A BEGINNERS' GUIDE TO ATMEL AVR DEVELOPMENT

■ **ARUN DAYAL UDAI**

Learning microcontrollers is difficult, as any book meant for them starts with dry theory of their architecture and jargon of Assembly language programming. Most of my students frequently come to me asking "where to start from" and "how to proceed." I always suggest them a line of readings, proper tools and software which one should learn to begin with a microcontroller. Here is a guide to Atmel AVR development which would be very helpful to beginners for learning microcontrollers.

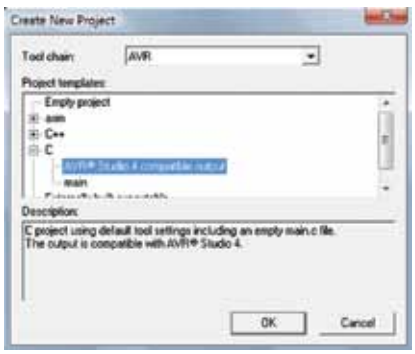

Fig. 1: Creating new project window

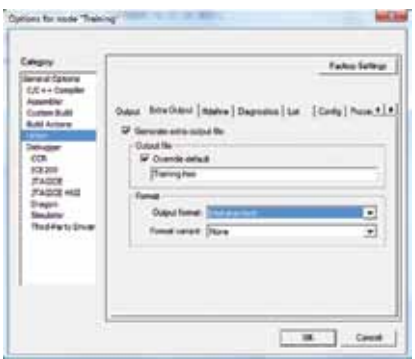

Fig. 2: Options window for selection of Intel-standard hex file

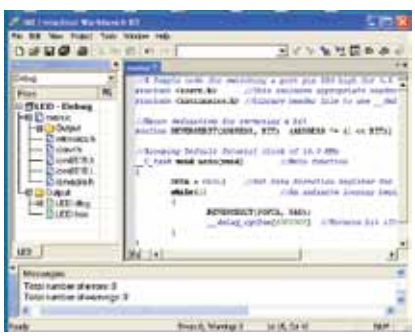

Fig. 3: Main window for compilation of source program

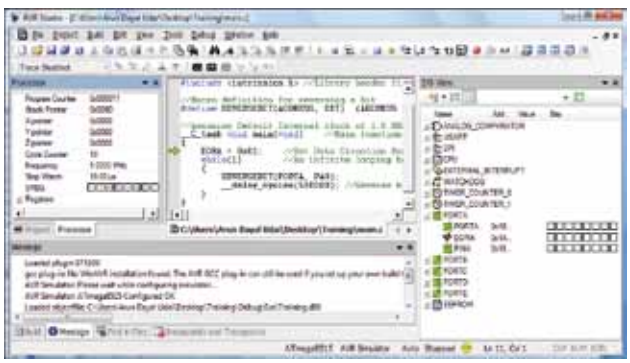

Fig. 4: AVR Studio window for simulation

Any microcontroller would typically have some input/output pins, a few of which may have multiple functions. It has a reset switch, DC supply pins and a clock source which helps the device to send the data through different sections within the chip.

Without going much into the details of a microcontroller, let us learn practically by implementing various parts of a microcontroller. For this reason, I would like to introduce ATmega8515, which belongs to the 8-bit family of AVR microcontrollers—one of the most basic ones.

All AVR microcontrollers are similar when it comes to programming and implementation, apart from some of the added features. The only prerequisite for a novice learner is knowledge of binary arithmetic and C programming basics.

IAR Embedded Workbench

The IAR Embedded Workbench provides a suite of AVR development tools for embedded systems. It can be downloaded for free from IAR website www.iar.com. IAR Embedded Workbench is being parallelly developed by IAR Systems and ATMEL developers and hence it generates the optimised code which uses full 'C' coding capabilities of AVR devices. Coding in IAR is nicely supported by its library functions, which makes a normal C user comfortably program any embedded device too.

Let us begin using our coding platform. To start the application, click 'IAR Embedded Workbench' icon in 'All Programs' menu of Windows. From the main menu, select Project→Create New Project and start an AVR Studio compatible C project (refer Fig. 1), as later on you will need to simulate the code in AVR Studio 4.0. Save the new project to a new folder as an embedded workbench project (.ewp) file and then press 'Alt+F7.' 'Options' window appears as shown in Fig. 2. Select the Atmega8515 microcontroller as follows:

Project→Options→General Options (under Category)→Target→Processor Configuration→Select cpu=m8515, Atmega8515

Enable bit definitions in I/O include files as follows:

Project→Options→General Options (under Category)→System→Enable bit definitions in I/O include files

Generate the Intel hex file for burning the code into Atmega8515 microcontroller as follows (refer Fig. 2):

Project→Options→Linker (under Category)→Extra Output→Generate

Extra Output (to click)→Override Default (to click)→Type Project Name.hex→Output Format→intel-standard (to select)

Press 'ok' after all the above settings are done.

