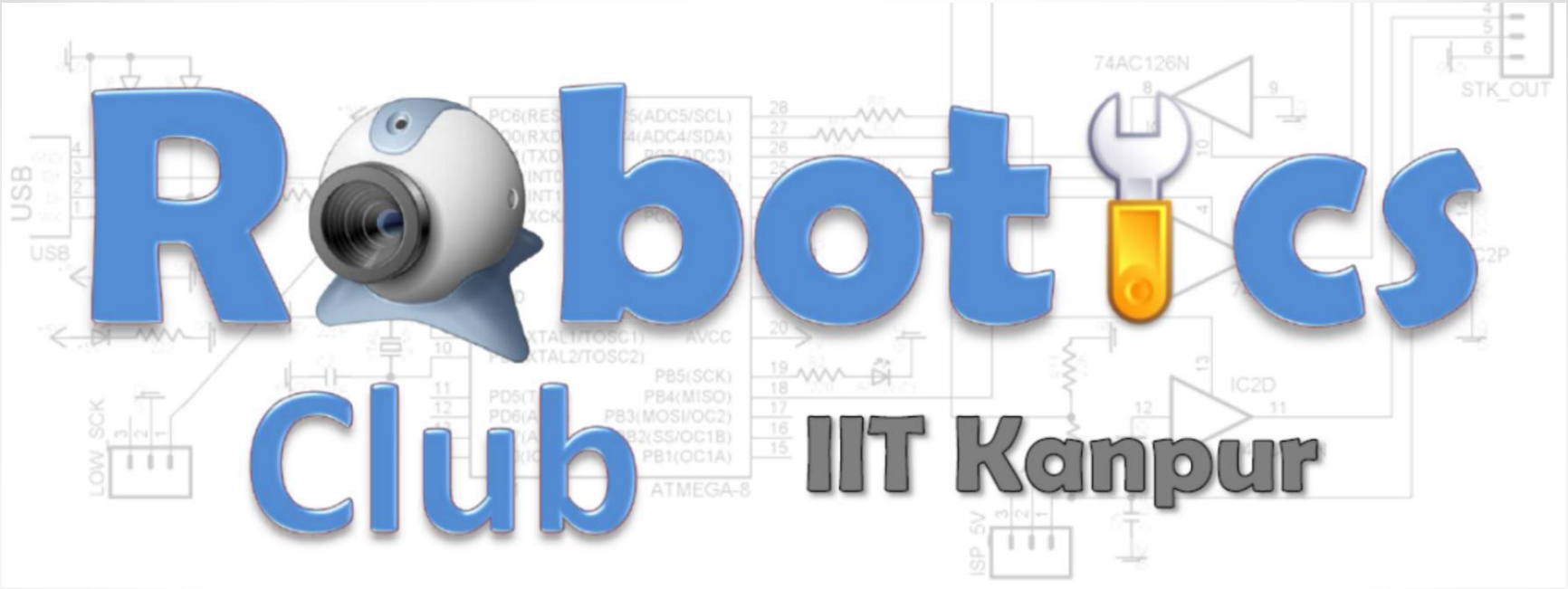
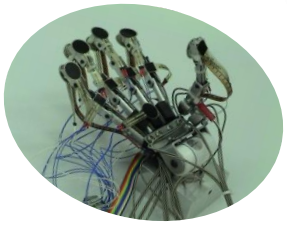


MicroControllers Basics

- Vivek Kumar
- Abhishek Sharma

3 major fields of robotics

- Mechanical
  - Making the "Skeleton" of your bot.



- Electronics
  - Implanting "Nerves & Muscles" into that skeleton.

- Programming
  - Giving "Brain" to your bot.

Topics for Today

- *Introduction to Microcontrollers*
- *Inside a Microcontroller*
- *Working of Microcontrollers*
- *Applications of Microcontrollers*
- *Writing a program in CV avr*
- *Compiling the program into .HEX file*
- *Upload the program into Microcontroller*

So... Lets Start

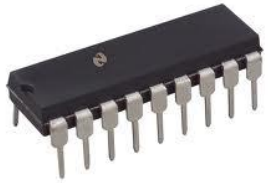
- ***Introduction to Microcontrollers***
- *Inside a Microcontroller*
- *Working of Microcontrollers*
- *Applications of Microcontrollers*
- *Writing a program in CV avr*
- *Compiling the program into .HEX file*
- *Upload the program into Microcontroller*

So... what a Microcontroller (µC) is?

The brain of your Robot?

A black chip or IC (Integrated circuit)?

Like this…



But it is much more than that…

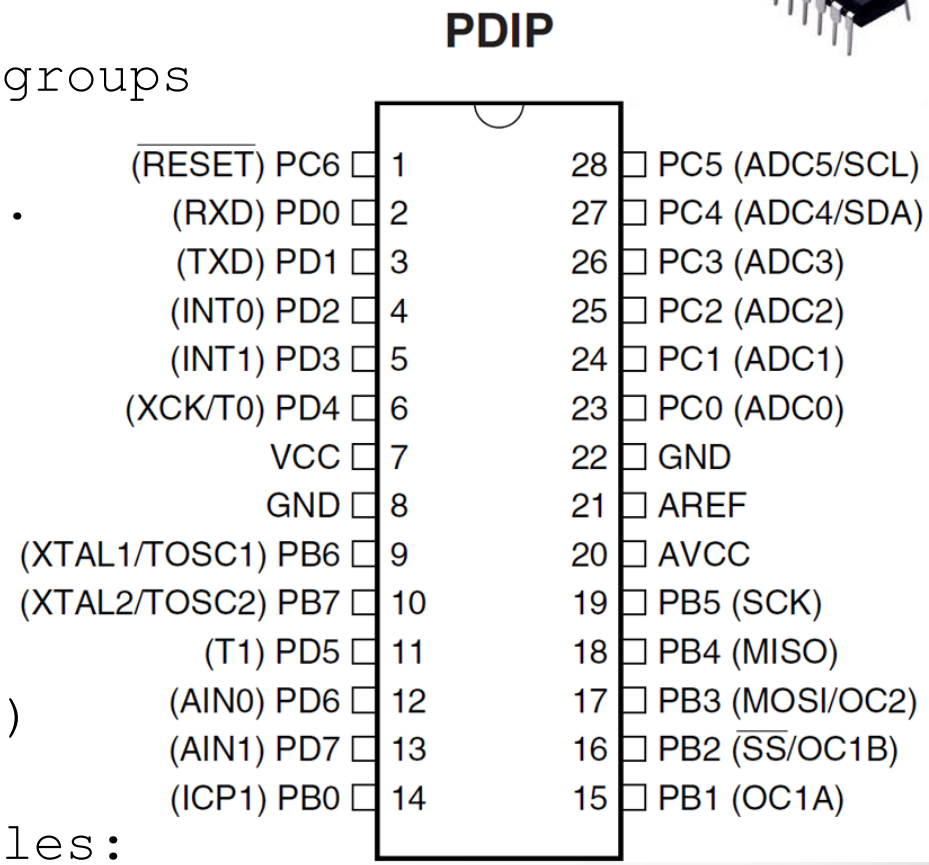
- It's a mini-computer inside a single IC.
- It has processing power.
- It executes a program just like a computer.
- It has pins, memory, registers, clocks, timers, interrupts and much more…

Features of a Microcontroller (µC)...

- **They are Programmable**
  - That's what makes them different from a normal IC.
  - Programs can be written in C lanuage.
- **Various microcontrollers available in the market**
  - PAVR, PIC, Intel 8051, Rabbit, Zilog and Many, Many more…
- **We'll be using Atmel ATmega microcontrollers**
  - ATmega8, ATmega16…
  - Because they are cheap, easy to use and powerful.

Features of ATmega 8

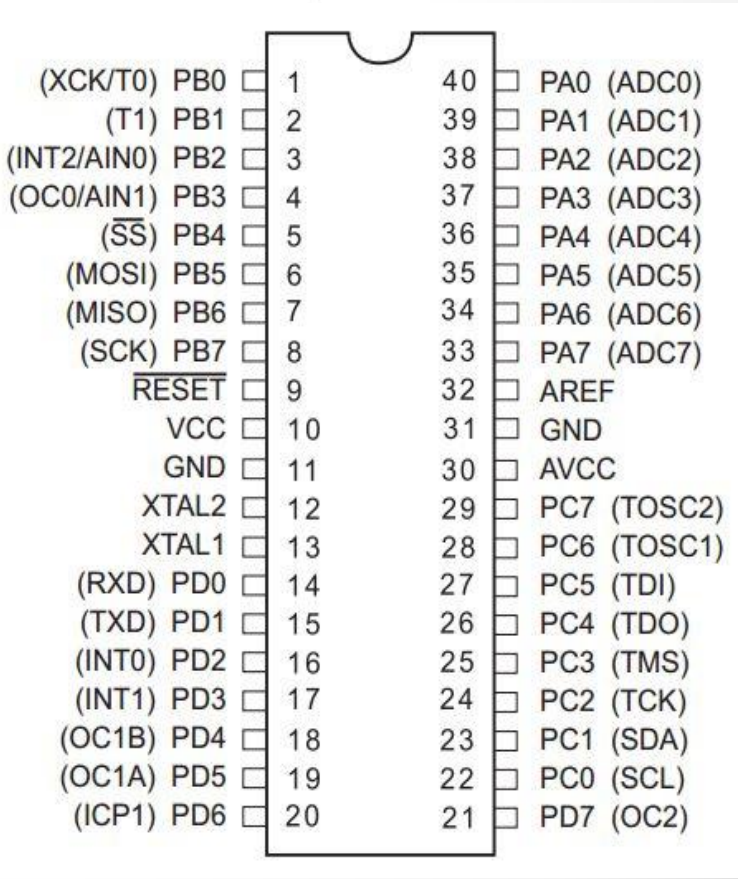
- Has 28 pins
- 23 pins for I/O
- 5 pins reserved
- Clock Frequency: 8Mhz
- I/O pins divided into 3 groups
  - Named as Ports.
  - Three ports B,C and D.
- 8Kb Flash memory
- 512Bytes EEPROM
- 1Kb Internal SRAM
- Three inbuilt timers
  - Two 8-bit
  - One 16-bit
- Operating Voltages
  - 2.7V - 5.5V (ATmega8L)
  - 4.5V - 5.5V (ATmega8)
- Maximum programmable Cycles:
  - 10,000 for Flash memory
  - 100,000 for EEPROM



ATmega 8 & Its Pin Configuration

Features of ATmega 16

- Has 40 pins
- 32 pins for I/O
- 8 pins reserved
- Clock Frequency: 16Mhz
- I/O pins divided into 4 groups
  - Named as Ports.
  - Four ports A,B,C and D.
- 16Kb Flash memory
- 512Bytes EEPROM
- 1Kb Internal SRAM
- Three inbuilt timers
  - Two 8-bit
  - One 16-bit
- Operating Voltages
  - 2.7V - 5.5V (ATmega16L)
  - 4.5V - 5.5V (ATmega16)
- Maximum programmable Cycles:
  - 10,000 for Flash memory
  - 100,000 for EEPROM

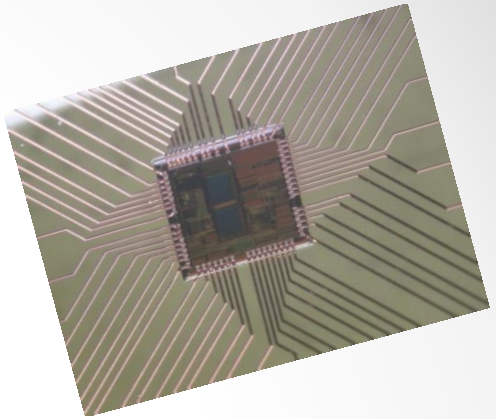


ATmega 16 & Its Pin Configuration

Next...

- *Introduction to Microcontrollers*
- ***Inside a Microcontroller***
- *Working of Microcontrollers*
- *Applications of Microcontrollers*
- *Writing a program in CV avr*
- *Compiling the program into .HEX file*
- *Upload the program into Microcontroller*

Memory Structure

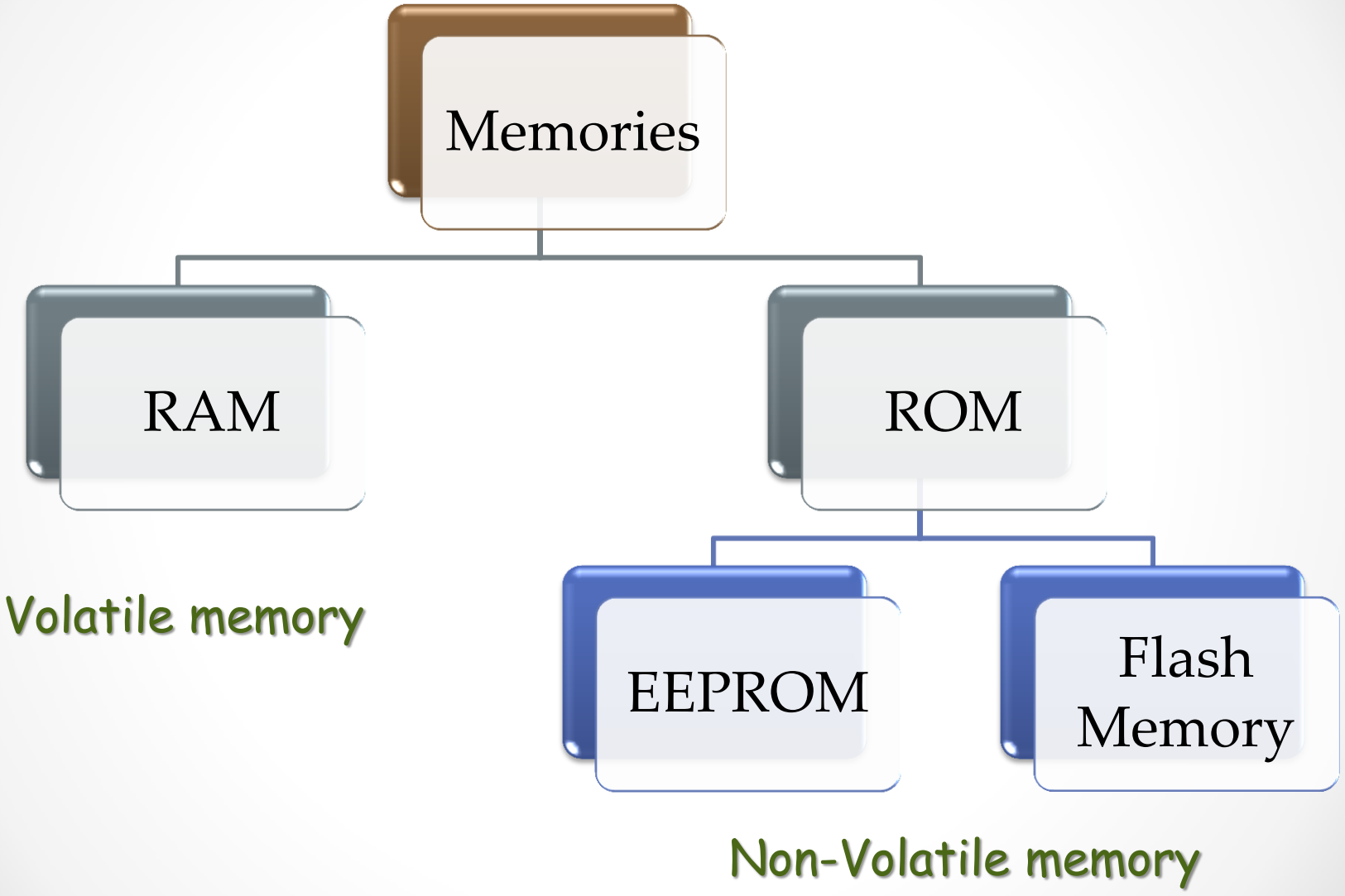


Several type of memories in the μC.

