CHAPTER 4

DIRECT AC-AC CONVERTER

-

PRACTICAL IMPLEMENTATION

Contents

Direct AC-AC Converter- Practical Implementation

In the previous chapter, a novel AC-AC conversion technique is proposed. The research on Direct AC- AC conversion had started in late seventies; Alesina and Venturini being the first to propose the mathematical model. Due to the non-availability of modern semiconductor devices and difficulty in implementation of complex control circuitry, these converters had not been so popular then. Now a days, with the revolutionary advancement in the semiconductor industries with development of devices like GTOs, IGBTs, MCTs etc; direct AC –AC conversion has revoked the interest and has been the most preferred topic for research.

Moreover the control logic implementation from hard-core analog circuits (based on Opamps and logical ICs) has advanced to integrated (ASICs) and digital circuits involving the microprocessors, micro-controllers and DSPs. The usage of these fast calculating processors having lot of inherent features like inbuilt ADCs, in built memory storage capacity, programmable PWM output pins etc has made the implementation more simple and easy. In the previous chapter, a simple and novel PWM based control technique for direct AC-AC is proposed and explained. Detailed Simulation study has been carried out and the results are incorporated in chapter 3.

In this chapter, the proposed technique is realized using simple analog circuits. With the implementation of the control using the analog circuits, the size of the control circuits is large and to make the control circuit size more compact the same is implemented with DSP. Important modules of control circuit are explained here in detail and results of the same are included in this chapter. The power circuit is realized and implemented using discrete components (MOSFETs and diodes). This

chapter explains in detail practical implementation and all the experimental results are incorporated.

4.1 Control Circuit Implementation

The block diagram of the control circuit is shown in the figure 4.1 below. Three phase mains are scaled down to low voltages using PTs that are compatible to the control board and then these signals are fed to the control boards as inputs.

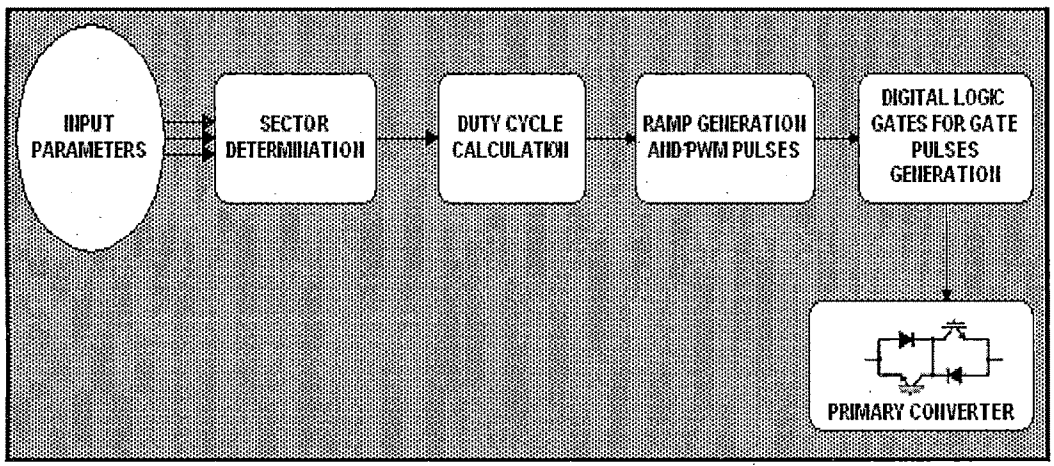


Fig 4.1 Control block diagram for Direct AC-AC conversion

The control process is divided in following four stages:

- ZCD and Sector determination
- Duty cycle calculation
- Ramp generation and PWM generation
- Digital logical gates for generation of gate pulses

4.1.1 ZCD and sector determination:

As mentioned in chapter 3, for sector determination it is required to find the phase having maximum instantaneous value of all the three input phases at a specific instant of time or interval. To achieve this, it is required to find the absolute values of each phase using a rectifier circuit with a resistive load. The output waveforms for each phase are like unfiltered DC voltages as shown in figure 4.2.

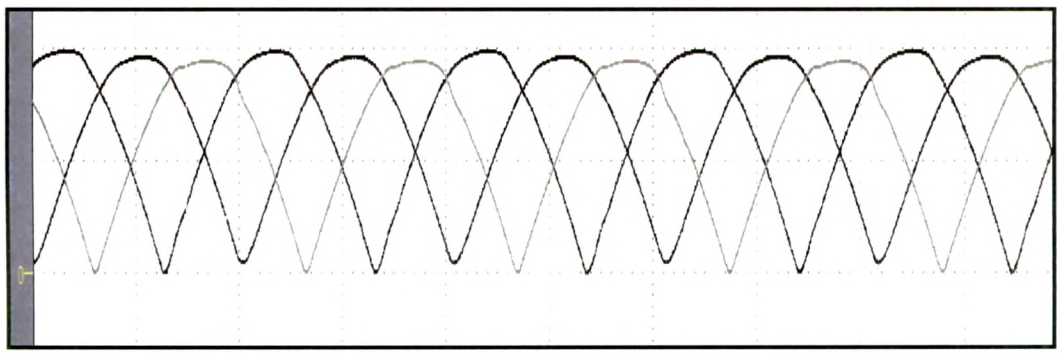


Fig 4.2 Absolute instantaneous value of three phase voltages

The output of this rectifier circuit acts as an input to a high-speed comparator that compares the absolute values of one phase with other two phases, resulting in generation of two pulses per cycle of input phase.

This shows that every phase attains the maximum value twice in a complete cycle. Zero crossing detector circuit has also major role in determining the exact sector and its conducting phase during the positive and negative half cycle. Thus stage one yield in all nine outputs, i.e. three zero crossing square pulses and six pulses for six sectors. Output wave forms of zero crossing detector circuits and sector selection circuits for all the three phases are shown in figure 4.3, 4.4 and 4.5 below:

