```c
//##############################################################################
//
//    Name: Closed loop control code for a DC-DC converter (boost) doing MPPT
//
//    Author:          Silje Odland Simonsen
//    Last change:     3rd of July, 2009
//
//------------------------------------------------------------------------------
// Purpose of program:
//
// A DC-DC boost converter is used for Maximum Power Point Tracking (MPPT) in
// a PV converter system. The program is collecting values from the input
// and output of the converter, and adjusting the reference voltage value to
// find the MPP of the PV source.
//
// The program contains code for:
// 1) P/PI-controller
// 2) MPPT algorithm (Perturb and Observe)
//
//#############################################################################

#include "DSP280x_Device.h"     // DSP280x Headerfile Include File
#include "DSP280x_Examples.h"   // DSP280x Examples Include File

// ADC start parameters
#define ADC_MODCLK 0x4 //HSPCLK = SYSCLKOUT/2*ADC_MODCLK2 = 100/(2*4) = 12.5MHz
#define ADC_CKPS   0x1 //ADC module clock = HSPCLK/2*ADC_CKPS = 6.25MHz
#define ADC_SHCLK  0xf // S/H width in ADC module periods  = 16 ADC clocks
#define AVG        256 // Average sample limit

#define SHIFT_AVG  8               // Defining the speed of the controller
#define X           180                  // Size of log arrays
#define DELTAV      480                  // Duty cycle step of MPPT

// Duty cycle boundaries
#define D_MAX           4750        // Maximum duty cycle, 5000*0.95 = 4750
#define D_MIN           0           // Minimum duty cycle


// Prototype statements for functions found within this file.
interrupt void adc_isr(void);
interrupt void cpu_timer0_isr(void);

// GLOBAL VARIABLES:

// Counters
Uint16 LoopCount;
Uint16 Counter;

// Readings of sampled values
Uint16 ADC_result_B4;   // Input voltage
Uint16 ADC_result_B5;   // Input current
Uint16 ADC_result_B6;   // Output voltage
Uint16 ADC_result_B7;   // Output current

// Sums up AVG values to find average
Uint32 ADC_sum_B4;
Uint32 ADC_sum_B5;
Uint32 ADC_sum_B6;
Uint32 ADC_sum_B7;
```

```
// Average of ADC_sum_Bx after AVG counts
Uint16 ADC_avr_B4;
Uint16 ADC_avr_B5;
Uint16 ADC_avr_B6;
Uint16 ADC_avr_B7;

// Offset values
Uint16 Vin_off;
Uint16 Vout_off;
Uint16 Iin_off;
Uint16 Iout_off;

// Real values
Uint16 Vin;
Uint16 Vout;
int16 Iin;
int16 Iout;

// Slope [mV/bit] or [mA/bit]
Uint16 m1;
Uint16 m2;
Uint16 m3;
Uint16 m4;

// Controller variables
Uint16 Vref;            // Reference value of the input
int16 error_k;          // Difference: error = Vref - Vin
int16 error_k_1;        // e(k-1)
Uint16 u_k;             // u(k)
Uint16 u_k_1;           // u(k-1)
int16 g0;               // parameter for e(k)
int16 g1;               // parameter for e(k-1)
int16 x0;               // on/off value for u(k-1) in the integral part
int16 PWM;              // Temporary control value
int16 P_part;           // proportional part of P-/PI-controller
int16 I_part;           // integral part of the PI-controller
int16 I_part1;          // e(k-1)*g1

// MPPT variables