- **Flash Memory**
  - A non-volatile Read Only Memory (ROM)
  - Our program is stored here.
- **Random Access Memory (RAM)**
  - A volatile memory used during runtime of the code.
  - It is quite fast. Reading and writing takes much less time.
- **Electronically Erasable Programmable Read Only Memory (EEPROM)**
  - A non-volatile memory used to store important data.
  - It is quite slow. Reading and writing takes much much time.

- **Volatile memory** : loses memory when power removed
- **Non-Volatile memory** : Retains memory when power removed

Memory Structure



Till now... Any Questions?

- Then, here's a question for u...

Q. What is the full form of EEPROM?

A. **E**lectronically **E**rasable **P**rogrammable **R**ead **O**nly **M**emory

Q. What is the full form of EEPROM?

A. **E**lectronically **E**rasable **P**rogrammable **R**ead **O**nly **M**emory

Next...

- *Introduction to Microcontrollers*
- *Inside a Microcontroller*
- ***Working of Microcontrollers***
- *Applications of Microcontrollers*
- *Writing a program in CV avr*
- *Compiling the program into .HEX file*
- *Upload the program into Microcontroller*

Registers?

- I/O Pins of a Microcontroller are divided generally into groups of 8 pins called **PORT**.
- Each PORT contains some registers associated with it.

- **Registers** are the links between our code and the hardware (pins).
- They actually are the memory locations inside the µC whose names and sizes are predefined.

- We'll talk about the 3 registers that control the I/O pins: **DDR**, **PORT** and **PIN** registers.
- Thus, each port (port A,B,C,D) contains these 3 registers.
- e.g. port A has registers **DDRA**, **PORTA**, **PINA**;
  port B has registers **DDRB**, **PORTB**, **PINB**;

DDR (Data Direction Register)

- It is 8-bit register which sets the pins either input or output. **Each bit of the register corresponds to a pin.**

- 0 - Sets the corresponding pin INPUT
- 1 - Sets the corresponding pin OUTPUT

- To access a specific bit of DDR register we write:
  DDRX.0 = 0; where, X is the port name.
- To set values of all bits simultaneously , we write **DDRA = 0b11001010**

- Remember setting a pin OUTPUT does put +5 v on the pin and vice versa.
- Hence, if u wrote **DDRA.1=0**, you **can't write +5V** on that pin, you **can only read** voltage from that pin.

PIN Register

- It is also **8-bit register** which reads the incoming voltage on the pins.

- If the incoming voltage is 0V, it reads 0
- If the incoming voltage is +5V, it reads 1

- Similarly, To read a specific pin we write:

  int value = **PINX.1**; reads the voltage at pin X.1

- If the pin is input, then the voltage at that pin is **floating** until an external voltage is applied.
- If the pin is output, then the voltage at that pin is fixed either 0V or +5V.

PORT Register

- It is also **8-bit register** which sets the voltage on the pins.

- If bit set to 0 , Output on the pin is 0V.
- If bit set to 1 , Output on the pin is +5V.

- To write a specific pin we write:

  PORTX.1 = 1; Sets the +5V on the pin X.1

- To set values of all bits simultaneously , we write

  PORTA = 0b11001010

It only works when DDR value of that bit is set to 1

Let's Summarize…

6.3 PINX (Data Read Register)

This register is used to read the value of a PORT. If a pin is set as input then corresponding bit on PIN register is,

- 0 for Low Input that is V < 2.5V
- 1 for High Input that is V > 2.5V (Ideally, but actually 0.8 V - 2.8 V is error zone !)

For an example consider I have connected a sensor on PC4 and configured it as an input pin through DDR register. Now I want to read the value of PC4 whether it is Low or High. So I will just check 4th bit of PINC register.

We can only read bits of the PINX register; can never write on that as it is meant for reading the value of PORT.

Summary

- PINX ----> Read complete value of PORTX as a byte.
- PINX.y --> Read y^{th} pin of PORTX as a bit (works only with CVAVR).

6.1 DDRX (Data Direction Register)

First of all we need to set whether we want a pin to act as output or input. DDRX register sets this. Every bit corresponds to one pin of PORTX. Let's have a look on DDRA register.

Bit	7	6	5	4	3	2	1	0
PIN	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0

Now to make a pin act as I/O we set its corresponding bit in its DDR register.

- To make Input set bit 0
- To make Output set bit 1

If I write **DDRA = 0xFF** (0x for Hexadecimal number system) that is setting all the bits of DDRA to be 1, will make all the pins of PORTA as Output.

Similarly by writing **DDRD = 0x00** that is setting all the bits of DDRD to be 0, will make all the pins of PORTD as Input.

Now let's take another example. Consider I want to set the pins of PORTB as shown in table,

PORT-B	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
Function	Output	Output	Input	Output	Input	Input	Input	Output
DDRB	1	1	0	1	0	0	0	1

For this configuration we have to set DDRB as **11010001** which in hexadecimal is **D1**. So we will write **DDRB=0xD1**

Summary

- DDRX -----> to set PORTX as input/output with a byte.
- DDRX.y ---> to set yth pin of PORTX as input/output with a bit (works only with CVAVR).

6.2 PORTX (PORTX Data Register)

This register sets the value to the corresponding PORT. Now a pin can be Output or Input. So let's discuss both the cases.

6.2.1 Output Pin

If a pin is set to be output, then by setting bit 1 we make output **High** that is +5V and by setting bit 0 we make output **Low** that is 0V.

Let's take an example. Consider I have set DDRA=0xFF, that is all the pins to be Output. Now I want to set Outputs as shown in table,

PORT-A	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
Value	High(+5V)	High(+5V)	Low(0V)	Low(0V)	Low(0V)	High(+5V)	High(+5V)	Low(0V)
PORTA	1	1	0	0	0	1	1	0

For this configuration we have to set **PORTA** as **11000110** which in hexadecimal is **C6**. So we will write **PORTA=0xC6;**

6.2.2 Input Pin

If a pin is set to be input, then by setting its corresponding bit in PORTX register will make it as follows,

- Set bit 0 ---> Tri-Stated
- Set bit 1 ---> Pull Up

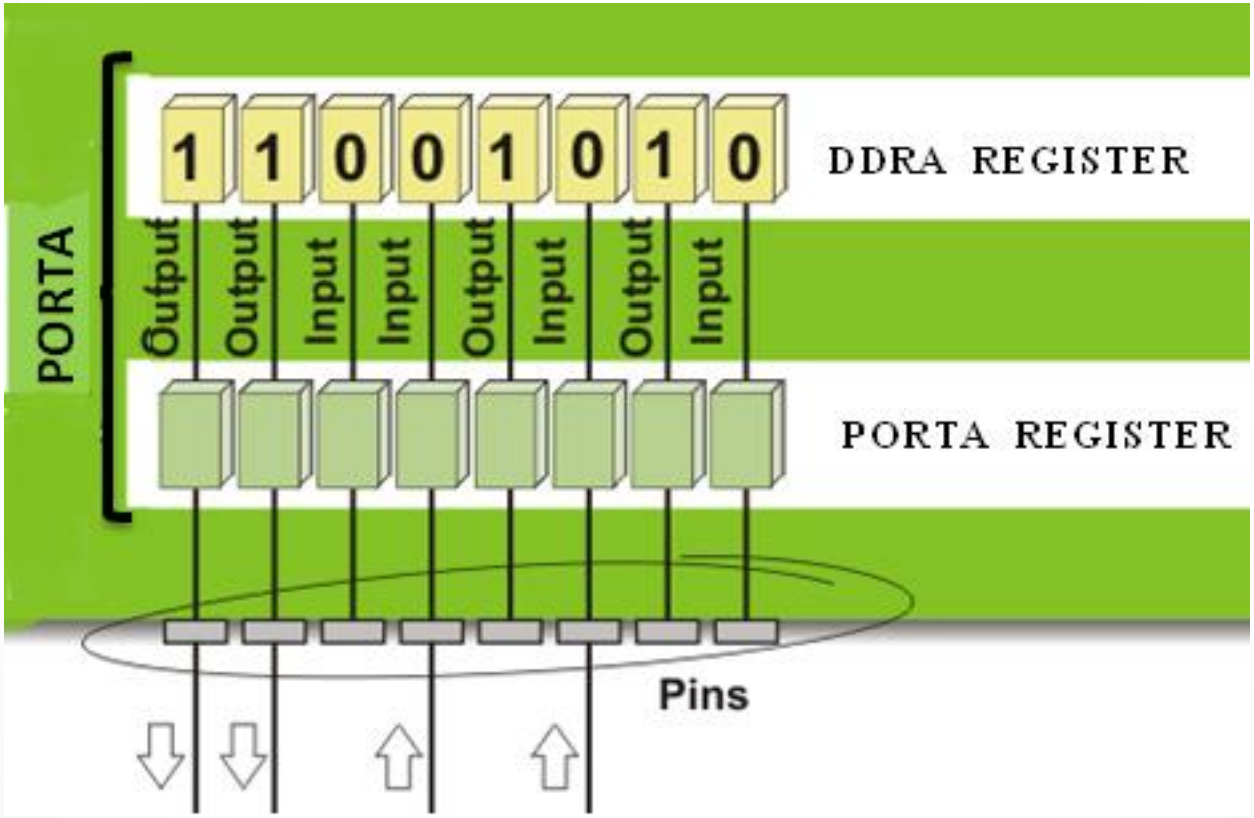
Tristated means the input will *hang* (no specific value) if no input voltage is specified on that pin.

Pull Up means input will go to +5V if no input voltage is given on that pin. It is basically connecting PIN to +5V through a 10K Ohm resistance.

Summary

- PORTX ----> to set value of PORTX with a byte.
- PORTX.y --> to set value of y^{th} pin of PORTX with a bit (works only with CVAVR).

Registers?



VISUALISATION OF REGISTERS

How do I program my µC?

I write my program on an IDE

I compile my program

I upload it to my µC

Where do I write my program?

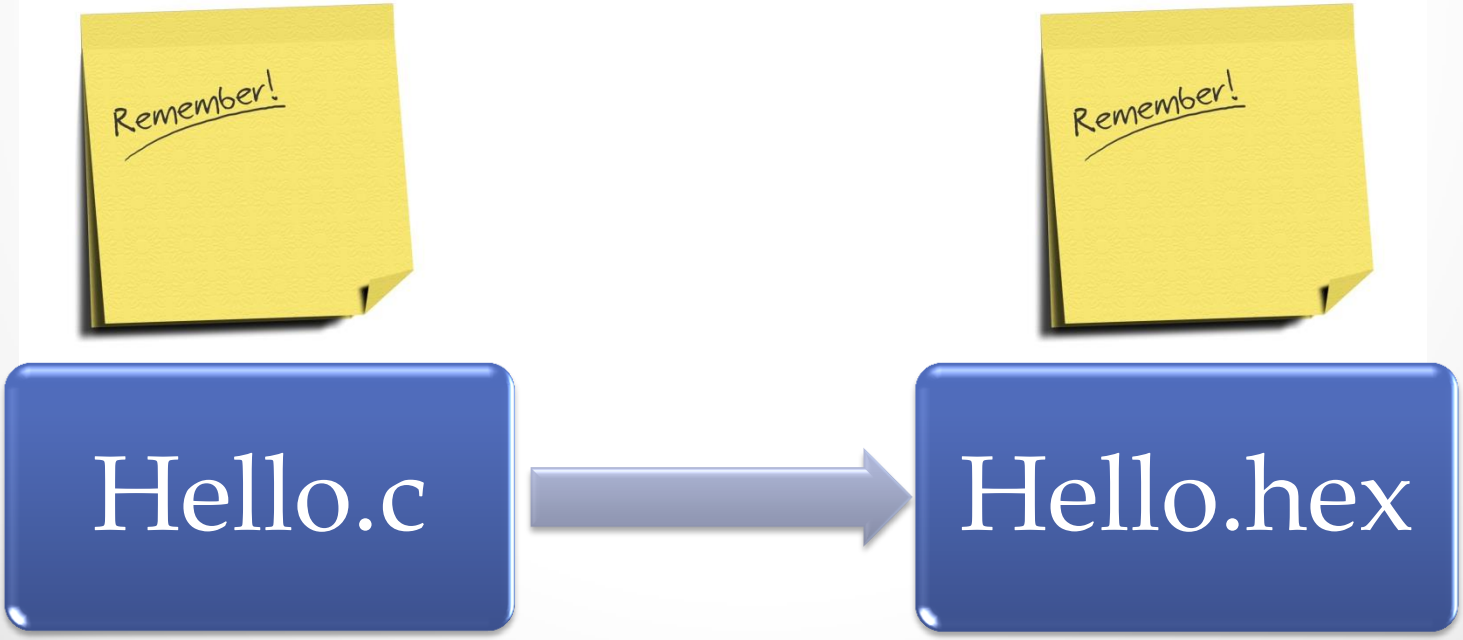
An IDE [Integrated Development Environment] capable of entering, editing and compiling my source program.

