AVR PROGRAMMER USING AVR910 PROTOCOL

■ ARUN DAYAL UDAI, BRIJ MOHAN

Newbies often have difficulty getting started with programming of AVR microcontroller memory. What they need is a simple documentation and the entire code for AVR memory programming.

This article attempts to simplify the existing AVR910 technical note available on ATMEL's website. It includes a source program for the microcontroller written in 'C' language for easy understanding of the principles and a user interface software program written in Microsoft Visual Basic .Net for interfacing and transferring a binary code to the target device.

The complete system for programming an AVR microcontroller uses a universal asynchronous receiver transmitter (UART) with serial peripheral interface (SPI) to program the target microcontroller's fuses, lock bits, flash or EEPROM. Any AVR board can be converted into a master programmer using this protocol.

Circuit description

Fig. 1 shows the block diagram for programming a target device with the help of a PC and AVR board.

Fig. 2 shows the AVR microcontroller programmer circuit to burn the binary code into the target device. At the heart of the circuit is ATmega8515 (IC2). It features 8kB Flash memory with read-while-write capabilities, 512-byte EEPROM, 512-byte SRAM, 35 general-purpose input/output (I/O) 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 an 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. T_{XD} and R_{XD} pins of the microcontroller too are used for serial communication with the help of COM port.

In order to derive the power supply for the circuit, the 230V, 50Hz AC

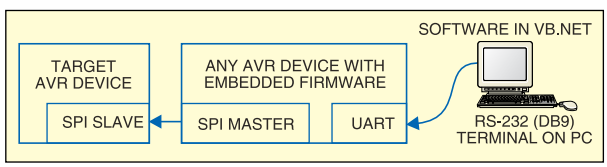

Fig. 1: Block diagram for programming a target device with the help of a PC and AVR board

PARTS LIST

Semiconductors:

IC1, IC4	- 7805, 5V regulator
IC2, IC5	- ATmega8515 AVR microcontroller
IC3	- MAX232 RS-232 driver
D1-D4	- 1N4007 rectifier diode
LED1-LED9	- 5mm LED

Resistors (all ¼-watt, ±5% carbon)

R1, R11	- 10-kilo-ohm
R2-R10	- 330-ohm

Capacitors:

C1	- 1000µF, 25V electrolytic
C2, C15	- 0.1µF ceramic disk
C3, C14	- 4.7µF, 16V electrolytic
C4-C9	- 1µF, 16V electrolytic
C10-C13	- 22pF ceramic disk

Miscellaneous:

X1	- 230V AC primary to 9V, 500mA secondary transformer
X_{TAL1}, X_{TAL2}	- 16MHz crystal
S1-S10	- Push-to-on tacktile switch
S11	- On/off switch
BATT.	- 9V, battery
	- 9-pin D-type female connector

