificare soluții
- Display LCD
- Exerciții

Săptămâna 2

Ziua 1 (5.07.2021 - luni) - Verificare soluții
- Recapitulare
- ADC (Convertor Analog-Digital)

Ziua 2 (6.07.2021 - marți) - Verificare ADC
- Aprofundare cunoștințe

Ziua 3 (7.07.2021 - miercuri) - Timers
- Exerciții

Ziua 4 (8.07.2021 - joi) - 4 Digit, 7 Segment Display
- Exerciții

Ziua 5 (9.07.2021 - vineri) - Recapitulare
- Verificare soluții propuse
- Introducere în PWM (Pulse Width Modulation)

Săptămâna 3

Ziua 1 (12.07.2021 - luni) - PWM
- Exerciții

Ziua 2 (13.07.2021 - marți) - Prezentare proiect final
- Discuție pe baza cerințelor

Ziua 3 (14.07.2021 - miercuri) - Lucru individual proiect

Ziua 4 (15.07.2021 - joi) - Lucru individual proiect +
- Prezentări proiecte

Ziua 5 (16.07.2021 - vineri) - Prezentări proiecte

R1 221
R2 221
R3 221
R4 221
R5 221
R6 221
R7 221
R8 221
D1 LED-RED
D2 LED-GREEN
D3 LED-RED
D4 LED-GREEN
D5 LED-RED
D6 LED-GREEN
D7 LED-RED
D8 LED-GREEN
DSW1 DIPSW_8
R9 221
R10 221
R11 221
D9 LED-RED
D10 LED-GREEN
D11 LED-BLUE
DSW2 DIPSW_3
DSW7 DIPSW_3
R28 100
U2 DS18B20
VCC
DQ
GND
27.0
LCD1 LM016L
VSS VDD VEE RS RW E D0 D1 D2 D3 D4 D5 D6 D7
DSW3 DIPSW_8
BUZ1 BUZZER
R31 1k
Q5 NPN
DSW8 DIPSW_3
R29 270
R30 270
RV1 100
RV2 100
ABCDEFG DP
1234
R24 10k
R25 10k
R26 10k
R27 10k
R1 221
R2 221
R3 221
R4 221
R5 221
R6 221
R7 221
R8 221
Q3 PNP
Q4 PNP
Q2 PNP
Q1 PNP
R15 221
R14 221
R13 221
R12 221
U1 ATMEGA32
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
DSW5 DIPSW_4
DSW4 DIPSW_8
DSW6 DIPSW_8

C SHORT BRIEF

program:

- consists of a sequence of functions which mostly are placed in different modules

module:

- at least one source file (*.c file) – it may contain a header file if it is needed
- a group of functions and data.
- large program can be divided into a collection of separately compiled modules.

function:

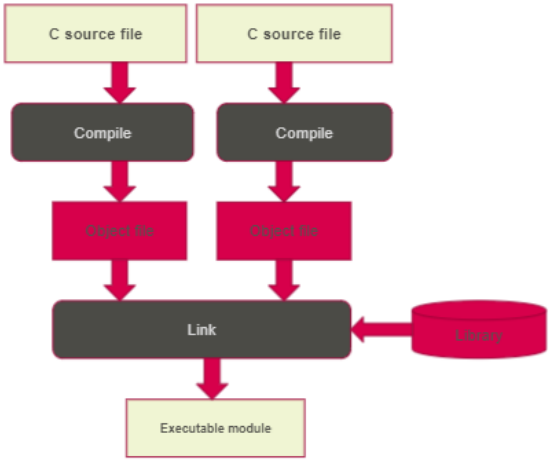
- consists of a function header and a function block
- The function header contains the function name and the function parameters.
- The function block can contain a definition part and a statement part

block structure:

- The curly brackets **{** and **}** are used for combining definitions and statements to one block.
- Definitions and statements are terminated by a "**;**".
- A single "**;**" represents a statement
- In a function block further blocks can be nested

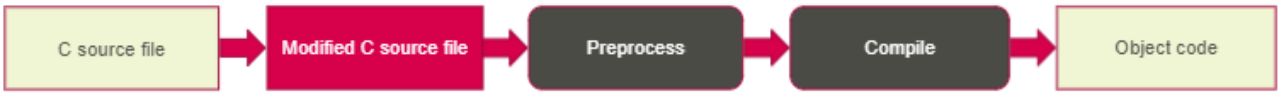
OVERVIEW OF C PROGRAMMING

- A C compiler ***independently processes*** each source file and ***translates*** the C program text into instructions understood by the computer
- The output of the compiler is usually called ***object code*** or an ***object module***
- When all source files are compiled, the object modules are given to a program called the ***linker***
- The linker ***resolves references*** between the modules, ***adds functions*** from the standard run-time library
- The linker produces ***a single executable program*** which can then be invoked or run



THE C PREPROCESSOR – OVERVIEW

The C preprocessor is a simple macro processor that conceptually processes the source text of a C program **before** the compiler proper reads the source program.



To use any preprocessor directives, first, we must prefix them with pound symbol #.

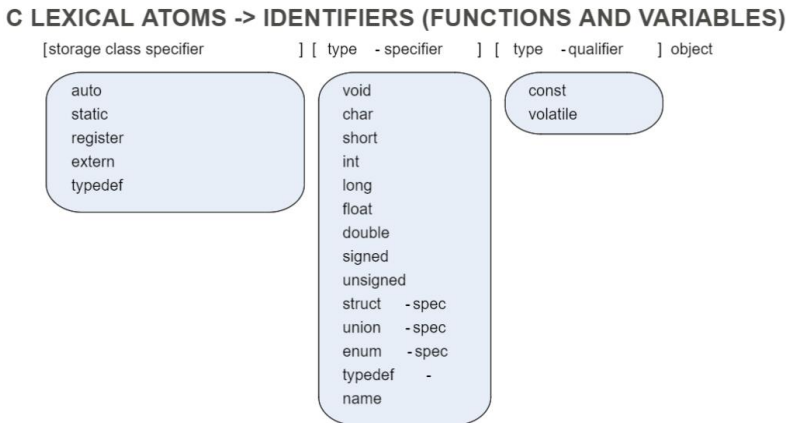
C MACROS

A macro is a segment of code which is replaced by the value of macro.

Macro is defined by #define directive.

There are two types of macros:

- An object like macro takes no arguments. It is invoked by mentioning its name
- A function like macro declares the names of formal parameters within parentheses separated by commas



auto:

- default storage class
- local lifetime
- visible only in the block in which it is declared
- is not initialized automatically
- must be initialized explicitly

static:

- global lifetime
- visible only within the block in which it is declared
- all module – global variables must be declared as static
- initialized to 0 by default

register:

- used for optimizations for heavily used variables
- the variable is assigned to a high-speed CPU register (rather than an ordinary memory location), if possible
- limitation: you cannot generate pointers to them using the & operator
- mostly obsolete, modern compilers do an excellent job without such "hints"

extern:

- reference to a variable with the same name defined at the external level in any of the source files of the program
- the identifier declared has its *defining* instance somewhere else
- used to make the external-level variable definition visible within the block
- **is visible only in the block in which it is declared**

volatile <*type*> <*name*>

- must always be read from its original location and is not kept in a register at any time
- compiler optimizations are not available; therefore, it must be used only if necessary

const <*type*> <*name*>

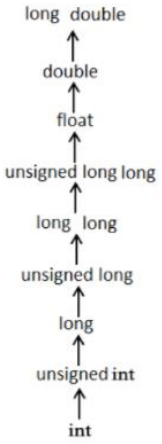
- allows only read access to the variable
- may only be initialized once in a program. The initialization is performed in the startup code
- allows the compiler to perform type checking

Pointers are used in C program to access the memory and manipulate the address of another variable.