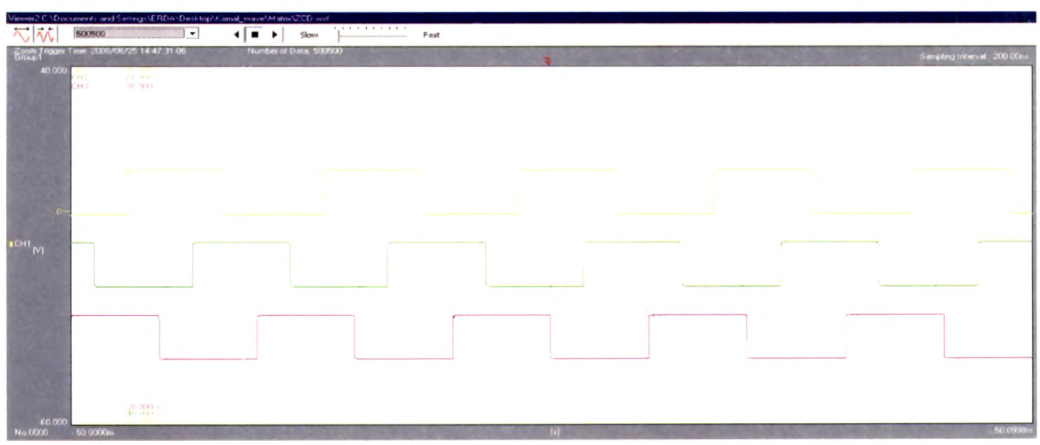


Fig 4.3 Zero crossing detector waveforms

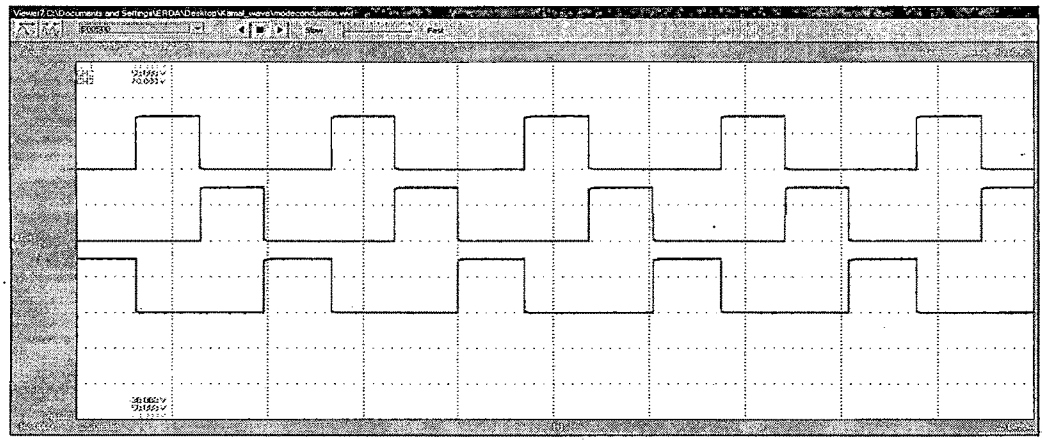


Fig. 4.4 sector detection for all the phases

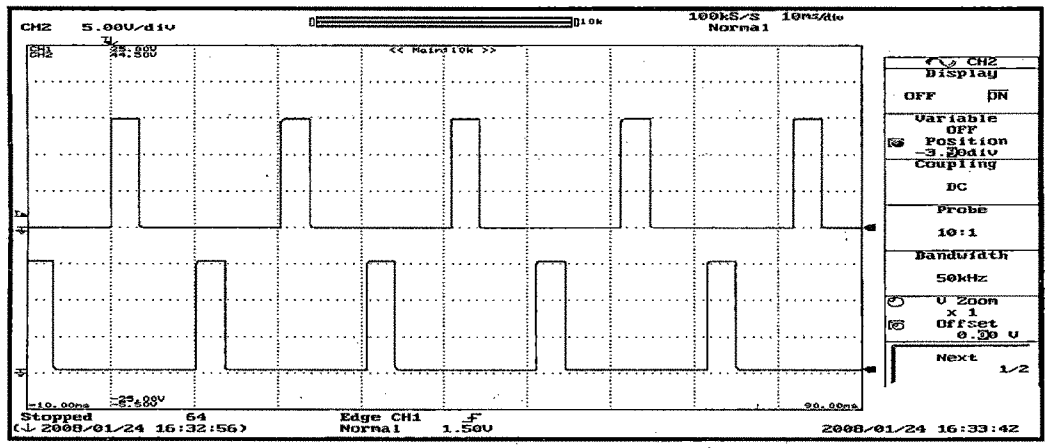


Fig 4.5 Sector detection for R-phase (positive and negative half cycles)

4.1.2 Duty Cycle Calculation

Once the sectors are determined, it is required to calculate the duty cycle for each bidirectional switch. For determining the duty cycle it is required to implement the following equation using analog circuits.

$$d_r = \frac{|v_r|}{\max(|v_r|,|v_y|,|v_b|)};\quad d_y = \frac{|v_y|}{\max(|v_r|,|v_y|,|v_b|)};\quad d_b = \frac{|v_b|}{\max(|v_r|,|v_y|,|v_b|)}$$

By determining the duty cycle using the above equation will always generate the maximum possible voltage thus increase the efficiency of the converter. By the above equation the voltage vector selection is

appropriate and maximum output can be generated by usage of one voltage vector having the maximum absolute value, which contributes the most. It a also possible to use the $T_{sw}/2$, for each pair of V_r-V_y and remaining Tsw/2 for other pair Vr-V_b. but in that case we are not in the position to maintain the voltage balance across the transformer which will lead to saturation. In order to implement these equations, it is required to calculate maximum instantaneous value of all the three phase voltages. Then the second step is to divide the absolute value of each phase with the maximum instantaneous values of the three phases. This division will result in the generation of modulation function for each switch. To implement this mathematical operation of dividing the two values using analog circuits it a difficult task. This task is performed using a multiplier IC as shown in circuit in figure 4.6.

Consider A and B are two inputs fed to the non inverting amplifier circuit shown above and G is the final output obtained at pin 7 of a inverting summing amplifier. So from the circuit, it can be said that the input to the summing amplifier op-amp at point C is A+G and output of this amplifier at point E is –(A+G).

