ATMEL APPLICATIONS NUMBER 10 • SUMMER 2008

On-Line JOURNAL

# An Inexpensive AVR Controlled 16-Channel Serial Servo Motor Controller Design for Robotic Applications

**By: ß Arun Dayal Udai,** ***Research Scholar***
***Birla Institute of Technology, Mesra, Ranchi, INDIA***

**Useful Links in this Article:**

**MegaAVR:** When your designs call for a bit of extra muscle, you need megaAVR. Developed for applications that need to store a large amount of program code, megaAVR offers substantial program and data memories, and performance approaching 1 MIPS per MHz. Better yet, megaAVR delivers the power of self-programmability for fast, secure, cost-effective remote upgrades.

http://www.embeddeddeveloper.com/processors/1768/Atmel/ATmega8515.htm

## SUMMARY

*Motor control is always a challenge for hobbyists and professionals who intend to develop any autonomous robots. Several types of motors and their control have been covered in previous issues of this Journal under different titles. However, their uses are limited either due to their restrictions on the number of motors to be used at a time or lack of feedback system and precision. Stepper motor controllers are normally operated in open loop with load varying within the predetermined torque. Position of DC motors is difficult to control as they have inherent high rpm and no inbuilt encoders. Motor controllers available in the market today are normally costly and have limited number of channels to control motors (normally 2, 4 or maximum 8 channels). Hardly any one of them is readily available as they are not manufactured in developing countries.*

*The given circuit design can be developed at cheaper rate and uses closed loop control which makes it a very precise positioning device. A standard servo is a DC Motor with a mechanical feedback and a reduction gear to reduce the rpm. It has a potentiometer position encoder and a mechanical feedback arrangement built in to its enclosure. It is driven by a square wave pulse of 5V (TTL) in amplitude its width ranges from 1.0 mS to 2.0 mS with refresh rate of 50Hz i.e. 20 mS. The duration of the pulse i.e. its width, is used to judge the angular position of the motor.*

*The angular position 0 to 180o, is linearly dependent on the pulse width. Several brands like Futaba, JR, Airtronics (Sanwa), GWS, Hitec, BMS, Multiplex, Parallax (Futaba) etc. manufacture compatible servos. Most of these are available at various hobby shops.*

*The given schematic used makes full use of cheap, simple and yet robust microcontroller family AVR so as to increase the domain of applications. As previously quoted in one of the issue, "It is fun working with AVR microcontrollers", hope the readers will find the given design with AVR very useful for amateur and advanced robotic applications.*

## MICROCONTROLLER

The designed circuit uses an ATMEL® AVR 8-bit Microcontroller ATmega8515. It is a 40 pin IC with four 8 bit ports and one 4 bit port. Although it has only 2 dedicated PWM channels, all I/O pins can be internally timed through program to get a PWM signal. So, altogether 36 pins can be used for PWM control. In the given schematic 3 pins are used for serial data transfer so altogether 33 pins can be timed for PWM signals. Only 16 of which is used which satisfies the current need of my dream project, which is to control the axes of a 10 Degree of Freedom (DOF) static Biped/Humanoid walker. Various compilers are available in the market for their programming. Popular among them are GNU C Compiler (WinAVR) which is freely available at http://www.gnu.org and IAR C Compiler. A trial version of IAR Embedded Workbench V4.2 can be downloaded from

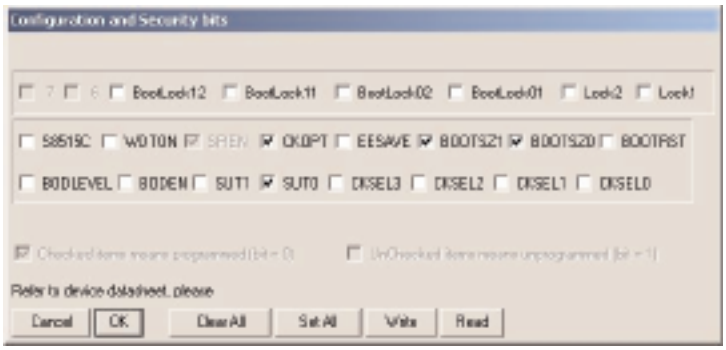
http://www.iar.com. AVR Studio® which is available free on ATMEL AVR site http://www.atmel.com/avr/ supports Assembly language programming and C compilation and linking through GCC. IAR Embedded Workbench supports AVR Studio compatible C program that can be simulated before actual downloading the program to the microcontroller. Due to wide range of functions available in IAR C compiler, it has been used for this project.

