APPENDIX C4: PWM-Rectifier Program (open/closed loop)

```
.module pwmrcl1;

#include <def2101.h>;

#define DD_CONST 0x6000
#define DQ_CONST 0x0600

#define ID_SS      0x6000
#define IQREF_CST 0x0000

#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_TRESHOLD  0x0200  { 0.125 of the full scale in 1.15 }
                          { if treshold is changed        }
                          { normal.dat must be changed          }

#define SWITCHING_TRESHOLD 0x051e { 0.2 of the full scale in 1.15   }

#define TRANSITION_LENGTH   0x04   { UNIT IS DOUBLE CLOCK = 125 ns}
#define CHARGING_LENGTH     0x02
#define DISCHARGING_LENGTH  0x01
#define MINIMUM_LENGTH      0x07
#define CURRENT_LIMIT       0x7a00 { 1.1/1.25 * 32767 is 0x7100 }

#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;
.var/dm/ram   sector;
.var/dm/ram   old_sector;
.var/dm/ram   invert_flag;
.init       invert_flag: 0;

{ 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; 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;
.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;

.var/dm/ram   case_table[36];
.init       case_table:
              b#110, { b#XYZ  }
              b#001,
              b#101, { X  = 1  d1 and d2 are swaped     }
              b#000, { X  = 0  d1 and d2 are not swaped }
              b#100, { YZ = 00 output waveform type 0   }
              b#010, { YZ = 01 output waveform type 1   }
              b#101, { YZ = 10 output waveform type 2   }
              b#010,
              b#001,
              b#100,
              b#110,
              b#000,
              b#001,
              b#000,
              b#010,
              b#110,
```

```
                b#101,
                b#100,
                b#010,
                b#100,
                b#000,
                b#101,
                b#001,
                b#110,
                b#000,
                b#110,
                b#100,
                b#001,
                b#010,
                b#101,
                b#100,
                b#101,
                b#110,
                b#010,
                b#000,
                b#001;

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

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

interrupt_two:
    DM(new_cycle_flag) = AY1;      { irq2 }
    rti;
    rti;
    rti;

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

timer_interrupt:
    AR = 1;
    DM(disable_altera_outputs) = AR;
    idle;

    jump timer_interrupt; { timer is used as a watch dog }

start:

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

{******************** 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;

    I0 = ^IaIn;
    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;
    SR = lshift AR by 8 (lo);
{
    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;

{
SR = lshift MR1 by -8 (hi);
AR = SR1 xor AY1;
DM(dac1) = AR;
}

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

    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);

{
    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;

{
SR = lshift MR1 by -8 (hi);
AR = SR1 xor AY1;
DM(dac3) = AR;
}
{-------------------------------------------------------------------------}

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

voltage_wait_loop:
    {toggle flag_out;}

    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;
```

```
}

MR1 = SR0;

    DM(CosTheta) = MR1;

{
SR = lshift MR1 by -8 (hi);
AR = SR1 xor AY1;
DM(dac2) = AR;
}
{-------------------------------------------------------------------}

    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;
}

MR1 = SR0;
DM(Vb) = MR1;

{
SR = lshift MR1 by -8 (hi);
AR = SR1 xor AY1;
DM(dac3) = AR;
}
{-------------------------------------------------------------------}
{ ******************* 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_TRESHOLD;
    AR = MR1 - AY0, AX0 = MR1;         { AR = offset in normal_table }
    if lt jump voltage_wait_loop;

    SR = lshift AR by -6 (lo);
    AY0 = ^normal_table;
    AR = SR0 + AY0;           { AR = correction factor address }
    I7 = AR;
    MY0 = PM(I7,M7);          { MY0 = correction / 8 }

    MR = MX0 * MY0 (ss);
    SR = ashift MR1 by 2 (hi);
    SR = SR or lshift MR0 by 2 (lo);
    AY0 = SR1;
    AR = SR1 + AY0;
    DM(CosTheta) = AR;

{
SR = lshift AR by -8 (hi);
AR = SR1 xor AY1;
DM(dac2) = 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;

{
SR = lshift AR by -8 (hi);
AR = SR1 xor AY1;
DM(dac3) = AR;
}

    AY0 = SWITCHING_TRESHOLD;
    AR = AX0 - AY0;
    if lt jump voltage_wait_loop;

{------------------- 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;
}


MR1 = SR0;
    DM(CosTheta) = MR1;

{
SR = lshift MR1 by -8 (hi);
AR = SR1 xor AY1;
DM(dac2) = AR;
}

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

    CNTR = 5;
    do loop4 until ce;

loop4:      nop;

    AR = 0;
    DM(new_cycle_flag) = AR;
    DM(disable_altera_outputs)=AR;

sync:
    AR = DM(new_cycle_flag);
    AR = pass AR;
    if eq jump sync;
{
    ena 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, adjust sign bit }
    SR = lshift AR by 8 (lo);



MR1 = SR0;
    DM(Vb) = MR1;

{-----------------------------------------------------------------}
```

```
{ ******************* 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 }

{-------------------- Wait loop -----------------}
