APPENDIX C3: ZVT-Rectifier Program (open/closed loop)

```
.module boost;

#include <def2101.h>;

#define DD_CONST 0x5000
#define DQ_CONST 0x0600

#define IQREF_CST 0x0000
#define IDREF_CST 0x44f3

#define MAX_LENGTH_SQ 0x0a8f { 0.866^2 = 0.75 => 4.12 integer format }
                      { If this constant is changed       }
                      { scaling.dat has to be changed           }

#define SCALING_THRESHOLD  0x0200 { 0.125^2 of the full scale in  }
                         { 1.15. If threshold is changed }
                         { normal.dat must be changed        }

#define SWITCHING_THRESHOLD 0x051e { 0.2^2 of the full scale in 1.15 }

#define TRANSITION_LENGTH   0x04   { unit is double clock = 125 ns}
#define CHARGING_LENGTH     0x08
#define DISCHARGING_LENGTH  0x04
#define MINIMUM_LENGTH      0x07
#define CURRENT_LIMIT       0x6000 { 1.1/1.25 * 32768 is 0x7100 }

{ macro adjust is used to adjust      }
{ decimal point in 4.12 x 4.12 = 8.24 }

#define adjust SR = ashift MR1 by 4 (hi); \
            SR = SR or lshift MR0 by 4 (lo)

.const      CHARGE        = TRANSITION_LENGTH + CHARGING_LENGTH;
.const      DISCHARGE = TRANSITION_LENGTH + DISCHARGING_LENGTH;

.var/dm/ram/abs=0x0800   data;
.var/dm/ram/abs=0x0801   type;
.var/dm/ram/abs=0x0802   disable_altera_outputs;
.var/dm/ram/abs=0x0803   reserved;
.var/dm/ram/abs=0x0804   time1;
.var/dm/ram/abs=0x0805   time2;
.var/dm/ram/abs=0x0806   time3;
.var/dm/ram/abs=0x0807   time4;

.var/dm/ram/abs=0x0400   dac0;
.var/dm/ram/abs=0x0401   dac1;
.var/dm/ram/abs=0x0402   dac2;
.var/dm/ram/abs=0x0403   dac3;

.var/dm/ram/abs=0x0000   adcVa;
.var/dm/ram/abs=0x0001   adcVb;
.var/dm/ram/abs=0x0002   adcIa;
```

```
.var/dm/ram/abs=0x0003   adcIb;
.var/dm/ram/abs=0x0004   adcIdref;
.var/dm/ram/abs=0x0005   adcIqref;
.var/dm/ram/abs=0x0006   adc6;
.var/dm/ram/abs=0x0007   adc7;

{ Measured values }

.var/dm/ram   CosTheta;    { Channel 0 = Va = CosTheta }
.var/dm/ram   Vb;    { Channel 1 = Vb            }
.var/dm/ram   Ialpha;      { Channel 2 = Ia = Ialpha   }
.var/dm/ram   Ib;    { Channel 3 = Ib            }
.var/dm/ram   Idref;       { Channel 4 = Idref         }
.var/dm/ram   Iqref;       { Channel 5 = Iqref         }

{ Calculated values }

.var/dm/ram   SinTheta;
.var/dm/ram   Ic;
.var/dm/ram   Ibeta;
.var/dm/ram   Id;
.var/dm/ram   Iq;
.var/dm/ram   dd;
.var/dm/ram   dq;
.var/dm/ram   dalpha;
.var/dm/ram   dbeta;
.var/dm/ram   d1;
.var/dm/ram   d2;
.var/dm/ram   new_cycle_flag;
.var/dm/ram   offset;

{ Constant variables }

.var/dm/ram   OneOverSqrTwo;
.init       OneOverSqrTwo:         0x5a82; {32757 * 0.707106781 }
.var/dm/ram   OneOverTwoSqrThree;
.init       OneOverTwoSqrThree:    0x24f3; {32767 * 0.288675134 }
.var/dm/ram   OneOverSqrThree;
.init       OneOverSqrThree:       0x49e6; {32767 * 0.577350269 }
.var/dm/ram   OneOverSqrTwoSqrThree;
.init       OneOverSqrTwoSqrThree: 0x3441; {32767 * 0.40824829 }
.var/dm/ram   OneOverTwo;
.init       OneOverTwo:      0x3FFF; {32767 * 0.5 }
.var/dm/ram   SqrThreeOverTwo;
.init       SqrThreeOverTwo:       0x6ED9; {32767 * 0.866025403 }
.var/dm/ram   Kp;
.init       Kp:              0x1000; {Kp=0.5/2; Kpmax=1; 4.12 }
.var/dm/ram   wL;
.init       wL: 0; {0x0c10;}       { wL=2*Pi*60*2e-3=0.75; 4.12 }

{ Circular buffers and tables }

.var/dm/ram/circ   IaIn[3];
.init          IaIn: 0,0,0;
.var/dm/ram/circ   IaOut[2];
.init          IaOut: 0,0;

.var/dm/ram/circ   IbIn[3];
.init          IbIn: 0,0,0;
.var/dm/ram/circ   IbOut[2];
.init          IbOut: 0,0;
```

```
.var/dm/ram/circ   IdrefIn[3];
.init          IdrefIn: 0,0,0;
.var/dm/ram/circ   IdrefOut[2];
.init          IdrefOut: 0,0;

.var/dm/ram/circ   IqrefIn[3];
.init          IqrefIn: 0,0,0;
.var/dm/ram/circ   IqrefOut[2];
.init          IqrefOut: 0,0;

.var/dm/ram/circ   VaIn[3];
.init          VaIn: 0,0,0;
.var/dm/ram/circ   VaOut[2];
.init          VaOut: 0,0;

.var/dm/ram/circ   VbIn[3];
.init          VbIn: 0,0,0;
.var/dm/ram/circ   VbOut[2];
.init          VbOut: 0,0;

.var/dm/ram   VbInPtr;         { Pointers to circular buffers }
.init       VbInPtr:       ^Vbin;
.var/dm/ram   VbOutPtr;
.init       VbOutPtr:    ^VbOut;
.var/dm/ram   IaInPtr;
.init       IaInPtr:       ^IaIn;
.var/dm/ram   IaOutPtr;
.init       IaOutPtr:    ^IaOut;
.var/dm/ram   IbInPtr;
.init       IbInPtr:       ^IbIn;
.var/dm/ram   IbOutPtr;
.init       IbOutPtr:    ^IbOut;
.var/dm/ram   IdrefInPtr;
.init       IdrefInPtr:  ^IdrefIn;
.var/dm/ram   IdrefOutPtr;
.init       IdrefOutPtr: ^IdrefOut;
.var/dm/ram   IqrefInPtr;
.init       IqrefInPtr:  ^IqrefIn;
.var/dm/ram   IqrefOutPtr;
.init       IqrefOutPtr: ^IqrefOut;

{ This table contains swapping and waveform type information         }
{ Swaping means that time intervals d1 and d2 have to be exchanged }
{ Due to ZVT, there are three waveforms types:                     }
{     1. d2-zvt-d0-d1                                             }
{     2. d0-zvt-d1-d2                                             }
{     3. d0/2-zvt-d0/2-d1-d2                                      }
{ This table also contains information what type should be used in }
{ the specific case.                                              }

{ b#XYZ  }
{ X  = 1  d1 and d2 are swaped         }
{ X  = 0  d1 and d2 are not swaped }
{ YZ = 00 output waveform type 0   }
{ YZ = 01 output waveform type 1   }
{ YZ = 10 output waveform type 2   }

.var/dm/ram   case_table[36];
.init       case_table:
            b#110,       { Voltage sector 1, current case 1 }
```

```
            b#001,       { Voltage sector 1, current case 2 }
            b#101,       { Voltage sector 1, current case 3 }
            b#000,       { Voltage sector 1, current case 4 }
            b#100,       { Voltage sector 1, current case 5 }
            b#010,       { Voltage sector 1, current case 6 }
            b#101,       { Voltage sector 2, current case 1 }
            b#010,       { Voltage sector 2, current case 2 }
            b#001,       { Voltage sector 2, current case 3 }
            b#100,       { Voltage sector 2, current case 4 }
            b#110,       { Voltage sector 2, current case 5 }
            b#000,       { Voltage sector 2, current case 6 }
            b#001,       { Voltage sector 3, current case 1 }
            b#000,       { Voltage sector 3, current case 2 }
            b#010,       { Voltage sector 3, current case 3 }
            b#110,       { Voltage sector 3, current case 4 }
            b#101,       { Voltage sector 3, current case 5 }
            b#100,       { Voltage sector 3, current case 6 }
            b#010,       { Voltage sector 4, current case 1 }
            b#100,       { Voltage sector 4, current case 2 }
            b#000,       { Voltage sector 4, current case 3 }
            b#101,       { Voltage sector 4, current case 4 }
            b#001,       { Voltage sector 4, current case 5 }
            b#110,       { Voltage sector 4, current case 6 }
            b#000,       { Voltage sector 5, current case 1 }
            b#110,       { Voltage sector 5, current case 2 }
            b#100,       { Voltage sector 5, current case 3 }