int32 dP;               // change in power
Uint32 P_k;             // power at sample k
Uint32 P_k_1;           // power at sample k-1
int32 dV;               // change in voltage
Uint16 V_k;             // voltage at sample k
Uint16 V_k_1;           // voltage at sample k-1
Uint16 I_k;             // current at sample k
Uint16 deltaV;          // duty cycle step
int16 direction;        // direction of perturbation (MPPT)


// Storage arrays
Uint16 Vinlog[X];
Uint16 Vreflog[X];
int16 Ilog[X];
Uint32 Plog[X];
Uint16 Dlog[X];
Uint16 datalog_count;

int16 MPPTstart;        // initialization of the MPPT
```

```
// Counting variables for MPPT
int16 eee;
int16 fff;
int16 ggg;
int16 hhh;


main()
{
// -----------------------------
//  ALLOCATE VARIABLES
// -----------------------------
   Counter = 0;

   ADC_sum_B4 = 0;
   ADC_sum_B5 = 0;
   ADC_sum_B6 = 0;
   ADC_sum_B7 = 0;

// Calibration (manual)
   Vin_off = 250;
   Vout_off = 115;
   Iin_off = 2180;
   Iout_off = 2080;

   m1 = 12;       // m1 = ((45000-0)/(4040-25)) = 11.21 [mV]
   m2 = 8;        // m2 = ((9000-0)/(3665-2500))= 7.73  [mA]
   m3 = 19;       // m3 = ((48000-0)/(2640-55)) = 18.57 [mV]
   m4 = 8;        // m4 = ((9000-0)/(3265-2157)) = 8.12 [mA]


   Vref = 36000;  // Vref = 45*0.8 = 36 V

   g0 = -2;
   g1 =  -2;
   x0 = 1;

   I_part = 0;
   I_part1 = 0;
   P_part = 0;

   u_k = D_MIN;   // will be assigned to last value in interrupt
   error_k = 0;

   V_k_1 = 0;
   P_k_1 = 0;
   deltaV = DELTAV;
   direction = 1;

   MPPTstart = 0;

   eee = 0;
   fff = 0;
   ggg = 0;
   hhh = 0;
```

```
// zero setting the log arrays
   for (datalog_count=0;datalog_count<X;datalog_count++)
   {
         Vinlog[datalog_count] = 0;
         Vreflog[datalog_count] = 0;
         Plog[datalog_count] = 0;
         Ilog[datalog_count] = 0;
         Dlog[datalog_count] = 0;
   }
   datalog_count = 0;


// ------------------------------
// START OF SYSTEM INITIALIZATION
// ------------------------------
// STEP 1 "Initialize System Control"
// PLL, WatchDog, enable Peripheral Clocks
// This example function is found in the DSP280x_SysCtrl.c file.
   InitSysCtrl();

// Specific clock setting:
   EALLOW;
   SysCtrlRegs.HISPCP.all = ADC_MODCLK;  // HSPCLK = SYSCLKOUT/ADC_MODCLK
   EDIS;

// STEP 2: "Initialize GPIO"
// This example function is found in the DSP280x_Gpio.c file and
// illustrates how to set the GPIO to it's default state.
   InitEPwm6Gpio();

// STEP 3: "Clear all interrupts and initialize PIE vector table":
// Disable CPU interrupts
   DINT;

// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the DSP280x_PieCtrl.c file.
   InitPieCtrl();

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

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
// This will populate the entire table, even if the interrupt
// is not used in this example.  This is useful for debug purposes.
// The shell ISR routines are found in DSP280x_DefaultIsr.c.
// This function is found in DSP280x_PieVect.c.
   InitPieVectTable();

// Interrupts that are used in this example are re-mapped to
// ISR functions found within this file.