Downloading of the code to the microcontroller can be done using various downloader supported by AVR Studio like STK, JTAG, ICE, AVRISP or even through simple and cheap circuits without debugging facilities like the one given by Third Party solutions like at http://www.lancos.com/prog.html. It can be probed with burning software like Ponyprog2000 by selecting the menu option, Setup > Interface Setup > Parallel > Avr ISP I/O. The burning software PonyProg2001 can be freely downloaded form the same site. It is found that Parallel port downloader is the easiest one and a cheaper choice. Before downloading the device must be selected through main menu of PonyProg2001 to ATmega8515. Setup must be done for Calibration of PonyProg2001 to the system with the option given in the main menu.

## SERIAL PORT AND CLOCK SETUP

By default a new AVR microcontroller is driven by its internal clock. It is required to change the security and configuration bits i.e. fuse bits to use external crystal oscillator. It is always advisable to refer datasheet before handling these bits. PonyProg2001 supports programming Security and Configuration bits. The given circuit uses an 11.0592 MHz crystal, as it is one of the most widely available crystals and can be obtained from any electronics component shop. The fastest option would be the 14.7456 MHz but it is not frequently used in standard market. A standard 16 MHz crystal is having the maximum frequency that can be used with this microcontroller which would cause serial communication error of 2.1% at 115.2 kbps. Slower option is 3.6864 MHz crystal that also is readily available but it causes the PWM error of 0.5% with the given program. The optimum one is 11.0592 MHz crystal which makes serial communication error of 0 % and PWM error of 0.125%. Programming the fuse bits can be done with PonyProg2001 by selecting Command > Security and Configuration Bits … in the main menu of PonyPorg2001. Referring to the datasheet, the checkboxes should be clicked as shown in Fig. 1.

The program is designed to accept or send serial data at 115.2 kbps which may be modified to any other value based on UBRR setting as per the datasheet. USART_Init (UBRR value) function sets the baud rate as required.




(11.0592 MHz Crystal)

**Figure 1: Security and Configuration Bits**

USART_Transmit subroutine has been incorporated for debugging purpose and for learners who want to learn USART Transmit function in AVR. The program will work even without USART_Transmit function in such a case 34 Channels can be used for Servo Motor Control. Modifications required for that have been kept as remarks with '*' Mark.

** For using 3.6864 MHz Crystal, lines below the commented line should be made active and the respective lines for 11.0592 MHz crystals should be removed.

## THE COMPILER AND THE PROGRAM

The microcontroller program listed should be compiled using IAR Embedded Workbench V4.2. Following Options must be selected before building the raw binary for burning it to the microcontroller. Sequence of options that must be selected from the main menu of IAR Embedded Workbench is as follows:

- Project > Create New Project > Expand C >Select AVR® Studio 4 Compatible output > Save As (project name) > Type the given program listing under main.c program space.
- Project > Options > General Options > Target [tab] > Processor configuration > Select [--cpu=m8515, ATmega8515] from the drop down combo box.
- Project > Options > General Options > System [tab] > Check [√] Enable bit definitions in I/O - include files
- Project > Options > Linker > Output [tab] > Click Other [Radio Button] > Output format > Select raw-binary [from Drop down combo box]
- Project > Options > Linker > Extra Output [tab] > Check Generate extra output file > Output format > Select ubrof 8 (forced) [from Drop down combo box]

For building binary file for burning to the microcontroller through PonyProg2001.

- Project > Rebuild All

A raw binary file (Projectname.bin) has to be generated in the folder Debug\Exe by default. This device file is to be opened by onyProg2001 and issue Command (Write All) for downloading it to ATmega8515. It should be assured at this junction that the fuse bits are set as discussed above, before putting the circuit in operation. If Servos below 33 Nos. are used, the 'if statements' for controlling Servo Motor above the actual number of motors may be removed or appropriately modified. This makes the program more compact and the remaining pins can be used for added functionality of the AVR microcontroller.