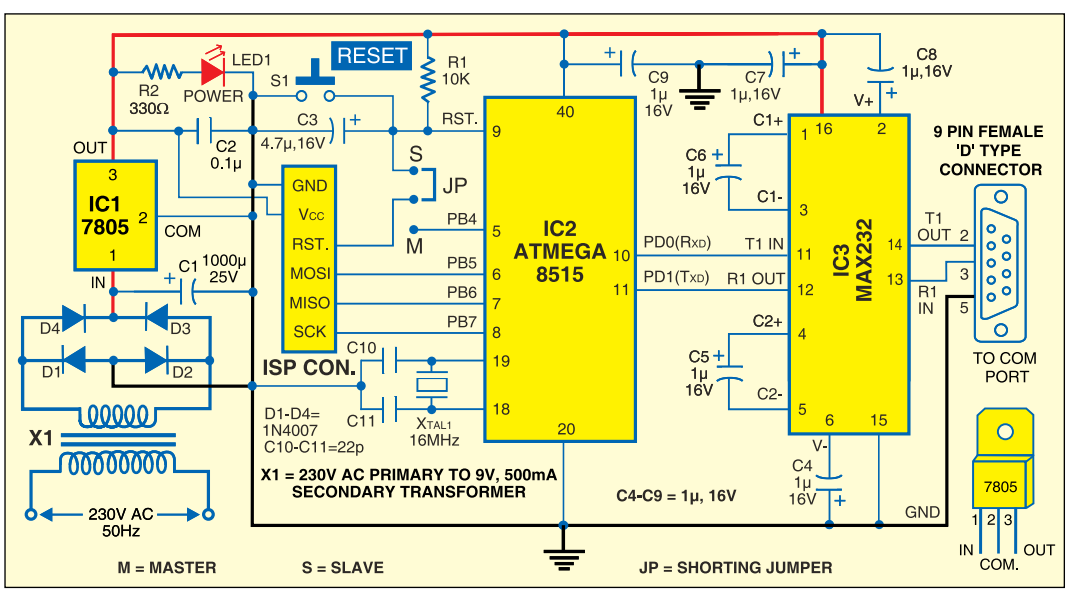

Fig. 2: AVR microcontroller programmer circuit

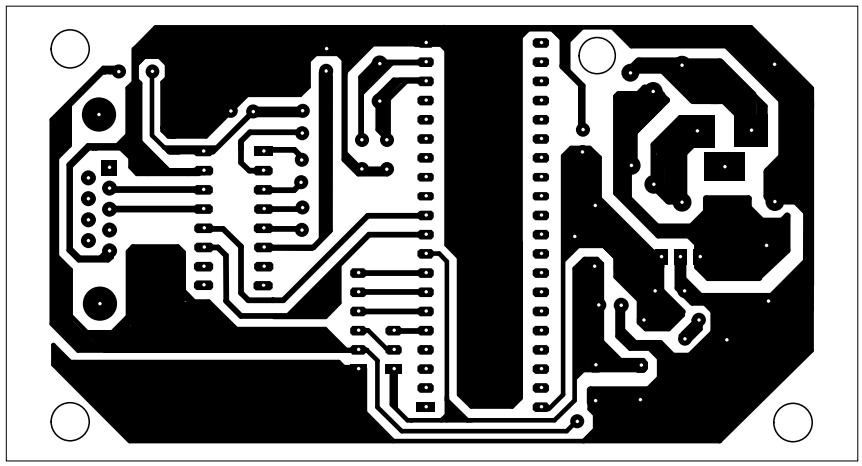

Fig. 3: An actual-size, single-side PCB for the circuit of AVR programmer using AVR910 protocol (Fig. 2)

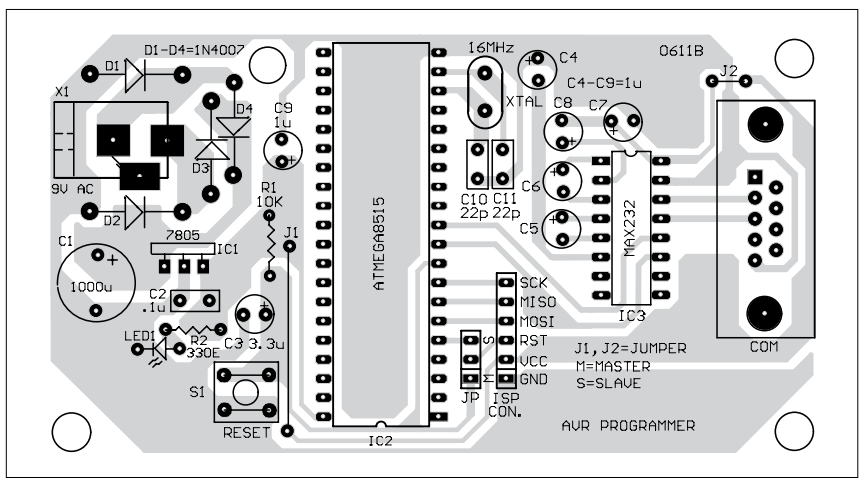

Fig. 4: Component layout for the PCB

mains is stepped down by transformer X1 to deliver a secondary output of 9V, 500 mA. 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 the regulated supply. LED1 acts as the power indicator and resistor R2 limits the current through LED1.

An actual-size, single-side PCB for the circuit of AVR programmer using AVR910 protocol (Fig. 2) is shown in Fig. 3 and its component layout in Fig. 4. Assemble the circuit on a PCB to minimise time and assembly errors. Carefully assemble the components and double-check for any overlooked error.

The target AVR microcontroller enters serial programming mode only when its reset pin is active-'low'. Immediately after the reset pin becomes low, serial clock (SCK) pin is driven to zero by the programmer. The reset pin makes the device SPI slave and the target device ready to accept instructions from the programmer through MOSI (master out, serial data in) and MISO (master in, serial out) pins.