- If a **pointer** in C is assigned to **NULL**, it means it is **pointing to nothing**.
- **Normal** variable **stores** the **value** whereas **pointer variable stores** the **address** of the variable.
- ***& symbol*** is used to get the address of the variable.
- **** symbol*** is used to get the value of the variable that the pointer is pointing to.

CAST PRIORITIES

- The **usual arithmetic conversions** are implicitly performed to cast their values in a common type.
- Compiler first performs ***integer promotion***, if operands still have several types, then they are converted to the type that appears highest in the following hierarchy.



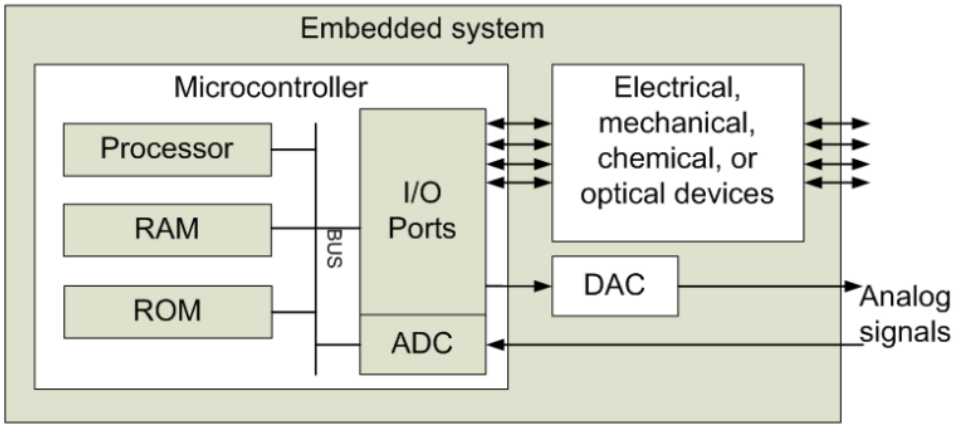
REAL-TIME SYSTEMS

What is a *Real-Time System?*

- the overall correctness of the system depends on both the functional correctness and the timing correctness. The timing correctness is at least as important as the functional correctness

- The system responds in a timely, predictable way to unpredictable external stimuli arrivals
- A ***RTOS*** (Real-Time Operating System)
- Real-time systems must quickly react to external events and process a certain part of code that is associated with the event. This is usually performed using interrupts.
- As soon as an event generates an interrupt, the CPU suspends the current program execution and branches to the appropriate interrupt service routine. The current program status is saved on the system stack.

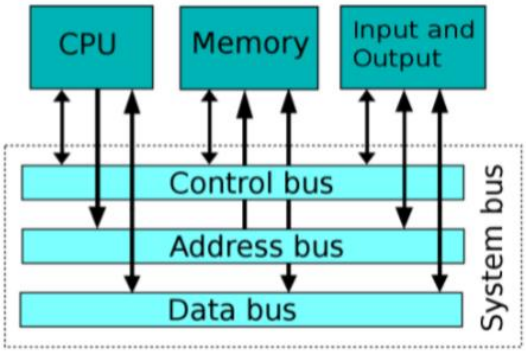
A GENERIC EMBEDED SYSTEM



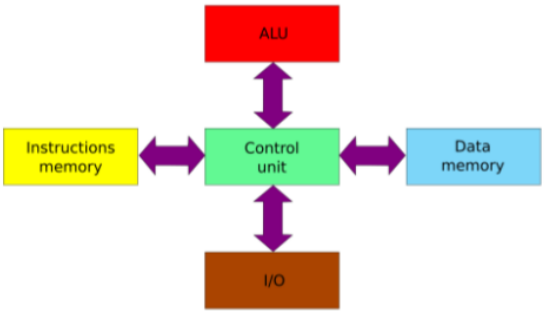
MICROCONTROLLER ARCHITECTURES

The most popular microcontroller / CPU architectures (from internal structure point of view) are:

- von Neumann - Control, address and data bus shared for access to instructions and data



- modified Harvard - Independent instruction and data busses



MICROCONTROLLER ARCHITECTURES

- The most popular microcontroller / CPU architectures (from instruction set point of view) are:
  - CISC (Complex Instruction Set Computer)
    - A single instruction can execute several low-level operations
    - Can use different addressing modes in an instruction
    - Variable instruction length
  - RISC (Reduced Instruction Set Computer)
    - Simple addressing modes, with complex addressing performed via sequences of arithmetic and/or load-store operations
    - Uniform instruction format

CPU

- Central processing unit components
  - Registers & Flags - Registers and flags hold values depending on program flow
  - ALU – Arithmetic Logic Unit – ALU is used to perform arithmetic operations on fixed point values
  - CU – Control Unit - CU coordinates the function of the processor
  - FPU – Floating Point Unit - FPU is used to perform arithmetic operation on floating point values

CPU – INTEGER RANGE

- Integer range
  - Defines the minimum/maximum number a processor register used for arithmetic operations can hold
  - Gives the processor designation – 4bit, 8bit, 64bit, etc.
    - A 32bit processor can hold values in the range 0 - 2^{32}-1 (unsigned)

CPU – OSCILLATOR

- A microcontroller requires a time base (from internal / external oscillator), using either:
  - quartz (frequency accuracy 0.1%) / ceramic resonator (0.5%)
  - RC (5-10%)
  - external clock

- **Instruction cycle is not the same as a clock cycle!**
- Clock rate
    - Number of clock cycles required to execute an instruction – called the machine cycle
    - Maximum frequency with which the CPU can operate

MEMORY

Two types of memory exist (in context of the embedded world)

- Volatile memory – RAM
    - Does not store information if it is not powered
    - Also named "temporary memory"
    - Implemented as RAM (Random Access Memory)
- Non-volatile memory – ROM
    - Holds the memory contents even if not powered
    - Implemented as Flash, EEPROM in modern microcontrollers
    - Other types of NVM exist but are not object of this presentation

TIMERS

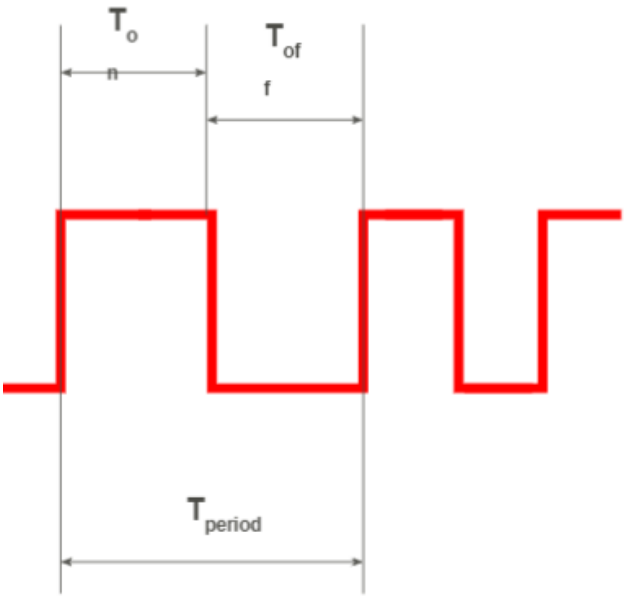
- The registers of Timers are loaded with an input value.
- The value of a Timer register increases/decreases by one after every **machine cycle**.
- When an overflow/underflow occurs, a new value is loaded, and an interrupt may be generated.
- A Timer can be started or stopped on will

PRESCALER

- The prescaler is a HW block preceding the timer to allow longer intervals to be measured.
- The prescaler divides the input signal by a quantity (e.g., 2, 4, 8, 16)
- For example, a prescaler of 1:4 will reduce the input frequency to 1/4 the source.

