Chapter 4
Computer Interfacing and Programming

4.1 Introduction

As was mentioned earlier, the computer interface is used to control the computer processor to send step and direction signals to the feed drives. C codes were written on DOS to build the interface. The codes will include predefined values of the characteristics of the required motion in addition to commands for sending pulses from the data port of the parallel port as going to be discussed later. The printer cable is used to connect the parallel port to three pairs of clock and the direction pins on three motor drive boards, two switch limit circuits and a relay circuit. Figure 4.1 demonstrates the pin assignments on a 25 DB Female side connector.

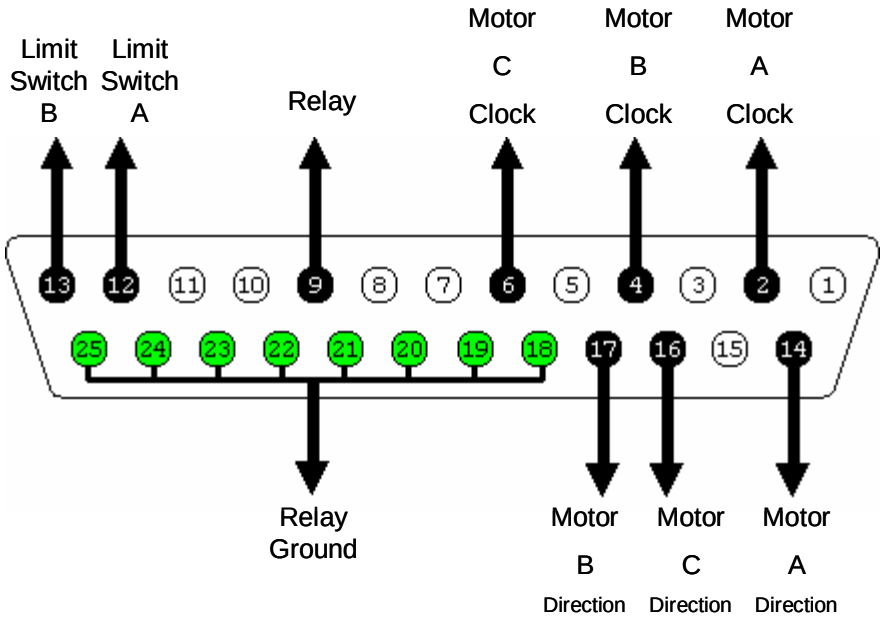

Figure 4.1: Pins assignments on a 25 DB female side connector

Analysis of kinematics of motion should be done before writing the codes. This includes motion in one access, and motion in the 2D plane. A trapezoidal velocity vs. time motion shape will be considered, where the velocity will increase from zero value at a constant rate until reaching a constant speed, before decreasing, at a rate equal to the increase rate, to zero velocity.

4.2 Parallel Port Interfacing

4.2.a Parallel Port Addressing.

The first step reasonably is to find out the base address of the PC Parallel Printer Port. Each printer port consists of three port addresses; data, status and control port. These addresses are in sequential order. That is, if the data port address was *x*, where *x* is a hexadecimal number, the corresponding status port and control port addresses will essentially be *x+1* and *x+2*, respectively. The port address can be obtained by taking the following steps on a Windows 95 or higher operating system:

Right click on the My Computer icon and select properties → Hardware → Device Manager → right click on Printer Port and select properties → Resources

You will be given an I/O range. The number to the left is the data port address. For Example: I/O Range 0378 – 037F

The data port address is then the hexadecimal number 0x378, which is most common for built-in printer ports. The port addresses are defined in C as constant values:

```
#define DATA 0x378
#define STATUS DATA + 1
#define CONTROL DATA + 2
```

4.2.b Parallel Ports pins assignments.

The parallel ports pins assignments are demonstrated at figure 4.2:

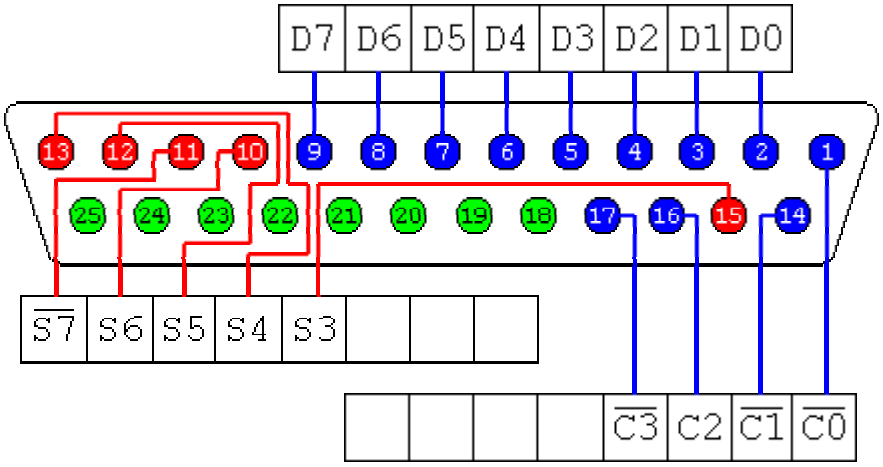

Figure 4.2: Pin Assignments for female side printer parallel port connector

The Letter D, S, and C, corresponds to Data, Status and Control ports, respectively. There are eight outputs on the Data Port (Data 0 - Data 7) and four additional outputs on the Control Port (Control 0 – Control 3). A signal can be outputted from any of these pins by writing to the corresponding bit, on the required port. C uses the function outportb to send signals from the parallel port. It takes two arguments: the port being used, and the hexadecimal representing, the pin/pins from which signals are going to be sent. For example:

```
outportb(DATA, 1);
outportb(CONTROL, 0);
```

The first sentence is used to send a +5V step signal from pin 2 represented by Data bit 0. This is called true logic. This means that writing a logic one to a bit causes the

corresponding output to go high, and vice versa. All output pins on the data port in addition to pin 16 (Control 2) are of true logic. The second sentence is used to send a +5V step signal from pin 14 represented by Control bit 1. This is because Control 1 is inverted, which means that writing a logic one to it will set it low, and vice versa. Control bits 0 and 3 are of the same nature.

There are five inputs on the Status port (Status 3 –Status 7). Input pins are used to read signals received by the parallel port. C will use the function inportb to read the input signal to a pin or a group of pins. For example:

detectenda = inportb(STATUS);

The hexadecimal generated by a sequence of high and low bits on the statues port is stored at the variable detectenda. Notice that the Status bit 7 should always be inverted using exclusive OR function in order to obtain the correct number on the Status port.

4.3 Motion along one axis

In this section, the motion of the nut along the screw is analyzed. The pulse frequency will be increased to a certain predefined value before it starts decreasing to zero again. The number of pulses generated per second is proportional to the motor RPM and so, to the linear speed of the actuator. As a result, the actuator will be accelerating at the beginning of its motion, until it reaches a certain constant speed. When the required end position is about to be reached, the actuator will decelerate to a zero speed at the end position.

We will assume that the number of stepper motor steps is equal to the number of pulses sent from the computer to the drive board. The physical equations describing the signal required to obtain the actuator motion described above were deduced as follows:

Number of steps required to reach a maximum frequency f_{max} is equal to N_1. The frequency will remain constant for N_2 steps at a value of f_{max}.

Total number of steps N_{total}, $N = N_1 + N_2 + N_3$ where N_1 is the number of steps sent at constant rate f_{max}, N_3 is the number of steps left to decelerate to zero. The time intervals at which N_1 , N_2 and N_3 pulses will be sent, are T_1 , T_2 and T_3 respectively.

The frequency increase rate in the beginning of the motion is assumed to be equal to it's decrease rate. Thus, if the maximum acceleration was taken to be a_{max} , then the maximum deceleration is $-a_{max}$. The acceleration time T_1 is then, equal to the deceleration time T_3, and the total number of steps can be described as

$$N_{total} = 2\,N_1 + N_2 \qquad (4.1)$$

Figures 4.3, 4.4 and 4.5 represents the acceleration *a*, frequency *f*, and number of steps respectively *n*, vs. Time.

From figure 4.2, we can deduce that:

$$a_{max} = f_{max} / T_1 \qquad (4.2)$$

$$N_1 = 0.5\, f_{max} . T_1 \qquad (4.3)$$

Thus, given the values of f_{max} and a_{max}, we can obtain N_1 and T_1.

The first pulse is sent at t_0. At the instance t_1, sending the first pulse will be finished. So, the *n*th pulse will be actually sent at t_{n-1}, and the sending will be finished at t_n.

The total number of pulses *n* that was already sent at an instance t_n, in the time interval from 0 to t_n , can be evaluated as

