ON-LINE MICROCOMPUTER CONTROL OF DELTA MODULATED INVERTERS.

by

Mohammad Nazmul Anwar

A Thesis

Submitted to the Department of Electrical and Electronic Engineering in partial fulfillment of the requirements for the degree

of

Master of Science in Electrical and Electronic Engineering.

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING.
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY.

DECEMBER, 1995.

DEDICATED TO

ENGINEERS

WHOSE ENDEAVORS HAVE USHERED US UP
TO THIS POINT.

DECLARATION

I hereby declare that this thesis work has not been submitted elsewhere for the award of any degree or diploma or for publication.

Countersigned :

machoudhury

(Dr. Mohammad Ali Choudhury) **(Mohammad Nazmul Anwar)**

The thesis titled, **"On-line Microcomputer Control of Delta Modulated Inverters."**, submitted by Mohammad Nazmul Anwar, Roll No.: 921345P, Registration No.: 87218 of M. Sc. in Engineering has been accepted as satisfactory in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering.

Board of Examiners:

1. machoudhury Dec 31, 1995

Dr. Mohammad Ali Choudhury — Chairman
Associate Professor — (Supervisor)
Department of Electrical and Electronic Engg.,
BUET, Dhaka-1000.

2. 31/12/95

Dr. A.B.M. Siddique Hossain — Member
Professor and Head — (Ex-Officio)
Department of Electrical and Electronic Engg.,
BUET, Dhaka-1000.

3. 31/12/95

Dr. Md. Saifur Rahman — Member
Assistant Professor
Department of Electrical and Electronic Engg.,
BUET, Dhaka-1000.

4. 31.12.95

Dr. Kazi Khairul Islam — Member
Associate Professor — (External)
Department of Electrical and Electronic Engg.,
BIT, Rajshahi.

ACKNOWLEDGMENT

My sincerest gratitude goes to Dr. Mohammad Ali Choudhury without whose ardent initiatives, compassionate advice and erudite guidance this research work would not have materialized. I am indebted to him not only for making the endeavor a success, but also for arousing keen interest in the field of Power Electronics and nurturing my ability to conduct such kind of research in future.

I wish to express thanks and regards to Dr. A.B.M. Siddique Hossain, Professor and Head of the Department of Electrical and Electronic Engineering, BUET for his encouragement and co-operation to complete this work.

My heartiest thanks are to Dr. Md. Saifur Rahman and Dr. Joarder Kamruzzaman, assistant professors of the Electrical and Electronic department, BUET whose invaluable suggestions are gratefully acknowledged.

Sincerest thanks are also due to my friends Mr. A.S.M. Monzurul Morshed, Mr. Md. Manzur Murshed and Mr. Md. Khairul Hasan for their co-operation during the course of this thesis work.

ABSTRACT

A novel and simple on-line microcomputer generation of PWM waveforms to control static converters is presented in this thesis. On-line implementation is based on newly proposed algebraic equations to calculate switching points. Implementation has been carried out on an Intel 80386 DX-2 based microcomputer using high level turbo C++ language. The system uses keyboard as input, processor for calculation and pin-2 through pin-7 of the parallel port to output switching pulses. No other external circuit is necessary as required in other microcomputer controlled pulse width modulated converters thus far reported in literature. The implementation allows the operator to vary any of the on-line modulation parameters i.e. window width, modulating wave's frequency, magnitude and slope of the carrier wave in the proposed on-line microcomputer controlled delta modulation technique. At present, this implementation allows parameter variation by keyboard input at any point of operation without interruption of service. In this thesis the proposed method is described with analysis, implementation outlines and results of three phase inverter operation with resistive loads.

PWM schemes provide simultaneous voltage and frequency control of inverters. Until now, the switching signals for PWM schemes are achieved by Off-line

computation of switching points for various operating frequencies and storing them in EPROM as a look-up table and then outputting the waveform for a desired frequency of operation as required. This implies that a very large number of tables must be stored in memory in order to keep the frequency and voltage resolution within the acceptable limits, other wise a step variation of fundamental voltage during frequency control leads to undesirable current surges. Even if we think of a very large memory, the present microprocessor-based modulators suffers from inadequate and poor angle resolution response in real-time with change in voltage and frequency commands. On-line computation was not feasible thus far because, calculation of switching points of PWM waveforms required the solutions of transcendental equation and as a result instantaneous computation was not possible. In a recent research a novel and simple algebraic equation has been developed to find switching points of sine delta modulated wave for analysis of inverter waveforms.

To alleviate the problems related to frequency limited poor switching angle resolution of off-line schemes, the new switching point calculation technique is used in an on-line microcomputer based Delta-PWM technique in this thesis work which uses only an INTEL 80386 DX-2 microcomputer to solve the algebraic equation to find switching points and output the generated pulses through parallel port of the microcomputer. The on-line microcomputer generated sine delta PWM waveforms have been used successfully in the operation of the both single-phase and three-phase voltage source inverters with on-line parameter control by keyboard input.

CONTENTS

CHAPTER-ONE: INTRODUCTION

CHAPTER-TWO: REQUIREMENTS OF ON-LINE MICROCOMPUTER CONTROLLED DM INVERTER OPERATION

LIST OF SYMBOLS

f_m	Frequency of the modulating signal.
ω_m	Angular frequency of the modulating signal.
Vm	Maximum voltage level of the modulating signal.
t_m	Time period of the PWM pulse pattern.
s	Slope of the error signal at the output of the integrator.
dv	Width of the window envelope.
num_seg	Number of time segments to be scanned in a total time period.
(N+1)	Number of timing instants in the first half cycle.
t[n]	Array of the timing instants.
dlay[n]	Array of time durations of positive or negative voltage levels.
del	Time duration of each time segment.
init[k]	Value of initial position of the pulses in time scale.
r	Segment number.
YY	Total number of iteration used in the dummt for-loop 'empty'
deci[num_seg]	Array of decimal numbers which corresponds the pulse pattern

LIST OF ABBREVIATIONS

ACC	Accumulator.
ALO	Arithmetic Logic Unit.
BIOS	Basic Input Output System.
CPU	Central Processing Unit.
CMOS	Complementary Metal Oxide Semiconductor.
DMA	Direct Memory Access.
DPWM	Delta Pulse Width Modulation.
EMI	Electromagnetic Interference.
EPROM	Erasable Programmable ROM.
HSI	High Speed Input.
HSO	High Speed Output.
IGBT	Insulated Gate Bipolar Transistor.
LED	Light Emiting Diode
MIPS	Millions of Instructions Per Second.
MAT	Memory Access Time.
PWM	Pulse Width Modulation.
PTDMWG.CPP	Program for Three-phase Delta Modulated Waveform Generation With Extension of CPP.
PSDMWG.CPP	Program for Single-phase Delta Modulated Waveform Generation With Extension of CPP.
PDMA	Programmable Direct Memory Access.
RDPWM	Rectangular Wave DPWM.
RAS	Reliability, Availability, and Serviceability.
ROM	Random Access Memory.
RAM	Read Only Memory.
SMPS	Switch Mode Power Supply

SPWM	Sine Pulse Width Modulation.
TTL	Transistor Transistor Logic.
VCO	Voltage Controlled Oscillator.
VLSI	Very Large Scale Integration.

LIST OF FIGURES

LIST OF TABLES

LIST OF FLOWCHARTS

CHAPTER-ONE

INTRODUCTION

□ INTRODUCTION

□ LITERATURE REVIEW

□ REVIEW OF PWM TECHNIQUES IN STATIC INVERTERS

□ REVIEW OF MICROCOMPUTER CONTROLLED PWM TECHNIQUES FOR STATIC INVERTERS

□ THESIS OBJECTIVES AND OUTLINES

1.1 INTRODUCTION

Microcomputer-based intelligent motion control systems offer a significant participation in modern industrial automation. It is going to be the leader in this field in the foreseeable future. In the early 1970's the advent of microcomputers intensely influenced motion control systems, not only by simplifying the control hardware, but by adding intelligence as well as diagnostic capability to the system. In an automated industrial environment on-line control system is essential which makes decisions about actions based on required strategy, embracing many diverse disciplines of electrical machines, power semiconductor devices, converter circuits, dedicated hardware signal electronics, and control theory. From the view point of higher-level and as well as lower-level automated supervisory control, power electronics and drive technology, which is already interdisciplinary, has reached a new dimensions of complexity. To successfully overcome the complexities, on-line microcomputer control is the best available helping hand to power electronics. An electronic power converter translates the switching signals to control the output power On-line control of switching signals to the converters is necessary for control with a microcomputer to improve reliability and stability, to eliminate drift and electromagnetic interference problems, and to design universal hardware and flexible open/closed-loop control. In an on-line control, software can be altered as the system performance demands.

Inverters are dc to ac static converters. The simplest of the inverters is the voltage-fed inverter where a dc is converted to a square wave of particular frequency. Since the ac wave is non-sinusoidal, researchers introduced modulated switching for inverters to reduce low order harmonics so that small size filters can be used to minimize harmonics, losses, and other side effects encountered in various applications. PWM schemes also provide simultaneous voltage and frequency control of inverters. At present the switching signals for PWM schemes are achieved by Off-line computation of switching points for various operating frequencies and storing them in EPROM as a look-up table and then outputting the waveform for a desired frequency of operation as required. Stand Alone on-line computation was not feasible thus far because, calculation of switching points of PWM waveforms requires the solutions of transcendental equation to allow instantaneous computation. In a recent research a simple algebraic equation has been developed to find switching points of sine delta modulated wave for analysis of inverter waveforms [1], [2].

In the proposed research, the newly developed algebraic equation has been implemented to produce delta PWM switching signals by using high-level turbo C^{++} language at different operating frequencies for various modulation parameters in the logic to control a single phase and a three phase inverter. The switching signals thus generated by an INTEL 80386 DX-2 processor has been brought out of the computer through the parallel port and interfaced to an external logic board to switch an inverter. This implementation supports variation of modulation parameters by keyboard input at any point of operation without interruption of the service of inverter.

1.2 LITERATURE REVIEW

Literature review for this thesis work will be discussed under two subtitle. The first one will expound the chronological development of pulse-width modulation (PWM) techniques for static power converter operation and second one will take a look over the microcomputer-based generation of PWM switching pulses reported in literature so far.

1.2.1 REVIEW OF PWM TECHNIQUES IN STATIC INVERTERS

Static converters that convert dc power to ac are known as inverters. According to their source in general there are two types of inverters: *voltage-fed inverters and current-fed inverters.* The output of normal square-wave voltage-fed and current-fed inverters are non-sinusoidal and contain low frequency harmonics. Besides, the utility line power factor deteriorates due to phase shifts and the system may suffer instability at low speed due to the low-pass filter in the dc link for these types of inverters.

Problems stated above have led to the developments of pulse width modulated (PWM) inverters where the switching pulses of the power converters are operated at higher frequencies following a particular modulation pattern to enjoy the following advantages,

a) to achieve simultaneous voltage, frequency, and harmonic contents control of inverter output in a single power stage. [4] - [12].

b) to perform harmonic reduction and ripple elimination using small size filters at the input/output of the converters resulting low loss.

c) to enhance performance and conversion efficiency of converters.

d) to improve operating power factor. [13].

There are several types of modulation schemes to switch inverters, these are ,

i) Single pulse-width modulation.

ii) Multiple pulse-width modulation.

iii) Sine pulse- width modulation (PWM) [14].

- Asynchronous
- Synchronous

- Regular Sampled

iv) Optimal PWM

v) Delta PWM (DPWM)

The initial technique of modulation applied to inverter operation were *single pulse-width modulation* and *multiple pulse-width modulation* [15] -[19], [8], [12]. In single pulse-width modulation technique, there is only one pulse per half cycle and the width of the pulse is varied to control the output voltage of the inverter. Here the switching signals are generated by comparing a rectangular reference signal with a triangular carrier wave. The frequency of the reference signal determines the fundamental frequency of the output voltage. In this technique the dominant harmonic is the third harmonic, and the distortion factor increases significantly at low output voltage [20] -[22], [7].

In multiple pulse-width modulation technique, the harmonic content and the distortion factor of single pulse-width modulation is reduced by producing several pulses in each half-cycle of output voltage. However, due to large number of switching per half-cycle, the switching loss and the amplitude of higher order harmonics would increase although the amplitude of lower order harmonics would decrease.

Among several PWM techniques in industrial applications, sine pulse-width modulation (SPWM) is common [22] -[32], [9], [11]. In sine pulse-width modulation technique, a high frequency isosceles triangle carrier wave is compared with a fundamental-frequency sine modulating wave, and the natural points of intersection determine the switching points of power drives. As a result, in SPWM , instead of maintaining same width of each pulse as in multiple pulse-width modulation the width of each pulse is varied in proportion to the amplitude of the sine wave evaluated at the center of the same pulse. Eventually the distortion factor and low order harmonics are reduced significantly [33[-[39], [8], [9], [20].

As for drive applications, the fixed frequency modulation has been proved to be problematic at different operating frequencies and the ordinary SPWM was replaced by variable ratio PWM schemes. At present, three distinct SPWM schemes are in use for inverters [40] -[42], [32]. These are: *Natural sampling method, Regular sampling method,* and *Optimal switching strategy.* Natural sampling method resembles the multiple pulse modulation where a triangular wave is compared with sine wave to generate PWM pulses. The sine wave is replaced by a sampled or a stepped sine wave in the regular sampling method. This method is very common in microcomputer implementation [43], [44]. The third scheme uses optimized switching strategies based on certain performance criteria [45]. Evolution of microprocessor technology supports the implementation of optimized pulse-width modulation for switching inverters [46] -[48].

A recent modulation method for switching inverters is the Adaptive PWM technique utilizing hysteresis control which is referred to as Delta pulse-width modulation (DPWM) [49]. So far for various power converter operations, use of several types of DPWM have been investigated [50], [51]. In delta modulation, a triangular wave is allowed to oscillate within a defined window above and below the reference sine modulating wave. This simple and idealized PWM scheme offers the advantages of continuous converter voltage control and a direct control on the line harmonics. A detailed description and explanation on DPWM is outlined in the next chapter.

1.2.2 REVIEW OF MICROCOMPUTER CONTROLLED PWM TECHNIQUES FOR STATIC INVERTERS

Analog implementation of PWM switching circuit suffers from the problems associated with thermal drift, component tolerance, dc offset and imperfections in the amplitude and phase of three-phase analog sine-wave generation which produces harmonic effects in the inverter systems. Regular switching action of analog PWM circuit also generates conducted and radiated electromagnetic interference (EMI), and may also generate acoustical disturbance [52].

The above problems are solved by microcomputer-based digital control technique of switching pulses which have profoundly influenced power electronics and static drive system for industrial automation [4] -[8], [20].

In contrast to the rapid development of single-chip microcomputers in the early 1970's, works on microprocessor based power electronics can be traced back to early 1977 at the furthest. The first computer revolution began with Von Neumann's work in 1945 [53], and second computer revolution began with the commercialization of the microprocessor in 1971, when INTEL introduced the 4004, 4-bit single-chip central processing unit (CPU). From the first generation of microprocessor using 4-bit architecture, 32/64-bit microprocessors are now available having tremendous speed like 100 MHz [36]. Microprocessor has largely contributed the control system to provide them with powerful RAS (Reliability, Availability, and Serviceability), together with improved flexibility & performance control system. The trend is found not only in the rising robotics control, factory automation and so forth but also in the replacement of plants in the established industries.

The control of static converters and drives by microprocessor can easily be recognized for complex drive control system. The digital control has inherently improved noise immunity which is particularly important for power electronics because of large power switching transients in the converters. The works on microprocessor-based firing schemes for three-phase full-wave dual converter [54], [55], for phase controlled rectifier connected to a weak ac system [56] for thyristor choppers [57], [58], have proved the authentic implementation of microprocessors in this field to optimize drive system performance to improve power factor and effective diagnostics of the relevant system. These are achieved using I/O interfaces and peripheral equipment as programmable counters, programmable memory, interrupt controller, VCO, multipliers, D/A & A/D converters etc. together with the CPU microprocessor use which utilizes look-up tables for firing control . As a result, these techniques can not prove smooth control and high resolution over wide range of variation of the control variable.

Microprocessor- based control of dc motor drives [59]- [61], electric vehicles [62], and ac drives like induction motor [63]- [65], synchronous motor [66], commutatorless motor drives [67], and reluctance motor [68] etc. is being implemented for automatic and efficient control using dedicated hardware and software.

To write about microprocessor controlled inverters, an expound review of microprocessor-based PWM waveform generation necessarily precedes since PWM is by far the best available technique for the production of switching pulses of an inverter. The microprocessor-based generation technique is advantageous not only for PWM waveform but also for high-resolution real-time sine wave generation by programmable logic cell array for the same inverter. Real-time implementation [69] of SPWM switching strategies for inverters by Optimal, Regular and Natural sampling method using microprocessors has been a constant effort of the researchers [70], [71]. The implementation techniques reported are based on transcendental equations relating PWM switching angles and lacks generality when implemented by microcomputer [72]. The computation of switching points has so far been complex and time consuming in the absence of substantially developed algebraic equation requiring solution of transcendental formulas. As a result the switching points are stored in EPROM based look-up tables computed on off-line basis. This implies that a very large number of tables must be stored in memory in order to keep the frequency and voltage resolution within the acceptable limits, other wise a step variation of fundamental voltage during frequency control would lead to undesirable operations of inverters [71].

A possible solution to this drawback in the implementation of optimal PWM waveforms is to store the PWM patterns corresponding to selective frequencies and to use interpolation to compute the intermediate patterns [73], having direct multiplication and division capability with co-processors. This approach posses similar implementation difficulties because of off-line calculations [74]. Sampling techniques posses well-defined modulation procedures which reveals the possibilities of developing real-time algorithms capable of generating PWM control in the frequency range of the inverter having minimal storage requirements [75]. The ratios of carrier to fundamental frequency tends to be very high at low frequencies which needs huge tables of switching angles for optimized or harmonic elimination strategies. Moreover, optimized waveforms have no significant advantages over sampling waveforms as far as harmonic performance is concerned at low fundamental voltage. The number of notch-angles for a switching pattern tends to increase at lower fundamental frequency, demanding large look-up table memory. As a result, the lower frequency and the number of switching patterns stored tend to be limited. Even if we think of a very large memory, the present microprocessor-based modulators suffers from inadequate and poor angle resolution response in real-time with change in voltage and frequency commands. Techniques to reduce the computational requirements for SPWM switching has been studied [70], [71], [76] and to save CPU time the DMA technique is used as reported in reference [77] for transferring the switching pattern from memory to the pulse amplifier . The DMA transfer methods have deficiencies as mentioned above. Attempt has also been made to implement three-phase PWM waves by hybrid implementation with computation intensive uniform sampling method in the low -frequency region and look-up tables in the higher frequency region [78].

A programmable SPWM scheme to generate switching pulses with display provision associated with converter output voltage- current waveforms and their respective frequency spectra has been reported in reference [79] which uses peripheral digital IC's like PDMA-16 card and mass storage medium for implementation. This implementation method takes a total calculation and preparation time of approximately 120 seconds, of which 110 seconds were spent to compute the required gating signals for using non-linear equations and 10 seconds were used to prepare and deliver the gating signals to the converter.

In the work reported in [52] programmed optimal SPWM techniques have been applied using methods to control amplitude of harmonic peaks by the EMI generated by the switched power circuits. This technique has been implemented by 64 bytes of EPROM and 16 MHz oscillator and presettable downcountes as peripheral circuitry. Here the relatively large sensitivity of the spectrum to random perturbation (such as round-off) in the switching instants can cause difficulties.

The above types of implementation needs microcomputer along with peripherals of digital IC's of on-chip 8-or 10-b A/D converters, 28 interrupt sources, programmable high speed input (HSI) and programmable high speed output (HSO) [79], PDMA-16 card and mass storage medium [78], DMA chips and 256 x 4 RAM'S [76], 8259 interrupt controller , 2.5 Kb EPROM, oscillator etc. [77]. They are basically more off-line microprocessor-based switching pulse generator and may be considered as modification to unified approach to the real-time implementation [70] and harmonic minimization [80] for various types of SPWM techniques only.

Technical trends are to implement sophisticated control algorithm of the system in the software, there by reducing peripheral hardware chips to provide higher performance. To alleviate the problems related to frequency limited poor switching angle resolution in the discussed off-line schemes, an on-line microcomputer based Delta-PWM technique will be devised in this thesis work which uses only an INTEL 80386 DX-2 microcomputer to solve a simple algebraic equation [1], [2] to find switching points of sine delta modulated wave and output the generated pulses through parallel port of microcomputer.

1.3 THESIS OBJECTIVES AND OUTLINES

The objective of this thesis work is to generate on-line microcomputer controlled delta modulated switching waveforms for a single and a three-phase static inverter and to implement the generated switching pulses to operate MOS-inverters. The on-line implementation is based on recently proposed algebraic equations to calculate switching points. The implementation is being carried out on an Intel 80386 DX-2 based microcomputer using high level turbo C++ language. The system uses keyboard as input, processor for calculation and pin-2 through pin-7 of the parallel ports to output switching pulses. No other external circuit is necessary as required in other microcomputer controlled pulse width modulated converters thus far reported in literature. This implementation would allow the operator to vary any of the modulation parameters i.e. window width, frequency of the modulating wave, magnitude and slope of the carrier wave by keystrokes at any point of operation without interruption of service.

This thesis work designs the system software to generate DPWM waveforms at the parallel port of the computer and then practically implements this generated switching signals to a three-phase inverter with appropriate interfacing and isolation circuits for the safe operation of both the computer and the inverter system. The total system is to be checked for on-line variation of the modulation parameters in the laboratory. And finally, the work is to analyze the input switching waveforms and the output line to neutral voltage waveforms with three-phase Y connected resistive load with the calculated switching instants.

Chapter-two outlines the switching and computer requirements for this implementation of on-line strategy microcomputer control of inverter. Chapter-three gives a detailed explanation on the methodology, software design, and program organization for the proposed on-line control system for both single-phase and three-phase inverter systems. Chapter-four explores the implementation of the practical MOS-inverter circuit with appropriate isolation circuits. Chapter four also explains the design strategy of each stage for proper interfacing and isolation of the computer and the static inverter system. The last chapter is the concluding chapter which discusses the results and limitations of the designed system with recommendations for future works.

CHAPTER-TWO

REQUIREMENTS OF ON-LINE MICROCOMPUTER CONTROLLED DM INVERTER OPERATION

2.1 INTRODUCTION

This chapter explains the requirements of on-line microcomputer control of Delta modulated inverters for both the single and the three-phase operations. The advantages of the DPWM technique are explained in this chapter for which the technique is selected to generate switching pulses. This chapter also explains the superiority of the proposed on-line strategy to the traditional implementation techniques. Review of the available facilities of a standard microcomputer, the switching and computer requirements for the implementation of the developed on-line system is also provided to give an outline of memory size, storage capacity, and operational speed to support the software designed as described in chapter-three.

2.2 DELTA PULSE- WIDTH MODULATION (DPWM)

Delta pulse-width modulation is the simple and effective modulation technique, where, the comparator output (error signal) of modulating wave and estimated signal is quantized to have the modulated wave. In this method a triangular wave is made to oscillate within a specified window above and below the reference sine wave.

2.2.1 TYPES OF DPWM TECHNIQUE

There are several types of delta modulators that have been investigated so far to switch various power converters [81]. These are:

1) The linear delta modulator
2) The sigma delta modulator and
3) The rectangular wave delta modulator.

Block diagrams of these modulators are shown in figure-2.1.

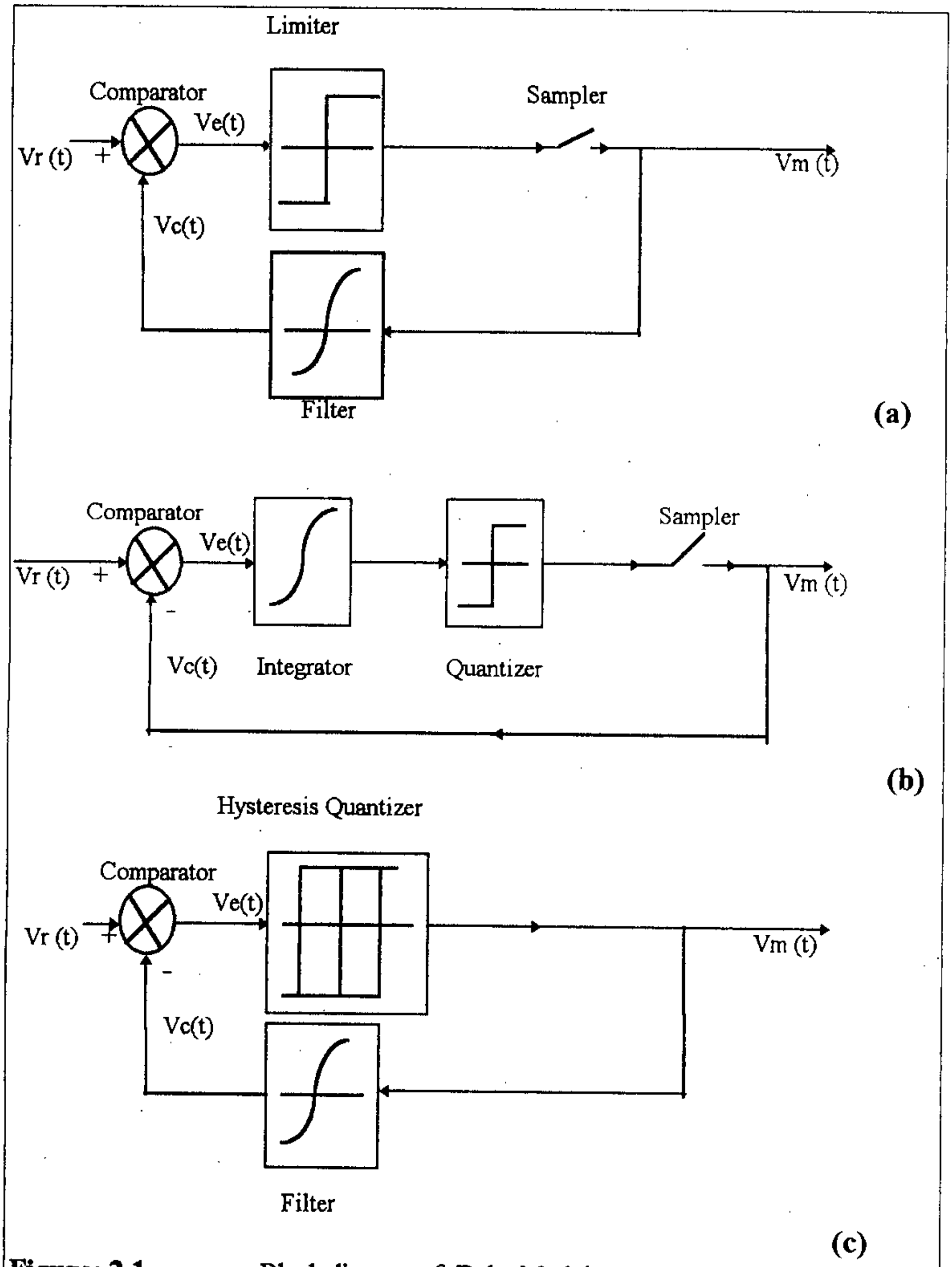


Figure: 2.1 Block diagram of Delta Modulators--

(a) The Linear Delta Modulator.

(b) The Sigma Delta Modulator.

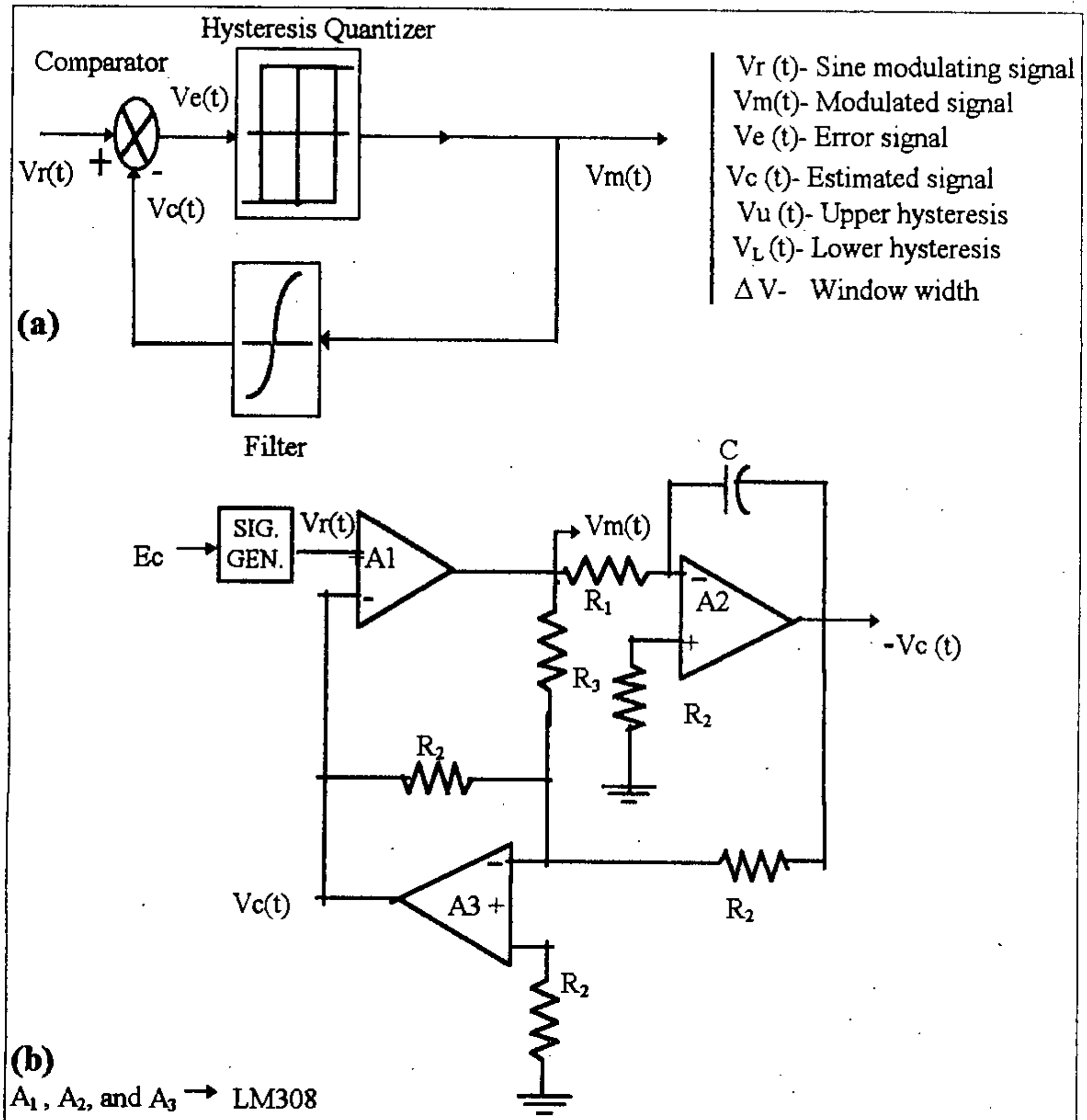
(c) The Rectangular Wave Delta Modulator.

2.2.2 RECTANGULAR WAVE DPWM TECHNIQUE

For ac drives where a constant air gap flux, i.e., volts/Hz is desired, gradual transition from modulated operation into square wave operation is needed. Rectangular wave delta pulse width modulator to switch ac static converters provides inherent implementation of V/f (Volts/ Hz) control in the system. Block diagram of rectangular wave DPWM, the Analog implementation [81], the Waveforms of a rectangular wave DPWM, and Voltage- frequency relation of delta modulated SPWM waveform are shown in figures 2.2 (a) (b), 2.3, and 2.4 respectively.

A rectangular wave delta modulator consists of a comparator that compares the input sine modulating signal with an estimated signal reproduced by the filtering of the modulated wave in the feedback path. The error resulting from the comparison is fed to a hysteresis quantizer. As long as the error signal is positive, a positive quantised pulse is produced and a negative quantised pulse is produced if the error signal is negative. Due to the presence of feedback filter, the output always tries to keep up with the input by intermittent change of pulses from positive to negative or negative to positive respectively.

The analog implementation circuit of figure 2.2 (b) explains the principle of rectangular wave DPWM. The circuit consists of a Schmitt trigger in series with an Integrator, and an Inverting amplifier. The integrator output is fed back to the schmitt trigger through the inverting amplifier. The sinusoidal reference signal Vr (t) is fed at the input. The output signal Vm (t) is the desired PWM wave. With Vr (t) of positive polarity, Vm (t) will also be positive until the feedback voltage Vc (t) generated by the integrator and the inverter exceeds Vr(t) by the preset hysteresis - band $Vm(t)\, {}^{R_2}\!/_{R_3}$. Then Vm (t) becomes negative



A_1, A_2, and A_3 → LM308

$+V_S$ and $-V_S$ → Biasing voltage of Op-Amp

$R_1 = 68K$

$R_2 = 10K$

$R_3 = 100K$

$C = 0.068\mu F$

Figure: 2.2 Block diagram and the practical circuits of a rectangular wave delta-modulator-

(a) Block Diagram of Rectangular Wave Delta Modulator.

(b) Analog implementation of Rectangular Wave Delta Modulator.

and the slope of the integrator is reversed, thus forcing Vc (t) to oscillate around the reference wave Vr (t) as shown in figure 2.3.

2.2.3 ADVANTAGES OF DPWM SWITCHING

For a sinusoidal reference signal, the modulated waveform has a fundamental frequency component equal to the reference frequency and the dominant harmonics appear at ripple frequency and its multiples. Therefore, by adjusting the hysteresis limits and feedback filter characteristics, the dominant harmonics can be laterally shifted towards the higher end of the frequency spectrum [82]. This modulator has the inherent tendency to ensure the ramp variation of the fundamental voltage during pulse width modulated mode of operation. DPWM offers several advantages for static inverter operations and these are,

1. Ease of implementation of the modulator circuit with least amount of hardware.
2. Means of true A/D conversion with minimum loss of information as hysteresis control approach to minimize the error signal within certain critical limits.
3. Volts/ Hz control of modulator wave, acquired from slope overload condition of the modulator.
4. Inherent elimination of selective low order harmonics.
5. Allows hysteresis current control without additional complexity of the modulator.

2.3.1 ADVANTAGES OF MICROCOMPUTER IMPLEMENTATION OF SWITCHING PULSES

Microcomputer controlled generation of switching pulses for static power

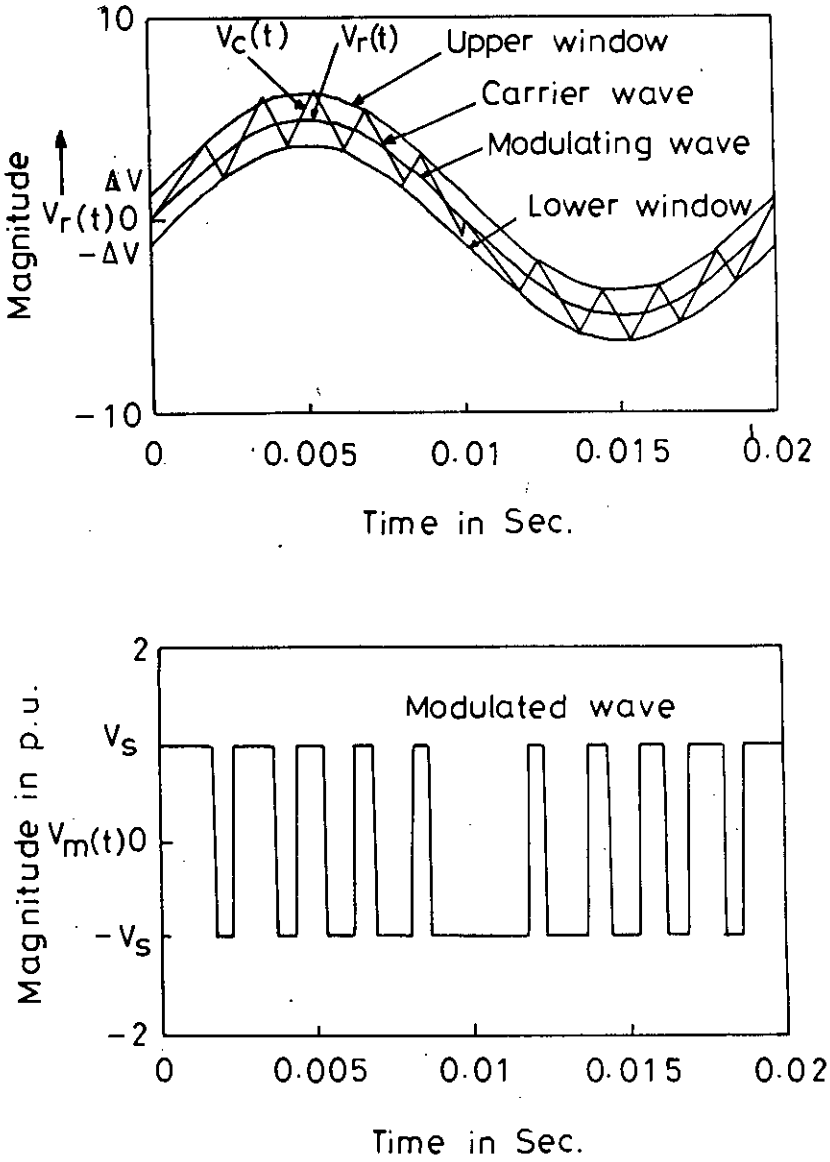


Fig. 2.3 Waveforms of a delta modulator with sine modulating wave.