            b#001,       { Voltage sector 5, current case 4 }
            b#010,       { Voltage sector 5, current case 5 }
            b#101,       { Voltage sector 5, current case 6 }
            b#100,       { Voltage sector 6, current case 1 }
            b#101,       { Voltage sector 6, current case 2 }
            b#110,       { Voltage sector 6, current case 3 }
            b#010,       { Voltage sector 6, current case 4 }
            b#000,       { Voltage sector 6, current case 5 }
            b#001;       { Voltage sector 6, current case 6 }

{******************** Interrupt vectors ************************}

    AR = 1;             { Reset Vector }
    DM(disable_altera_outputs) = AR;
    jump start;
    rti;

interrupt_two:                      { Irq2 is generated in altera.  }
    DM(new_cycle_flag) = AY1;       { It is used as switching       }
    rti;                        { cycle interrupt. When it comes}
    rti;                        { new_cycle_flag becomes           }
    rti;                        { different than zero.        }
                                { The program does not change   }
                                { AY1 (AY1=0x0080 all the time).}

    rti; rti; rti; rti;             { sport0 TX - unused }
    rti; rti; rti; rti;             { sport0 RX - unused }
    rti; rti; rti; rti;             { irq1 - unused      }
    rti; rti; rti; rti;             { irq0 - unused      }

timer_interrupt:
    AR = 1;                     { timer is used as a watch dog }
    DM(disable_altera_outputs) = AR;  { It disables altera outputs   }
    set flag_out;               { and halt DSP. Only reset can }
    idle;                       { wake up the DSP.           }
```

```
    jump timer_interrupt;

start:

    AR = 1;
    DM(disable_altera_outputs) = AR;  { disable outputs }

{******************** T I M E R   S T U F F ********************}

    AX0 = 0;
    DM(Tscale_Reg)  = AX0;   { 540 x 62.5ns = 33.75us  }

    AX0 = 540;
    DM(Tcount_Reg)  = AX0;   { Watchdog for 31.4 us    }

    AX0 = 540;
    DM(Tperiod_Reg) = AX0;

{******** S Y S T E M   A N D M E M O R Y   S T U F F *********}

    AX0=h#0052;
    DM(Dm_Wait_Reg)=AX0;   {1-2-2 DM wait states            }

    AX0=h#0028;
    DM(Sys_Ctrl_Reg)=AX0;  {SPORT0 disab, SPORT1 disab,        }
                    {use FI,FO,IRQ0,IRQ1,SCLK instead   }
                    {BOOT_PAGE=0, BMWAIT=5, PMWAIT=0    }

    IFC = 0x03f;    { Clear all interrupts just for any case   }
    IFC = 0x000;

    ICNTL=b#00111;  { Nesting disable, all ints edge triggered }

    IMASK=b#100001; { Interrupts IRQ2, SPORT0 TX, SPORT0 RX,   }
                {        SPORT1 TX, SPORT0 RX, Timer   }

    MSTAT=b#1001000; {Go mode, Timer disable, Frac, Sat mode,  }
                 {Ovr ltch dis, Bit rev dis, Prim regs     }


{************************ Initialization ************************}

    AR = DM(adcIqref);  { Start Channel 5 = Iqref }

    AY1 = 0x0080; { DO NOT CHANGE AY1 EVER ! }

    I0 = ^IaIn;   { Set all pointers }
    M0 = 1;
    L0 = 3;

    I1 = ^IaOut;
    M1 = 1;
    L1 = 2;

    I2 = ^VaIn;
    M2 = 1;
    L2 = 3;

    I3 = ^VaOut;
    M3 = 1;
    L3 = 2;
```

```
    I4 = 0; { spare so far }
    M4 = 0;
    L4 = 0;

    I5 = ^b1000;
    M5 = 1;
    L5 = 3;

    I6 = ^a1000;
    M6 = 1;
    L6 = 2;

    I7 = 0;    { Misc. pointer                   }
    M7 = 1;    { It is loaded before each access }
    L7 = 0;

    nop;       { Wait to be sure that conversion is over }
    nop;

{-------------------- Read and start conversion ------------------}

    AR = DM(adcIdref);     { Read Ch 5 = Iqref, Start Ch 4 = Idref  }

    AR = AR xor AY1;    { AR = AR xor 0x0080, adjust sign bit }
    SR = lshift AR by 8 (lo);
                          { Digital filter }
    I0 = DM(IqrefInPtr);
    I1 = DM(IqrefOutPtr);

{    DM(I0,M0) = SR0;
    MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (rnd);
    MY0 = 0x7fff;
    MR = MR + MX0 * MY0 (rnd);
    DM(I1,M1) = MR1;

    DM(Iqref) = MR1;

    DM(IqrefInPtr)  = I0;
    DM(IqrefOutPtr) = I1;}

DM(Iqref)=SR0;
MR1=SR0;

{-----------------------------------------------------------------}

    CNTR = 12;
    do loop0 until ce;
loop0:
    nop;

{-------------------- Read and start conversion ------------------}

    AR = DM(adcVa);     { Read Ch 4 = Idref, Start Ch 0 = Va  }
    AR = AR xor AY1;    { AR = AR xor 0x0080, adjust sign bit }
    SR = lshift AR by 8 (lo);
```

```
                          { Digital filter }
{    I0 = DM(IdrefInPtr);
    I1 = DM(IdrefOutPtr);

    DM(I0,M0) = SR0;
    MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (rnd);
    MY0 = 0x7fff;
    MR = MR + MX0 * MY0 (rnd);
    DM(I1,M1) = MR1;

    DM(Idref) = MR1;

    DM(IdrefInPtr)  = I0;
    DM(IdrefOutPtr) = I1;}

DM(Idref)=SR0;
MR1=SR0;

{--------------------------------------------------------------------}

    CNTR = 8;
    do loop1 until ce;
loop1:
    nop;

{*****************************************************************}
{*                                                               *}
{* If input voltages drop below the scaling threshold then the      *}
{* program exit the main loop and waits inside voltage_wait_loop  *}
{*                                                               *}
{*****************************************************************}

voltage_wait_loop:

AR=0;
DM(data)=AR;
DM(data)=AR;
DM(data)=AR;
DM(data)=AR;
DM(data)=AR;
DM(data)=AR;
DM(type)=AR;
AR=30;
DM(time1)=AR;
DM(time2)=AR;
DM(time3)=AR;
DM(time4)=AR;


{------------------- Read and start conversion --------------------}

    AR = DM(adcVb);     { Read Ch 0 = Va, Start Ch 1 = Vb     }
    AR = AR xor AY1;    { AR = AR xor 0x0080, adjust sign bit }
    SR = lshift AR by 8 (lo);
                               { Digital filter }
{    DM(I2,M2) = SR0;
```

```
    MX0 = DM(I2,M2), MY0 = PM(I5,M5);
    MR = MX0 * MY0 (ss), MX0 = DM(I2,M2), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I2,M2), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I3,M3), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I3,M3), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (rnd);
    MY0 = 0x7fff;
    MR = MR + MX0 * MY0 (rnd);
    DM(I3,M3) = MR1;

    DM(CosTheta) = MR1;}

DM(CosTheta)=SR0;
MR1=SR0;


{-------------------------------------------------------------------}

    CNTR = 15;
    do loop2 until ce;

loop2: nop;

{-------------------- Read and start conversion -----------------}

    AR = DM(adcVa);     { Read Ch 1 = Vb, Start Ch 0 = Va     }
    AR = AR xor AY1;    { AR = AR xor 0x0080, adjust sign bit }
    SR = lshift AR by 8 (lo);

    I0 = DM(VbInPtr);
    I1 = DM(VbOutPtr);
                                { Digital filter }
{    DM(I0,M0) = SR0;
    MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (rnd);
    MY0 = 0x7fff;
    MR = MR + MX0 * MY0 (rnd);
    DM(I1,M1) = MR1;

    DM(Vb) = MR1;

    DM(VbInPtr)  = I0;
    DM(VbOutPtr) = I1;}

DM(Vb)=SR0;
MR1=SR0;


{-------------------------------------------------------------------}
{ ******************* Calculate SinTheta ********************** }
{ ******* SinTheta = 2*[CosTheta/(2*Sqrt(3)) + Vb/Sqrt(3)] ****** }