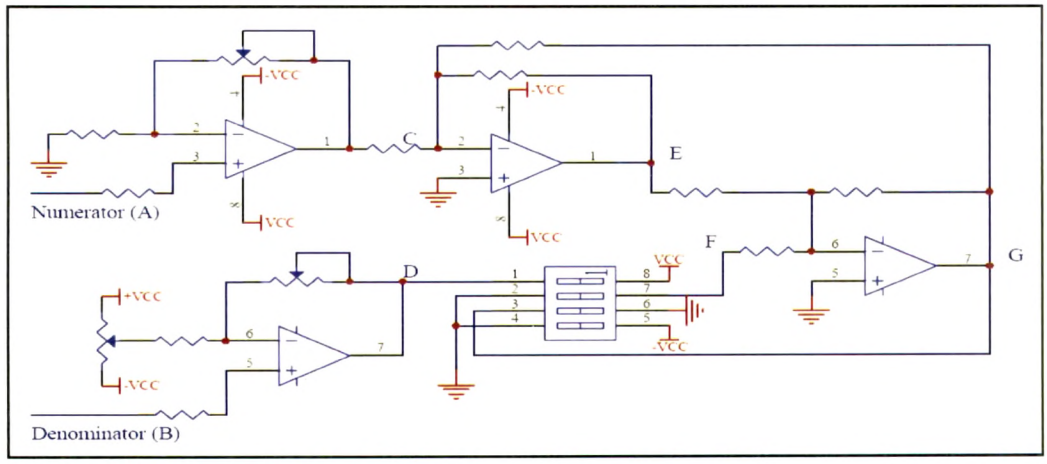


Fig 4.6 Divider Circuit using multiplier IC and opamps

The multiplier IC shown in the fig 4.6 is fed with two inputs B and G and hence the output of this multiplier IC is B*G at point D. The output of

the multiplier IC serves as one of the input to the inverting summing amplifier at point F. The other input is –(A+G). So the output of this summing amplifier as per logical calculations should be –(B*G - (A+G)).

But the output is G; so the logically G=-(B*G - (A+G)).
That means B*G – A=0; means G=A/B.

Here A is equal to the instantaneous absolute value of each phase voltage and B is the instantaneous maximum absolute value of all the three phase voltages. And the output of the division circuit will be the modulating function for corresponding phase. The modulating functions for all the three phases are shown in figure 4.7.

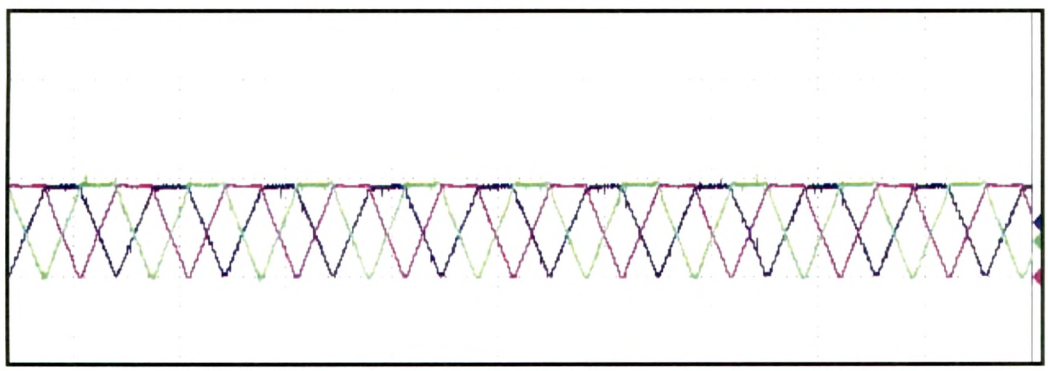


Fig 4.7 Modulation function for all three phases

4.1.3 Ramp circuit

The modulating functions hence derived are compared with ramp waveforms to generate the required pulses for the switching of bidirectional switches. Both positive and negative slope ramps are required for comparison purpose. Ramp waveforms are generated using LM555 timer IC as shown in circuit below.

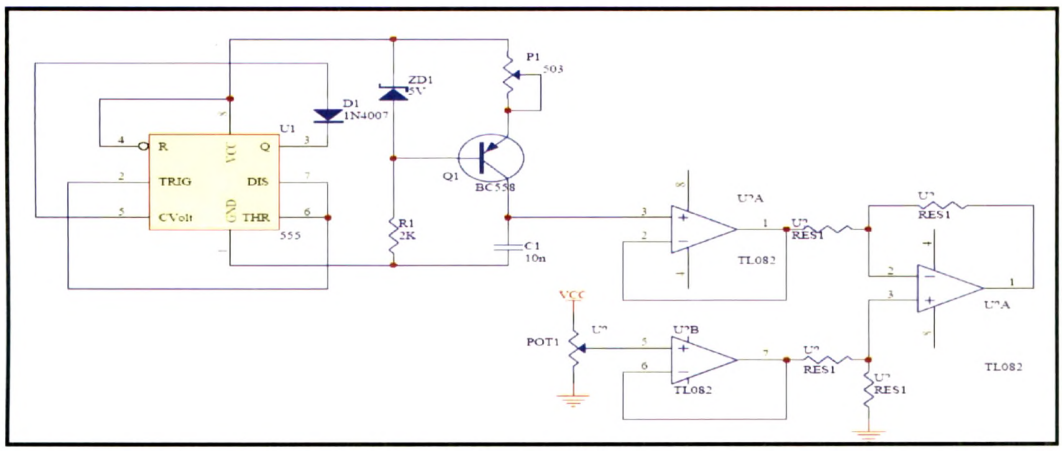


Fig 4.8 Positive and negative ramp generating circuit

The circuit shown above generates both positive slope ramp as well as negative slope ramp. These ramp waves act as a carrier frequency signal to be used for pulse width modulation to generate the switching pulses per sector.