For our purpose of programming an Atmega, we will be using CV AVR

A variety of them are available according to your needs. You are familiar with many of them- Turbo C3, GCC, Eclipse, Dev C++ and more…

My IDE, My CV AVR

The code is written in C language and Atmega doesn't understand C, so it needs to be converted to a language that Atmega understands.



Now when we are aware of where to write our program, let us move on to How to write a program?

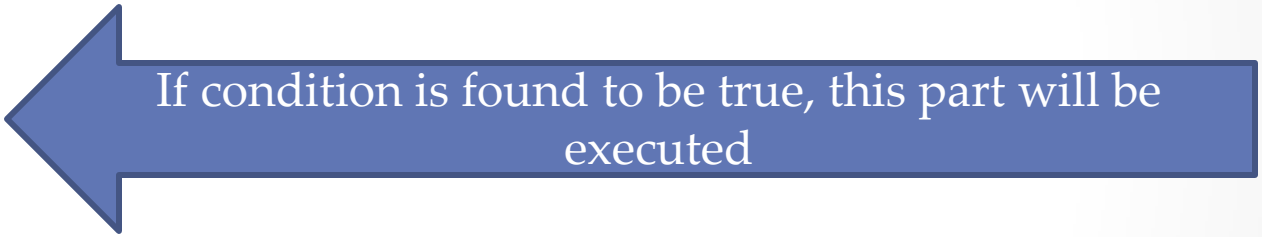
Basic digital arithmetic tools

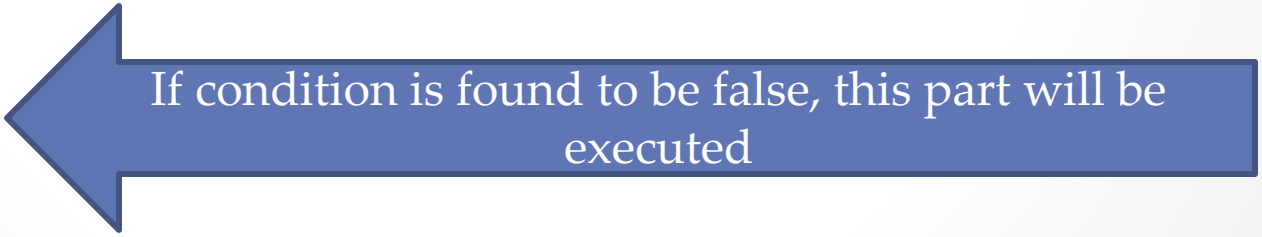
- OR "||" Eg. 0||0 = 0, 0||1 = 1, 1||0 = 1, 1||1 = 1
- AND "&&" Eg. 0&&0 =0, 0&&1 =0, 1&&0 =0, 1&&1= 1
- Bitwise OR "|" Eg. 10100111 | 11000101 = 11100111
- Bitwise AND "&" Eg. 10100111 & 11000101 = 10000101
- Bitwise NOT "~" Eg. ~10100110 = 01011001
- Left Shift "<<" Eg. <<11011011 = 10110110
- Right Shift ">>" Eg. >>11011011 = 01101101

Basics of C language

if-else block

```
If(condition)
{
…
}
else
{
…
}
```





Basics of C language

- **While**

```
while(condition)
{
…
}
```

Unless the condition is false, this part will be executed

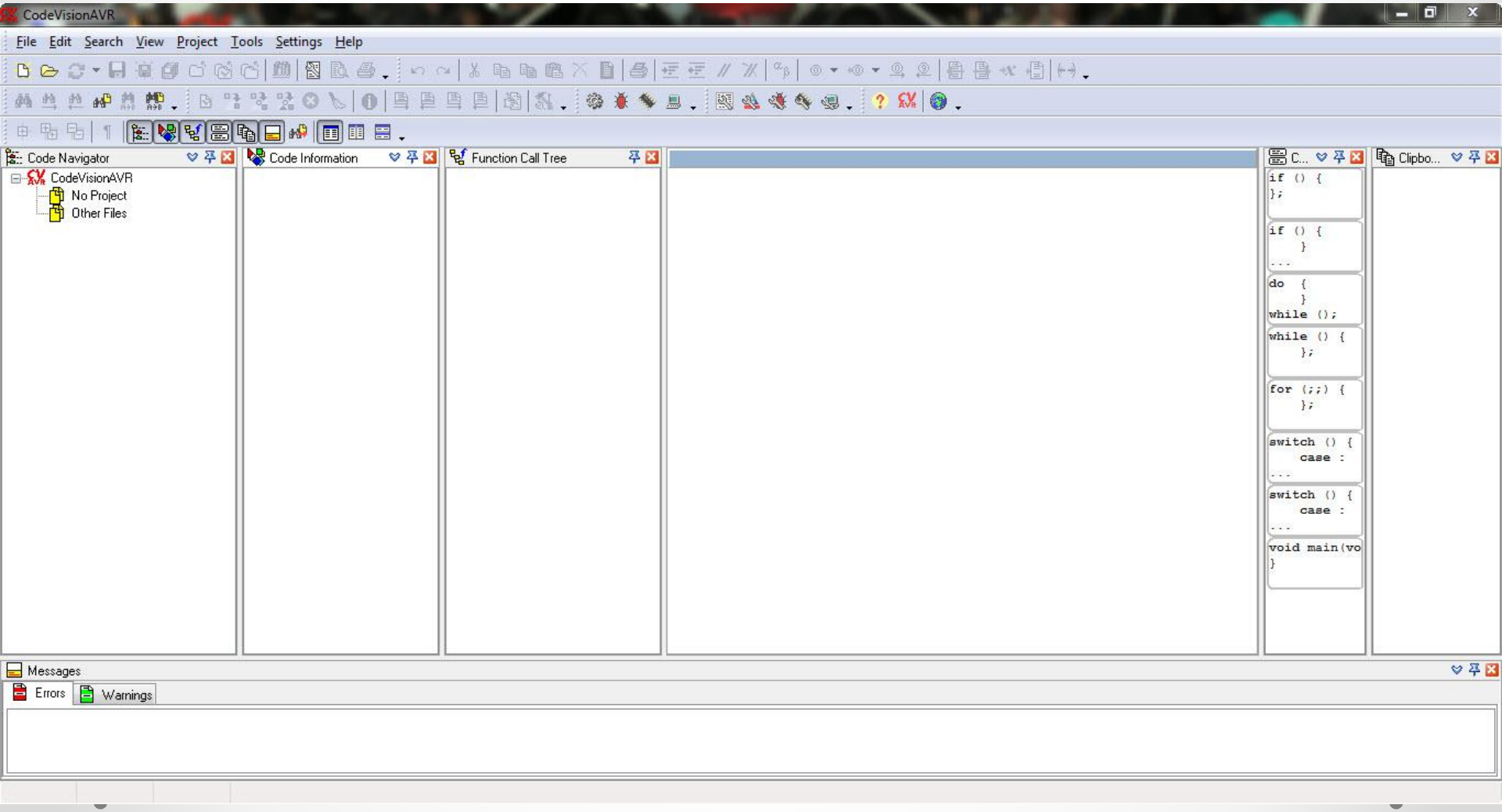
- **For**

```
for(initialisation; condition; increment)
{
…
}
```

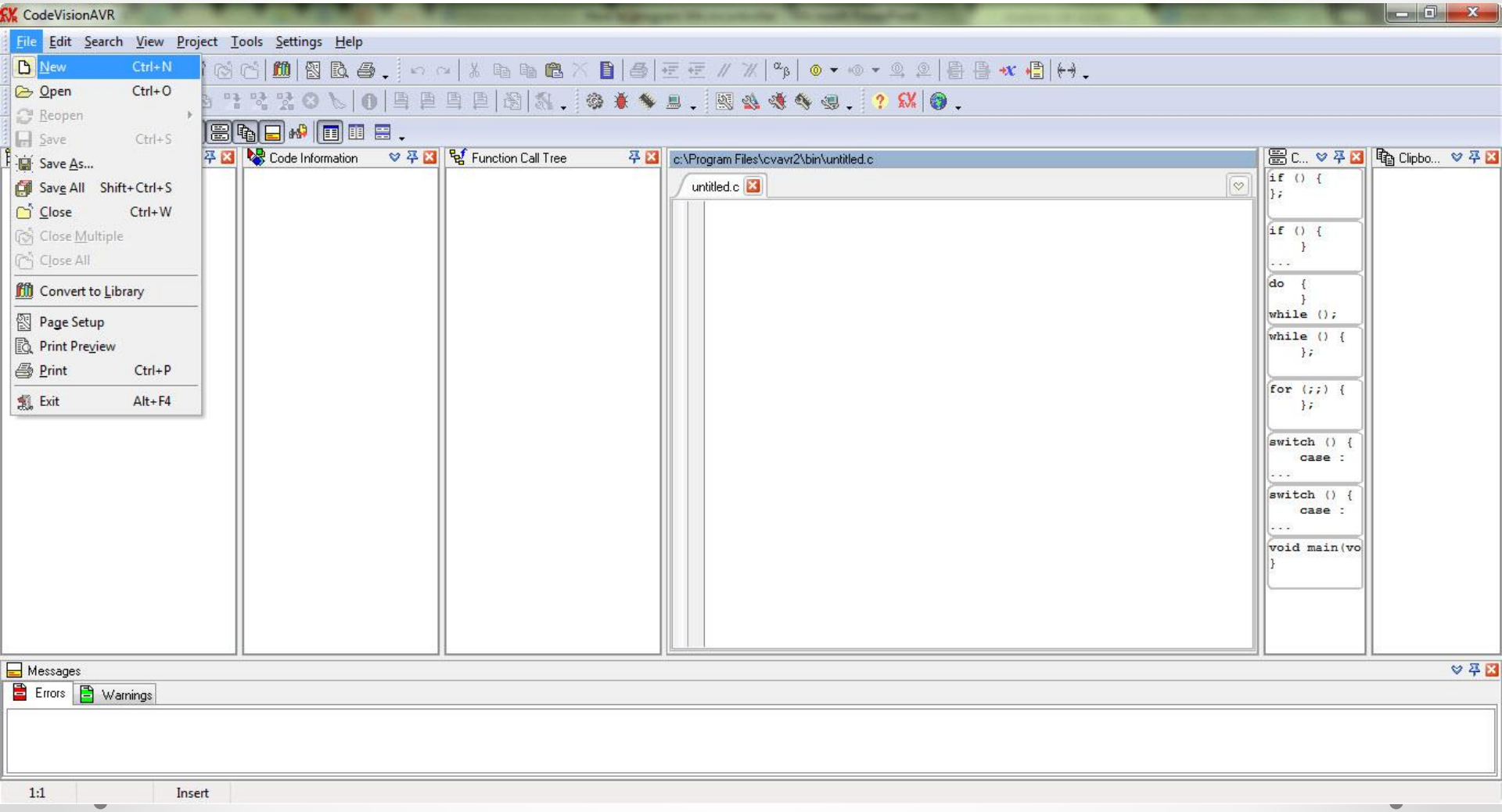
Unless the condition is false, this part will be executed and a variable incremented

Now when we are done with basic concepts of C, let's come back to CV AVR and familiarize ourselves with writing a code on CV AVR.

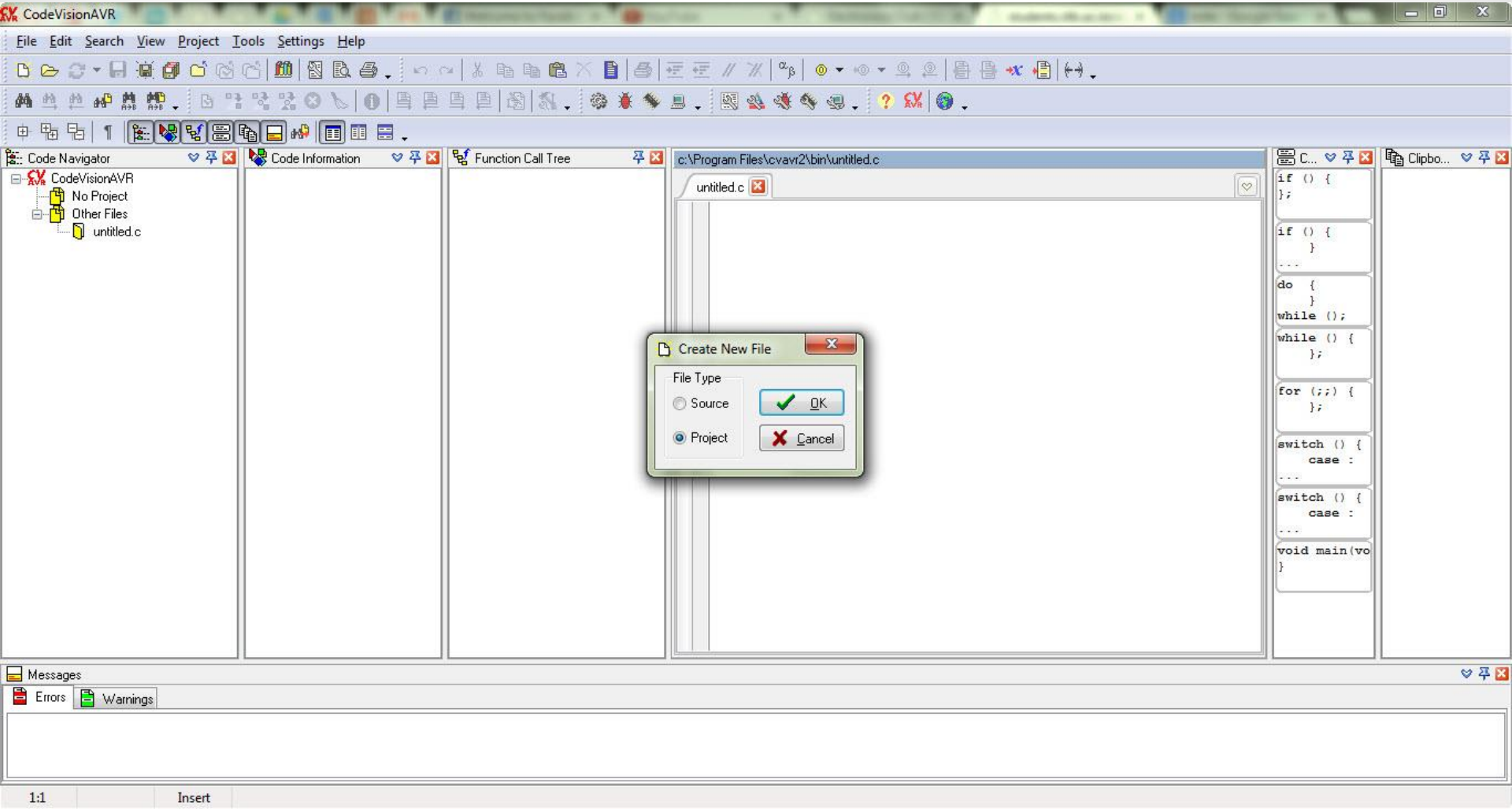
Open CV AVR


CodeVisionAVR
File Edit Search View Project Tools Settings Help
Code Navigator
CodeVisionAVR
No Project
Other Files
Code Information
Function Call Tree
C...
Clipbo...
if () {
};
if () {
}
do {
}
while ();
while () {
};
for (;;) {
};
switch () {
case :
switch () {
case :
void main(vo
}
Messages
Errors
Warnings