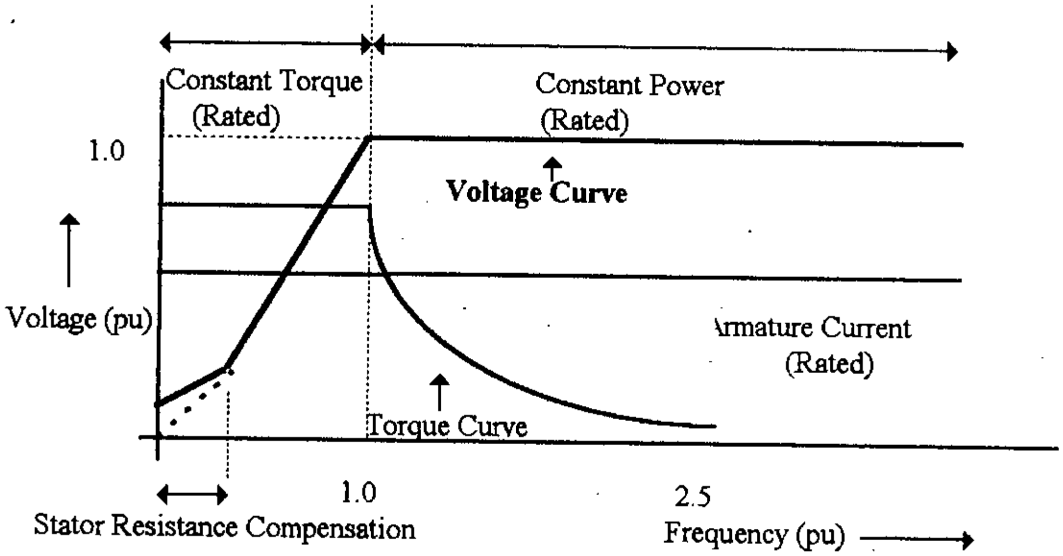


Figure: 2.4

Voltage-Frequency Relation of Delta Modulated SPWM Waveform.

converters embracing many diverse disciplines of electric machines is the best available helping hand in today's Power Electronics system. Microcomputer control have been used in the areas of gate firing, control of phase-controlled converters, non linearity compensation, digital filtering, programmable delay, sequencing of control modes, programmable set point commands, system signal monitoring, warning, and data acquisition. These features successfully alleviates the modern complexities of power electronic system to make it compatible with automated industrial environments.

Due to large voltage and current transients in power electronic circuits, analog implementation of PWM switching circuit suffers from the problems associated with thermal drift, component tolerance, dc offset and imperfections in the amplitude and phase of three-phase analog sine-wave generation which produces harmonic effects in the inverter systems. Regular switching action of analog PWM circuit also generates conducted and radiated electromagnetic interference (EMI), and may also generate acoustic disturbance [64].

Microcomputer, or generously speaking digital control system strategy is very cost-effective than analog control hardware due to the fact that microcomputer control performs many a functions with custom or semicustom VLSI controller chips in a software and dedicated hardware integrations having reduced power consumption.

Superiority of microcomputer control system from the view point of improved reliability, stability, accurate and fast calculation is easily perceivable. The most remarkable superiority of this strategy is its universal hardware system controlled by software modules which is flexible to add, alter, or to upgrade as the physical system demands. As a result, complex computations, sophisticated control functions, decision making, and easy monitoring can be incorporated in a microcomputer control strategy with enhanced execution speed.

2.3.2 NEED FOR THE PROPOSED ON-LINE MICROCOMPUTER CONTROLLED SWITCHING PULSE GENERATION

To establish the advantages of microcomputer controlled switching pulse generation of static converters stated in the preceding article, implementations reported in literature so far used look-up table based microprocessor strategy to store PWM waveforms in EPROM and output the desired waveforms via peripheral digital IC's. These techniques are based on the off-line calculation of switching points. Look-up tables of PWM switching waveforms stored in EPROM cannot provide smooth control over wide range of variation of the control parameters even with large memory size. This problem arises due to the fact that the computation of PWM switching points uses transcendental equation which requires long time for calculations. Consequently, these techniques use switching point calculations first for different modulation frequencies and various PWM control parameters, and then store them in peripheral EPROM. The required pulses are produced there after through DMA transfer technique with software commands. These techniques fail to satisfy the steady state and transient performance specifications of any static drive systems which may demand PWM parameter variations within a fraction of a millisecond. Besides, these techniques need different peripheral hardware arrangement for drives of different types and ratings to perform even off-line controls. It is clear that the above problems of EPROM-based microcomputer control of PWM switching pulses have not yet been eliminated fully due to lack of easy solution of PWM switching times intervals to successfully replace the transcendental equation based solution. Indubiously, an algebraic equation to

PWM switching point calculation requiring only few nano-seconds to be converged, will open the horizon of on-line calculation of intervals of PWM switching pulses. For the lack of this type of algebraic equation, research works on microcomputer/microprocessor-based PWM switching pulse generation for static converters (specially for inverters) are so far carried out for unified approach [70], harmonic and EMI minimization [80], [64], display facilities and other improvements of only SPWM techniques with off-line control, which is outlined in the literature review. Naturally such obstacles hindered works on microcomputer based DPWM switching pulse generation even with off-line calculation depending on EPROM based look-up table.

Recently a successful, accurate, and simplified algebraic equation for DPWM switching intervals has been developed [1], [2]. And this thesis work uses that algebraic equation to calculate DPWM switching pulse intervals to employ an on-line microcomputer control of delta modulated inverters. On-line microcomputer control and computation devised in the proposed thesis implementation will represent a system where computation of the DPWM switching intervals with newly assigned parameters will not stop the generation of switching pulses of the former pattern at the pins of parallel ports. As a result, unlike the traditional off-line microprocessor control, this set up will satisfy steady state and transient performance specifications of any sort of drive system providing smooth control over wide range of variation of the control parameters.

Unlike dedicated hardware control, a microcomputer performs control in serial fashion, i.e. multitasking operations are executed in a time sharing or time multiplexed manner. In this case the execution of fast control loop as well as sharp response to parameter changes in the output pulses may suffer serious problem of slow computations. But the implementation of the developed algebraic equation [1] for switching pulse intervals will ensure sharp response of the control system.

Schematic hardware block diagram of two traditional microcomputer controlled pulse width modulators are shown in figures 2.5 and 2.6. Figure 2.5 represents the system [77] which synthesizes precision three-phase PWM waves by hybridizing the computation intensive and look-up table methods at lower and higher frequency regions respectively. The microcomputer used here is based on an Intel 8086 CPU, and a single board computer SDK-86 with appropriate peripheral hardware. RAM of 128 bytes supports the assembly language program, including the look-up tables which are stored in 2.5 kbytes of EPROM memory. The 8253 chips are used as three-phase pulse width and T_{C1} counters which generate the INTERRUPT signals for the 8259 chips. The synchronizing flip-flops interfaced with PWM output signals are used to prevent any timing error. The high-speed timing counters use Schottky TTL chips and are triggered by a 40 MHz oscillator as shown in figure 2.5.

Figure 2.6 represents the hardware schematic of a unified system of a real-time implementation [70] of microcomputer-based PWM waveforms. The hardware of this strategy has been built around the 16-bit 8086 processor and its peripherals. The system contains two RAM; RAM_1 is the system RAM used for developing the overall software, whereas, the voltage levels of three phases (R, Y, and B) at the switching angles are stored in the consecutive memory locations of the RAM_2 in a 3-bit binary form. Depending on the address input of RAM_2 the particular RYB pattern is outputted from RAM_2 and gets latched. An 8-bit synchronous counter is used to generate address for RAM_2 . The counter output (Q_0-Q_7) changes its state at the rising edge of its clock pulse and to generate the clock pulses at every switching instant the timer 8253 and interrupt controller 8259 are used as shown in the block diagram of figure 2.6.

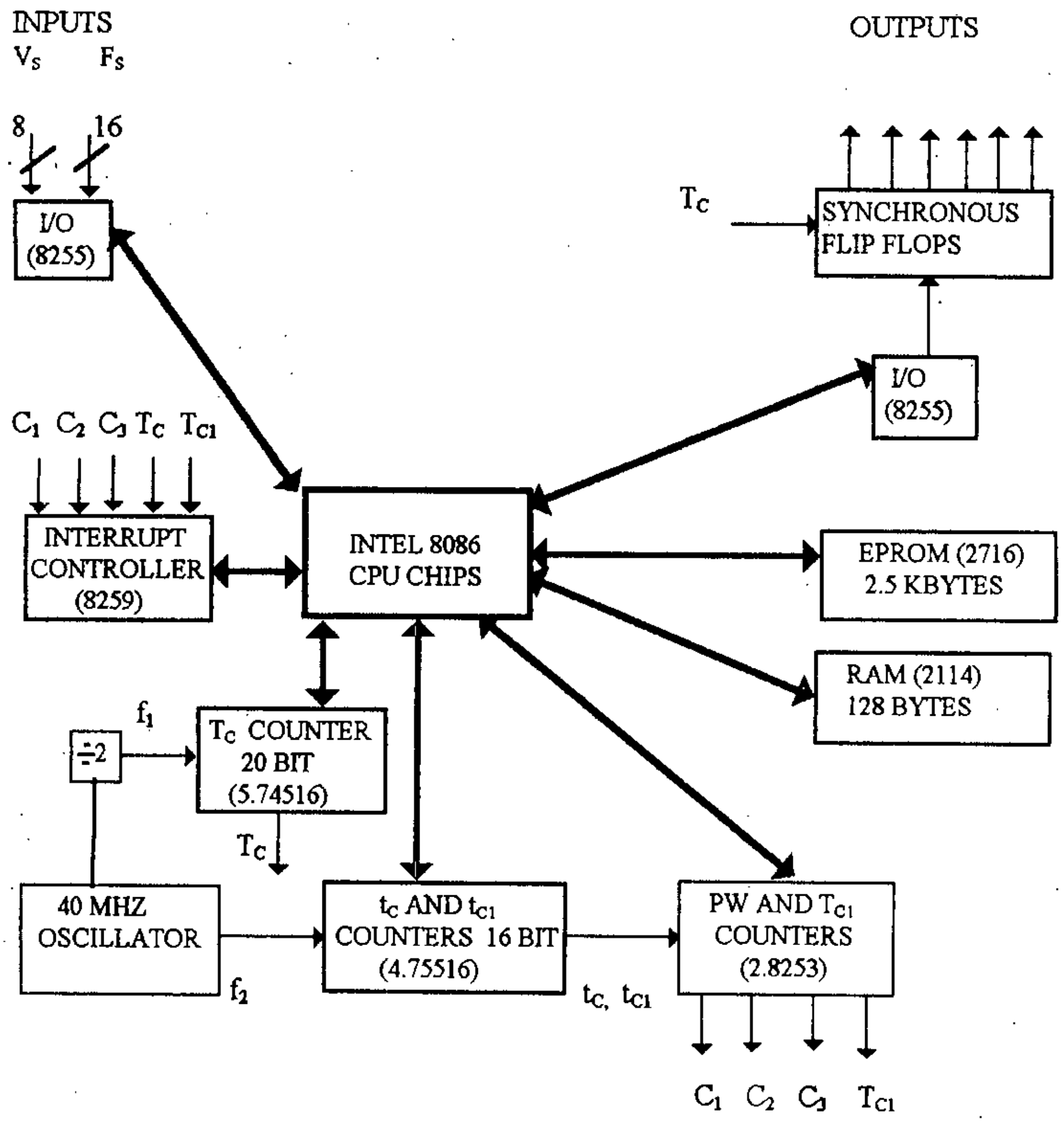

INPUTS
V_S
F_S
8
16
I/O (8255)
OUTPUTS
T_C
SYNCHRONOUS FLIP FLOPS
I/O (8255)
C_1 C_2 C_3 T_C T_{C1}
INTERRUPT CONTROLLER (8259)
INTEL 8086 CPU CHIPS
EPROM (2716) 2.5 KBYTES
RAM (2114) 128 BYTES
f_1
÷2
T_C COUNTER 20 BIT (5.74516)
T_C
40 MHZ OSCILLATOR
f_2
t_C AND t_{C1} COUNTERS 16 BIT (4.75516)
t_C, t_{C1}
PW AND T_{C1} COUNTERS (2.8253)
C_1 C_2 C_3 T_{C1}

Figure: 2. 5

Schematic block diagram of hybrid microcomputer hardware to generate PWM waveforms.

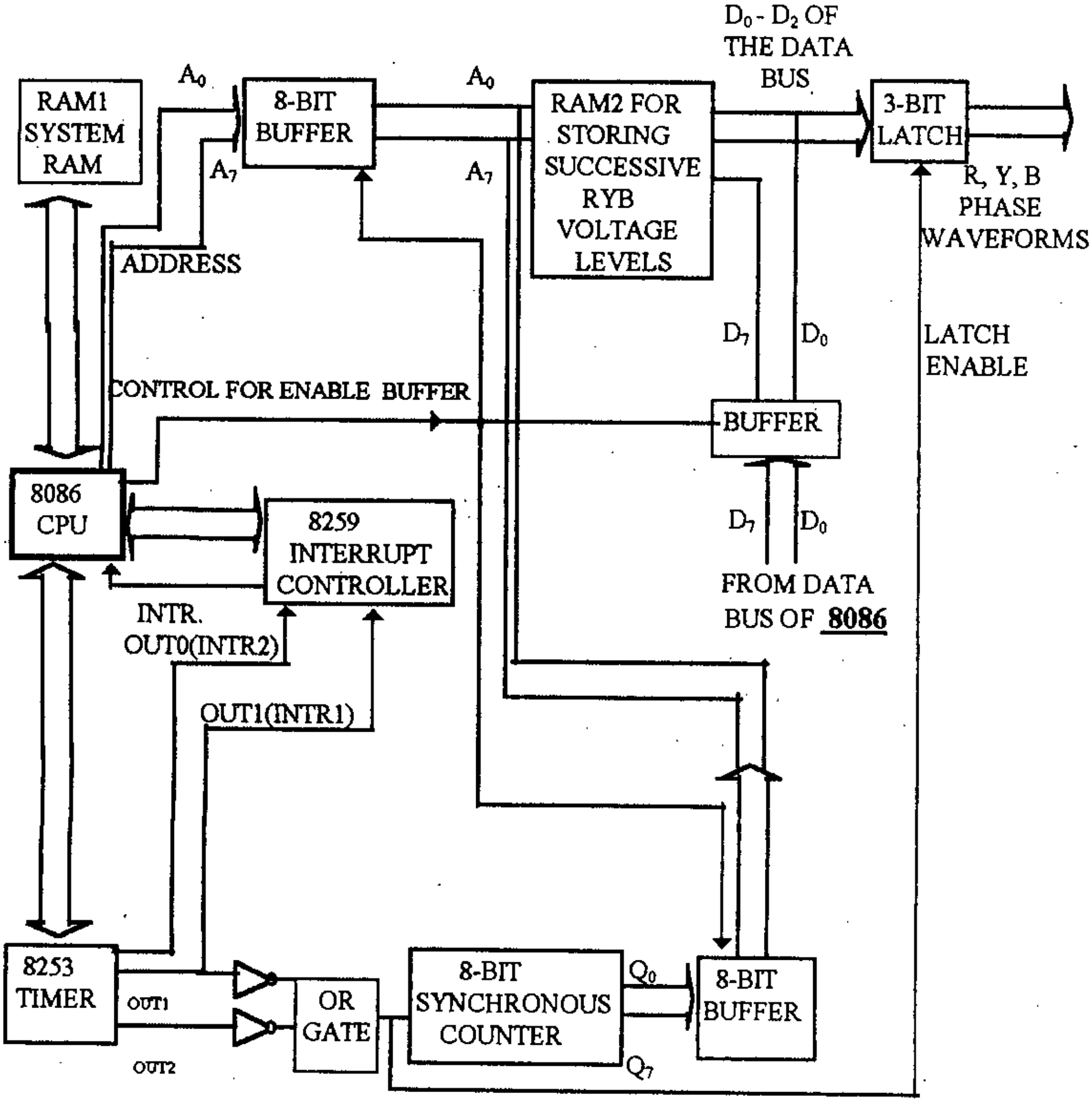


Figure 2. 6

Schematic block diagram of unified real time microcomputer hardware to generate PWM waveforms.

Unlike the traditional off-line microprocessor control, the proposed hardware implementation of the on- line strategy will use only an INTEL 80386 DX-2 computer with general facilities available to support the software used without any special control cards and facilities inside or outside the CPU except the peripheral digital chips. This is possible due to the fact that, for this implementation there is no need to store the PWM intervals in the EPROM or passing the pulses of required pattern through DMA transfer. This will offer an unified, and unique setup supporting drives of any type or rating with simplicity of maintenance and operation. User is to change the control parameters from keyboard or to alter the related software to work with new environment. The details of this implementation with necessary diagram is explained in the following articles.

2.4 ALGEBRAIC EQUATION FOR SWITCHING POINTS OF DPWM [1], [2]

One of the transcendental equation used to calculate PWM switching wave generation using DPWM technique is [83] (Figure 2.7),

$$t_i = t_{i-1} + \frac{2\,dv + (-1)^i V_m(\sin\omega_m t_i - \sin\omega_m t_{i-1})}{S} \quad \text{.........................(2.1)}$$

Where,

S = A, slope of rising edge, i = even

S = B, slope of falling edge, i = odd

dv = window width

$V_m \sin w_m t_i$ = input sine wave at the instant t_i

$V_m \sin w_m t_{i-1}$ = input sine wave at the instant t_{i-1}

V_m = Maximum voltage of the modulating sine wave

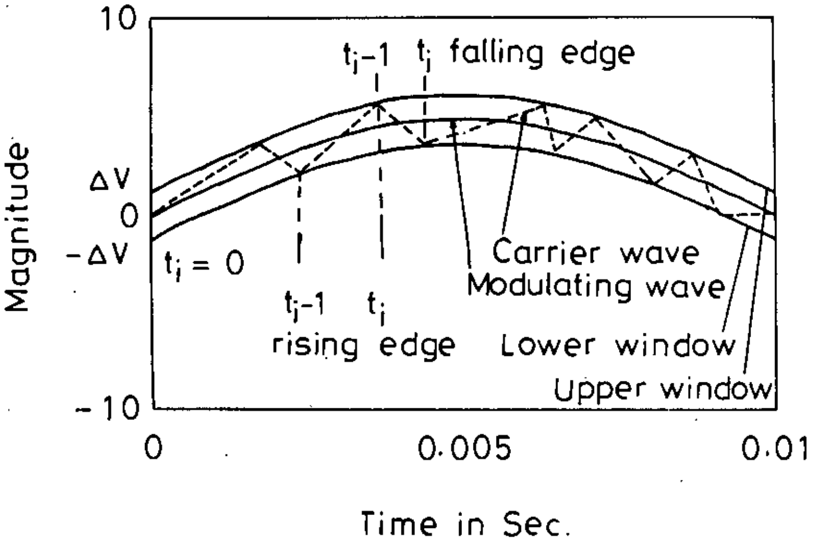


Fig. 2·7 Expanded view of modulation process for sine wave.

ω_m = Angular frequency of the sine wave

A scheme is developed in reference [1] which is similar to the above formulation but finally offers an algebraic form of equation rather than a transcendental form as the above .

According to the developed concept the equation for any rising edge of carrier wave is,

$$2dv + V_m(\sin\omega_m t_i - \sin\omega_m t_{i-1}) = A(t_i - t_{i-1}) \quad \text{..............................(2.2)}$$

And with fair degree of approximation within the operating limit of converters for any drives, equation (2.2) can be written simply as,

$$2dv + (t_i - t_{i-1}) \times \text{slope of } V_m \sin\omega_m t_i = A(t_i - t_{i-1}) \quad \text{......................(2.3)}$$

Since slope of $V_m \sin \omega_m t_i$ at $t = t_{i-1}$ is $\omega_m V_m \cos \omega_m t_{i-1}$ equation (2.3) can be written as,

$$2dv + (t_i - t_{i-1})\omega_m V_m \cos\omega_m t_{i-1} = A(t_i - t_{i-1}) \quad \text{..............................(2.4)}$$

$$\text{or,} t_i = t_{i-1} + \frac{2dv}{A - \omega_m V_m \cos\omega_m t_{i-1}} \quad \text{..(2.5)}$$

Similarly, for any arbitrary falling edge of the carrier wave the equation can be written as,

$$2dv - (t_i - t_{i-1})\omega_m V_m \cos\omega_m t_{i-1} = B(t_i - t_{i-1}) \quad \text{...........................(2.6)}$$

$$\text{or, } t_i = t_{i-1} + \frac{2dv}{B + \omega_m V_m \cos \omega_m t_{i-1}} \quad \text{...(2.7)}$$

The first switching point is $t_1 = 0.0$ second. Considering equation (2.5) and (2.7), subsequent switching points can be written in the form of following general expression,

$$t_i = t_{i-1} + \frac{2dv}{S + (-1)^{i-1} V_m \omega_m \cos \omega_m t_{i-1}} \quad \text{...(2.8)}$$

Where,

S = A, slope of rising edge, with i = even.

S = B, slope of falling edge, with i = odd.

The simple algebraic equation expressed by equation (2.8) is the required equation to be used for on- line microcomputer implementation for RDPWM switching pulses for inverters. It will take only a fraction of microseconds for the above algebraic equation to be solved.

2.5 A PERSONAL COMPUTER SYSTEM

Any computer system consists of a combination of three types of subsystems : Central Processing Unit's (CPU), Direct -Memory subsystems having RAM and / or ROM chips, and Input/Output (I/O) interfaces for peripheral control. Between them, the CPU subsystems supports all the classical arithmetic, logical, and control functions of the combined system. The Direct-Memory subsystems contain the programmed instructions to be executed by the individual CPU's and the recent active data on which the CPU's are operating.

The I/O interfaces performs the critical communication links between the internal computer operations and the external input/output devices.

A typical block diagram of the fundamental subsystem of a microcomputer and internal block diagram of a CPU are shown in figure 2.8 (a) and figure 2.8 (b) respectively [52], [84]. The CPU is connected to the other elements through the address, data, and control buses. The width of the data bus normally determines the bit size of a computer. The address bus specifies the memory location, the control bus determines the operation from or a memory location, and the data bus bears data or instruction from the specified memory location.

The computer interface contains adapters to support speaker, keyboard, display monitor, printer, disk drives etc.[85], [86]. The printer adapter or parallel adapter provides the parallel I/O port with a 25-pin D-shell connector. In this research, an INTEL 80386 DX-2 machine containing a 32-bit 80386 microprocessor is used.

The printer is connected from printer adapter or parallel adapter of input/ output (I/ O) channel. These are shown in the block diagram of System Unit of figure 2.9. The I/O channel supporting various adapter contains bi-directional data bus, address lines, interrupt, control lines for memory and I/O read or write, clock and timing lines, DMA control lines, memory refresh timing control lines, channel-check line, and power and ground for the adapters. Four voltage levels are provided for I/O cards : +5 volt, -5 volt, +12 volt, and -12 volt dc. The adapters are addressed according to the I/O address map. Of the connectors, the printer or parallel connector is the only parallel I/O port which is addressed as 378-37F in Hexadecimal range. Here, 3 indicates parallel port 1. If we write 2 instead of 3, it will address parallel port 2.

MICROCOMPUTER

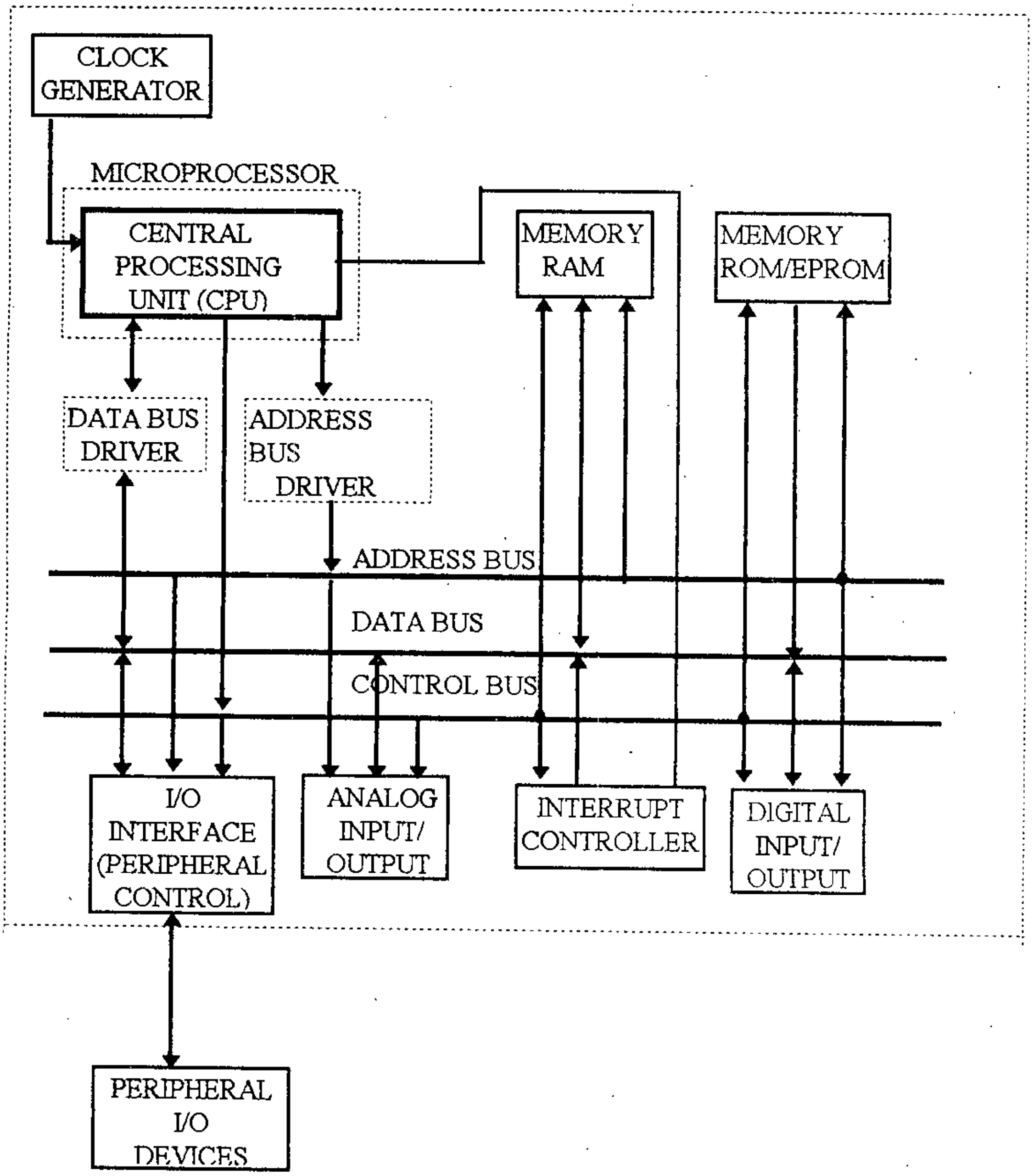


Figure 2. 8(a) : Fundamental subsystems of a microcomputer system.

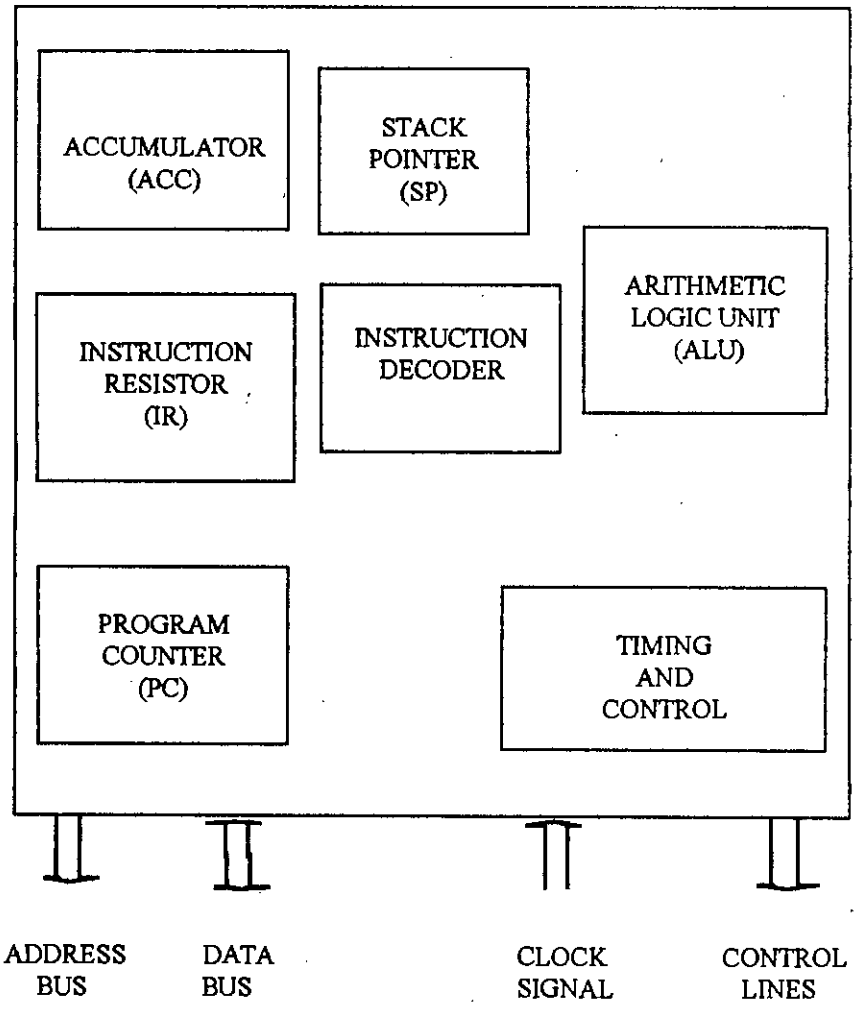


Figure 2. 8(b) : The internal block diagram of a microcomputer central processing unit (CPU).

The simple communication ports, i.e. parallel printer and serial modem ports, are non-critical components in the personal computer machine. Of the two ports, parallel interface port is the easiest interface to understand. Although there is no well- defined standard for parallel and serial ports, the IBM style connector have become the de facto standard. Most IBM machines and clones use a 25- pin D-shell female connector which is marked either "parallel port" or " printer port" [87]. The parallel adapter or printer adapter makes possible the attachment of various devices that accept eight bits of parallel data at standard TTL levels. The block diagram of a printer adapter and connection specifications of a 25-pin D-shell connector is shown in figures 2.10 and 2.11 (a) & (b) respectively [85].

The printer adapter or parallel adapter responds to five I/O instructions : two output and three input. The output instructions transfer data into 2 latches whose outputs are presented on pins of the 25- pin D-shell connector numbered as pin-2 to pin-9 for the corresponding data bit-0 to data bit-7 as in figure 2.11 (b). Two of the three input instructions allow the processor to read back the contents of the two latches. The third allows the processor to read the real time status of a group of pins on the connectors. When the printer adapter is used to attach a printer or any device that accept eight bits of parallel data at standard TTL levels , data or commands are loaded into an 8-bit, latched, output port, and the strobe line is activated, writing data to the printer or passing pulses of required patterns to the interfaced device connected with the adapter. The DPWM switching pulses for a three- phase inverter system will be available at this connector.

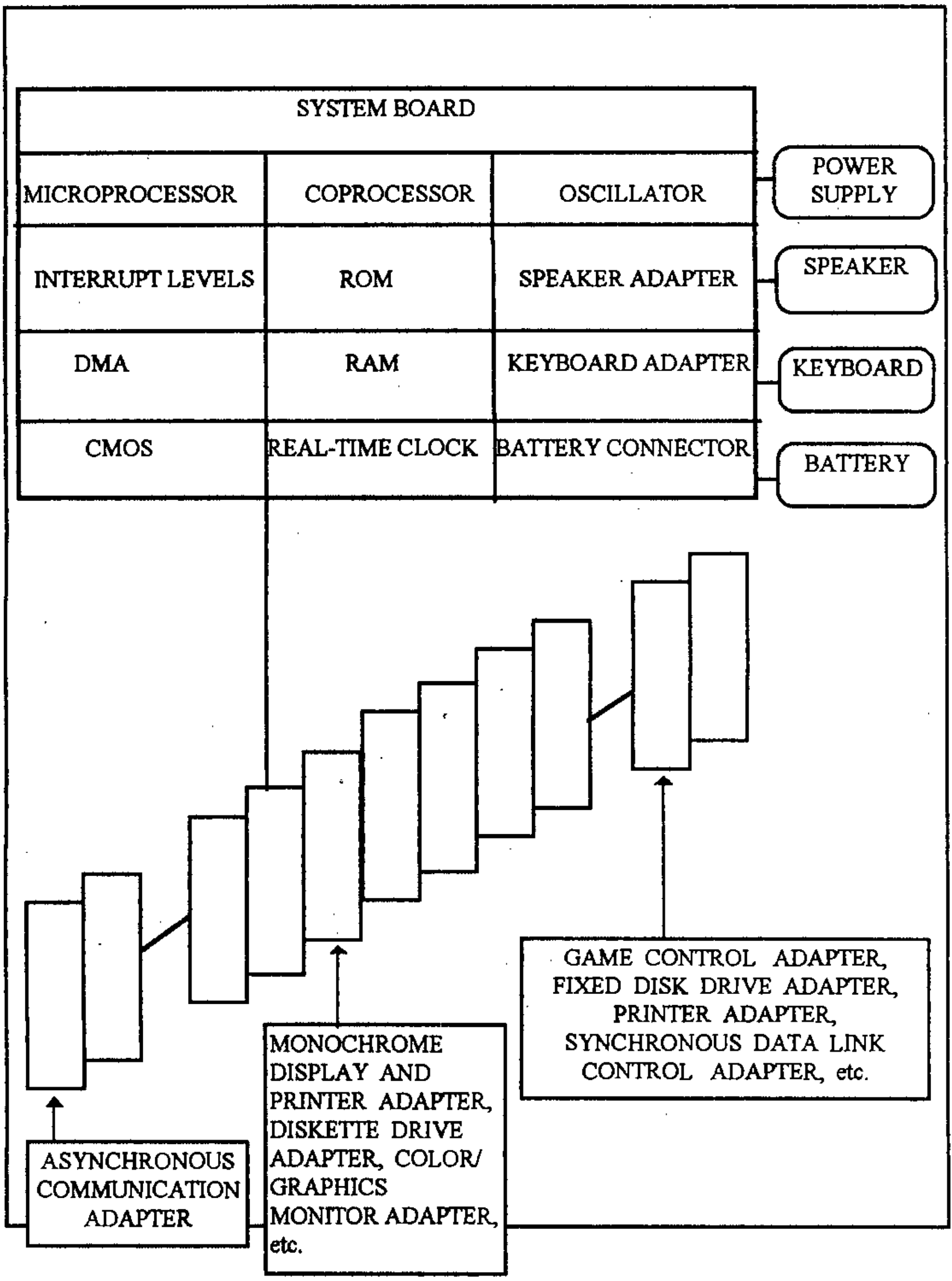

SYSTEM BOARD
MICROPROCESSOR
COPROCESSOR
OSCILLATOR
POWER SUPPLY
INTERRUPT LEVELS
ROM
SPEAKER ADAPTER
SPEAKER
DMA
RAM
KEYBOARD ADAPTER
KEYBOARD
CMOS
REAL-TIME CLOCK
BATTERY CONNECTOR
BATTERY
GAME CONTROL ADAPTER, FIXED DISK DRIVE ADAPTER, PRINTER ADAPTER, SYNCHRONOUS DATA LINK CONTROL ADAPTER, etc.
MONOCHROME DISPLAY AND PRINTER ADAPTER, DISKETTE DRIVE ADAPTER, COLOR/ GRAPHICS MONITOR ADAPTER, etc.
ASYNCHRONOUS COMMUNICATION ADAPTER

Figure: 2.9

Block diagram of the system unit of a microcomputer.