CNTR=8;
do loop3 until ce;
loop3:
nop;

{-------------------- Read and start conversion -----------------}
    AR = DM(adcIb);     { Read Ch 2 = Ia, Start Ch 3 = Ib }
    AR = AR xor AY1;
    SR = lshift AR by 8 (lo);
                               { Digital filter }
    DM(Ialpha) = SR0;
    MR1 = SR0;

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

trt:
    if lt jump start;
    SR = lshift AR by -6 (lo);
```

```
    AY0 = ^normal_table;
    AR = SR0 + AY0;           { AR = correction factor address }
    I7 = AR;
    MY0 = PM(I7,M7);          { MY0 = correction / 8 }

    MR = MX0 * MY0 (ss);
    SR = ashift MR1 by 2 (hi);
    SR = SR or lshift MR0 by 2 (lo);
    AY0 = SR1;
    AR = SR1 + AY0;
    DM(CosTheta) = AR;

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


{SR = lshift MR1 by -8 (hi);
AR = SR1 xor AY1;
DM(dac2) = AR;}
{-------------------- Read and start conversion ------------------}

    AR = DM(adcIdref);  { Read Ch 3 = Ib, Start Ch 4 = Idref }
    AR = AR xor AY1;
    SR = lshift AR by 8 (lo);
                               { Digital filter }
    DM(Ib) = SR0;
    MR1 = SR0;

    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;


{-------------------------------------------------------------------}
```

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

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

{AY0 = ID_SS;
AR  = MR1 - AY0;
SR = lshift AR by -6 (hi);
AR = SR1 xor AY1;
DM(dac1) = AR;}


{ ********** 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 -4 (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 }
    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(Iqref);
    SR = ashift SI by -3 (hi);
    MX1= SR1;                   { MX1 = 4.12 Iqref }
```

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

    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);
    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 }
    if le jump no_scaling;

scaling:
jump shutdown;
    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 1.14 }

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

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

    jump DdDq_to_DalphaDbeta;

no_scaling:
    SI = DM(dd);
    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 }

{AR = DM(dd);
SR = lshift AR by -8 (hi);
AR = SR1 xor AY1;
DM(dac0) = AR;}

{-------------------- 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 }
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);

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

    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 #) ******** }

    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 }
        AR = 2;
        jump continue;

      sector_5:              { dalpha < 0, dbeta < 0 }
        MY0 = 4;       { SecNumber = 4 }
        AR = 5;
        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;
        AR = 3;
        jump continue;

      sector_6:              { dalpha < 0, dbeta < 0 }
        MY0 = 5;       { SecNumber = 5 }
        AR = 4;
        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 }
            AR = 6;
            jump continue;

          sector_4:                { dalpha < 0, dbeta > 0 }
            MY0 = 3;          { SecNumber = 3 }
            AR = 1;
            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 }
            AR = 4;
            jump continue;

          sector_3a:               { dalpha < 0, dbeta > 0 }
            MY0 = 2;          { SecNumber = 2 }
            AR = 3;



continue:
                  { decomment when combinations are fixed }

    AF = pass AR; { Comment this two lines to use the regulator }
    AF = AF - 1;

              { Decoment the next 13 line to use the regulator }
{
    AF = pass 0;
    AX0 = 0x02;
    AR = DM(Ic);
    AR = pass AR;
    if ge AF = AF + 1;

    AR = DM(Ib);
    AR = pass AR;
    if ge AF = AX0 + AF;

    AX0 = 4;
    AR = DM(Ialpha);
    AR = pass AR;
    if ge AF = AX0 + AF;

    AF = AF - 1;
}
sequence:
```

```
    MX0 = 6;
    MR = MX0 * MY0 (ss);       { MY0 = SecNumber }
    SR = lshift MR0 by -1(lo); { Frac adjust is not needed }
    AR = SR0 + AF;             { entry=6*SecNumber + offset}
    DM(offset) = AR;

    AY0 = ^case_table;
    AR = AR + AY0;
    I7 = AR;
    AX1 = DM(I7,M7);    { AX1 stores value from case_table }
    AY0 = 0x04;
    AR = AX1 and AY0;
    if eq jump check_waveform_type;

swap:
    AR = DM(d1);
    AX0 = DM(d2);
    DM(d1) = AX0;
    DM(d2) = AR;

check_waveform_type:
    AY0 = 0x03;
    AR = AX1 and AY0;    { AX1 = entry from the case table }
    DM(type) = AR;       { AR = waveform_type }

AF = pass AR;
SR = lshift AR by 5 (hi);
AR = SR1 xor AY1;
DM(dac1) = AR;
AR = pass AF;

    AY0 = 0x01;
    AR = AR - AY0;
    if eq jump waveform_one;
    AR = AR - AY0;
    if eq jump waveform_two;

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

waveform_zero:

    jump shutdown;
    AY0 = MINIMUM_LENGTH;

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




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

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

{ ******** Calculating the entry point to the sector's  ********* }
{ ******** switching table and the output waveform type ********* }

label1:


    AY0 = ^entry_points;
    MR0 = DM(offset);
    AR = MR0 + AY0;
    I7 = AR;           { I7 = entry address from entry_points   }