    MY0 = DM(OneOverSqrThree);
    MX1 = DM(CosTheta);
    MY1 = DM(OneOverTwoSqrThree);

    MR      = MR1 * MY0(ss);       { MR= Vb/Sqrt(3) }
```

```
    MR      = MR + MX1 * MY1(rnd); { MR= MR + CosTheta/(2*Sqrt(3) }
    if MV sat MR;
    AY0 = MR1;
    AR      = MR1 + AY0;             { AR = 2 * MR }
    DM(SinTheta) = AR;



{ ************* SinTheta and CosTheta normalization ************* }

    MX0 = DM(CosTheta);
    MY0 = MX0;
    MX1 = DM(SinTheta);
    MR = MX0 * MY0 (ss), MY0 = MX1; { MR = CosTheta^2 }
    MR = MR + MX1 * MY0 (rnd);          { MR = CosTheta^2 + SinTheta^2 }
    if MV sat MR;

    AY0 = SCALING_THRESHOLD;
    AR = MR1 - AY0, AX0 = MR1;         { AR = offset in normal_table }
    if lt jump voltage_wait_loop;  { if MR1 less than scaling      }
                           { threshold go back to      }
                           { voltage_wait_loop begining  }

    SR = lshift AR by -6 (lo);         { else take only upper 9 bits }
    AY0 = ^normal_table;         { as offset in normal table       }
    AR = SR0 + AY0;           { AR = correction factor address }
    I7 = AR;
    MY0 = PM(I7,M7);         { read in MY0 (multiplier factor)/8 }

    MR = MX0 * MY0 (ss);       { CosTheta * (Multiplier factor/8) }
    SR = ashift MR1 by 2 (hi);            { shift by 2 means times by 4
}
    SR = SR or lshift MR0 by 2 (lo);
    AY0 = SR1;
    AR = SR1 + AY0;            { then adding result to itself means }
    DM(CosTheta) = AR;         { multiply by 2 ( total by 8 )         }
                         { Don't forget that ALU saturates !  }

    MR = MX1 * MY0 (ss);       { The same thing is done for SinTheta }
    SR = ashift MR1 by 2 (hi);
    SR = SR or lshift MR0 by 2 (lo);
    AY0 = SR1;
    AR = SR1 + AY0;
    DM(SinTheta) = AR;

    AY0 = SWITCHING_THRESHOLD;        { if CosTheta^2+SinTheta^2 is less
}
    AR = AX0 - AY0;             { than switching_threshold go to   }
    if lt jump voltage_wait_loop; { the voltage_wait_loop beginning  }

{------------------- Read and start conversion -------------------}

    AR = DM(adcVb);     { Read Ch 0 = Va, Start Ch 1 = Vb     }
    AR = AR xor AY1;    { AR = AR xor 0x0080, adjust sign bit }
    SR = lshift AR by 8 (lo);
                               { Digital filter }
{    DM(I2,M2) = SR0;
    MX0 = DM(I2,M2), MY0 = PM(I5,M5);
    MR = MX0 * MY0 (ss), MX0 = DM(I2,M2), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I2,M2), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I3,M3), MY0 = PM(I6,M6);
```

```
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I3,M3), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (rnd);
    MY0 = 0x7fff;
    MR = MR + MX0 * MY0 (rnd);
    DM(I3,M3) = MR1;

    DM(CosTheta) = MR1;}

DM(CosTheta)=SR0;
MR1=SR0;


{-------------------------------------------------------------------}

    CNTR = 5;
    do loop3 until ce;

loop3:      nop;

    AR = 0;
    DM(new_cycle_flag) = AR;       { reset the flag }
    DM(disable_altera_outputs) = AR; { enable outputs }

sync:
    AR = DM(new_cycle_flag);       { wait irq2 to sinchronize   }
    AR = pass AR;            { begining of the main loop. }
    if eq jump sync;

{    ena timer; }              { enable watch_dog timer }

{ ********************************************************** }
{ ****************** End of Initialization ******************* }
{ ********************************************************** }

begin:
{-------------------- Read and start conversion ----------------}

    AR = DM(adcIa);     { Read Ch 1 = Vb, Start Ch 2 = Ia     }
    AR = AR xor AY1;    { AR = AR xor 0x0080, adjusy sign bit }
    SR = lshift AR by 8 (lo);

    I0 = DM(VbInPtr);
    I1 = DM(VbOutPtr);
                              { Digital filter }
{    DM(I0,M0) = SR0;
    MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (rnd);
    MY0 = 0x7fff;
    MR = MR + MX0 * MY0 (rnd);
    DM(I1,M1) = MR1;

    DM(Vb) = MR1;

    DM(VbInPtr)  = I0;
    DM(VbOutPtr) = I1;}

DM(Vb)=SR0;
```

```
MR1=SR0;


{-------------------------------------------------------------------}

{ ******************* Calculate SinTheta ********************** }

{ ******* SinTheta = 2*[CosTheta/(2*Sqrt(3)) + Vb/Sqrt(3)] ****** }

    MY0 = DM(OneOverSqrThree);
    MX1 = DM(CosTheta);
    MY1 = DM(OneOverTwoSqrThree);

    MR      = MR1 * MY0(ss);       { MR= Vb/Sqrt(3) }
    MR      = MR + MX1 * MY1(rnd); { MR= MR + CosTheta/(2*Sqrt(3) }
    if MV sat MR;
    AY0 = MR1;
    AR      = MR1 + AY0;             { AR = 2 * MR }
    DM(SinTheta) = AR;



{ ********************** Update watchdog ********************** }

    AR = 540;
    DM(Tcount_Reg)  = AR;

    AR = 0;
    DM(new_cycle_flag) = AR; { reset new_cycle_flag }

{-------------------- Read and start conversion -----------------}

    AR = DM(adcIb);     { Read Ch 2 = Ia, Start Ch 3 = Ib }

DM(dac2) = AR;

    AR = AR xor AY1;
    SR = lshift AR by 8 (lo);
                                { Digital filter }
    I0 = DM(IaInPtr);
    I1 = DM(IaOutPtr);

    DM(I0,M0) = SR0;
    MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (rnd);
    MY0 = 0x7fff;
    MR = MR + MX0 * MY0 (rnd);
    DM(I1,M1) = MR1;

    DM(Ialpha) = MR1;

    DM(IaInPtr)  = I0;
    DM(IaOutPtr) = I1;

SR = lshift MR1 by -8 (hi);
AR = SR1 xor AY1;
DM(dac0) = AR;
```

```
    AY0 = CURRENT_LIMIT;
    AR = abs MR1;
    AR = AR - AY0;
    if gt jump shutdown;

{-------------------------------------------------------------------}

{ ************* SinTheta and CosTheta normalization ************* }

    MX0 = DM(CosTheta);
    MY0 = MX0;
    MX1 = DM(SinTheta);
    MR = MX0 * MY0 (ss), MY0 = MX1; { MR = CosTheta^2 }
    MR = MR + MX1 * MY0 (rnd);          { MR = CosTheta^2+SinTheta^2 }
    if MV sat MR;

    AY0 = SCALING_THRESHOLD;
    AR = MR1 - AY0, AX0 = MR1;         { AR = offset in normal_table }

    if lt jump start;            { if CosTheta^2+SinTheta^2 is }
    SR = lshift AR by -6 (lo);         { less than scaling_threshold }
                           { jump to the very begining       }
    AY0 = ^normal_table;
    AR = SR0 + AY0;           { AR = correction factor address }
    I7 = AR;
    MY0 = PM(I7,M7);          { MY0 = correction / 8 }

    MR = MX0 * MY0 (ss);             { This part is the same as }
    SR = ashift MR1 by 2 (hi);             { in initialization part
}
    SR = SR or lshift MR0 by 2 (lo);  { of the program             }
    AY0 = SR1;
    AR = SR1 + AY0;
    DM(CosTheta) = AR;

    MR = MX1 * MY0 (ss);
    SR = ashift MR1 by 2 (hi);
    SR = SR or lshift MR0 by 2 (lo);
    AY0 = SR1;
    AR = SR1 + AY0;
    DM(SinTheta) = AR;

{-------------------- Read and start conversion -----------------}

    I0 = DM(IbInPtr);
    I1 = DM(IbOutPtr);

    AR = DM(adcIdref);  { Read Ch 3 = Ib, Start Ch 4 = Idref }

    AR = AR xor AY1;
    SR = lshift AR by 8 (lo);
                               { Digital filter }
    DM(I0,M0) = SR0;
    MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (rnd);
```

```
    MY0 = 0x7fff;
    MR = MR + MX0 * MY0 (rnd);
    DM(I1,M1) = MR1;

    DM(Ib) = MR1;

    DM(IbInPtr)  = I0;
    DM(IbOutPtr) = I1;