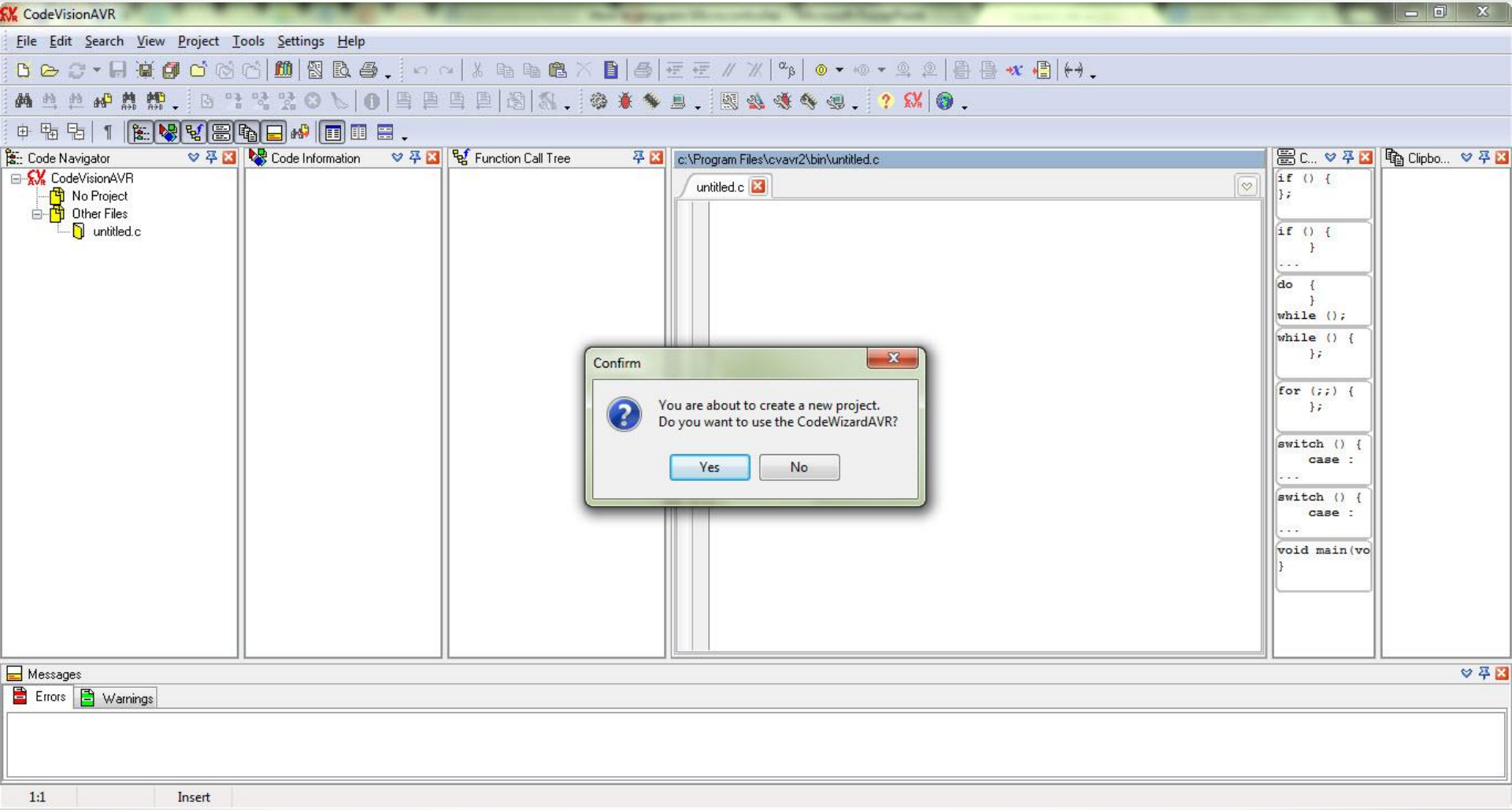
Click on “File” and then select “New”



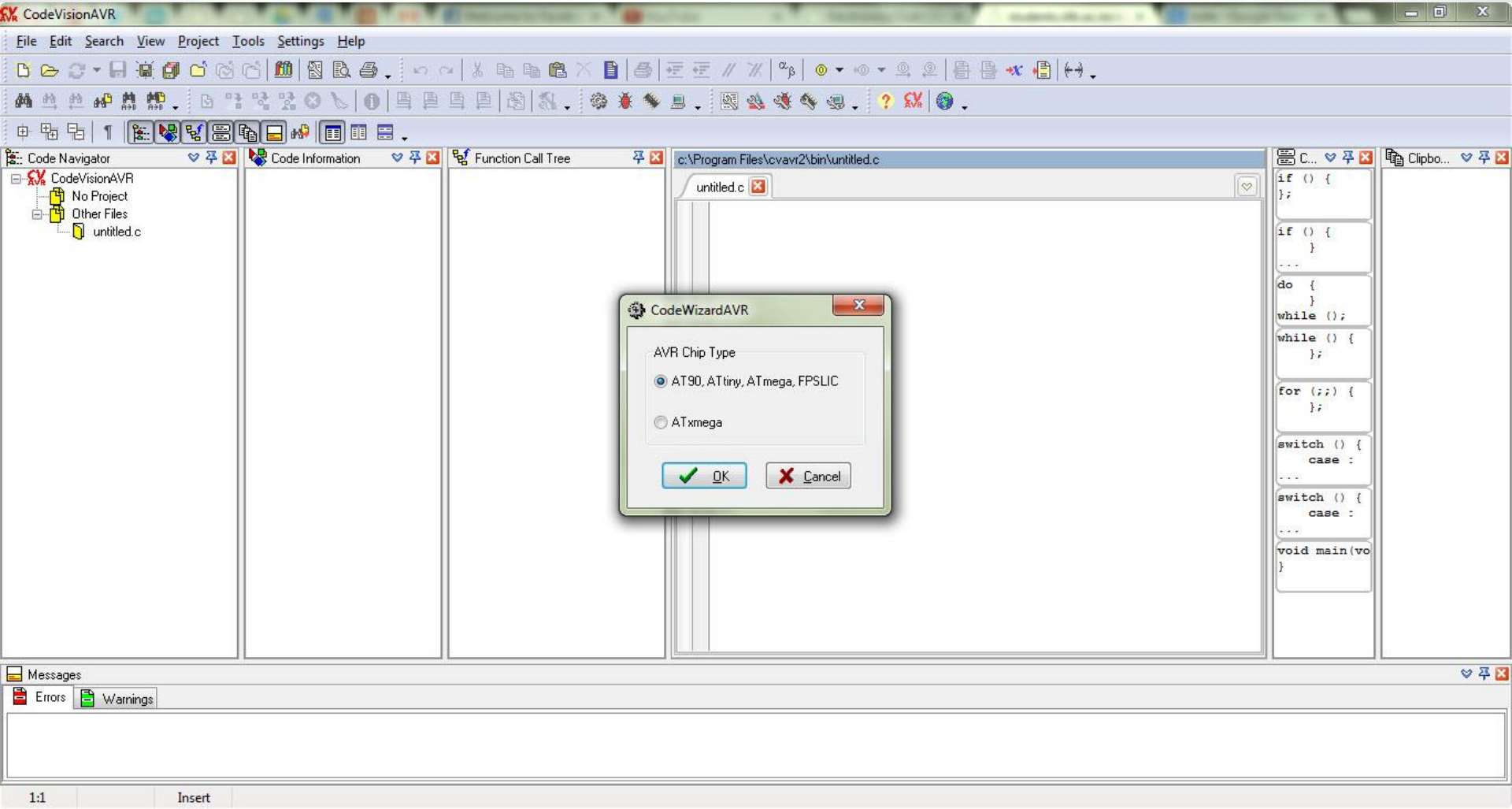
Select “Project” and Click “OK”



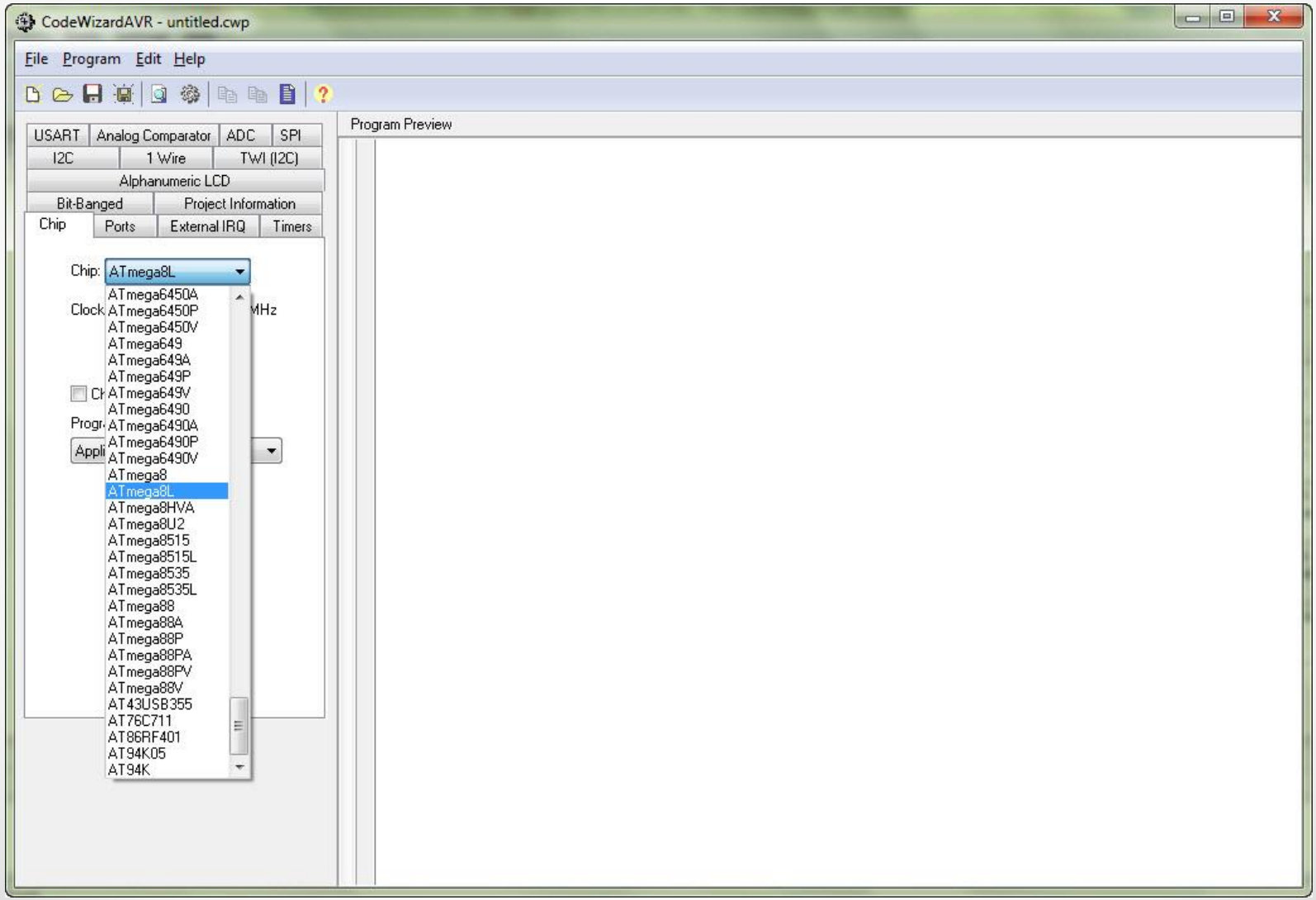
Do you want to use Code Wizard AVR? “Hell Yes!”



