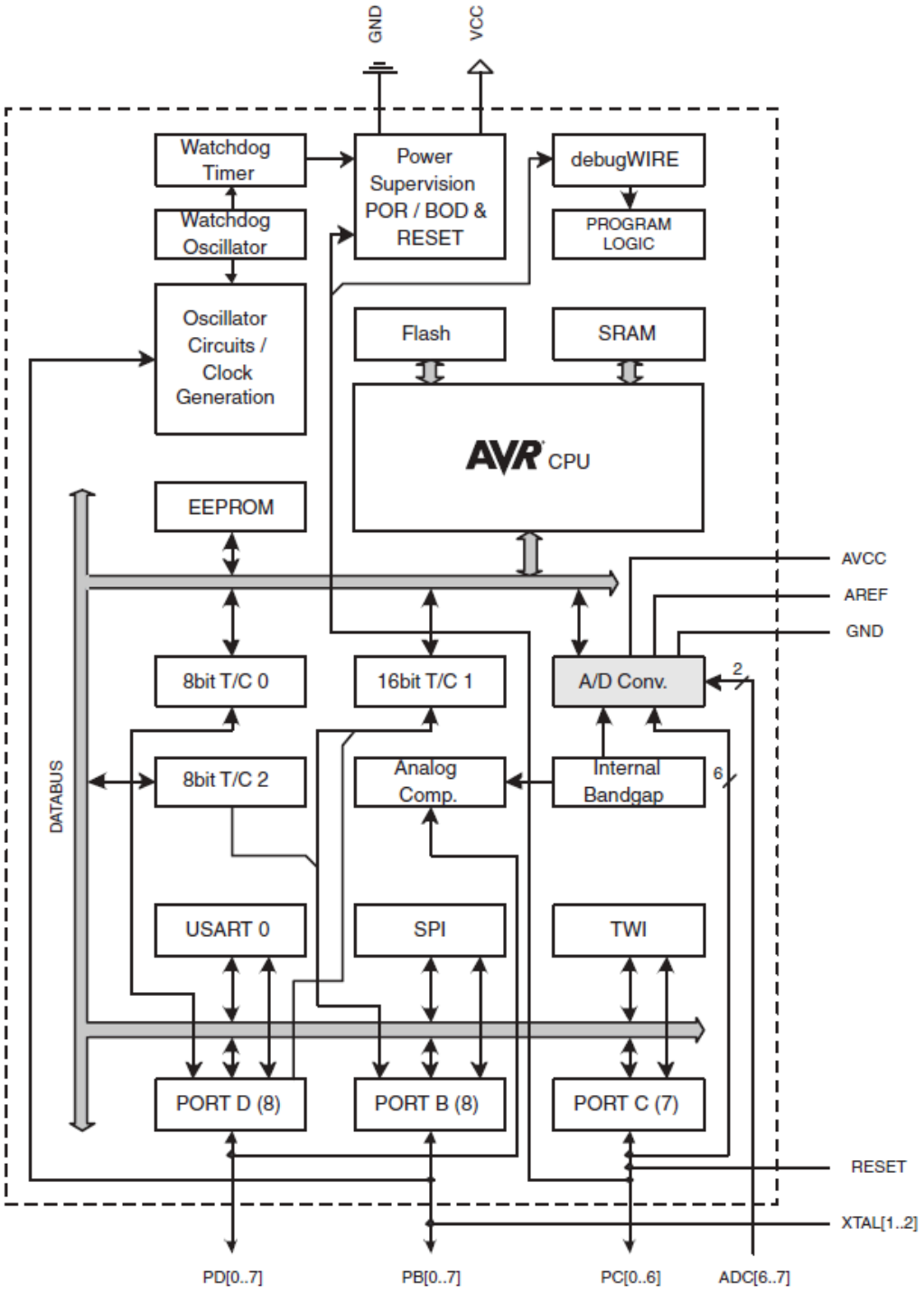
DAY2.

Figure 2-1. Block Diagram



AVR Memory guide: https://docs.arduino.cc/learn/programming/memory-guide/

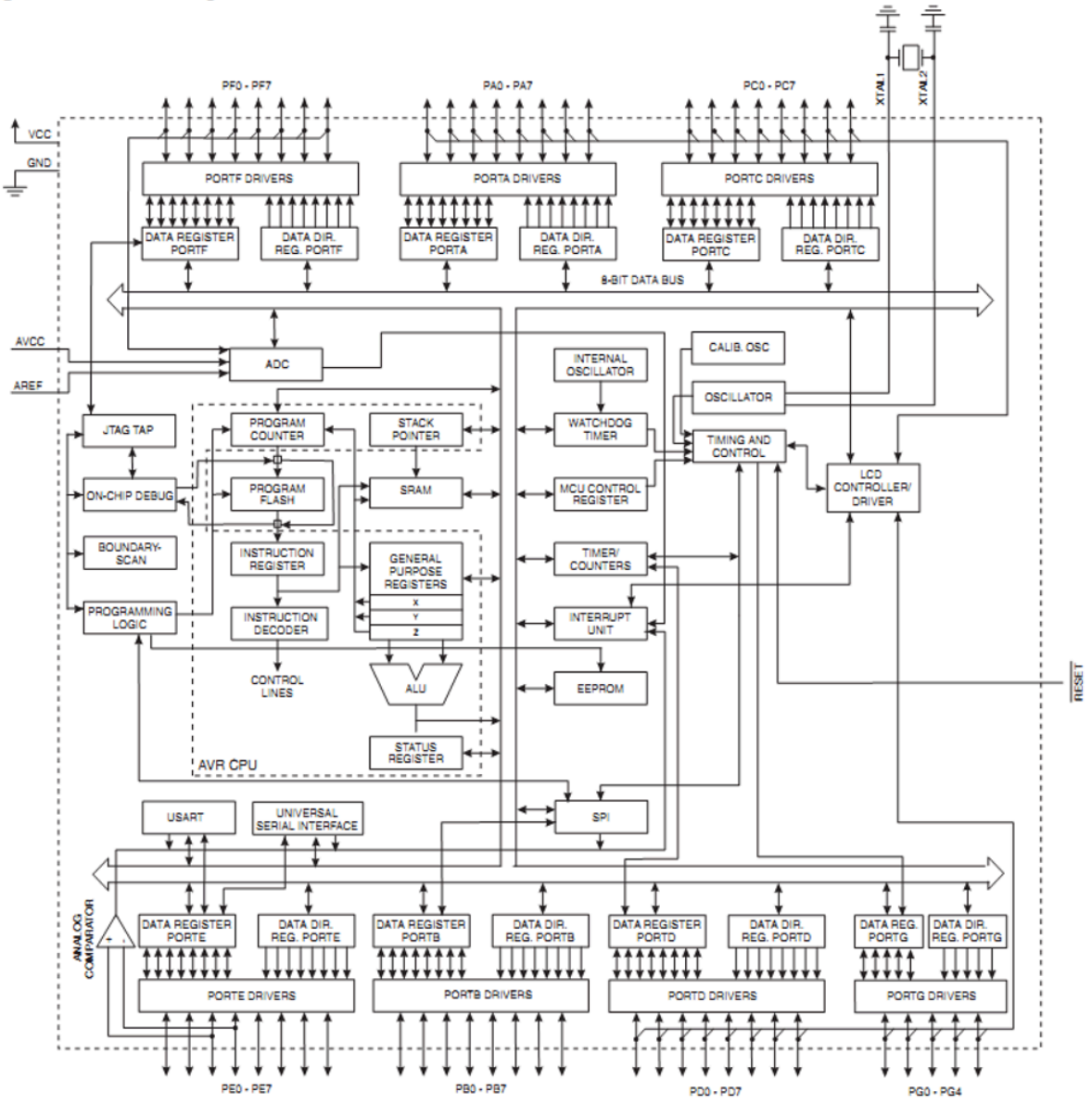
Board	Microcontroller	Family	Architecture	Flash	SRAM	EEPROM
UNO Mini	ATmega328P	AVR	Harvard	32kB	2kB	1kB
UNO Rev3	ATmega328P	AVR	Harvard	32kB	2kB	1kB
UNO WiFi Rev2	ATmega4809	AVR	Harvard	48kB	6kB	256B
UNO Rev3 SMD	ATmega328P	AVR	Harvard	32kB	2kB	1kB
Leonardo	ATmega32u4	AVR	Harvard	32kB	2.5kB	1kB
Mega 2560 Rev3	ATmega2560	AVR	Harvard	256kB	8kB	4kB
Micro	ATmega32u4	AVR	Harvard	32kB	2.5kB	1kB
Zero	ATSAMD21G18	Arm® Cortex®-M0+	Von Neumann	256kB	32kB	-
Portenta H7 (basic configuration)	STM32H747	Arm® Cortex®-M4/M7	Harvard	16MB	8MB	-
Nicla Sense ME	nRF52832	Arm® Cortex®-M4	Harvard	512kB	64kB	-
Nano RP2040 Connect	RP2040	Arm® Cortex®-M0+	Von Neumann	-	264kB	-
MKR FOX 1200	ATSAMD21G18	Arm® Cortex®-M0+	Von Neumann	256kB	32kB	-
MKR NB 1500	ATSAMD21G18	Arm® Cortex®-M0+	Von Neumann	256kB	32kB	-
MKR Vidor 4000	ATSAMD21G18	Arm® Cortex®-M0+	Von Neumann	256kB	32kB	-
MKR WiFi 1010	ATSAMD21G18	Arm® Cortex®-M0+	Von Neumann	256kB	32kB	-
MKR Zero	ATSAMD21G18	Arm® Cortex®-M0+	Von Neumann	256kB	32kB	-
MKR1000 WIFI	ATSAMW25H18	Arm® Cortex®-M0+	Von Neumann	256kB	32kB	-
MKR WAN 1300	ATSAMD21G18	Arm® Cortex®-M0+	Von Neumann	256kB	32kB	-
MKR WAN 1310	ATSAMD21G18	Arm® Cortex®-M0+	Von Neumann	256kB	32kB	-
Nano	ATmega328P	AVR	Harvard	32kB	2kB	1kB
Nano Every	ATmega4809	AVR	Harvard	48kB	6kB	256B
Nano 33 IoT	ATSAMD21G18	Arm® Cortex®-M0+	Von Neumann	256kB	32kB	-
Nano 33 BLE	nRF52840	Arm® Cortex®-M4	Harvard	1MB	256kB	-
Nano 33 BLE Sense	nRF52840	Arm® Cortex®-M4	Harvard	1MB	256kB	-