The source code for LED blinking (main.c) along with comments for instructions (also shown in the text editor window in Fig. 3) is:

```
//A Sample code for switching a port pin PA0 high for 0.5 seconds and low //for 0.5 seconds
#include <ioavr.h>          //This includes appropriate header file for the //device as selected
#include <intrinsics.h>   //Library header file to use __delay_cycles()

//Macro definition for reversing a bit
#define REVERSEBIT(ADDRESS, BIT) (ADDRESS ^= (1 << BIT))

//Assuming Default Internal clock of 16.0 MHz
__C_task void main(void)  //Main function
{
        DDRA = 0x01;      //Set Data Direction Register for PA0 as output
        while(1)          //An infinite looping begins
        {
                REVERSEBIT(PORTA, PA0);
                __delay_cycles(500000);
//Reverse bit after 0.5 seconds
        }
}
```

After entering the code, it is ready for compilation and linking to generate the .d90 debug file and Intel standard hex file for downloading into the target microcontroller. These files are generated in Project→Debug→Exe folder. To compile the code, click 'Rebuild All' option in 'Project' menu.

Save the embedded workbench project (.ewp) file to the working folder. After this, you are done with coding and IAR Embedded Workbench.

AVR Studio. After compilation, start simulation and debugging using AVR Studio 4.0 from 'All Programs' menu of Windows. When the application starts, open the .d90 debugging file located in Project→Debug→Exe folder and save the AVR Studio project (.aps) file to 'Project' folder. Now from the selection window, select the platform and device for simulation as AVR Simulator and ATmega8515, respectively. This will start simulating the generated C code in IAR Embedded Workbench. You may continuously edit and recompile the code in parallel with AVR Studio, which automatically reloads the object file and C file for simulation (as shown in Fig. 4).

Immediately before doing anything else, change the system clock frequency to correct value so that the code runs with correct clock frequency. Now proceed as follows:

Main menu→Debug→AVR Simulator Options (Alt+O)

You can keep pressing Debug→Step Into (F11) to proceed with simulation. Expand the port which is to be monitored (Port A for the current code). As soon as PA0 is set, right-