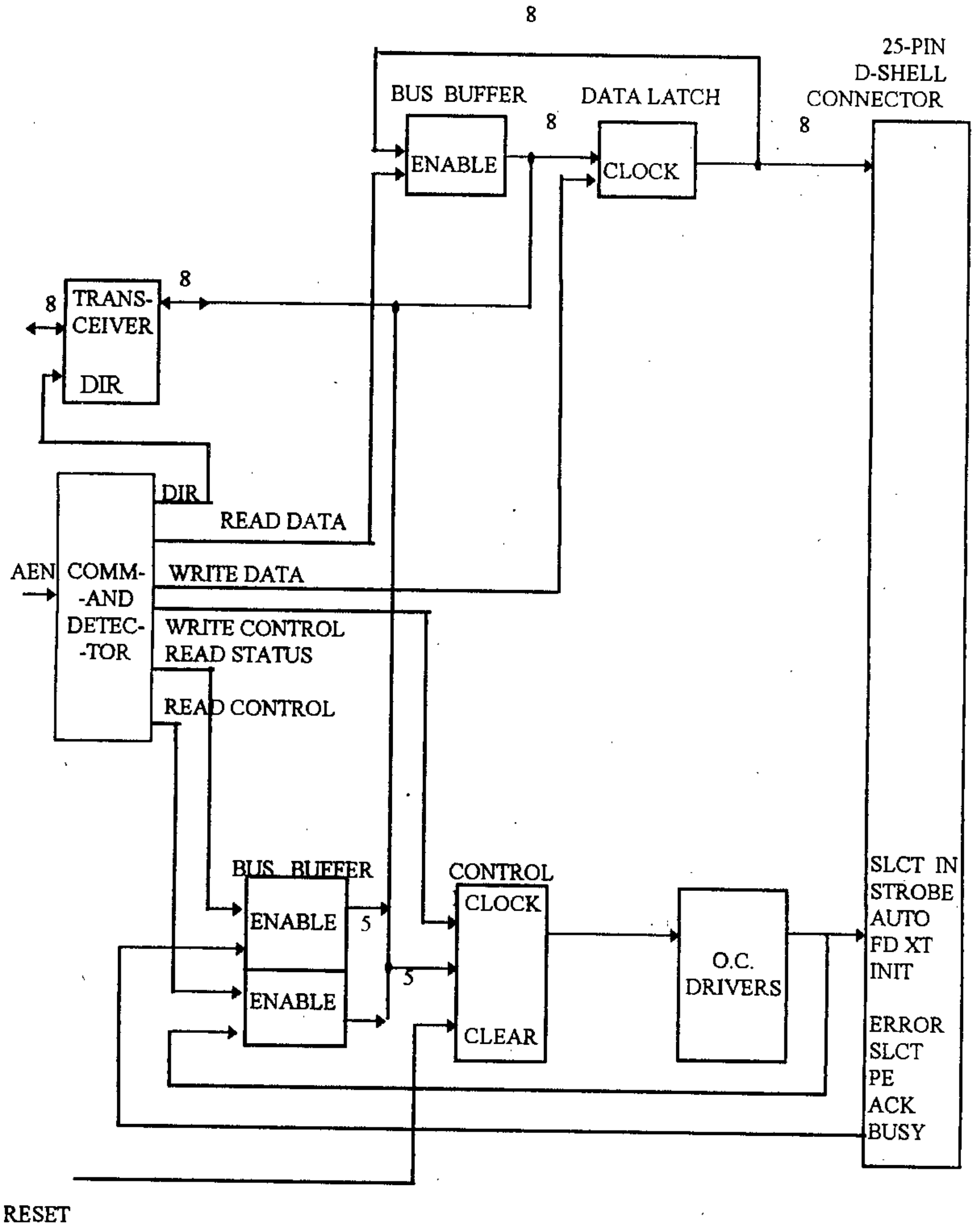


Figure: 2.10

Block Diagram of a Printer Adapter of a Microcomputer.

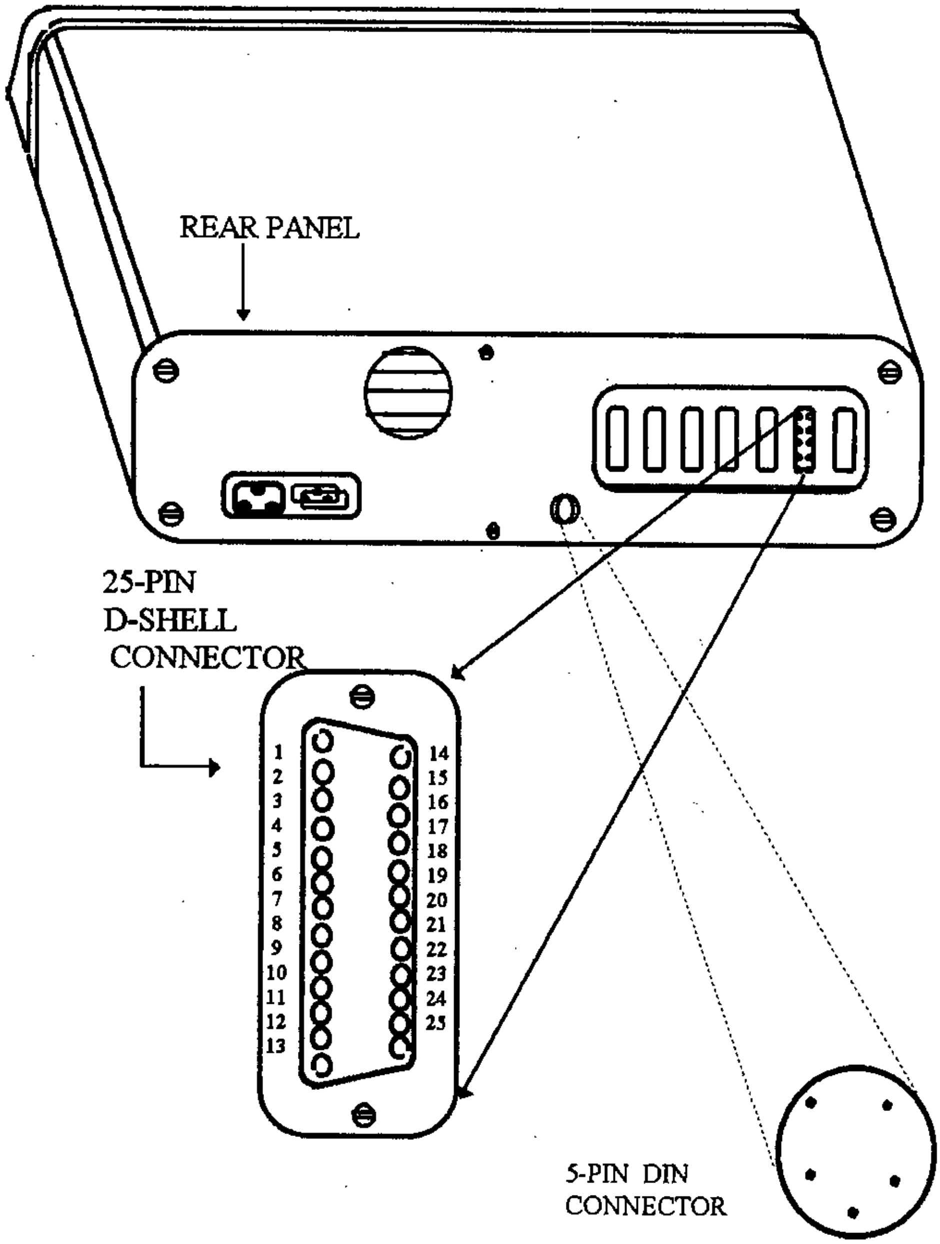
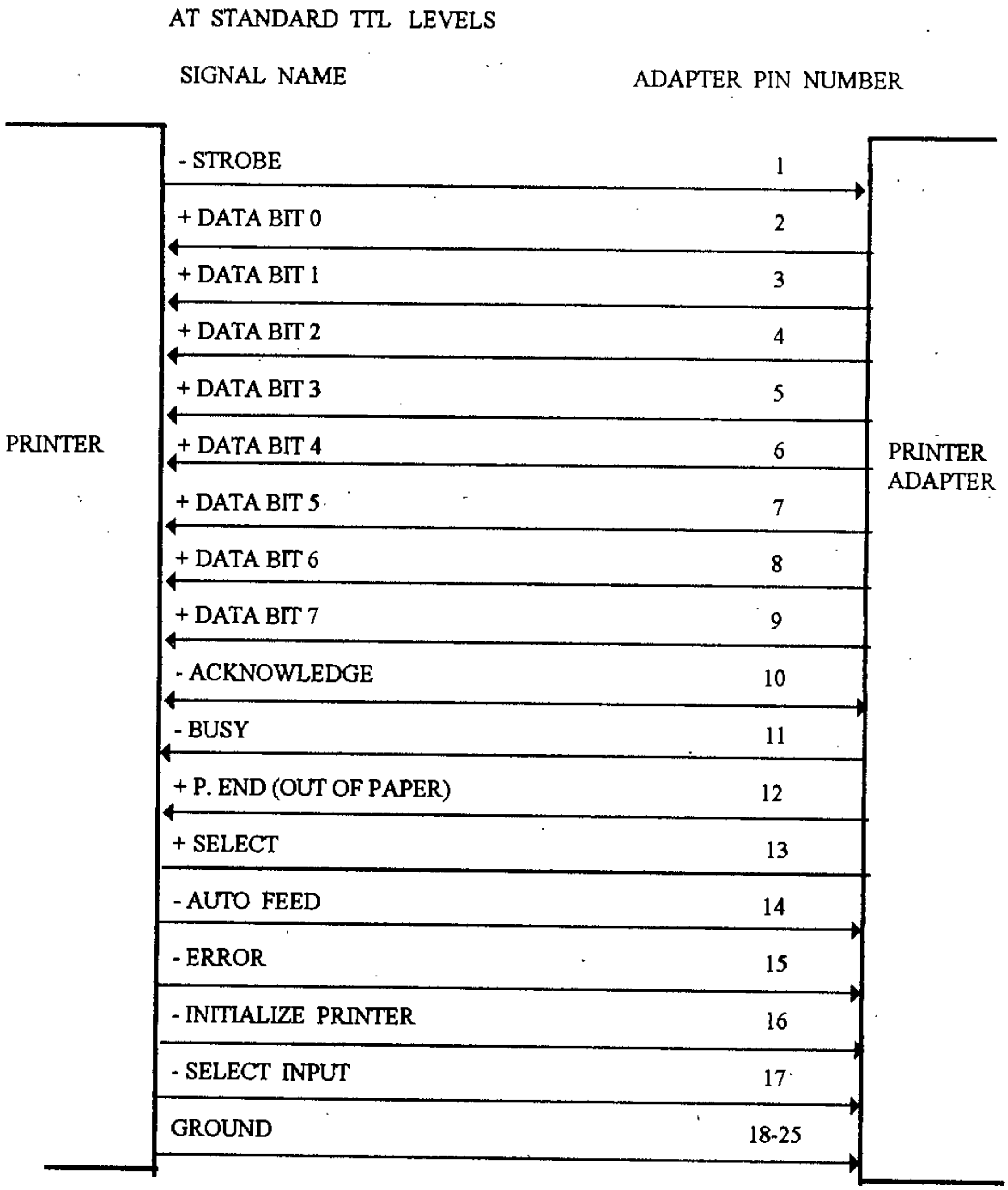


Figure: 2.11(a)
Connector Specification of a Parallel I/O Port .



SIGNAL NAME	ADAPTER PIN NUMBER
- STROBE	1
+ DATA BIT 0	2
+ DATA BIT 1	3
+ DATA BIT 2	4
+ DATA BIT 3	5
+ DATA BIT 4	6
+ DATA BIT 5	7
+ DATA BIT 6	8
+ DATA BIT 7	9
- ACKNOWLEDGE	10
- BUSY	11
+ P. END (OUT OF PAPER)	12
+ SELECT	13
- AUTO FEED	14
- ERROR	15
- INITIALIZE PRINTER	16
- SELECT INPUT	17
GROUND	18-25

NOTE : OUTPUT TO ADDRESS HEX 378 FOR PIN NUMBER - 2 TO PIN NUMBER- 9.

NOTE : ALL OUTPUTS ARE SOFTWARE-GENERATED AND ALL INPUTS ARE REAL-TIME SIGNALS (NOT LATCHED).

Figure: 2.11(b)
Data-pin Specifications of a 25-pin D-shell Parallel I/O Port.

2.6 REQUIREMENTS OF ON-LINE MICROCOMPUTER CONTROLLED DM INVERTER OPERATION

Performance of the output of PWM inverters which are dc to ac converters depends on the switching techniques of the switching devices like power-transistors, FET, MOSFET, IGBT, SCR etc. Requirements of on-line microcomputer controlled DPWM switching implementation for inverters is discussed in this section. The implementation requirements depends on the type of output voltage, type of control strategy or type of the drive to be supplied by the system.

2.6.1 SWITCHING REQUIREMENTS FOR SINGLE / THREE PHASE INVERTER

The DPWM switching pulses for the switching devices differs in patterns due to the variation of control parameters required to support a desired operating environment. Figure 2.12 (a) shows the general configuration of the single-phase bridge inverters having MOSFET as switching devices. If we need lower fundamental output voltage having the waveshape of figure 2.12 (b), we have to generate switching pulses of the pattern shown in figure 2.12 (c). On the contrary, if we need higher fundamental output voltage having the waveshape of figure 2.12 (d), we have to generate switching pulses of the pattern shown in figure 2.12 (e). For single-phase inverter, in this thesis work we have devised software implementation to have both the switching waveforms of figures 2.12 (c) and 2.12 (e).

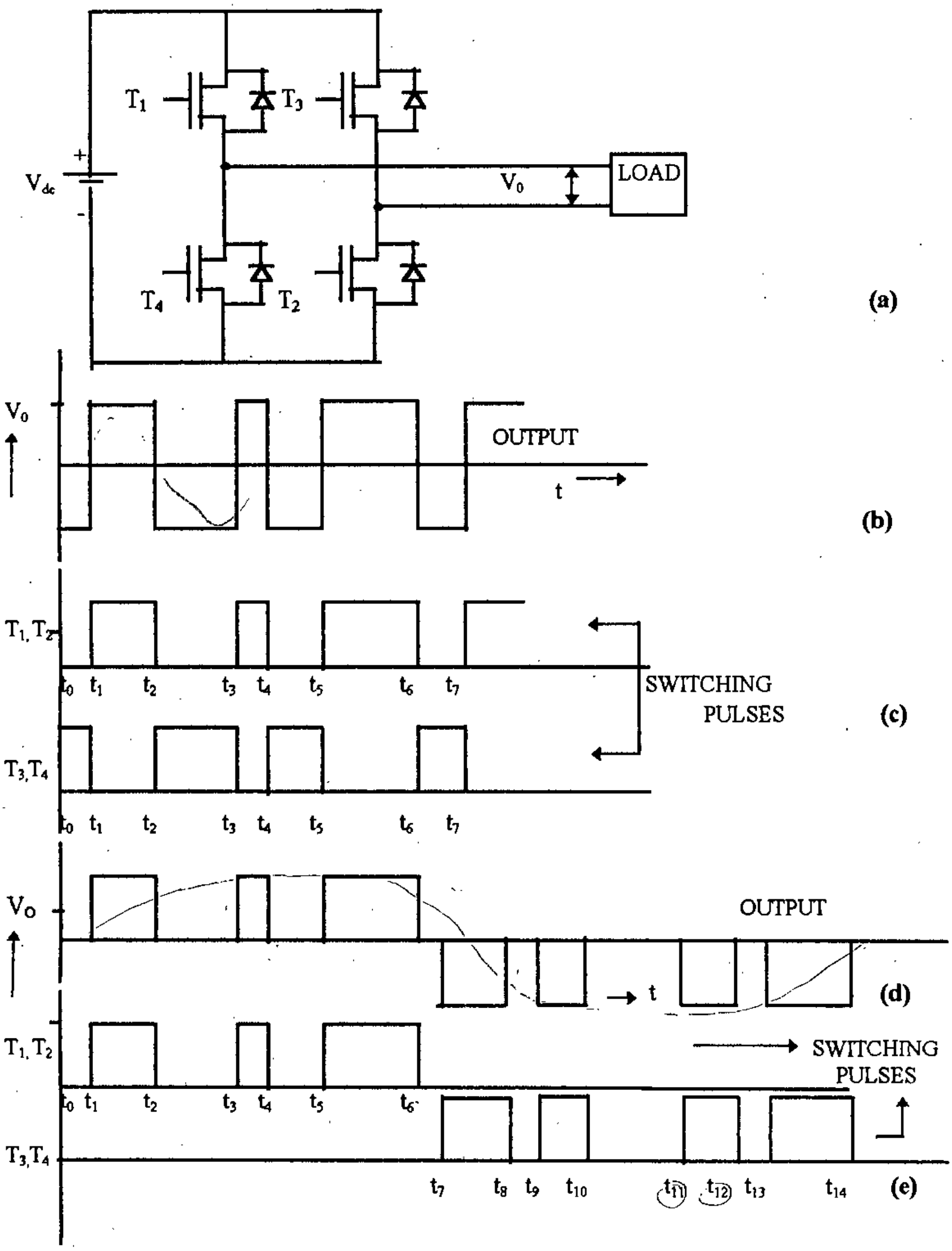


Figure: 2.12 Switching signals required for a single-phase inverter--

(a) General configuration of single-phase bridge inverter.
(b) Timing diagram of output voltage where voltage availability is lower.
(c) Timing diagram of switching pulses for output voltage of fig. 2.12(b).
(d) Timing diagram of output voltage where voltage availability is higher.
(e) Timing diagram of switching pulses for output voltage of fig. 2.12(d).

The general configuration of the bridge inverter used in converting three-phase dc voltage to ac level is shown in figure2.13 (a). The switching pulse patterns required for an output voltage for higher fundamental voltage availability are outlined in figure 2.13 (b). The six pulses for six of the switching elements are in 120° phase shift with one another. On the other hand, implementation of three-phase voltage output of having lower fundamental voltage availability like that for single-phase voltage output of figure 2.12 (b) is a difficult to achieve by inverter switching. As a result, in this thesis work for three-phase DPWM inverter we have implemented switching pulses of the pattern outlined in figure 2.13 (b). In this regard, it can be mentioned that three-phase bridge converters have six pulses at their output which leads to lower ripple contents and consequently small filter requirements. So, three-phase inverter are more suitable for industrial applications due to the availability of higher output voltage level having the ability to accommodate loads requiring high power level.

If the switching elements are SCRs other than power transistors, FETs or MOSFETs, we need commutation signals for the system as in figure 2.14. The commutation signals can be produced by two methods : Computer generation process and external one-shot multivibrator process. In this thesis we have used MOSFET as switching device and consequently the use of SCR as well as generation of commutation signals are omitted.

Gating pulses require variation of the modulation parameters of *f, dv, s, and vm* to control output frequency, ripple frequency, harmonics, and available voltage at the load [84] respectively. The timing instants $t_0, t_1, t_2, ...t_n$ of gating pulses are calculated from the equation number (2.8) stated in the previous section of

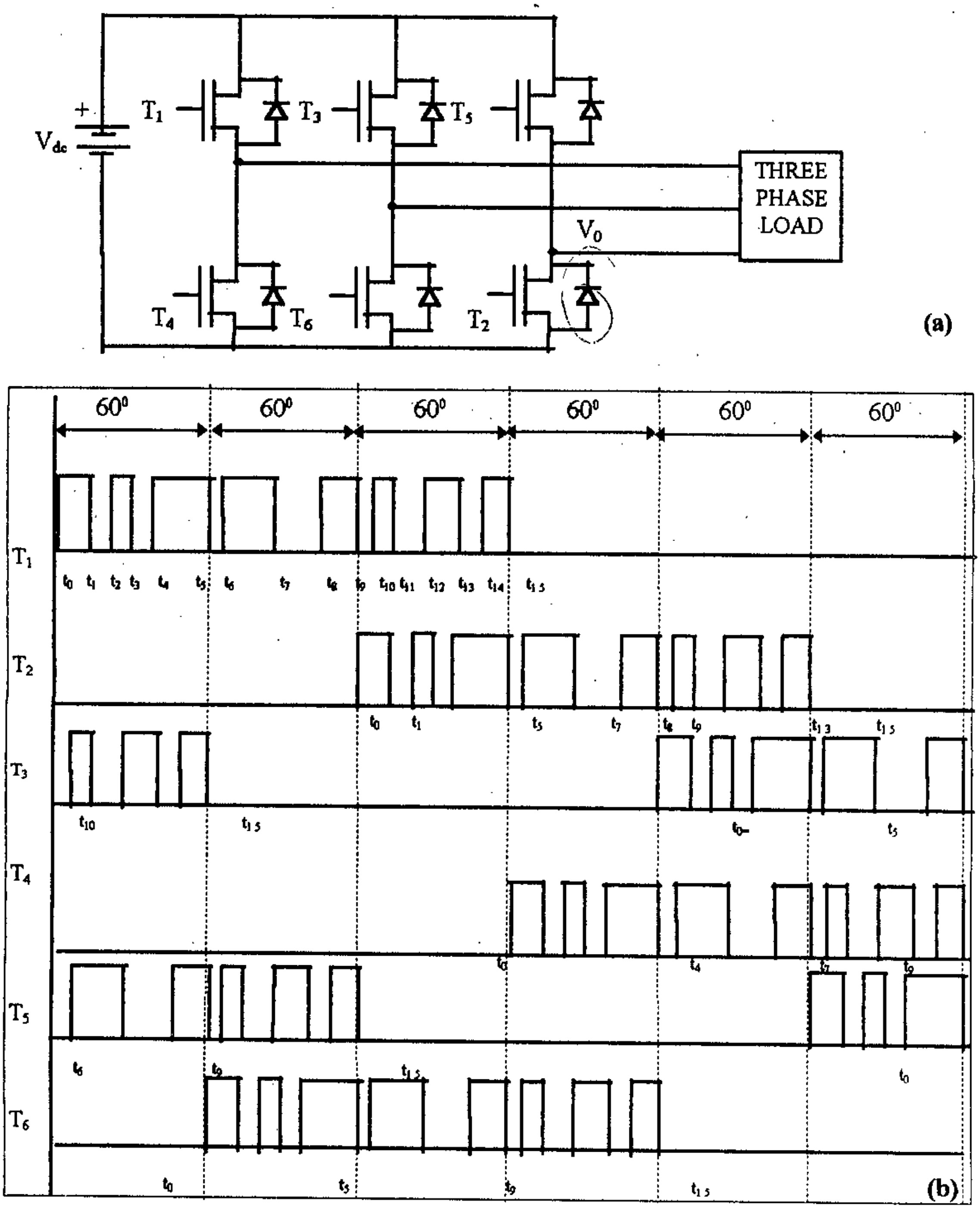


Figure: 2.13 Switching signals required for a three-phase PWM inverter---
(a) General configuration of three-phase bridge inverter.
(b) Timing diagram of six switching pulse-pattern to have a output voltage where voltage availability is higher.

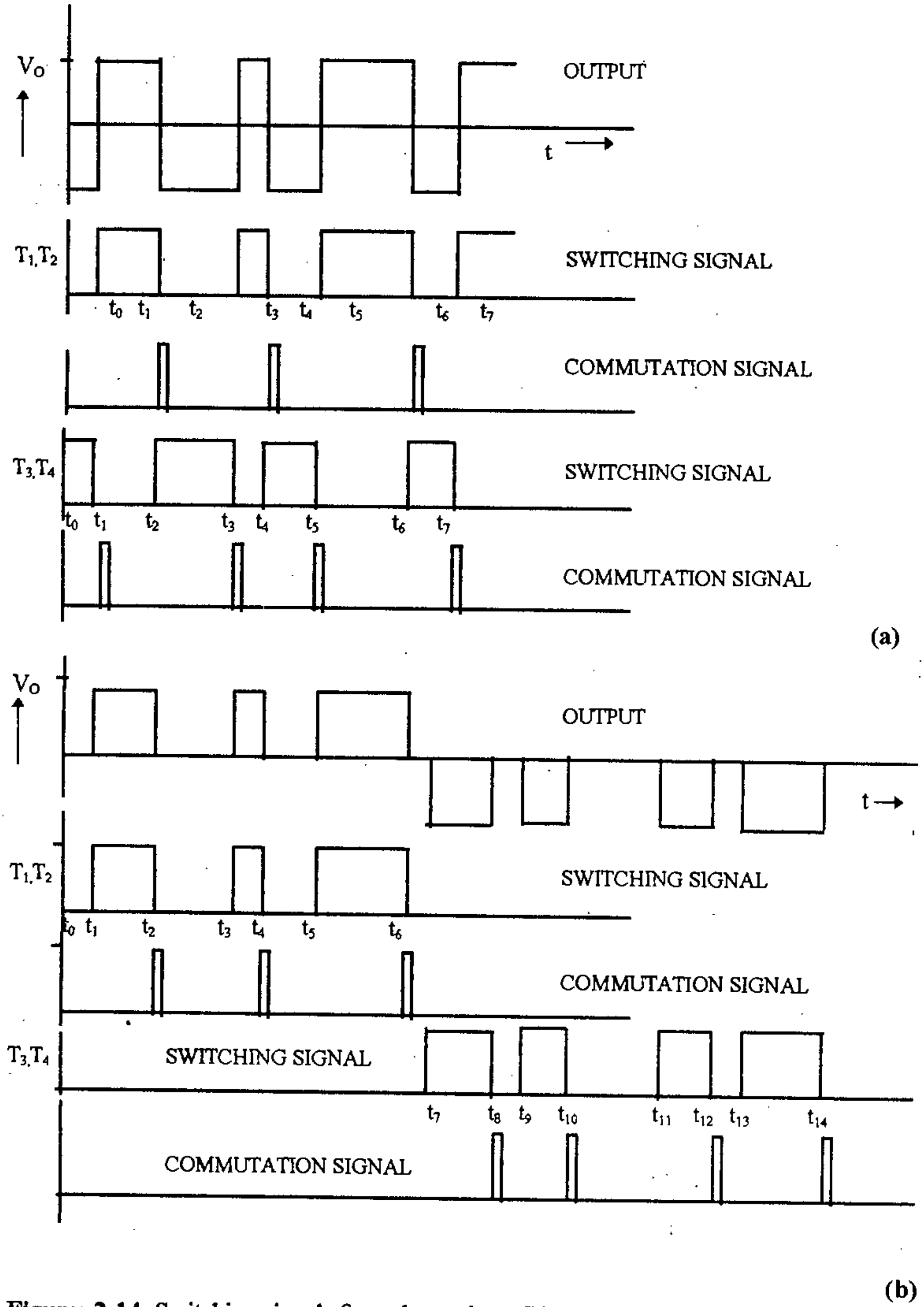


Figure: 2.14 Switching signals for a three-phase SCR PWM inverter--
(a) Timing diagram of output voltage, switching pulses, and the corresponding commutation signal where voltage availability is lower.
(b) Timing diagram of output voltage, switching pulses, and the corresponding commutation signal where voltage availability is higher.

(2.4) as a function of *f, dv, s, and vm.* For on-line control of an inverter as well as a drive connected from the inverter, the implementation have to confirm that computation of timing instants with new parameters will not stop the generation of the previous switching signal patterns. An inverter-fed drive will allow only about 20 μs of delay between two types of switching pulse patterns. Consequently, on-line microcomputer controlled implementation requires calculation and other steps to be done immediately to maintain the above timing condition.

2.6.2 COMPUTER REQUIREMENTS

Generation of on-line microcomputer controlled DPWM switching pulse will be implemented for an inverter using a program in high level language (C++). Computer requirements with consideration of RAM size, storage capacity, ROM size, selection of language, speed of operation and instruction set, register set and port availability and addressability are discussed in the following sections.

RAM SIZE AND STORAGE CAPACITY :

Memory size of a computer means the RAM size required for proper accommodation of the program. In most cases estimation of memory size is difficult, and therefore it should be designed conservatively with provision for expansion. The assembly language program for a typical application can hardly exceed 4 Kbytes, but the size will be higher in the case of high-level language as used in this thesis requiring about 99.4 kbytes of free RAM size for its execution, whereas, any normal microcomputer supports a total RAM size of 640 kbytes. As the program is in C++ it will need about 5 Mbytes of hard disk storage capacity for the storage of the compiler program. Besides, about 4.5 Mbytes of hard disk memory capability will be needed for the occupation of

DOS operating environment where the soft-ware system works. Consequently, a total storage capacity of about 10 Kbytes for the program compilation and about 99.4 Kbytes of RAM size for program execution and 20 MB of hard disk space will be enough for the system development in this thesis work.

ROM SIZE & SELECTION OF LANGUAGE:

The program developed does not use any prererquisite data indeed. So, no ROM size is needed besides the system ROM (system BIOS), Key-board controller chip, and the math co-processor chip.

The control of power electronic system are very time critical, and therefore assembly language is normally used which provides fast execution time. But, a program developed in assembly language is time consuming, tedious, complex to interpret, and may require many iterations. On the other hand, with toady's faster microcomputer, the amount of execution time saved by the assembly language other than high level language is not so important. That is why we have used a high-level language (C++) to develop the software for our control system and thus made the program easy to develop, generous to perceive and compatible to other automation environment using microcomputer control.

SPEED OF OPERATION AND INSTRUCTION SET:

The most important consideration for our on-line microcomputer controlled PWM pulse generation is the speed of operation and instruction set. One of the reasons for which the speed of operation of the microcomputer to be used in the implementation is of utmost important is that, an inverter-fed drive will allow only about 20 μs of delay between two types of switching pulse patterns. That means, when a new pattern of switching pulses are to be generated at the parallel port, it has to replace the on going pulse pattern at the same port within a time of about 20 μs. Otherwise, the drive connected will treat the inverter as operating in OFF line mode. This implies that, on-line microcomputer controlled generation of PWM pulses will not allow any break of pulse availability at the parallel port higher than 20 μs during the calculation of timing instants and execution of other pre-generation steps. This condition for on-line control is maintained in the developed program PTDMWG.CPP and PSDMWG.CPP by employing the multitasking capability of the microcomputer in a time-sharing DOS environment. This technique is explained in a greater detail in section 3.2.4. In this context the requirement of speed of operation is justified by the quick response of the parallel adapter to adapt the new pattern of the switching pulses to be made available at the parallel port as the system demands.

The other and the most important reason, for which the speed of operation of the microcomputer bears remarkable consideration, is the reality of changing any modulation parameter in several steps to cope with the physical demands of the system stability. The subfunction *'CalTime( )'* of the *'main( )'* program performs the calculation of timing instants of the PWM pulses and other pre-generation steps to enable the subfunction *'GenPulse( )'* to generate the required pattern of the pulses. Definitely the subfunction *'CalTime( )'* will take an execution time which depends on the values of modulation parameters *fm, dv, s,* and *vm* as they determine the number of switching instants in a complete cycle. For a standard setting of modulation parameters the calculation of timing instants of PWM pulses requires a time of about 4 ~ 6 μs and in addition, the other pre-generation steps of creating an array of decimal numbers *deci[num_seg]* takes a time of about 20 μs which are determined using some extra stop watches other than Timer1 to be executed in two check programs executed in a 80386 DX-2 microprocessor. As a result, we have a total time of about 26 μs for the execution of the subfunction *'CalTime( )'* of the program PTDMWG.CPP for the generation of three-phase switching pulses and only about 6μs for the execution of the subfunction *'CalTime( )'* of the program PSDMWG.CPP for the generation of single-phase switching pulses. Although the execution time of the subfunction *'CalTime( )'* of PTDMWG.CPP depends on the number of time segments *num*_seg corresponding to the required resolution of the output PWM pulses, the above execution time is made for an average number of time segments. For on-line strategy we want this time to be as small as possible. This is required due to the fact that, when we need to change any of the modulation parameters e.g. modulating frequency *fm* from 50Hz to 200Hz, we cannot allow a step change of frequency from 50Hz to 200Hz directly to ensure stability of the drive system, and what we can do is to reach the frequency level of 200Hz from 50Hz performing several steps of frequency variations e.g. 50Hz, 55Hz, 60Hz, ..., 200Hz. This step variation implies that we shall need a total *'CalTime'* of about (31x 26)= 806 μs (as in this case the total steps to be performed is 31) to reach the required patterns of switching pulses to the inverter. Although the inverter is receiving PWM pulses during this period of *'CalTime'*, yet we have to think whether the system performance will allow us to take this amount of time in a particular operational environment. That is why the question of speed of operation arises in the case of on-line microcomputer controlled PWM waveform generation.

There is no single factor that makes a computer system to work as fast as possible under all conditions. Even systems based on the same processor running at the same speed show considerable differences in their ability to tackle particular tasks [87]. It is beyond our task to explore all the reasons in this concern. But we shall simply discuss the operational speed of a microcomputer to check whether this will meet the speed up to the demand of the system to have an on-line microcomputer controlled DPWM switching pulses. A processor executes the instructions which make up a program. How fast the program is carried out depends on how many instructions the processor can execute per second. Again not all instructions that the processor is called on to execute will take the same amount of time. It depends on how complex or simple the instruction set is. Consequently, the number of instructions per second is measured by introducing as how many instructions on average the processor will obey in a second. This corresponds to the often quoted measure of processor speed : MIPS (Millions of Instructions Per Second). The first desktop computers offered performances below 0.5 MIPS, whereas, today's 386 / 486 based systems offer performances in the region of 10 MIPS [87]. Every computer has a master clock [Figure 2.8(a) & 2.8(b)] which serves to co-ordinate every thing that happens. The original PC's had a clock speed of 4.77 MHz and the first AT 6 MHz or 8 MHz. The faster 286 machines used clock speeds of 10 MHz, 12 MHz, 16 MHz, and even 20 MHz. The 386 system have been available in clock speeds of 16 MHz, 20 MHz, 25 MHz, 33 MHz, and even higher clock speeds. The INTEL 80386 DX-2 has a clock speed of 40 MHz. The 486 has been manufactured in 25 MHz, 33 MHz, 40 MHz, and 66 MHz versions. Figure 2.15 will give some idea of how the clock speed and MIPS compare and how the different processor types offer different ranges of power [87].

One complication of bus transfer rates is the *Wait State*. If a bus attempts to transfer data faster than a device can cope with the data then it is obvious that it will have to wait until the device is ready. This is a *Wait State* whose duration is usually quoted in terms of number of clock pulses. The access time of memory and I / O chips should be compatible with the processor speed to avoid *Wait States* which slow down the program execution.

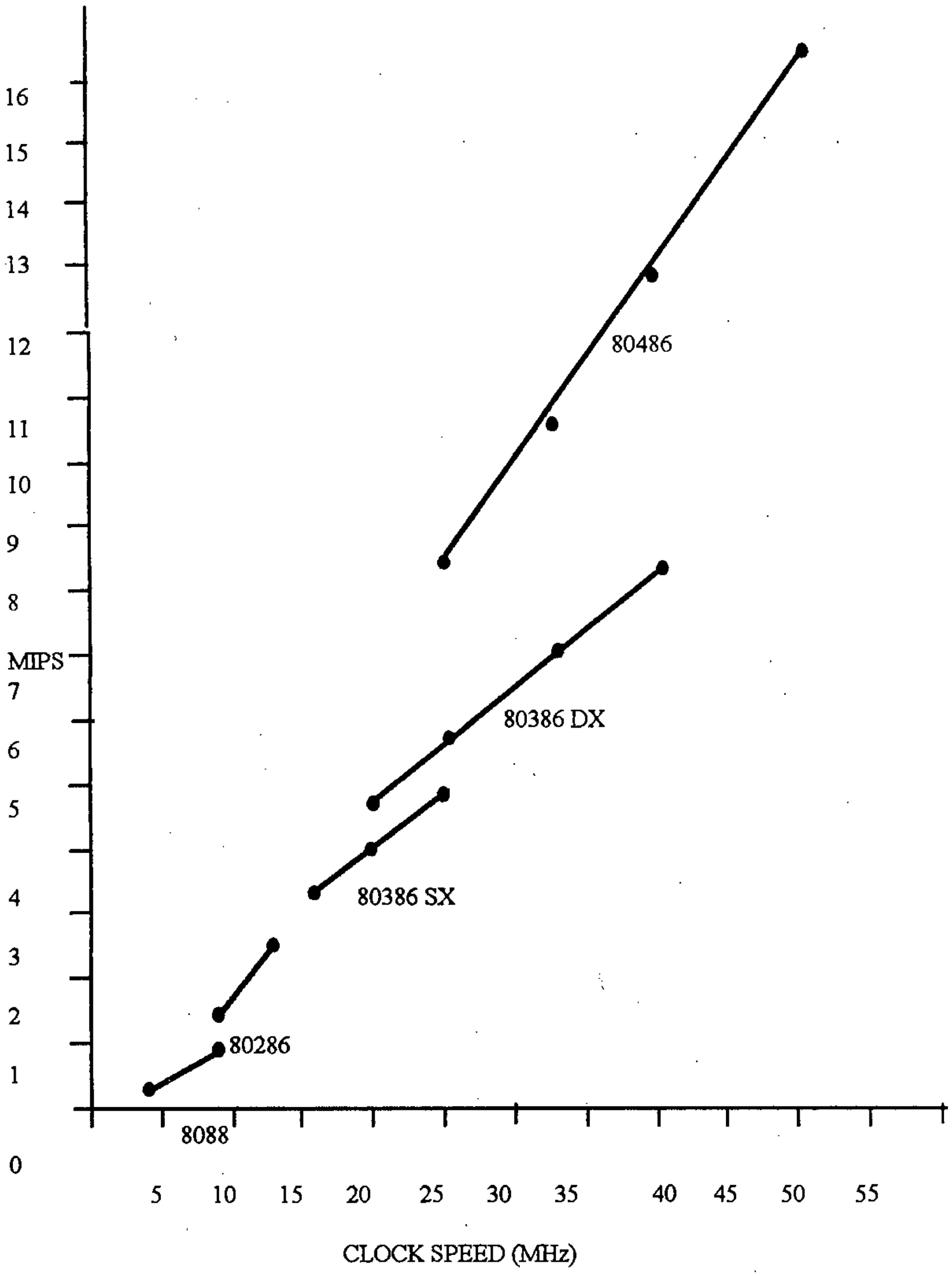


Figure: 2.15
Clock speed and MIPS for the intel family.

The 386 / 486 processors typically take two clock pulses on an average to read or write memory. This means that, addition of a single *Wait State* makes it to take three clock cycles to read / write and thus slowing it down by 33%. For example, if we are using a machine whose standard clock speed is 40 MHz will be a 26 MHz (40x2/3 = 26) machine with *1Wait State*, 20 MHz (40x2/4 = 20) machine with *2 Wait State*, 16 MHz (40x2/5 = 16) machine with *3 Wait State.* Here standard clock speed of a machine means the speed of the machine with *0 Wait State.* The effect of wait states on the standard clock speeds are shown in Table: 1.

As 386 / 486 processor typically takes two clock cycles on an average to read / write memory, their memory access time is equal to the time span of two clock cycles. For example, a 386 processor with 40 MHz standard clock speed can access memory every two clock pulses, giving a fastest memory access time of 50 nano-seconds $\left(2\text{x}\frac{10^9}{40\text{x}10^6}\ \text{ns}\right)$. i.e. 50 ns is the memory access time (MAT) with *0 Wait State* for this microprocessor. And the MAT will be 75 ns, 100 ns, and 125 ns with *1, 2,* and *3 Wait States* respectively. The MAT for some typical processor clock speeds are shown in Table: 2 with different wait states.