The servo motor used for testing is GWS Micro 2BBMG, and it is found that end to end pulse width required for motor to rotate 0 to 180o is 800_S to 2560_S. A Servo efficiency is based on Torque / Weight ratio, the higher the value, the better it is. Any motor selected, it is better to go through its datasheet once to obtain the pulse width required for end positions and centre position. The servo used is a double ball bearing with metal gear. It is a very efficient servo but is relatively costly. A linear interpolation is used to convert angles to corresponding pulse width. A serial port can send data of one byte at a time that ranges from 0 – 255. So angles ranging from 0 to 180o is converted to 0 to 255 Scale and sent via serial port to the controller. So a minimum angle which the controller can turn is 180/255 = 0.705o.

i.e. pulse_duration = 800 + (2560 – 800)/255 _ relative position in (0 – 255 Scale)

This converts the angles to corresponding pulse duration. A delay routine is generated that requires this time to be readjusted for accounting the function call duration, 'for loop' duration and error caused due to taking integral time duration in __delay_cycles() function. So a correcting function is used as below.

corrected_duration = (pulse_duration - 3.617)/1.537

This is based on duration calculated by AVR® Studio results. A direct calling of __delay_cycles() function cannot be used as the duration required for this functions cannot be a variable. The AVR® Studio can open the Object file in Ubrof 8 format (*.dbg) generated along with the binary file. This is used to simulate the program before actual downloading.

## ServoController.C

```
/* 33 Channel Servo Motor Controller program for AVR Microcontroller
                                                        __Arun Dayal Udai
Extra lines for controlling motors other than 16 motors (0 – 15) may be removed if not used.
This Program should be compiled with IAR Embedded Workbench V4.2 */

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

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

/* Prototypes */
void USART_Init (unsigned int baud);
unsigned char USART_Receive (void);
void USART_Transmit (unsigned char data);
void Move_motor(unsigned char Motor_number, unsigned int corrected_duration);
void delay_us(unsigned int duration_in_us);

__C_task void main (void)
{
 //Set all the ports as output
 DDRA = DDRB = DDRC = DDRD= 0xFF;
 DDRE = 0x07;
 unsigned char Motor_number, Motor_position;
 /* Set the baudrate to 115.2 kbps using a 11.0592 MHz crystal (Refer Datasheet for UBRR
          values)
 Fuse bits set to handle CPU with External Crystal Oscillator */
 USART_Init (5);
 while (1)
  {
   Motor_number = USART_Receive();
//   USART_Transmit('M');
   Motor_position = USART_Receive();
//   USART_Transmit('P');

   //Rescaling to 800uS to 2560uS pulse duration
   unsigned int pulse_duration = 800 + 6.901960784 * Motor_position;
   //Correcting duration, compensating function call duration
   //For Crystal of 11.0592 MHz
   unsigned int corrected_duration = (pulse_duration - 3.617)/1.537;
   //** For Crystal of 3.6864 MHz
//   unsigned int corrected_duration = (pulse_duration - 10.85)/2.441;
   Move_motor(Motor_number, corrected_duration);
  }
}
//Initialize USART
void USART_Init (unsigned int baud)
{
 /* Set the baud rate */
 UBRRH = (unsigned char)(baud >> 8);
 UBRRL = (unsigned char)(baud);
 /* Enable UART receiver and transmitter */
 SETBIT(UCSRB, RXEN);
 /*Setting TXEN line below may be removed if any user is not willing
 to use Transmit function*/
 SETBIT(UCSRB, TXEN);
 /* Set frame format: 8 data bits, 1 stop bit */
 UCSRC = (1 << URSEL) | (1 << UCSZ1) | (1 << UCSZ0);
}
//USART Read function
unsigned char USART_Receive (void)
{
 /* Wait for data to be received */
 while (!CHECKBIT(UCSRA, RXC));
```

```
  /* Get and return received data from buffer */
  return UDR;
}
//USART Write function – used while debugging.
void USART_Transmit (unsigned char data)
{
  /* Wait for empty transmit buffer */
  while (!CHECKBIT(UCSRA, UDRE));
  /* Put data into buffer, transmits the data */
  UDR = data;
}
void Move_motor(unsigned char Motor_number, unsigned int corrected_duration)
{
  if ((Motor_number > 0) && (Motor_number <= 8))
    {
     SETBIT(PORTA, Motor_number - 1);
     delay_us(corrected_duration);
     CLEARBIT(PORTA, Motor_number - 1);
//     USART_Transmit('A');
    }
  if ((Motor_number > 8) && (Motor_number <= 16))
    {
     SETBIT(PORTC, Motor_number - 9);
     delay_us(corrected_duration);
     CLEARBIT(PORTC, Motor_number - 9);
//     USART_Transmit('C');
    }
  if ((Motor_number > 16) && (Motor_number <= 24))
    {
     SETBIT(PORTB, Motor_number - 17);
     delay_us(corrected_duration);
     CLEARBIT(PORTB, Motor_number - 17);
//     USART_Transmit('B');
    }
  if ((Motor_number > 24) && (Motor_number <= 30))
//* Modification for using 34 Channel PWM Outputs
//*  if ((Motor_number > 24) && (Motor_number <= 31))
    {
     SETBIT(PORTD, Motor_number - 25 + 2);
//*     SETBIT(PORTD, Motor_number - 25 + 1);
     delay_us(corrected_duration);
     CLEARBIT(PORTD, Motor_number - 25 + 2);
//*     CLEARBIT(PORTD, Motor_number - 25 + 1);
//     USART_Transmit('D');
    }
  if ((Motor_number > 30) && (Motor_number <= 33))
//*  if ((Motor_number > 31) && (Motor_number <= 34))
    {
     SETBIT(PORTE, Motor_number - 31);
//*     SETBIT(PORTE, Motor_number - 32);
     delay_us(corrected_duration);
     CLEARBIT(PORTE, Motor_number - 31);
//*     CLEARBIT(PORTE, Motor_number - 32);
//     USART_Transmit('E');
    }
}
void delay_us(unsigned int duration_in_us)
{
  unsigned int i;
  for (i = 0; i < duration_in_us; i++)
  {
    //For Crystal of 11.0592 MHz
    __delay_cycles(11);          // (int)(11.0592)
//    __delay_cycles(3);           // (int)(3.6864)
  }
}
```

