ATMEGA 128 Development Board
June 2010
ATMEGA 128 Development Board

Introduction
ATMEGA128 is a big and one of the most feature rich AVR microcontroller from Atmel featuring 128K Flash, 4K RAM, 53 I/O lines arranged in seven 8 bit ports, 8 ADCs, 2 UARTs, 4 timers, 8 interrupts and much more. ATMEGA128 Development Board exposes all the pins of the microcontroller in neat PORT wise configuration. It is made from double sided PTH PCB board to provide extra strength to the connector joints for increased reliability. Board can work on 7 to 15V AC or DC supply. It has built-in reverse polarity protection. 7805 voltage regulator has heat sink for heat dissipation so that it can supply 1Amp current continuously without getting over heated. It has switches for boot loading, reset and power. It also has RS232 interface with DB9 female connector based on MAX232. All the ports are connected to standard 10 pin FRC connectors.

Specifications
- Microcontroller: ATMEGA128 with 14.7456MHz crystal (Also supports ATMEGA640)
- Double side high quality PTH PCB for added strength.
- Power: 7 to 15V, AC or DC, Heat sink on 7805 for better current rating
- Reverse polarity protected
- Switches: Boot, Reset, Power
- RS232 serial interface
- 10 pin FRC connectors for all ports
- Compatible with General purpose prototyping board for development board for stackable design
- Application examples in AVR studio provided in the documentation CD

Package contains
- ATMEGA 128 Development Board with ATMEGA 128 microcontroller
- Documentation CD containing tutorial for ATMEGA 128 microcontroller programming in AVR studio and application examples.
Documentation CD contain following Application examples

- I/O Port operation
- Interrupt control
- Timer to generate accurate delay
- Timer as external Counter
- PWM (Pulse Width Modulation) for power control
- ADC (Analog to Digital Conversation)
- UART (Serial Communication)
- LCD (Print string of data on LCD)

**Warning:**
Do not give Power supply more than 15 V AC/DC.

**Note:**
ATMEGA 128 Development Board schematic is in the documentation CD

**PORT FRC connector pin connections**

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Pin Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pin 0 of PORT X</td>
</tr>
<tr>
<td>2</td>
<td>Pin 1 of PORT X</td>
</tr>
<tr>
<td>3</td>
<td>Pin 2 of PORT X</td>
</tr>
<tr>
<td>4</td>
<td>Pin 3 of PORT X</td>
</tr>
<tr>
<td>5</td>
<td>Pin 4 of PORT X</td>
</tr>
<tr>
<td>6</td>
<td>Pin 5 of PORT X</td>
</tr>
<tr>
<td>7</td>
<td>Pin 6 of PORT X</td>
</tr>
<tr>
<td>8</td>
<td>Pin 7 of PORT X</td>
</tr>
<tr>
<td>9</td>
<td>5V 1A output of 7805 voltage regulator</td>
</tr>
<tr>
<td>10</td>
<td>Ground</td>
</tr>
</tbody>
</table>
Power Port pin connections
Power port is used to give power to the accessories boards compatible to the microcontroller development board

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Pin Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5V 1A output of 7805 voltage regulator</td>
</tr>
<tr>
<td>2</td>
<td>5V 1A output of 7805 voltage regulator</td>
</tr>
<tr>
<td>3</td>
<td>Ground</td>
</tr>
<tr>
<td>4</td>
<td>Ground</td>
</tr>
<tr>
<td>5</td>
<td>Vin DC (Input supply – 1.2V DC drop due to bridge rectifier)</td>
</tr>
<tr>
<td>6</td>
<td>Vin DC (Input supply – 1.2V DC drop due to bridge rectifier)</td>
</tr>
<tr>
<td>7</td>
<td>NC</td>
</tr>
<tr>
<td>8</td>
<td>NC</td>
</tr>
<tr>
<td>9</td>
<td>NC</td>
</tr>
<tr>
<td>10</td>
<td>NC</td>
</tr>
</tbody>
</table>
Using AVR studio for ATMEGA 128 Development Board Programming