The serial programming instruction set can be referred through any AVR datasheet. It can be implemented on any AVR board by just making the device at 19,200 baud rate for UART and selecting a proper pin on the master device which will make the reset pin of the slave device low, SPI slave select input PB4 (SS) pin being the most convenient one. Three-pin connectors are provided on the programmer board for burning the code into the programmer or the target microcontroller.

To program the ATmega8515 on the programmer board, connect a 6-wire cable between ISP connectors (marked as ISP6PIN) of the STK500 board and the programmer board. Connect a serial cable from the connector marked as 'RS232 CTRL' on the STK500 board to COM port of the PC. Connect the reset pin (RST) of the ISP connector to reset pin 9 of ATmega8515 microcontroller IC2 by shorting the jumper (JP).

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. Select 'ATmega8515' as the device, 'ISP' as the programming mode and browse your project hex file (prog.hex) from Project→Debug→Exe folder.

You may verify fuse bits showing the default setting. Now press 'Program' button in 'Program' tab to burn the hex file into your microcontroller (IC1). Disconnect IC1 from the ISP cable. Now the programmer board is ready to burn the binary file into the target AVR microcontroller.

Software

The software (AVR910) for the microcontroller is written in 'C' using the IAR Embedded Workbench integrated development environment. IAR Embedded Workbench is being developed by IAR Systems and ATMEL developers in parallel and hence it generates the optimised code which uses full 'C' coding capabilities of AVR devices. AVR development tools for embedded systems can be

TABLE I
SPI Command Sets (in Hex Format) for Serial Programming of the Target AVR Microcontroller*

Instruction/operation	SPI instruction format			
	Byte 1	Byte 2	Byte 3	Byte 4
Programming Enable	AC	53	00	00
Chip Erase (program memory/ EEPROM)	AC	80	00	00
Poll RDY/BSY	F0	00	00	data byte out
Load instructions				
Load extended address byte	4D	00	Extended adr	00
Load program memory page, high byte	48	adr MSB	adr LSB	high data byte in
Load program memory page, low byte	40	adr MSB	adr LSB	low data byte in
Load EEPROM memory page (page access)	C1	00	adr LSB	data byte in
Read instructions				
Read program memory, high byte	28	adr MSB	adr LSB	high data byte out
Read program memory, low byte	20	adr MSB	adr LSB	low data byte out
Read EEPROM memory	A0	adr MSB	adr LSB	data byte out
Read lock bits	58	00	00	data byte out
Read signature byte	30	00	0000 000aa	data byte out
Read fuse bits	50	00	00	data byte out
Read fuse high bits	58	08	00	data byte out
Read extended fuse bits	50	08	00	data byte out
Read calibration byte	38	00	0b00 000bb	data byte out
Write instructions				
Write program memory page	4C	000a aaaa	aa00 0000	00
Write EEPROM memory	C0	adr MSB	adr LSB	data byte in
Write EEPROM memory page (page access)	C2	adr MSB	adr LSB	00
Write lock bits	AC	E0	00	data byte in
Write fuse bits	AC	A0	00	data byte in
Write fuse high bits	AC	A8	00	data byte in
Write extended fuse bits	AC	A4	00	data byte in

**Commands are sent by the programmer board to the slave device which is being programmed*

downloaded for free from IAR website www.iar.com. For details of IAR Embedded Workbench, you may refer to 'A Beginners' Guide to ATMEL AVR Development' article published in January issue of EFY.

Coding for AVR programmer. When writing serial data to the AVR microcontroller, data is clocked on the rising edge of SCK. When reading data from the AVR microcontroller, data is clocked on the falling edge of SCK. To program and verify the AVR microcontroller in the SPI serial programming mode, the following sequence is recommended (refer Table I for serial programming command):

1. Power-up: Apply power between V_{CC} and ground (GND) while RST and SCK are set to '0'. In some systems, the programmer cannot guarantee that SCK is held low during power-up. In this case, RST must be given a positive pulse of at least two CPU clock cycles duration after SCK has been set to '0'.

2. Wait for at least 20 ms and enable SPI serial programming by sending the 'Programming Enable' serial instruction to pin MOSI.