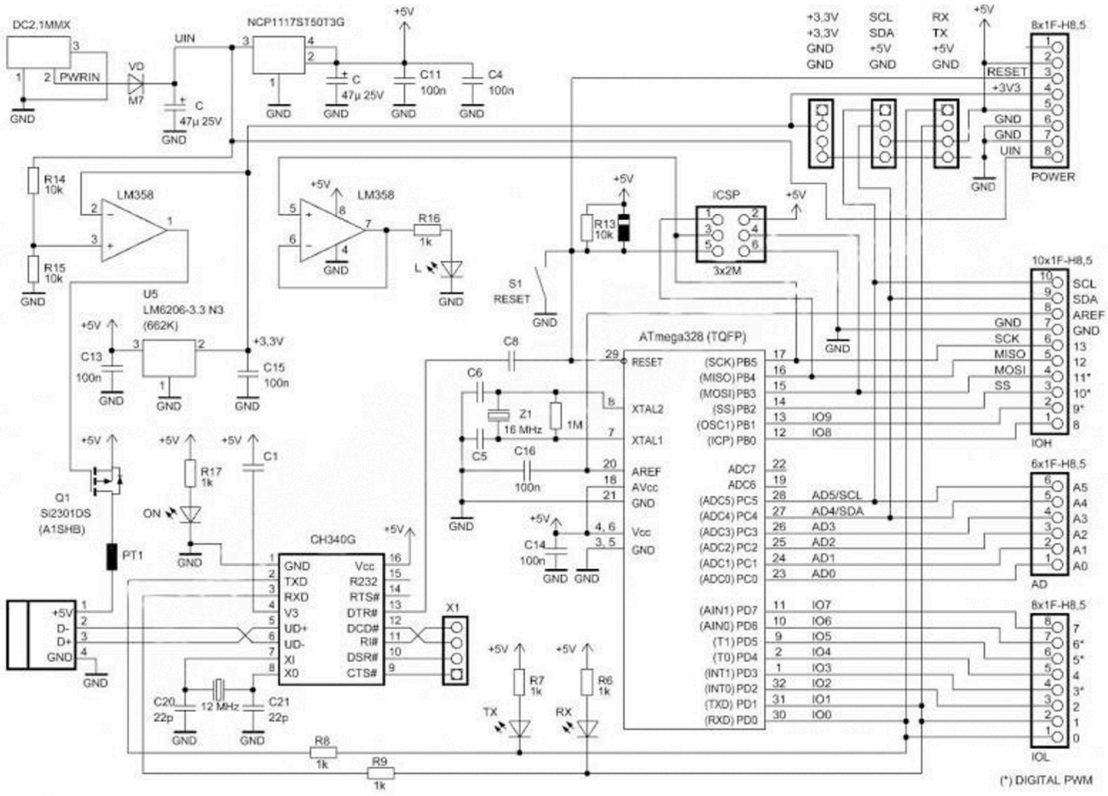
In which types of memory is the C/C++ code stored?
The C/C++ code is splitted into 5 main sections (different programming languages employ different "sectioning", this term may differ too): .bss, .text, .data, .stack, .heap. These sections are stored into different types of memories, this link explains this concept:

https://electronics.stackexchange.com/questions/237740/what-resides-in-the-different-memory-types-of-a-microcontroller

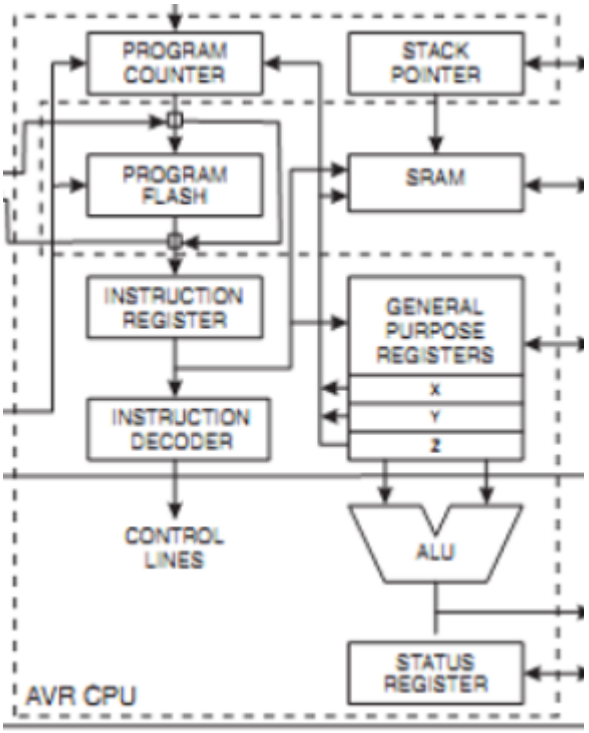
ATMEGA169P: https://eclipse.umbc.edu/robucci/cmpe311/Lectures/L02-AVR_Archetecture/

Figure 2-1. Block Diagram



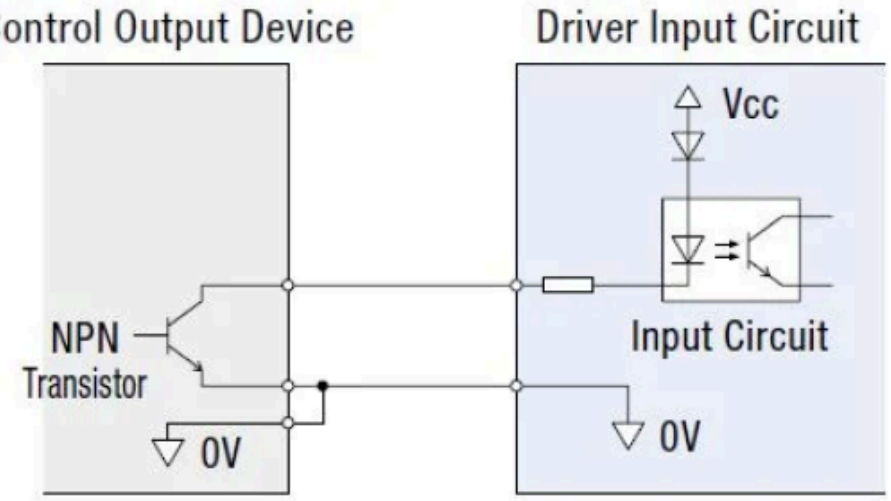

DC2.1MMX
PWRIN
VD
M7
UIN
NCP1117ST50T3G
C11
100n
C4
100n
47µ 25V
GND
+5V
+3,3V
SCL
SDA
RX
TX
8x1F-H8,5
POWER
RESET
+3V3
R14
10k
R15
10k
LM358
R16
1k
R13
10k
ICSP
3x2M
S1
RESET
U5
LM6206-3.3 N3
(662K)
C13
100n
C15
100n
C8
C6
C5
C16
100n
Z1
16 MHz
1M
ATmega328 (TQFP)
C1
R17
1k
ON
Q1
Si2301DS
(A1SHB)
PT1
CH340G
C14
100n
X1
R7
1k
R6
1k
TX
RX
C20
22p
C21
22p
12 MHz
R8
1k
R9
1k
10x1F-H8,5
SCL
SDA
AREF
GND
13
12
11*
10*
9*
8
IOH
6x1F-H8,5
A5
A4
A3
A2
A1
A0
AD
8x1F-H8,5
7
6*
5*
4
3*
2
1
0
IOL
(*) DIGITAL PWM

