Jomo Kenyatta University of Agriculture and Technology

Department of Mechatronic Engineering

EEE 2412 Microprocessors II

Simulation of control of stepper motor with Atmega 16 on Proteus

Group Members

1. **Kilyungi John Nduli** **EN292-0396/2011**
2. **Omondi Andrew Eastman** **EN292-1658/2011**
3. **Kihara Samuel Wandai** **EN292-1649/2011**
4. **Mutuku Faridah Mwende** **EN292-0402/2011**
5. **Bartonjo Christopher** **EN292-0393/2011**
6. **Lubisia Laura Nelima** **EN292-1652/2011**
7. **Muthakye James Vundi** **EN292-0401/2011**
8. **Samer Ahmed** **EN292-1645/2011**

Introduction

A stepper motor is a brushless DC motor that divides a full rotation into an equal number of steps. It is known to convert a train of input pulses into a movement of fixed angle of the shaft.

The construction of the stepper motor consists of multiple electromagnets arranged around a gear toothed piece of iron. The electromagnets are usually powered by a driver circuit. To make the shaft to turn, one electromagnet is first given power to lead to the attraction of the gear teeth of the iron shaft. The electromagnet is then turned off and the next one turned on to lead to the shaft aligning to the other electromagnet. This causes a small rotation in the shaft. The process is then repeated to cause rotation of the shaft in these steps. Hence the term "stepper motor". With this it is therefore possible to turn the motor in specific angle rotations.

There exist several types of stepper motors. These include: -

1. Permanent magnet Stepper motor
2. Variable Reluctance Stepper motor
3. Hybrid synchronous Stepper motor

Procedure

To perform the task in hand, we used Proteus Simulation software for circuit simulation and Atmel Studio for the writing, debugging and compilation of the code.

We first acquired the datasheet of the Atmega 16 from Atmel Website and then studied the pinout diagram of the Atmega 16 to decide on our components list and how we would connect them up for the simulation on Proteus.

Working of the circuit

The circuit was to consist of two switches (to control speed and rotation), the Atmega 16 and the UL2003 (motor driver). The switches are actually push buttons to toggle between two modes of forward and reverse for the direction of the circuit and Speed 1 and speed 2.

The speed button was connected to pin5 of port D and the direction button connected to pin 7 of port D of the motor. The first four pins of port A were then connected to the motor driver circuit and then the stepper motor was then connected to the driver circuit.

The pins of the stepper motor are connected a sequential order to provide sequential stepping of the motor so as to provide rotation.

The screenshot of the circuit design is as shown below.