3. The SPI serial programming instructions will not work if the communication is out of synchronisation. When in synchronisation, the second byte (0x53) will echo back when issuing the third byte of the 'Programming Enable' instruction. Whether the echo is correct or not, all four bytes of the instruction must be transmitted. If the 0x53 doesn't echo back, give a positive pulse to reset pin and issue a new 'Programming Enable' command.

4. The Flash is programmed one page at a time. The memory page is loaded one byte at a time by supplying the six least significant bits (LSBs) of the address and data together with the "load program memory page" instruction. To ensure correct loading of the page, the low-byte data must be loaded before high-byte data is applied for a given address. The program memory page is stored by loading the "write program memory page" instruction with the seven most significant bits (MSBs) of the address. If polling is not used, the user must wait for a period of at least t_{WD_FLASH} before issuing the next page. Accessing the SPI serial programming interface before completion of the Flash-write operation can result in incorrect programming.

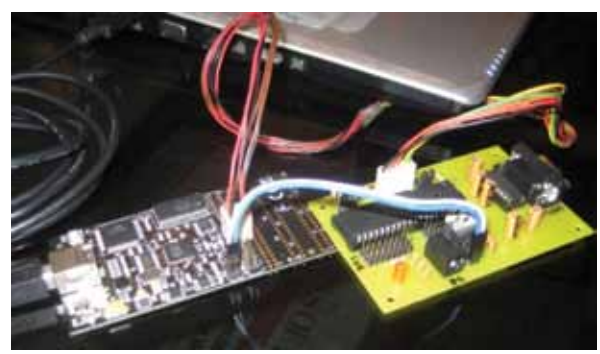

Fig. 5: Burning the hex code into the programmer board microcontroller

5. The EEPROM array is programmed one byte at a time by supplying the address and data together with the appropriate write instruction. An EEPROM location is au-

TABLE II
AVR910 UART Serial Commanding Protocol

	Host PC writes		Host PC reads	
	ID	Data	ID	Data
Enter programming mode	'P'	—	—	13d
Report auto increment address	'a'	—	—	'Y'
Set address	'A'	ah al	—	13d
Write program memory, low byte	'c'	Dd	—	13d
Write program memory, high byte	'C'	Dd	—	13d
Issue page write	'm'	—	—	13d
Read program memory	'R'	—	dd(dd)	—
Write data memory	'D'	dd	—	13d
Read data memory	'd'	—	dd	—
Chip erase	'e'	—	—	13d
Write lock bits	'l'	dd	—	13d
Write fuse bits	'f'	dd	—	13d
# Read fuse and lock bits	'F'	—	dd	—
Leave programming mode	'L'	—	—	13d
Select device type	'T'	dd	—	13d
Read signature bytes	's'	—	3*dd	—
Return supported device codes	't'	—	n*dd	00d
Return software identifier	'S'	—	s[7]	—
Return software version	'V'	—	dd dd	—
Return hardware version	'v'	—	dd dd	—
Return programmer type	'p'	—	dd	—
Set LED	'x'	dd	—	13d
Clear LED	'y'	dd	—	13d
Universal command	':'	3*dd	dd	13d
New universal command	'.'	4*dd	dd	13d
Special test command	'Z'	2*dd	dd	—