   EALLOW;  // This is needed to write to EALLOW protected register
   PieVectTable.ADCINT = &adc_isr;
   PieVectTable.TINT0 = &cpu_timer0_isr;
   EDIS;  // This is needed to disable write to EALLOW protected registers

// STEP 4: "Initialize all the Device Peripherals"
// These functions are found in DSP280x_InitPeripherals.c
```

```
   InitAdc();           // init the ADC
   InitCpuTimers();     // initialize the Cpu Timers

// Configure CPU-Timer 0 to interrupt every second:
// 100MHz CPU Freq, 1 second Period (in uSeconds)
   ConfigCpuTimer(&CpuTimer0, 100, 1000000);
   StartCpuTimer0();

// Enable ADCINT in PIE
   PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
   PieCtrlRegs.PIEIER1.bit.INTx7 = 1;

   IER |= M_INT1;              // Enable CPU Interrupt 1
   EINT;                       // Enable Global interrupt INTM
   ERTM;                       // Enable Global realtime interrupt DBGM

   LoopCount = 0;


// ------------------------------
//  ADC Setup
// ------------------------------

  AdcRegs.ADCTRL1.bit.ACQ_PS = ADC_SHCLK;
  AdcRegs.ADCTRL3.bit.ADCCLKPS = ADC_CKPS;
  AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0C;// select channel B4 for conversion
  AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x0D;   // select channel B5 for conversion
  AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x0E;   // select channel B6 for conversion
  AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 0x0F;   // select channel B7 for conversion
  AdcRegs.ADCMAXCONV.all = 0x0003;             // 4 conversions in the sequence

  AdcRegs.ADCTRL2.bit.EPWM_SOCA_SEQ1 = 1; //Enable SOCA from ePWM to start SEQ1
  AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;   // Enable SEQ1 interrupt (every EOS)


// ------------------------------
//  ePWM6 Setup
// ------------------------------
      //set PWM-period
      EPwm6Regs.TBPRD = 5000; // Period (one count = 10ns) --> fPWM = 20 kHz

      // set compare values A and B
      EPwm6Regs.CMPB = 2500;             // initial duty cycle = 50%
      EPwm6Regs.CMPA.half.CMPA = 25 ;    // Compare A = 25 TBCLK counts, used for
                                                starting ADC-conversion

      EPwm6Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;     //sawtooth, start counting up
      EPwm6Regs.TBCTR = 0;                           // clear TB counter
      EPwm6Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Phase loading disabled
      EPwm6Regs.TBCTL.bit.PRDLD = TB_SHADOW;
      EPwm6Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_DISABLE;
      EPwm6Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // TBCLK = SYSCLK
      EPwm6Regs.TBCTL.bit.CLKDIV = TB_DIV1;
      EPwm6Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
      EPwm6Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
      EPwm6Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;  // load on CTR = Zero
      EPwm6Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;  // load on CTR = Zero
      EPwm6Regs.AQCTLA.bit.ZRO = AQ_SET;             // Action-Qualifier
      EPwm6Regs.AQCTLA.bit.CAU = AQ_CLEAR;
      EPwm6Regs.AQCTLB.bit.ZRO = AQ_SET;
      EPwm6Regs.AQCTLB.bit.CBU = AQ_CLEAR;
```

```
    EPwm6Regs.ETSEL.bit.SOCAEN = 1;    // Enable SOC on ADC-sequencer A
    EPwm6Regs.ETSEL.bit.SOCASEL = 4;   // Select SOC from from CPMA on upcount
    EPwm6Regs.ETPS.bit.SOCAPRD = 1;

// Infinite loop
   for(;;)
   {
          LoopCount++;

      // MPPT loop
      if (CpuTimer0.InterruptCount == 1) // once every second
      {
      V_k = Vin;
      I_k = Iin;

      P_k = (Uint32)V_k*(Uint32)I_k;

      if(datalog_count < X)              // log values
      {
      Vinlog[datalog_count] = Vin;
      Vreflog[datalog_count] = Vref;
      Plog[datalog_count] = P_k;
      Ilog[datalog_count] = Iin;