There are lots of IDEs (Integrated Development Environment) available for the AVR microcontrollers. There are free IDEs which are based on AVR GCC like AVR Studio from ATML and WIN AVR and proprietary IDEs like ICC AVR, Code vision AVR, IAR and KEIL etc. IDEs like ICC AVR and code vision AVR are very simple to use because of their GUI based code generator which gives you generated code. Almost all the proprietary IDEs works as full version for first 45 days and then there code size is restricted to some size. For this development board we are using AVR studio from the ATML. It uses WIN AVR open source C compiler at the back end. It has many attractive features like built-in In-Circuit Emulator and AVR instruction set simulator. After writing and compiling the program it gives “.hex” file. This “.hex” file needs to be loaded on the robot using In System Programmer (ISP).

Installing AVR Studio and WIN AVR

Before Installing AVR studio, we need to install WIN AVR as AVR studio uses WIN AVR at the backend. Both software’s should be installed in the same drive. AVR Studio and WIN AVR are located in the software folder in the documentation CD.

Installing WIN AVR
Go to “Software” folder in the documentation CD, copy folder “WIN AVR 2009-03-13” on the PC and click on “WinAVRxxxx.exe” to start the installation process.

WIN AVR installation package will open. Choose language as “English”.

![WIN AVR installation package](image.png)
Click next in the WIN AVR setup wizard.

Press “I Agree” after going through license agreement.
Make sure that you select drive on which operating system is installed.

Select all the components and press “Install”.
Click “Finish” to complete WIN AVR installation
Installing AVR Studio

Go to “Software” folder from the documentation CD, copy folder “AVR Studio 4.17” on the PC and click on “AvrStudio417Setup.exe” to start the installation process.

Click on “Run”
Click “Next” to start installation of AVR Studio 4

After clicking “Next” go through the license agreement. If it is acceptable then click “Next”

Now choose the destination drive. Select the same drive in which your operating system and WINAVR is installed.
Select for the “Install / upgrade Jungo USB Driver” to support In System Programming (ISP) by AVRISP mkII
**Important:** If “Install / upgrade Jungo USB Driver” is not selected then AVRISP mkII programmer will not work with the AVR Studio.

Click “Next” to start the installation process.

Click “Finish” to complete the installation process.
Setting up Project in AVR Studio
AVR studio is an Integrated Development Environment (IDE) for writing and debugging AVR applications. As a code writing environment, it supports included AVR Assembler and any external AVR GCC compiler in a complete IDE environment.

AVR Studio gives two main advantages:
1. Edit and debug in the same application windows. Faster error tracking.
2. Breakpoints are saved and restored between sessions, even if codes are edited.

Middle window shows current code under development. Window on the left side shows view of source files, header files, External dependencies, and other files. Right side window shows all the ports and other peripheral’s status. Bottom window is known as Build window. It shows results of the compilation, errors, HEX file size and other warning messages etc.

1. Open AVR Studio. If any project is running it can be closed by clicking on Project in the menu and select Close Project.
2. To create new project click on Project in the menu and select “New Project”.

NEX Robotics Pvt. Ltd.
www.nex-robotics.com
3. Select Project Type as “AVR GCC”. Type project name in the “Project name” window. In this case it is “buzzer_test”. Also check on Create folder check box. This will create all the files inside the new folder. In the Location window select the place where would like to store your project folder and then click “Next”.

4. Select debug platform and Device. In this case we have selected “AVR simulator” and “ATMEGA 2560” microcontroller and click finish.
5. Now we are almost ready to write our first code. Before we start coding we will check other settings to make sure that they are set properly.

6. Open Project menu and click on the Configuration option.
7. In the Project Option “General” tab will open. Select device as “ATMEGA 128” and frequency (Crystal Frequency) as 16 MHz i.e. 16000000. Set the optimization level be at “-O0”.

**Selecting proper optimization options**

“Optimization” option defines the optimization level for all files. Higher optimization levels will produce code that is harder to debug. Stack variables may change location, or be optimized away, and source level debugging may "skip" statements because they too have been optimized away.