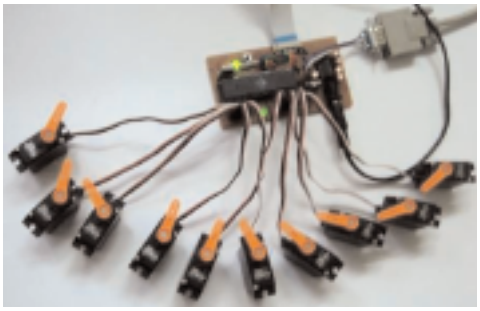
## CIRCUIT DESIGN

The circuit uses a MAXIM® MAX 232 lever converter, which converts a RS-232 level of a PC to TTL levels accepted by microcontroller. Any similar level shifter of the same series, converters from Texas Instruments or any other brands which is compatible may be used after properly referring to the datasheet. As MAX 232 supports a maximum baud rate of 120 kbps only, a standard baud rate of 115.2 kbps is used. If a single power supply is used for microcontroller as well as for the servos, it should be fitted with a heat sink. Failure of power supply may cause a high current to pass through the microcontroller and damage it in the worst case. Maximum current with LM 7805 voltage controller should be below 1A. I the current project three LM 7805 is used in parallel to use the same supply for everything including 10 servos. The power supply should account for the actual number and type of servo motors used and the current required for AVR to operate (i.e. 15mA). A separate supply for the servos may be used if it is of higher rating, with a common ground. The circuit consumes very less power in idle state, which makes the batteries if used last longer.

A provision may be given for six ISP Programmer pins MOSI, MISO, SCK, RESET, VCC and GND pins. As it the alternate functions of the same I/O pins, the microcontroller can be programmed without taking it out from the system. It can be programmed with any standard ISP programmer like STK, JTAGICE, AVRISP, ICE etc. or any other circuits discussed above.

Any clock crystal may be used, subject to the modification in the program and the fuse bits as discussed above. The microcontroller accepts a serial data in binary format i.e. it should not bear the carriage return or line feed character after the value. It expects the first byte to be the motor number ranging from 1 to 33 and second byte as the motor position in 0 to 255 scale. Position data for any motor number greater than 33 (34 if modified for handling 34 servos as commented in program) is ignored.