      Dlog[datalog_count] = u_k;
      datalog_count++;
      }

      // MPPTstart - first running
      if (MPPTstart == 0)
      {
      P_k_1 = P_k;
      V_k_1 = V_k;

      MPPTstart = 1;
      }

      dP = (P_k) - (int32)(P_k_1);
      dV = V_k - (int32)V_k_1;
```

```
// Start evaluation
if (dP > 0)
            {
                  if (dV > 0)
                  {
                        eee++;
                        direction = 1;
                  }
                  if (dV < 0)
                  {
                        fff++;
                        direction = -1;
                  }
            }
            if (dP < 0)
            {
                  if (dV > 0)
                  {
                        ggg++;
                        direction = -1;
                  }
                  if (dV < 0)
                  {
                        hhh++;
                        direction = 1;
                  }
            }

      // Perturbation
      Vref = Vref + direction*deltaV;

      // Boundary test
      if (Vref < 200)
            { Vref = 200;}
      if (Vref > 45000)
            {Vref = 45000;}


      // Value updates
      V_k_1 = V_k;
      P_k_1 = P_k;


      CpuTimer0.InterruptCount = 0; // Reset counter


      } // end MPPT
   } // end LoopCount loop

} // end main
```

```c
// ------------------------------
//  INTERRUPTS
// ------------------------------

interrupt void cpu_timer0_isr(void)
{
   CpuTimer0.InterruptCount++;

   // Acknowledge this interrupt to receive more interrupts from group 1
   PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}


interrupt void  adc_isr(void)
{

Counter++;

            // Storing sampled values
            ADC_result_B4 = ((AdcRegs.ADCRESULT0>>4));
            ADC_result_B5 = ((AdcRegs.ADCRESULT1>>4));
            ADC_result_B6 = ((AdcRegs.ADCRESULT2>>4));
            ADC_result_B7 = ((AdcRegs.ADCRESULT3>>4));

            // Adding to the sum
            ADC_sum_B4 = ADC_sum_B4 + ADC_result_B4;
            ADC_sum_B5 = ADC_sum_B5 + ADC_result_B5;
            ADC_sum_B6 = ADC_sum_B6 + ADC_result_B6;
            ADC_sum_B7 = ADC_sum_B7 + ADC_result_B7;

if (Counter == AVG)
{
     // Find average value
       ADC_avr_B4 = (ADC_sum_B4 >> SHIFT_AVG);
       ADC_avr_B5 = (ADC_sum_B5 >> SHIFT_AVG);
       ADC_avr_B6 = (ADC_sum_B6 >> SHIFT_AVG);
       ADC_avr_B7 = (ADC_sum_B7 >> SHIFT_AVG);

       // Set sum equal to 0
       ADC_sum_B4 = 0;
       ADC_sum_B5 = 0;
       ADC_sum_B6 = 0;
       ADC_sum_B7 = 0;

     // Calculate real values of voltages and currents
     Vin = (ADC_avr_B4 - Vin_off) * m1;
     Iin = (ADC_avr_B5 - Iin_off) * m2;
     Vout = (ADC_avr_B6 - Vout_off) * m3;
     Iout = (ADC_avr_B7 - Iout_off) * m4;
```

```
// ------------------------------
//  P-/PI-CONTROLLER
// ------------------------------

u_k_1 = u_k;
error_k_1 = error_k;

// P-part
error_k = Vref - Vin;
P_part = (error_k>>5)*g0;

// I-part

I_part1 = (error_k_1>>5)*g1;
I_part = x0*u_k_1 + I_part1;

// Calculation of u(k)
PWM = (I_part) + (P_part);    // CMPB without saturation

// Boundary check
      if (PWM < 0)
      {
            PWM = D_MIN;
      }
      if (PWM > D_MAX)
      {
            PWM = D_MAX;
      }

u_k = PWM;

// update ePWM6 duty cycle
 EPwm6Regs.CMPB = u_k;

// Reset counter
Counter = 0;
}

// Reinitialize for next ADC sequence
  AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;          // Reset SEQ1
  AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;        // Clear INT SEQ1 bit
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;    // Acknowledge interrupt to PIE


  return;
}

//=============================================================================
// End of program
//=============================================================================
```