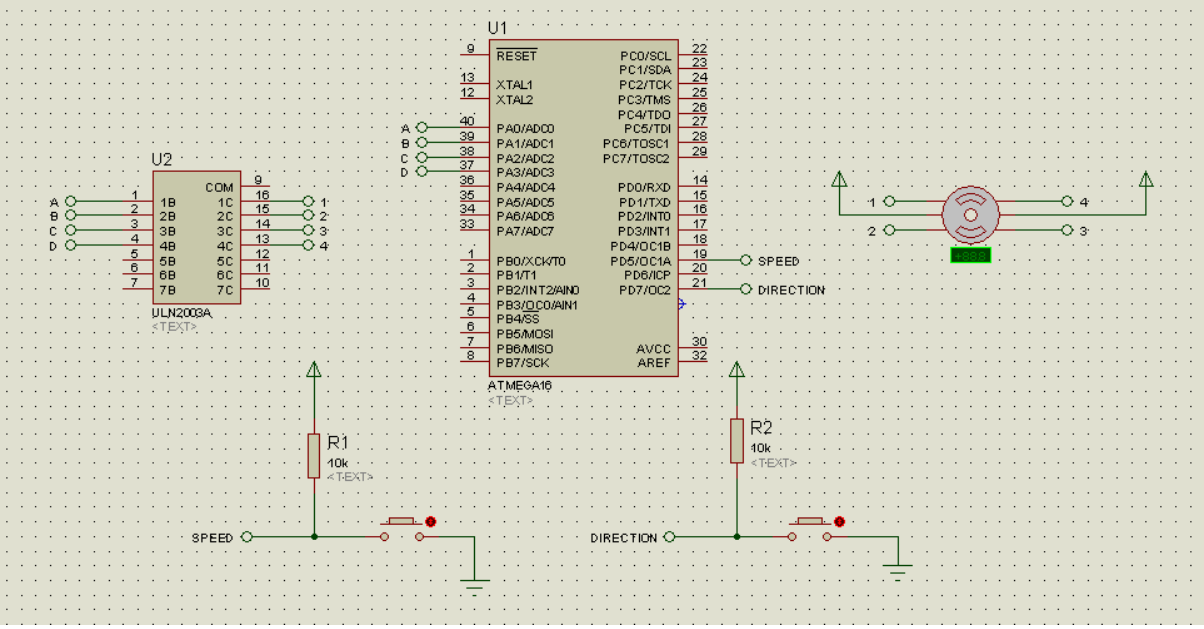
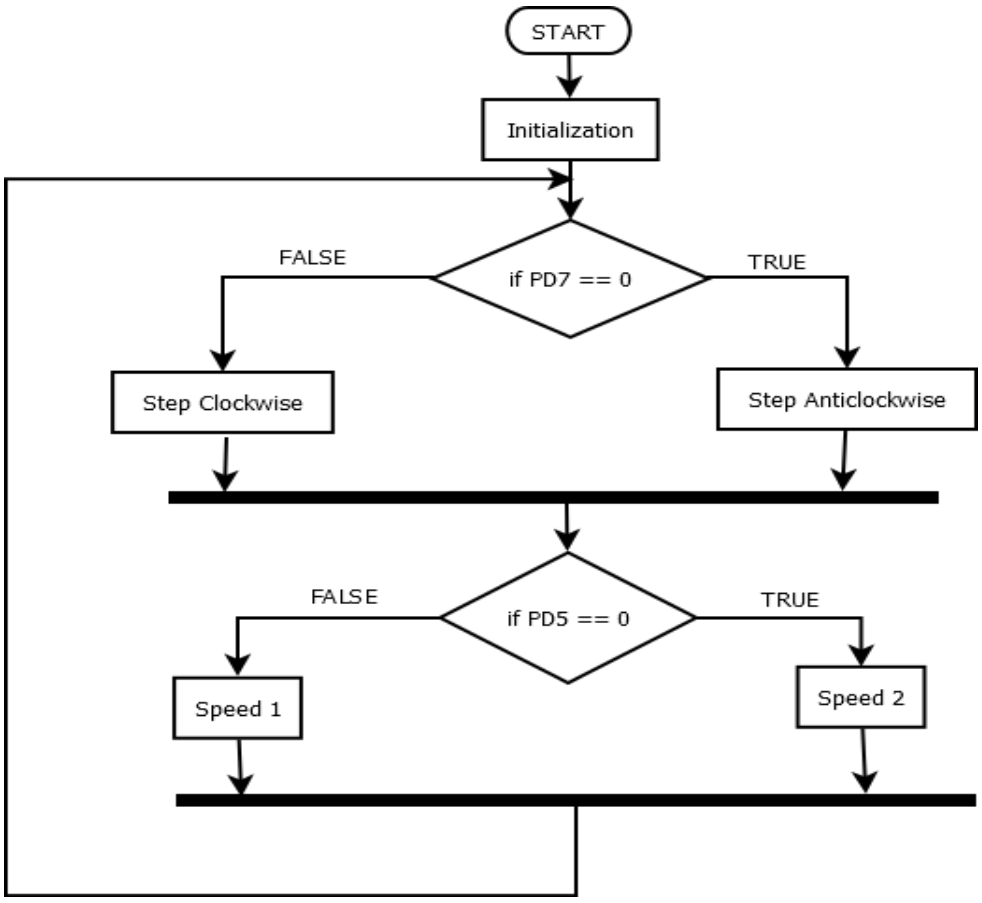


Figure 1 Circuit Design Screenshot.

The Algorithm

The code running in the microcontroller was written in assembly language. The task involved changing the direction of rotation and also changing the speed of the motor. Thus the flowchart below showed the algorithm implemented to achieve this.



The algorithm involved checking the input from the two push button and setting the appropriate action. By default, the two pins are high due to the presence of the pull up resistors at the pins. Thus to change the action one would require to hold the button to change direction or speed. Once released, rotation continues in the original direction.