As per the simulation results and datasheet the first data after reset button is pressed is 6.38 mS. Motor Number and Motor position should be sent at a minimum gap of 0.994 _S. Two consecutive motor data should be sent at a minimum gap of 2271.2 _S. If any data faster than this rate is sent, the data over run will occur which may lead to undesired results. Fig. 2 shows functioning of ten servos attached to the circuit. Ribbon wire from the top in the Fig. 2 is ISP programming cable which can be detached after downloading of the microcontroller program. It shows a serial cable from the computer through which motor number and positions are sent via software program.



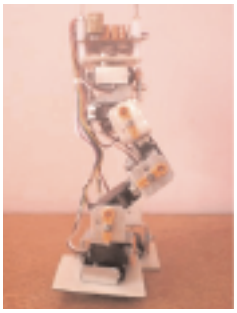


**Figure 2: Testing ten servo motors with author's prototype and the Biped using the Circuit**

## SAMPLE COMPUTER CODE IN MATLAB® AND VISUAL BASIC®

The custom code for interfacing the Motor control board may be written based on the codes as given below. A sample code is given in MATLAB® for scientific applications as utilized by advanced users for research purposes, whereas a code in Visual Basic is given for general users of Computer science streams and amateur robot developers.

### ServoControl.m

```
% A working program which turns motor 1 and 9 in %anti clockwise direction from 0 – 180o,
then %Motor 9 comes back to 0o, then both the motors %rotates in reverse direction twice,
then motor 9 %again comes to the motor 1 position.

clc;
clear;
s = serial('COM1');
set(s,'BaudRate',115200);
fopen(s);
% Both motors 0 - 180 degrees
for i=255:-2:0
   fwrite(s, 9, 'uchar')
   fwrite(s, i, 'uchar');

   fwrite(s, 1, 'uchar')
   fwrite(s, i, 'uchar');
   pause(0.01);
end
% Motor 9, 180 - 0 degrees
for i=0:2:255
   fwrite(s, 9, 'uchar')
   fwrite(s, i, 'uchar');
   pause(0.03);
end
% Motor 9, 0 - 180
% Motor 1, 180 - 0
for i=255:-2:0
   fwrite(s, 9, 'uchar')
   fwrite(s, i, 'uchar');

   fwrite(s, 1, 'uchar')
   fwrite(s, 255 - i, 'uchar');
   pause(0.01);
end
for i=0:2:255
   fwrite(s, 9, 'uchar')
   fwrite(s, i, 'uchar');
   pause(0.03);
end
fclose(s)
delete(s)
clear s
```

### ServoControl.BAS

```
'A Scroll Bar named as ServoSld which is used to 'change the servo position. Index number is
created 'for representing each servo.
'The text boxes are named as ServoTxt which is 'used to display the servo position, Index
value is 'created for each servo position display.
'A Form needs to be created with the following sub 'routines.
'**************************************
'**  Channel Serial Servo Controller          **
'**          For Robotic Applications          **
'**************************************
Private Sub cmdCenter_Click(Index As Integer)
   sldServo(Index).Value = 128
   sldServo_Change (Index)
End Sub

Private Sub Form_Load()
'    On Error Resume Next
   MSComm1.Settings = "115200,N,8,1"
   Open "16CBSCNT.CFG" For Input As #1
      Input #1, COMPort
      MSComm1.CommPort = COMPort
      MSComm1.PortOpen = True
   Close #1
End Sub

Private Sub Form_Unload(Cancel As Integer)
      MSComm1.PortOpen = False
End Sub

Private Sub Feedback_Click()
   FeedBackBox.Text = MSComm1.Input
End Sub

Private Sub mnuAbout_Click()
   About.Show 1
End Sub

Private Sub mnuExit_Click()
   Unload Me
End Sub

Private Sub sldServo_Change(Index As Integer)
'    On Error Resume Next
   txtServo(Index).Text = Str(Int((sldServo(Index).Value / 255) * 180))
   MSComm1.Output = Chr$(Index)
   MSComm1.Output = Chr$(sldServo(Index).Value)
End Sub

Private Sub sldServo_Scroll(Index As Integer)
   sldServo_Change (Index)
End Sub
```