We have used an INTEL 80386 DX-2 microcomputer for our program implementation which contains a 32-bit microprocessor having standard clock speed of 40 MHz. The time require for the execution of the subfunction *'CalTime( )'* with a standard set of modulation parameters : *fm=50 Hz, vm=5 volt, s=2500 v/µs,* and *dv=0.5 volt* with an INTEL 80386 DX-2 microcomputer is about 26 µs mentioned before. If we think that this time accounts a *0 Wait State* then the execution time for *'CalTime( )'* with different wait states will be as table-3 which also explores approximate execution times

Standard Clock Speed (MHz)	Clock Speed With 1 Wait State (MHz)	Clock Speed With 2 Wait State (MHz)	Clock Speed With 3 Wait State (MHz)
6	4	3	2.4
8	5.3	4	3.2
10	6.7	5	4
12	8	6	4.8
16	10.7	8	6.4
20	13.3	10	8
25	16.7	12.5	10
33	22	16.5	13.2
40	26.7	20	16
66	44	33	26.4

Table: 1 Variation of various standard clock speeds with different wait states.

Standard Clock Speed (MHz)	MAT With 0 Wait State (neno-sec)	MAT With 1 Wait State (neno-sec)	MAT With 2 Wait States (neno-sec)	MAT With 3 Wait States (neno-sec)
6	333	500	667	833
8	250	375	500	625
10	200	300	400	500
12	167	250	333	417
16	125	188	250	313
20	100	150	200	250
25	80	120	160	200
33	61	91	121	152
40	50	75	100	125
66	30	46	61	76

Table: 2 Memory access time (MAT) for some typical processor clock speeds with different wait states.

Standard Clock Speed (MHz)	Execution Time With 0 Wait State (µs)	Execution Time With 1 Wait State (µs)	Execution Time With 2 Wait States (µs)	Execution Time With 3 Wait States (µs)
6	173	260	347	433
8	130	196	260	325
10	104	155	208	260
12	87	130	173	217
16	65	98	130	163
20	52	78	104	130
25	42	62	83	104
33	32	47	63	79
40	26	39	52	65
66	16	24	32	39

Table: 3 The execution time for the subfunction *'CalTime( )'* of the program PTDMWG.CPP for some typical processor clock speeds with different wait states.

Standard Clock Speed (MHz)	Execution Time With 0 Wait State (μs)	Execution Time With 1 Wait State (μs)	Execution Time With 2 Wait States (μs)	Execution Time With 3 Wait States (μs)
6	40	60	80	100
8	30	45	60	75
10	24	36	48	60
12	20	30	40	50
16	15	23	30	38
20	12	18	24	30
25	10	14	19	24
33	7	11	15	18
40	6	9	12	15
66	4	6	7	9

Table: 4 The execution time for the subfunction *'CalTime( )'* of the program PSDMWG.CPP for some typical processor clock speeds with different wait states.

for *'CalTime( )'* of the program PTDMWG.CPP with some typical processors. In this table we have assumed that the program execution time is directly proportional to the clock speed. A similar representation as table-3 of the execution times for *'CalTime( )'* of the program PSDMWG.CPP is outlined in table-4.

To select a microcomputer for the implementation of on-line strategy of this thesis from the consideration of speed of operation, at first we have to define the range of variation of the modulation parameters to determine the maximum number of steps of variation to be performed to reach the final value from the initial value. The second thing is to know the maximum time allowed by the drive system performance within which the demanded pulse patterns are to be supplied. These will then enable us to select the microcomputer having a particular clock speed determining the total execution time taking a maximum wait states as in table- 3 for three-phase inverter systems and as in table-4 for single-phase inverter system.

CHAPTER-THREE

METHODOLOGY OF ON-LINE MICROCOMPUTER INVERTER SWITCHING PULSES

□ INTRODUCTION

□ DESIGN AND IMPLEMENTATION METHODOLOGY

□ RECEPTION OF PERFORMANCE SPECIFICATIONS

□ SYSTEM DESIGN AND ALGORITHMS DEVELOPMENT

□ SOFTWARE DESIGN AND PROGRAM ORGANIZATION

□ PROGRAM ORGANIZATION

□ DESCRIPTION OF THE PROGRAM 'PTDMWG.CPP'

□ DESCRIPTION OF THE PROGRAM 'PSDMWG.CPP'

□ HARDWARE DESIGN AND INPUT / OUTPUT LINKAGE

3.1 INTRODUCTION

This chapter explores the design steps of the development of on-line microcomputer controlled switching pulse generation for both the single and the three-phase inverter systems. This chapter also explains the organization of two softwares, one is the program PTDMWG.CPP which generates six pulse-patterns at the parallel I/O port for a three-phase inverter system and the another is the program PSDMWG.CPP which generates only two pulse-patterns for the single-phase inverter system. It is noteworthy to mention that the pulse patterns generated by the program PTDMWG.CPP can also be used for a single-phase inverter system using two of the generated pulse-patterns which are 180^0 out of phase with each other.

3.2 DESIGN AND IMPLEMENTATION METHODOLOGY

In this thesis a system is designed for on-line microcomputer controlled DPWM switching pulse generation. Generation of pulses has been implemented for the single and the three-phase inverter-fed drive system. The design and implementation methodology for this system involves the following stages,

1. Reception of performance specifications,
2. System design and algorithms development,
3. Hardware design and input / output linkage, and
4. Software design and program organization.

3.2.1 RECEPTION OF PERFORMANCE SPECIFICATIONS

The first stage of design methodology is to know the ratings and type of the drive to feed by the inveter, the operating ranges of the system, and the control parameters of the same to have an idea about the system performance. To consider the above specifications we have to think of the switching requirements described in the section 2.6.1.

3.2.2 SYSTEM DESIGN WITH ALGORITHMS DEVELOPMENT

The system design starts after receiving the performance specifications. As a first step of system design we have selected the delta modulated switching pulses to switch the MOSFETs of an inverter where the modulation parameters are : 1. Modulating frequency, *fm* 2. Windows width, *dv* or Δv 3. Slope of the error signal, *s* and 4. Peak voltage of the modulating sine wave, *vm*. Second step of system design is the development of equation to calculate the timing instants of the DPWM switching pulses with the above parameters as functions. The equation outlined in the article 2.4 is used to calculate timing instants t[0], t[1], t[2], ..., t[n] for a particular set of modulation parameters. To have the PWM waveform corresponding to those timing instants we have to generate 'logic-1' from t[0] to t[1], 'logic-0' from t[1] to t[2] and so on up to t[n]. The system design stage terminates with a description of generation algorithms in the form of flowcharts as shown in flowchart: 1 through flowchart: 9 which becomes input to the software design.

3.2.3 SOFTWARE DESIGN AND PROGRAM ORGANIZATION

The software involves program modules which are executed in real-time in a time-sequential manner. The software design starts with the microcomputer function algorithms described in the form of flowcharts. Then the program is developed in high-level C^{++} language with a provision of interface through parallel I/O ports. Parallel port uses eight data pins, one for each bit in a byte to be sent. The use of eight wires (or pins) means that eight bits can be sent at once, in parallel. To send a byte of information through the parallel port, we first used to determine which port we shall use. Either the BIOS or direct access to the port registers or adapters can be sent to send the required byte. BIOS interrupt is made when the direct access is unknown. As we have the I/O memory map as shown in Table:5, we have the direct address of the parallel port as 378 in hexadecimal range (0x378), we have used it in our program instead of using BIOS interrupt 17h to be sent by $LPT_{1,}$ $LPT_{2,}$ or LPT_3 as assigned to DX [89]. It is noteworthy that, parallel adapter has three resisters; data resister, interrupt register and strobe register all of which can be read or written. The program used here may be developed either in assembly language or by a mixture of both high-level and assembly language. Thus, software design includes the following steps,

1. Selection of Language: C^{++}, explained in the article 2.6.2,
2. Development of Flowcharts: As shown in flowchart: 1 through 9, and
3. Program Organization.

HEX RANGE	USAGE
000-00F	DMA Chip 8237A-5
020-021	Interrupt 8259A
040-043	Timer 8253-5
060-063	PPI 8255A-5
080-083	DMA Page Registers
OAX	NMI Mask Register
OCX	Reserved
OEX	Reserved
200-20F	Game Control
210-217	Expansion Unit
220-24F	Reserved
278-27F	Reserved
2F0-2F7	Reserved
2F8-2FF	Asynchronous Communications (Secondary)
300-31F	Prototype Card
320-32F	Fixed Disk
378-37F	Parallel printer port
380-38F	SDLC Communications
3A0-3AF	Reserved
3B0-3BF	IBM Monochrome Display/Printer
3C0-3CF	Reserved
3D0-3DF	Color/Graphics
3E0-3E7	Reserved
3F0-3F7	Diskette
3F8-3FF	Asynchronous Communications (Primary)

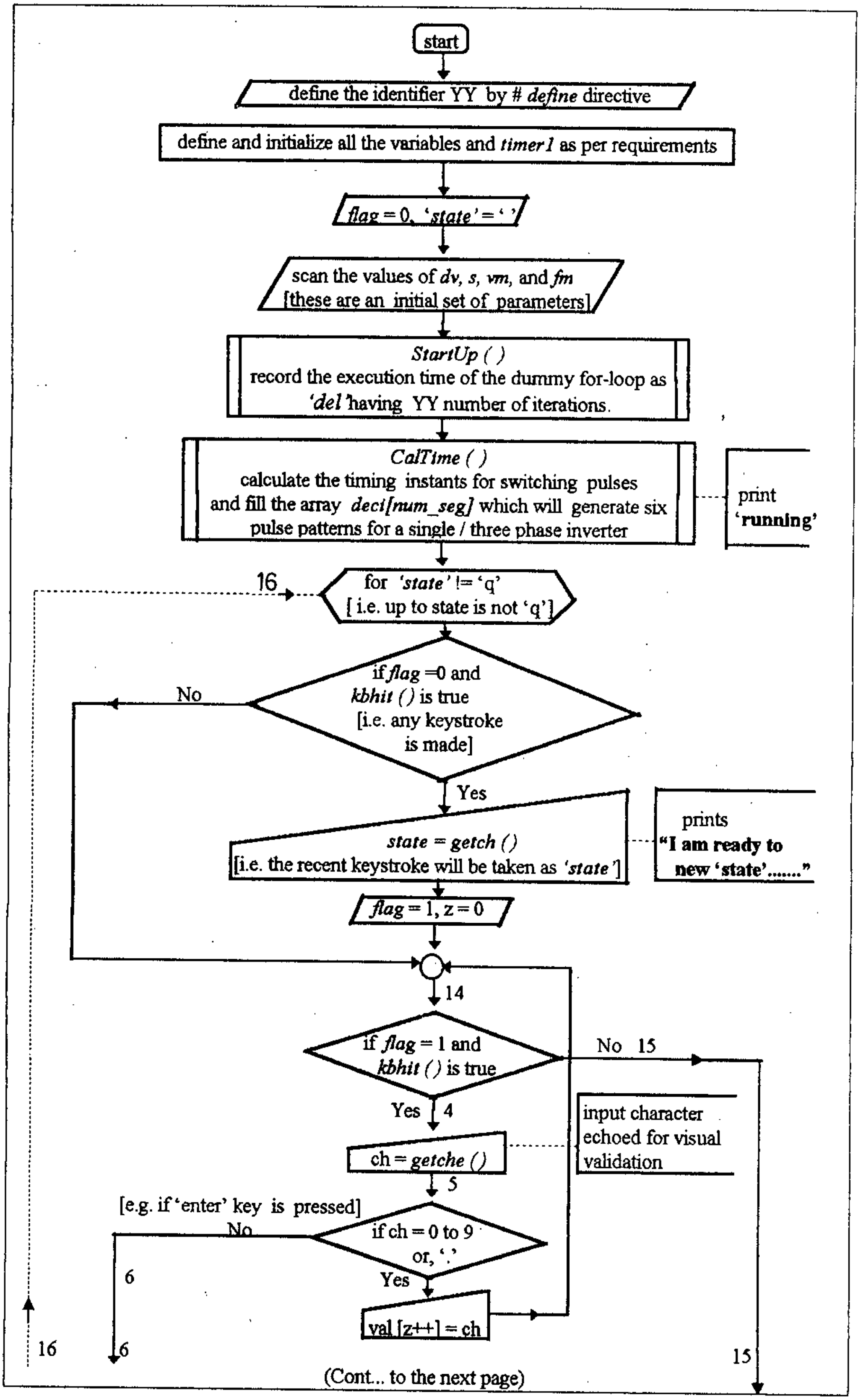
Table: 5 The I / O Address Map of a Microcomputer.

3.2.4 PROGRAM ORGANIZATION

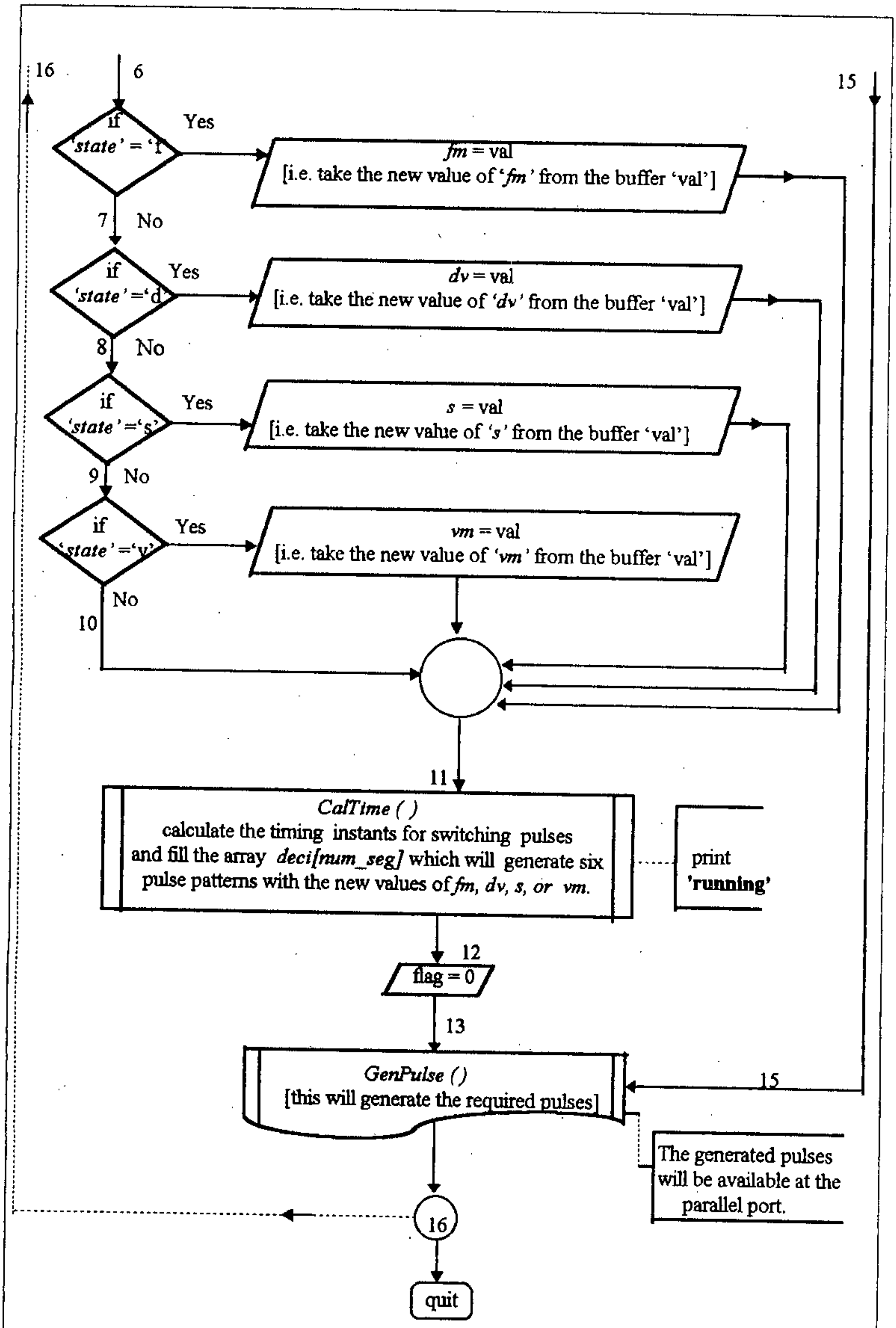
Actually we have developed two programs to implement the system. One of them, named PTDMWG.CPP, is to generate a pattern of six switching pulses at the six individual pins named pin-2, pin-3, pin-4, pin-5, pin-6, and pin-7, for a three-phase inverter switching. In this case, The pulses available at pin-2 and pin-4 can also be used for switching a single-phase inverter. The other, named PSDMWG.CPP, is to generate a pattern of two pulses at the two pins named pin-2 and pin-3 for a single-phase inverter only. Both of the programs contain three types of subfunctions *'CalTime( )', 'StartUp( )',* and *'GenPulse( )'* under the accommodation of *'main( )'* program. Detailed description and organization of the programs are outlined below.

(a) DESCRIPTION OF THE PROGRAM 'PTDMWG.CPP' :

The program PTDMWG.CPP, shown in Appendix.A, contains three subfunctions *StartUp( )', 'CalTime( )'* and *'GenPulse( )'*, under the accommodation of *' main( )'* program. We shall describe the program as outlined in the Flowchart : 1, Flowchart : 2, Flowchart : 3, and Flowchart : 4. First of all, all the appropriate *Header Files* for the library functions are to be included. It is noteworthy that, TIMER.H and TIMER.CPP are included from *Classlib.* Here the 'Timer' is an instance class implementing a stop watch. Then we have to define the total number of iteration YY for the dummy for-loop which will determine the time span of each segment *'del'* . Now the total number of time segments *num_seg* to be scanned for the generation of the pulse pattern will be obtained as *num_seg= int (tm / del)* where *tm* is the period of the pulse pattern as in figure 3.1. It is obvious that a lower value for YY will give better resolution of the generated pulses. Then we have to define and initialize all the variables and the *timer1* as per requirement as shown in Appendix.A.

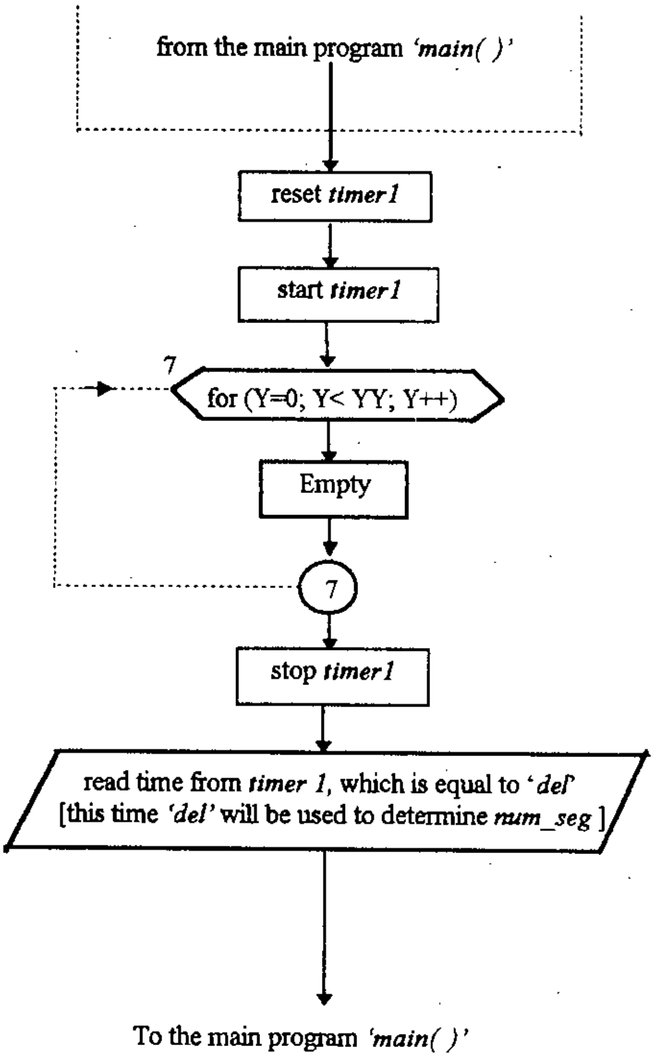

start
define the identifier YY by # define directive
define and initialize all the variables and timer1 as per requirements
flag = 0, 'state' = ' '
scan the values of dv, s, vm, and fm
[these are an initial set of parameters]
StartUp ( )
record the execution time of the dummy for-loop as 'del' having YY number of iterations.
CalTime ( )
calculate the timing instants for switching pulses and fill the array deci[num_seg] which will generate six pulse patterns for a single / three phase inverter
print 'running'
16
for 'state' != 'q'
[i.e. up to state is not 'q']
if flag =0 and kbhit () is true [i.e. any keystroke is made]
No
Yes
state = getch ()
[i.e. the recent keystroke will be taken as 'state']
prints "I am ready to new 'state'......."
flag = 1, z = 0
14
if flag = 1 and kbhit () is true
No 15
Yes 4
ch = getche ()
input character echoed for visual validation
5
[e.g. if 'enter' key is pressed]
No
if ch = 0 to 9 or, '.'
Yes
6
val [z++] = ch
16
6
15

(Cont... to the next page)


16
6
15
if 'state' = 'f'
Yes
fm = val
[i.e. take the new value of 'fm' from the buffer 'val']
7
No
if 'state' ='d'
Yes
dv = val
[i.e. take the new value of 'dv' from the buffer 'val']
8
No
if 'state' ='s'
Yes
s = val
[i.e. take the new value of 's' from the buffer 'val']
9
No
if 'state' ='v'
Yes
vm = val
[i.e. take the new value of 'vm' from the buffer 'val']
No
10
11
CalTime ( )
calculate the timing instants for switching pulses and fill the array deci[num_seg] which will generate six pulse patterns with the new values of fm, dv, s, or vm.
print 'running'
12
flag = 0
13
GenPulse ()
[this will generate the required pulses]
15
The generated pulses will be available at the parallel port.
16
quit

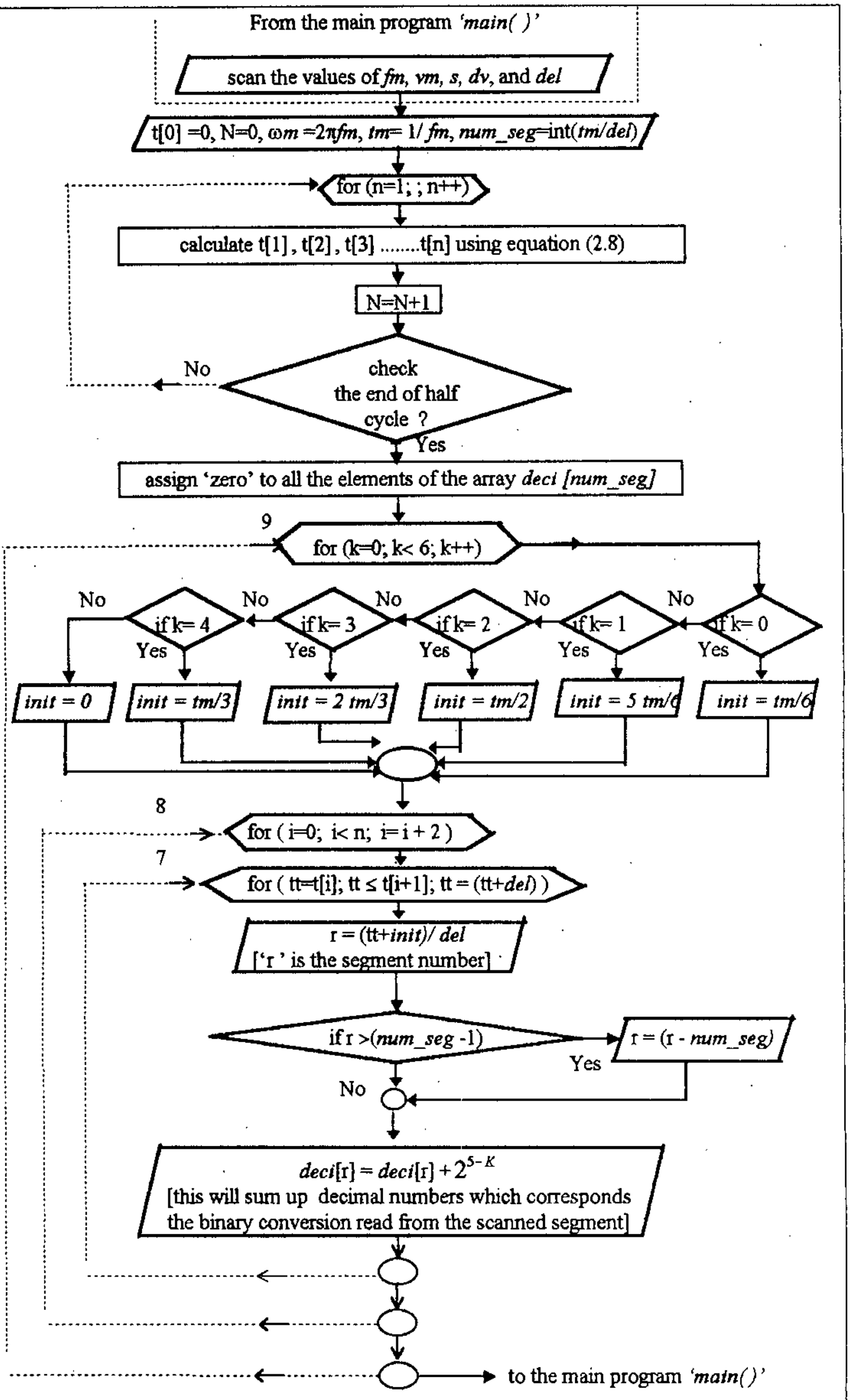
Flowchart: 1

Flow chart of the main program *'main( )'* of PTDMWG.CPP covering the total system.


from the main program *'main( )'*
reset *timer1*
start *timer1*
7
for (Y=0; Y< YY; Y++)
Empty
7
stop *timer1*
read time from *timer 1*, which is equal to '*del*'
[this time *'del'* will be used to determine *num_seg*]
To the main program *'main( )'*

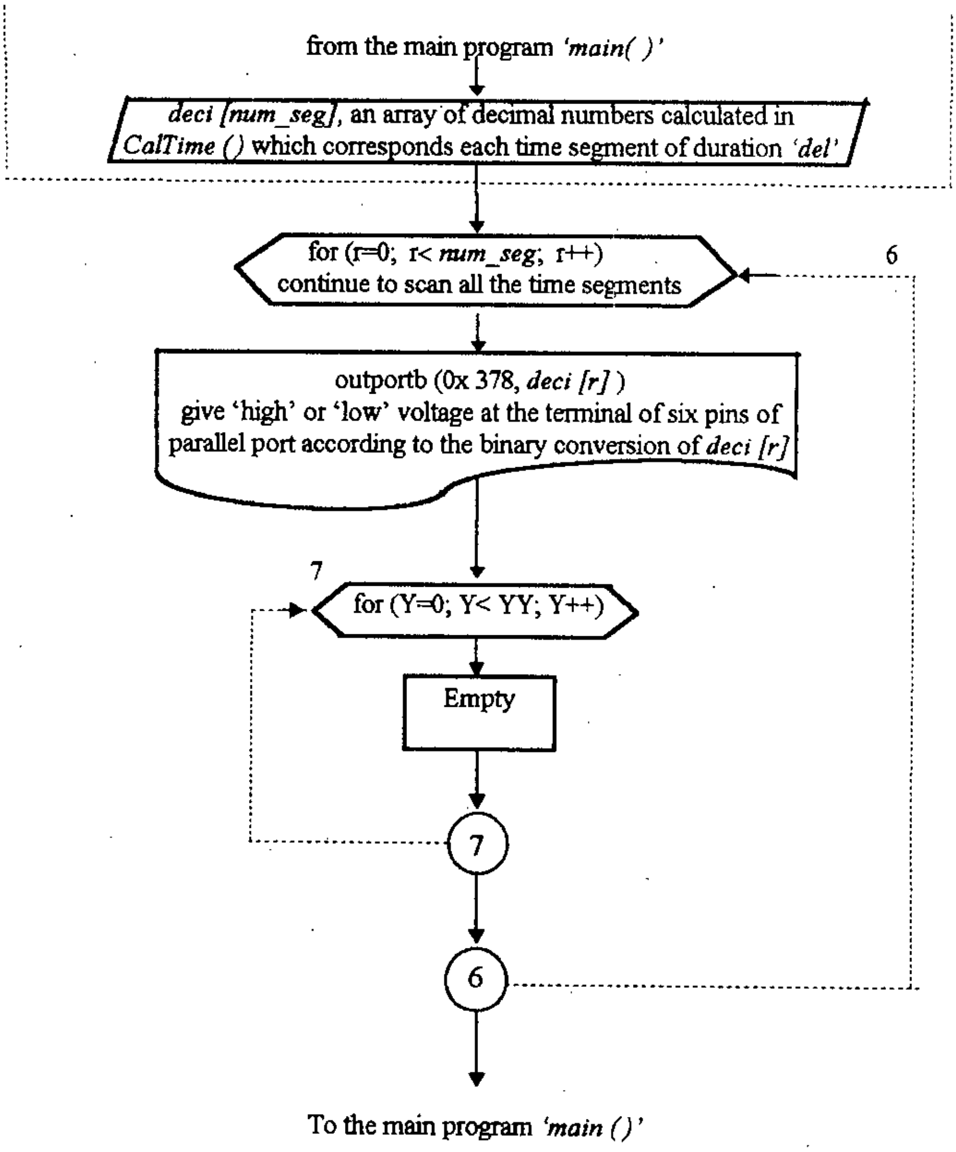
Flowchart : 2

Flow chart of the subfunction *'StartUp ( )'*of the Program PTDMWG.CPP.


From the main program 'main( )'
scan the values of fm, vm, s, dv, and del
t[0] =0, N=0, ωm =2πfm, tm= 1/ fm, num_seg=int(tm/del)
for (n=1; ; n++)
calculate t[1] , t[2] , t[3] ....t[n] using equation (2.8)
N=N+1
No
check the end of half cycle ?
Yes
assign 'zero' to all the elements of the array deci [num_seg]
9
for (k=0; k< 6; k++)
No
if k= 4
Yes
No
if k= 3
Yes
No
if k= 2
Yes
No
if k= 1
Yes
No
if k= 0
Yes
init = 0
init = tm/3
init = 2 tm/3
init = tm/2
init = 5 tm/6
init = tm/6
8
for (i=0; i< n; i=i + 2)
7
for (tt=t[i]; tt ≤ t[i+1]; tt = (tt+del))
r = (tt+init)/ del
['r ' is the segment number]
if r >(num_seg -1)
Yes
r = (r - num_seg)
No
deci[r] = deci[r] + 2^(5-K)
[this will sum up decimal numbers which corresponds the binary conversion read from the scanned segment]
to the main program 'main()'

Flowchart : 3 Flow chart of the subfunction '*CalTime( )*' for the program PTDMWG.CPP.

89406 89405


from the main program 'main( )'
deci [num_seg], an array of decimal numbers calculated in CalTime () which corresponds each time segment of duration 'del'
for (r=0; r< num_seg; r++)
continue to scan all the time segments
6
outportb (0x 378, deci [r])
give 'high' or 'low' voltage at the terminal of six pins of parallel port according to the binary conversion of deci [r]
7
for (Y=0; Y< YY; Y++)
Empty
7
6
To the main program 'main ()'

Flowchart : 4

Flow chart of the subfunction *'GenPulse ( )'* of the program PTDMWG.CPP.

STEP-1: The program will start execution of the main body *void main (void)* which includes an initial set of modulation parameters *dv=1.0 volt, s=2500.0 v/μs, Vm=5.0 volt,* and *fm=50.0 Hz*. It will call the subfunction *StartUp ( )'* which will record the execution time of a dummy for-loop called *'empty'* having YY number of iterations. Here execution time is recorded as *'del'* by the stopwatch defined as *'timer1'*.

STEP-2: The program will call the subfunction *'CalTime( )'* to calculate the timing instants for PWM switching waveforms and to create an array *deci[num_seg]* which contains converted decimal numbers representing each and every time segment of time span '*del*'. This array will be used in the subfunction *'GenPulse( )'* to reproduce each and every time segment which will reveal the entire pulse pattern as a whole.

The portion of *'main( )'* uses the function '*kbhit( )*' which is a check for any recent keystroke made from the keyboard. i.e. '*kbhit( )*' is TRUE if any keystroke is made and FALSE if no keystroke is made.

STEP-3: In this stage the program checks the variable character *'state'* using the function '*kbhit( )*'. If character 'q' is stroked from the keyboard the function '*getch( )*' makes the value of 'state'='q' and the *'main( )'* program quits the generation of pulses at the port and if not the following statements execution is made.

STEP-4: Again '*kbhit( )*' function is introduced to check the keystroke. In this stage it is desired that the user will strike-

'f ' to change the value of *'fm'*,

'd ' to change the value of *'dv'*,

's ' to change the value of *'s'*, and

'v ' to change the value of *'vm'*.

One of the above character stroked is assigned as 'state' by the function *'getch( )'* and for visual validation the monitor screen prints, **'I am ready to take new ['state']'**. In this stage the value of 'z' becomes 0 (zero) and 'flag' becomes 1 (one) which was 0 (zero) before.

STEP-5: The keystroke is again checked with an additional check of 'flag'=1. In this stage it is desired that the user will strike a numeric character (i.e. from 0 to 9) or a decimal point (.) which will give a floating or integer number to be assigned as the new value of the 'state' selected in step-4. Here the stroked character struck is assigned as 'ch' by the function *'getche( )'* and is echoed in the monitor screen for visual validation. Another function *'isdigit (ch)'* is introduced here which returns non-0 if 'ch' is a digit, that is, 0 through 9, otherwise it returns 0. Here the program checks the stroked 'ch' whether it is 0 to 9 or '.'. If 'TRUE', these digits are stored in the buffer memory 'val[z++]' one by one. It is important to mention that the generation of pulses at the parallel port is not disturbed to perform the above work of storing as this is done by the computer in a time-sharing environment. When the new value is given by striking a key, we have to press any key other than 0 to 9 or '.'(e.g. the 'Enter' key). The storing process of buffer memory will then be terminated and the assigned buffer value 'val' is treated as new parameter *fm, dv, s,* or *vm* in accordance with the value of 'state'. As for example, if the 'state' is 'f ' the buffer value 'val' will give the new value of *fm*.

STEP-6: With new values of modulation parameter the *'main( )'* program again calls the subfunction *'CalTime( )'* to recalculate the new timing instants and to refill the array *deci[num_seg]*. At this stage 'flag' is again made 0 (zero). The generation of pulses at the parallel port is not disturbed to perform the execution of the subfunction *'CalTime( )'* as this is done by the computer in a time-sharing environment.

STEP-7: Finally the program calls the subfunction *'GenPulse( )'* which generates the pulse pattern as in figure 3.1 at the prescribed pins using the new array *deci[num_seg]* . The new pulse pattern will replace the on going pattern at an instant only and only when all the pre-generation tasks are completed in a time-sharing environment of the CPU.

It is noteworthy to mention the following points:

1. The integer variable 'flag' is introduced to confirm numerical characters pressed and decimal point will be allowed as new parameter only after defining the 'state' from keystroke as *fm, dv, s,* or *vm.*

2. If the user unmindfully press a key other than 'f ', 'd', 's', or 'v' in step-4, then although step-5 will be executed storing the buffer memory 'val[z++]', the value of 'val' will not be assigned as any new parameter and the program execution will follow the flow-path 4,5,6,7,8,9,10,11,12,13 to continue the generation of previous pulse pattern as shown in the Flowchart:1.

3. If no key is pressed at all in step-3 then the generation of pulse pattern will continue with the initial modulation parameters assigned in step-1 following the flow-path 1,14,15 as shown in Flowchart: 1.

SUBFUNCTION *'StartUp( )'* :

This subfunction determines the time span *'del'* of each segment to be scanned as shown in figure 3.1. A stop watch called *'timer'* records the execution time of a dummy for-loop called *'empty'* as shown in the Flowchart :2 . The stop watch is started at the starting of the for-loop 7 and is stopped at the end of the statement. The execution time recorded in the stop watch is stored as *'del'* to be used in the following subfunctions of *'CalTime ( )'* and *'GenPulse ()'*.

SUBFUNCTION *'CalTime( )'* :

The input for this subfunction are the values of modulation parameters *fm, vm, dv, s* and *del* . It prints, **'running......'** on the monitor screen when execution starts for visual validation.

STEP-1: Calculation of Timing Instants:

In this stage the subfunction will calculate the timing instants t[1], t[2],......., t[n] for the PWM pulse pattern using the following equations,

t[0]= 0.0, N=0,

tm= 1.0/*fm* (period of the pulse pattern which covers an angular distance of 360^0 as in figure....),

$\omega m=2\pi fm$,

$num\_seg = tm / del$,

The timing instants [section 2.4],

$$\mathbf{t[n] = t[n-1] + \frac{2\,dv}{S + (-1)^{n-1} V_m \omega_m \cos\omega_m t[n-1]}} \quad ....(3.1)$$

The loop step to calculate timing instants will be continued up to the check of-

$$t[n] \geq \frac{\pi}{\omega_m} = \frac{t_m}{2.0} \qquad \text{...(3.2)}$$

And the value of N will be increased by 1 at each loop step. The value (N+1) will give the number of timing instants calculated.