AVR CPU:

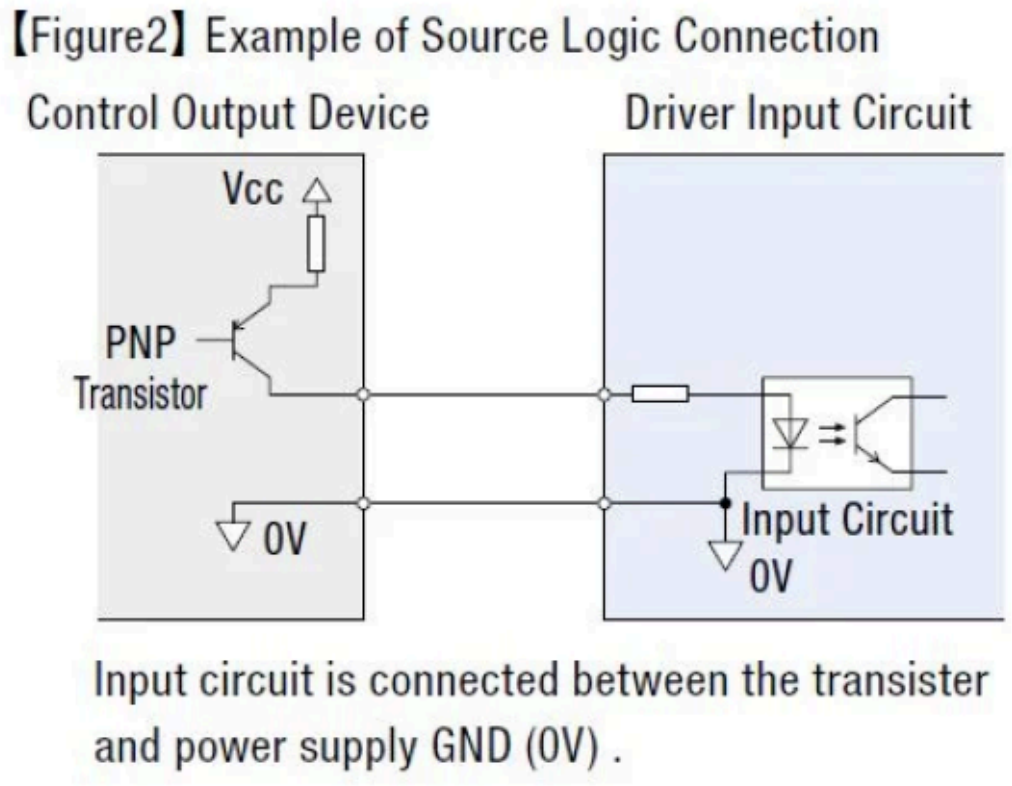

PROGRAM COUNTER
STACK POINTER
PROGRAM FLASH
SRAM
INSTRUCTION REGISTER
GENERAL PURPOSE REGISTERS
X
Y
Z
INSTRUCTION DECODER
CONTROL LINES
ALU
STATUS REGISTER
AVR CPU

==========ANDREI==========

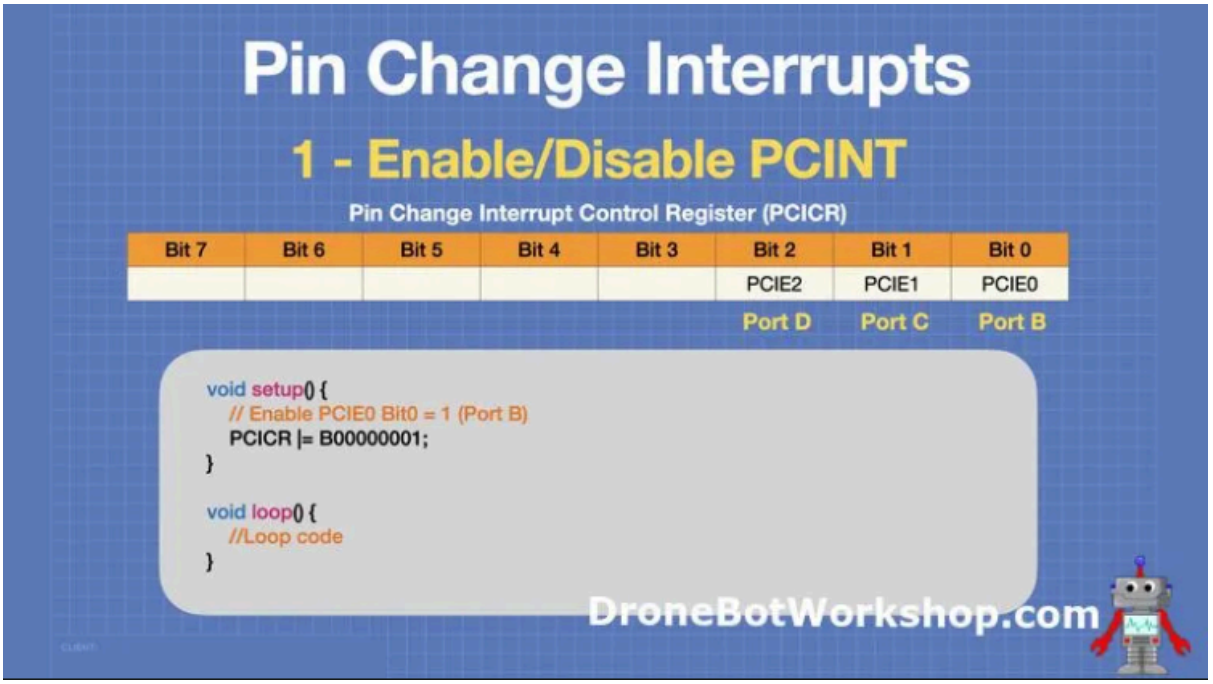
【Figure1】 Example of Sink Logic Connection