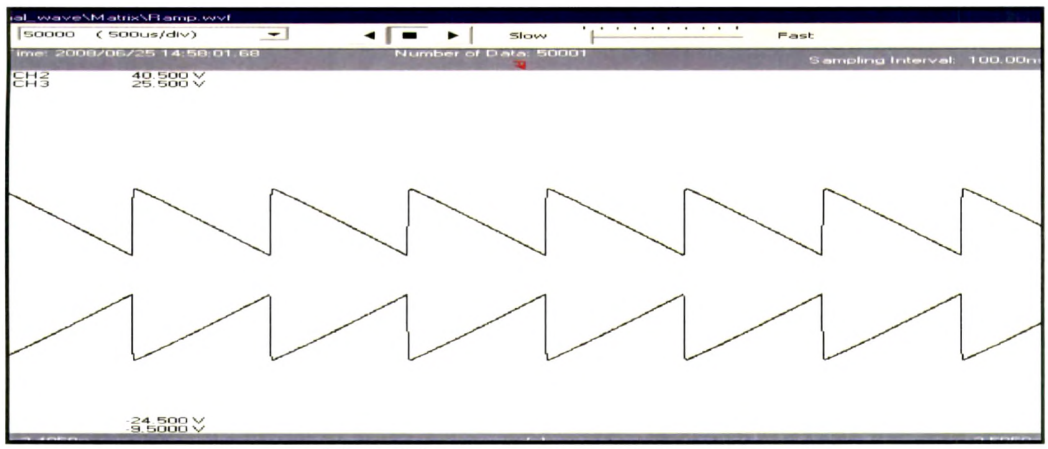
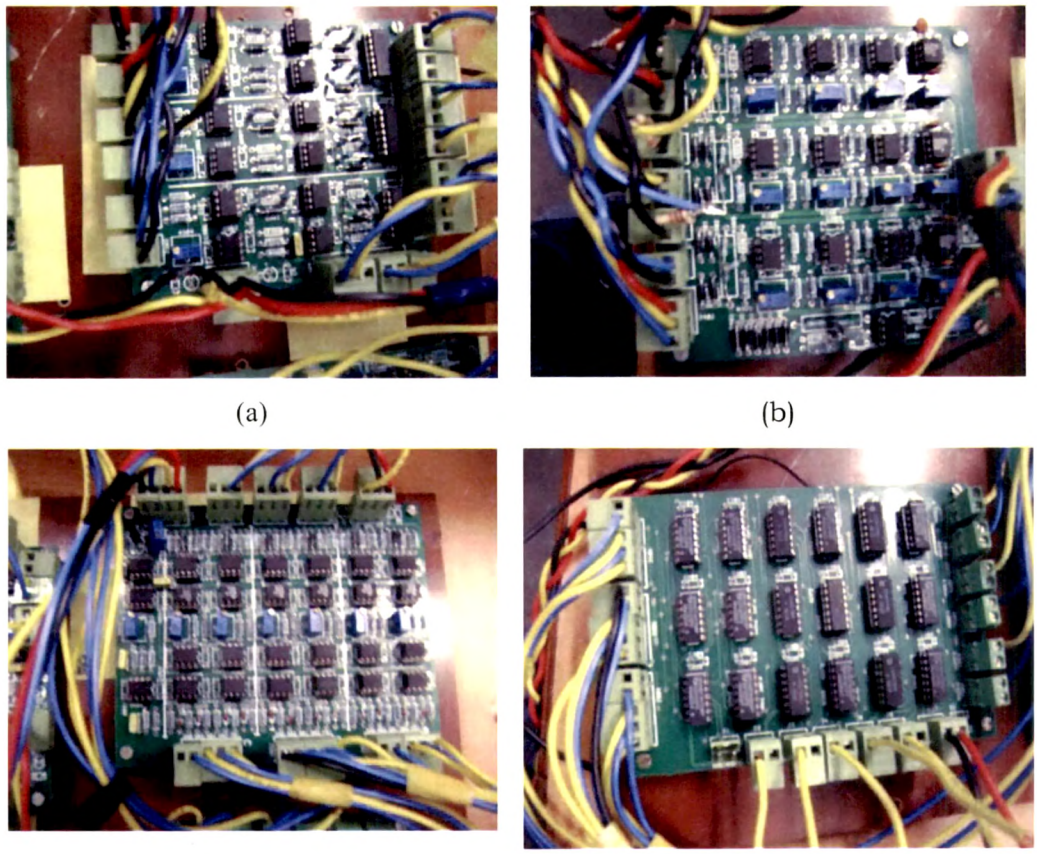


Fig 4.9 Positive and negative slope ramps

Using various digital logic gates are used to generate final gates pulses for all the six bi directional switches. These pulses are then fed to gate driver circuits to generate proper level of voltages and provide necessary isolation. The gate driver is developed using hybrid driver IC M57962L. Schematic layout of all the pcbs are included in Annexures.



(a) (b)

(c) (d)

Fig 4.10 Pictorial View of Analog circuits based control cards

a) **Sector determination and ZCD card**
b) **Divider card and duty cycle calculation card**
c) **Ramp generation card**
d) **Digital card for gate pulses**

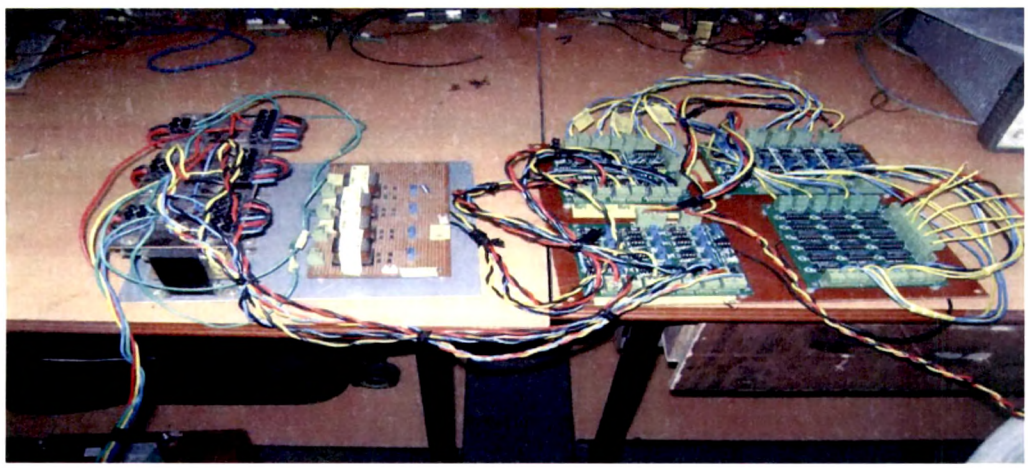


Fig 4.11 Pictorial view of setup for all control cards inter connected to one another

Due to the implementation of control logic using analog Ics, the size of the control board is too big. So an attempt has been made to implement the control using the DSP TMS320F2812 processor. Figure gives the pictorial view of the developed DSP board with all the peripherals mounted and signal conditioning circuit mounted on it. The flow of the program remains the same as it is in case of analog circuit case. Initially all the port pins, interrupt pins and I/o pins are initialized.