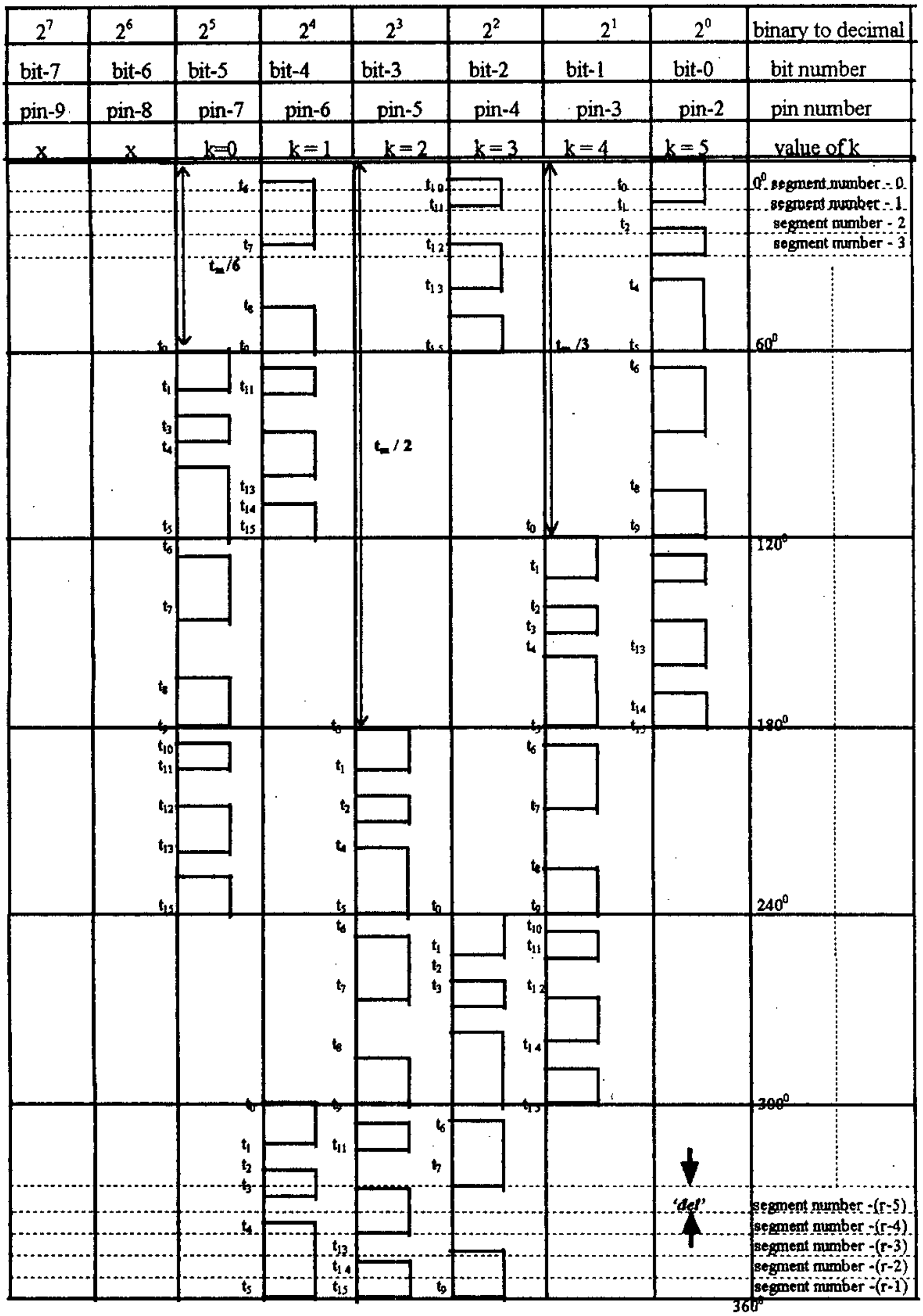
STEP-2: Creation of The Array *deci[num_seg]* :

This step creates an array *deci[num_seg]* of decimal numbers, each element of which represents the respective time segment of the pulse pattern to be generated.

To explain the technique applied to scan the segments, let us consider any of the segments having time span of *del* of the pulse pattern to be sent as a byte at the data pins of the parallel port as shown in figure 3.1. We find that, the segments are a combination of eight 'logic-1' (for positive voltage) or 'logic-0' (for negative voltage)'s. If we can send the converted decimal number of this binary 1/0 combination to the address 0x378 to be sustained for a time span of *del*, then the pattern of this time segment will be reproduced at the eight data pins. And similarly for the other segments we apply the same technique in a time sequential manner for which the total pulse pattern will be generated in the total time period of *tm* (*tm* = *num_seg* x *del*) and this pattern of period *tm* will be repeated again and again according to the on going or refill values of the array *deci[num_seg]*. As we need six individual switching pulses for a three-phase inverter the voltage level of pin-8 and pin-9 are always treated as negative (i.e. logic-0). From the figure 3.1 the conversion equation of decimal numbers to create the array *deci[num_seg]* is,

$$deci[r] = b_{r2}\,2^0 + b_{r3}\,2^1 + b_{r4}\,2^2 + b_{r5}\,2^3 + b_{r6}\,2^4 + b_{r7}\,2^5 + 0\,x2^6 + 0\,x\,2^7$$

$$\text{..(3.3)}$$



Here, r = *num_seg* = Number of segments to be scanned in the total period of *tm* or 360^0 .
del = Time span of each segment to be scanned.

Figure 3.1 : Timing diagram of switching pulses showing the technique of their generation at the corresponding data pins, one for each bit in a byte to be sent at 0x378.

where, r ⇒ represents the segment number. = 0, 1, 2,............, (*num_seg*-1).

b_{r2} ⇒ represents the binary 1 or 0 of the 'element' of segment no. -r and pin no.- 2. and so on for b_{r3}, b_{r4}, ..., b_{r7}.

Here $_{r2, r3, r4, ..., r7}$ are the 'element' number.

In this program the creation of *deci[num_seg]* is made easier applying the reality that we do not require to sum up the decimal converted numbers of the 'elements' which bears a logic-0 (i.e. negative voltage level). As a result, we initially assign '0' to all the elements of the array *deci[num_seg]* and begin to create the *deci[num_seg]* by summing up the decimal converted values only for the 'elements' where a logic-1(i.e. positive voltage) exists and here the array creation will be performed as **pin-wise** instead of **segment-wise** executing the following formula,

$$deci[r] = deci[r] + 2^{5-k} \quad \text{...}(3.4)$$

where, r ⇒ represents the segment number. = 0, 1, 2,............, (*num_seg*-1).

k = 5, for the pulses available at pin-2 or bit-0,

k = 4, for the pulses available at pin-3 or bit-1,

k = 3, for the pulses available at pin-4 or bit-2,

k = 2, for the pulses available at pin-5 or bit-3,

k = 1, for the pulses available at pin-6 or bit-4,

k = 0, for the pulses available at pin-7 or bit-5,

as shown in figure 3.1.

At first we select a 'k' and with this 'k' (i.e. particular pin) the value 'r' is varied from 0 to (*num_seg* -1). And then r is varied for another k and so on for other pins. To make PWM switching pulses with the timing instants calculated in step-1 we make a logic-1 to be sustained from t[0] to t[1] and logic-0 to be sustained from t[1] to t[2] and so on up to t[n] as explained in section 3.2.2.

We know that, six switching pulses to be generated have phase-shifts from one another in a three-phase inverter system. For which, the waveshapes of the switching pulses for each and every pin will be same with the fact that they starts from different initial positions. The initial positions for the pulses are as below, [figure 3.1]

$init = 0.0$ sec $= 0^0$, for k = 5,

$init = tm / 3.0$ sec $= 120^0$, for k = 4,

$init = 2.0\text{x}\, tm / 3.0$ sec $= 1240^0$, for k = 3,

$init = tm / 2.0$ sec $= 180^0$, for k = 2,

$init = 5.0\text{x}\, tm / 6.0$ sec $= 300^0$, for k = 1 and

$init = tm / 6.0$ sec $= 60^0$, for k = 0.

Creation of The Array *deci[num_seg]* from the Flowchart: 3 is explained as below,

At start all the elements of the array *deci*[*num_seg*] are made 0 (zero). i.e. *deci[0]=0, deci[1]=0,................, deci[num_seg - 1]= 0.* For-loop no. 9 is then executed which is to select k from 0 to 5. Let us start with k = 0, which selects the initial position *init = tm /6.0* for the pulse at pin-7.

For-loop 8 and 7 are next executed which includes integer i and N. Here N+1 is the total number of timing instants in a waveshape. The integer i increases by 2 at each step starting from 0. That is the for-loop 7 will be executed for the positive voltage levels (logic-1) only with tt = t[0] to tt = t[1] or with tt = t[2] to tt = t[3] and so on up to t[N-1]. The logic-o segments are not scanned at all. If we start with tt = t[i]= t[0] up to tt = t[i+1] = t[1]. For-loop 7 adds a '*del*' with t[0] at each step up to the t[1]. Then the segment number r becomes,

r = (tt+*init*)/ *del.*

With this r the equation (3.4) will sum up the decimal converted number 2^{5-k} to the previous *deci*[r].

In this manner execution of for-loops 7, 8 and 9 will be completed scanning all the positive voltage levels of the waveshape to be available at pin-7, then of the pin-6, through pin 2 sequentially.

It is noteworthy to mention that, when the value of segment number r exceeds (*num_seg* - 1) i.e. 360^0 it means that the next portion of the pulse will start from zero position i.e. 0^0 . This provision is introduced by the following check,

If r > (*num_seg* - 1)

Then r = r - (*num_seg*).

SUBFUNCTION *'GenPulse( )'* :

The input for this subfunction is an array *deci*[*num_seg*] of decimal numbers each of which represents the required pulse level (1 or 0) at eight pins to be sent as a byte for a time span of '*del*'. It will be explained with reference to the Flow chart: 4.

At start, it executes the for-loop 6 which starts from 0 and reaches (*num_seg* -) with an increment of 1 at each step. i.e. it will deliver the decimal numbers stored in *deci*[0], *deci*[1], *deci*[2],...........*deci*[*num_seg* -1] one by one. This numbers are sent to the address 0x378 using the following statement,

outportb (0x378, *deci*[r]) (Here b ⇒ is to mean byte).

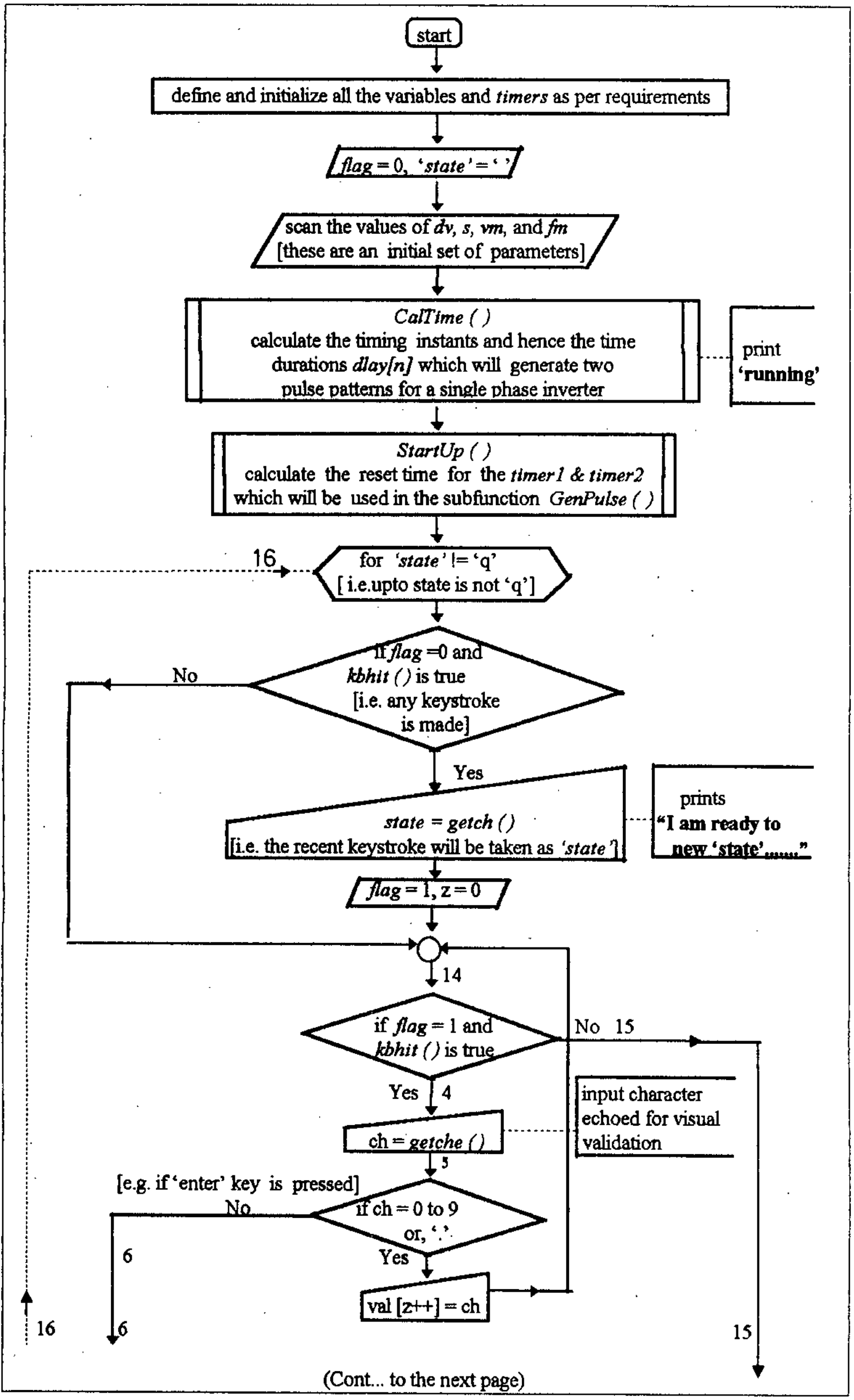
To sustain the above logic levels at the pins up to the time span of '*del*' , we have made the above statement to go through a dummy for-loop named *'empty'*. The iteration number for this dummy for-loop is YY which has been used in the subfunction '*Start Up( )*' . The logic used here is that the execution of this dummy for-loop will make the statement outportb (0x378, *deci*[r]) to be sustained for a time duration of *'del'*.

(b) DESCRIPTION OF THE PROGRAM 'PSDMWG.CPP' :

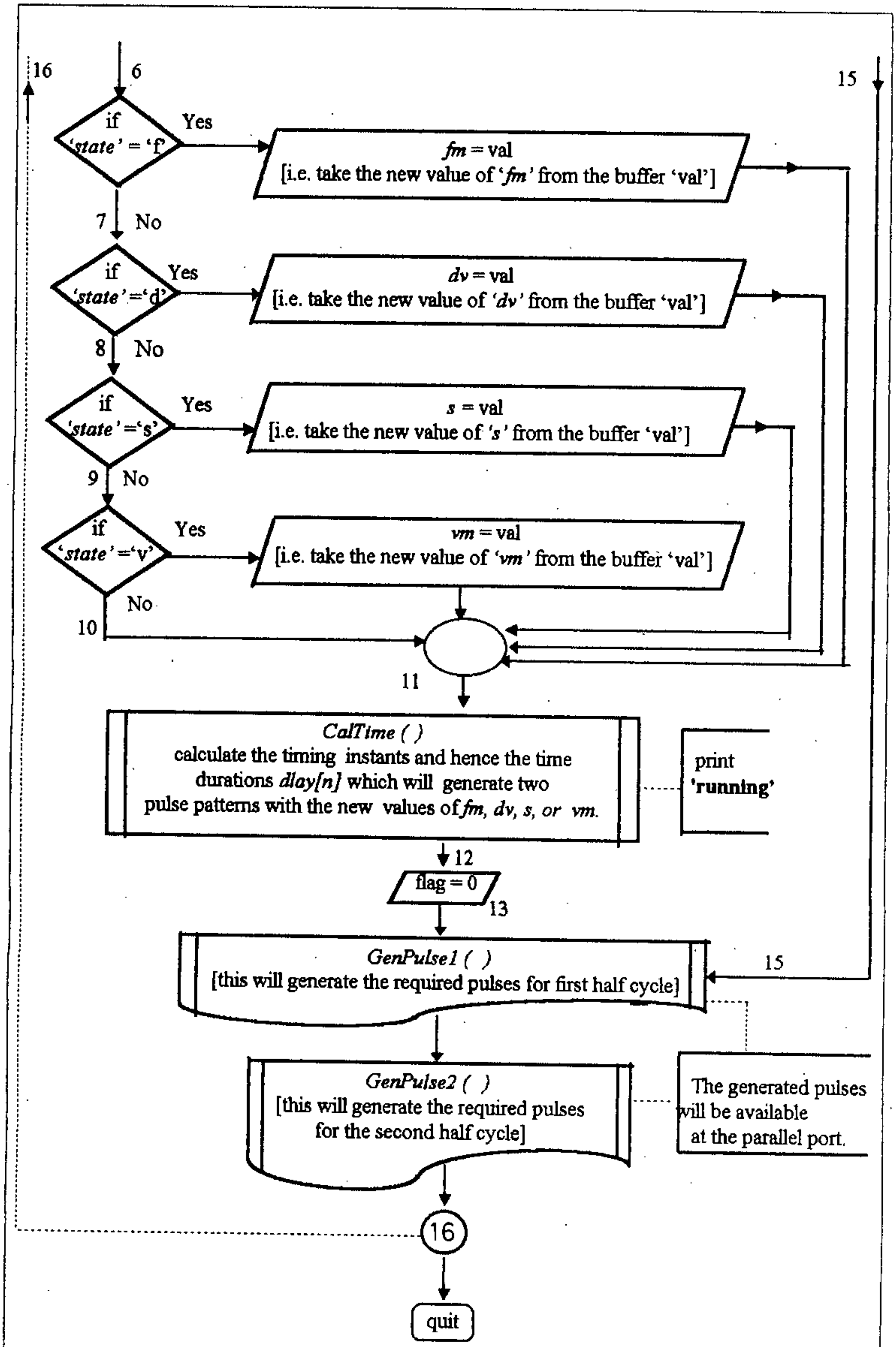
The program PSDMWG.CPP, shown in Appendix.B, contains four subfunctions *'CalTime( )', StartUp( )', 'GenPulse1( )'*, and *'GenPulse2( )'*, under the accommodation of *' main( )'* program. We shall describe the program as outlined in the Flowcharts : 5 - 9. First of all, we have to include all the appropriate *Header Files* for the library functions to define and initialize all the variables and *timers* as per requirement and shown in Appendix.A & B.

STEP-1: The program will start executing the main body *void main (void)* which includes an initial set of modulation parameters *dv=1.0 volt, s=2500.0, vm=5.0 volt,* and *fm=50.0* Hz. It calls the subfunction *'CalTime( )'* to calculate the timing instants and thus the corresponding delay times for PWM switching waveforms which will be used in the subfunctions *'GenPulse1( )'* and *'GenPulse2( )'* to generate the required pulses for a single-phase inverter at pin-2 and pin-3 of the parallel port.

STEP-2: The *'main( )'* program calls the subfunction *'StartUp( )'* which calculates the reset times of two timers *timer1* and *timer2* which is be used in the subfunction *'GenPulse1( )'* and *'GenPulse2( )'*.

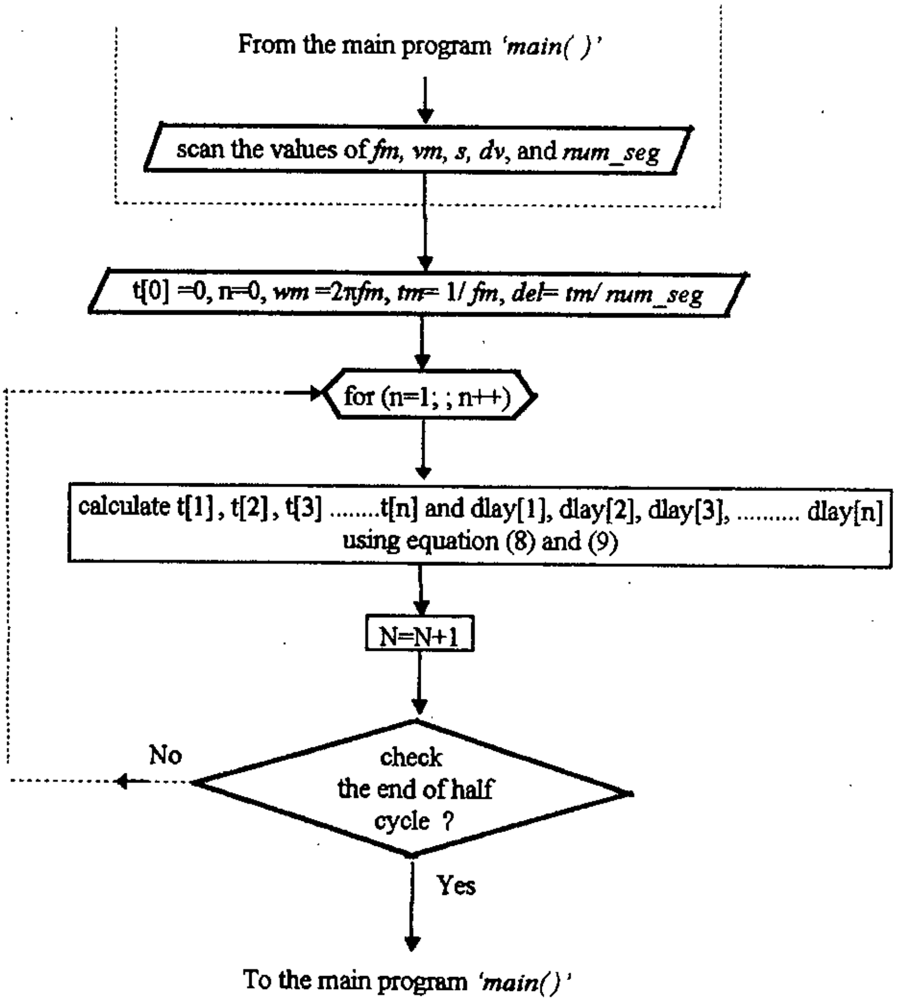

start
define and initialize all the variables and timers as per requirements
flag = 0, 'state' = ' '
scan the values of dv, s, vm, and fm
[these are an initial set of parameters]
CalTime ( )
calculate the timing instants and hence the time durations dlay[n] which will generate two pulse patterns for a single phase inverter
print
'running'
StartUp ( )
calculate the reset time for the timer1 & timer2 which will be used in the subfunction GenPulse ( )
16
for 'state' != 'q'
[i.e.upto state is not 'q']
if flag =0 and
kbhit ( ) is true
[i.e. any keystroke
is made]
No
Yes
state = getch ( )
[i.e. the recent keystroke will be taken as 'state']
prints
"I am ready to
new 'state'......"
flag = 1, z = 0
14
if flag = 1 and
kbhit ( ) is true
No 15
Yes
4
ch = getche ( )
input character
echoed for visual
validation
5
[e.g. if 'enter' key is pressed]
No
if ch = 0 to 9
or, '.'
Yes
6
val [z++] = ch
16
6
15

(Cont... to the next page)


16
6
15
if
'state' = 'f'
Yes
fm = val
[i.e. take the new value of 'fm' from the buffer 'val']
7
No
if
'state'='d'
Yes
dv = val
[i.e. take the new value of 'dv' from the buffer 'val']
8
No
if
'state'='s'
Yes
s = val
[i.e. take the new value of 's' from the buffer 'val']
9
No
if
'state'='v'
Yes
vm = val
[i.e. take the new value of 'vm' from the buffer 'val']
No
10
11
CalTime ( )
calculate the timing instants and hence the time durations dlay[n] which will generate two pulse patterns with the new values of fm, dv, s, or vm.
print
'running'
12
flag = 0
13
GenPulse1 ( )
[this will generate the required pulses for first half cycle]
15
GenPulse2 ( )
[this will generate the required pulses for the second half cycle]
The generated pulses will be available at the parallel port.
16
quit

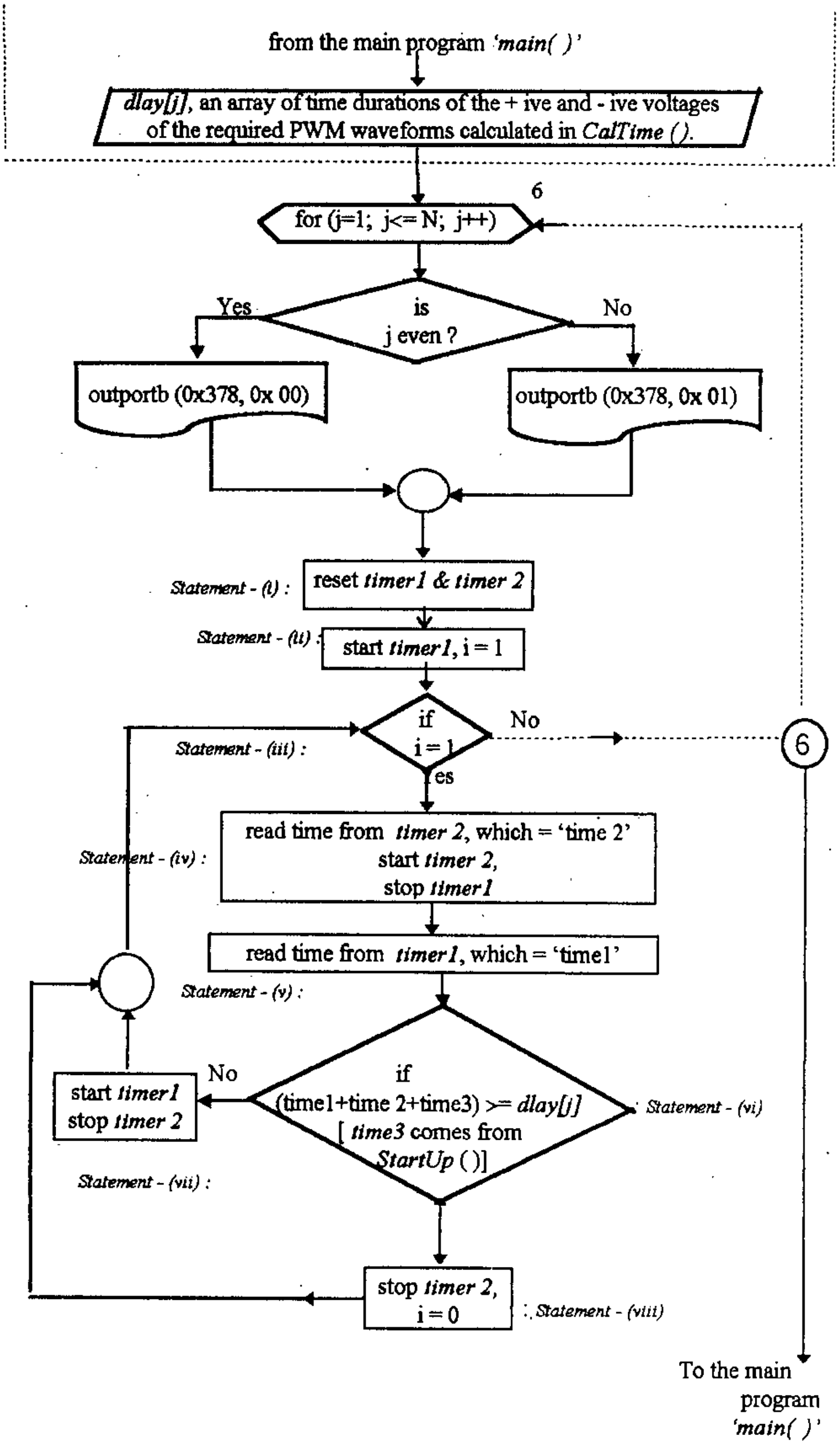
Flowchart: 5

Flow chart of the main program *'main( )'* of PSDMWG.CPP covering the total system.


From the main program 'main( )'
scan the values of fm, vm, s, dv, and num_seg
t[0] =0, n=0, wm =2πfm, tm= 1/ fm, del= tm/ num_seg
for (n=1; ; n++)
calculate t[1], t[2], t[3] ....t[n] and dlay[1], dlay[2], dlay[3], ... dlay[n] using equation (8) and (9)
N=N+1
check the end of half cycle ?
No
Yes
To the main program 'main()'

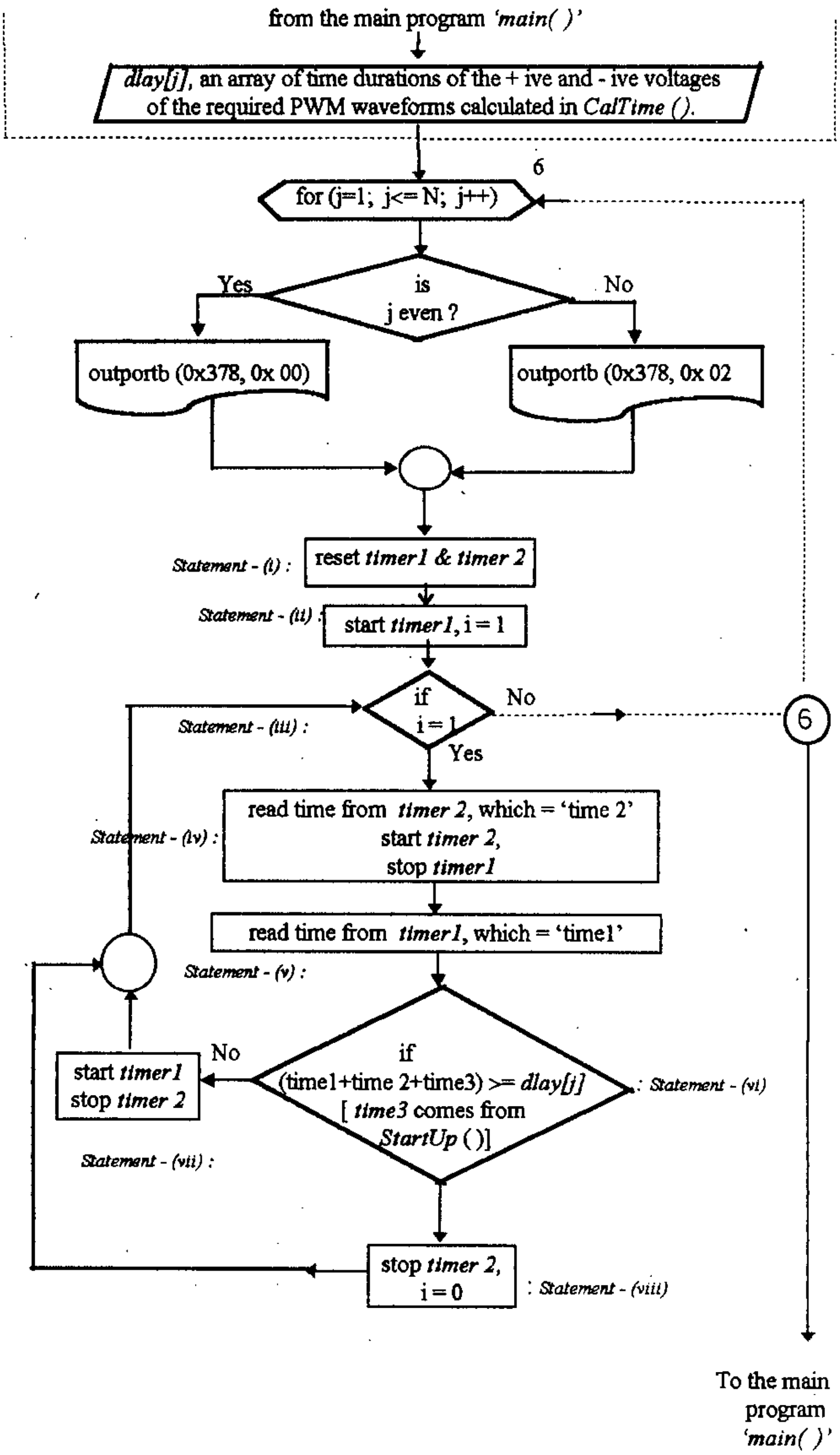
Flowchart : 6

Flow chart of the subfunction '*CalTime( )*' for the program PSDMWG.CPP.



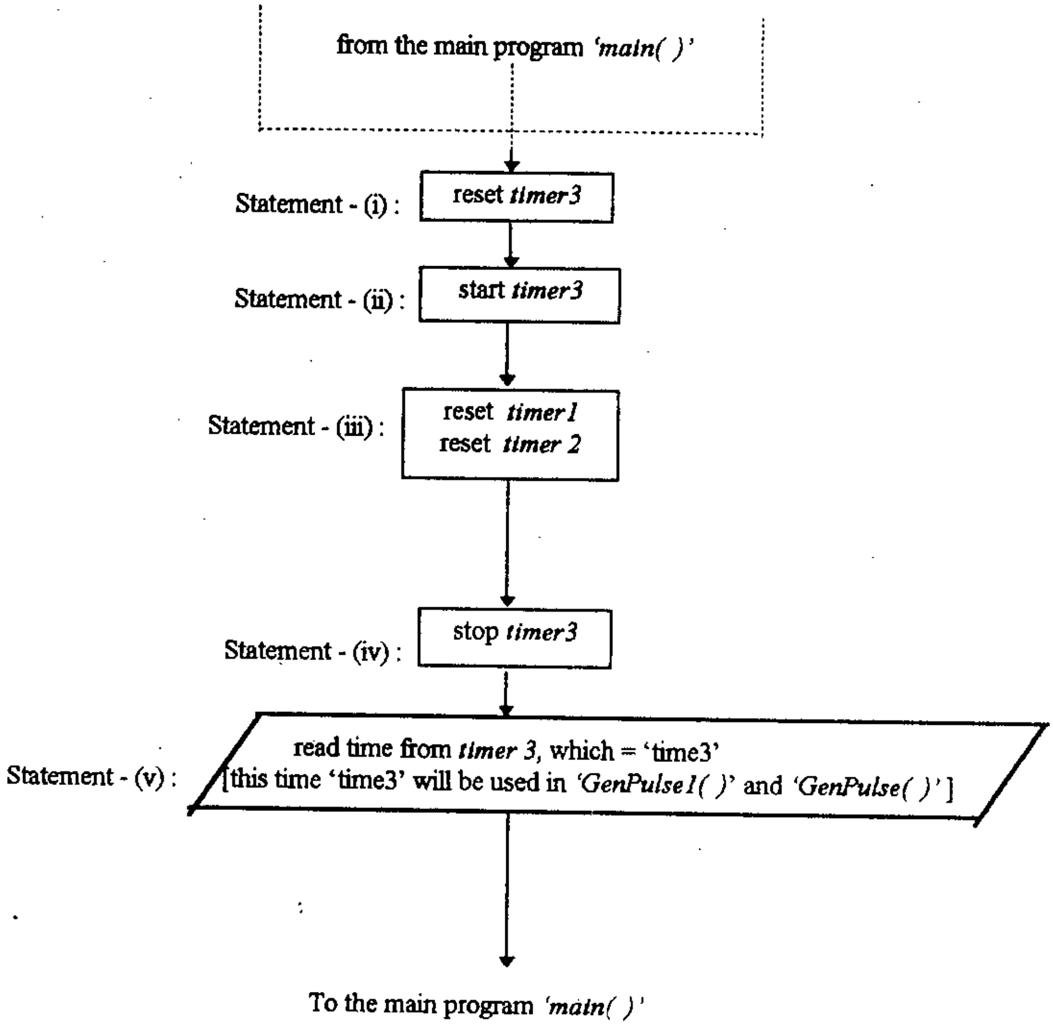
Flowchart: 7

Flow chart of the subfunction '*GenPulse1 ( )*'of the program PSDMWG.CPP.



Flowchart: 8

Flow chart of the subfunction '*GenPulse2 ( )*' of the program PSDMWG.CPP.


from the main program *'main( )'*
Statement - (i) :
reset *timer3*
Statement - (ii) :
start *timer3*
Statement - (iii) :
reset *timer1*
reset *timer 2*
Statement - (iv) :
stop *timer3*
Statement - (v) :
read time from *timer 3*, which = 'time3'
[this time 'time3' will be used in *'GenPulse1( )'* and *'GenPulse( )'*]
To the main program *'main( )'*

Flowchart : 9

Flow chart of the subfunction *'StartUp ( )'* of the Program PSDMWG.CPP.

STEP-3: In this stage the program checks the variable character *'state'* using the function *'kbhit( )'*. If character 'q' is struck from the keyboard the function *'getch( )'* makes the value of 'state'='q' and the *'main( )'* program stops the generation of pulses at the port and if it does confront following statements execution .

STEP-4: *'kbhit( )'* function is introduced at this stage to check the keystroke. It is desired that the user will strike,

'f ' to change the value of *'fm'*,

'd ' to change the value of *'dv'*,

's ' to change the value of *'s'*, and

'v ' to change the value of *'vm'*.

One of the above character struck will be assigned as 'state' by the function *'getch( )'* and for visual validation the monitor screen prints, **'I am ready to take new ['state']'** and the value of 'z' becomes 0 (zero) and 'flag' becomes 1 (one) which was 0 (zero) before.