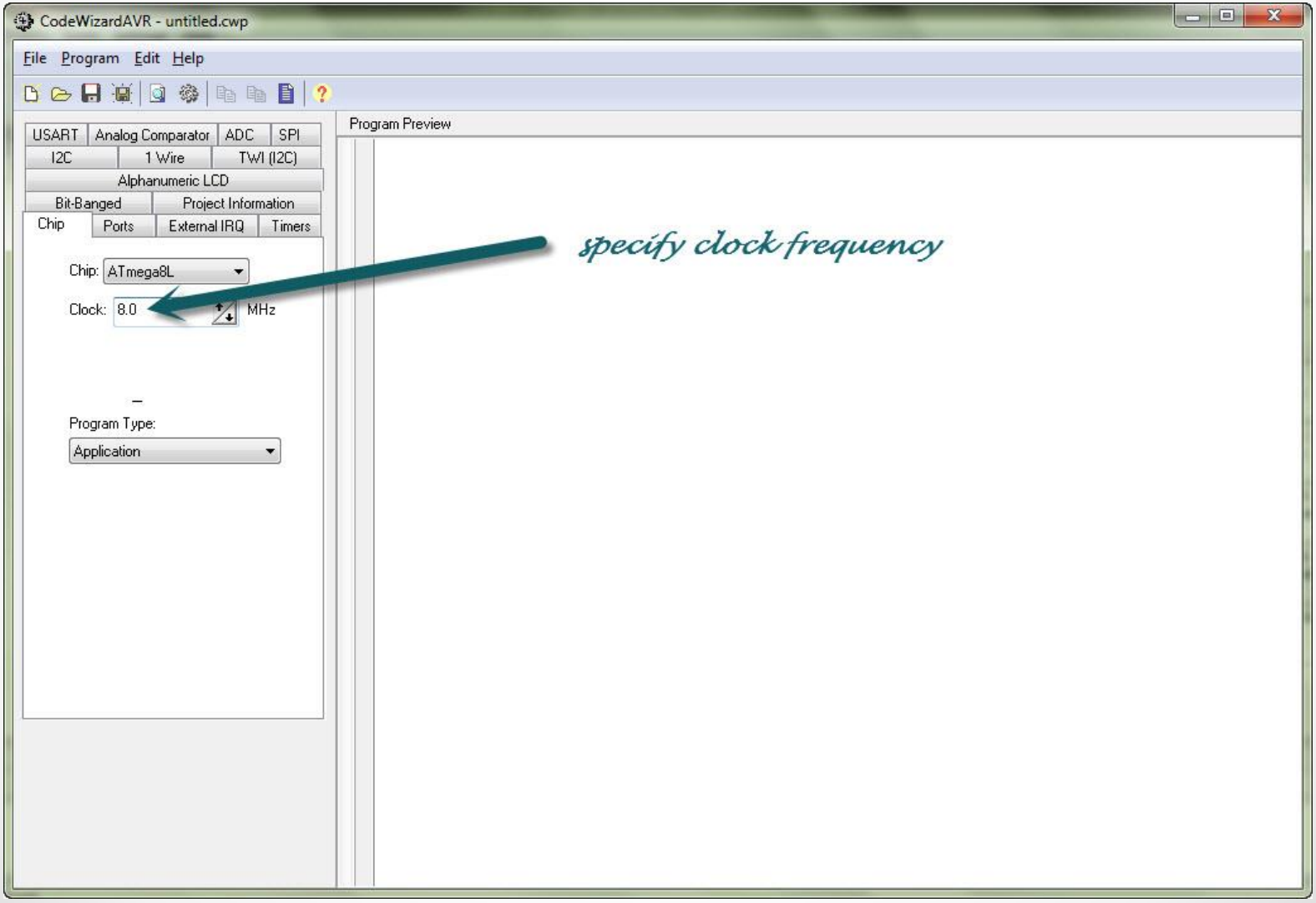
Select Chip Type



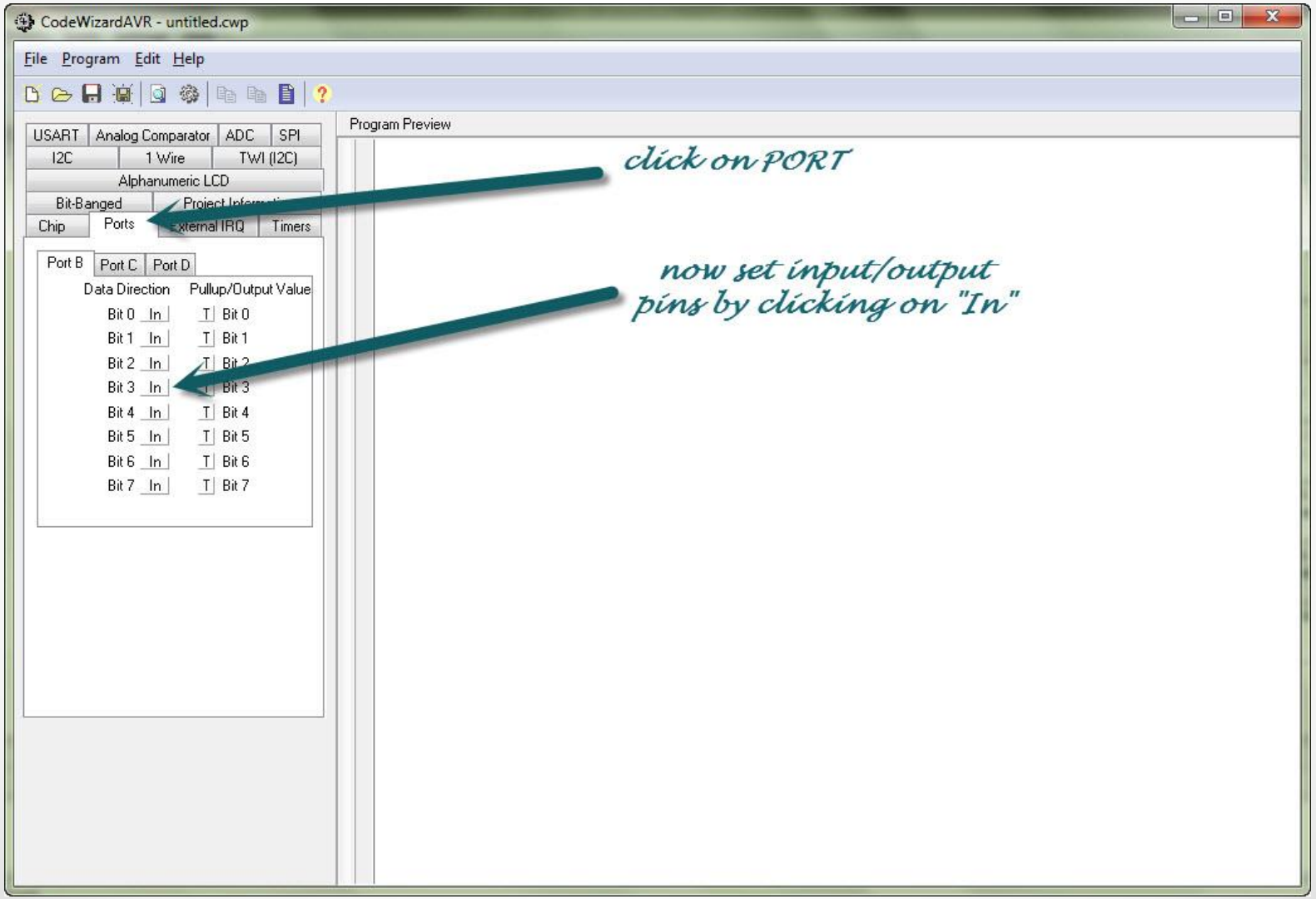
A new window will be opened, select chip