Software Implementation

```
// Prototype statements for functions found within this file.

// Global variables used in this example


// These are defined by the linker (see F2812.cmd)
extern Uint16 RamfuncsLoadStart;
extern Uint16 RamfuncsLoadEnd;
extern Uint16 RamfuncsRunStart;

void main(void)
{

  //InitGpio();
// Step 1. Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
// This example function is found in the DSP281x_SysCtrl.c file.
  InitSysCtrl();
```

```
// Step 2. Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
  DINT;

// Initialize PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the DSP281x_PieCtrl.c file.
  InitPieCtrl();

// Disable CPU interrupts and clear all CPU interrupt flags:
  IER = 0x0000;
  IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
// This will populate the entire table, even if the interrupt
// is not used in this example.  This is useful for debug purposes.
// The shell ISR routines are found in DSP281x_DefaultIsr.c.
// This function is found in DSP281x_PieVect.c.
  InitPieVectTable();

// Interrupts that are used in this example are re-mapped to
// ISR functions found within this file.
  EALLOW;  // This is needed to write to EALLOW protected registers
  EDIS;   // This is needed to disable write to EALLOW protected registers


// Step 3. Initalize GPIO:
  InitGpio();

 // Step 4. Initialize all the Device Peripherals:
// This function is found in DSP281x_InitPeripherals.c
  InitPeripherals();

// Step 5. User specific code, enable interrupts:
  EALLOW;  // This is needed to write to EALLOW protected registers
  //PieVectTable.T2PINT = &eva_timer2_isr;
  EDIS;   // This is needed to disable write to EALLOW protected registers


// Copy time critical code and Flash setup code to RAM
// This includes the following functions:  InitFlash(),
// The  RamfuncsLoadStart, RamfuncsLoadEnd, and RamfuncsRunStart
// symbols are created by the linker. Refer to the F2812.cmd file.
  MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
```

```
// Call Flash Initialization to setup flash waitstates
// This function must reside in RAM
  InitFlash();

    // Step 6. IDLE loop. Just sit and loop forever:


while(1)
  {

/*************************************************************************
*****/
                // Enable Capture Interrupt

        // Enable PIE group 3 interrupt 5 for CAP1
                PieCtrlRegs.PIEIER3.all = M_INT5;
   // Enable CPU INT3 for CAP1
                IER |= M_INT3;

/*************************************************************************
*******/

  // Enable global Interrupts and higher priority real-time debug events:

                EINT;  // Enable Global interrupt INTM
                //ERTM;  // Enable Global realtime interrupt DBGM

                while (StartFlg <= 4)
                {
                InitADCOffset();
                }

                ReadADC();
                max_value();
                sector();


                }

        }
```

After initializing of all the port pins, the ADC is initialized and offset value is calculated, as the input to the ADC is uni-polar in nature. The

subtraction of this form the ADC result registers will yield in accurate input signals.

```
void InitADCOffset(void)
{

  while(1)
                {
                if (StartFlg >= 3)
                        {

                        AdcRegs.ADCTRL2.bit.SOC_SEQ1 = 1;
                        while (AdcRegs.ADCASEQSR.bit.SEQ_CNTR ||
AdcRegs.ADCST.bit.SEQ1_BSY);   //Wait for completion of conv seq
                        if((AdcRegs.ADCRESULT0 >> 4) > max[0])
                                max[0] = (AdcRegs.ADCRESULT0 >> 4);
                        if((AdcRegs.ADCRESULT0 >> 4) < min[0])
                                min[0] = (AdcRegs.ADCRESULT0 >> 4);

                        if((AdcRegs.ADCRESULT1 >> 4) > max[1])
                                max[1] = (AdcRegs.ADCRESULT1 >> 4);
                        if((AdcRegs.ADCRESULT1 >> 4) < min[1])
                                min[1] = (AdcRegs.ADCRESULT1 >> 4);

                        if((AdcRegs.ADCRESULT2 >> 4) > max[2])
                                max[2] = (AdcRegs.ADCRESULT2 >> 4);
                        if((AdcRegs.ADCRESULT2 >> 4) < min[2])
                                min[2] = (AdcRegs.ADCRESULT2 >> 4);

                        }
                if(StartFlg >= 4)
                        break;
                }
  for(scounter = 0; scounter < 3; scounter++)
                {
                ADCOffset[scounter] = ( ( max[scounter] + min[scounter] ) >> 1 );//16
subtracted for offset comp

                }                                               //2116
      scounter = 0;
}
```

ADC registors are 12 bit registers and are programmed in sequential mode so that ADC converters the analog voltage signals into equivalent digital count, the resultant digital count is stored in ADCresult registers.

```
void ReadADC(void)
{
        vr = /*_IQ19*/( (AdcRegs.ADCRESULT0 >> 4) - (ADCOffset[0]) );
        vr = vr << 19;//_IQ19(vr);
        vy = /*_IQ19*/( (AdcRegs.ADCRESULT1 >> 4) - (ADCOffset[1]) );
        vy = vy << 19;//_IQ19(vy);
        vb = /*_IQ19*/( (AdcRegs.ADCRESULT2 >> 4) - (ADCOffset[2]) );
        vb = vb << 19;//_IQ19(vb);
 }
```

This sector of program yields the maximum value of absolute instantaneous voltages of all the three phases input voltages. Once these values are known it is easy to identify the each sector and also calculate the modulation function for each phase

```
void max_value(void)
{

        VoltageR = vr;
        VoltageY = vy;
        VoltageB = vb;
        VoltageR = _IQ19abs(VoltageR);
        VoltageY = _IQ19abs(VoltageY);
        VoltageB = _IQ19abs(VoltageB);
        if(VoltageR > VoltageY && VoltageR > VoltageB )
        {
                maxV = VoltageR;
        }

        if(VoltageY > VoltageB && VoltageY > VoltageR )
        {
                maxV = VoltageY;
        }
        if(VoltageB > VoltageR && VoltageB > VoltageY )
        {
                maxV = VoltageB;
}

}
```

```
 void sector(void)
{
            if(VoltageR > VoltageY && VoltageR > VoltageB && vr > 0)
            {
            //      sector1();
            }
            else if(VoltageR > VoltageY && VoltageR > VoltageB && vr < 0)
            {
            //      sector4();
            }
            else if(VoltageY > VoltageB && VoltageY > VoltageR && vy > 0)
            {
                  // sector3();
            }
            else if(VoltageY > VoltageB && VoltageY > VoltageR && vy < 0)
            {
                  //sector6();
            }
            else if(VoltageB > VoltageR && VoltageB > VoltageY && vb > 0)
            {
                  //sector5();
            }
            else if(VoltageB > VoltageR && VoltageB > VoltageY && vb < 0)
            {
                  //sector2();
            }

 }
void dutycycle (void)
{
            Dr = VoltageR/maxV;
            Dy = VoltageY/maxV;
            Db = VoltageB/maxV;

}
```

```
// Initialize ADC Peripheral To default State:
  //InitAdc();
       // To powerup the ADC the ADCENCLK bit should be set first to enable
  // clocks, followed by powering up the bandgap and reference circuitry.
  // After a 5ms delay the rest of the ADC can be powered up. After ADC
  // powerup, another 20us delay is required before performing the first
  // ADC conversion. Please note that for the delay function below to
  // operate correctly the CPU_CLOCK_SPEED define statement in the
  // DSP28_Examples.h file must contain the correct CPU clock period in
  // nanoseconds. For example:

       AdcRegs.ADCTRL3.bit.ADCBGRFDN = 0x3;      // Power up
bandgap/reference circuitry
       for(del = 0; del < 65000; del++)asm(" RPT #7 || NOP");
       //DELAY_US(ADC_usDELAY);              // Delay before powering up rest of
ADC
       AdcRegs.ADCTRL3.bit.ADCPWDN = 1;            // Power up rest of ADC
       for(del = 0; del < 2500; del++)asm(" RPT #7 || NOP");
       //DELAY_US(ADC_usDELAY2);             // Delay after powering up AD
       AdcRegs.ADCTRL1.bit.ACQ_PS = ADC_SHCLK; // Sequential mode: Sample
rate  = 1/[(2+ACQ_PS)*ADC clock in ns]
                                           //              = 1/(3*40ns) =8.3MHz
                                           // If Simultaneous mode enabled: Sample
rate = 1/[(3+ACQ_PS)*ADC clock in ns]
  AdcRegs.ADCTRL3.bit.ADCCLKPS = ADC_CKPS;
  AdcRegs.ADCTRL3.bit.SMODE_SEL = 0;       //Sequential sampling mode
  AdcRegs.ADCTRL1.bit.SEQ_CASC = 0;      // 1  Cascaded mode
  AdcRegs.ADCTRL1.bit.CONT_RUN = 1;      // Setup start/stop mode
  AdcRegs.ADCTRL2.bit.SOC_SEQ1 = 0;          //Clear pending SOC trigger

  //AdcRegs.ADCTRL1.bit.SEQ_OVRD = 1;     // Enable Sequencer override feature
  AdcRegs.ADCCHSELSEQ1.all = 0x0210;      // Initialize all ADC channel selects
  //AdcRegs.ADCCHSELSEQ2.all = 0x0BA98;//0x7654;//0x0BA98
 // AdcRegs.ADCCHSELSEQ3.all = 0x0FEDC;//0x0BA98;
  //AdcRegs.ADCCHSELSEQ4.all = 0x0FEDC;
  AdcRegs.ADCMAXCONV.bit.MAX_CONV1 = 0x02;//0x0F;  // convert and store in
16 results registers
```