STEP-5: The keystroke is again checked with an additional check of 'flag'=1. It is now desired that the user will strike a numeric character (i.e. from 0 to 9) or a decimal point (.) which will give a floating or integer number to be assigned as the new value of the 'state' selected in step-4. Here the struck character is assigned as 'ch' by the function *'getche( )'* and is echoed in the monitor screen for visual validation. Another function *'isdigit (ch)'* is introduced here which returns non-0 if 'ch' is a digit, (i.e. 0 through 9), otherwise it returns 0. The program checks the stroked 'ch' whether it is 0 to 9 or '.'. If TRUE, these digits are stored in the buffer memory 'val[z++]' one by one. It is important to mention that the generation of pulses at the parallel port is not disturbed to perform the above work of storing as this is done by the computer in a time-sharing environment mode when the new value is given from the keystroke, we have to press any key other than 0 to 9 or '.'(e.g. the 'Enter' key) to terminate storing of buffer memory and the assigned buffer value 'val' is treated as new parameter *fm, dv, s,* or *vm* in accordance with the value of 'state'. As for example, if the 'state' is 'f' the buffer value 'val' will give the new value of *fm.*

STEP-6: With the new value of modulation parameters the *'main( )'* program calls the subfunction *'CalTime( )'* to recalculate the new timing instants and 'Flag' is again made 0 (zero). The generation of pulses at the parallel port is not disturbed to perform the execution of the subfunction *'CalTime( )'* as this is done by the computer in a time-sharing environment mode.

STEP-7: The program calls the subfunction *'GenPulse1( )'* which generates the pulses up to the first half cycle of the total period i.e. this subfunction will generate pulses from 0^0 through 180^0 as in figure 2.21 at the pin-2 and pin-3.

STEP-8: Then the program calls the subfunction *'GenPulse2( )'* which generates the pulses up to the second half cycle of the total period i.e. this subfunction will generate pulses from 180^0 through 360^0 as in figure 2.21 at the pin-2 and pin-3.

After the completion of step-7 and 8 the entire pulse pattern will reveal and repeat. The new pulse pattern will replace the on going pattern at an instant only and only when all the pre-generation tasks are completed in a time-sharing environment mode of the CPU.

It is noteworthy to mention the following points,

1. The integer variable 'flag' is introduced to make confirm that numerical characters and decimal point given from the keystroke will be allowed as new parameter only after defining the 'state' from keystroke as *fm, dv, s,* or *vm*.

2. If the user unmindfully strikes a key other than 'f ', 'd', 's', or 'v' in step-4, then although step-5 will be executed storing the buffer memory 'val[z++]', the value of 'val' will not be assigned any new parameter and the program execution will follow the flow-path 4,5,6,7,8,9,10,11,12,13 to continue the generation of previous pulse pattern as shown in the Flowchart:5.

3. If key is pressed in step-3 then the generation of pulse pattern will go on with the initial modulation parameters assigned in step-1 following the flow-path 1,14,15 as shown in Flow chart: 5.

SUBFUNCTION *'CalTime( )'* :

The input data for this subfunction are the values of modulation parameters *fm, vm, dv,* and *s* . It prints, **'running......'** on the monitor screen when execution starts for visual validation. This subfunction will calculate the timing instants t[1], t[2],......., t[n] and the corresponding delay times *dlay*[1], *dlay*[2], *dlay*[n] for the PWM pulse pattern using the following equations,

t[0]= 0.0, N=0,

tm= 1.0/ *fm* (period of the pulse pattern which covers an angular distance of 360^0 as shown in figure3.2.),

$\omega_m = 2\pi fm$,

The timing instants [section 2.4],

$$\mathbf{t[n] = t[n-1] + \frac{2\,dv}{S + (-1)^{n-1} V_m \omega_m \cos \omega_m t[n-1]}}$$

$$dlay[n] = t[n] - t[n-1] \quad \text{..} \quad (3.5)$$

where n starts from 1.

e.g. *dlay*[1] = t[0] - t[1], *dlay*[2] = t[1] - t[2] and so on.

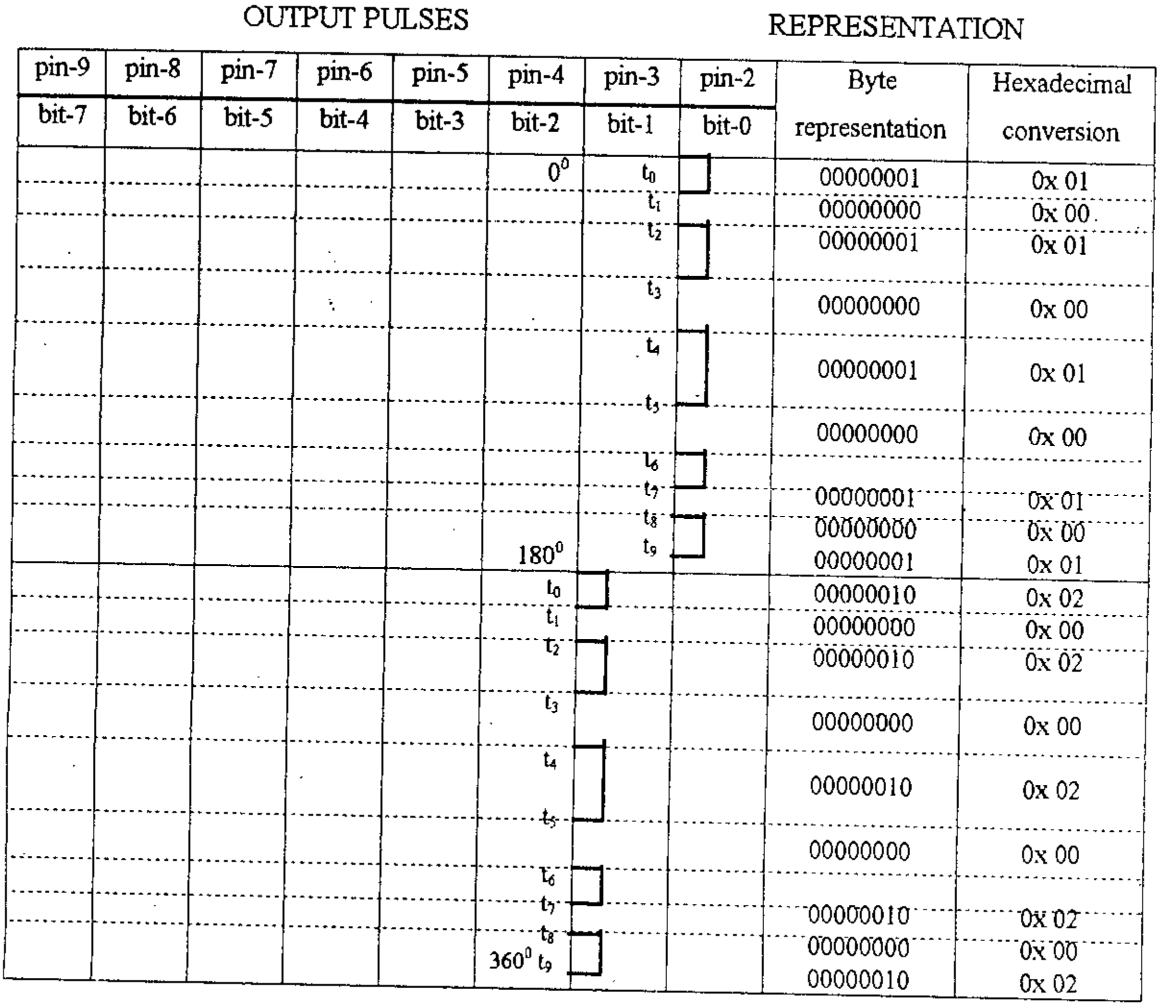
The loop step of calculating timing instants will be continued up to the check of-

$$t[n] \geq \frac{\pi}{\omega_{m,}} = \frac{t_m}{2.0} \quad \text{..} (3.6)$$

And the value of N will be increased by 1 at each loop step. The value (N+1) will give the number of timing instants calculated here.

SUBFUNCTION *'GenPulse1( )'* :

The input data for this subfunction is an array dlay[n] of the delay times calculated from equation (3.5). This subfunction will generate pulses from 0^0 through 180^0 as in figure 3.2 at the pin-2 and pin-3. The technique applied here is that, logic-1(i.e. positive voltage) will be sustained for the time duration of *dlay*[1], then logic-0 will be sustained for a time duration of *dlay*[2], then again logic-1 for time duration of *dlay*[3] and so on up to the first half cycle at pin-2 (i.e. bit-0). On the contrary, for this total half cycle logic-0 is to be sustained at pin-3 (i.e. bit-1). Now we can say from figure 3.2 that, to generate pulses for this half cycle we have to send 00000001_b or Hex 01 to the address 0x 378 to be sustained for the time durations of *dlay*[1], *dlay*[3], *dlay*[5]..... (i.e. for *dlay*[j], where, j is odd) and we have to send 00000000_b or Hex 00 for the time durations of *dlay*[2], *dlay*[4],....(i.e. for *dlay*[j], where, j is even). The statements used are,



OUTPUT PULSES REPRESENTATION

pin-9	pin-8	pin-7	pin-6	pin-5	pin-4	pin-3	pin-2	Byte	Hexadecimal
bit-7	bit-6	bit-5	bit-4	bit-3	bit-2	bit-1	bit-0	representation	conversion
					0^0	t_0		00000001	0x 01
						t_1		00000000	0x 00
						t_2		00000001	0x 01
						t_3		00000000	0x 00
						t_4		00000001	0x 01
						t_5		00000000	0x 00
						t_6			
						t_7		00000001	0x 01
						t_8		00000000	0x 00
					180^0	t_9		00000001	0x 01
					t_0			00000010	0x 02
					t_1			00000000	0x 00
					t_2			00000010	0x 02
					t_3			00000000	0x 00
					t_4			00000010	0x 02
					t_5			00000000	0x 00
					t_6				
					t_7			00000010	0x 02
					t_8			00000000	0x 00
					360^0 t_9			00000010	0x 02

Figure 3.2:

Timing diagram of switching pulses showing the technique of their generation at the corresponding data pins, to be sent at 0x 378 for a single-phase inverter only.

outportb (0x378, 0x 01)..................... (1) with j odd.
outportb (0x378, 0x 00).......................(2) with j even.
(Here b ⇒ is to mean byte).

These bytes are sent up to 180^0 . With reference to the Flow chart: 7. At first it executes the for-loop no.6 which is to start j from 0 and reaches N with an increment of 1 at each step, i.e. it will deliver the delay times stored in *dlay*[1], *dlay*[2], *dlay*[3],...........*dlay*[N] one by one. An 'if statement' will check whether j is even or odd and will choose 'statement (1)' or 'statement (2)' stated above accordingly. To sustain the above logic levels at pin-2 and pin-3 up to the time durations of dlay[j] three stop watches are being used these are: *timer1* and *timer2* -used in the subfunction and *timer3* -used in subfunction '*StartUp( )*'.

<u>From Flow chart: 7</u>

Statement - (i) Resets *timer1* and *timer2*.

Statement - (ii) Starts *timer1* and assigns i = 1.

Statement - (iii) Checks i = 1; if TRUE then goes to Statement - (iv).

Statement - (iv) Reads 'time2' of the *timer2*, starts *timer2*, and stops *timer1*.

Statement - (v) Reads 'time1' of the *timer1*. i.e. 'time1' includes the execution times for Statement - (ii), Statement - (iii), and Statement - (iv).

Statement - (vi) Checks ('time1' + 'time2' + 'time3')≥ *dlay*[j] ; if FALSE then executes Statement - (vii).

Statement - (vii) Starts *timer1* and stops *timer2*. And the above statements are again executed from Statement - (iii).

In the next executions 'time1' will include the execution times of Statement - (vii), Statement - (iii), and Statement - (iv). And 'time2' will include the execution times of Statement - (v) and Statement - (vi).

In this way, when Statement - (vi) will be TRUE, then Statement - (viii) will stop *timer2* and assign i = 0 which makes Statement - (iii) FALSE to recall the for-loop no.6 to supply the next delay time *dlay*[j] and so on up to *dlay*[N].

The 'time3' used above includes the time required to execute Statement-(i) which resets the *timer1* and *time2* .

<u>SUBFUNCTION *'GenPulse2( )'* :</u>

The input data for this subfunction is the same array *dlay*[n] used in the subfunction *'GenPulse1( )'*. As this subfunction is to generate pulses from 180^0 through 360^0 as in figure 3.2, this will be executed after the execution of the subfunction *'GenPulse1( )'* . Here logic-1(i.e. positive voltage) is sustained for the time duration of *dlay*[1], then logic-0 is sustained for a time duration of *dlay*[2], logic-1 for time duration of *dlay*[3] and so on up to the first half cycle at pin-3 (i.e. bit-1). On the contrary, the total half cycle logic-0 is sustained at pin-2 (i.e. bit-0). This means that, to generate pulses for second half cycle we have to send 00000010$_b$ or Hex 02 to the address 0x 378 to be sustained for the time durations of *dlay*[1], *dlay*[3], *dlay*[5]..... (i.e. for *dlay*[j], where, j is odd) and we have to send 00000000$_b$ or Hex 00 for the time durations of *dlay*[2], *dlay*[4],....(i.e. for *dlay*[j], where, j is even). The statements to be used are,

outportb (0x378, 0x 02)...................... (1`)   with j odd.

outportb (0x378, 0x 00).........................(2`)   with j even.

(Here b ⇒ is to mean byte).

These bytes are to be sent up to 360^0 . With reference to the Flow chart: 8 the program execution will be same as that of subfunction *'GenPulse2( )'* as stated above with the change that 'statement (1)' will be replaced by 'statement (1` ) and 'statement (2)' will be replaced by 'statement (2`)'

SUBFUNCTION *'StartUp( )'* :

This subfunction determines 'time3' used in the subfunction *'GenPulse( )'* using the stop watch *timer3.*

From Flow chart: 9

Statement - (i) Resets *timer3.*

Statement - (ii) Starts *timer3.*

Statement - (iii) Resets *timer1* and *timer2.*

Statement - (iv) Stops *timer3.*

Statement - (v) Reads 'time3' stored in *timer3,* which includes the time to reset *timer1* and *timer2.*

3.2.5 HARDWARE DESIGN AND INPUT / OUTPUT LINKAGE :

The hardware to be used for the development of the on-line microcomputer controlled strategy is an INTEL 80386 DX-2 microcomputer or any other microcomputer fulfilling the ' Computer Requirements' as explained in section 2.6.2. In brief, we can summarize that the microcomputer to be used for our implementation should have,

1. About 9.5 Mbytes of hard disk memory for C^{++} language compilation and creation of MS-DOS environment.
2. About 58 Kbytes of free RAM size for the execution of the developed program.
3. System BIOS (EPROM), Keyboard Controller (EPROM), and Math Co-processor (if not inbuilt as in 486 and pentium processor).
4. Enough Clock Speed and as well as MIPS to support the time requirements of step variations of the modulation parameters up to the maximum operating range.
5. Display monitor with Monitor Adapter for the input character to be echoed and for screen printing for visual validation.
6. Keyboard with Keyboard Adapter for manual input of modulation parameters.
7. Parallel or printer Adapter with 25-pin D-shell female connector I / O port.

The generated switching pulses will be available at the prescribed pins of the D-shell parallel port as shown in figures 3.1 and 3.2 for the program PTDMWG.CPP or PSDMWG.CPP respectively. The next step is to apply these six-pulse pattern to the switching elements (MOSFET's) of the inverter with proper isolating and interfacing circuits as explained in the next chapter.

CHAPTER-FOUR

MICROCOMPUTER CONTROLLED INVERTER OPERATION

□ INTRODUCTION

□ INVERTER SWITCHING SIGNALS PRODUCED BY MICROCOMPUTER

□ INVERTER OPERATION

□ DESIGN AND IMPLEMENTATION OF MOSFET-INVERTER

□ OUTPUT WAVEFORMS OF INVERTER

4.1 INTRODUCTION

This chapter explains the design of practical isolation circuits between the microcomputer producing six-channel pulse pattern generated and the MOSFET of the inverter. Switching signals and their corresponding inverter output waveforms for three-phase resistive load with the variation of PWM parameters are described in the following sections. Comparison of practical waveforms with the calculated timing instants of DPWM switching pulses are presented to validate the approach taken during this research.

4.2 INVERTER SWITCHING SIGNALS PRODUCED BY MICROCOMPUTER

Six switching pulses are available at the parallel port of the computer, the waveforms of which are presented in this section for various sets of modulation parameters. The switching pulses obtained at the pin-2 through pin-7 of the parallel port have phase-shifts as shown in figure 3.1 of the previous chapter. As we can obtain only two of the switching- waveshape outputs in a dual-trace oscilloscope at a time, switching-waveshape of pin-2 and pin-7 are presented as illustration. The waveshapes of pin-2 and pin-7 are 60^0 out of phase.

4.2.1 SWITCHING SIGNALS FOR FREQUENCY VARIATION

Typical switching waveforms for various modulation patterns generated by microcomputer to control a three-phase inverter are shown in figures 4.1 to 4.12. The following waveforms of figures 4.1 to 4.3 show the effect of variation of operating frequency of the inverter.

(a) Switching signals for **s = 2500.0 v/μs, dv = 1.0 volt, Vm = 5.0 volt** and **fm = 25.0 Hz** are shown in figure 4.1. Here the number of off and on times (N) are 25 having the timing instants as given in table-6.

(b) Switching signals for **s = 2500.0 v/μs, dv = 1.0 volt, Vm = 5.0 volt** and **fm = 50.0 Hz** are shown in figure 4.2. Here the number of off and on times (N) are 11 having the timing instants as given in table-7.

(c) Switching signals for **s = 2500.0 v/μs, dv = 1.0 volt, Vm = 5.0 volt** and **fm = 65.0 Hz** are shown in figure 4.3. Here the number of off and on times (N) are 5 having the timing instants as given in table-8.

It can be inferred from above observation that the frequency of switching decreases with increasing operation frequency of the reference wave as all other parameters are kept constant. Also, variation of switching frequency is accompanied by widening of pulses, indicating that inverter output voltage will increase with the increase of operating frequency.

4.2.2 SWITCHING SIGNALS FOR SLOPE VARIATION

The following waveform of figures 4.4 to 4.6 show the effect of variation of slope of the modulator on the switching waveforms generated by microcomputer.

(a) Switching signals for **s = 2000.0 v/μs, dv = 1.0 volt, Vm = 5.0 volt** and **fm = 50.0 Hz** are shown in figure 4.4. Here the number of off and on times (N) are 7 having the timing instants as given in table-9.

(b) Switching signals for **s = 3000.0 v/μs, dv = 1.0 volt, Vm = 5.0 volt** and **fm = 50.0 Hz** are shown in figure 4.5. Here the number of off and on times (N) are 13 having the timing instants as given in table-10.

(c) Switching signals for **s = 3500.0 v/µs, dv = 1.0 volt, Vm = 5.0 volt** and **fm = 50.0 Hz** are shown in figure 4.6. Here the number of off and on times (N) are 17 having the timing instants as given in table-11.

It is evident from the above waveshapes that with slope variation of modulator's integrator, number of switching instants increases with the increase in slope. This would cause the spectra of the output waveform to have components of carrier frequency at higher values with no significant change of fundamental voltage as the slope of the switching waveshapes do not change significantly.

4.2.3 SWITCHING SIGNALS FOR REFERENCE WAVE'S MAGNITUDE VARIATION

The following waveforms of figures 4.7 to 4.9 show the effect of variation of the magnitude of the reference signal on the switching waveforms generated by the microcomputer.

(a) Switching signals for **s = 2500.0 v/µs, dv = 1.0 volt, Vm = 4.0 volt** and **fm = 50.0 Hz** are shown in figure 4.7. Here the number of off and on times (N) are 11 having the timing instants as given in table-12.

(b) Switching signals for **s = 2500.0 v/µs, dv = 1.0 volt, Vm = 5.0 volt** and **fm = 50.0 Hz** are shown in figure 4.8. Here the number of off and on times (N) are 11 having the timing instants as given in table-6.

(c) Switching signals for **s = 2500.0 v/µs, dv = 1.0 volt, Vm = 6.0 volt** and **fm = 50.0 Hz** are shown in figure 4.9. Here the number of off and on times (N) are 9 having the timing instants as given in table-13.

As the magnitude of reference wave is increased, number of switching instants decrease with increase in the pulse widths. This would increase the available fundamental voltage of the inverter with induction of low order harmonics.

4.2.4 SWITCHING SIGNALS FOR WINDOW WIDTH VARIATION

The following waveforms of figures 4.10 to 4.12 show the effect of variation of the window width on the switching waveforms generated by microcomputer.

(a) Switching signals for **s = 2500.0 v/μs, dv = 0.5 volt, Vm = 5.0 volt** and **fm = 50.0 Hz** are shown in figure 4.10. Here the number of off and on times (N) are 21 having the timing instants as given in table-14.

(b) Switching signals for **s = 2500.0 v/μs, dv = 0.75 volt, Vm = 5.0 volt** and **fm = 50.0 Hz** are shown in figure 4.11. Here the number of off and on times (N) are 14 having the timing instants as given in table-15.

(c) Switching signals for **s = 2500.0 v/μs, dv = 1.25 volt, Vm = 5.0 volt** and **fm = 50.0 Hz** are shown in figure 4.12. Here the number of off and on times (N) are 9 having the timing instants as given in table-16.

The number of switching instants of the modulated wave decreases with increase in the window width of the modulator. This causes increase in available fundamental voltage with increase of window width. This particular phenomenon can be utilized when inverter operation frequency is increased. As inverter operation frequency is increased, output voltage waveforms have lower number of modulated pulses and low order harmonics are introduced. The pulse numbers per cycle can be increased by decreasing the window width and thereby maintaining the same spectra as obtained in low frequency operation of the inverter.

4.3 INVERTER OPERATION

Practical interface between the pulses available at the parallel port of microcomputer and the MOS-inverter requires the design steps as mentioned in the following section. The design and implementation of MOSFET-inveter and the inverter waveforms corresponding to the inverter switching signals obtained at the parallel port of the microcomputer is described in the following section.

4.3.1 DESIGN AND IMPLEMENTATION OF MOSFET-INVERTER

4.3.1.1 COMPUTER INTERFACE

(a) 14-pin Hex inverter / buffers

This is the first stage of interfacing of the six PWM switching pulse-patterns from the parallel port of the microcomputer. Two 14 -pin DIP Hex-inverter/buffer ECG7406 having 6 inverters in each chip are used at this stage. As the input impedance of an inverter /buffer is very high ($\cong$ 100 MΩ) this stage is used for impedance matching between the computer I/O card and the subsequent stages. Each of the switching channels needs amplifier circuit of BD243 transistor via two of the inverting buffers of these chips. As the programs PTDMWG.CPP generates the pulses as shown in figure 3.1, the first two inverter buffers the switching pulses of pin-2, whereas, the next two inverter buffers the switching pulses of pin-7 and so on in the sequence of the switching pulses of pin-3, pin-5, pin-4, and pin-6. The switching pulses of channel-2 is 60^0 out of phase with that of the channel-1 and so on for the other channels as in figure 4.13. The biasing voltage for the hex-buffer ECG7406 is +5 volt and is given from the SMPS of the computer.

(b) Common-collector npn transistor amplifiers and the opto-couplers

This stage contains six fixed bias npn BD243 transistors connected as common-collector configuration and six opto-couplers as shown in figure 4.14. The signals at the output of buffer-stage are not strong enough to drive the LED's of the opto-coupler stage properly. The purpose of the amplifier stage is to amplify the output signals of buffer-stage available at channel-1 through channel-6 of figure 4.13 to have amplified emitter-currents. The LED's of the opto-couplers are connected in series with 100Ω resistances. The opto-couplers numbered 4N25 309Atk4 provide opto-isolation between the microcomputer and the power MOS-inverter circuit. Opto-isolation prevent any current flow between the two systems by ground isolation. The opto-couplers are in a package that contains both an infrared LED and a photodetector npn transistor. The wave-length response of each device is tailored to be as identical as possible to permit the highest possible coupling. There is a transparent insulation cap between each set of elements embedded in the structure to permit the passage of light. They are designed with response times so small that they can be used to transmit data in the megahertz range. The biasing of the photo transistors embedded inside the opto-couplers are given from the six-channel pulse amplifier used in the pulse amplifier stage. The output switching signals from the opto-couplers are received across 4.7 kΩ resistances with the common ground of the six-channel pulse amplifier named GND_2 in the figure 4.14. The 100Ω resistances connected in series with the LED's of the opto-couplers are current limiting resistances.

4.3.1.2 PULSE AMPLIFIER

This stage contains a six-channel pulse amplifier module with logic power supply to strengthen the switching signals of the opto-coupler outputs.

They amplify the switching signals available at channel-1′ through channel-6′ upto about +15 volt (peak) which is enough to trigger the switching devices (MOSFET's) of the three-phase inverter. The amplified switching signal outputs are now available at channel-1″ through channel-6″ as in figure 4.15.

4.3.1.3 BASE ISOLATION OF THE MOSFET'S

This stage contains six-pulse transformer (1:1) to imply the switching pulses across the gate to source terminal of the power MOSFET's. The use of transformers makes all the gate to source terminals of each MOSFET's isolated from one another as shown in figure 4.16 to be dedicated to receive its own pulse pattern. As a result the ground problem of biasing the power MOSFET's of the three-phase inverter is solved by using only one dc. power supply which is our dc. input for the inverter to be converted to ac. output at the terminals of three-phase resistive load. In figure 4.16 G_1 indicates the gate terminal of $MOSFET_1$, S_1 indicates the source terminal of the same MOS and so on for the other MOSFET's.

4.3.1.4 INVERTER CIRCUIT WITH POWER MOSFET'S

Three-phase inverter circuit consisting of six MOSFET's IRF840 as switching devices and are arranged as shown in figure 4.17 with the three-phase Y-connected resistive load. The ratings of the resistances used as three-phase load is 100Ω, 0.5 watt. IRF840's are n-channel enhancement mode MOSFET's which can sustain a maximum drain to source voltage of 500 volt. Power MOSFET's are chosen as switching device as their switching times are of the order of nanoseconds. They don't have the problems of second breakdown phenomena as do BJT's do. The other specialties of power MOSFET's to be used in inverters are temperature stability of the electrical parameters and ease of paralleling.

The schematic block diagram of the approached on-line control system is outlined in figure 4.18.

4.3.2 OUTPUT WAVEFORMS OF THE INVERTER

The three-phase inverter is operated by computer generated delta modulated switching signals to verify experimentally the success of implementation. The inverter has been successfully run in the laboratory under various operating parameters. Typical waveforms of the inverter with resistive load controlled on-line by a microcomputer generated signals are provided in figure 4.19 through 4.22. The waveforms are presented as illustration of on-line inverter controllability by microcomputer by delta modulated switching for various parameter variations as follows,

(a) Line to neutral voltage for **S = 2500 v/μs, Vm = 4.0 volt, dv = 0.5 volt,** and **fm = 50 Hz** is shown in figure 4.19.

(b) Line to neutral voltage for **S = 2500 v/μs, Vm = 4.0 volt, dv = 0.5 volt,** and **fm = 25 Hz** is shown in figure 4.20.

(c) Line to neutral voltage for **S = 2500 v/μs, Vm = 4.0 volt, dv = 0.5 volt,** and **fm = 40 Hz** is shown in figure 4.21.

(d) Line to neutral voltage for **S = 3000 v/μs, Vm = 2.0 volt, dv = 0.3 volt,** and **fm = 70 Hz** is shown in figure 4.22.

Switching Instants	Seconds
t_0	0.000000
t_1	0.000609
t_2	0.001773
t_3	0.002387
t_4	0.003518
t_5	0.004149
t_6	0.005215
t_7	0.005874
t_8	0.006861
t_9	0.007558
t_{10}	0.008464
t_{11}	0.009208
t_{12}	0.010041
t_{13}	0.010842
t_{14}	0.011611
t_{15}	0.012479
t_{16}	0.013194
t_{17}	0.014136
t_{18}	0.014808
t_{19}	0.015828
t_{20}	0.016468
t_{21}	0.017560
t_{22}	0.018179
t_{23}	0.019324
t_{24}	0.019934
t_{25}	0.021100

Table 6: Timing instants of the PWM waveform with S= 2500.0 v/μs, dv= 1.0 volt, Vm= 5.0 volt, and fm= 25.0 Hz.

Switching Instants	Seconds
t_0	0.000000
t_1	0.000491
t_2	0.002601
t_3	0.003161
t_4	0.004379
t_5	0.005092
t_6	0.005878
t_7	0.006843
t_8	0.007438
t_9	0.008855
t_{10}	0.009359
t_{11}	0.011440

Table 7: Timing instants of the PWM waveform with S= 2500.0 v/μs, dv= 1.0 volt, Vm= 5.0 volt, and fm= 50.0 Hz.

Switching Instants	Seconds
t_0	0.000000
t_1	0.000440
t_2	0.004515
t_3	0.005540
t_4	0.006066
t_5	0.008309

Table 8: Timing instants of the PWM waveform with S= 2500.0 v/μs, dv= 1.0 volt, Vm= 5.0 volt, and fm= 65.0 Hz.

Switching Instants	Seconds
t_0	0.000000
t_1	0.000438
t_2	0.001823
t_3	0.002286
t_4	0.003386
t_5	0.003918
t_6	0.004725
t_7	0.005363
t_8	0.005992
t_9	0.006787
t_{10}	0.007308
t_{11}	0.008329
t_{12}	0.008788
t_{13}	0.010086

Table 10: Timing instants of the PWM waveform with S= 3000.0 v/μs, dv= 1.0 volt, Vm= 5.0 volt, and fm= 50.0 Hz.

Switching Instants	Seconds
t_0	0.000000
t_1	0.000560
t_2	0.004971
t_3	0.005963
t_4	0.006774
t_5	0.008484
t_6	0.009073
t_7	0.013111

Table 9 : Timing instants of the PWM waveform with S= 2000.0 v/μs, dv= 1.0 volt, Vm= 5.0 volt, and fm= 50.0 Hz.

Switching Instants	Seconds
t_0	0.000000
t_1	0.000394
t_2	0.001425
t_3	0.001832
t_4	0.002748
t_5	0.003190
t_6	0.003944
t_7	0.004442
t_8	0.005062
t_9	0.005639
t_{10}	0.006163
t_{11}	0.006844
t_{12}	0.007303
t_{13}	0.008116
t_{14}	0.008532
t_{15}	0.009487
t_{16}	0.009883
t_{17}	0.010920

Table 11: Timing instants of the PWM waveform with S= 3500.0 v/μs, dv= 1.0 volt, Vm= 5.0 volt, and fm= 50.0 Hz.

Switching Instants	Seconds
t_0	0.000000
t_1	0.000456
t_2	0.003609
t_3	0.004215
t_4	0.005196
t_5	0.006035
t_6	0.006679
t_7	0.007969
t_8	0.008467
t_9	0.010878

Table 13: Timing instants of the PWM waveform with S= 2500.0 v/μs, dv= 1.0 volt, Vm= 6.0 volt, and fm= 50.0 Hz.

Switching Instants	Seconds
t_0	0.000000
t_1	0.000532
t_2	0.002119
t_3	0.002692
t_4	0.003892
t_5	0.004575
t_6	0.005432
t_7	0.006291
t_8	0.006958
t_9	0.008085
t_{10}	0.008651
t_{11}	0.010127

Table 12: Timing instants of the PWM waveform with S= 2500.0 v/μs, dv= 1.0 volt, Vm= 4.0 volt, and fm= 50.0 Hz.

Switching Instants	Seconds
t_0	0.000000
t_1	0.000368
t_2	0.001965
t_3	0.002361
t_4	0.003474
t_5	0.003945
t_6	0.004699
t_7	0.005265
t_8	0.005836
t_9	0.006552
t_{10}	0.007016
t_{11}	0.007971
t_{12}	0.008370
t_{13}	0.009696
t_{14}	0.010065

Table 15: Timing instants of the PWM waveform with S= 2500.0 v/μs, dv= 0.75 volt, Vm= 5.0 volt, and fm= 50.0 Hz.

Switching Instants	Seconds
t_0	0.000000
t_1	0.000246
t_2	0.001316
t_3	0.001570
t_4	0.002466
t_5	0.002742
t_6	0.003419
t_7	0.003727
t_8	0.004257
t_9	0.004606
t_{10}	0.005039
t_{11}	0.005443
t_{12}	0.005811
t_{13}	0.006286
t_{14}	0.006607
t_{15}	0.007181
t_{16}	0.007467
t_{17}	0.008181
t_{18}	0.008443
t_{19}	0.009341
t_{20}	0.009589
t_{21}	0.010650

Table 14: Timing instants of the PWM waveform with S= 2500.0 v/μs, dv= 0.5 volt, Vm= 5.0 volt, and fm= 50.0 Hz.

Switching Instants	Seconds
t_0	0.000000
t_1	0.000614
t_2	0.003223
t_3	0.003973
t_4	0.005222
t_5	0.006268
t_6	0.007072
t_7	0.008686
t_8	0.009321
t_9	0.011912

Table 16: Timing instants of the PWM waveform with S= 2500.0 v/μs, dv= 1.25 volt, Vm= 5.0 volt, and fm= 50.0 Hz.

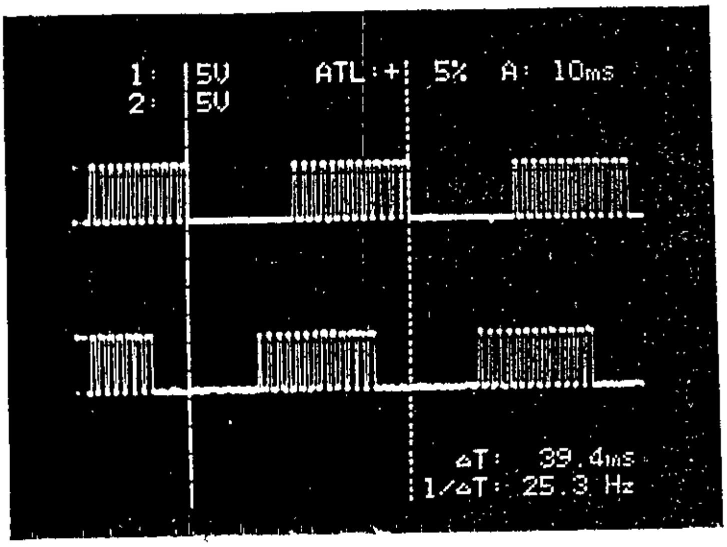


Figure 4.1 :

Switching signals for s = 2500.0 v/μs, dv = 1.0 volt, Vm = 5.0 volt and fm = 25.0 Hz.

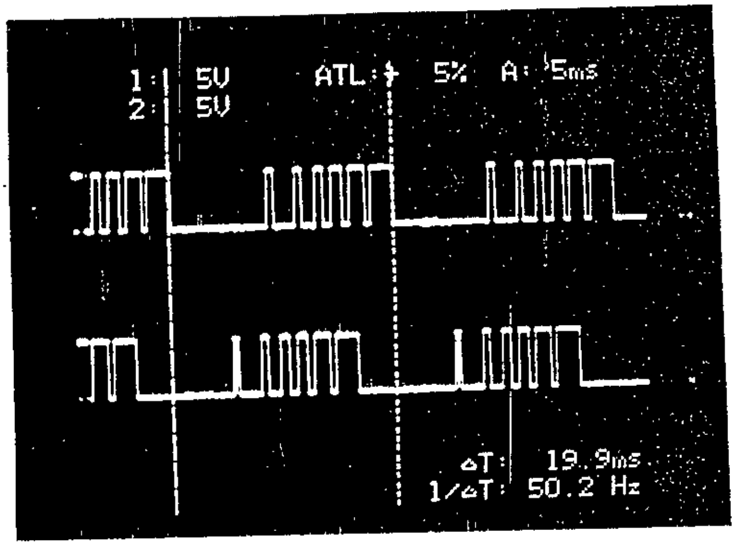


Figure 4.2 :

Switching signals for s = 2500.0 v/μs, dv = 1.0 volt, Vm = 5.0 volt and fm = 50.0 Hz.

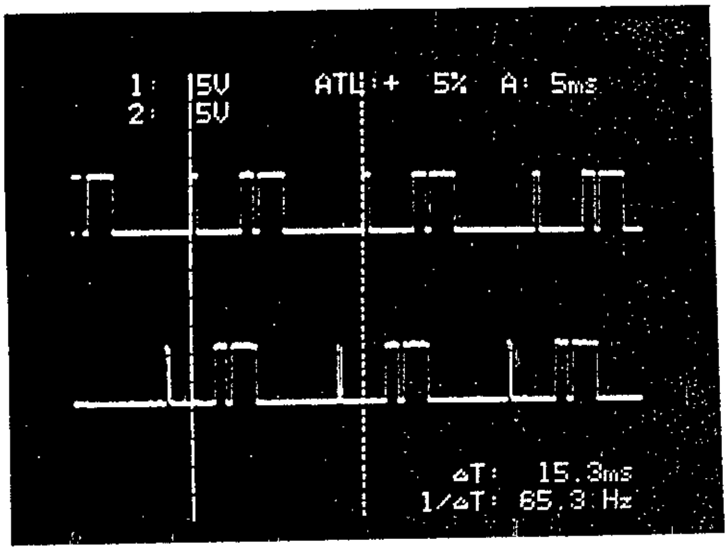


Figure 4.3 :

Switching signals for s = 2500.0 v/μs, dv = 1.0 volt, Vm = 5.0 volt and fm = 65.0 Hz.

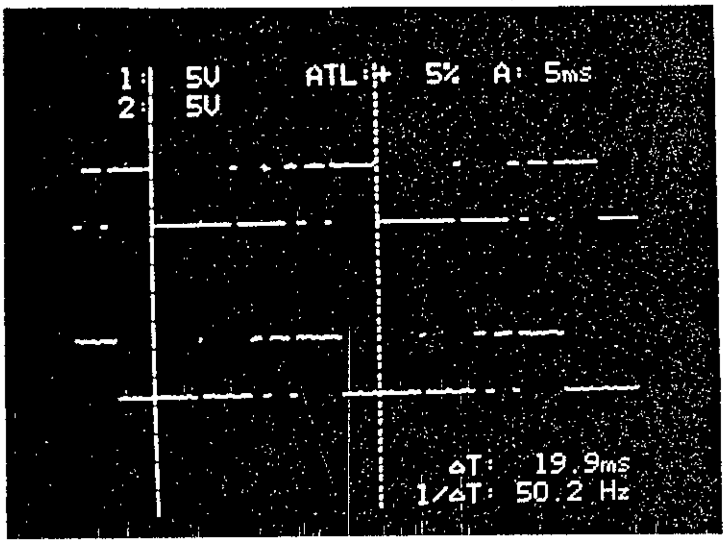


Figure 4.4 :

Switching signals for s = 2000.0 v/μs, dv = 1.0 volt, Vm = 5.0 volt and fm = 50.0 Hz.