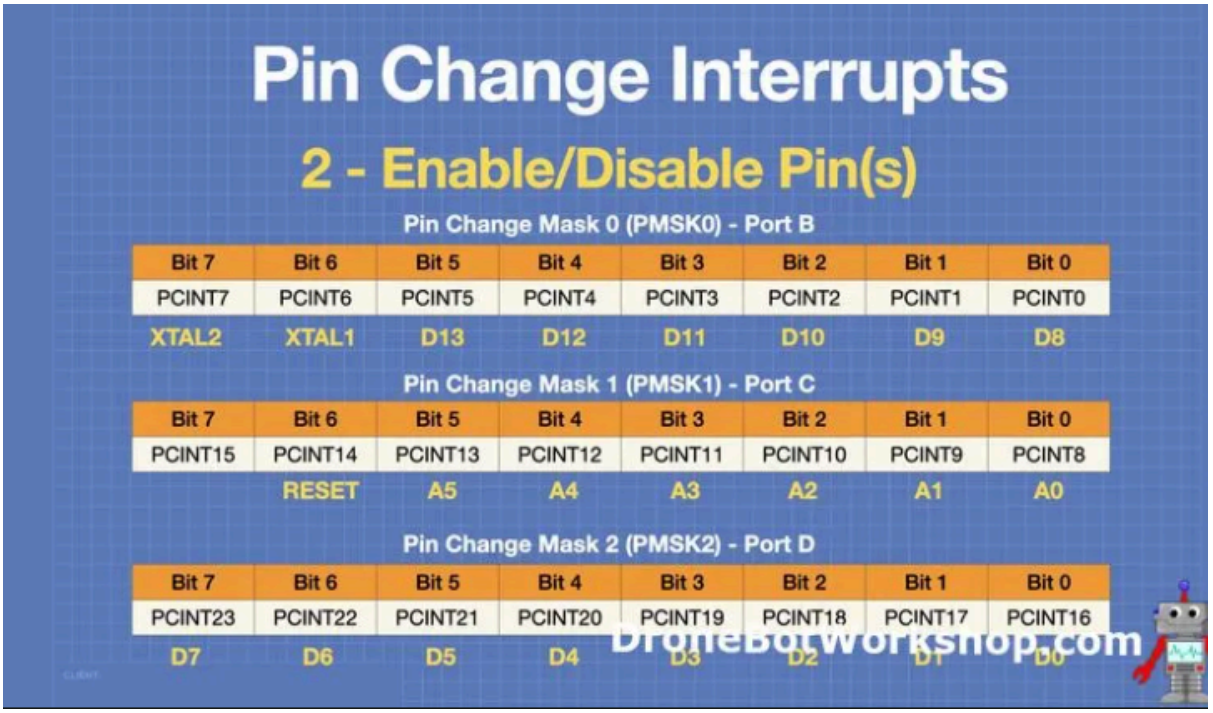


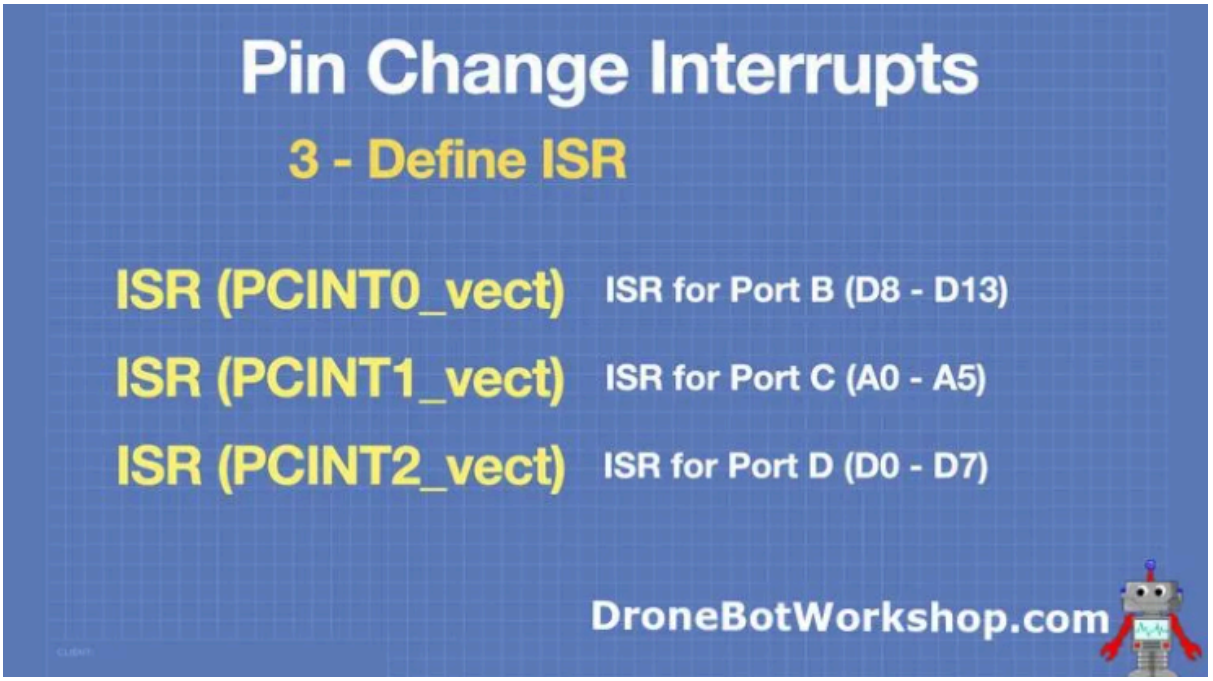
Input circuit is connected between the positive power supply side (Vcc) and transister.



Circuite sursa/drena deschisa:
https://blog.orientalmotor.com/what-is-the-difference-between-sink-and-source-logic







Intreruperi hardware mascabile:https://dronebotworkshop.com/interrupts/

==========MIHAI==========

DAY3.

I fucking start with assembly.

ASSEMBLER: AVRASM2
(https://www.avrfreaks.net/sfc/servlet.shepherd/document/download/0693I00000VHTI5AAH)

1. Line Continuation

Like in C, source lines can be continued by means of having a backslash (\) as the last character of a line. This is particularly useful when defining long preprocessor macros, and for long .db directives:

Example:

```
.db 0, 1, "This is a long string", '\n', 0, 2, \
"Here is another one", '\n', 0, 3, 0
```

2.Integer Constants

AVRASM2 allows underscores (_) to be used as separators for increased readability. Underscores may be located anywhere in the number except as the first character or inside the radix specifier.

Example:

0b1100_1010 and 0b_11_00_10_10_ are both legal, while _0b11001010 and 0_b11001010 are not.

3.Strings and Character Constants

A string enclosed in double quotes (**"**) can be used only in conjunction with the DB directive and the *MESSAGE/WARNING/ERROR* directives. The string is taken literally, no escape sequences are recognized, and it is not NULL-terminated. Quoted strings may be concatenated according to the ANSI C convention, i.e., **"This is a " "long string"** is equivalent to **"This is a long string"**. This may be combined with Line Continuation to form long strings spanning multiple source lines.
Character constants are enclosed in single quotes ('), and can be used anywhere an integer expression is allowed. The following C-style escape sequences are recognized, with the same meaning as in C:

Escape sequence	Meaning
\n	Newline (ASCII LF 0x0a)
\r	Carriage return (ASCII CR 0x0d)
\a	Alert bell (ASCII BEL 0x07)
\b	Backspace (ASCII BS 0x08)
\f	Form feed (ASCII FF 0x0c)
\t	Horizontal tab (ASCII HT 0x09)
\v	Vertical tab (ASCII VT 0x0b)

\\	Backslash
\0	Null character (ASCII NUL)

\ooo (ooo = octal number) and \xhh (hh = hex number) are also recognized.

Examples:

```
.db "Hello\n" // is equivalent to:
.db 'H', 'e', 'l', 'l', 'o', '\\', 'n'
.db '\0', '\177', '\xff'
```

To create the equivalent to the C-string "Hello, world\n", do as follows:

```
.db "Hello, world", '\n', 0
```

4.SEGMENTS

4.1.Code segment: CSEG

The CSEG directive defines the start of a Code Segment. An Assembler file can consist of several Code Segments, which are concatenated into one Code Segment when assembled. The default segment type is Code. The Code Segments have their own location counter which is a word counter. The ORG directive can be used to place code and constants at specific locations in the Program memory. The directive does not take any parameters.

4.2.Data segment: DSEG

The DSEG directive defines the start of a Data segment. An assembler source file can consist of several data segments, which are concatenated into a single data segment when assembled. A data segment will normally consist of BYTE directives (and labels) only. The Data Segments have their own location counter which is a byte counter. The ORG directive can be used to place the variables at specific locations in the SRAM. The directive does not take any parameters.

4.3.EEPROM segment: ESEG

The ESEG directive defines the start of an EEPROM segment. An assembler source file can consist of several EEPROM segments, which are concatenated into a single EEPROM segment when assembled. An EEPROM segment will normally consist of DB and DW directives (and labels) only. The EEPROM segments have their own location counter, which is a byte counter. The ORG directive can be used to place the variables at specific locations in the EEPROM. The directive does not take any parameters.

5.DATA TYPES

5.1.BYTE

The BYTE directive reserves memory resources in the SRAM or EEPROM. In order to be able to refer to the reserved location, the BYTE directive should be preceded by a label. The directive takes one parameter, which is the number of bytes to reserve. The directive can not be used within a Code segment (see directives ESEG, CSEG, and DSEG). Note that a parameter must be given. The allocated bytes are not initialized.

Syntax:
[LABEL]: .BYTE expression

ex:
……
.DSEG
var1: .BYTE 69
…..