The input phase voltages are fed to the DSP via a signal conditioning circuit consisting of op-amps. The outputs of the signal conditioning board are fed to the ADC pins of the DSP. With the sequential mode of sampling selected, the DSP samples all the three input voltages one after the other and the digital value obtained are stored in three registers.

Then the mathematical calculation process starts for determination of sectors, calculation of duty cycles, ramp comparison and PWM generation etc. The pulses generated are then fed to the driver cards.

The flow of the complete program remains the same as it follows the same mathematical model and algorithm to determine the sector, calculate the duty cycle for each switch, generate the PWM and deliver the pulses to the driver cards.

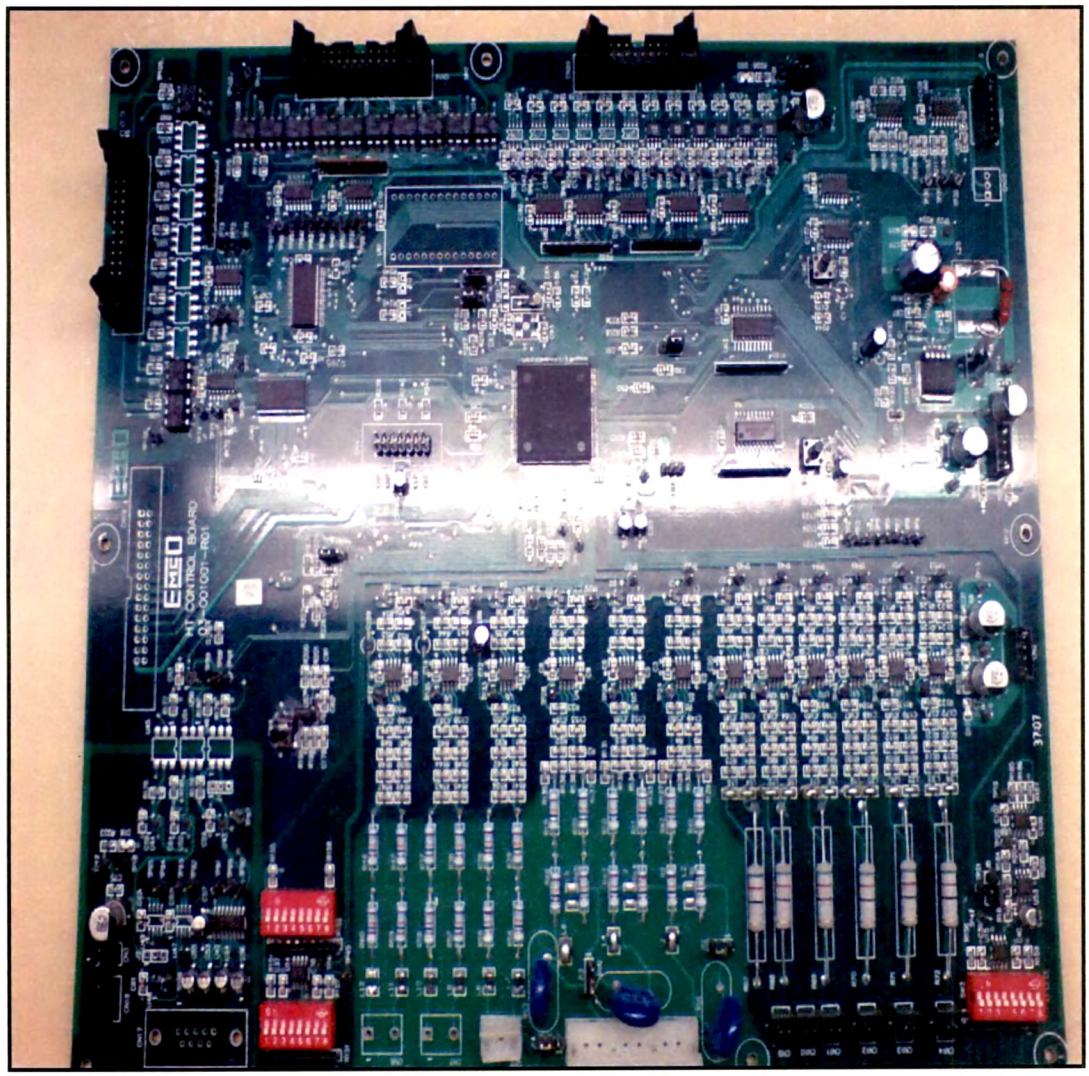


Fig 4.12 Pictorial view of the DSP based control board

4.2 Power Circuit:

Primary Converter power circuit is developed using six bidirectional switches. These bidirectional switches are fabricated using discrete components such as 500V, 8 amps MOSFETS, and BY299 fast diodes for using them in the prototype model. It is observed though usage of discrete components is beneficial for development of prototype model and proving the principle, in practice it is advisable to use bidirectional switch modules available in market. This leads to more compact size and more convenient usage.