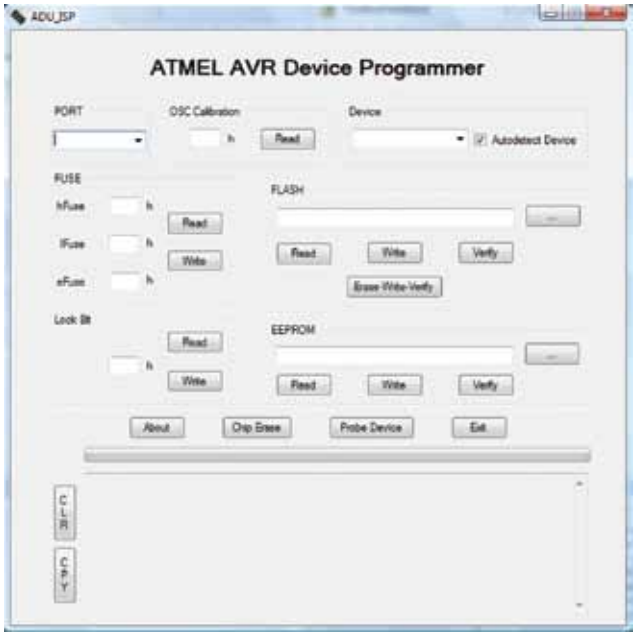

Fig. 6: VB.NET form layout for programmer software

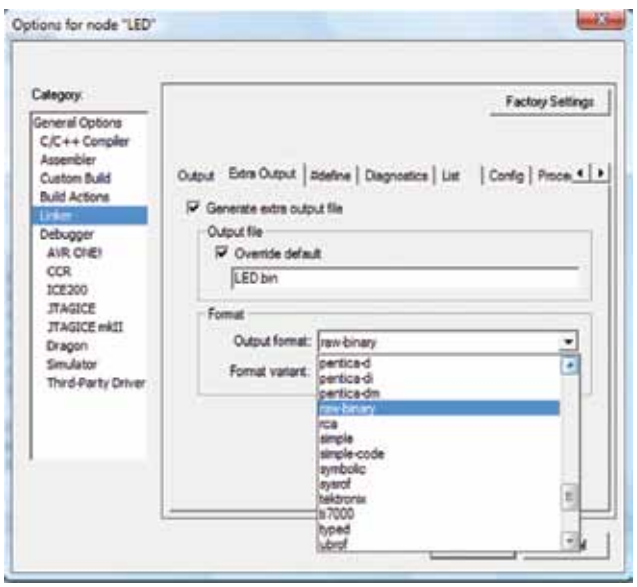

Fig. 7: Screenshot of raw binary file generation

tomatically erased before new data is written. If polling is not used, the user must wait for a period of at least t_{WD_EEPROM} before issuing the next byte. In a chip-erased device, no 0xFFs in the data files need to be programmed.

6. Any memory location can be verified by using the read instruction, which returns the content at the selected address at serial output MISO.

7. At the end of the programming session, RST can be set high to commence normal operation.

8. Power-off (if needed): Set RST to '1' and turn V_{CC} power off.

Programming the target AVR microcontroller (for user interface) using Visual Basic .NET (VB.NET). Once the master device (IC2) is ready to receive data or command from the PC and is connected to the slave device (IC5), it only requires an interface software on the PC which can supply data/command to it. This interface software is written in Visual Basic .NET. It supports AVR910 in-system programming protocol for commands. The interface software (ADU-ISP) given here can implement almost all ATmega AVR microcontrollers having Flash size of less than 128 kB. Just construct a form in VB which has the controls as shown in Fig. 6, with SerialPort and OpenFileDialog controls, and insert the code that follows.

The source program includes remarks for easy understanding.

Table II shows AVR910 UART serial commanding protocol. These commands are sent to the programmer board through serial communication using the VB.NET program. All commands comprise a single letter. The programmer returns 13d, which is a carriage-return character or the data read after the command is finished.

The VB.NET program accepts raw binary file (*.bin) file, which is generated after compiling and linking a source program for the microcontroller. This binary file data has to be burnt into the

target AVR microcontroller.

Currently, all the ATmega AVR microcontroller devices, except ATmega640/1280/1281/2560/2561, have Flash size of up to 128 kB. Writing (burning the code into) the Flash of ATmega640/1280/1281/2560/2561 requires extended addresses. Also, as these devices have a higher pin count and come in TQFP package, they will not fit into the programmer board described here. This AVR programmer works fine with ATmega8515 and ATmega16/32.

If the user is programming using IAR Embedded Workbench, the steps for generating a raw binary file (*.bin) file are:

Goto Project→Options→Linker→Extra Output→Tick 'Generate Extra Output File'→Tick 'Override Default'→Select 'Output Format' as 'Raw-Binary' (refer Fig. 7).

A VB.NET program writes the raw binary file starting from address '0000' of the flash memory. So when writing a code which does not start from '0000' address, the user needs to take care by editing the linker (*.xcl) file before compiling the AVR source code.