$$n(t_n) = \int_0^{t_n} a_{\max} t_n \, dt_n \qquad (4.4)$$

$$n(t_n) = \frac{a_{\max} t_n^{\,2}}{2} \qquad (4.5)$$

from which, $$t_n = \sqrt{\frac{2n(t)}{a_{\max}}} \qquad (4.6)$$

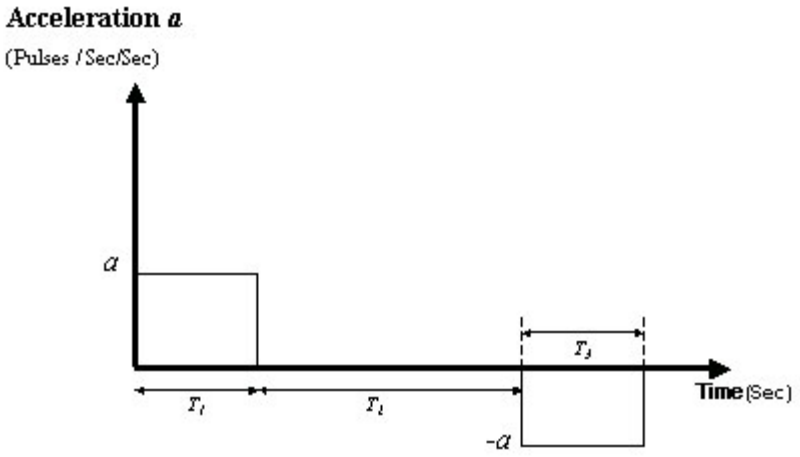

Fig 4.3: Acceleration vs. Time

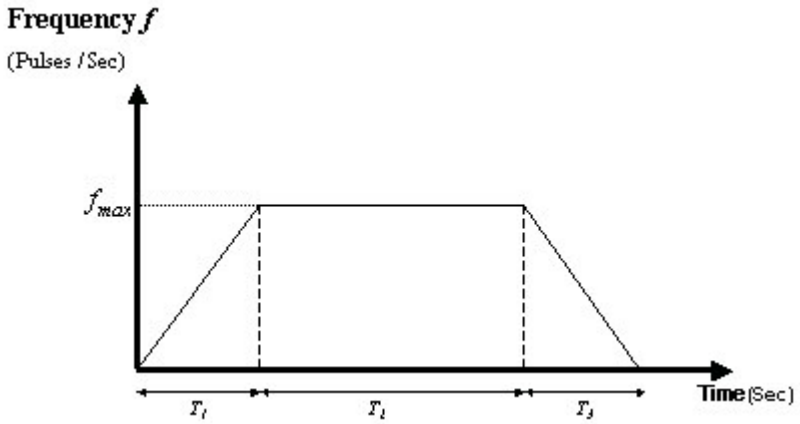

Fig 4.4: Frequency vs. Time

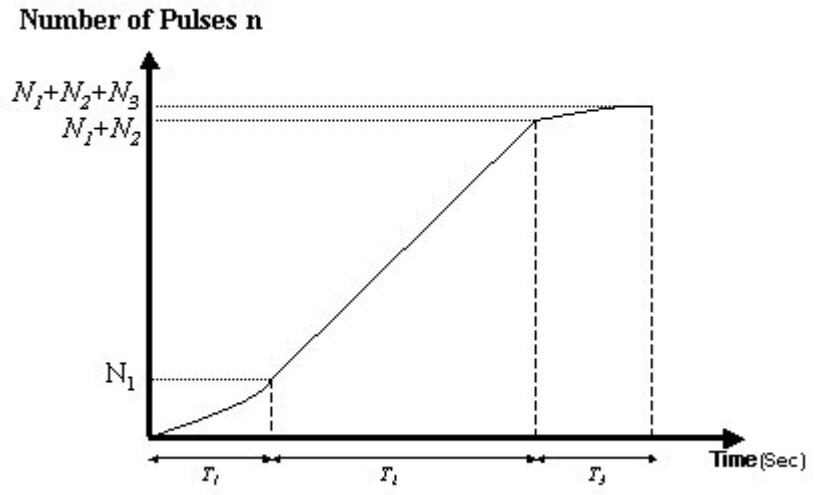

Fig 4.5: Number of pulses vs. Time

Thus, the time required for the *n*th pulse to be sent (periodic time) is

$$T_{pn} = t_n - t_{n-1} \quad (4.7)$$

and $$T_{pn} = T_{x\,n} + w \quad (4.8)$$

w is a predefined value for the pulse width that can take any value lower than the reciprocal of the maximum frequency f_{max}. T_{xn} is the variable part of the periodic time. Decreasing T_{xn} will Accelerate pulse sending and vice versa. From (4.7) and (4.8)

$$T_{xn} = t_n - t_{n-1} - w \quad (4.9)$$

Values of the maximum velocity f_{max}, maximum acceleration a_{max}, and total number of pulses N_{total} should be initially defined. The maximum speed of the nut, is limited to 3.0003 mm/s for Δt = 5 msec at which the maximum frequency is 200 Hz corresponding to a full motor shaft revolution per second, 1:1 transmission ratio and 3 1/3 threads per cm of the screw length. The number of pulses that will be sent at maximum frequency f_{max}, N_2 can be obtained from (4.1). C represents constant speed, and deceleration regions by three do-loops. Each loop remains running while the time is not exceeding T_1 for the

acceleration loop, T_2 for the constant speed loop and T_3 for the deceleration loop. The idea of the program is to send a pulse of width w before sleeping a period T_{xn} defined at each step n.

While T_{xn} has a constant value at the constant frequency region, that is equal to $(1/f_{max})$-w, (4.6) *and* (4.9) are going to be used with an ascending counter to accelerate pulse sending, and a descending counter to decelerate pulse sending.