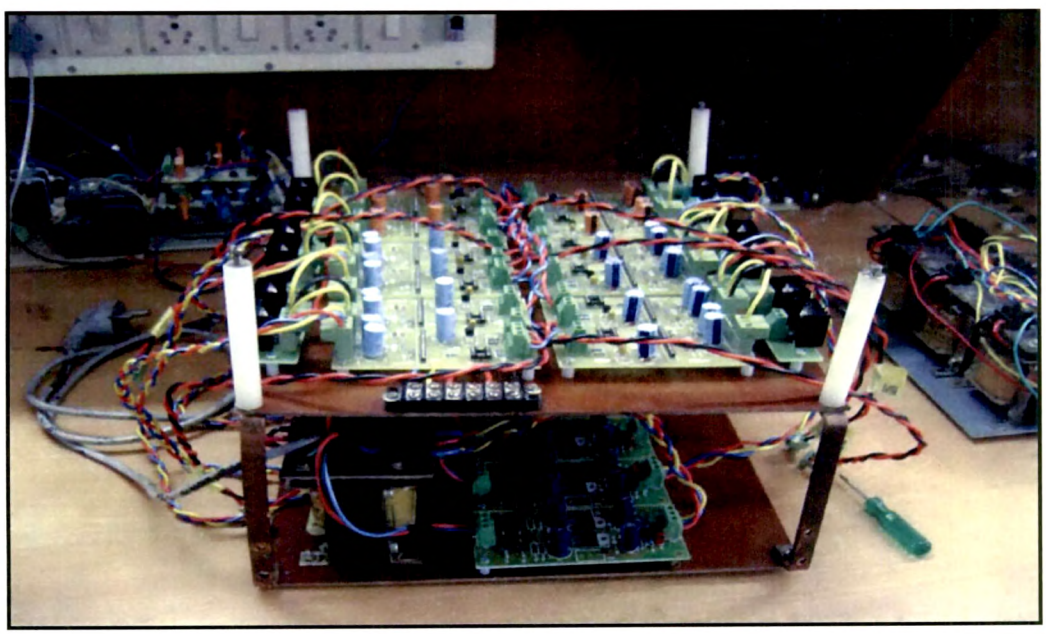


Fig 4.13 Pictorial view of the analog based complete set-up

Nowadays integrated bidirectional switches are available of SEMIKRON make and EUPEC make. Still these switches are costly and not easily available as the delivery period of these switches is too high.

The output of the primary converter is a high frequency transformer having a turn ratio of 1:1 and is designed for the switching frequency of 1.5 kHz. The power rating of the transformer is fixed as 440 volts, 1

amps. The waveform across the high frequency transformer is shown in figure 4.15.