    AY0 = CURRENT_LIMIT;
    AR = abs MR1;
    AR = AR - AY0;
    if gt jump shutdown;

{-----------------------------------------------------------------}

{ ********************* Calculate Ic  ************************** }

    AY0 = DM(Ialpha);
    AR = MR1 + AY0;        { AR = Ib + Ialpha    }
    AR = -AR;              { AR = -(Ib + Ialpha) }
    DM(Ic) = AR;

{ ******************** Calculate Ibeta    ************************ }
{ ************ Ibeta = Ib/Sqrt(3) - Ic/Sqrt(3) ***************** }

                         { MR1 = Ib; AR = Ic    }
    MY0 = DM(OneOverSqrThree);
    MR = MR1 * MY0 (ss);       { MR = Ib/Sqrt(3)      }
    MR = MR - AR * MY0 (rnd);  { MR = MR - Ic/Sqrt(3) }
    if MV sat MR;
    DM(Ibeta) = MR1;
    MY0 = MR1;                 { MY0 = Ibeta }

{ ****************** Calculate Id and Iq ********************** }
{ ******** Id = Ialpha * CosTheta + Ibeta * SinTheta ********** }

                        { MX1 = CosTheta, MY0 = Ibeta }
    MY1 = DM(Ialpha);
    MX0 = DM(SinTheta);
    MX1 = DM(CosTheta);
    MR = MX1 * MY1 (ss);       { MR = CosTheta * Ialpha     }
    MR = MR + MX0 * MY0 (rnd); { MR = MR + SinTheta * Ibeta }
    if MV sat MR;
    DM(Id) = MR1;



{ ********** Iq = - Ialpha * SinTheta + Ibeta * CosTheta *******}

    MR = MX1 * MY0 (ss);      { MR = CosTheta * Ibeta   }
    MR = MR - MX0 * MY1(rnd); { MR = MR - SinTheta * Ialpha }
    if MV sat MR;
    DM(Iq) = MR1;


{ SR = lshift MR1 by -5 (hi);
 AR = SR1 xor AY1;
 DM(dac2) = AR;}
```

```
{-------------------- Read and start conversion -----------------}

    AR = DM(adcIqref);     { Read Ch 4 = Idref, Start Ch 5 = Iqref}
    AR = AR xor AY1;    { AR = AR xor 0x0080, adjust sign bit }
    SR = lshift AR by 8 (lo);
                               { Digital filter }
{    I0 = DM(IdrefInPtr);
    I1 = DM(IdrefOutPtr);

    DM(I0,M0) = SR0;
    MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (rnd);
    MY0 = 0x7fff;
    MR = MR + MX0 * MY0 (rnd);
    DM(I1,M1) = MR1;

    DM(Idref) = MR1;

    DM(IdrefInPtr)  = I0;
    DM(IdrefOutPtr) = I1;}

DM(Idref)=SR0;


{**************** Regulator starts here !!! ********************}
{*                                                             *}
{*        Arithmetics is changed from 1.15 to 4.12           *}
{*            ADSP2101 int type, not fract                   *}
{*                                                             *}
{*************************************************************}

    MSTAT=b#1111000; { disable fractional arithmetics     }
                { Go mode, Timer en, INT, Sat mode,    }
                { Ovr ltch dis, Bit rev dis, Prim regs }



    MY0 = DM(Kp);
    MY1 = DM(wL);

    SR = ashift MR1 by -3 (hi);   { MR1 = frac Iq   }
    MX0 = SR1;                  { MX0 = (4.12) Iq }

    SI = DM(Id);
    SR = ashift SI by -3 (hi);        { SR1 = (4.12) Id }

    MX1 = DM(Iqref);


    MR = MX0 * MY0 (ss);            { MR = Iq * Kp         }
    MR = MR - MX1 * MY0(ss), MX1=SR1; { MR = MR - Iqref * Kp }
    MR = MR - MX1 * MY1 (rnd);            { MR = MR - Id * wL    }
    adjust;
    DM(dq) = SR1;

    SI = DM(Idref);
```

```
{SI=IDREF_CST;}

    SR = ashift SI by -3 (hi); { SR1 = (4.12) Idref }

    MR = MX1 * MY0 (ss);       { MR = Id * Kp         }
    MR = MR - SR1 * MY0 (ss);  { MR = MR - Idref * Kp }
    MR = MR + MX0 * MY1 (rnd); { MR = MR + Iq * wL    }
    adjust;
    DM(dd) = SR1;

{ ****** Limiting (dd,dq) => dd^2 + dq^2 < MAX_LENGTH_SQ ******** }

limiting:
    MY0 = SR1;                    { MY0 = dd }
    AR = DM(dq);
    MR = SR1 * MY0 (ss), MY1 = AR;   { MR1 = dd^2, MY1 = dq  }
    MR = MR + AR * MY1 (rnd);      { MR  = dd^2 + dq^2     }
    adjust;                   { SR1 = dd^2 + dq^2     }

    AY0 = MAX_LENGTH_SQ;
    AR      = SR1 - AY0;              { AR = overlength in 4.12 }
AF = pass AR;
    if le jump no_scaling;

scaling:
{    jump duty_limit;}

{MR1 =0x00f0;
DM(dac0)=MR1;}

    SR = lshift AR by -6 (lo); { SR0 = offset in the table }
    AY0 = ^scale_table;        { upper 8 bits are ofset    }
    AR = SR0 + AY0;
    I7 = AR;
    MX0 = PM(I7,M7);           { Correction factor in 1.15 }

    MR = MX0 * MY0 (ss);       { MR = (Correction * dd) in 5.27 }
    adjust;              { SR1 = dd in 1.15           }
    DM(dd) = SR1;

    MR = MX0 * MY1 (ss);       { MR = (Correction * dq) in 5.27 }
    adjust;              { SR1 = dq in 1.15           }
    DM(dq) = SR1;

    jump DdDq_to_DalphaDbeta;

no_scaling:

{MR1 =0x0080;
DM(qedac0)=MR1;}

    SI = DM(dd);         { convert dd and dq to 1.15 }
    SR = lshift SI by 3 (hi);
    DM(dd) = SR1;        { SR1 = frac dd }

    SI = DM(dq);
    SR = lshift SI by 3 (hi);
    DM(dq) = SR1;        { SR1 = frac dq }
```

```
{ ************************************************************* }
{ *            The end of the regulator                 * }
{ *            Go back to fractional type               * }
{ ************************************************************* }

DdDq_to_DalphaDbeta:

    MSTAT=b#1101000;   {enable fractional arithmetics      }
                   {Go mode, timer en, FRAC, Sat mode,   }
                   {Ovr ltch dis, Bit rev dis, Prim regs }




{CNTR=30;
do looptm until ce;
looptm:
nop;}



{-------------------- Read and start conversion ------------------}

    AR = DM(adcVa);     { Read Ch 5 = Iqref, Start Ch 0 = Va  }
    AR = AR xor AY1;    { AR = AR xor 0x0080, adjust sign bit }
    SR = lshift AR by 8 (lo);
                          { Digital filter }
    I0 = DM(IqrefInPtr);
    I1 = DM(IqrefOutPtr);

{    DM(I0,M0) = SR0;
    MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I0,M0), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I1,M1), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (rnd);
    MY0 = 0x7fff;
    MR = MR + MX0 * MY0 (rnd);
    DM(I1,M1) = MR1;

    DM(Iqref) = MR1;

    DM(IqrefInPtr)  = I0;
    DM(IqrefOutPtr) = I1;}

DM(Iqref)=SR0;
MR1=SR0;



{-------------------------------------------------------------------}

{-------------------------------------------------------------------}

{ ******** Conversion from (dd,dq) to (dalpha,dbeta) *********** }

{ ********** dalpha = dd * CosTheta - dq * SinTheta ************ }

    MY0 = DM(CosTheta);
```

```
    MY1 = DM(SinTheta);

    MX0 = DM(dd);
    MX1 = DM(dq);

{MX0 = DD_CONST;
MX1 = DQ_CONST;}

{MX0=DM(Idref);
MX1=DM(Iqref);}

    MR      = MX0 * MY0(ss);    { MR = dd * CosTheta }
    MR      = MR - MX1 * MY1 (rnd);   { MR = MR - dq * SinTheta }
    DM(dalpha) = MR1;

{ ********** dbeta = dd * SinTheta + dq * CosTheta ************ }

    MR      = MX0 * MY1 (ss);   { MR = dd * SinTheta }
    MR      = MR + MX1 * MY0 (rnd);   { MR = MR + dq * CosTheta }
    DM(dbeta) = MR1;

{ ******* Conversion (dalpha,dbeta) to (d1, d2, sec #) ******** }
{ *                                                  * }
{ * Please see the flowchart of conversion algorhitm        * }
{ *                                                    * }
{ ****************************************************** }

    MY0 = DM(dalpha);          { MR1 = dbeta }

    AX1 = MR1;                 { AX1 = dbeta }