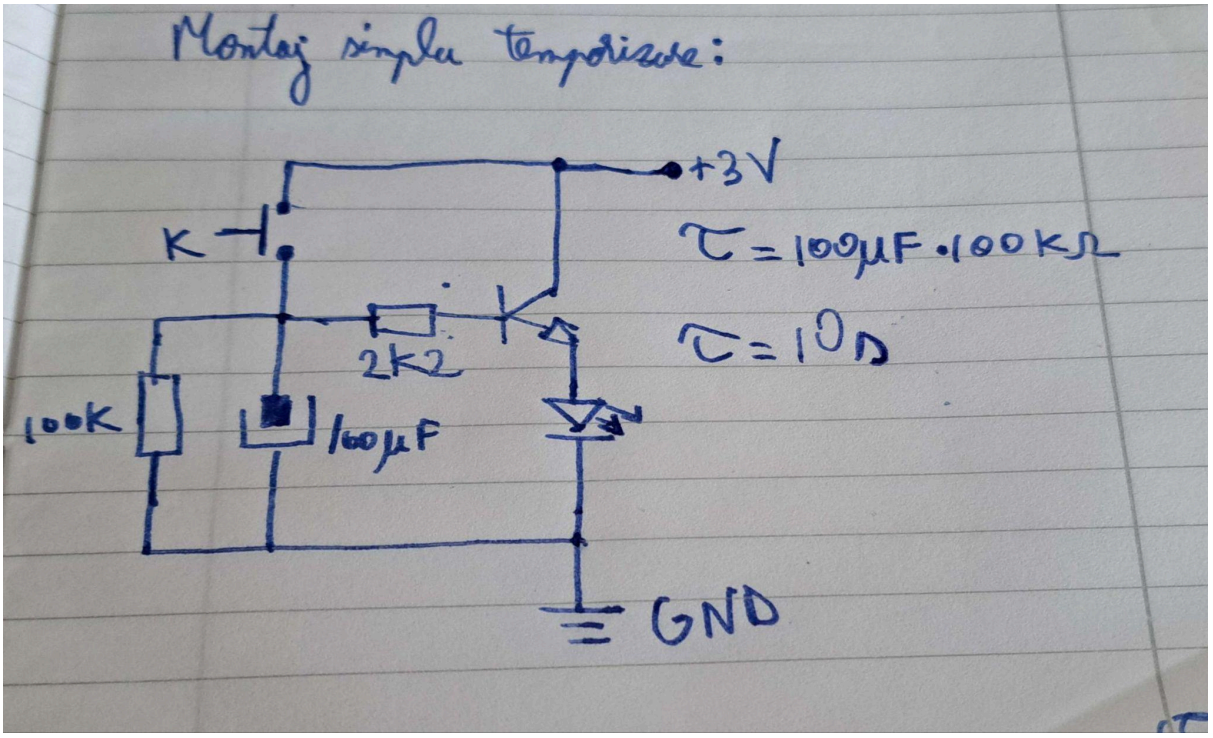
5.2. .DX (define x - .db, .dw, .dd, .dq)

The DX directive reserves memory resources in the program memory or the EEPROM memory. In order to be able to refer to the reserved locations, the DX directive should be preceded by a label. The DX directive takes a list of expressions, and must contain at least one expression. The DX directive must be placed in a Code Segment or an EEPROM Segment.

SYNTAX:
[LABEL]: .Dx expressionlist

EX:
…..
.CSEG
VAR1: .DB 0,1,2,3,45
VAR2: .DW 2452,52425,2
VAR3: .DD 3,52425,2
VAR4: .DQ 69,69
….

==========ANDREI==========



Montaj temporizare:
https://www.youtube.com/watch?v=IqP0__GjQ1k&t=174s&ab_channel=NicolaeAlupoaie

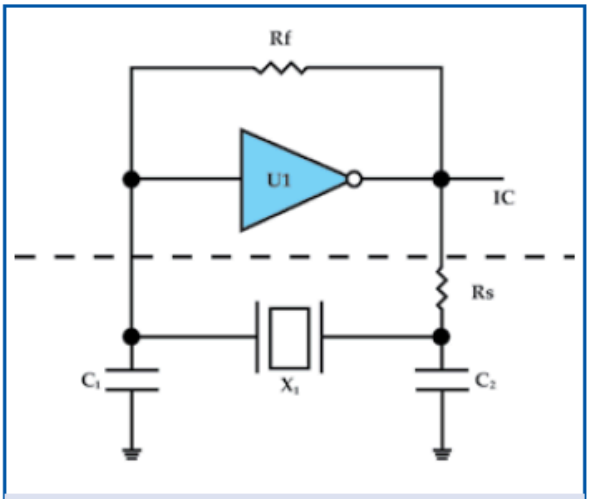


Figure 1: Fundamental Mode Isolated Pierce- Gate Oscillator

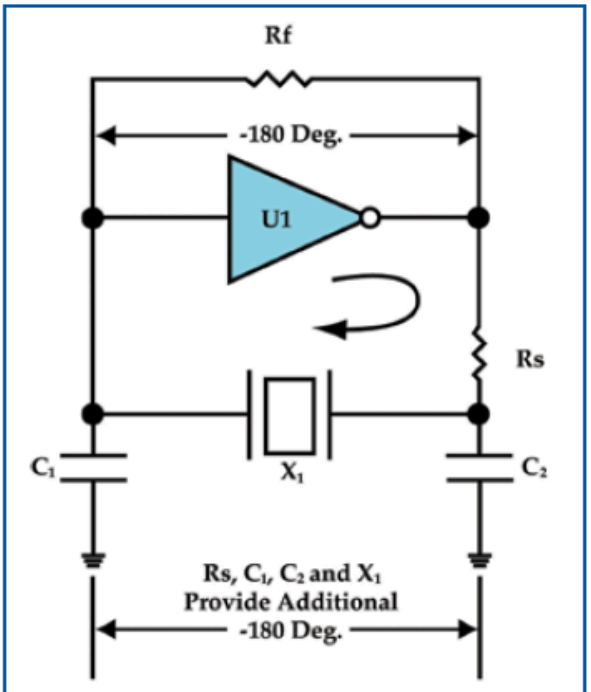
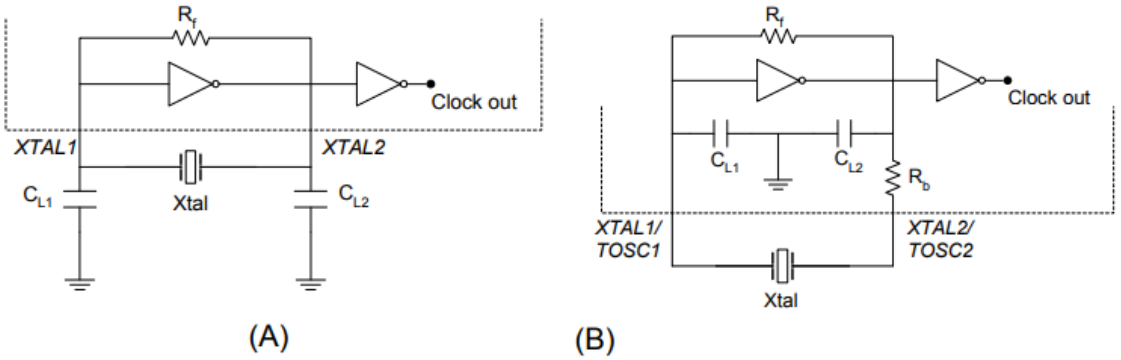


Figure 2: Pierce-Gate Phase Shift Analysis

not func- working design.
leased to The gain around the loop is a func-
se. Many tion of gm (transconductance) of the
elayed in inverter and reactance of C1 and C2
venty-five (Xc1, Xc2) and Rs. Without Rs in

Figure 5-1. Basic inverter circuits equivivalent to the oscillator circuits in AVRs.



Notes: i. Oscillator circuit for crystals and ceramic resonators faster than 400kHz
ii. Circuit for low frequency crystals (32.768kHz) (not on all Atmel®AVR®)

Oscillator Pierce:https://www.crystek.com/documents/appnotes/pierce-gateintroduction.pdf
Oscilator AVR pg 14:
https://ww1.microchip.com/downloads/en/appnotes/atmel-2521-avr-hardware-design-considerations_applicationnote_avr042.pdf
==========MIHAI==========

DAY4.

ATMEL AVR Instruction Set Manual - IS THE KEY

1.Diferenta dintre **directive preprocesor** si **directive**:

Directivele preprocesor (Sintaxa: "#[keyword] [argument]") sunt expresii/ variabile constante care sunt inlocuite de preprocesor inaintea compilarii, deci ele teoretic nu exista dupa compilare. Nu ocupa spatiu in symbol table sau in niciun fel de memorie.

Directivele (Sintaxa: .[keyword] [expresie]) sunt instructiuni executate de asamblor/ linkeditor. Aici se rezerva spatiu de memorie in unele cazuri, sau in symbol table.

Exemplu:

#define MACRO1 3 ; am facut o directiva preprocessor de tip (1) macro constant numita MACRO1 si are valoarea ; 3

.def SYM13=3; am definit un symbol cu numele SYM1 care are valoarea 3

#define, este o directiva de tip macro care are 2 forme:
(1) object-like, care functioneaza ca o constanta

(2) function-like, care functioneaza ca o functie.

.def si #define functioneaza similar in primul caz, cand #define este de prima forma. Dar exista directiva .MACRO pentru #define de forma a doua, care functioneaza la fel. Directiva .MACRO ia parametrii similar ca si in bash cu @0-9, adica poate primii maxim 10 parametrii.