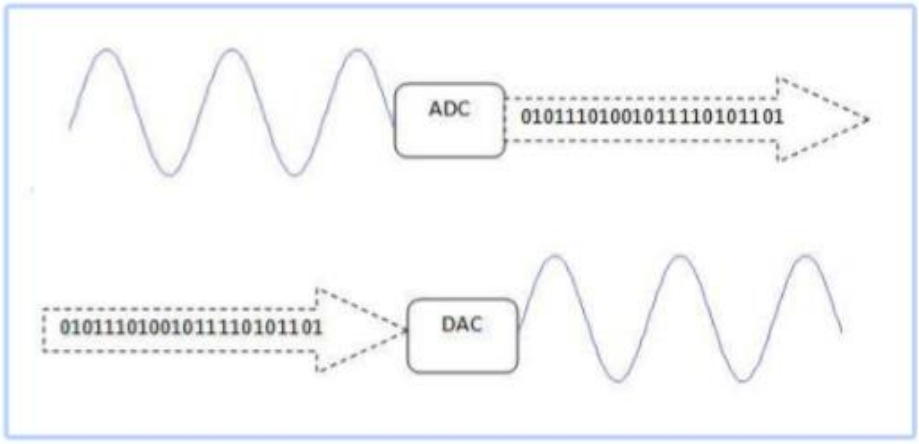
PWM = Pulse Width Modulation

- Pulse Width Ratio [%] = Ton / Tperiod
- What is PWM used for?
- Digital to Analog Conversions
- Electric motor control
- Sine wave generation
- Variable Valve Control



ANALOG SIGNALS ARE CONTINUOUS

- Temperature, pressure, sound intensity, light intensity, voltage is continuous.
- A continuous function would provide an infinite list of numbers → difficult to handle
- Problem: how we could work *digitally* with continuous functions?



ADC

- The ADC (Analog-to-Digital Conversion) is a system that converts an analog signal, such as a sound, into a digital signal.
- Digital information is different from its continuous counterpart in two important respects:
  - it is *sampled*
  - it is *quantized*

Prezentarea plăcii

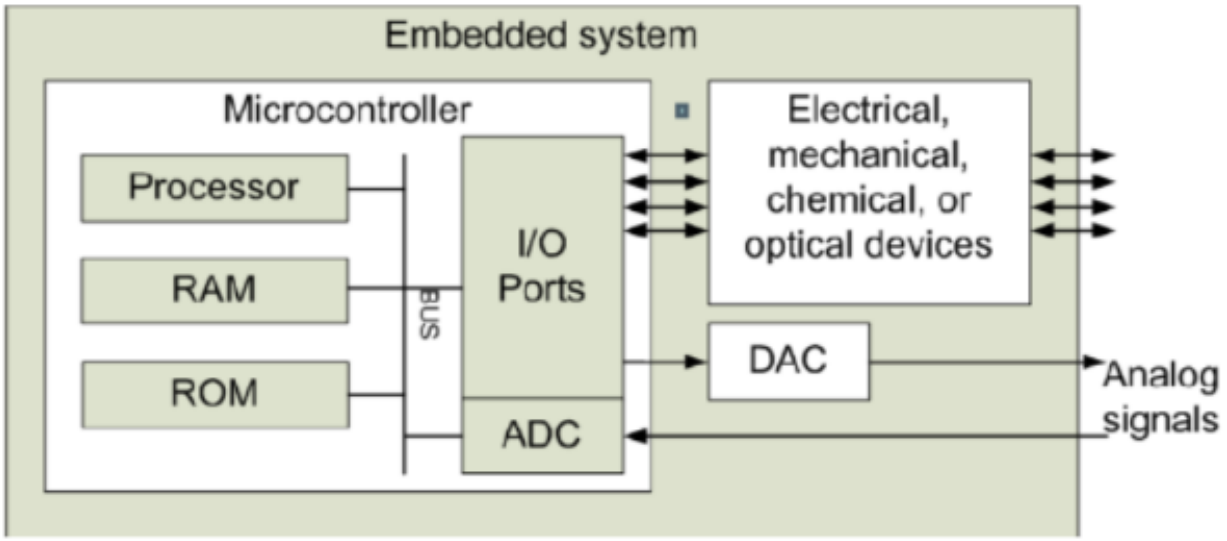
Termeni de bază ai sistemelor embedded

Puterea de procesare – Cantitatea de putere de procesare necesară pentru efectuarea unui anumit task. În general se măsoară în MIPS (milioane de instrucțiuni pe secundă).

Memoria – Memoria necesara (ROM si RAM) pentru a stoca programul și datele utilizate.

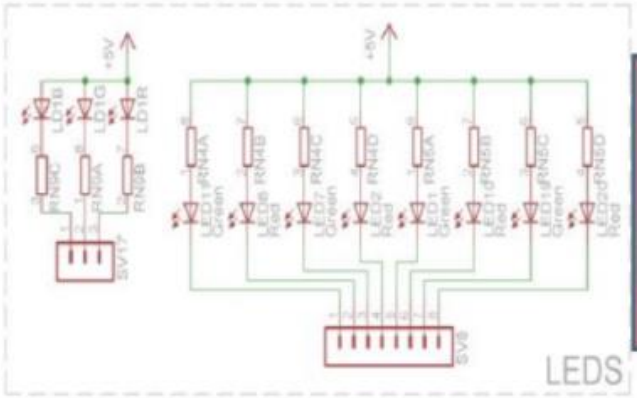
IDE – (Integrated Development Environment) Mediul integrat de dezvoltare.