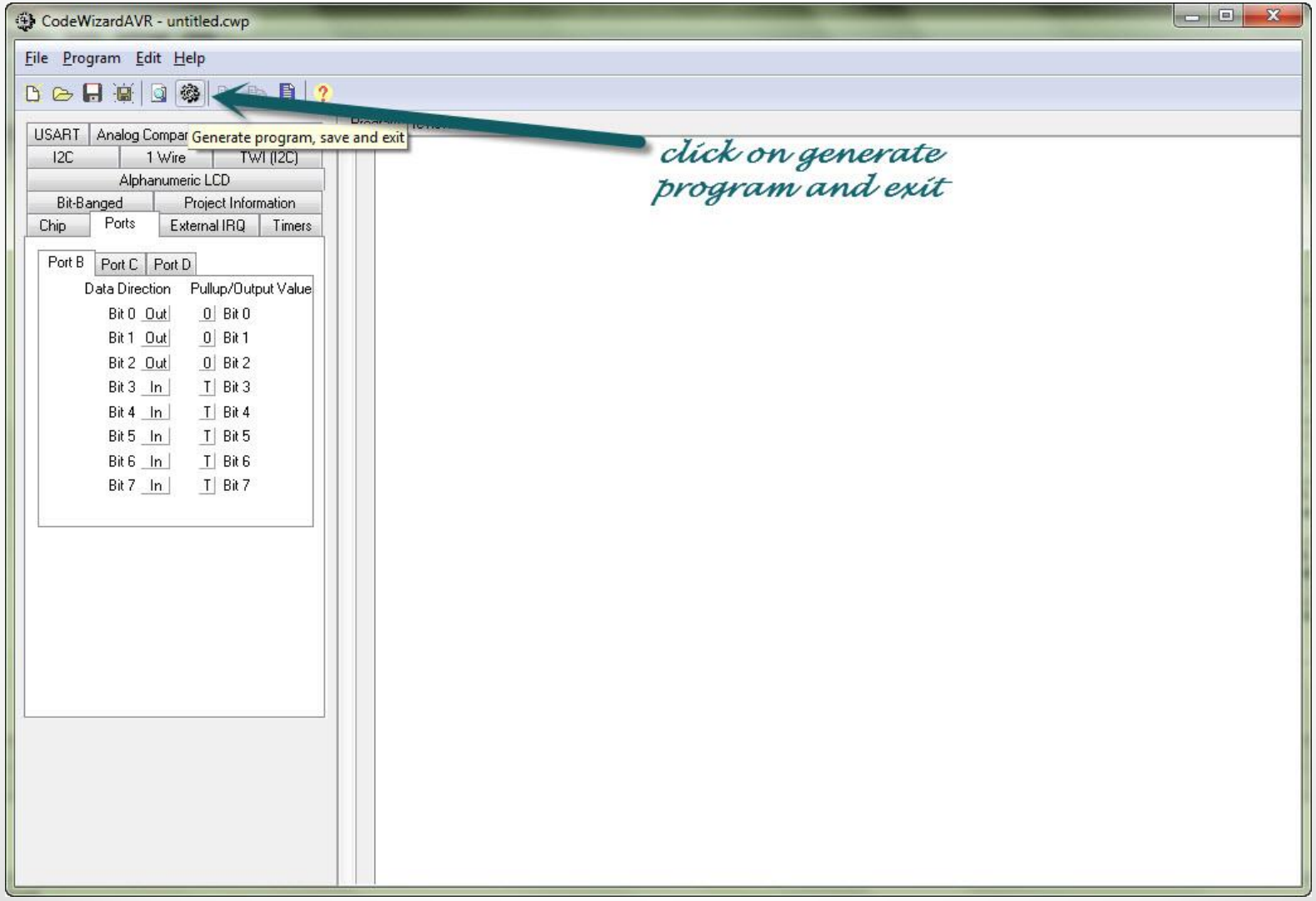
Set clock frequency



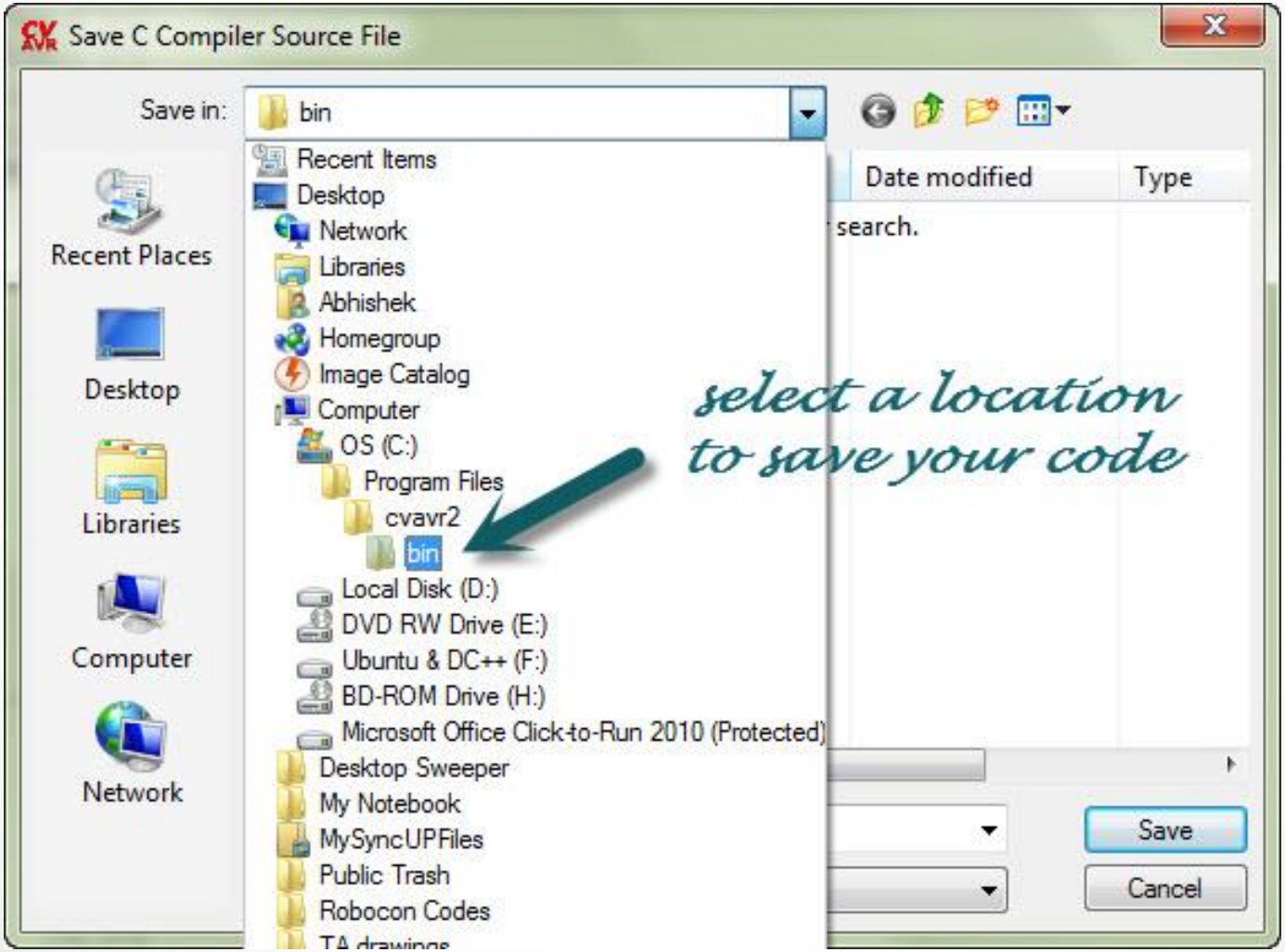
Setting I/O pins



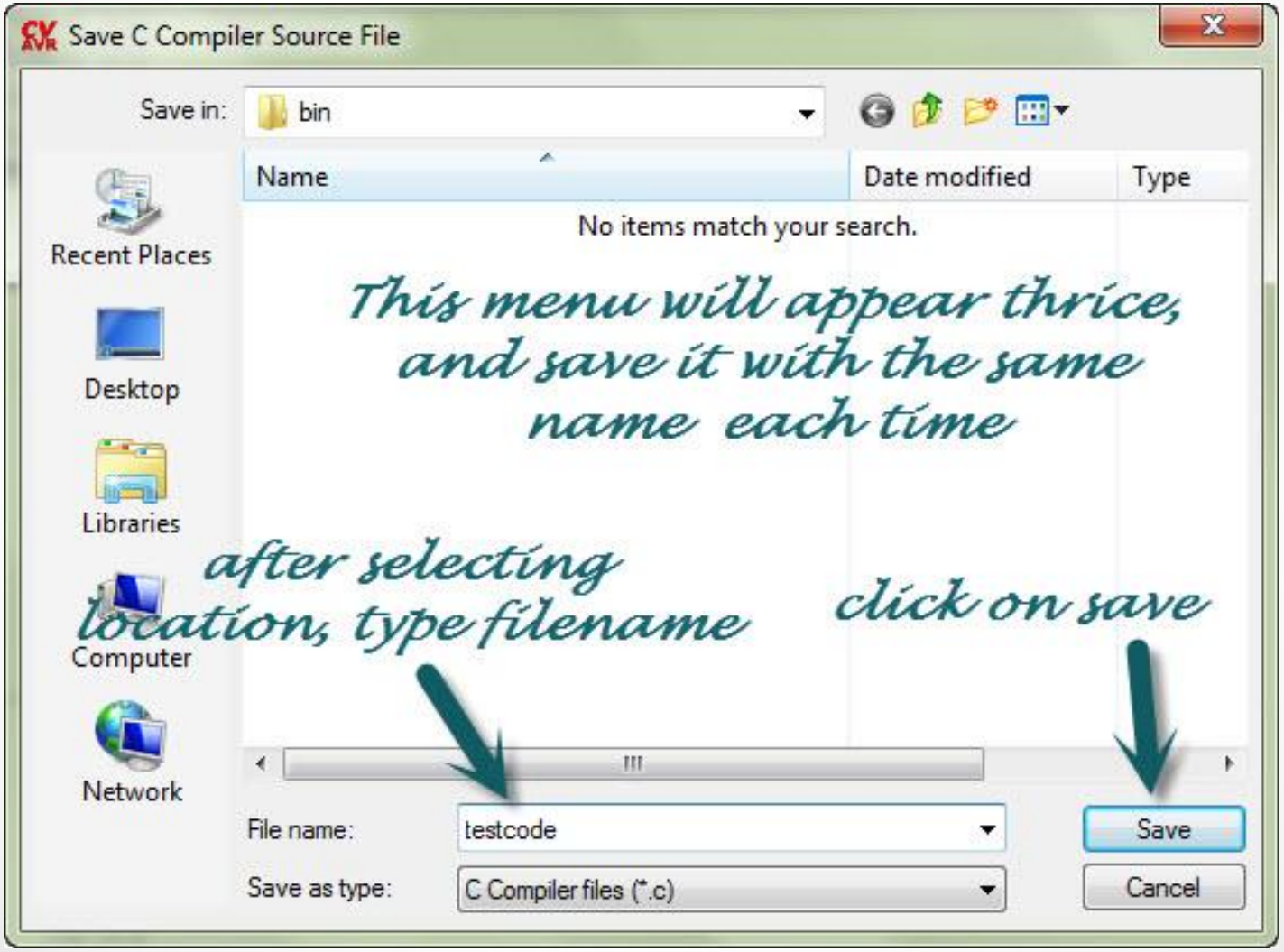
Generating program from your settings



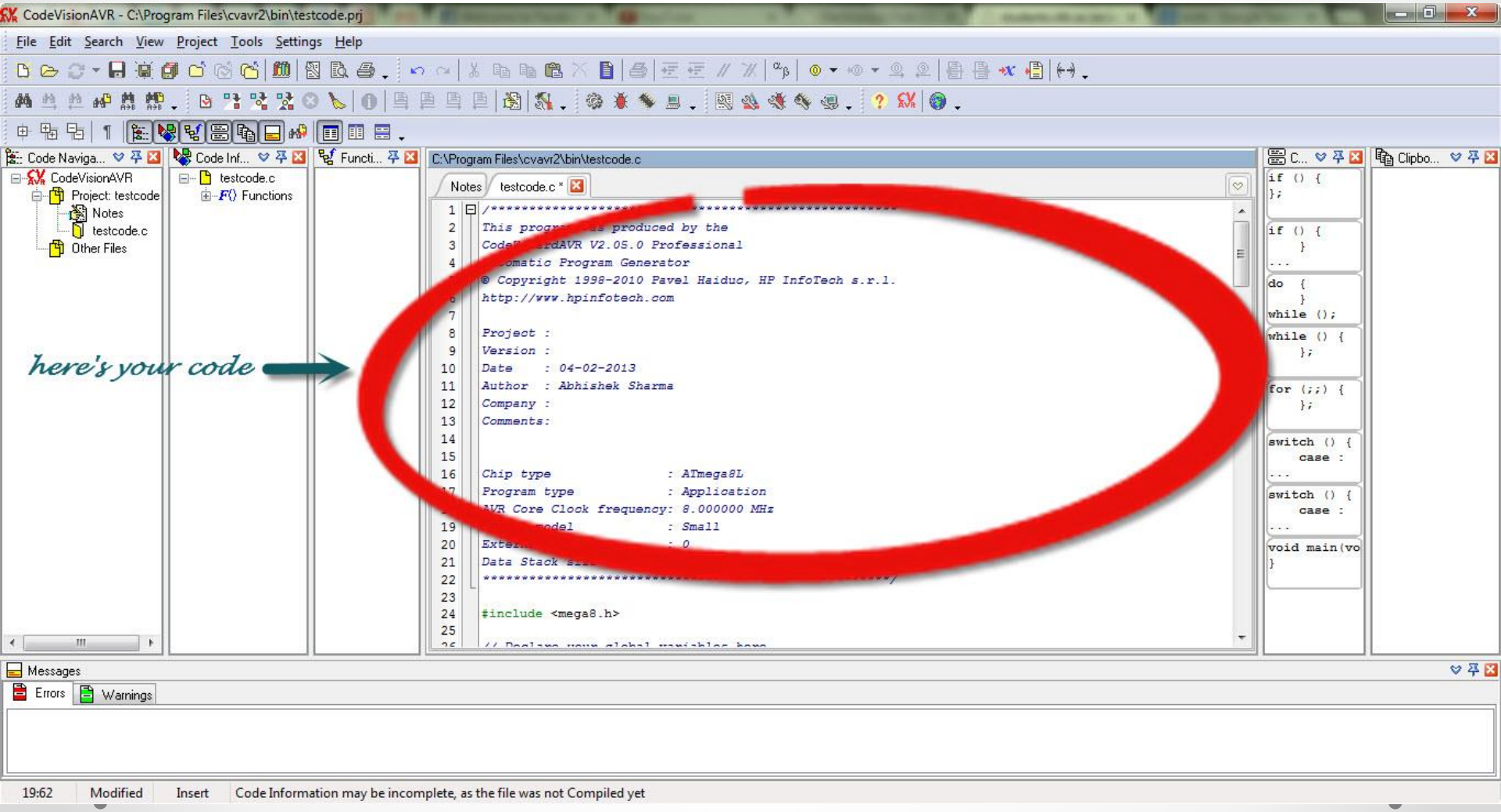
Select a location to save your program



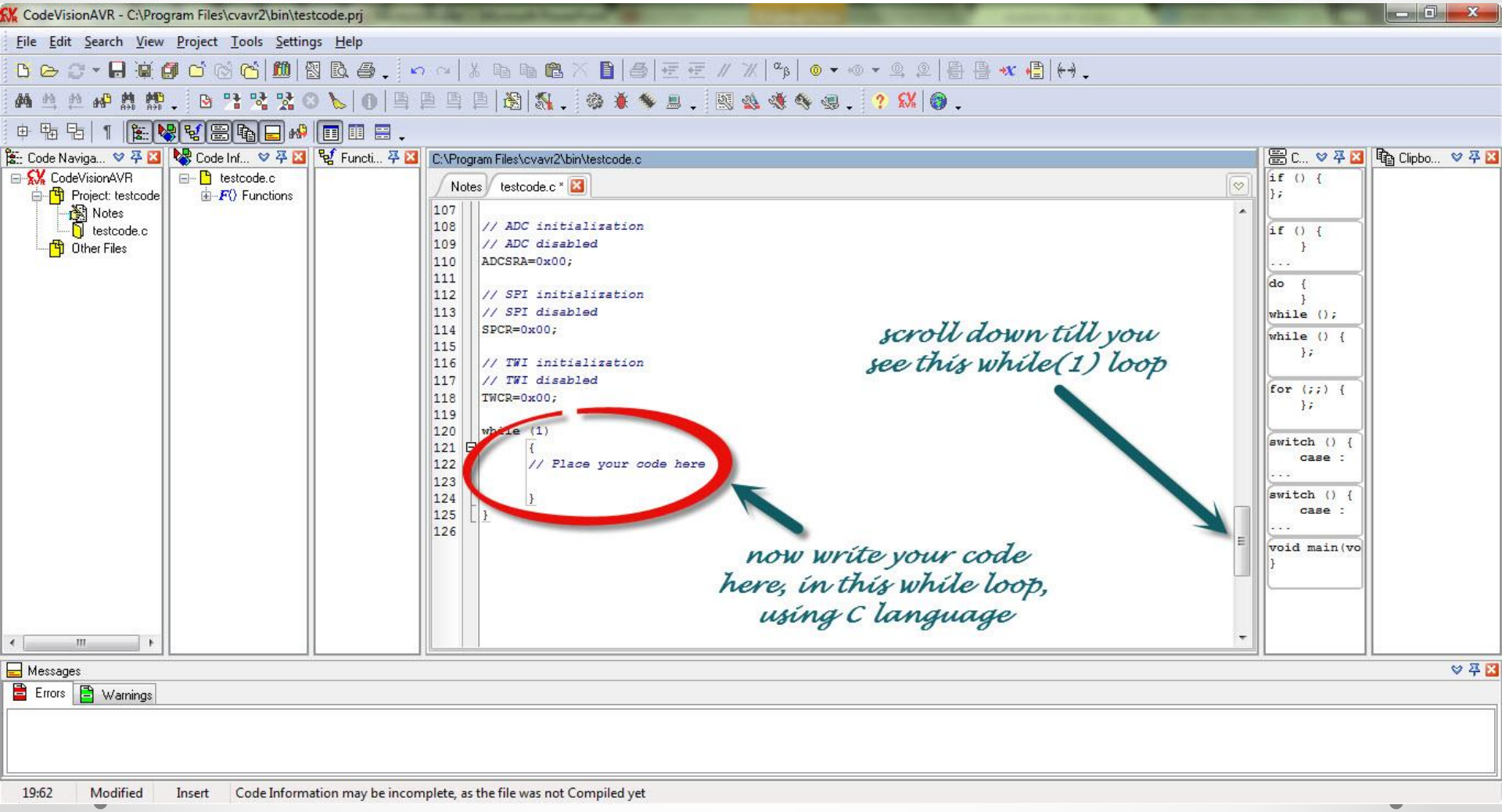
Save your file thrice with the same name



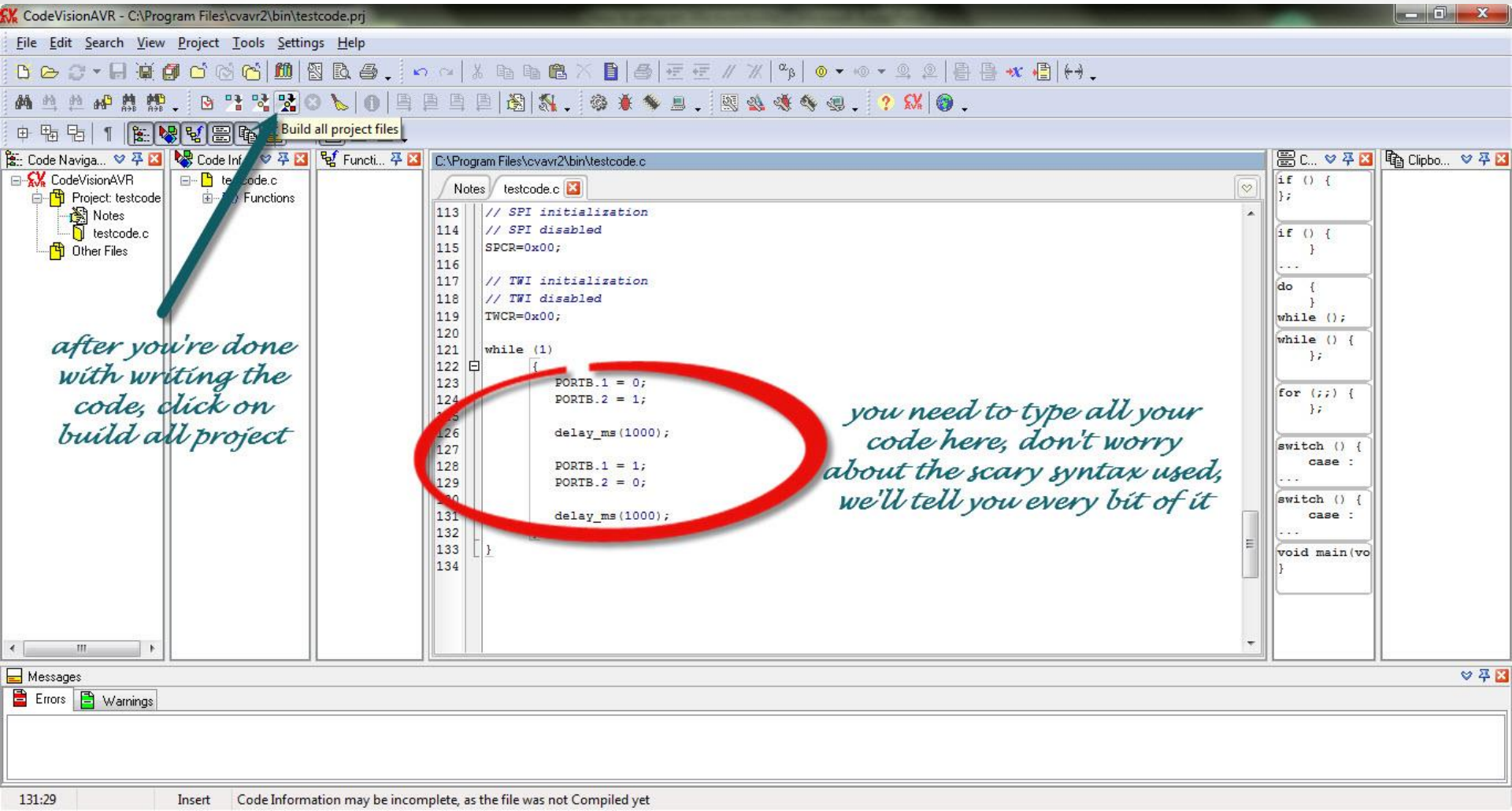
After saving, all pop-up windows will close and this is what you'll see -