Exemplu:

```
#define MACRO2(x) (x<<3)

.MACRO MACROU
LDI R15,@0
INC R15
ADD @1, @2
;..... etc
.ENDM; sau .ENDMACRO daca iti place.

.CSEG

MACROU 3,R16,R17
```

Cumva iti defineti propria instructiune.

#I need to see what listfile generation is. And what .LIST, .LISTMAC, .NOLIST does#

List file-urile sunt doar text file-uri facute de compilator/ asamblor pentru a fi mai usor pentru programator sa depaneze programul. List file-urile contin symbol table-uri, memory usage, codul asm, etc.

Directivele .LIST, .LISTMAC, .NOLIST sunt folosite pentru a-i spune asamblorului ce sa scrie si ce nu in list file.

2.Proceduri/ Functii/Subrutine

O subrutina (asa se numeste in avr, chit ca se foloseste si termenul de procedura sau functie) arata asa:

[Nume subrutina]:
.....
ret

Si cam atat, sunt mai similare cu cele din LC3.

==========ANDREI==========

11.1 Interrupt Vectors in ATmega328P

Table 11-1. Reset and Interrupt Vectors in ATmega328P

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete
23	0x002C	EE READY	EEPROM ready
24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready

AVR datasheet pg 49:

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

Caracteristicile principale ale unui ADC sunt rezoluţia şi frevenţa de eşantionare. *Rezoluţia* precizează numărul de valori posibile pe care ADC-ul poate să le furnizeze la ieşire (după conversie), în intervalul de măsură. Ţinând seama că rezultatele conversiei sunt stocate sub formă binară, rezoluţia unui ADC se exprimă, firesc, în biţi. Iată un exemplu: dacă rezoluţia unui ADC este de 10 biţi, acesta poate furniza la ieşirea sa $2^{10} = 1024$ valori diferite.

Se defineşte şi o *rezoluţie de măsurare*, care indică diferenţa minimă dintre două valori ale semnalului analogic care sunt convertite în valori distincte. De exemplu, dacă intervalul de măsurare este 0 – 5 V, rezoluţia de măsurare va fi:

$$R = \frac{5\text{V} - 0\text{V}}{1024} = 0,0049\text{V} = 4,9m\text{V} \qquad (4)$$

129

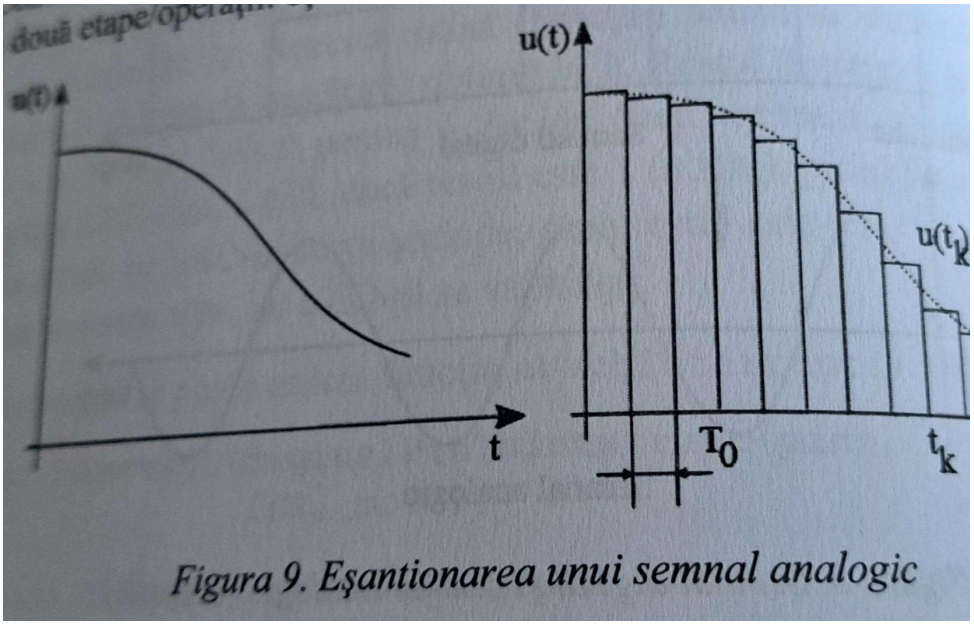
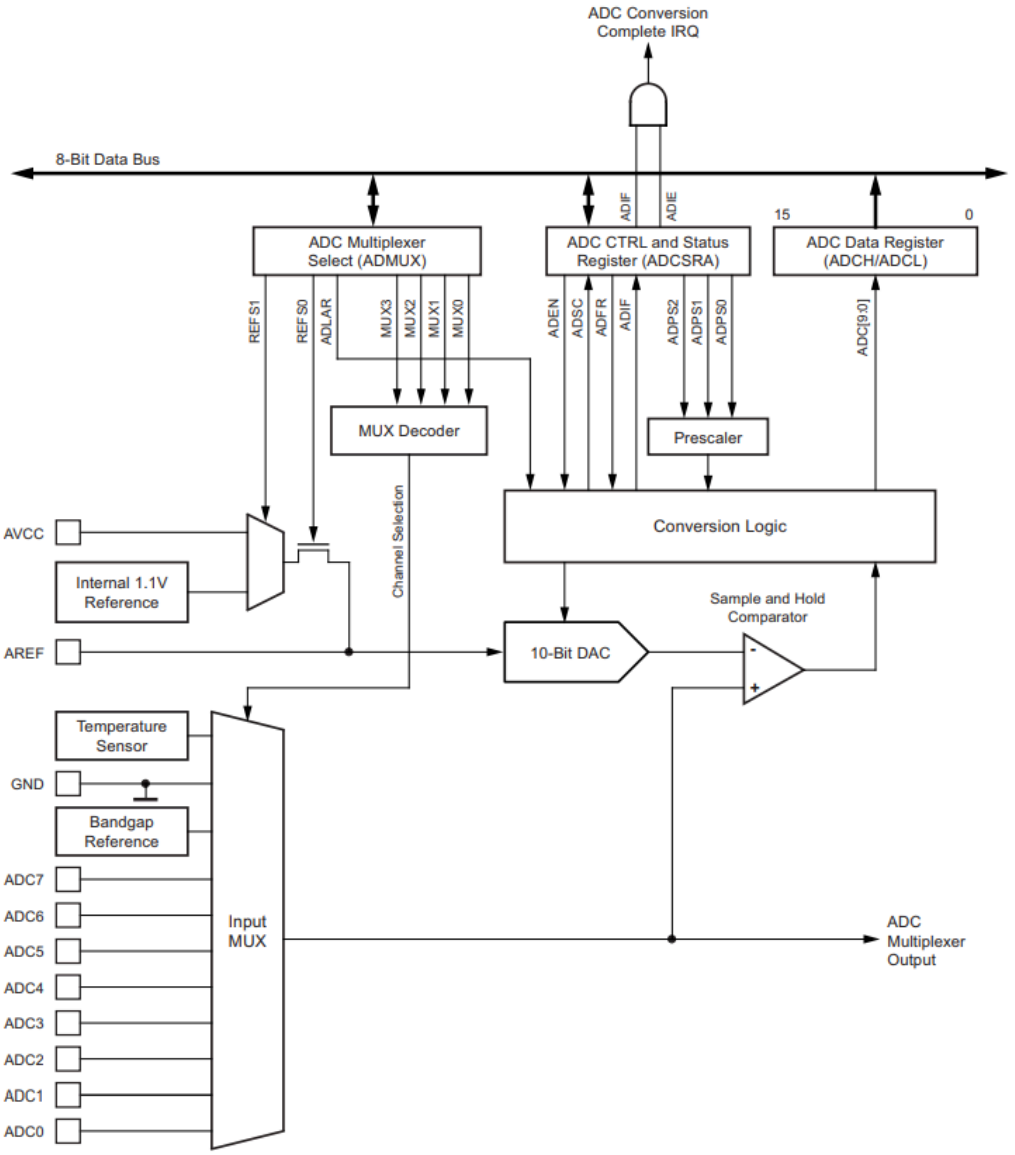


Figura 9. Eşantionarea unui semnal analogic

pune problema de a determina care este rata de eşantionare mi
cesară pentru a reproduce fără pierderi un semnal analogic. P
mularea răspunsului, în ajutor vine teorema Nyquist–Shannon (n
eorema eşantionării), potrivit căreia un semnal analogic cu ban
venţă limitată poate fi reconstituit din eşantioanele sale dacă fre
şantionare este mai mare decât dublul frecvenţei maxime a a
$> 2\nu_M$). Frecvenţa egală cu jumătatea frecvenţei de eşantionare
ele de *frecvenţă Nyquist* ($\nu_N = \nu_0/2$).

gura 9, la momentul $t_k = kT_0$ (k este un număr întreg), v
alului eşantionat este $u(t_k)$ sau $u(k)$.