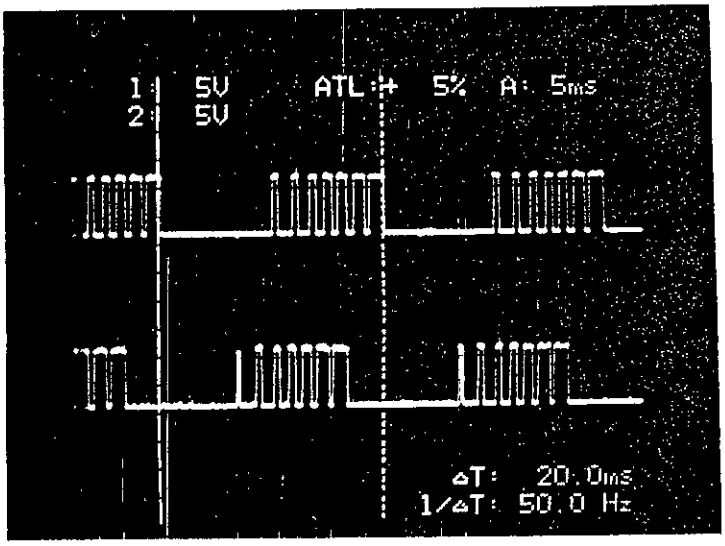


Figure 4.5 :

Switching signals for s = 3000.0 v/μs, dv = 1.0 volt, Vm = 5.0 volt and fm = 50.0 Hz.

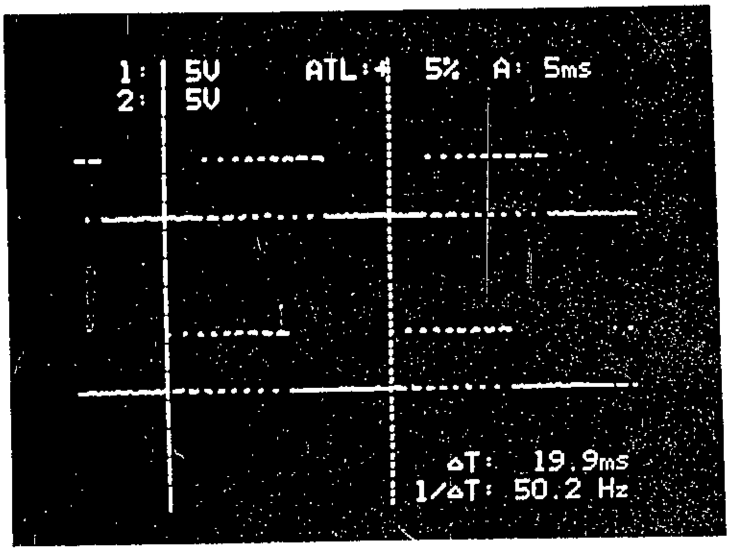


Figure 4.6 :

Switching signals for s = 3500.0 v/μs, dv = 1.0 volt, Vm = 5.0 volt and fm = 50.0 Hz.

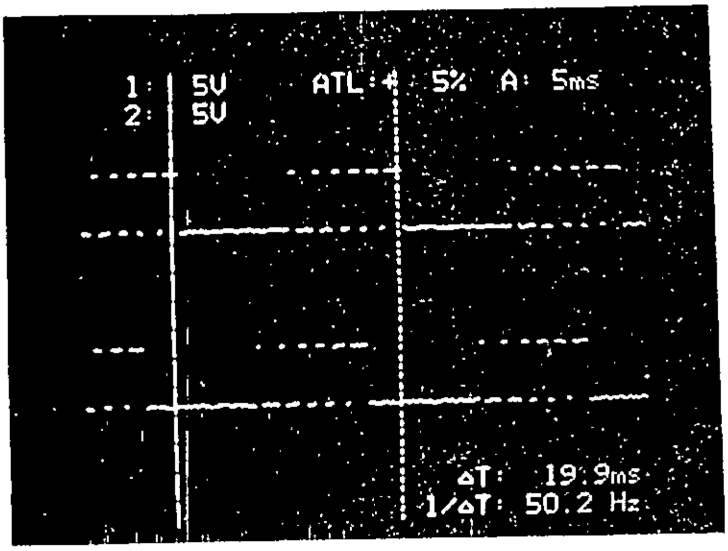


Figure 4.7 :

Switching signals for s = 2500.0 v/μs, dv = 1.0 volt, Vm = 4.0 volt and fm = 50.0 Hz.

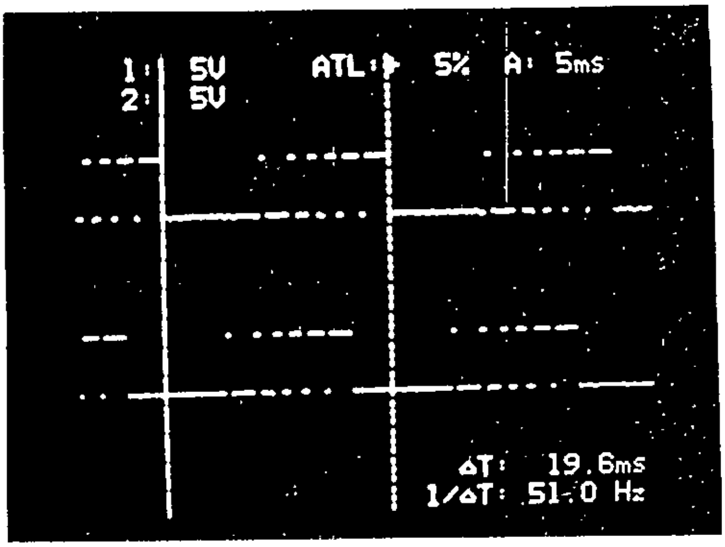


Figure 4.8 :

Switching signals for s = 2500.0 v/μs, dv = 1.0 volt, Vm = 5.0 volt and fm = 50.0 Hz.

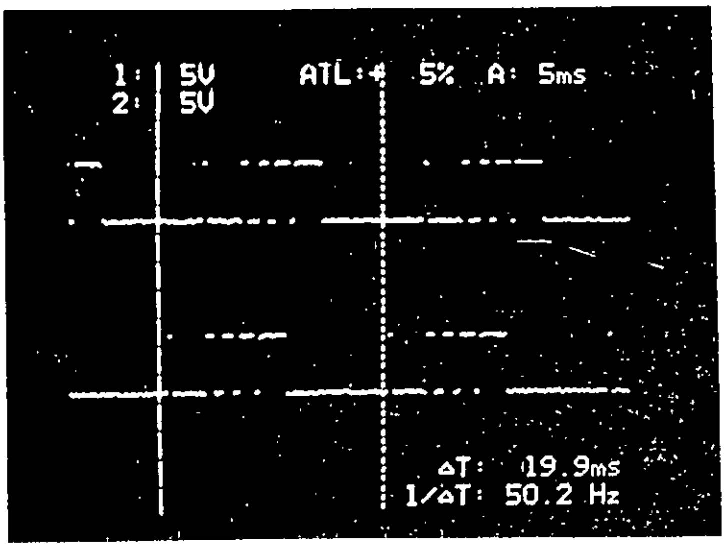


Figure 4.9 :

Switching signals for s = 2500.0 v/μs, dv = 1.0 volt, Vm = 6.0 volt and fm = 50.0 Hz.

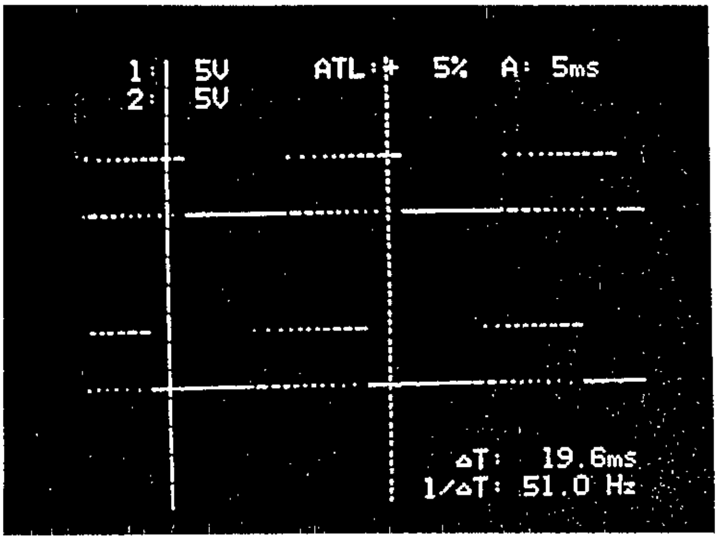


Figure 4.10 :

Switching signals for s = 2500.0 v/μs, dv = 0.5 volt, Vm = 5.0 volt and fm = 50.0 Hz.

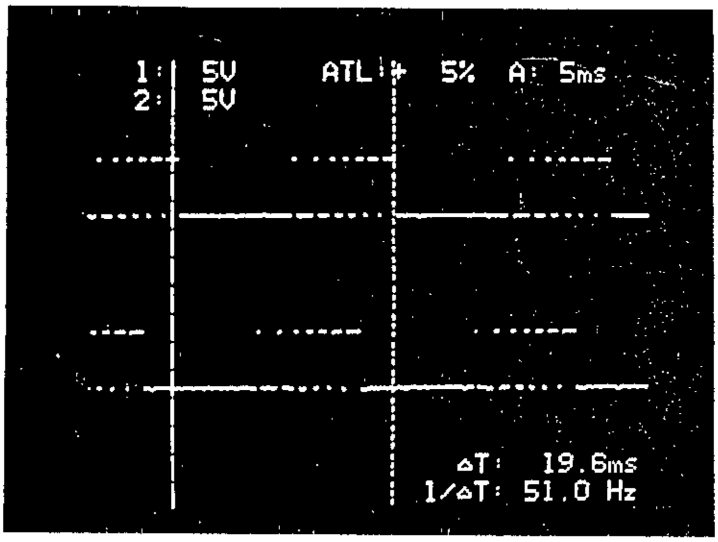


Figure 4.11 :

Switching signals for s = 2500.0 v/μs, dv = 0.75 volt, Vm = 5.0 volt and fm = 50.0 Hz.

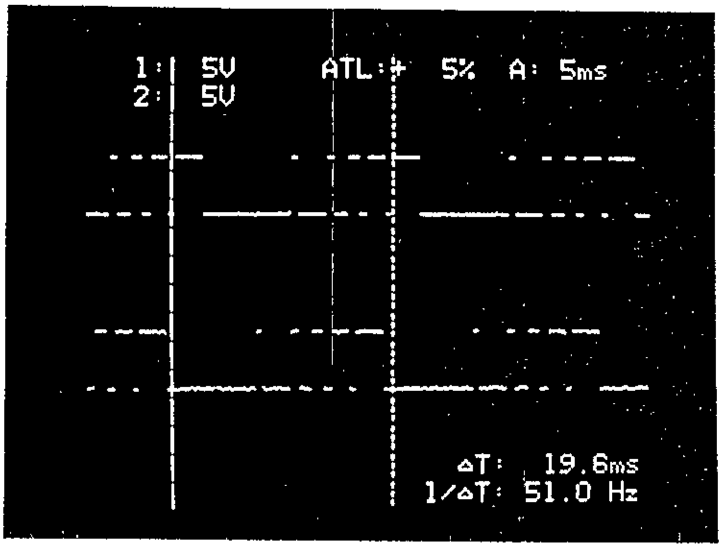


Figure 4.12 :

Switching signals for s = 2500.0 v/μs, dv = 1.25 volt, Vm = 5.0 volt and fm = 50.0 Hz.

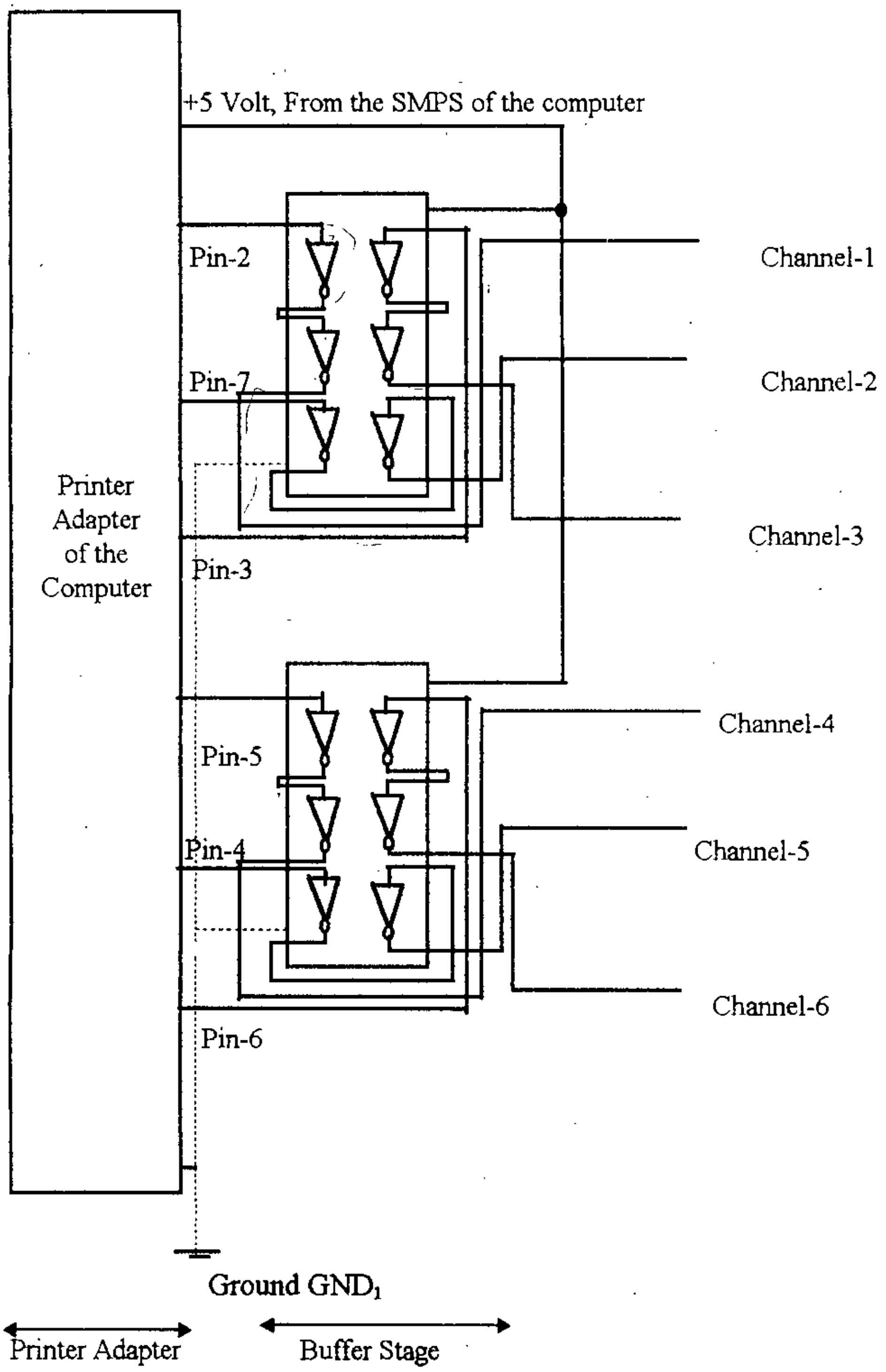


Figure 4.13 : Printer adapter to buffer stage.

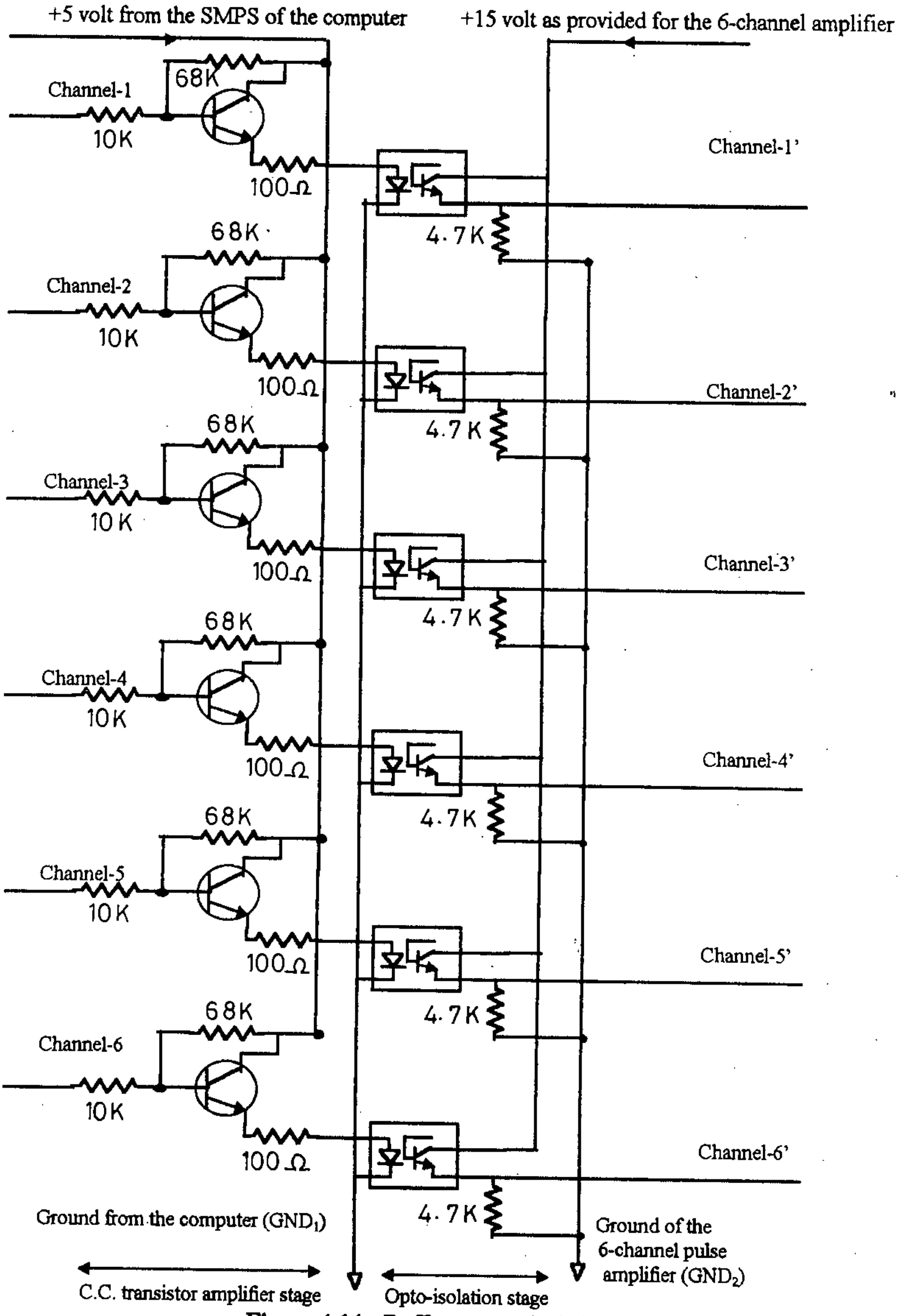


Figure 4.14 : Buffer to opto-isolation stage.

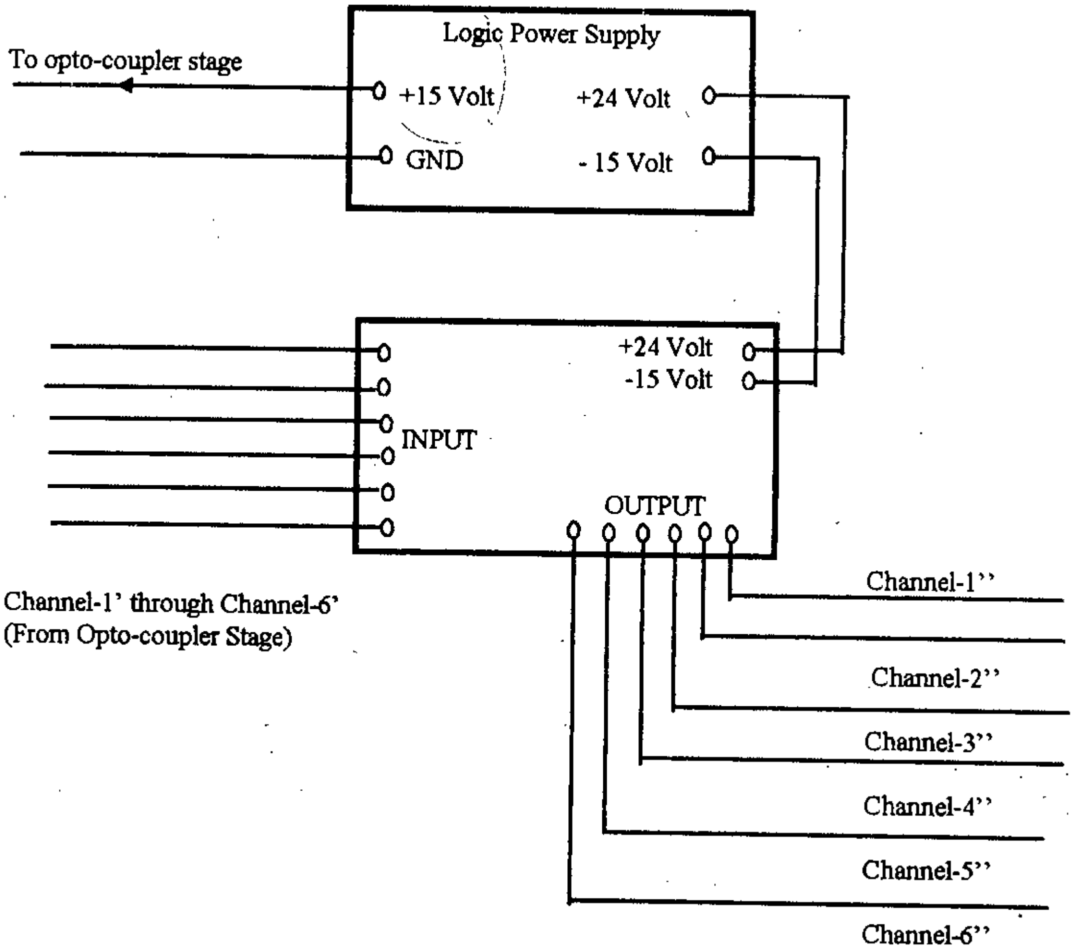


Figure 4.15 : Six-channel pulse amplifier stage.

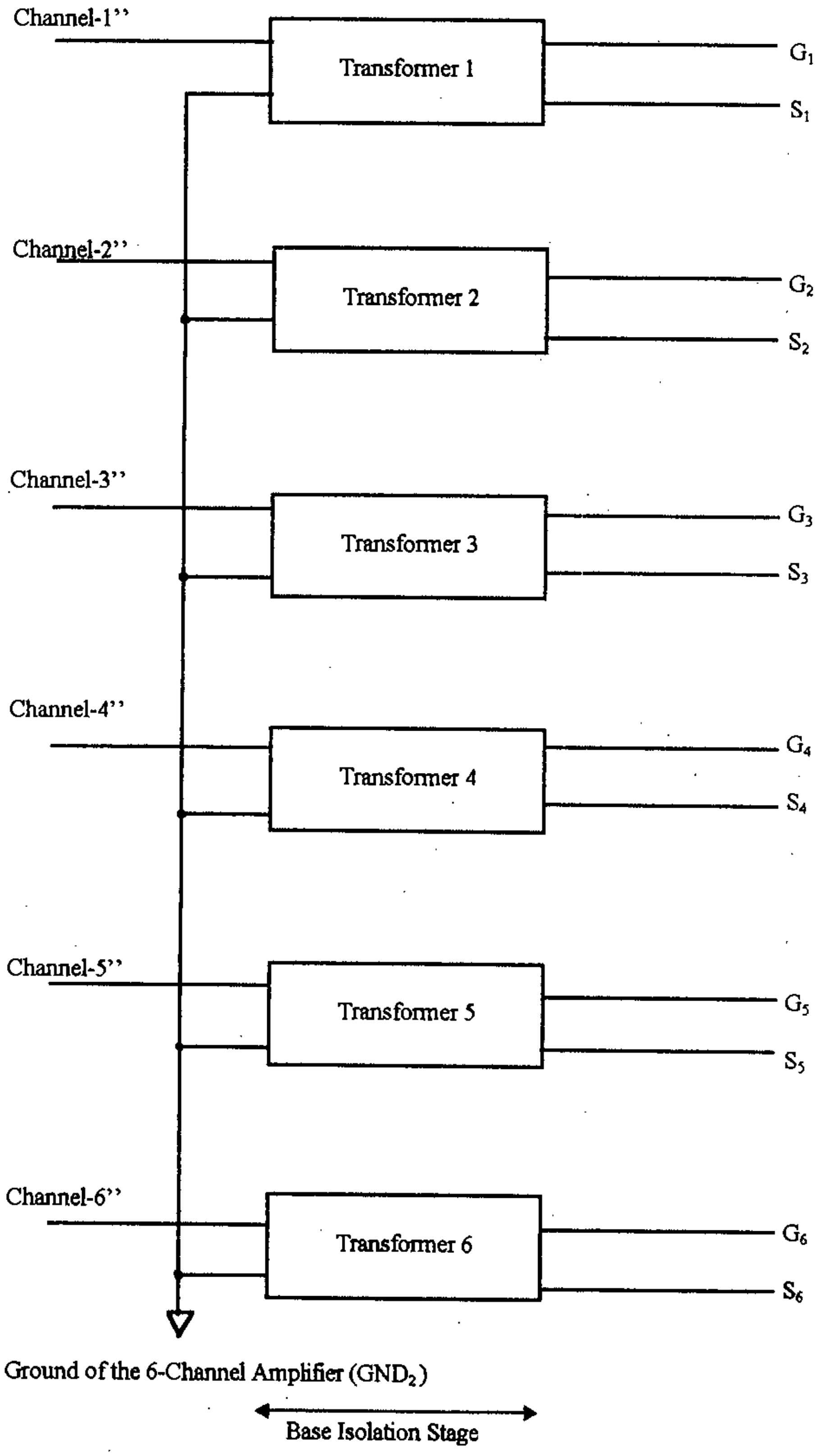


Figure 4.16 : Pulse amplifier to base isolation stage.

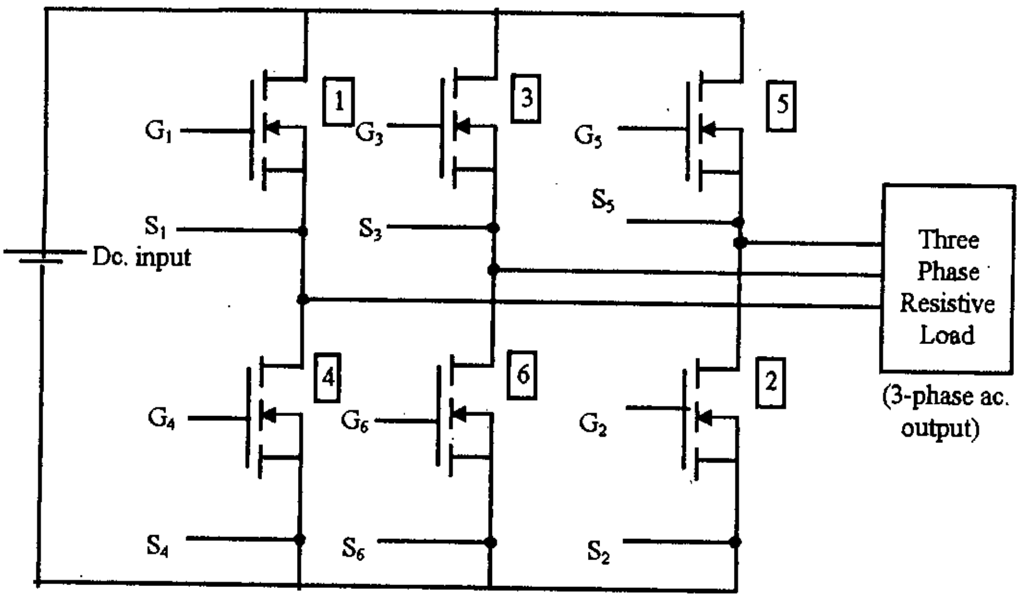


Figure 4.17 : Inverter circuit with Power MOSFET.

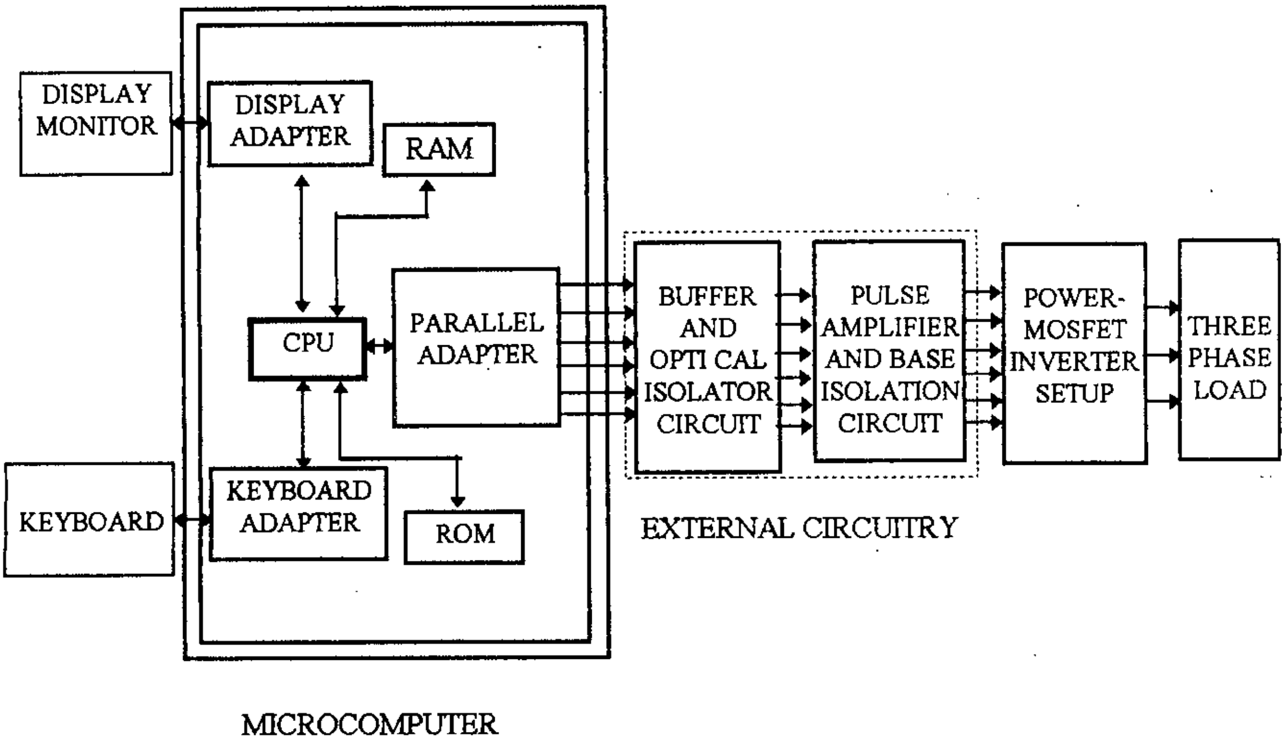


Figure 4.18 :
Schematic block diagram for on-line microcomputer control of delta modulated three-phase inverters.

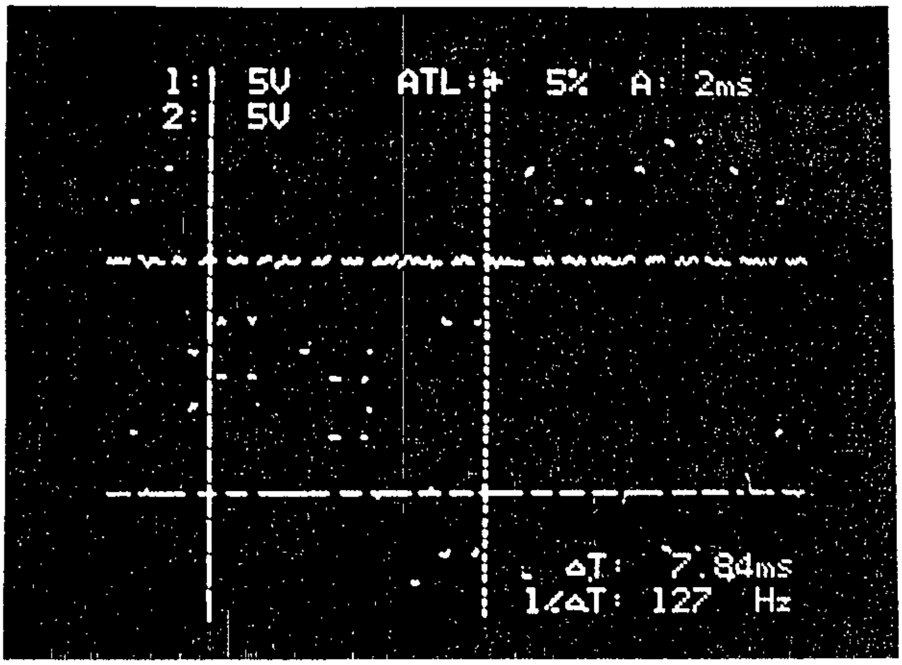


Figure 4.19 :

Line to neutral voltage for S = 2500 v/μs, Vm = 4.0 volt, dv = 0.5 volt, and fm=50.0Hz.

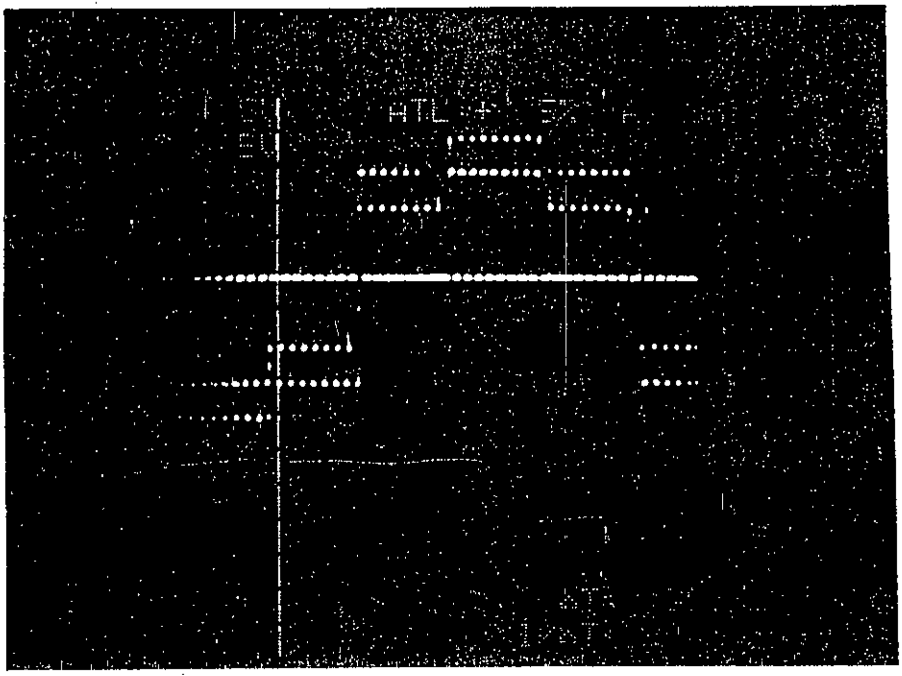


Figure 4.20 :

Line to neutral voltage for S = 2500 v/μs, Vm = 4.0 volt, dv = 0.5 volt, and fm = 25.0 Hz.

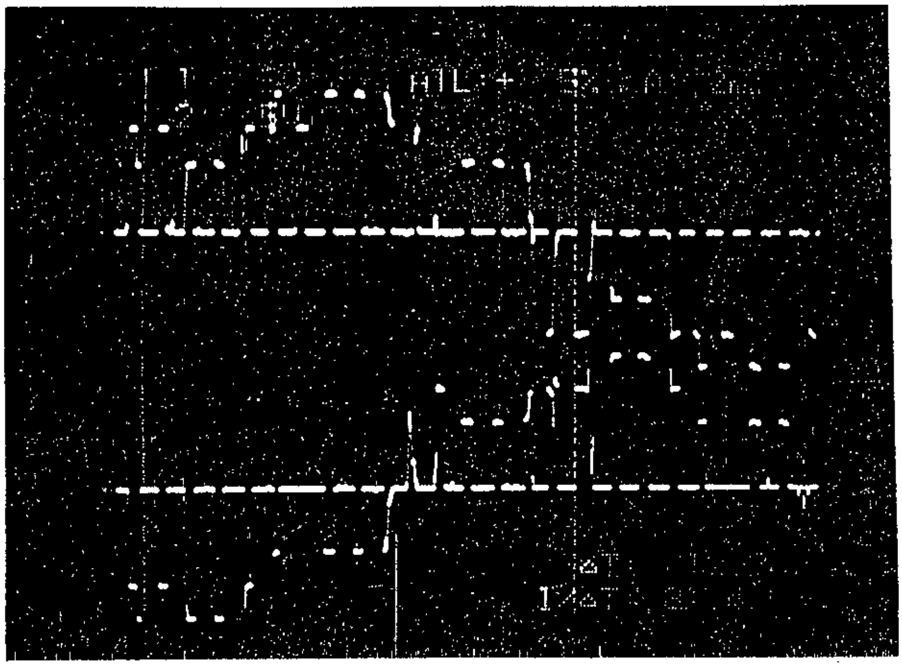


Figure 4.21 :

Line to neutral voltage for S = 2500 v/μs, Vm = 4.0 volt, dv = 0.5 volt, and fm = 40.0 Hz.