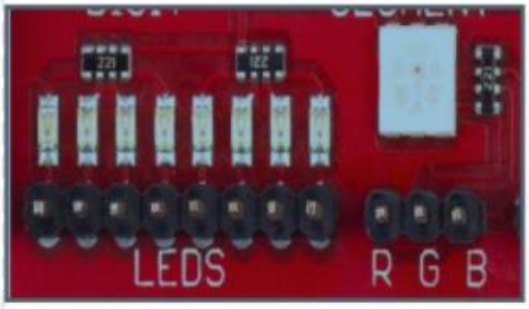
Un sistem embedded este alcătuit dintr-un procesor și un software. Pentru a putea stoca programul avem nevoie de memorie, iar pentru ca programul să poată avea un efect exterior are nevoie de porturi de intrare/ieșire (exemplu: butoane, display-uri etc.). Spre urmare, un sistem embedded va arata în modul următor:



Leduri

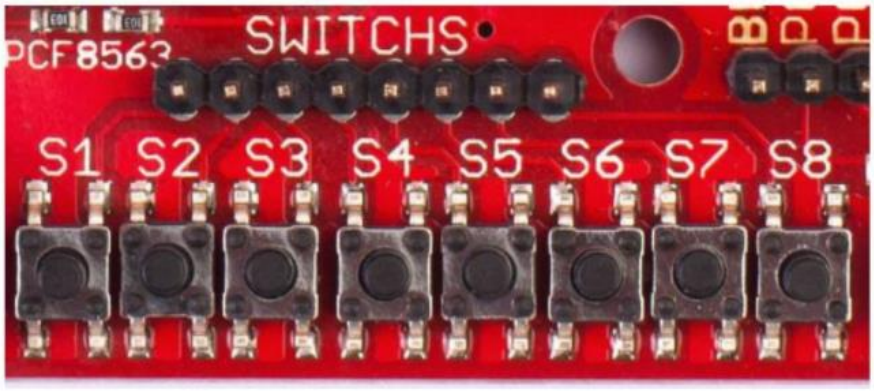
Pentru a aprinde un led situat pe placa va trebui să îi conectăm pinul aferent la

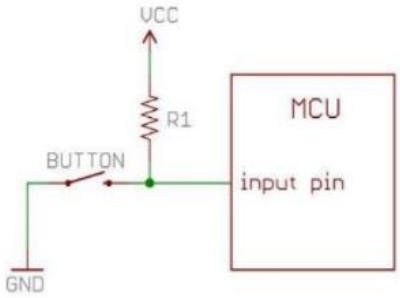


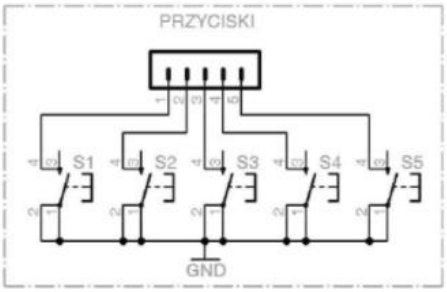


masă (logic 0).

Butoane



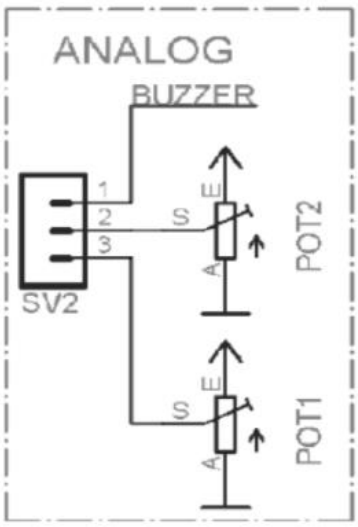


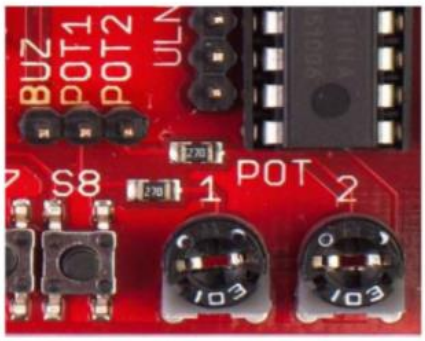


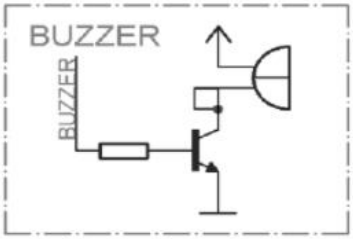
Butoanele, când sunt apăsate, îți conectează pinii la masă. Pentru a detecta o apăsare a butonului trebuie aplicată o rezistență pull-up.

Potențiometru și Buzzer

Pe placa de dezvoltare sunt situate două potentiometre ce pot fi folosite pentru a genera tensiuni în gama 0-5V.







Buzzer-ul poate fi folosit pentru a genera semnale sonore. Acest lucru poate fi realizat prin transmiterea unor semnale PWM către Buzzer sau prin conectarea unuia dintre pinii lui la potențiometru.

Display cu 7 segmente

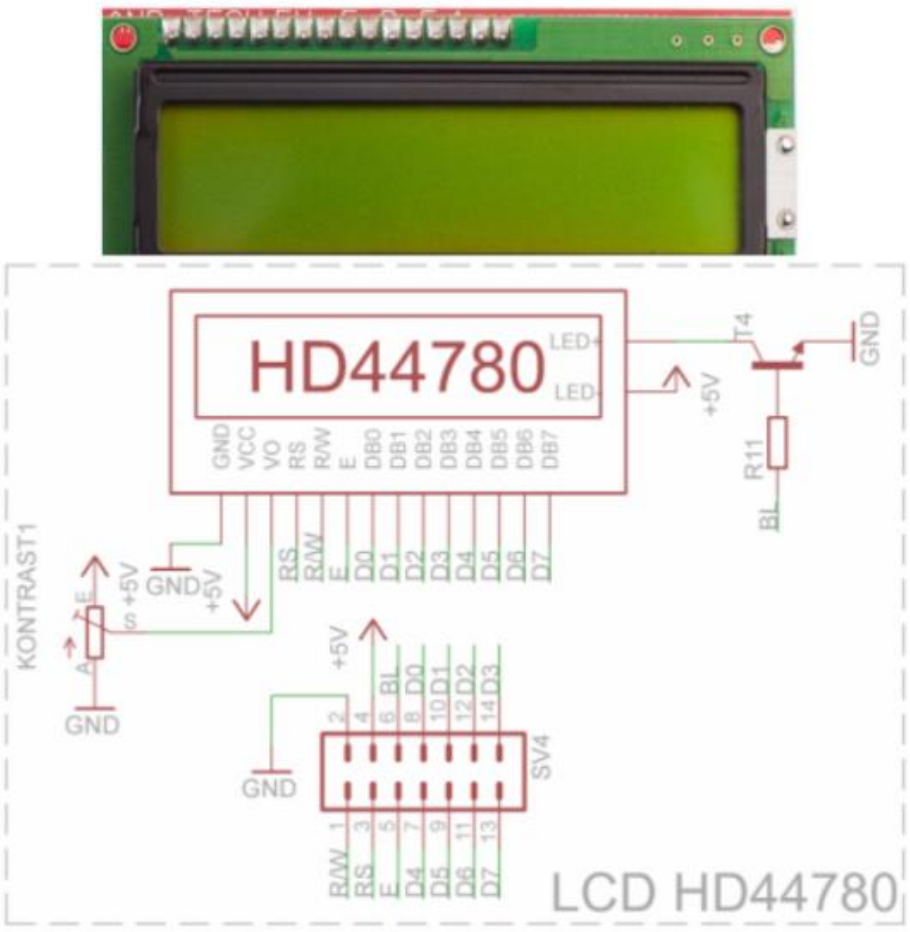
Pe placa sunt prezente 4 display-uri cu 7 segmente. Pentru a aprinde un anumit segment trebuie să:

• Selectăm cifra (display-ul) pe care dorim să afișăm

• Selectăm segmentul dorit

Display LCD

LCD-ul este echipat cu un controler HD44780 și se conectează prin intermediul a 16 pini după cum se observă în imagini. LCD-ul poate fi comandat prin cuvinte de 4 sau 8 biți. Contrastul display-ului poate fi modificat din potentiometru. Jumperul BL activează lumina de fundal (BackLight) a LCD-ului dacă este conectat la 5V.