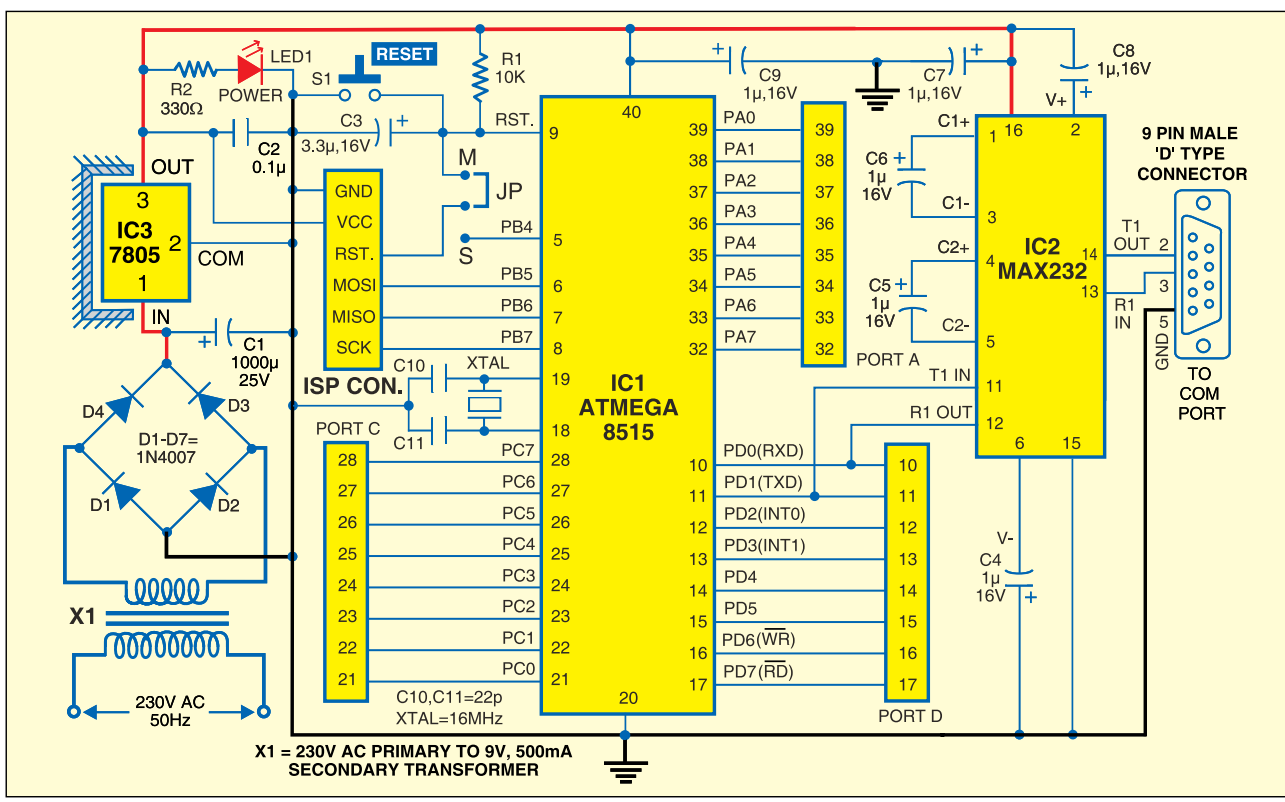

Fig. 5: Circuit of the development board for ATmega8515

Fig. 6: STK500 window for data downloading into microcontroller

click 'Stop Watch' and reset the stop watch. Continue debugging by pressing F11 key and note the time when it is cleared again and finally set. This should be approximately one second. After this, code simulation is complete. You have successfully coded and simulated your AVR program.

In the following section, you will practically download your code to ATmega8515 flash memory and see things happening.

Circuit description

Fig. 5 shows the circuit of the development board for ATMega8515. The 230V, 50Hz AC mains is stepped down by transformer X1 to deliver a secondary output of 9V, 500mA. The transformer output is rectified by a full-wave rectifier comprising diodes D1 through D4, filtered by capacitor C1 and regulated by IC 7805 (IC1). Capacitor C2 bypasses the ripples present in regulated supply. LED1 acts as the power indicator and resistor R2 limits the current through LED1.

PARTS LIST

Semiconductors:

IC1	- ATmega8515 AVR microcontroller
IC2	- MAX232 RS-232 driver
IC3	- 7805, 5V regulator
D1-D4	- 1N4007 rectifier diode
LED1	- 5mm LED

Resistors (all ¼-watt, ±5% carbon):

R1	- 10-kilo-ohm
R2	- 330-ohm

Capacitors:

C1	- 1000µF, 25V electrolytic
C2	- 0.1µF ceramic disk
C3	- 3.3µF, 16V electrolytic
C4-C9	- 1µF, 16V electrolytic
C10, C11	- 22pF ceramic disk

Miscellaneous:

X1	- 230V AC primary to 9V, 500mA secondary transformer
S1	- Push-to-on tactile switch
X_{TAL}	- 16MHz ceramic crystal
	- 9-pin D-type male connector
	- 3-pin berg strip (male)
	- 6-pin ISP male connector
	- 3×8-pin SIP male connector