Traian Anghel-Programarea placii Arduino pg. 129-130



[AVR datasheet pg 206: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf]

==========MIHAI==========

Day 5.

Obiective: Label-uri, pointerii X,Y,Z

Ce salveaza mai exact label-urile?
Valoarea care se afla sfarsitul fisierului .map.
Acolo avem adressa de inceput a fiecarui label. Valoarea pe care o are label-ul semnifica cuvantul la care se afla in memorie (addresa).

Ex:

```
.cseg
.org $0003
MyLabel: .db $69,0
.....
```

MyLabel va avea valoarea 3 deci o executie de genul “LDI R16,MyLabel” va incarca in r16 valoarea 3.

==========ANDREI==========

Day 6.

Obiective: pointerii x,y,z

==========ANDREI==========

Day7.

Am incercat sa lucrez recursiv cu stiva, ca in x86, insa este forbidden in avr deoarece consuma prea multe resurse si nu exista un bp prin care sa accesam eficient cadrul de stiva, asa ca trebuie sa evitam pe cat se poate folosirea de functii recursive.

==========ANDREI==========

Day9 i think

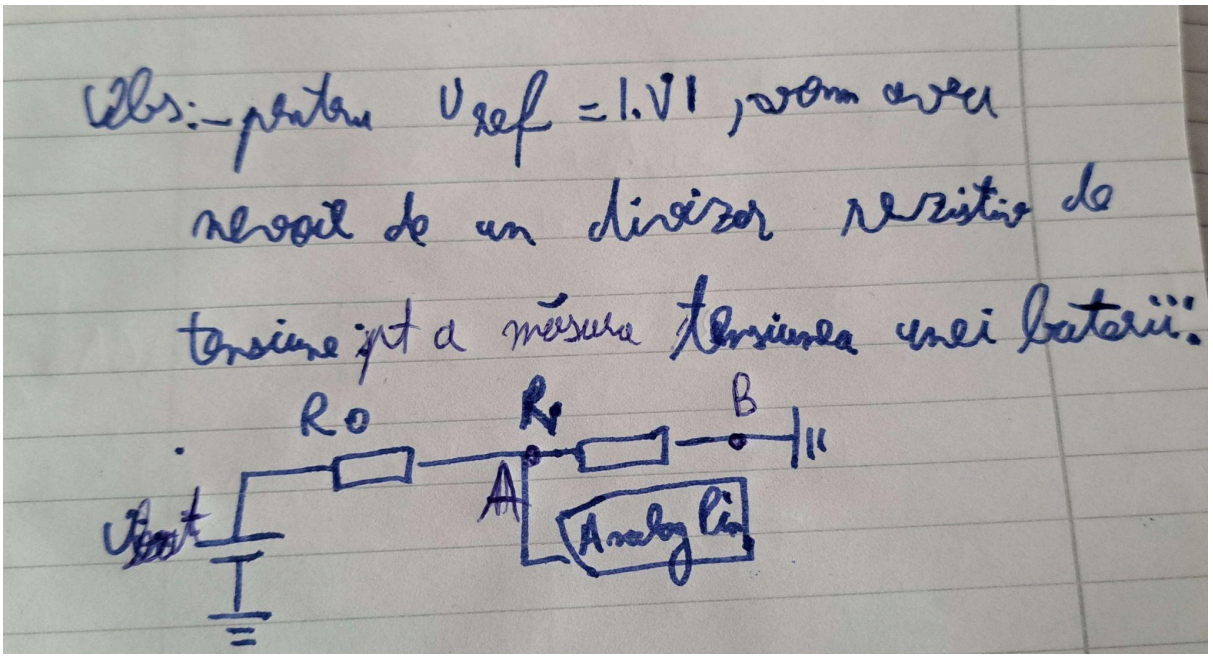
Timer/Counters theory:

https://www.embeddedtutor.com/2019/02/timercounter-in-embedded-system.html

https://www.instructables.com/AVR-Microcontroller-LEDs-Flasher-Using-Timer-Timer/

DAY X,$X \in N$

Convertorul analog-numeric(CAN) de pe ATmega328 are o tensiune de referinta interna de 1.1 V cu o abatere de $\pm$ $0.1V$.Pentru precizie se recomanda masurarea referintei reale de



pe chip.De asemenea,abatere exista si pentru referinta implicita de 5 V.

Momentan vom folosi tensiunea de referinta AVCC(5V) datorita lucrului facil cu aceasta(totusi,va fi o problema cand acelasi AVCC este alimentarea pentru uController).
==========

Convertorul analog-numeric(CAN):

Pentru conversie este nevoie de doua etape:esantionare(gestiune frecventa) si cuantizarea(gestiune zgomot)

Esantionare:se bazeaza pe teorema Nyquist-Shannon

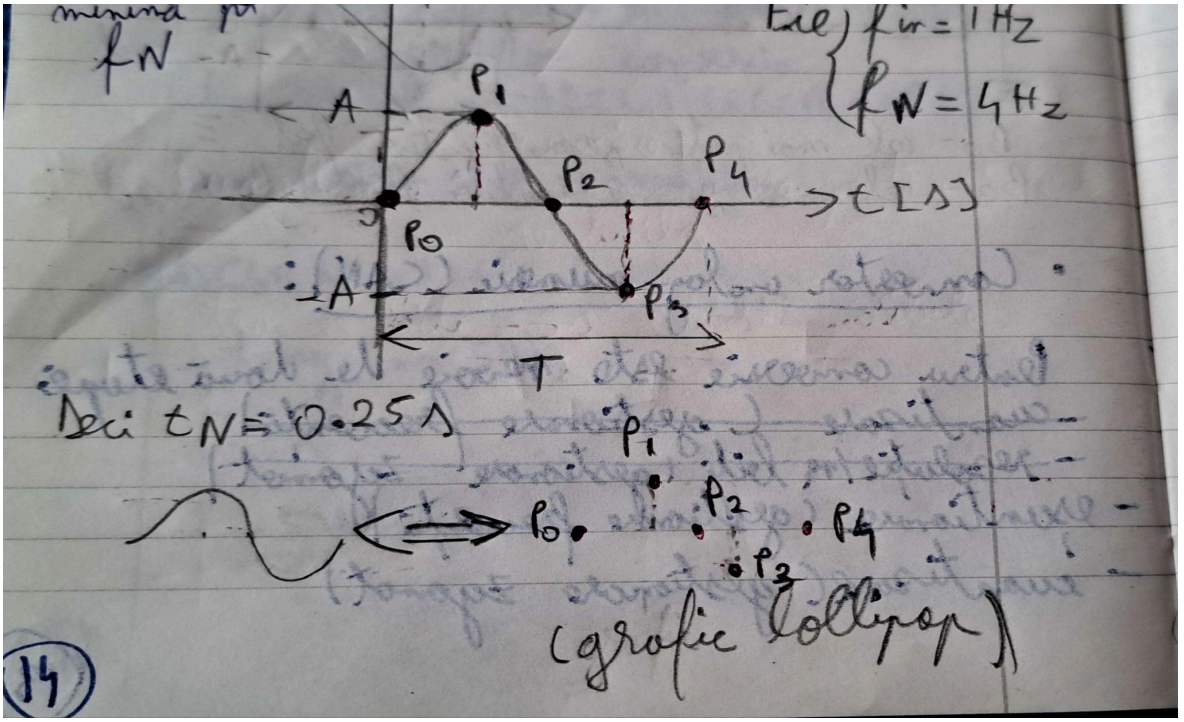
fie Fin-frecventa semnalului analogic de intrare