    MR = MR1 * MY0 (rnd);      { MR = dalpha * dbeta }
    AR = pass MR1, AX0 = MY0;  { Check sign, AX0 = dalpha }
    if lt jump not_same_sign;

    same_sign:
      AF = abs AX0;              {AF = abs(dalpha) }
      AR = abs AX1;              {AR = abs(dbeta)  }
      MY0 = DM(OneOverSqrThree);
      MR = AR * MY0 (rnd);       {MR1=1/Sqrt(3) * abs(dbeta)}
      AR = MR1 - AF, AY0 = MR1;  {AY0=1/Sqrt(3) * abs(dbeta)}
      SR0 = AR;          {SR0=1/Sqrt(3) * abs(dbeta) -}
                         {   -abs(dalpha)       }

      if ge jump Sqr3dalpha_lt_dbeta_same_sign;

          AR = MR1 + AY0;    { AR = 2/Sqrt(3) * abs(dbeta)  }

          DM(d2) = AR;  { d2 = 2/Sqrt(3) * abs(dbeta)       }
          AR = - SR0;
          DM(d1) = AR;  {d1=abs(dalpha)-1/Sqrt(3)*abs(dbeta)}

          AF = pass AX0;
          if lt jump sector_5;

          sector_2:              { dalpha > 0, dbeta > 0 }
            MY0 = 1;       { SecNumber = 1 }
            jump continue;

          sector_5:              { dalpha < 0, dbeta < 0 }
            MY0 = 4;       { SecNumber = 4 }
            jump continue;
```

```
  Sqr3dalpha_lt_dbeta_same_sign:

      AR = MR1+AF;  {AR=1/Sqrt(3)*abs(dbeta)+abs(dalpha)}

      DM(d1) = AR;  {d1=1/Sqrt(3)*abs(dbeta)+abs(dalpha)}
      DM(d2) = SR0; {d2=1/Sqrt(3)*abs(dbeta)-abs(dalpha)}

      AF = pass AX1;
      if lt jump sector_6;

      sector_3:              { dalpha > 0, dbeta > 0 }
        MY0 = 2;
        jump continue;

      sector_6:              { dalpha < 0, dbeta < 0 }
        MY0 = 5;       { SecNumber = 5 }
        jump continue;

not_same_sign:
  AF = abs AX0;              { AF = abs(dalpha) }
  AR = abs AX1;              { AR = abs(dbeta)  }
  MY0 = DM(OneOverSqrThree);
  MR = AR * MY0 (rnd);       {MR1=1/Sqrt(3) * abs(dbeta)}
  AR = MR1 - AF, AY0 = MR1;  {AY0=1/Sqrt(3) * abs(dbeta)}
  SR0 = AR;          {SR0=1/Sqrt(3) * abs(dbeta) -}
                     {   -      abs(dalpha)       }

  if ge jump Sqr3dalpha_lt_dbeta_not_same_sig;

      AR = MR1 + AY0;    { AR  = 2/Sqrt(3) * abs(dbeta) }

      DM(d1) = AR;       { d1 = 2/Sqrt(3) * abs(dbeta)  }
      AR = - SR0;
      DM(d2) = AR; {d2=abs(dalpha)-1/Sqrt(3)*abs(dbeta) }

      AF = pass AX0;
      if lt jump sector_4;

      sector_1:              { dalpha > 0, dbeta < 0 }
        MY0 = 0;         { SecNumber = 0 }
        jump continue;

      sector_4:              { dalpha < 0, dbeta > 0 }
        MY0 = 3;         { SecNumber = 3 }
        jump continue;

  Sqr3dalpha_lt_dbeta_not_same_sig:

      AR = MR1+AF; {AR=1/Sqrt(3)*abs(dbeta)+abs(dalpha)}

      DM(d2) = AR; {d2=1/Sqrt(3)*abs(dbeta)+abs(dalpha)}
      DM(d1) = SR0;{d1=1/Sqrt(3)*abs(dbeta)-abs(dalpha)}

      AF = pass AX1;
      if ge jump sector_3a;
      sector_6a:             { dalpha > 0, dbeta < 0 }
        MY0 = 5;         { SecNumber = 5 }
        jump continue;

      sector_3a:             { dalpha < 0, dbeta > 0 }
```

```
            MY0 = 2;         { SecNumber = 2 }
continue:

    AF = pass 0;     { Based on the directions of currents  }
    AX0 = 0x02;      { Ic, Ib, and Ia, AF contains a number }
    AR = DM(Ic);     { between 1 and 6.                     }
    AR = pass AR;
    if ge AF = AF + 1;     {  Ia Ib Ic -> AF }

    AR = DM(Ib);     {   +  +  -      6  }
    AR = pass AR;    {   +  -  +      5  }
    if ge AF = AX0 + AF;   {   +  -  -      4  }
                     {   -  +  +      3  }
    AX0 = 4;                {   -  +  -      2  }
    AR = DM(Ialpha);        {   -  -  +      1  }
    AR = pass AR;
    if ge AF = AX0 + AF;   { This number is used to find offset in }
                     { the case_table                       }
    AF = AF - 1;

sequence:
    MX0 = 6;                    { MY0 = SecNumber                 }
    MR = MX0 * MY0 (ss);        { MR = SecNumber * 6              }
    SR = lshift MR0 by -1(lo); { Cancel automatic frac adjust          }
    AR = SR0 + AF;              { offset = 6*SecNumber + current_case }
    DM(offset) = AR;

    AY0 = ^case_table;
    AR = AR + AY0;
    I7 = AR;             { I7 = address in the case_table    }
    AX1 = DM(I7,M7);    { AX1 stores read value from case_table }
    AY0 = 0x04;   { isolate swapping bit              }
    AR = AX1 and AY0;
    if eq jump check_waveform_type; { if it is 0 then no swapping }

swap:
    AR = DM(d1);       { swap d1 and d2 }
    AX0 = DM(d2);
    DM(d1) = AX0;
    DM(d2) = AR;

check_waveform_type:
    AY0 = 0x03;       { isolate waveform type          }
    AR = AX1 and AY0; { AX1 = entry from the case table  }
    DM(type) = AR;    { load altera type reg by waveform_type  }

    AY0 = 0x01;    { determine waveform type and jump on }
    AR = AR - AY0;       { apropriate label                  }
    if eq jump waveform_one;
    AR = AR - AY0;
    if eq jump waveform_two;

{ ****************** Load time registers ******************* }

{************************************************************}
{*                                                 *}
{* Waveform zero:                                  *}
{*                                                 *}
{* | d2out     | CHARGE    | DISCHARGE | dzero    | d1out    |*}
{* | time1 reg | time2 reg | time3 reg | time4 reg| leftover |*}
{* |                                               |*}
```

```
{* |<-------------- switching cycle (256 clocks)------------>|*}
{*                                                           *}
{***********************************************************}

waveform_zero:
    AY0 = MINIMUM_LENGTH;

    SI = DM(d2);
    SR = lshift SI by -7 (hi); { d2out = d2 >> 7 }
    AR = SR1 - AY0;
    if ge jump zero1;
      SR1 = AY0;    { SR1 = d2out or MINIMUM_LENGTH }
zero1:
    DM(time1) = SR1;
    AF = pass SR1;

    AR = CHARGE;
    DM(time2) = AR;
    AF = AR + AF;

    AR = DISCHARGE;
    DM(time3) = AR;
    AF = AR + AF;

    SI = DM(d1);
    SR = lshift SI by -7 (hi); { d1out = d1 >> 7 }
    AR = SR1 - AY0;
    if ge jump zero2;
      SR1 = AY0;
zero2:
    AF = SR1 + AF;
    AR = 255;
    AR = AR - AF;
    DM(time4) = AR;

    jump label1;

{************************************************************}
{*                                                          *}
{* Waveform one:                                            *}
{*                                                          *}
{* | CHARGE    | DISCHARGE | d1out    | d2out     | dzero    | *}
{* | time1 reg | time2 reg | time3 reg| time4 reg | leftover | *}
{* |                                              | *}
{* |<-------------- switching cycle (256 clocks)------------>| *}
{*                                                          *}
{************************************************************}

waveform_one:
    AY0 = MINIMUM_LENGTH;

    AR = CHARGE;
    DM(time1) = AR;

    AR = DISCHARGE;
    DM(time2) = AR;

    SI = DM(d1);
    SR = lshift SI by -7 (hi); { d1out = d1 >> 7 }
    AR = SR1 - AY0;
    if ge jump one1;
```

```
      SR1 = AY0;
one1:
    DM(time3) = SR1;
    SI = DM(d2);
    SR = lshift SI by -7 (hi); { d2out = d2 >> 7 }
    AR = SR1 - AY0;
    if ge jump one2;
      SR1 = AY0;