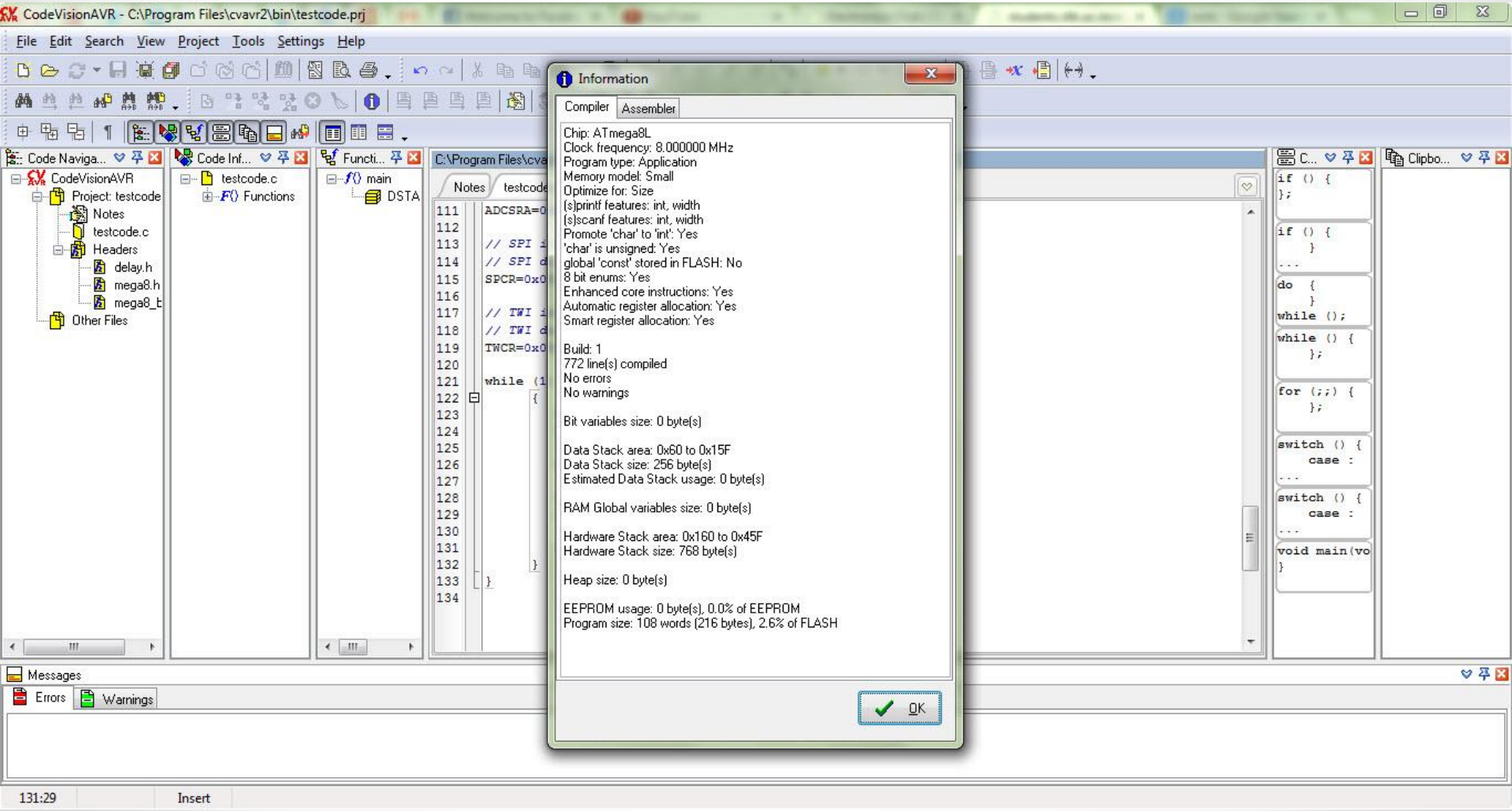
Where to write your code?



Writing and building your code

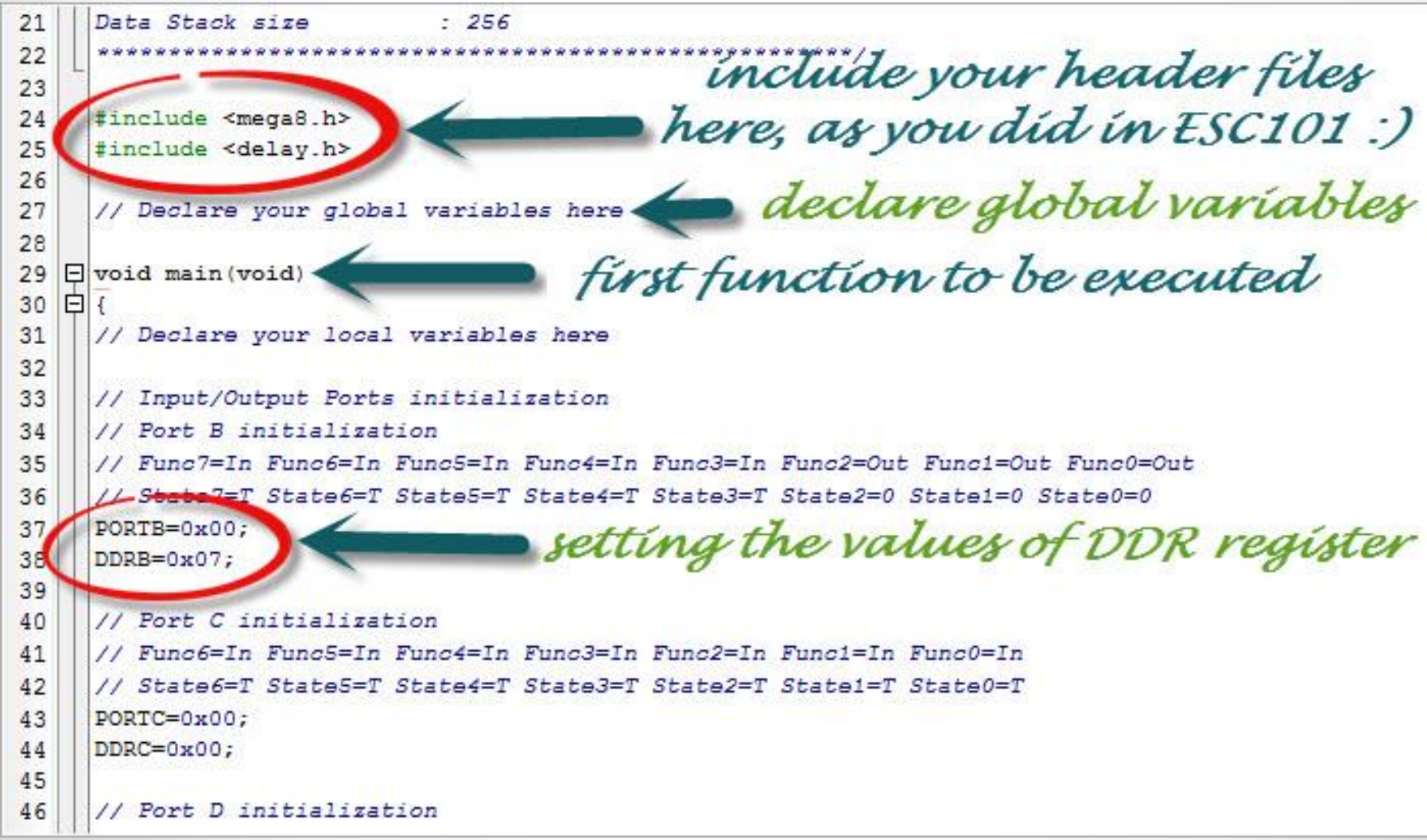


This is what you'll see after building your project files



Now when we have learnt how to use CV AVR, let's study our code in detail.

Inside story of the code



Inside story of the code

```
120
121  while (1)
122        {
123           PORTB.1 = 0;
124           PORTB.2 = 1;
125
126           delay_ms(1000);
127
128           PORTB.1 = 1;
129           PORTB.2 = 0;
130
131           delay_ms(1000);
132        }
133  }
134
```

Now let's see what code have we written!

Scary Syntax? ☹

PORTB.1

This is used to give an output voltage to B.1 pin

PORTB.1 = 0 for 0 Volts

PORTB.1 = 1 for 5 Volts

delay_ms(2000);

This is used to pause code execution for 2000 milliseconds,

PINC.3

This is used to take input from C.3 pin

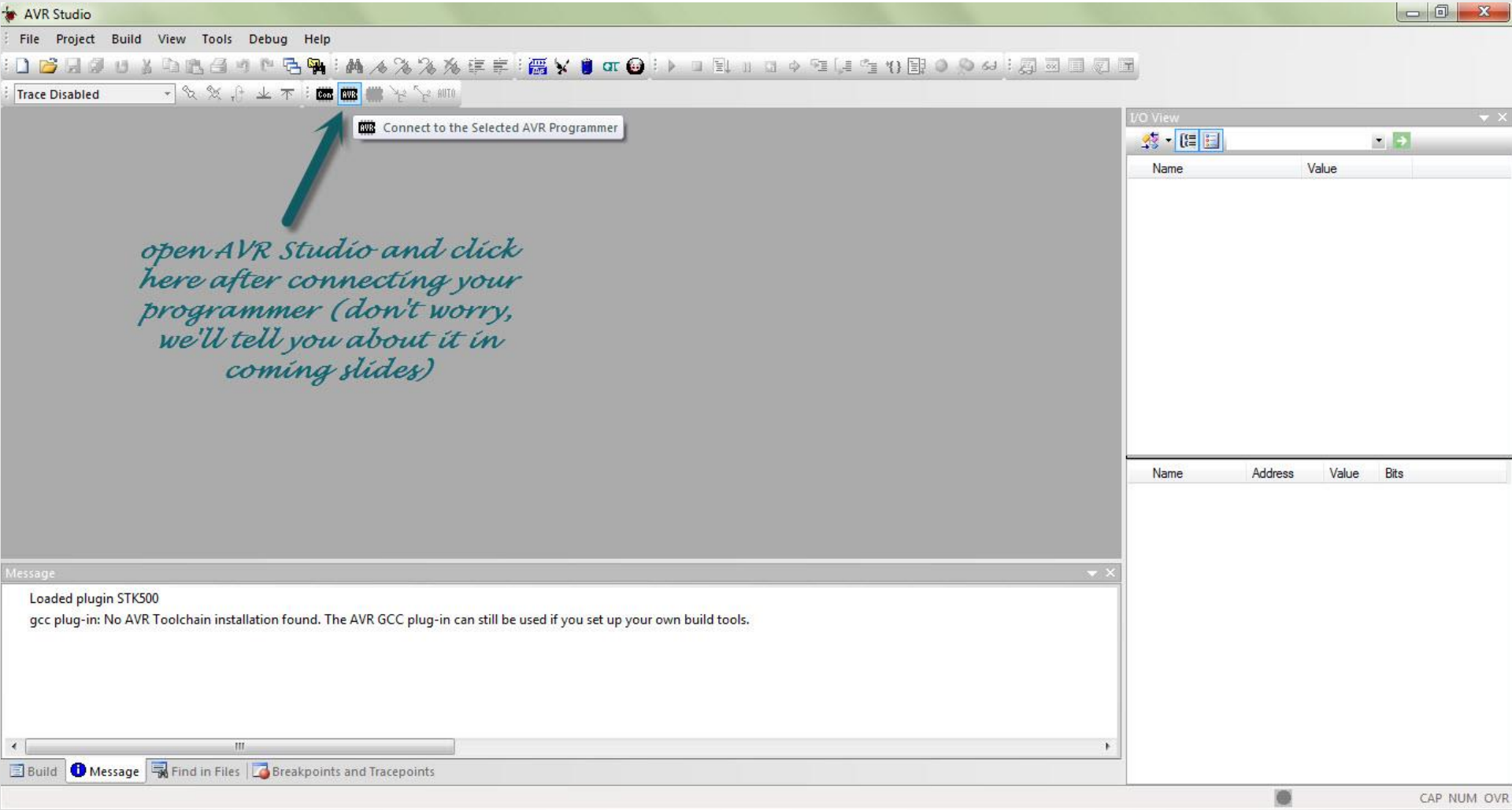
int a = PINC.3;

will put a=0 if we give 0 volts on pin C.3 and will put a=1 if we give 5 volts on pin C.3