The code

The instruction set for the Atmega 16 was discovered to be different from that of the intel 8086 and thus new instructions were learnt to achieve this task. The code used is as attached below.

```
/*
 * StepperMotor.asm
 *
 *  Created: 09/08/2015 10:15:08
 *   Author: Kuzan
 */

 rjmp start
 turn: .db 0x01,0x03,0x02,0x06,0x04,0x0c,0x08,0x09;
         .db 0x09,0x08,0x0c,0x04,0x06,0x02,0x03,0x01;
.equ turnsize = 0

 START:
       ldi r16, low(RAMEND)
       out spl, R16
       LDI r16, HIGH(RAMEND)       ;RAMEND is a value that show the value of the last
address
       out SPH, R16                ; intialize stack pointer to the last memory
location

       ldi r16, 0xff
       out DDRA, r16               ;set port A as output

       ldi r16, 0x00
       out DDRD, r16               ;set port D as input

init:
       ldi ZL, low(turn)
       ldi ZH, high(turn)          ;use the Z pointer to point to the data
       ldi r22, turnsize           ;reinitialize                              ;start
clockwise
       in r16, PIND
       SBRS r16,7                          ;check pin 7 if clockwise/anticlockwise
selected
       adiw zl,8                           ;if pin low, increase Z pointer to the next
array

loop:
       lpm r16,Z+
       out portA, r16                      ; output data
       rcall pause
       inc r22                                    ;increase step count
       cpi r22, 9                          ;check if stepcount is equal to 9
brne loop

rjmp init


PAUSE:
```

```
	in  r23, PIND
	LDI R16, 0x02
	SBRS r23,5
	LDI R16, 0x03		;if pin is low, change the value in the register to
change the speed
	OUT TCCR0, R16			; SETS PRESCALER OF 10248 bit timer
COMPARE:
	IN R16, TIFR		;check timer
	ANDI R16, 0X01			; tov0 flag bit
	BREQ COMPARE		; branches when overflow has not occured. This is
due to nature of and
	LDI R16, 0X01
	OUT TIFR, R16		; clear the timer overflow flag by writing one
RET
```

Working of the Code

The code first involved initialization of an array that contained the values of the step values that were to be sent to the stepper motor to provide rotation. This is the variable called **turn**.

Next, the stack pointer is SPL and SPH are initialized to RAMEND. Since the microcontroller uses 16 bit addresses, the low and high byte are initialized separately. RAMEND is a variable that specifies the last address of the microprocessor provided by m16def.inc file provided by Atmel studio to help in the programming effort. It is important to note that we initialize the stack to the last address because the stack grows with decreasing addresses(It grows downwards). Also, without initialization of the stack, it would not be possible to call function because addresses of current instruction would not be able to be pushed to a stack that is not initialized to enable resumption of operation once the called function returns.

We then initialize the port A to be output and port D to be an input port.

We then head into the label init where the Z pointer is made to point to the first element in the variable turn. We then check if the input pin 7 is low. If this is the case, the pointer is increased by 8 to point to the second array which holds the step values in the reverse order.

At the loop label, we output the data pointed by the Z pointer and then increase the Z pointer to the next element and output the next element. If the counter reaches 8 we know we are at the end of the array and thus we need to reinitialize the pointer to the start. Hence the jump to **init** again.

It should be noted that during each step, the subroutine PAUSE is called. This function creates a delay. This is because a delay between each step causes an effect on the speed of the motor. This is what was used to alter the speed of the motor.

The delay was created using the 8 bit timer present in the Atmega16. To initialise it a prescaler has to be set by sending an appropriate value to the register TCCR0 (Timer/Counter Control Register). Referring to the documentation the values used in the code are prescaler values of 2 and 3 correspond to prescaler values of 8 and 64. The prescaler changes the execution of an instruction to happen once every prescaler clock cycle. If the value is 8, instructions are executed once every 8 clock cycles. This prescaler value is set depending on whether or not the speed button is pressed or not. Since the input is connected to a pull up resistor, the value is by default set to a prescaler value of 8. When the button is pressed and held, it changes to the value of 64.

Since the timer is 8 bit, it can only count up to a max value of 2^8-1 which is 255. Thus when this value is reached, an overflow bit is set in the TIFR (Timer/Counter Interrupt flag register). We therefore monitor this bit(bit 0) and therefore stop looping. We restore the flag value to its original value and then the function returns.

Essentially, the delay function therefore works by counting from 0 to 255 but at different rates set by the prescaler value changed by the pin.

Conclusion

The circuit simulation was shown to move in the reverse and forward direction by use of the push button. The speed could also be altered by use of the other button. However, the simulation was seen to show some jittering during rotation. This could be blamed to the fact that the simulation produced by Proteus was not real-time. Despite this, the task was successfully achieved.