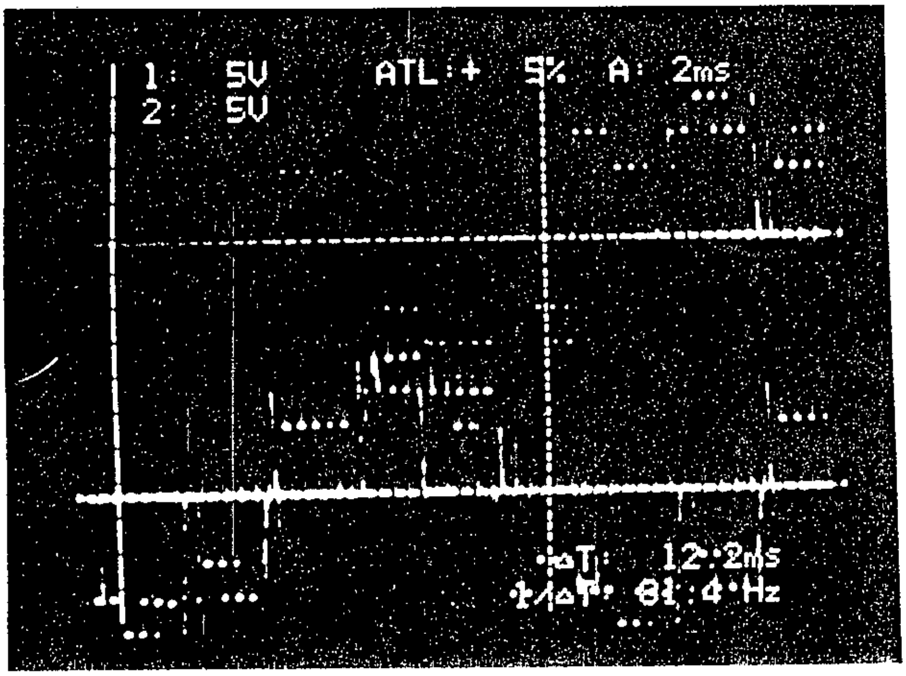


Figure 4.22 :

Line to neutral voltage for S = 3000 v/μs, Vm = 2.0 volt, dv = 0.3 volt, and fm = 70.0 Hz.

CHAPTER-FIVE

SUMMARY AND CONCLUSION

- □ INTRODUCTION
- □ CONCLUSION ON RESULTS
- □ LIMITATIONS
- □ RECOMMENDATION FOR FUTURE WORK

5.1 INTRODUCTION

This chapter concludes the work of thesis on-line microcomputer control of DPWM switching technique. The practical implementation and observation has been carried out successfully in the laboratory for various settings of control parameters. A brief discussion on the limitations of software design and practical implementation of the thesis work is provided in this chapter. It also outlines some of the possible future research on this work.

5.2 CONCLUSION ON RESULTS

On-line microcomputer control of delta modulated inverter has been practically carried out in the laboratory with reasonable acceptance. The practical implementation of delta modulated power MOS-inverter as in figure 4.18 of chapter four has been run with an INTEL 80386 DX-2 microcomputer together with proper interfacing and isolation circuits as explained in chapter-four. The waveshapes of six-channel switching pulse-pattern available at the parallel-port of the computer and at the output waveforms of the three-phase inverter with resistive load have been observed. The frequency and timing instants of each and every switching pulse-patterns have been verified with the guide menu of the same oscilloscope. The photographs of the output and input waveforms with a set of parameter variations are shown in figures 4.1 through 4.12 and figures 4.19 through 4.22. The timing instants for each switching waveform with a particular set of parameters have been calculated with the execution of the subfunction '*CalTime( )*' of the program PTDMWG.CPP which are given in table-6 through table-16 of chapter-four. The comparison of the timing instants as obtained in those of tables with those as observed practically by the storage oscilloscope show that they are in close agreement. The line to neutral output voltage waveforms for some set of DPWM parameter variations as given in figures 4.19 through figure 4.22, also provide satisfactory results. The output waveforms have been observed as sampled sine-wave following the DPWM switching signals applied at the interfacing stages. The main success of this implementation complies with the on-line control of DPWM parameters from the keyboard operation of the computer.

The program execution time which is important for on-line inverter switching operation is given in table-3 of chapter-two. Computer requirements as explained in section 2.6.2 implies that any AT-machine with inbuilt math-coprocessor will be enough for this type of on-line control with certain parameter limitation. By changing the number YY which is defined at the beginning of the program, to select *'del'* the resolution of the pulse-pattern to be generated can be selected in the program algorithm. In the proposed program the total number of iteration YY has been selected as 80 to have optimum resolution of the generated switching pulses. With YY=80 the time span of each segment *'del'* is about 0.000034 second, whereas, the minimum switching time duration for the observed parameters is approximately 0.0001 second.

The program algorithm is so designed that any faulty keystroke other than the selected ones will not hamper the generation of the switching pulses. Any combination of integer and floating point parameters of the modulator can be selected in the implemented method.

5.3 LIMITATIONS

(A) LIMITATIONS OF THE SOFTWARE PROPOSED

(1) Execution time of any for-loop having a fixed iteration number may differ, although in small amount, due to caching at different times using the

same CPU. In the program PTDMWG.CPP the duration of a small time segment to be scanned is defined by *'del'* which is determined by the execution of the subfunction *'StartUp( )'* as mentioned before. In the subfunction *'StartUp( )'* we have recorded the execution time of a dummy for-loop having YY number of iterations and this iteration number is defined at the starting of the program as *#define YY*. In the subfunction *'GenPulse ( )'* to generate the desired switching pulse pattern every segment of pulse pattern is to be delayed for a time duration of *'del'*. To have this we have used the same dummy for-loop with iteration number YY in the subfunction *'GenPulse ( )'* as in the subfunction *'StartUp ( )'* assuming that the execution time will be equal to *'del'* in two cases. But the execution time may not be same due to caching of for-loop. This give rise to error with small iteration number YY.

(2) The stop-watch named *Timer1* used to record the execution time of the for-loop in the subfunction *'StartUp ( )'* will need its own command execution time embedded in the library function *timer.cpp and timer.h* . This extra time adds some error to the segment duration time *'del'* .

(3) Resolution of the generated switching pulse-pattern depends on the selection of time duration *'del'* of each small segment to be scanned. It is obvious that the resolution will be better with a short time duration *'del'* i.e. with a small iteration number YY as defined at the starting of the program. But a small iteration number YY will intensify the error as mentioned in the item (1) of this section. So we have to select YY which will give optimum resolution of the generated pulse-pattern.

(4) In the program PTDMWG.CPP, N is the number of timing instants of the switching pulse pattern in a half cycle which may be an even or an odd

number. The program algorithm is such that an odd N will make the last timing instant unused.

(5) With this program algorithm used in this thesis we can't choose any value of the PWM switching parameters which will give abnormal program termination and floating point error. The reason is due to the fact that, with some set of parameters the modulator may be in slope over load mode of operation. The program in this thesis does not account for operation beyond PWM operation.

(B) LIMITATION OF THE PRACTICAL IMPLEMENTATION

(1) The transformers used in the base isolation stage were of iron core which suffers saturation problems and causes power loss in terms of heat at high frequencies. The use of ferrite core transformers will reduce this problem.

(2) MOSFET's have the problems of electrostatic discharge and require special care in handling and it is relatively difficult to protect them under short-circuited fault.

5.4 RECOMMENDATION FOR FUTURE WORKS

In the future, algorithm may be developed to eliminate the problems relating to stop-watch, caching of for-loop, and the other problems as mentioned in section 5.3. The algorithm may also include such provisions that, with a particular parameter change, the computer will choose the other parameters for best operating condition of the drive system connected to the inverter. The developed on-line microcomputer control system for delta modulated inverter is an open-loop control system where the parameter change is carried out manually by key-depression. Any future work may include designing the same system with feedback control where the switching parameters will chang automatically sensing the output performances of the drive system to provide the best operating condition. For this feedback control system, we can use the input pins of the printer adapter which will receive information in digital form coming from the output drive system sensed by appropriate transducers. Provision may be provided for displaying the harmonic analysis waveforms in the monitor from where the user can take decision of changing any of the switching parameters. Study and implementation of DM microcomputer control in the operation of other types of static converters may be undertaken in future research work. Also, unsymmetrical slopes for positive and negative portion of the estimated signals can be implemented in future works to increase the voltage availability at the output of static converter. A provision may be incorporated in future works to distribute the harmonics due to ripple frequencies over the total frequency spectra judiciously, so as to reduce acoustic noise created by these harmonics keeping the low order harmonic losses below acceptable range.

REFERENCES

[1] Md. Bashir Uddin, " *Stability Analysis of PWM Inverter-fed Synchronous Machine.*" Ph.D. Thesis submitted for defense to the Department of EEE, BUET.

[2] M.A. Choudhury, Md. Bashir Uddin, S. Uddin and M.A. Rahman, "Analysis of delta PWM cycloconverters", accepted for publication in IEEE Transaction on Industrial Electronics.

[3] M. A. Choudhury, Md. Bashir Uddin, M. A. R. Bhuyia, and M. A. Rahman, "Simplified Analysis of PWM Inverter Waveforms.", submitted *for publication in Proc. of IEE, Part B, 1995.*

[4] G. Olivier, V.R. Stefanovic, and G.E. April, "Microprocessor Controller for a Thyristor Converter with an Improved Power Factor.", *IEEE Transactions on Industrial Electronics and Control Instrumentation, vol.IECI-29, Aug: 1981, pp. 188-194.*

[5] S. R. Bowes and M. J. Mount, "Microprocessor Control of PWM Inverters.", *IEE proc., Vol. 128, Pt. B, No. 6, Nov. 1981, pp. 123-127.*

[6] F.C. Zach, R.Martinez, S.Keplinger, and A.Seiser, “Dynamically Optimal Switching Patterns for PWM Inverter Drives.”, *IEEE Transaction on Ind.Appl.Vol.21, No.4, Jul/Aug. 1985, pp. 320-326.*

[7] M. Varnovitsky, “A Microcomputer-based Control Signal Generator for Three Phase Switching PWM Inverter.”, *IEEE Trans. on Ind. Appl. Vol. 1A-19, No.2, March/April 1983, pp. 187-194.*

[8] M.H. Rashid, “Power Electronics Circuits Device and Applications.”, *Prentice Hall Inc., 1983, pp. 356-378, 484-487, 497-504.*

[9] D.A. Grant, M. Stevens, and J.A. Houldsworth, “The Effect of Word Length on The Harmonic Content of Microprocessor-based PWM Waveform Generators.”, *IEEE Trans. on Ind. Appl., Vol. IA-21, Jan./Feb. 1985, pp. 218-665.*

[10] S. Bolognani, G.S. Bhuja, and D. Longo, “Hardware and Performance Effective Microcomputer Control of a Three-phase PWM Inverter,”. *Int.Power Electronic Conf. Rec., Tokyo, 1983, pp. 360-371.*

[11] E. Dallago, D. Dontti, and P. Ferrari, “Application of Power MOSFETs in a Three Phase Inverter Controlled by Microprocessor.”, *Int. Power Electronic Conf. rec., Tokyo, 1983, pp. 1142-1149.*

[12] A. Bellini, C.D. Masro, G. Figalli, and G. Ulivi, “An Approach for The Implementation on a Microcomputer of The Control Circuit of Variable Frequency Three-phase Inverters.”, *IEEE / IAS annual meeting conf. rec.,1981, pp. 650-655.*

[13] T. Kataoka, K. Mizumachi, and S. Miyairi, "A Pulse Width Controlled AC-DC. Converter to Improve Power Factor and Waveform of AC. Line Current," *IEEE Trans. on Industry Application, Vol. IA-15, Nov. / Dec. 1979, pp. 670-675.*

[14] S.R. Doradla, C. Nayamani, and S. Sanyal, "A Sinusoidal Pulse Width Modulated Three-phase AC-DC. Converted Fed Dc. Motor Drive," *IEEE Trans. of Ind. Appli., Vol. IA-21, Nov. / Dec. 1985, pp. 1394-1408.*

[15] V.V. Athani and S.M. Deshpande, "Microprocessor Control of a Three-phase Inverter in Induction Motor Speed Control System.", *IEEE Trans. Ind. Electron. conf. rec., Vol. IECI-27, Nov. 1980, jpp. 291-298.*

[16] G. S. Bhuja and P. Fioni, "A Microcomputer- based, Quasi-continuous Output Controller for PWM Inverter.", *IEEE / IECI conf. rec., 1980, pp.107-111.*

[17] S.K. Tso and P.T.Ho, "Dedicated Microprocessor Scheme for Thyristor Phase Control of Multiphase Converters.", *IEE Proc. , Vol. 128, March 1981, pp. 231-239.*

[18] P.L.G. Malapelle and L.A.M. Mortarino, "Microprocessor- based Controller for AC / DC Converters.", *Microelectron power electron. electr. drives conf. rec., Darmstadt, Germany, 1982, pp. 163-170.*

[19] M.J. Case and P. Kulentic, "A Microprocessor Controller for The Cycloconverter.", *Microelectron. power electron. electr. drives conf. rec., Darmstadt, W. German, 1982, pp. 171-181.*

[20] R.G. Hoft, T. Khuwatsamrit, and R. Mc Laren, " Microprocessor Applications for Power Electronics in the North America.", *Microelectron. power electron. electr. drives conf. rec., Darmstadt, W.Germany, 1982, pp. 29-42.*

[21] A. Bohuss, P. Buzas, and K. Ganszky, "Microcomputer Controlled Inverter for Uniterruptible Power Supply," *Microelectron power electron.elect. drives conf. rec., Darmstadt, W. Germany, 1982, pp. 203-206.*

[22] E.A. Rothwell, "The Use of Microprocessors and Power Transistors in Modern Uniterruptible Power Supplies," *Int. power elec. and variable speed drives conf. rec.,London 1984, pp. 413-418.*

[23] T. Kutman, "Implementation of a Microprocessor- based Technique for Output Filter Optimization in UPS System,". *Int. power elec. and variable speed drives conf. rec., London, 1984, pp. 75-78.*

[24] C.D.M. Oates, "Optimal PWM on a Microcomputer," *Int. power elec. and variable speed drives conf. rec. London, 1984, pp. 341-344.*

[25] G.N. Acharya, U.M. Rao, W. Shephard, S.S. Shekhawat, and Y. M. Nag, "Microprocessor- based PWM Inverter Using Modified Regular Sampling,". *IEEE / IAS annual meeting conf. rec., 1984, pp. 541-552.*

[26] G. S. Bhuja and P. Florini, “Microprocessor Control of PWM Inverters.”, *IEEE Transactions on Industrial Electronics, Vol. IE-29, August 1982, pp.212-218.*

[27] S. Morinage, Y. Sugiura, N. Muto, H. Okuda, K. Nandoh, H. Fujii, and K. Yajjima, “Microprocessor Control System with I/O Processing Unit LSI for Motor Drive PWM Inverter.”, *IEEE Transactions on Industry Application, Vol. IA-20, Nov. / Dec. 1984, pp. 154-553..*

[28] S. J. Lukas, “Microprocessor Control of dc Drive.”, *IEEE / IAS Annual Meeting Conf. Rec., 1979, pp. 881-885.*

[29] K. Kakiyama, T. Ohmae, N. Azusawa and T. Koike, “Microprocessor Based Current and Current Rate Controllers for Speed Control in Industrial Drives.”, *U. S. Japan Seminar on Microprocessor Application in dc Motor Drive Conf. Rec.,1982, pp. 249-258.*

[30] S. Nonaka and H. Okada, “Methods to Control Pulse Width of three Pulse Inverters.”, *Journal of IEE, Japan, Vol. 86, July, 1972, pp. 71-79.*

[31] B. Mokrytzki, “Pulse Width Modulated Inverters for ac Motor Drives.”, *IEEE Trans. on IA, Vol.-IGA-3, November /December, 1967, pp. 493-503.*

[32] D.A. Grant and R. Seinder, “Ratio Changing in Pulse Width Modulated Inverters.”, *IEE Proc., Vol. 128, pt. B, No. 5, Sept., 1981, pp. 243-248.*

[33] F. Harashima and S. Kondo, "Microprocessor- Based Optimal speed Control System of Motor Drives.", *IEEE / IECI Conf. Rec., 1981, pp. 252-257.*

[34] U. Waschatz, " Adaptive Control of Electrical Drives Employing Microprocessor.", *Microprocessor, Power Electron, Electr. Drive Conf. Rec., Darmstadt, W. Germany, 1982, pp. 135-140.*

[35] W. J. O'Brien, "SILTRON Microprocessor Based Adjustable Speed Drives.", *Industrial Power System, June 1981, pp. 213-223.*

[36] Intel Microprocessor and Peripheral Handbook, 1983.

[37] F. Aldana, J. Piere and C.M. Penalver, "Microprocessor Control for Power Electronic Systems.", *Microelectron. Power Electron Electr. Drives Conf. Rec., Darmstadt, W.Germany, 1982, pp. 111-115.*

[38] R.G. Hoft, R.W. McLaren, R.L. Pimmel and K.P. Gokhale, "The Impact of Microelectronics and Microprocessors on Power Electronics and Variavle speed Drives.", *Int. Power Elec. and Variable speed Drives Conf. Rec., London, 1984, pp. 191-198.*

[39] S.B. Dewan and A. Mirbod, " Microprocessor- Based Optimum Control for four-Quadrant Chopper.", *IEEE Trans. on Ind. Appl., vol.-IA-17, Jan./Feb., 1981, pp. 34-40.*

[40] S.R. Bowes, " New Sinusoidal Pulse Width Modulated Inverters.", *IEE Proc. Vol. 122, pt. B, No. 11, 1975, pp. 1279-1285.*

[41] S.R. Bowes and M. J. Mount, "Microprocessor Control of PWM Inverters.", *IEE Proc. Vol. 128, pt. B, 1981, pp. 293-305.*

[42] S.R.Bowes and S.R. Clements, "Microprocessor-Based PWM Inverters." , *IEE Proc. Vol. 129, pt. B, No.1, January 1982, pp. 1-17.*

[43] S.R. Bowes and J.C. Clare, "PWM Inverter Drives.", *IEE Proc. Vol. 130, pt. B, No. 4, July 1983, pp. 229-240.*

[44] J. Zubeck, A. Abbodanti and C. J. Nordby, "Pulse- Width Modulated Inverter Motor Drives with Improved Modulation.", *IEEE Trans. on IA, Vol. IA-11, No. 6, 1975, pp. 695-703.*

[45] G.S. Bhuja and G.B. Indri, "Optimal Pulse Width Modulation for Feeding ac. Motors.", *IEEE Trans. on Ind. Appl., Vol. IA-13, 1977, pp. 38-39.*

[46] R.M. Green and J.T. Boys, "Implementation of Pulse Width Modulated Inverter Strategies.", *IEEE Trans. on Ind. Appl., Vol. IA-18, No. 2, March/April 1982, pp. 138-145.*

[47] F.D. Buck, P. Gistellinck and D.D. Backer, "Optimized PWM Inverter Driven Induction Motor Losses.", *IEE Proc. vol. 130, pt. B, No.5, 1983, pp. 310-320.*

[48] A. Pollman, "A Digital Pulse Width Modulator Employing Advanced Modulation Technique.", *IEEE Conference Record on IA, 1982, pp. 116-121.*

[49] P. D. Ziogas, "The Delta Modulation Technique in Static PWM Inverters.", *IEEE Trans. on Ind. Appl., No. 17, 1981, pp. 199-204.*

[50] A.R.D. Esmail, M. Rahman, M.A. Choudhury, "Analysis of delta modulated AC-DC converters.", IEEE Transaction on PE Vol-10, No. 4, July 1995, pp 494-503.

[51] M. A. Rahman, J.E. Quaicoe, A.R.D. Esmail, and M. A. Choudhury, "Delta Modulated Inverter for UPS.", *IEEE Intelec Conference Record 1986, pp. 445-450.*

[52] Andrew C. Wang and Seth R. Sanders, " Programmed PWM Waveform for Electromagnetic Interference Mitigation in DC-DC Converters," *IEEE Transactions on Power Electronics,vol.8, no.4, October 1993, pp.-596-605.*

[53] Paul M. Russo, " VLSI Impact on Microprocessor Evolution, Usage, and System Design.", *IEEE Trans. on Electron Devices, Vol. ED-27, Agug. 1980, pp. 1332-1341.*

[54] Pei-Chong Tang, Shui-Shong Lu, and Yung-Chum Wu, "Microprocessor - Based Design of a Firing Circuit for Three-Phase Full-wave Thyristor Dual Converter.", *IEEE Trans. on Ind. Elec., Vol. IE-29, No. 1, Feb., 1982, pp. 67-73.*

[55] W.L. Frederich, K.S. Swenson, " Microprocessor-Based Converter Firing Pulse Generator.", *IEEE/IAS Annual Meeting, 1982, pp. 439-445.*

[56] Ali Mirbod and Ahmed EC-Amawy, " A Novel Microprocessor-Based Controller for a Phase Controlled Rectifier Connected to a Weak Ac. System.", *IEEE/IAS Annual Meeting, 1985, pp. 1250-1258.*

[57] Eiichi Ohno, Mikio Ohta, Masao Yano, Hatsuhiko Naito, and Yoshihiko Yamamoto, "Microprocessor Applications for Thyristor Choppers and High Voltage Thyristor Switches.", *U.S. Japan Seminar Conf. Rec. 1982, pp. 269-278.*

[58] P.M.S. Nambison, R. Mulchandani, and C.M. Bhatia, "Microprocessor Implementation of a Four-Phase Thyristor Chopper Circuit.", *IEEE/IAS Annual Meeting, 1980, pp. 637-642.*

[59] Koichi Ishida, Katsunari Nakamura, Tetsuo Izumi, and Masaki Ohara, "Microprocessor Control of Converter-Fed DC. Motor Drives.", *IEEE/ IAS Annual Meeting 1982, pp. 619-623.*

[60] Satoru Ozaki, Masaki Ohara, and Kooichi Ishida, " A Microprocessor Based DC. Motor Drive with a State Observer for Impact Drop Suppression.", *IEEE/IAS Annual Meeting, 1983, pp. 771-775.*

[61] Dr. Haruo Naitoh, Dr. Masaru Hirano, and Dr. Susumu Tadakuma, "Microprocessor-Based Adjustable Speed DC. Motor Drives Using

Model Reference Adaptive Control.", *IEEE/IAS Annual Meeting, 1985, pp. 524-528.*

[62] Bimal K. Bose, " A Microprocessor-Based Control System for a Near-Term Electric Vehicle. ", *IEEE Trans. Ind. App., Vol. IA-17, No. 6, Nov./Dec. 1981, pp. 626-631.*

[63] R. Kurosava, T. Shimamura, H. Uchino, and K. Sugi, " A Microcomputer-Based High Power Cycloconverter-Fed Induction Motor Drive.", *IEEE/IAS Annual Meeting, 1982, pp. 462-467.*

[64] Franz C. Zach, Rudolf J. Berthold, Karl H. Kaiser, and Eftat D. Topuzoglu, "Automatic on-Line Optimization of Microprocessor Controlled AC Motor Drives.", *IEEE/IAS Annual Meeting, 1983, pp. 659-665.*

[65] Min Ho Park and Seung Ki Sul, " Microprocessor-Based Optimal-Efficiency Drive of an Induction Motor.", *IEEE Trans. Ind. Electron., Vol. IE-31, No. 1, Feb., 1984, pp. 69-73.*

[66] H. Le-Huy, A. Jakubowicz and F. Perret, "A self-controlled Synchronous Motor Drive Using Terminal Voltage Sensing.", *IEEE/IAS Annual Meeting, 1980, pp. 562-569.*

[67] Shoji Fukuda, Yuzo Itoh and Akio Nii, " A Microprocessor-Controlled Speed Regulator for Commutatorless Motor Drives.", *International Power Elec. Conf. Rec., Tokyo, 1983, pp. 938-947.*

[68] B.K.Bose, T.J.E. Miller, P.M. Szczesny, and W.H. Bicknell, "Microcomputer Control of Switched Reluctance Motor.", *IEEE/IAS Annual Meeting, 1985, pp. 542-547.*

[69] Mehran Mirkazemi-Moud and Tim C. Green, "Analysis and Comparison of Real-Time Sine-Wave Generation for PWM Circuits," *IEEE Transactions on Power Electronics, Vol. 8, No.1, January 1993, pp.46-54.*

[70] S.Vadivel, G. Bhuvaneswari, and G. Sridhara Rao, "A Unified Approach to the Real-Time Implementation of Microprocessor-Based PWM Waveforms," *IEEE Transactions on Power Electronics. Vol. 6. No. 4, October 1991, pp.565-575.*

[71] Julian Richardson and Osman T. Kukrer, "Implementation of a PWM Regular Sampling Strategy for AC Drives," *IEEE Transactions on Power Electronics Vol. 6. No. 4. October 1991, pp.645-655.*

[72] S. R. Bowes and P. R. Clark, "Transputer Based Harmonic Elimination PWM Control of Inverter Drives.", *IEEE Transaction on Industry Application, Vol. 28, No. 1, Jan/Feb 1992, pp. 73-79.*

[73] G.S. Bhuja and P. Fiorini, " Micro Computer Control of PWM Inverters," *IEEE Transactions on Industrial Electronics. Vol. 29, August 1982, pp.212-216, August 1982.*

[74] J.A. Taufiq, B. Mellit. and C.J. Goodman, "Novel Algorithm for Generating near Optimal PWM Waveform for AC Traction Drives," *IEEE Proc. B. Elect. Power Appl. Vol. 133, March 1986, pp. 85-94.*

[75] J. Richardson and S.V.S. Mohanram, "An Auto-adjusting PWM Waveform Generation Technique.", *IEE Conf. Pub. No. 291, July 1988, pp. 366-369.*

[76] K.S. Rajashekara and Joseph Vithayathil, "Microprocessor Based Sinusoidal PWM Inverter by DMA Transfer.", *IEEE Transactions on Industrial Electronics, Vol. IE-29, No. 1, February 1982, pp.46-51.*

[77] Bimal K. Bose and Hunt A. Sutherland, "A high-performance Pulse width Modulator for an Inverter-Fed Drive System Using a Microcomputer," *IEEE Transactions on Industry Applications, Vol. IA-19, No. 2, March/April 1983, pp.235-243.*

[78] Donato Vincenti, Phoivos D. Ziogas, and Rajnikant V. Patel, "A PC-Based Pulse-Width Modulator for Static Converters," *IEEE Transactions on Industrial Electronics, Vol. 37, No. 1, February 1990, pp.57-69.*

[79] Bong-Hwan Kwon and Byung-duk Min, "A Fully Software-Controlled PWM Rectifier with Current Link," *IEEE Transactions on Industrial Electronics, Vol. 40, No. 3, June 1993, pp.355-363.*

[80] S. R. Bowes, "Novel Real-Time Harmonic Minimized PWM Control for

Drives and Static Power Converters," *IEEE Transactions on Power Electronics, Vol. 9, No. 3, May 1994, pp.256-262.*

[81] M. A. Choudhury, *"An Analysis of Delta Modulated Inverters With Application to Submersible Motors."*, A Thesis submitted for the degree of Doctor of Philosophy, Memorial University of Newfoundland, Canada.

[82] Ali-Reza D. Esmail, *"Analysis and Design of Delta PWM Static Dc to Ac Converters."*, A Thesis submitted for the degree of Master of Engineering , Memorial University of Newfoundland, Canada.

[83] M. A. Rahman, J. E. Quaicoe, and M. A. Choudhury, "Performance Analysis of Delta PWM Inverters.", IEEE Trans. on PE., vol. PE-2, 1987, pp.227-233.

[84] *Microcomputer control of Power Electronics and Drives,* Edited by B.K. Bose.New York : IEEE Press Selected Reprint Series, 1987.

[85] *Personal Computer XT Hardware Reference Library,* Technical Reference.

[86] *Personal Computer 286 Hardware Reference Library,* Technical Reference. , 1984.

[87] Harry Fairhead, *The 386/486 PC, A Power User's Guide.,* 1994.

[88] *intel iAPX 286, Hardware Reference Manual.*

[89] Nelson Johnson, *Advanced Graphics in C : Programming and Techniques.,* Osborne Mc Graw-Hill.

[90] Stan Kelly-Bootle, *Mastering Turbo C,* sybex Inc., U. S. A., 1988.

[91] Herbert Schildt, *Turbo C*®*/ C*$^{++}$ *:The Complete Reference.,*Osborne Mc Graw-Hill, U. S. A., 1990.

[92] *Modern Power Electronics, Evolution, Technology, and Applications,* Edited by B. K. Bose, New York : IEEE Press Selected Reprint Series, 1991.

APPENDIX.A

PTDMWG.CPP : A Program which Performs On-line Microcomputer Control of Delta Modulated Three-Phase Inverter.

```
//*****************************************************
//PTDMWG.CPP, Program for on line control of inverter by DPWM
//*****************************************************

# include < stdio.h >
# include < ctype.h >
# include < stdlib.h >
# include < iostream.h >
# include < dos.h >
# include < conio.h >
# include < math.h >
# include <fstream.h>
# include < \tc\classlib\include\timer.h >
# include < \tc\classlib\source\timer.cpp >
# define YY 80

//*****************************************************
//This part identifies all the variable used in this program

        double p,x,y;
        int Y,n,N,i,j,k,r,num_seg,deci[25000];
        float t[100],s,dv,vm,wm,fm,anum,den,init,tm,tt;
        double time1=0.0,del=0.0;

        int flag = 0;
        char state = ' ', ch;
        char val[100], z;
        Timer timer1;
//**********************************************************
//This Subfunction will determine the time interval of each time segmant to be
scanned.

void StartUp( void )
{
        timer1.reset( );
        timer1.start( );
        for (Y=0;Y<YY;Y++){ }
        timer1.stop( );
        del = timer1.time( );
        printf( "\n del=%f\ n",del);

}

//*****************************************************
// This subfunction calculates the timing instants and creates an array which will
generate the required pulse pattern.
```

```
void CalTime( void )
{
// Calculation of timing instants.

        puts( "running..." );
        N = 0;
        t[0] = 0.0;
        wm=2.0*M_PI*fm;
        tm=1.0/fm;
        num_seg = int(tm/del);

        for(n=1; ; n++)
                {
                anum=2.0*dv;
                x=-1.0;
                y=(float)n-1.0;
                p=pow(x,y);
                den=s+(p*vm*wm*cos(wm*t[n-1]));
                t[n]=t[n-1]+anum/den;
                N=N+1;
                if (t[n] > = M_PI/wm) break;
                }
// Creation of the array deci [num_seg].
        for(r = 0; r< num_seg; r++ ) deci[r]=0;

        for (k=0; k < 6; k++)
        {
            if  (k= = 0)  init=tm/6.0;
        else if  (k= =1)     init=5.0*tm/6.0;
        else if  (k= =2)     init=tm/2.0;
        else if  (k= =3)     init=2.0*tm/3.0;
        else if  (k= =4)     init=tm/3.0;
        else                 init=0.0;

        for(i=0; i < N; i= i+2)
            { for (tt = t[i]; tt < = t[i+1]; tt = (tt+del))
                 { r = (int) ( ( tt-t[0] + init ) / del);
                      if ( r > ( num_seg-1 ) )
                      r = (r - (num_seg) );
                      deci[r] = deci[r] + (int) ( pow ( 2.0, (float) (5-k ) ) );
                 }
            }
        }
            printf (" \ n deci[30] = %d \ n", deci[30] );
            printf (" \ n N = %d \  n",N );
    }
    //****************************************************
    // This subfunction generates the required pulse pattern.

    void GenPulse( void )

    {
        for (r = 0; r < num_seg; r++ )
        {   outportb(0x378,deci[r] );
                  for (Y=0;Y< YY;Y++){ }
        }
}

//***********************************************************
// This is the main program

void main(void)
{

        dv=1.0;
        s=2500.0;
        vm=5.0;
        fm=50.0;

        StartUp( );
        CalTime( );

        for ( ; state != 'q'; )
        {
                if( !flag && kbhit( ) )
                {       state = getch( );
                        printf( " \ n I am ready to take new [%c]...\ n \ n", state );
                        flag = 1;
                        z = 0;
                }

                if( flag && kbhit( ) )
                {       ch = getche( );
                        if( isdigit (ch ) || ch == '.' )
```

```
             { val[z++] = ch;}
            else {  val[z] = 0;
                    switch( state ) {
                    case 'f' : sscanf ( val, "%f", & fm ); break;
                    case 'd' : sscanf ( val, "%f", & dv ); break;
                    case 's ' : sscanf ( val, "%f", & s  ); break;
                    case 'v ' : sscanf ( val, "%f", & vm ); break;
                    }

                    CalTime(  );
                    flag = 0;
                }
        }
        GenPulse( );
    }
}
```

APPENDIX.B

PSDMWG.CPP: A Program which Performs On-line Microcomputer Control of Delta Modulated Single-Phase Inverters Only.

```
//*******************************************************
// PSDMWG.CPP, Program for on line control of inverter by DPWM
//*******************************************************

# include < stdio.h >
# include < ctype.h >
# include < stdlib.h >
# include < iostream.h >
# include < dos.h >
# include < conio.h >
# include < math.h >
# include < fstream.h >
# include < \tc\classlib\include\timer.h >
# include < \tc\classlib\source\timer.cpp >

//*******************************************************
//This part identifies all the variable used in this program.

      double p,x,y;
      int n, N, i, j, k, ss;
      float t[100],dlay[100],s, dv, vm, wm, fm, anum, den;
```

```
        double time1=0.0, time2=0.0, time3=0.0;
        int flag = 0;
        char state = ' ', ch;
        char val[100], z;
        Timer timer1;
        Timer timer2;
        Timer timer3;
//*******************************************************
//This subfunction calculates the timing instants for PWM waveforms.

void CalTime( void )
{
        puts( "running..." );
        N = 0;
        t[0] = 0.0;
        wm=2.0*M_PI*fm;

        for(n=1; ; n++)
                {
                anum=2.0*dv;
                x=-1.0;
                y=(float)n-1.0;
                p=pow(x,y);
                den=s+(p*vm*wm*cos(wm*t[n-1]));
                t[n]=t[n-1]+anum/den;
                dlay[n]=t[n]-t[n-1];
                N=N+1;
                if (t[n]>=M_PI/wm) break;
                }
```

```
}
//*******************************************************
//This subfunction gives the time required to reset ‘timer1’ and ‘timer2’.

void StartUp( void )
{
        timer3.reset( );
        timer3.start( );
        timer1.reset( );
        timer2.reset( );
        timer3.stop( );
        time3 = timer3.time( );
}
//*******************************************************
// This subfunction generates the required pulse pattern for the first half-cycle.

void GenPulse1( void )
{
        for (j=1; j<= N; j++)
        {
                if((j-2*(j/2))==0)
                outportb(0x378, 0x 00);
                else
                outportb(0x378, 0x 01);

                timer1.reset( );
                timer2.reset( );
                timer1.start( );
                i = 1;
        while (i)
                {
                time2=timer2.time( );
                timer2.start( );
                timer1.stop();
                time1=timer1.time( );
        if ((time1+time2+time3) >= (dlay[j]/11.07)){timer2.stop( ); i = 0;}
                else {timer1.start( ); timer2.stop( );}
                }
        }
}
//*****************************************************
// This subfunction generates the required pulse pattern for the second half cycle.

void GenPulse2( void )
{
        for (j=1;  j<= N; j++)
        {

                if((j-2*(j/2))==0)
                outportb(0x378, 0x 00);
                else
                outportb(0x378, 0x 02);

                timer1.reset( );
                timer2.reset( );
                timer1.start( );
                i = 1;
        while (i)
            {
            time2=timer2.time( );
            timer2.start( );
            timer1.stop( );
            time1=timer1.time( );
        if ((time1+time2+time3) >= (dlay[j]/11.07)){timer2.stop( ); i = 0;}
            else {timer1.start( ); timer2.stop( );}
            }
        }
}

//******************************************************
// This is the main program

void main(void)
{

        dv=1.0;
        s=2500.0;
        vm=5.0;
        fm=50.0;

        CalTime( );
        StartUp( );

for ( ; state != 'q'; )
{
            if( !flag && kbhit( ) ) {
```

```
            state = getch( );
            printf( "\n I am ready to take new [%c]...\n\n", state );
//          printf( "\n[N=%d] \n", N );
            flag = 1;
            z = 0;
            }
        if( flag && kbhit( ) ) {
            ch = getche( );
            if( isdigit( ch ) || ch == '.' ) {
                val[z++] = ch;
                }
            else {
                val[z] = 0;
                switch( state ) {
                    case 'f' : sscanf( val, "%f", &fm ); break;
                    case 'd' : sscanf( val, "%f", &dv ); break;
                    case 's' : sscanf( val, "%f", &s  ); break;
                    case 'v' : sscanf( val, "%f", &vm ); break;
                    }
                CalTime( );
                flag = 0;
            }
        }

    GenPulse1( );
    GenPulse2( );
}
}
```