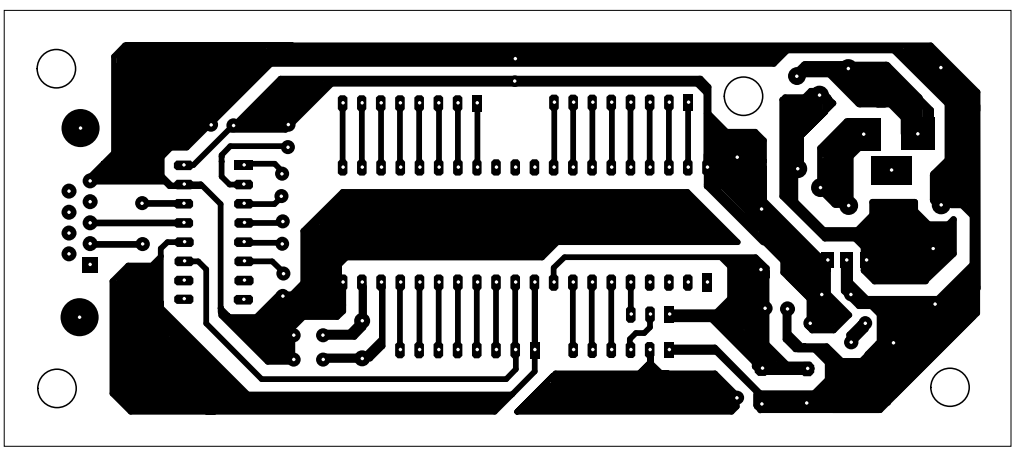

Fig. 7: An actual-size, single-side PCB for the ATmega8515 development board

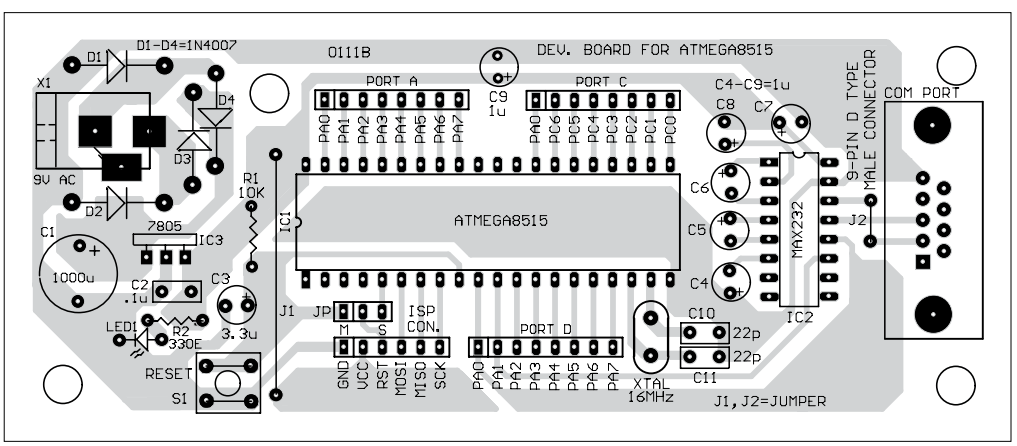

Fig. 8: Component layout for the PCB

Atmega8515 is the heart of the circuit. It features 8kB Flash memory with read-while-write capabilities, 512-byte EEPROM, 512-byte SRAM, 35 general-purpose input/output lines, 32 general-purpose working registers, two flexible timers/counters with compare modes, internal and external interrupts, a serial programmable USART, a programmable watchdog timer with internal oscillator and SPI serial port.

Switch S1 is used for manual reset. Port pins PB5 (MOSI), PB6 (MISO) and PB7 (SCK) are used for in-system programming (ISP). IC MAX232 (IC3) is used for serial communication. TXD and RXD pins of the microcontroller are used for serial communication with the help of COM port.

An actual-size, single-

side PCB for the circuit of the ATmega8515 development board (Fig. 5) is shown in Fig. 7 and its component layout in Fig. 8. Assemble the circuit on a PCB to minimise time and assembly errors. Carefully assemble the components and double-check for any overlooked error.

Downloading your test code

Once simulation is complete, you may want to see its working on your own development board. To program the ATmega8515, connect the 6-wire cable between ISP connectors of the STK500 board (marked as ISP6PIN) and the target board (marked as ISP CON).

The STK500 is a complete starter kit and development system for the AVR Flash microcontroller from Atmel Corporation. It is designed to give designers a quick start to develop code on the AVR and for prototyping and testing of new designs. The STK500 is supported by AVR Studio. The information on STK500 and other AVR tools can be found in the AVR section of the Atmel website (www.atmel.com).

Connect a serial cable from the connector marked 'RS232 CTRL' on the STK500 board to a COM port on the PC. A three-pin connector (JP) is provided on development board. Connect the reset pin of the ISP connector to reset pin 9 of ATmega8515 microcontroller by shorting the jumper (JP) to 'M' point. M and S are meant for 'master' and 'slave' mode respectively for serial data transfer application.

Now start AVR Studio 4.0 without opening any project file. Optionally, you may proceed as follows: Main Menu→Tools→Program AVR→Select AVR Programmer. Press 'Connect...' after selecting STK500 or AVRISP in the platform window and COM port (say, COM1) in the port window.

After selection, 'AVR Studio STK500 Programming Menu' window appears as shown in Fig. 6. Select ATmega8515 as 'Device,' ISP as 'Programming Mode' and browse your project hex file from Project→Debug→Exe folder.

Press 'Program' button on the STK500 dialogue box to burn the hex file into your microcontroller. Now remove the JP jumper from 'M' point. Disconnect your device from the ISP cable and press reset switch. Now you can use an LED at port PA0 of ATMEGA8515 to check whether it flashes once every second. A current-limiting resistor (470 ohm) in series with the LED may be connected between PA0 and ground to check blinking of the LED.

EFY note. The source code of this article has been included in this month's EFY-CD and is also availabe on efymag.com website.

The author is an assistant professor in Department of Mechanical Engineering at Birla Institute of Technology, Mesra, Ranchi