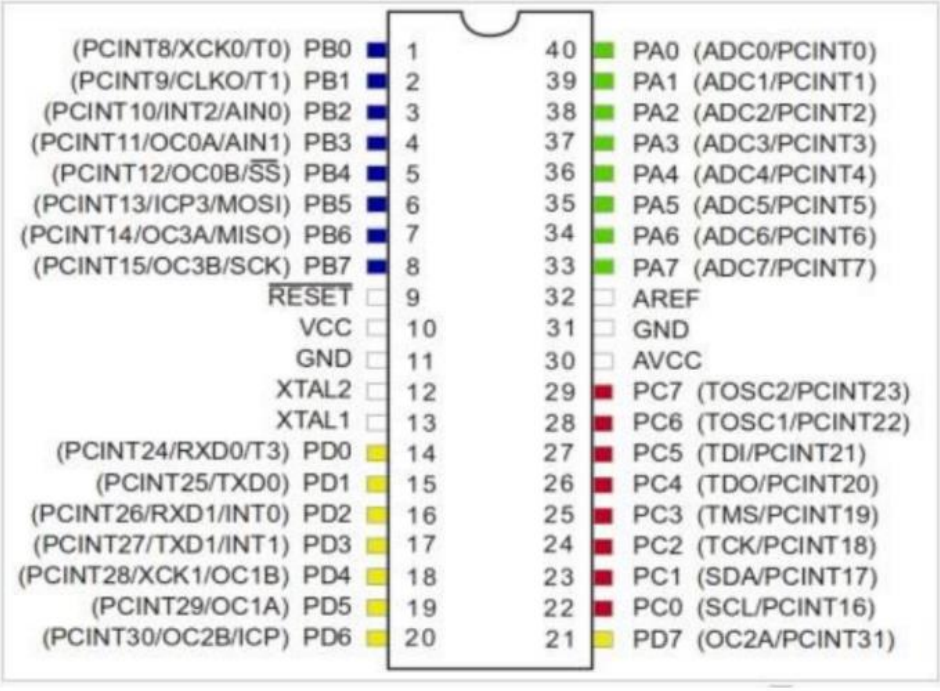


Porturi - Leduri – Butoane

Porturi

Este greu de imaginat un microcontroler care să nu aibă pini de intrare/ieșire întrucâttotul se rezumă la preluarea de date dinexterior, procesarea lor șireturnarea unor rezultate.Spre urmare, e foarte important să știm să configurăm aceste porturi!

Microcontrolerul cu care vom lucra prezintă 4 grupuri(porturi) de câte 8 pini: A, B, C, D.



Cele 4 porturi pot fi configurate prin intermediul a 3 registre:

1. DDRx

2. PORTx

3. PINx, unde x reprezintă denumirea portului(A, B, C, D)

Registrul DDRx

DDRx (Data Direction Register) configurează direcția pe care circulă datele prin port:

- Dacă toți biții din registrul A sunt setați cu valoarea „0” (DDRA = 0x00), atunci toți pinii vor fi de input (primesc date dinspre exterior).
- Dacă toți biții din registrul B sunt setați cu valoarea „1” (DDRB = 0xFF), atunci toți pinii vor fi de output (transmit date spre exterior).

Se folosec pinii de input la apasarea butoanelor, iar pinii de output se folosesc la aprinderea Becurilor.

Registrul PINx

PINx (Port IN) este folosit pentru citirea datelor de pe porturile setate ca intrare (DDRx=0).

Dacă portul este setat ca ieșire, atunci în urmă citirii se va returna valoarea ce a fost transmisă pe acel pin.

PORTx are doua utilități:

1. scrie data pe un anumit pin ca output.

Dacă pinii sunt de ieșire (DDRx=1) atunci se pot da date pe aceștia prin intermediul registrului PORTx. Această scriere va modifica imediat ieșirile pinilor.

2. activează /dezactivează rezistorii pull-up cand porturile sunt de intrare.

Când pinii sunt de intrare (DDRx=0) registrul PORTx este folosit pentru activarea rezistorilor pull-up aferenți acestora.

Pentru activarea rezistorilor PORTx = 1.

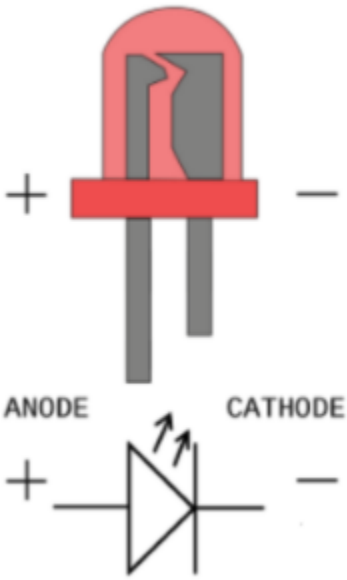
Pentru dezactivarea rezistorilor PORTx = 0.

Leduri

Un Led (light-emitting diode) este o diodă semiconductoare ce emite lumina la polarizarea

directă a joncțiunii p-n..

Un Led este o sursă de lumina mică. De cele mai multe ori acestea sunt utilizate ca indicatori în cadrul dispozitivelor electronice, dar din ce în ce mai mult au început să fie utilizate în aplicații de putere ca surse de iluminare.

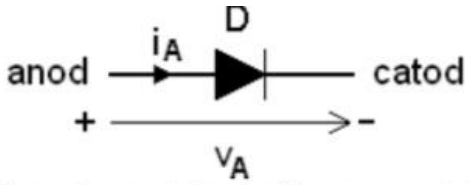


Culoarea luminii emise depinde de compoziția și de starea materialului semiconductor folosit, și poate fi în spectrul infraroșu, vizibil sau ultraviolet.

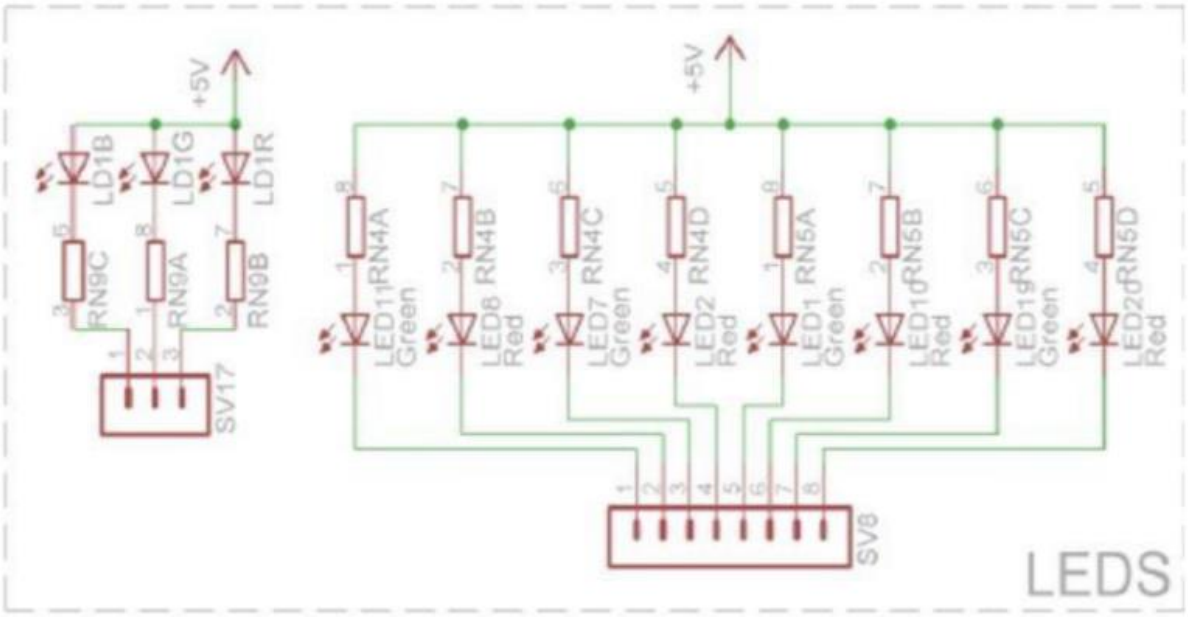
Dioda este o componentă cu 2 terminale ce poate conduce doar într-un singur sens. Aceasta este de rezistență foarte mică (ideal 0) când trece curentul într-o direcție și rezistență foarte mare (ideal infinită) în cealaltă direcție.