one2:
    DM(time4) = SR1;
    jump label1;

{*************************************************************}
{*                                                           *}
{* Waveform two:                                             *}
{*                                                           *}
{* |d0/2     |CHARGE   |DISCHARGE|d0/2       |d1out    |d2out   | *}
{* |time1 reg|time2 reg|time3 reg|time1 reg|time4 reg|leftover| *}
{* |                                                         | *}
{* |<------------ switching cycle (256 clocks) -------------->| *}
{*                                                           *}
{*************************************************************}

waveform_two:
    AY0 = MINIMUM_LENGTH;

    SI = DM(d1);
    SR = lshift SI by -7 (hi); { d1out = d1 >> 7 }
    AR = SR1 - AY0;
    if ge jump two1;
       SR1 = AY0;
two1:
    DM(time4) = SR1;
    AF = pass SR1;

    AR = CHARGE;
    DM(time2) = AR;
    AF = AR + AF;

    AR = DISCHARGE;
    DM(time3) = AR;
    AF = AR + AF;

    SI = DM(d2);
    SR = lshift SI by -7 (hi); { d2out = d2 >> 7 }
    AR = SR1 - AY0;
    if ge jump two2;
      SR1 = AY0;
two2:
    AF = SR1 + AF;
    AR = 255;
    AR = AR - AF;

    SR = lshift AR by -1 (hi);      { Devide by 2 }
    DM(time1) = SR1;

label1:


{ ******** Calculating the entry point to the sector's  ********* }
{ ************************ switching table ********************** }
```

```
    AY0 = ^entry_points; { entry_points table contains begining }
    MR0 = DM(offset);    { addresses of the switching sequences }
    AR = MR0 + AY0;
    I7 = AR;          { I7 = address of the begining address     }
    AR = PM(I7,M7);
    I7 = AR;      { I7 = begining address of the switching sequence }

{ ************* Coping data into the altera buffer  ************* }

    AR = PM(I7,M7);   { load altera shift registers by }
    DM(data) = AR;    { switching sequence             }
    AR = PM(I7,M7);
    DM(data) = AR;
    AR = PM(I7,M7);
    DM(data) = AR;
    AR = PM(I7,M7);
    DM(data) = AR;
    AR = PM(I7,M7);
    DM(data) = AR;
    AR = PM(I7,M7);
    DM(data) = AR;

{ ******************** wait new switching cycle ***************** }

{-------------------- Read and start conversion ------------------}

    AR = DM(adcVb);     { Read Ch 0 = Va, Start Ch 1 = Vb     }
    AR = AR xor AY1;    { AR = AR xor 0x0080, adjust sign bit }
    SR = lshift AR by 8 (lo);
                              { Digital filter }
{    DM(I2,M2) = SR0;
    MX0 = DM(I2,M2), MY0 = PM(I5,M5);
    MR = MX0 * MY0 (ss), MX0 = DM(I2,M2), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I2,M2), MY0 = PM(I5,M5);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I3,M3), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (ss), MX0 = DM(I3,M3), MY0 = PM(I6,M6);
    MR = MR + MX0 * MY0 (rnd);
    MY0 = 0x7fff;
    MR = MR + MX0 * MY0 (rnd);
    DM(I3,M3) = MR1;

    DM(CosTheta) = MR1; }


DM(CosTheta)=SR0;
MR1=SR0;


{-----------------------------------------------------------------}

old_cycle:         { wait irq2 to start new switching cycle }
    AR = DM(new_cycle_flag);
    AR = pass AR;
    if eq jump old_cycle;
    jump begin;

{ *** Jump to shutdown if overcurrent protection is activated *** }

shutdown:
    AR = 1;             { turn off drivers }
```

```
    DM(disable_altera_outputs) = AR;
    set flag_out;
    idle;
    jump shutdown;


{ *** Jump to shutdown if duty-cycle limit is reached *** }

duty_limit:
    AR = 1;              { turn off drivers }
    DM(disable_altera_outputs) = AR;
    set flag_out;
    idle;
    jump duty_limit;

{ ************************************************************* }
{ ****************** The end of the program ******************* }
{ ************************************************************* }

{ *************** Coeficients for digital filters *************** }
{ Chebyshev, fc = 3000 Hz, Half sampling 15.6 kHz, order 2, rip .5 }

.var/pm/ram/circ/abs=0x02f8 b1000[3];
.init        b1000:
             0x0b2e00, 0x165c00, 0x0b2e00;
                         { b3=0.0873, b2=0.1764, b1=0.0873}

.var/pm/ram/circ/abs=0x02fc a1000[2];
.init        a1000:
             0xc77600, 0x092d00; { a3=-0.4417, a1=1+0.0716}


{ ******************** Program RAM tables ********************** }

.var/pm/ram/abs=0x02fe  entry_points[36];      { entry_points        }
.init entry_points:     ^sector_one_case_one,  { contains begining }
                  ^sector_one_case_two,  { adresses of         }
                  ^sector_one_case_three,{ switching     }
                  ^sector_one_case_four, { sequences     }
                  ^sector_one_case_five,
                  ^sector_one_case_six,

                  ^sector_two_case_one,
                  ^sector_two_case_two,
                  ^sector_two_case_three,
                  ^sector_two_case_four,
                  ^sector_two_case_five,
                  ^sector_two_case_six,

                  ^sector_three_case_one,
                  ^sector_three_case_two,
                  ^sector_three_case_three,
                  ^sector_three_case_four,
                  ^sector_three_case_five,
                  ^sector_three_case_six,

                  ^sector_four_case_one,
                  ^sector_four_case_two,
                  ^sector_four_case_three,
                  ^sector_four_case_four,
                  ^sector_four_case_five,
```

```
                    ^sector_four_case_six,

                    ^sector_five_case_one,
                    ^sector_five_case_two,
                    ^sector_five_case_three,
                    ^sector_five_case_four,
                    ^sector_five_case_five,
                    ^sector_five_case_six,

                    ^sector_six_case_one,
                    ^sector_six_case_two,
                    ^sector_six_case_three,
                    ^sector_six_case_four,
                    ^sector_six_case_five,
                    ^sector_six_case_six;


.var/pm/ram/abs=0x0328  sector_one_case_one[6]; { Type 2 }
.init               sector_one_case_one:
                                    b#001101000000000,
                                    b#101101000000000,
                                    b#110010100000000,
                                    b#010010100000000,
                                    b#010100100000000,
                                    b#001100100000000;

.var/pm/ram/abs=0x032e  sector_one_case_two[6]; { Type 1 }
.init               sector_one_case_two:
                                    b#110011000000000,
                                    b#101100100000000,
                                    b#001100100000000,
                                    b#010100100000000,
                                    b#010101000000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x0334  sector_one_case_three[6]; { Type 1 }
.init               sector_one_case_three:
                                    b#101011000000000,
                                    b#110100100000000,
                                    b#010100100000000,
                                    b#001100100000000,