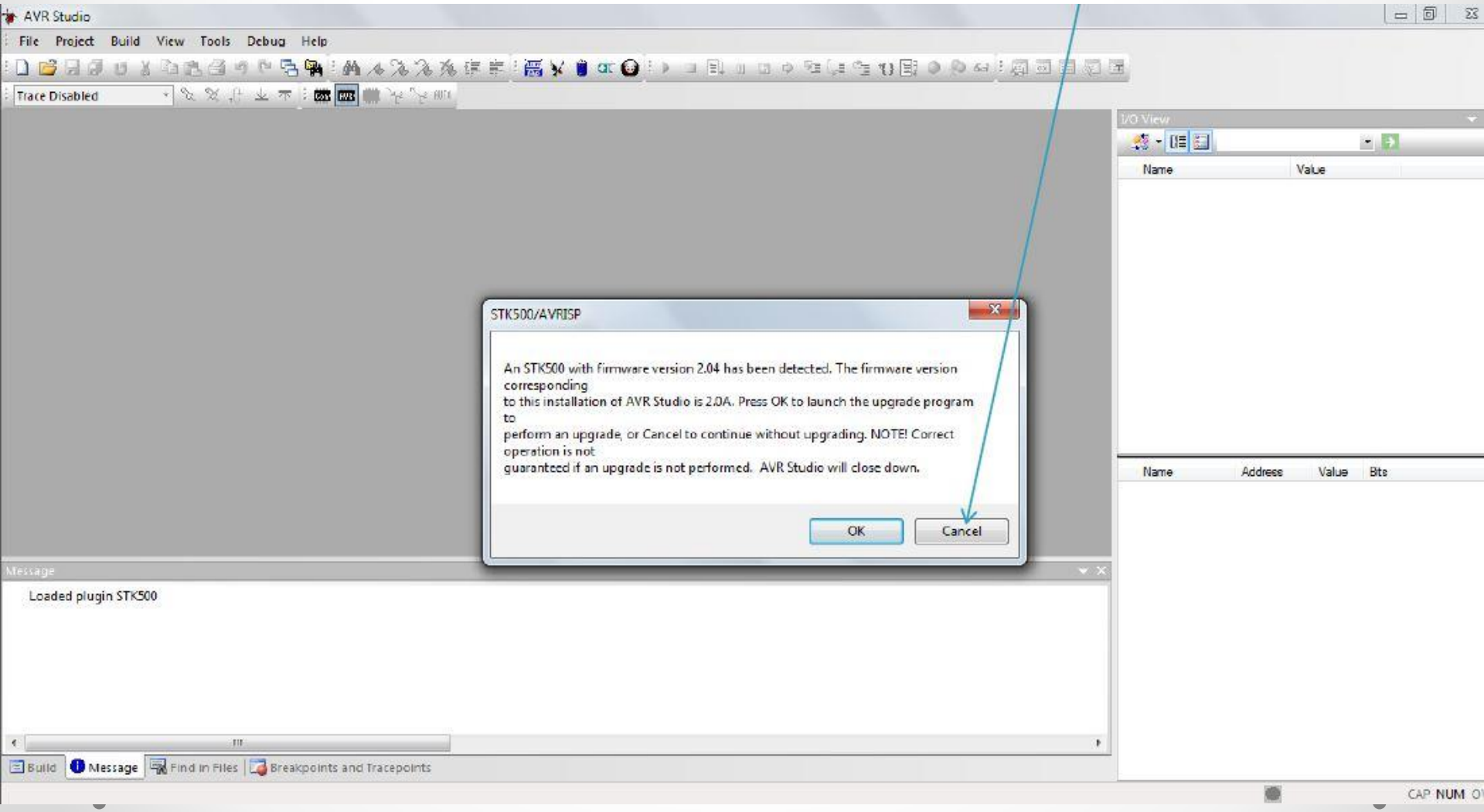
Now when we have learnt to write our code, let's learn how to transfer our code to the microcontroller (our Atmega-8)

We'll tell you how to upload your code to the Atmega using AVR Studio-4

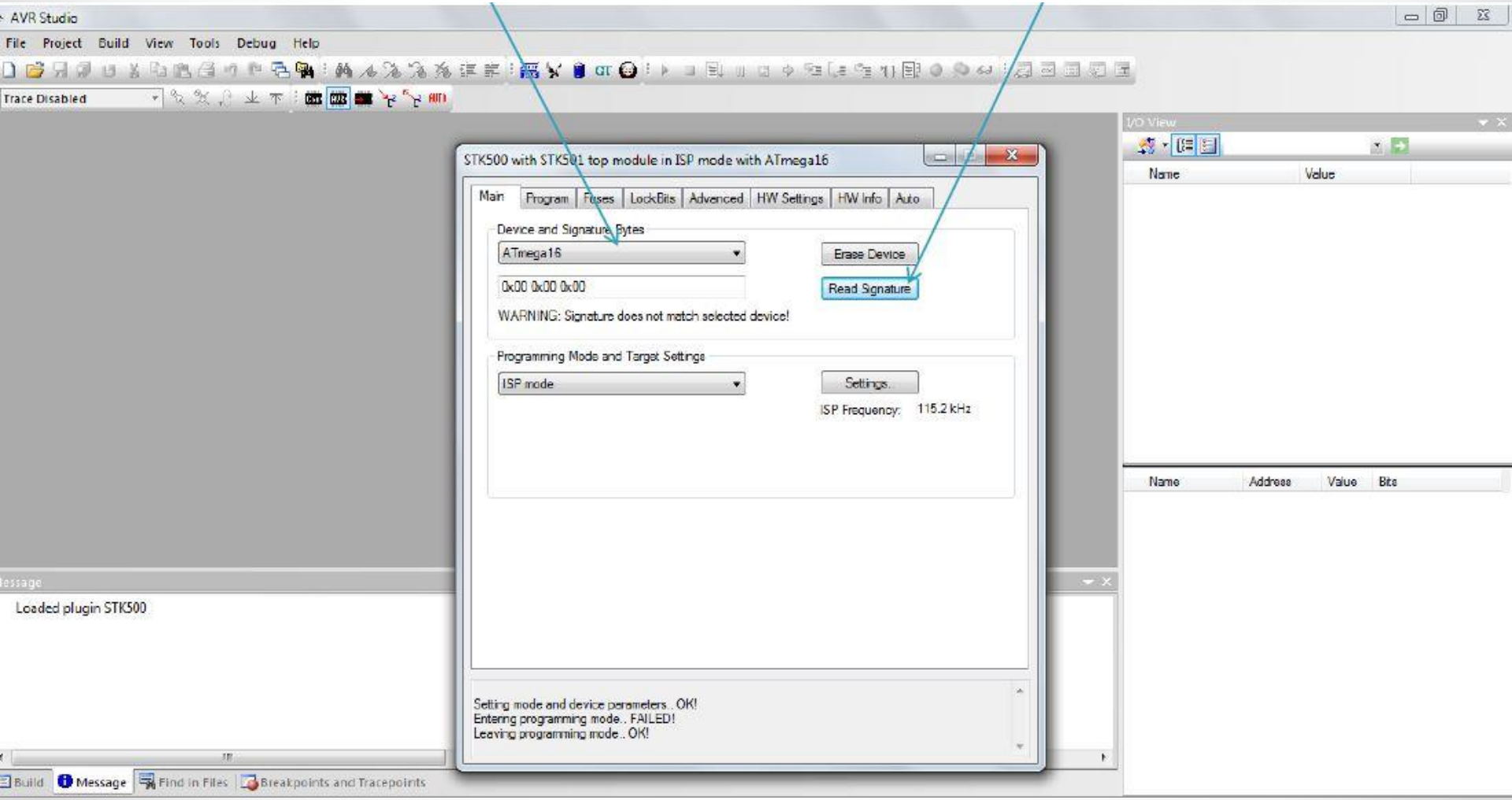
Open AVR Studio-4



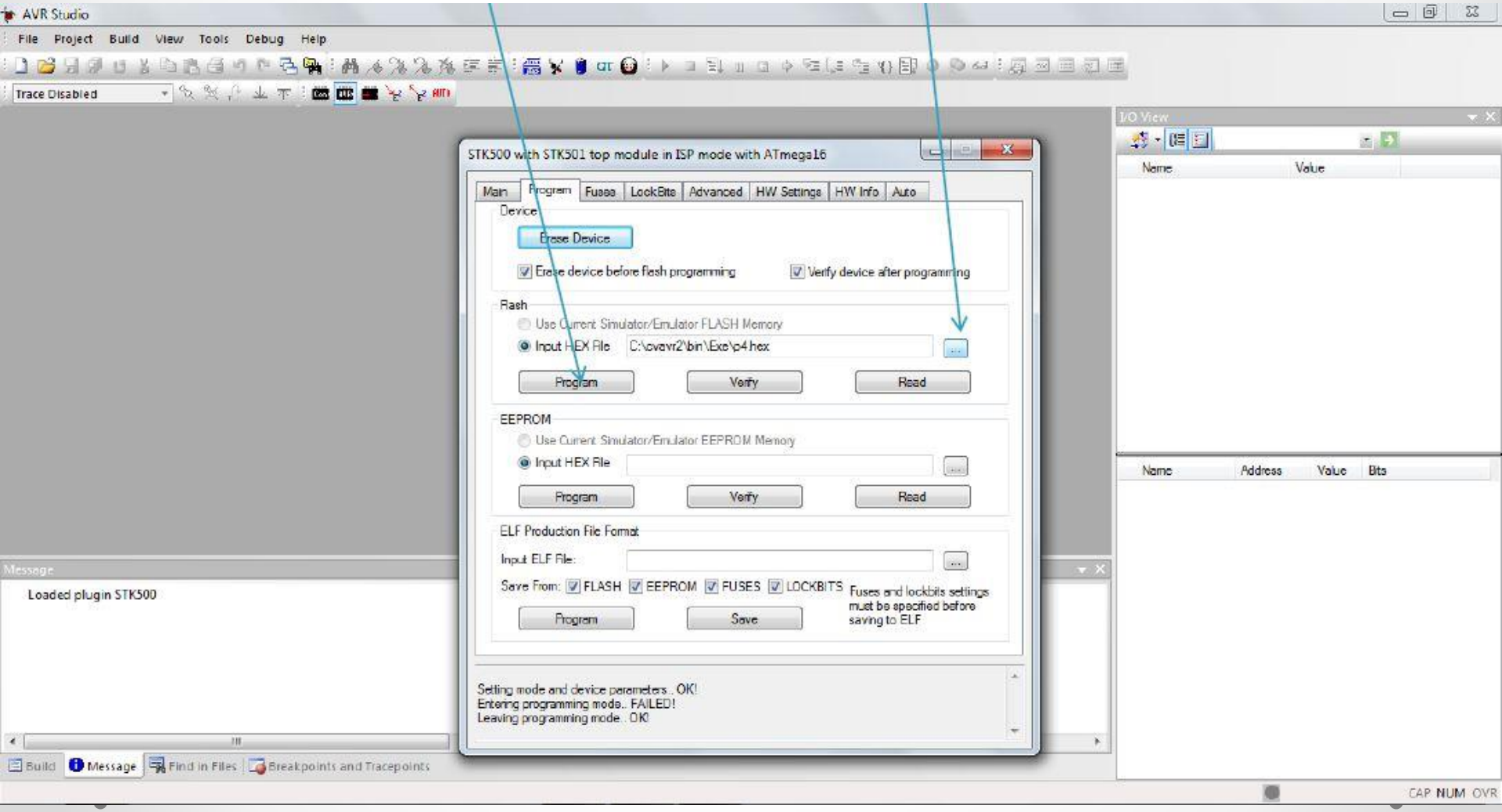
Click Cancel as indicated



Now select your µC and click on “Read Signature”



Now browse your .hex file and click program

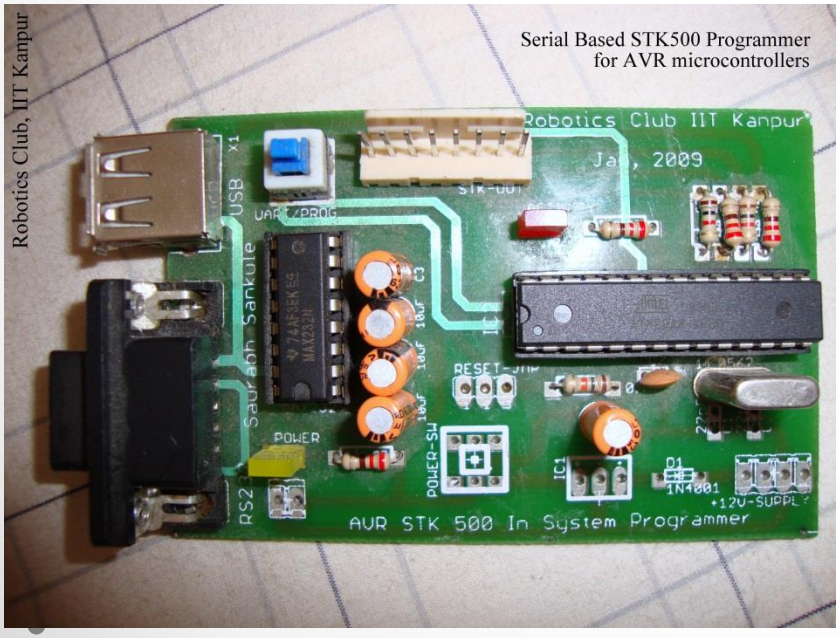


That's it!

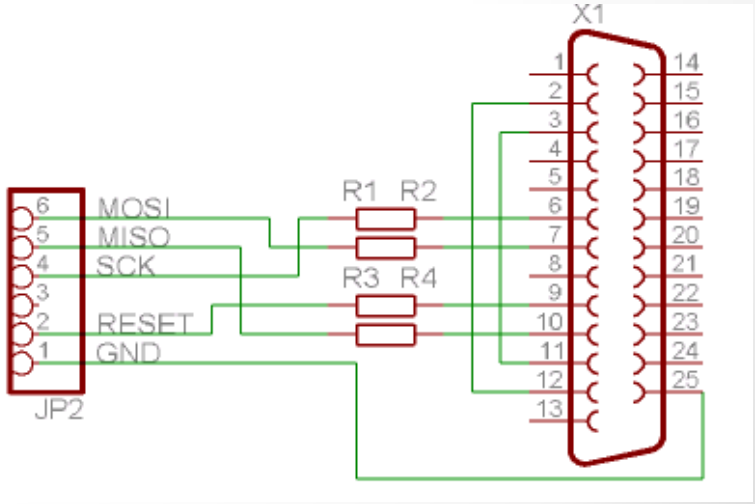
You've now successfully uploaded the program to your Atmega-8

Man! We don't know anything about a programmer.

Ok! Here it is, this is how it looks like:



And here's how we connect our Atmega to the programmer:



Thank you!