Dioda semiconductoare este un dispozitiv electronic constituit dintr-joncțiune p- n prevăzută cu contacte metalice la regiunile p și n și introdusă într-o capsulă din sticlă, metal, ceramic sau plastic. Regiunea p a joncțiunii constituie anodul diodei, iar joncțiunea n, catodul.

Cea mai utilizată funcție a diodei este de a permite trecerea unui curent electric într- o direcție (numit și curent direct al diodei), blocând totodată trecerea curentului în direcția opusă (numit și curent invers al diodei). Acest comportament unidirecțional este numit redresare și este utilizat la convertirea curentului alternativ în curent continuu.



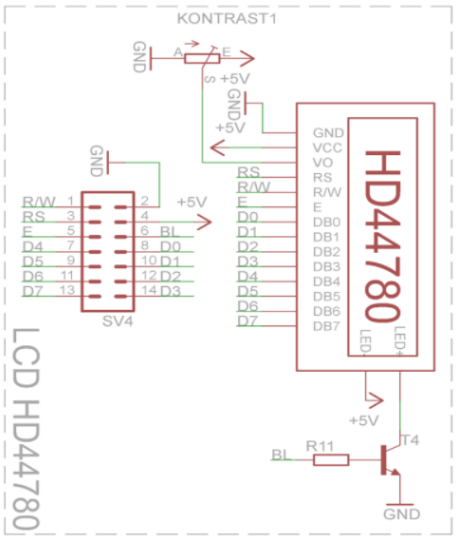
La noi, Led-urile sunt conectate după cum se vede în figura alăturată. Se poate observa că pentru închiderea circuitului, pinii la care sunt conectate **diodele** trebuie **conectate** la **masă (0 logic).**



Display LCD

Display-ul LCD folosit utilizează 16 pini de conectare. Datorită conectarii sale electrice, acesta poate fi comandat prin cuvinte de 4 sau 8 biți.

Contrastul display-ului poate fi controlat din potentiometru, iar jumperul dintre BL si VCC activează lumina de fundal a ecranului.



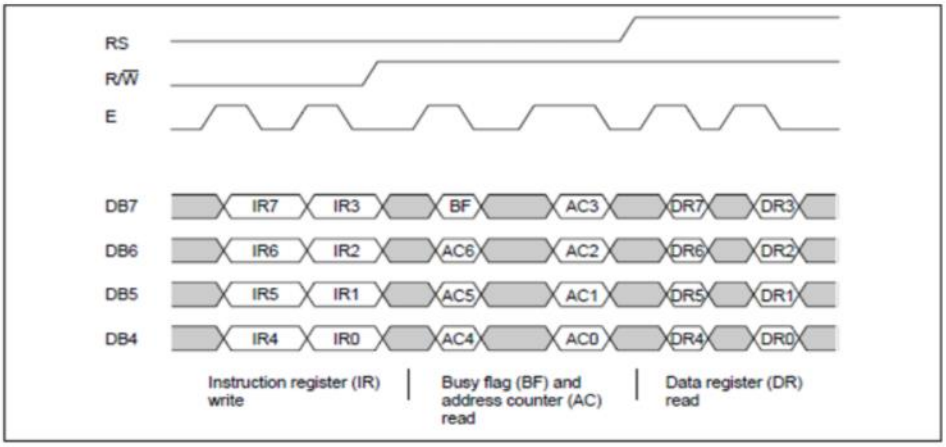
Descrierea pinilor:

1. **GND** (Ground)	8. **D1** (Data1)
2. **Vcc** (Supply Voltage)	9. **D2** (Data2)
3. **Vee** (Contrast Voltage)	10. **D3** (Data3)
4. **R/S** (Instruction/Register Select)	11. **D4** (Data4)
5. **R/$\overline{W}$** (Read/nWrite)	12. **D5** (Data5)
6. **E** (Clock)	13. **D6** (Data6)
7. **D0** (Data0)	14. **D7** (Data7)

O scriere normală pe 4 biti pe un LCD are loc in modul următor:

Interfața permite 4 sau 8 biți în paralel, ceea ce determina o citire/scriere rapidă a datelor de la și către LCD. Această formă de undă va scrie câte un byte ASCII pe ecranul LCD-ului.

Codul ASCII este de 1 byte = 8 biți ce sunt trimiși câte 4 sau 8 biți odată.



Dacă se folosește modul de 4 biți, se transmite către LCD 2 rânduri de date, adică un byte se împarte la 2 și se trimit pe rând. Se transmit primii 4 biți mai semnificativi și apoi ultimii 4 nesemnificativi.

Dacă se folosește modul de 8 biți, se transmite către LCD 1 rând de date, adică 1 byte.

Modul de 8 biți se folosește în aplicațiile în care viteză este foarte importantă și sunt măcar 10 pini disponibili.

Pinul E este folosit pentru inițierea transferului de date.

Pinul R/S este folosit pentru a selecta dacă sunt transmise date sau instrucțiuni între microcontroler și LCD.

Dacă pinul R/S este logic „1”, atunci byte-ul aflat pe poziția actuală a cursorului LCD-ului poate fi scris/citit.

Dacă pinul R/S este „0” logic, este transmisă ori o instrucțiune spre LCD ori se citește statusul ultimei instrucțiuni (dacă a fost terminată sau nu).