                                    b#001010100000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x033a  sector_one_case_four[6]; { Type 0 }
.init               sector_one_case_four:
                                    b#010100100000000,
                                    b#110100100000000,
                                    b#101011000000000,
                                    b#001010100000000,
                                    b#001100100000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x0340  sector_one_case_five[6]; { Type 0 }
.init               sector_one_case_five:
                                    b#001100100000000,
                                    b#101100100000000,
                                    b#110011000000000,
                                    b#010101000000000,
                                    b#010100100000000,
                                    b#000000000000000;
```

```
.var/pm/ram/abs=0x0346  sector_one_case_six[6]; { Type 2 }
.init             sector_one_case_six:
                                    b#010010100000000,
                                    b#110010100000000,
                                    b#101101000000000,
                                    b#001101000000000,
                                    b#001100100000000,
                                    b#010100100000000;

.var/pm/ram/abs=0x034c  sector_two_case_one[6]; { Type 1 }
.init             sector_two_case_one:
                                    b#101101000000000,
                                    b#110010100000000,
                                    b#010010100000000,
                                    b#010100100000000,
                                    b#010101000000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x0352  sector_two_case_two[6];  { Type 2 }
.init             sector_two_case_two:
                                    b#010011000000000,
                                    b#110011000000000,
                                    b#101100100000000,
                                    b#001100100000000,
                                    b#010100100000000,
                                    b#010010100000000;

.var/pm/ram/abs=0x0358  sector_two_case_three[6]; { Type 1 }
.init             sector_two_case_three:
                                    b#101011000000000,
                                    b#110100100000000,
                                    b#010100100000000,
                                    b#010010100000000,
                                    b#001010100000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x035e  sector_two_case_four[6];  { Type 0 }
.init             sector_two_case_four:
                                    b#010100100000000,
                                    b#110100100000000,
                                    b#101011000000000,
                                    b#001010100000000,
                                    b#010010100000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x0364  sector_two_case_five[6];  { Type 2 }
.init             sector_two_case_five:
                                    b#001100100000000,
                                    b#101100100000000,
                                    b#110011000000000,
                                    b#010011000000000,
                                    b#010010100000000,
                                    b#010100100000000;

.var/pm/ram/abs=0x036a  sector_two_case_six[6]; { Type 0 }
.init             sector_two_case_six:
                                    b#010010100000000,
                                    b#110010100000000,
                                    b#101101000000000,
                                    b#010101000000000,
```

```
                                    b#010100100000000,
                                    b#0000000000000000;

.var/pm/ram/abs=0x0370 sector_three_case_one[6];  { Type 1 }
.init               sector_three_case_one:
                                    b#101101000000000,
                                    b#110010100000000,
                                    b#010010100000000,
                                    b#010011000000000,
                                    b#010101000000000,
                                    b#0000000000000000;

.var/pm/ram/abs=0x0376 sector_three_case_two[6];  { Type 0 }
.init               sector_three_case_two:
                                    b#010011000000000,
                                    b#110011000000000,
                                    b#101100100000000,
                                    b#001010100000000,
                                    b#010010100000000,
                                    b#0000000000000000;

.var/pm/ram/abs=0x037c sector_three_case_three[6]; { Type 2 }
.init               sector_three_case_three:
                                    b#001011000000000,
                                    b#101011000000000,
                                    b#110100100000000,
                                    b#010100100000000,
                                    b#010010100000000,
                                    b#010011000000000;

.var/pm/ram/abs=0x0382 sector_three_case_four[6];   { Type 2 }
.init               sector_three_case_four:
                                    b#010100100000000,
                                    b#110100100000000,
                                    b#101011000000000,
                                    b#001011000000000,
                                    b#010011000000000,
                                    b#010010100000000;

.var/pm/ram/abs=0x0388 sector_three_case_five[6]; { Type 1 }
.init               sector_three_case_five:
                                    b#101100100000000,
                                    b#110011000000000,
                                    b#010011000000000,
                                    b#010010100000000,
                                    b#001010100000000,
                                    b#0000000000000000;

.var/pm/ram/abs=0x038e sector_three_case_six[6];  { Type 0 }
.init               sector_three_case_six:
                                    b#010010100000000,
                                    b#110010100000000,
                                    b#101101000000000,
                                    b#010101000000000,
                                    b#010011000000000,
                                    b#0000000000000000;

.var/pm/ram/abs=0x0394 sector_four_case_one[6]; { Type 2 }
.init               sector_four_case_one:
                                    b#001101000000000,
                                    b#101101000000000,
```

```
                                    b#110010100000000,
                                    b#010010100000000,
                                    b#010011000000000,
                                    b#001011000000000;

.var/pm/ram/abs=0x039a sector_four_case_two[6]; { Type 0 }
.init               sector_four_case_two:
                                    b#010011000000000,
                                    b#110011000000000,