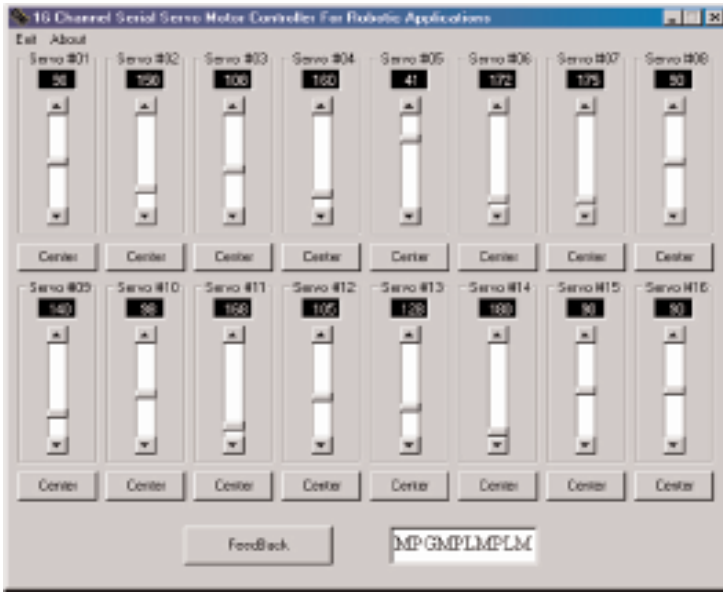



**Figure 3: Visual Basic Form layout for the Control Program**

## DESIGN STAGES

The entire schematic is firstly prepared using NI Circuit design Suite 10.0 in Multisim. A netlist file is generated with the available components and some components are custom build as per the dimensions of the available components like the ISP programming socket, Reset switch and a DC socket. The netlist (.ewnet) is then exported to Ultiboard for Board layout design, autoplacement and autorouting. The board is designed on a single sided board and the components are arranged as per the requirement manually and copper layers are adjusted manually to fine tune the system. A Gerber files are generated which is then imported to LPKF CircuitCAM 5.1 software for NC CAM profiling and tooling. From here a CAM file is generated and exported to Board Master 5.0 software as LMD file for final board printing. The system of automated system may be replaced by manual technique for amateur designers on simple breadboard too. The system for professional board development is explained for advanced readers only.