Setul de instrucțiuni:

R/S	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Instruction/Description
4	5	14	13	12	11	10	9	8	7	Pins
0	0	0	0	0	0	0	0	0	1	Clear Display
0	0	0	0	0	0	0	0	1	*	Return Cursor and LCD to Home Position
0	0	0	0	0	0	0	1	ID	S	Set Cursor Move Direction
0	0	0	0	0	0	1	D	C	B	Enable Display/Cursor
0	0	0	0	0	1	SC	RL	*	*	Move Cursor/Shift Display
0	0	0	0	1	DL	N	F	*	*	Set Interface Length
0	0	0	1	A	A	A	A	A	A	Move Cursor into CGRAM
0	0	1	A	A	A	A	A	A	A	Move Cursor to Display
0	1	BF	*	*	*	*	*	*	*	Poll the "Busy Flag"
1	0	D	D	D	D	D	D	D	D	Write a Character to the Display at the Current Cursor Position
1	1	D	D	D	D	D	D	D	D	Read the Character on the Display at the Current Cursor Position

Convertor Analog-Digital

Un convertor analog – digital (ADC) este un circuit electronic care convertește o tensiune analogică de la intrare într-o valoare digitală. Aceasta poate fi reprezentată în mai multe feluri în funcție de codificarea datelor : binar, cod Gray sau cod complement al lui doi.
O caracteristică importantă a unui ADC o constituie rezoluția acestuia. Rezoluția indică numărul de valori discrete pe care convertorul poate să le furnizeze la ieșirea să în intervalul de măsură. Deoarece rezultatele conversiei sunt de obicei stocate intern sub formă binară, rezoluția unui convertor analog-digital este exprimată în biți.
O altă caracteristică importată a unui convertor analog-digital o constituie rata de eșantionare. Această depinde de timpul dintre două conversii succesive și afectează modul în care forma de undă originală va fi redată după procesarea digitală.
Practic, se vor cunoaște valori din semnalul original doar în anumite momente de timp corespunzătoare momentului conversiei, deci reconstituirea semnalului nu va putea reproduce perfect originalul.

Figura 4 arată modul în care semnalul eșantionat din figura de mai sus va fi reconstituit în urmă trecerii printr-un convertor digital – analog (DAC). După cum se poate observa, semnalul reprodus nu este identic cu cel original.
Dacă rata de eșantionare ar crește, este de la sine înțeles că semnalul reprodus ar aproxima din ce în ce mai bine originalul.
Care este însă rată minimă de eșantionare pentru a reproduce fără pierderi un semnal de o frecvența dată?
Teorema lui Nyquist spune că o rată de eșantionare de minim două ori mai mare decât frecvența semnalului măsurat este necesară.
Acest lucru poate fi aplicat și dacă avem în schimb un semnal compus dintr-omulțime de frecvente, cum ar fi vocea umană sau o melodie.
Limitele maxime ale auzului uman sunt 20Hz – 20kHz dar frecventele obișnuite pentru voce sunt în gama 20-4000Hz, de aceea centralele telefonice folosesc o rată de eșantionare a semnalului de 8000Hz.
Rezultatul este o reproducere inteligibilă a vocii umane, suficientă pentru transmiterea de informații într-o convorbire obișnuită. Pentru reproducerea fidelă a spectrului audibil se recurge la rate mai mari de eșantionare. De exemplu, înregistrarea pe un CD are o rată de eșantionare de 44100Hz ceea ce este mai mult decât suficient pentru reproducerea fidelă a tuturor frecvențelor audibile.
În funcție de modul în care se execută conversia, convertoarele analog-digitale pot fi de mai multe tipuri :

- ADC paralel (Flash)
- ADC cu aproximare succesivă
- ADC cu integrare (single-slope, dual-slope);
- ADC Sigma-delta (delta-sigma, 1-bit ADC sau ADC cu oversampling).

Timer

Timerul/Counterul, după cum îi spune și numele oferă facilitatea de a măsura intervale fixe de timp și de a genera întreruperi la expirarea intervalului măsurat.

Un timer, odată inițializat va funcționa independent de unitatea centrală (core µP). Acest lucru permite eliminarea buclelor de delay din programul principal.

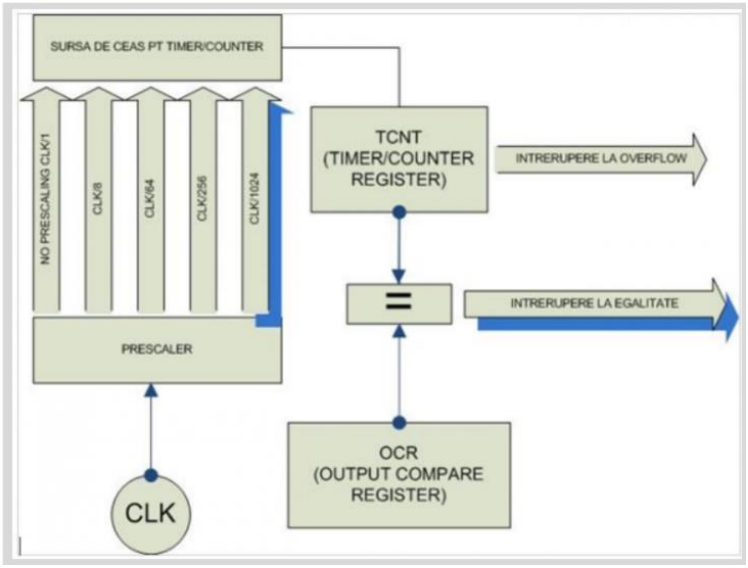
Principiul de funcționare a unui Timer poate fi descris în linii mari astfel:

- Registrul numărător (Timer Counter, TCNT) - măsoară efectiv intervalele de timp și este incrementat automat cu o frecvența dată.

- Prescaler-ul - are rolul de a diviza, în funcție de necesitățile aplicației, frecvența de ceas. La atingerea numărului de diviziuni se incrementează registrul contor al timerului (la ATmega 32 acesta este TCNT).

- La fiecare incrementare a registrului timerului are loc o comparație între acest registru și o valoare stocată în registrul comparator (la ATmega32 OCRn). Această valoare poate fi încărcată de către programator prin scrierea registrului OCRn.

Dacă are loc egalitatea se generează o întrerupere, în caz contrar incrementarea continuă.



Timerele sunt prevăzute cu mai multe canale astfel încât se pot desfășura diferite numărători în paralel. ATmega32 este prevăzut cu 3 unități de timer: două de 8 biți și unul de 16 biți.
Timer/Counter0 și Timer/Counter2 au patru moduri de funcționare (dintre care vom detalia 3, pentru cel de-al patrulea puteți citi datasheet-ul), ce se diferențiază prin:
• valorile pana la care se face incrementarea
• felul in care se numara (doar crescator sau alternativ crescator/descrescator)
• cand se face resetarea contorului
Pentru a configura un mod de funcționare, vom seta:
• biții WGM din timer-ul respectiv (care se găsesc în registrele TCCR din datasheet, la secțiunile aferente timerelor, "Register Description") pentru setarea modului(Normal, PWM, CTC, Fast PWM)
• biții CS din timer-ul respectiv pentru setarea prescalerului
• pragul de numărare, adică valorea lui OCR
• bitul corespunzător lui TIMK, dacă se dorește activarea întreruperilor