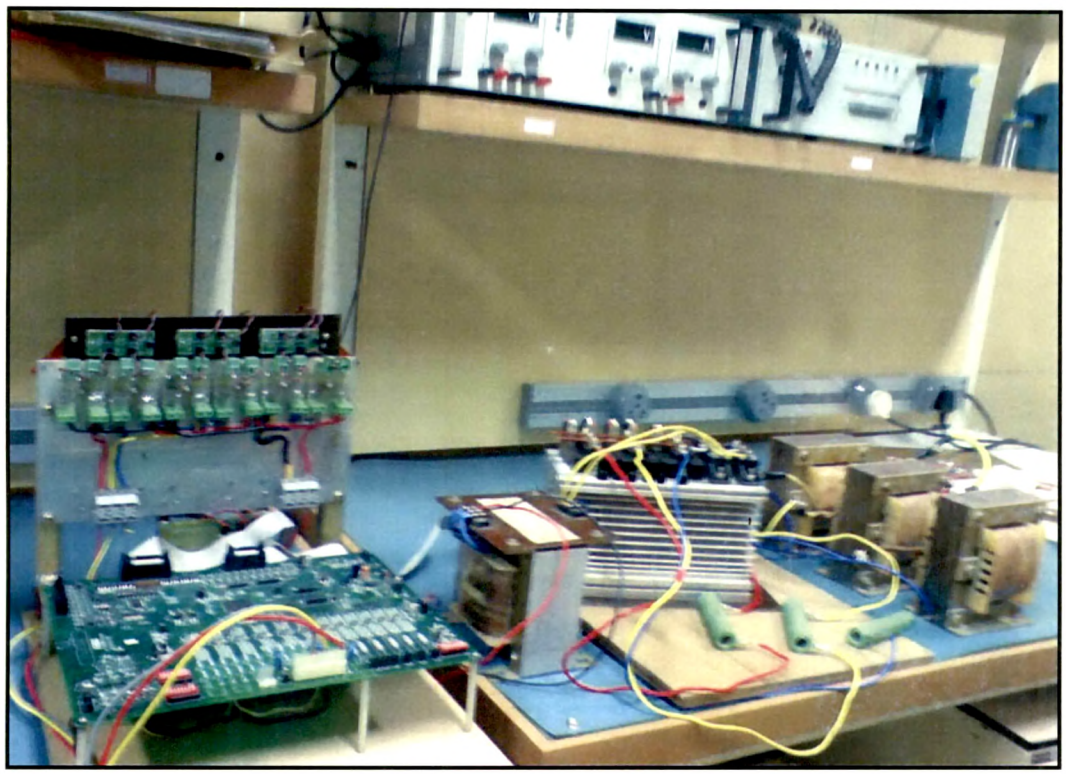


Fig 4.14 Pictorial view of the DSP based complete set-up

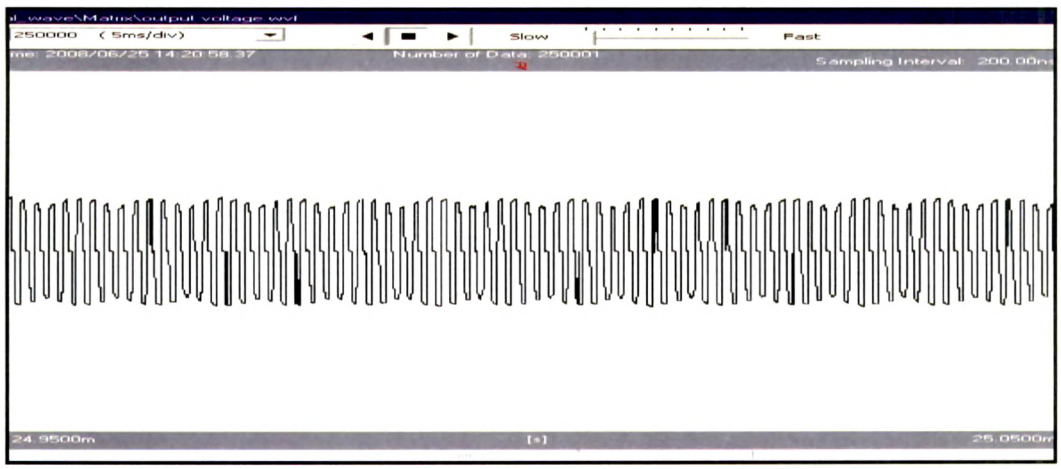
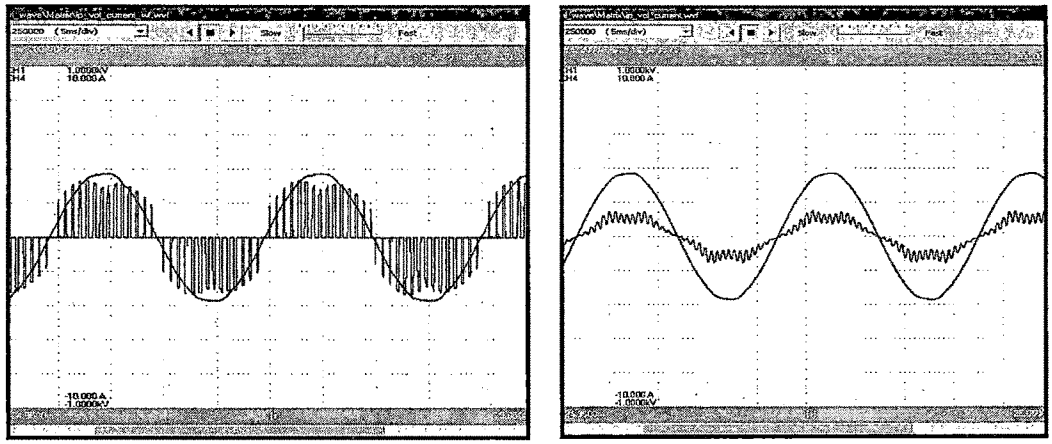


Fig 4.15 Output of the high frequency transformer.

A small LC filter is inserted in the power circuit at the input terminals to filter out the switching frequency ripple. The input voltage and current waveforms before and after the filter are shown in figure below.



(a) (b)

Fig 4.16. a) Input voltages and the input current without input ripple filter and b) Input voltages and the input current with input ripple filter

This high frequency voltage square pulses are then rectified into DC voltages using a high frequency bridge rectifier circuit developed using IGBTs. These DC voltages are the source for the three-phase bridge inverter circuit. The gate pulses for the inverter bridge are generated using the SPWM technique. The switching frequency of the carrier signal is set to be 3 kHz. And the modulation frequency is variable and can be settable as per user-defined frequency.

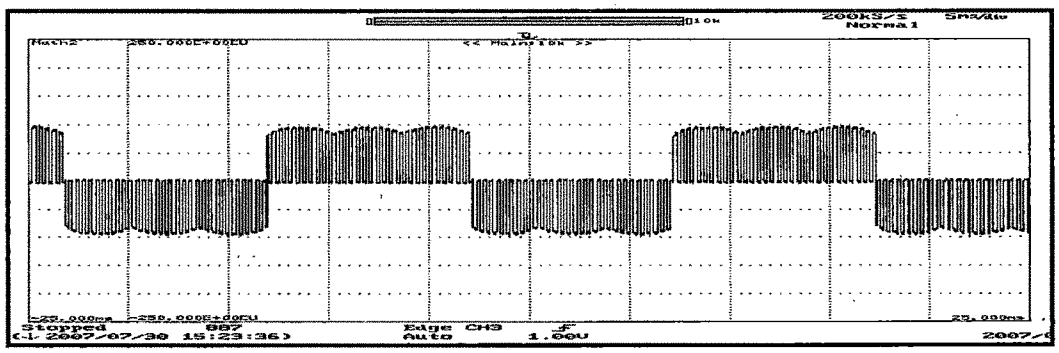


Fig 4.17 output line voltage for R-Y voltage

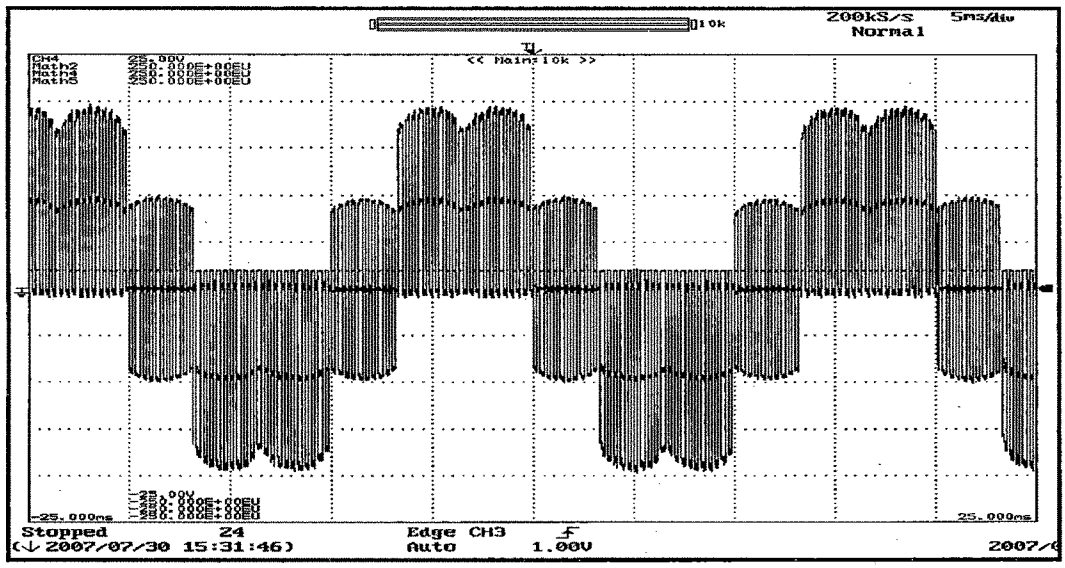


Fig 4.18 output phase R-N voltage

4.3 Conclusion

Practical implementation of Direct AC-AC converter is explained in detail. Implementations were initially done with analog ICs and then with the digital control using DSP. Experimental results verify the performance of the model proposed in chapter III.