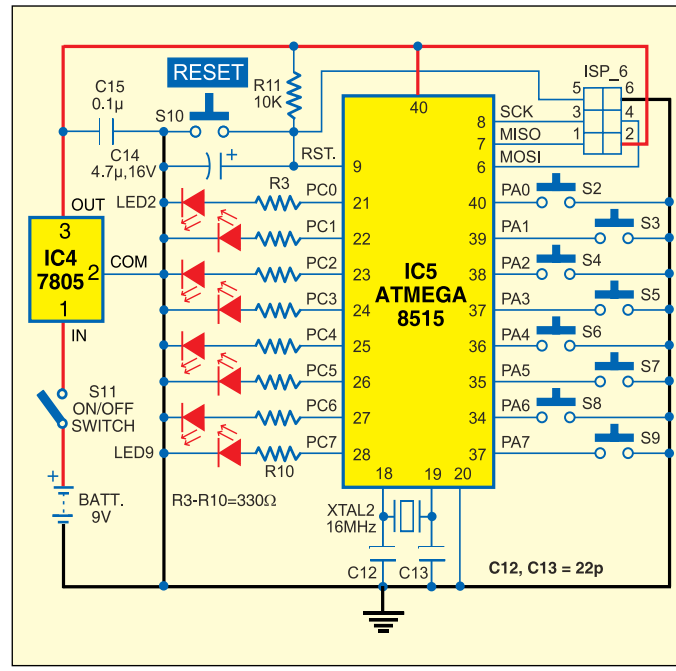

Fig. 8: LED switching circuit

A sample code for LED switching (led_sw.c) is given at the end of this article to test the working of the AVR programmer board. This code has to be burnt into the target device. Eight LEDs are connected to Port C of the microcontroller (IC5) with their controlling switches connected to Port A. Initially, when IC5 is programmed successfully, all the LEDs glow. Now if you press any of the switches, the corresponding LED should go off. Fig. 8 shows the circuit for LED switching.

E F Y note. The source codes AVR910.C, LED_SW.C and ADU_ISP are available on www.efymag.com website.

Arun Dayal Udai is an assistant professor in Department of Mechanical Engineering at Birla Institute of Technology, Mesra, and Brij Mohan is a project leader at Mphasis Ltd, Bengaluru

LED_SW.C

```
/* led_sw.c*/

#include <ioavr.h>
#include <intrinsics.h>

#define LED0 PORTC0  //Connected to PORTC
#define LED1 PORTC1  //Connected to PORTC
#define LED2 PORTC2  //Connected to PORTC
#define LED3 PORTC3  //Connected to PORTC
#define LED4 PORTC4  //Connected to PORTC
#define LED5 PORTC5  //Connected to PORTC
#define LED6 PORTC6  //Connected to PORTC
#define LED7 PORTC7  //Connected to PORTC

#define SW0 PINA0
#define SW1 PINA1         //All Switches
are connected to PORTA
#define SW2 PINA2
#define SW3 PINA3
#define SW4 PINA4
#define SW5 PINA5
#define SW6 PINA6
#define SW7 PINA7

#define SETBIT(ADDRESS, BIT)     (ADDRESS
|= (1 << BIT))
#define CLEARBIT(ADDRESS, BIT)   (ADDRESS
&= ~(1 << BIT))
#define CHECKBIT(ADDRESS, BIT)    (!(AD-
DRESS & (1 << BIT)))

__C_task void main(void)
{
     //Set all LEDs pins as Output
          DDRC = 0XFF;
               //Enable pull-ups on all
switches
          PORTA = 0xFF;

     while(1)
     {
               if (CHECKBIT(PINA, SW0))
//Press to LOW for Switches
                    CLEARBIT(PORTC, LED0);
               else
                     SETBIT(PORTC, LED0);

               if (CHECKBIT(PINA, SW1))
                    CLEARBIT(PORTC, LED1);
               else
                     SETBIT(PORTC, LED1);

               if (CHECKBIT(PINA, SW2))
                    CLEARBIT(PORTC, LED2);
               else
                     SETBIT(PORTC, LED2);

               if (CHECKBIT(PINA, SW3))
                    CLEARBIT(PORTC, LED3);
               else
                    SETBIT(PORTC, LED3);

               if (CHECKBIT(PINA, SW4))
                    CLEARBIT(PORTC, LED4);
               else
                    SETBIT(PORTC, LED4);

               if (CHECKBIT(PINA, SW5))
                    CLEARBIT(PORTC, LED5);
               else
                    SETBIT(PORTC, LED5);

               if (CHECKBIT(PINA, SW6))
                    CLEARBIT(PORTC, LED6);
               else
                    SETBIT(PORTC, LED6);

               if (CHECKBIT(PINA, SW7))
                    CLEARBIT(PORTC, LED7);
               else
                    SETBIT(PORTC, LED7);
          }
}
```