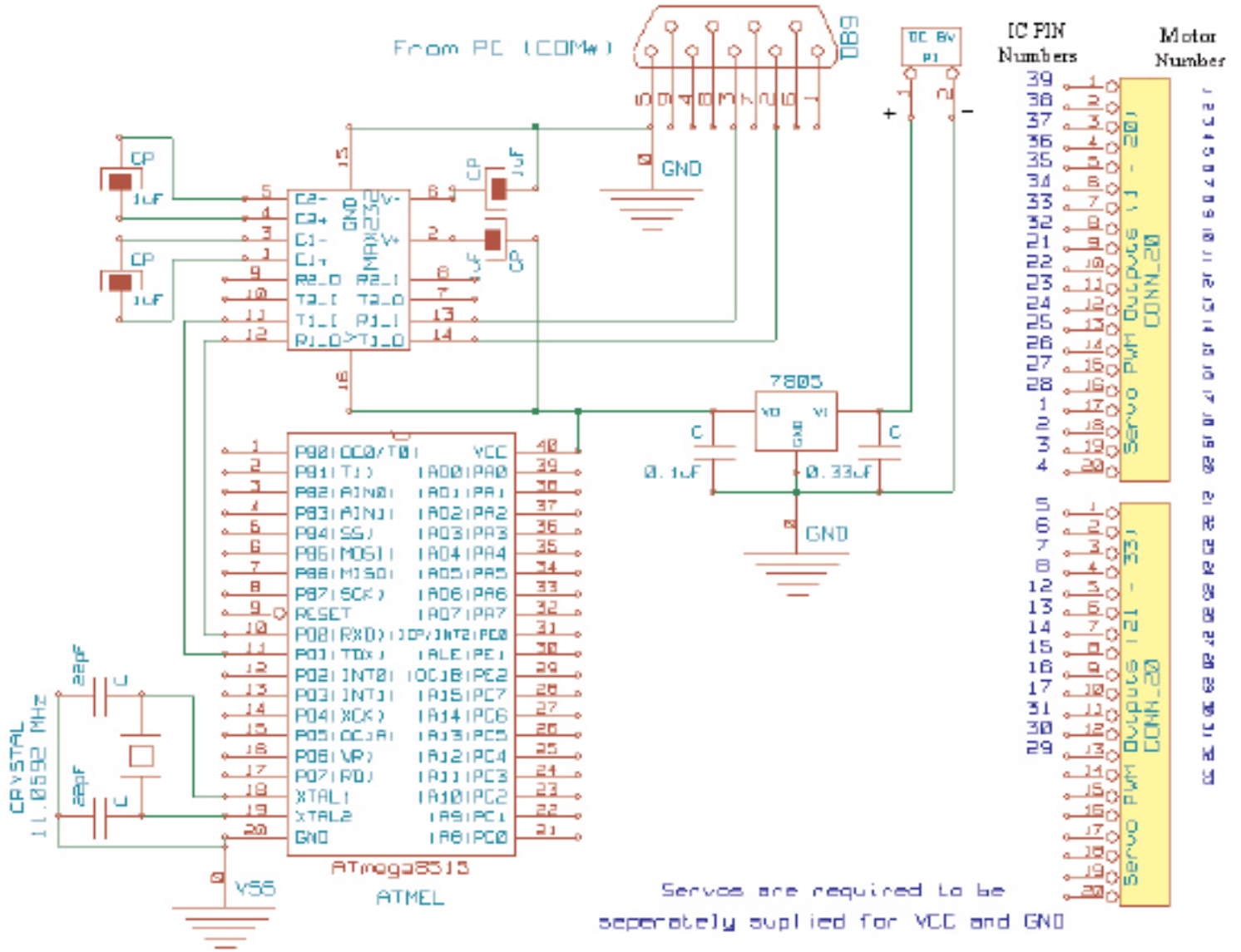



**Figure 4: Circuit Layout (Drawn using KICAD EESchema Circuit Design Tool), also developed on NI Circuit design suite 10.0 to transfer the Netlist file to Ultiboard for final board design.**

The details of these machines and design software may be obtained from http://www.lpkf.de. A series of pictures at different design stages demonstrated as follows:

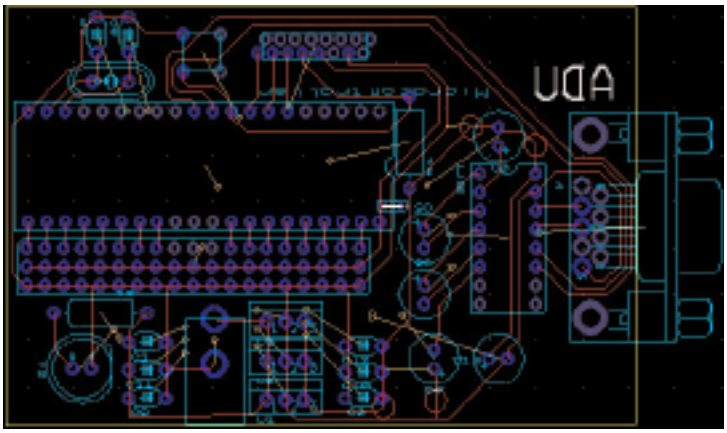


**Figure 5a: Ultiboard File showing the Copper Bottom Routes and top Components layout.**

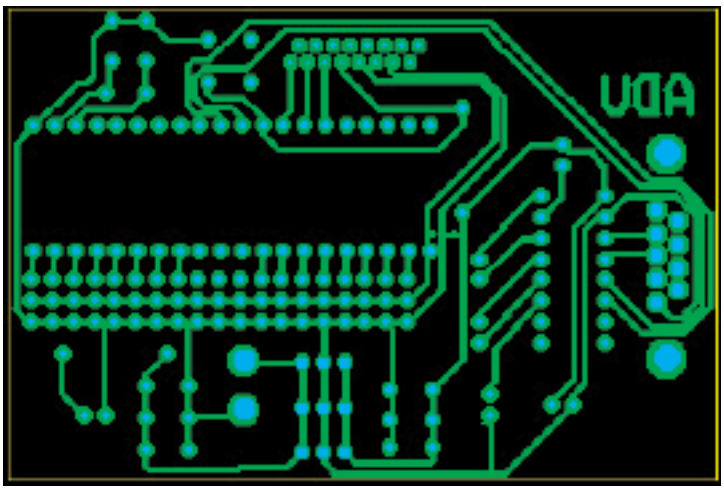


**Figure 5b: The Gerber files (Bottom Layer and the Drill Layers)**

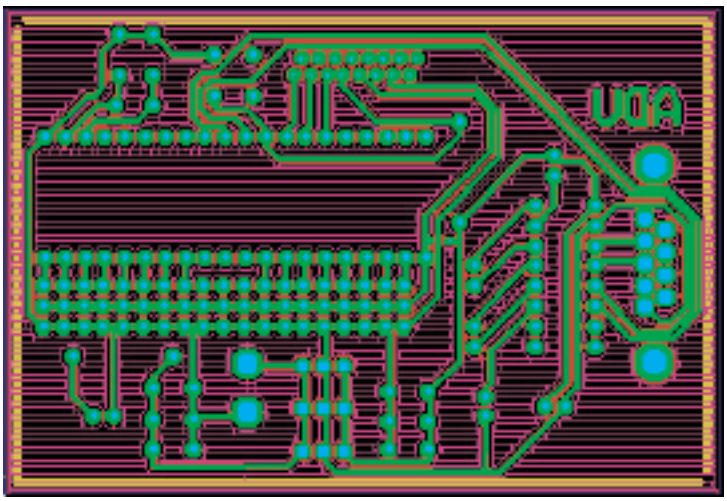


**Figure 5c: The CAM File showing NC machining profiles**

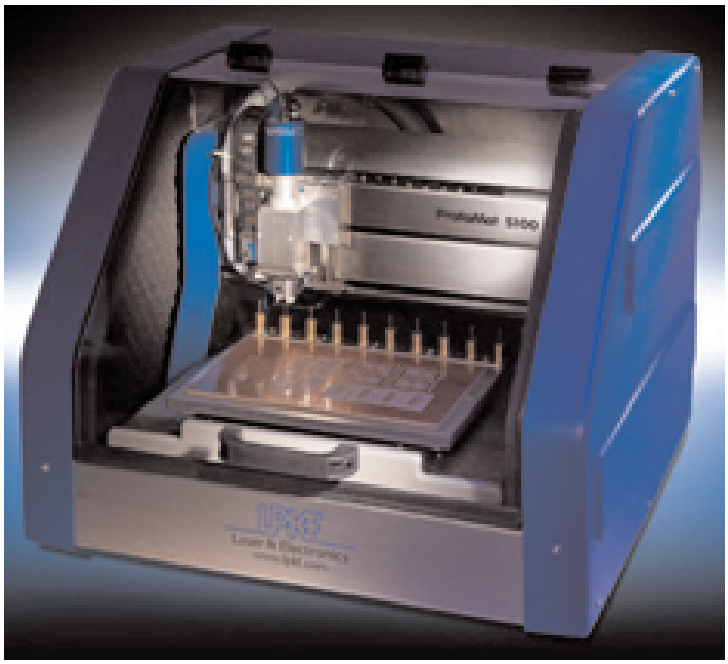


**Figure 6: LPKF® Circuit Board Plotter Machine**

| Sl. No | Component Required |
|---|---|
| 1 | ATmega8515 Microcontroller |
| 2 | Ceramic Clock Crystal (11.0592 MHz or 3.6864 MHz) |
| 3 | 22pF _ 2, 0.1_F, 0.33_F |
| 4 | LED |
| 5 | DC Socket |
| 6 | 40 Pin IC Base (0.6 inch) |
| 7 | LM 7805 |
| 8 | 1 k_ (1/4 W) Resistance |
| 9 | Blank Copper Bottom Board |
| 10 | Connecting Wires |
| 11 | Heat Sink |
| | **Total** |

**Table 1: List of discrete components required**

**Reference:** ATMEL AVR ATmega8515 Complete Datasheet