                                    b#101100100000000,
                                    b#001010100000000,
                                    b#001011000000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x03a0 sector_four_case_three[6]; { Type 0 }
.init               sector_four_case_three:
                                    b#001011000000000,
                                    b#101011000000000,
                                    b#110100100000000,
                                    b#010101000000000,
                                    b#010011000000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x03a6 sector_four_case_four[6]; { Type 1 }
.init               sector_four_case_four:
                                    b#110100100000000,
                                    b#101011000000000,
                                    b#001011000000000,
                                    b#010011000000000,
                                    b#010101000000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x03ac sector_four_case_five[6]; { Type 1 }
.init               sector_four_case_five:
                                    b#101100100000000,
                                    b#110011000000000,
                                    b#010011000000000,
                                    b#001011000000000,
                                    b#001010100000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x03b2 sector_four_case_six[6]; { Type 2 }
.init               sector_four_case_six:
                                    b#010010100000000,
                                    b#110010100000000,
                                    b#101101000000000,
                                    b#001101000000000,
                                    b#001011000000000,
                                    b#010011000000000;

.var/pm/ram/abs=0x03b8 sector_five_case_one[6]; { Type 0 }
.init               sector_five_case_one:
                                    b#001101000000000,
                                    b#101101000000000,
                                    b#110010100000000,
                                    b#001010100000000,
                                    b#001011000000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x03be sector_five_case_two[6]; { Type 2 }
.init               sector_five_case_two:
```

```
                                    b#010011000000000,
                                    b#110011000000000,
                                    b#101100100000000,
                                    b#001100100000000,
                                    b#001101000000000,
                                    b#001011000000000;

.var/pm/ram/abs=0x03c4 sector_five_case_three[6]; { Type 0 }
.init              sector_five_case_three:
                                    b#001011000000000,
                                    b#101011000000000,
                                    b#110100100000000,
                                    b#010101000000000,
                                    b#001101000000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x03ca sector_five_case_four[6]; { Type 1 }
.init              sector_five_case_four:
                                    b#110100100000000,
                                    b#101011000000000,
                                    b#001011000000000,
                                    b#001101000000000,
                                    b#010101000000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x03d0 sector_five_case_five[6];  { Type 2 }
.init              sector_five_case_five:
                                    b#001100100000000,
                                    b#101100100000000,
                                    b#110011000000000,
                                    b#010011000000000,
                                    b#001011000000000,
                                    b#001101000000000;

.var/pm/ram/abs=0x03d6 sector_five_case_six[6];  { Type 1 }
.init              sector_five_case_six:
                                    b#110010100000000,
                                    b#101101000000000,
                                    b#001101000000000,
                                    b#001011000000000,
                                    b#001010100000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x03dc  sector_six_case_one[6]; { Type 0 }
.init              sector_six_case_one:
                                    b#001101000000000,
                                    b#101101000000000,
                                    b#110010100000000,
                                    b#001010100000000,
                                    b#001100100000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x03e2  sector_six_case_two[6]; { Type 1 }
.init              sector_six_case_two:
                                    b#110011000000000,
                                    b#101100100000000,
                                    b#001100100000000,
                                    b#001101000000000,
                                    b#010101000000000,
                                    b#000000000000000;
```

```
.var/pm/ram/abs=0x03e8  sector_six_case_three[6]; { Type 2 }
.init             sector_six_case_three:
                                    b#001011000000000,
                                    b#1010111000000000,
                                    b#1101001000000000,
                                    b#0101001000000000,
                                    b#0011001000000000,
                                    b#0011010000000000;

.var/pm/ram/abs=0x03ee  sector_six_case_four[6];  { Type 2 }
.init             sector_six_case_four:
                                    b#0101001000000000,
                                    b#1101001000000000,
                                    b#1010111000000000,
                                    b#0010111000000000,
                                    b#0011010000000000,
                                    b#0011001000000000;

.var/pm/ram/abs=0x03f4  sector_six_case_five[6]; { Type 0 }
.init             sector_six_case_five:
                                    b#0011001000000000,
                                    b#1011001000000000,
                                    b#1100111000000000,
                                    b#0101010000000000,
                                    b#0011010000000000,
                                    b#0000000000000000;

.var/pm/ram/abs=0x03fa  sector_six_case_six[6]; { Type 1 }
.init             sector_six_case_six:
                                    b#1100101000000000,
                                    b#1011010000000000,
                                    b#0011010000000000,
                                    b#0011001000000000,
                                    b#0010101000000000,
                                    b#0000000000000000;

.var/pm/ram/abs=0x400   scale_table[512];
.init             scale_table: <scaling.dat>;

.var/pm/ram/abs=0x600   normal_table[512];
.init             normal_table: <normal.dat>;

.endmod;
```