    AR = PM(I7,M7);
    I7 = AR;           { I7 = entry point to switching sequence }

{ *************** Coping data into the altera buffer  ************* }
do_not_invert_it:
          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;
          AR = PM(I7,M7);
          DM(data) = AR;

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

old_cycle1:
{-------------------- 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;
}


MR1 = SR0;
    DM(CosTheta) = MR1;

{
SR = lshift MR1 by -8 (hi);
AR = SR1 xor AY1;
DM(dac2) = AR;
}
{-----------------------------------------------------------------}

old_cycle:
    AR = DM(new_cycle_flag);
    AR = pass AR;
    if eq jump old_cycle;
    toggle flag_out;
    jump begin;

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

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


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

{ *************** Coeficients for digital filters *************** }
{ ***** Buterrworth, fc = 1000 Hz, sampling 16 kHz, order 2 ****** }

.var/pm/ram/circ/abs=0x02f8 b1000[3];
.init        b1000:
             0x011300, 0x022600, 0x011300;
                         { b3=0.0084, b2=0.0169, b1=0.0084}

.var/pm/ram/circ/abs=0x02fc a1000[2];
.init        a1000:
             0x9f0b00, 0x5ca400; { a3=-0.7575, a1=1+0.7238}


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

.var/pm/ram/abs=0x02fe  entry_points[36];
.init entry_points:     ^sector_one_case_one,
                  ^sector_one_case_two,
```

```
                    ^sector_one_case_three,
                    ^sector_one_case_four,
                    ^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#001010100000000,
                                              b#001010100000000,
                                              b#001010100000000,
                                              b#001010100000000,
                                    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#010101000000000,
                                              b#010101000000000,
                                              b#010101000000000,
                                              b#010101000000000,
                                    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#000000000000000;

.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#000000000000000;

.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#000000000000000;

.var/pm/ram/abs=0x037c sector_three_case_three[6]; { Type 2 * }
.init              sector_three_case_three:
                                    b#001010100000000,
                                              b#001010100000000,
                                              b#001010100000000,
                                              b#001010100000000,
                                    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#000000000000000;

.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#000000000000000;

.var/pm/ram/abs=0x0394 sector_four_case_one[6]; { Type 2 * }
.init               sector_four_case_one:
                                    b#010101000000000,
                                              b#010101000000000,
                                              b#010101000000000,
                                              b#010101000000000,
                                    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#001010100000000,
                                              b#001010100000000,
                                              b#001010100000000,
                                              b#001010100000000,
                                    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#101011000000000,
                                    b#110100100000000,
                                    b#010100100000000,
                                    b#001100100000000,
                                    b#001101000000000;

.var/pm/ram/abs=0x03ee  sector_six_case_four[6];  { Type 2 * }
.init             sector_six_case_four:
                                    b#010101000000000,
                                              b#010101000000000,
                                              b#010101000000000,
                                              b#010101000000000,
                                    b#001101000000000,
                                    b#001100100000000;

.var/pm/ram/abs=0x03f4  sector_six_case_five[6]; { Type 0 }
.init             sector_six_case_five:
                                    b#001100100000000,
                                    b#101100100000000,
                                    b#110011000000000,
                                    b#010101000000000,
                                    b#001101000000000,
                                    b#000000000000000;

.var/pm/ram/abs=0x03fa  sector_six_case_six[6]; { Type 1 }
.init             sector_six_case_six:
                                    b#110010100000000,
                                    b#101101000000000,
                                    b#001101000000000,
                                    b#001100100000000,
                                    b#001010100000000,
                                    b#000000000000000;
.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;
```