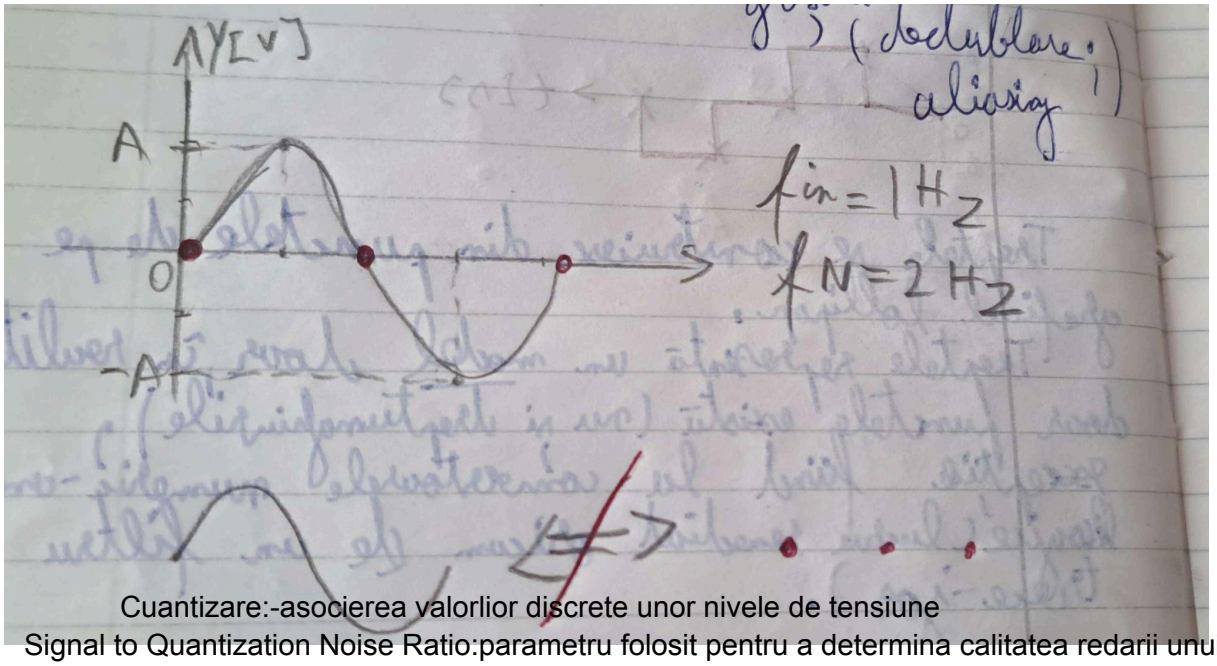
FN-frecventa de esantionare

tN-durata unui esantion(tn=1/FN)

Daca 2*Fin<FN=>reconstituirea semnalului analogic de intrare este posibila

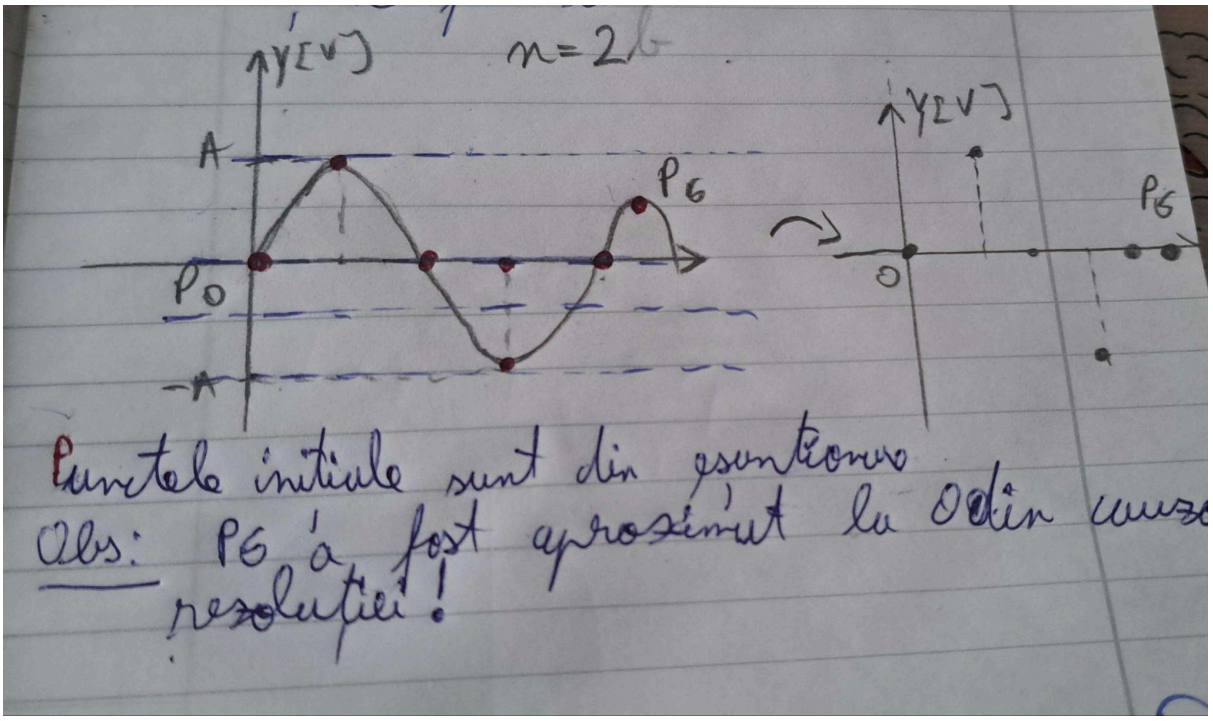
Daca conditia nu este satisfacuta=>aliasing(dedublare)

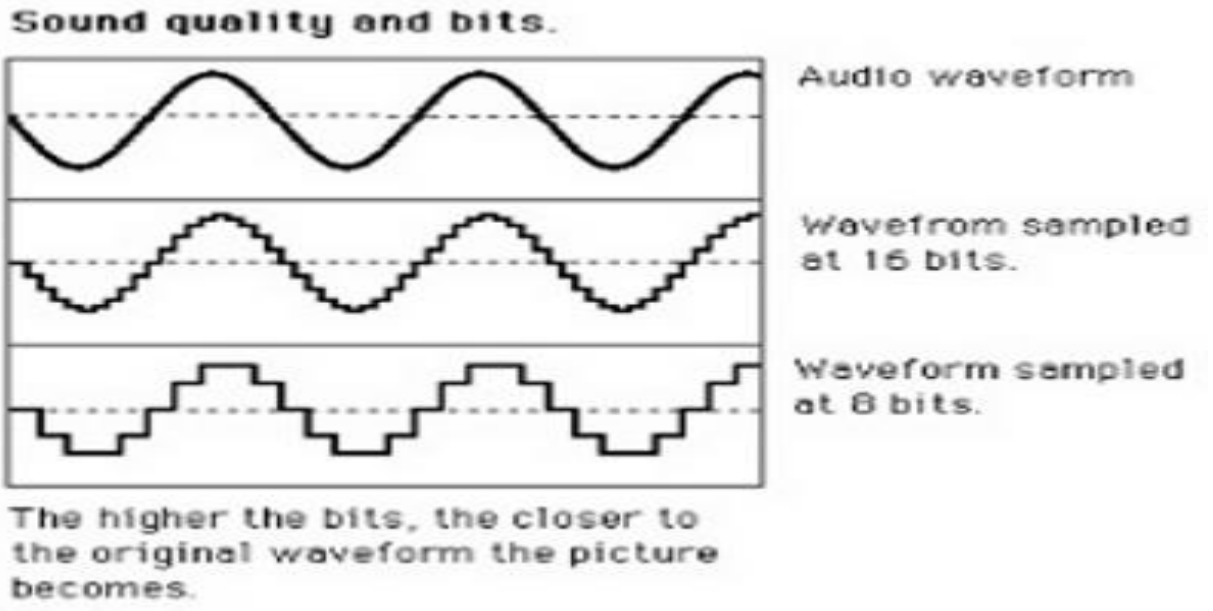




Cuantizare:-asocierea valorlior discrete unor nivele de tensiune

Signal to Quantization Noise Ratio:parametru folosit pentru a determina calitatea redarii unui sunet(SQNR).Cu cat SQNR este mai mare,cu cat este mai buna calitatea.





Pentru experimentare se poate folosi Audacity.

Modificare rata esantionare:Audacity>File>Open file(sau import)>selectare fisier>Rate

Modificare rezolutie:Audacity>Tools>Nyquist Prompt:return quantize(*track*,N)>Apply

unde $N = 2^{n-1}$,n=nr biti

Aflare zgomot:Audacity(includere fisier original+cel modificat)>Effects>Special>Invert(scaderea a doua semnale este o adunare cu un semnal de semn negativ)>Selectare cele doua semnale>Tracks>Mix>Mix and Render to new track

Surse:

https://www.youtube.com/watch?v=clQ9IXSUzuM&t=1289s

https://www.youtube.com/watch?v=zC5KFnSUPNo&list=PLMzeLoLuzIINdzIkqVjyNkreUw15NWfbI&index=38&t=455s

https://www.youtube.com/watch?v=vrXGaFV1AmE&list=PLbqhA-NKGP6B6V_AiS-jbvSzdd7nbwwCw&index=2

https://www.youtube.com/watch?v=1KBLguIXL30

https://www.youtube.com/watch?v=X4JEMCQMwOM

PS:Nevoie de a investiga rolul functiei Dirac delta in cadrul conversiei

==========MIHAI==========