The levels of optimization are:

- **-00** No optimization. This is the same as not specifying any optimization.
- **-01** Optimize. Reduces code size and execution time without performing any optimizations that take a great deal of compilation time.
- **-02** Optimize even more. avr-gcc performs almost all optimizations that don't involve a space-time tradeoff.
- **-03** Optimize yet more. This level performs all optimizations at -O2 along with –finline -functions and –frename –registers.
- **-Os** Optimize for size. Enables all -O2 optimizations that don't increase code size.

It also performs further optimizations designed to reduce code size.

For more information on optimization, see the 'man' pages for avr-gcc.
**Important:** During the coding choose appropriate optimization option. If you feel that code is not working properly as it should be then turn off all optimization by selecting optimization option as “-00”. Once you know that your code is properly working then you can incrementally increase optimization level.

We suggest that always use optimization level as “-00” at the beginner level.

8. Make sure that in the External Tools, proper path for avr-gcc.exe and make.exe are given and press ok.

Now we are ready to write our first code.
This program will make Board buzzer beep.
//Buzzer is connected at the third pin of the PORTC
//To turn it on make PORTC 3rd (PC3 )pin logic 1
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
//Function to initialize Buzzer
void buzzer_pin_config (void)
{
DDRC = DDRC | 0x08; //Setting PORTC 3 as output
PORTC = PORTC & 0xF7; //Setting PORTC 3 logic low to turnoff buzzer
}
void port_init (void)
{
buzzer_pin_config();
}
void buzzer_on (void)
{
unsigned char port_restore = 0;
port_restore = PINC;
port_restore = port_restore | 0x08;
PORTC = port_restore;
}
void buzzer_off (void)
{
unsigned char port_restore = 0;
port_restore = PINC;
port_restore = port_restore & 0xF7;
PORTC = port_restore;
}
void init_devices (void)
{
cli(); //Clears the global interrupts
port_init();
sei(); //Enables the global interrupts
}
int main(void) //Main Function
{
init_devices();
while(1)
{
buzzer_on();
_delay_ms(1000); //delay
buzzer_off();
_delay_ms(1000); //delay
}
}

We are now going to compile this code to generate the hex file which we will load on
the Microcontroller Board. Select “Build” menu and click on “Rebuild All”. It will
compile the “buzzer_test.c” code and will generate “buzzer_test.hex” file for the microcontroller Board.

You can verify successful compilation in the bottom most “Build” window of the AVR Studio.

You can also verify that “buzzer_test.hex” file is generated in the “default” folder inside the folder you have selected.

**Debugging the code in AVR studio**

After successful compilation of the code we can debug the code by AVR Debugger provided by AVRStudio. Here is the illustration of debugging of code given earlier.

NEX Robotics Pvt. Ltd.
www.nex-robotics.com
Click on Debug tab in the menu and click on “Start Debugging”.

Now debugging mode is started and an arrow is visible at the first line of our main function from where the debugging will start.

Press “F11” key or “Step into” button from the toolbar to start debugging statement by statement. Processor details are visible at left window and the I/O port status is displayed at the rightmost window.
By this way we can continuously monitor the bit changes in any of the registers of microcontroller and debug the code before actually burning it to the microcontroller. PORTC bits changes as per our commands and these changes can be seen in right window. After debugging is done select “Stop Debugging” from Debug tab.
Notice

The contents of this manual are subject to change without notice. All efforts have been made to ensure the accuracy of contents in this manual. However, should any errors be detected, NEX Robotics welcomes your corrections. You can send us your queries / suggestions at info@nex-robotics.com

Content of this manual is released under the Creative Commence cc by-nc-sa license. For legal information refer to:
http://creativecommons.org/licenses/by-nc-sa/3.0/legalcode

⚠️ Product’s electronics is static sensitive. Use the product in static free environment.
⚠️ Read the user manuals completely before start using this product

Recycling:
Almost all the part of this product are recyclable. Please send this product to the recycling plant after its operational life. By recycling we can contribute to cleaner and healthier environment for the future generations.