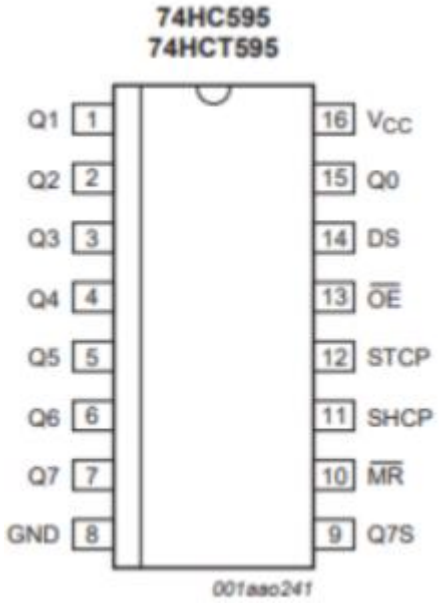
4 Digit, 7 Segment Display

Registrul de shiftare
Intr-o aplicatie in care este necesara utilizarea unui numar mare de pini de iesire pentru a comanda mai multe periferice (LCD, LED-uri, Display pe 7 segmente), putem ajunge in situatia in care sa ne confruntam cu o problema hardware: numarul de pini necesari sa fie mai mare decat numarul de pini disponibili ai microcontrollerului.
In cazul in care acest numar de pini necesari nu poate fi redus, putem apela la un mic „ajutor", adica un registru de shiftare. De exemplu, in cazul in care avem nevoie sa controlam 8 pini digitali de iesire, utilizand un registru de shiftare vom reduce numarul pinilor utilizati de microcontroller de la 8 la doar 3 pini.
Registrul de shiftare lucreaza asemanator unei comunicatii seriale, adica bitii sunt transmisi unul cate unul catre registru si stocati utilizand un clock produs de un pin al microcontrollerului, urmand ca iesirile sa fie active in urma unui impuls (clock) al celui de al treilea pin.
Un avantaj al utilizarii unui registru de shiftare il reprezinta reducerea numarului de pini utilizati de microcontroller, in timp ce un dezavantaj ar fi faptul ca sunt necesare 8 impulsuri de comanda pentru a obtine cele 8 iesiri paralele.

Totodata cu ajutorul acestui registru se poate extinde aplicatia legand in serie mai multi registri si se pot obtine 16, 24, etc. iesiri paralele, pastrand acelasi numar de pini utilizati de Micrcontroller.

74HC595

Pentru aplicatia noastra vom folosi registrul de shiftare 74HC595.



Symbol	Pin	Description
Q1	1	parallel data output 1
Q2	2	parallel data output 2
Q3	3	parallel data output 3
Q4	4	parallel data output 4
Q5	5	parallel data output 5
Q6	6	parallel data output 6
Q7	7	parallel data output 7
GND	8	ground (0 V)
Q7S	9	serial data output
$\overline{MR}$	10	master reset (active LOW)
SHCP	11	shift register clock input
STCP	12	storage register clock input
$\overline{OE}$	13	output enable input (active LOW)
DS	14	serial data input
Q0	15	parallel data output 0
V_{CC}	16	supply voltage

PWM

Regimuri de funcționare:
Timer/Counter0 și Timer/Counter2 au 4 moduri de funcționare:
• Normal
• Phase correct PWM
• CTC – Clear Timer on Compare
• FastPWM

FastPWM	* pornește de la 0 * numără până la valoarea maximă (255 pentru 8 biți, 65535 pentru 16 biți) * pragul se atinge pe parcursul numărătorii		* frecvența este fixată

Cele 4 moduri de funcționare se diferențiază prin:
• valorile până la care se face incrementarea
• felul în care se numără (doar crescător sau alternativ crescător/descrescător)
• când se face resetarea contorului

Registrele Timer:

Timer	Registre	Rol
Timer0 **8 biti**	TCNT0	Registrul contor al timer-ului 0 (*cel care numară*)
	TCCR0	Registre de control ale timer-ului 0 (*aici veți activa diverși biți pentru configurarea timer-ului*)
	OCR0	Registre prag pentru timer-ul 0 (*prag al numărătorii cu care se compară mereu numărătorul*)
	TIMSK,TIFR	Registre cu biți de activare întreruperi timer 0 / flag-uri de activare (*aici activați întreruperile*)
Timer1 **16 biti**	TCNT1H/L = TCNT1H + TCNT1L	Registrul contor al timer-ului 1 (*la fel ca la timer0, doar că pe 16 biți*)
	TCCR1A , TCCR1B	Registre control ale timer-ului 1 (*la fel ca la timer0*)
	OCR1AH/L , OCR1BH/L	Registre prag pe 16 biti ale timer-ului 1 (*la fel ca la timer0*)
	TIMSK, TIFR	(*la fel ca la timer0*)
	ICR1H/L	Registru folosit pentru a reține valoarea contorului la apariția unui eveniment extern pe pin-ul ICP (*nu vom folosi la laborator*)
Timer2 **8 biti**	aceleasi registre ca la Timer0 (TCNT0, TCCR2, OCR2, TIMSK, TIFR)	Diferența față de Timer-ul 0 este posibilitatea folosirii unui oscilator extern separat pentru Timer-ul 2, pe pinii TOSC1 și TOSC2
	ASSR, SFIOR	Registre ce țin de funcționarea asicronă a acestui timer față de restul sistemului

PWM (Pulse Width Modulation) este o tehnică folosită pentru a varia în mod controlat tensiunea dată unui dispozitiv electronic. Această metodă schimbă foarte rapid tensiunea oferită dispozitivului respectiv din ON în OFF și invers.

Perioada de timp corespunzătoare valorii ON dintr-un ciclu ON-OFF se numește factor de umplere (duty cycle) și reprezintă, în medie, ce tensiune va primi dispozitivul electronic. Astfel se pot controla circuitele analogice din domeniul digital. Practic, asta înseamnă că un LED acționat astfel

se va putea aprinde / stinge gradual, iar în cazul unui motor acesta se va învârti mai repede sau mai încet.

Un controler PWM este în esență un convertor digital-analog (DAC) pe un singur bit.

Această modulație vine în sprijinul multor aplicații deoarece înseamnă consum mic, eliminarea zgomotelor și aplicații cu cost redus.

Factorul de umplere se exprimă în procente și reprezintă cât la sută din perioada unui semnal acesta va fi pe ON.

Lucrul cu PWM-ul presupune inițializarea unui timer și apoi configurarea output-ului pe pin.

Fiecare timer are pini pe care poate da output un astfel de semnal (Timer0 are OC0, Timer 1 are OC1A și OC1B, Timer 2 are OC2).

Smart Home

Cuprins

Obiective

- Realizarea unui proiect pe baza cunostintelor dobandite anterior.

Implementare proiect

1. La pornirea sistemului pe LCD se va afisa "Smart Home".

2. La pornirea sistemului va porni un cronometru si se va afisa pe displayul cu 7 segmente ("HH.MM"), unde fiecare minut va fi egal cu o secunda din realitate. Cand cronometrul ajunge la ora 23:59, se va reseta.

3. Sistemul va avea 2 stari: "inchis"/"deschis". Atat timp cat este deshis, va fi aprins un LED verde si sistemul va functiona normal. In starea "inchis" va fi aprins un LED rosu si toate functionalitatile vor fi oprite (setarea alarmei si afisarea/modificarea temperaturii), cu exceptia cronometrului.

4. Sistemul va fi "inchis" intre orele 22:00-06:00 si "deschis" in restul intervalului.

5. Starea actuala va si afisata pe randul 2 al displaylui LCD.

6. Se va simula ajustarea temperaturii cu ajutorul potentiometrului (ca in laboratoarele precendente). Se va seta un prag limita de 27 de grade. Cand se depaseste acest prag, se va porni o alarma (buzzer) si, optional, se va porni un motor DC. Se va afisa temperatura pe primul rand al LCD-ului ("T = gg.g (ASCII)degrees C")

7. La apasarea unui buton se va intra in modul de setare al unei alarme. Ora la care se va activa alarma va fi setata cu ajutorul unui buton/potentiometru. Cand cronometrul va ajunge la ora setata, chiar daca sistemul este in starea inchis, pe primul rand al LCD-ului va fi afisat mesajul"ALARMA!" pentru 5 secunde si apoi se va reveni la starea precedenta.