# ADC\_SOC : Analog to Digital Conversion program

//###########################################################################

// FILE: adc\_soc\_c28.c

// TITLE: ADC2 Start-Of-Conversion (SOC) Example Program.

//

//! \addtogroup control\_example\_list

//! <h1> ADC1 Start of Conversion (adc\_soc)</h1>

//!

//! This ADC example uses ePWM1 to generate a periodic

//! ADC SOC - ADCINT2. Two channels are converted,

//! ADC2B0 and ADC2B2.

//!

//! \b Watch \b Variables \n

//! - Voltage1[10] - Last 10 ADCRESULT0 values

//! - Voltage2[10] - Last 10 ADCRESULT1 values

//! - ConversionCount - Current result number 0-9

//! - LoopCount - Idle loop counter

//###########################################################################

**#include** "DSP28x\_Project.h" // Device Headerfile and Examples Include File

**#include** <string.h>

**#define** DUTY\_CYCLE\_A 500

**#define** DUTY\_CYCLE\_B 500

**#define** PERIOD 1000

// Prototype statements for functions found within this file.

**\_\_interrupt** **void** **adc1\_isr**(**void**);

**void** **Adc\_Config**(**void**);

// Global variables used in this example:

Uint16 LoopCount;

Uint16 ConversionCount;

Uint16 Voltage1[10];

Uint16 Voltage2[10];

**main**()

{

// Step 1. Initialize System Control for Control and Analog Subsystems

// Enable Peripheral Clocks

// This example function is found in the F28M35x\_SysCtrl.c file.

**InitSysCtrl**();

**#ifdef** \_FLASH

// Copy time critical code and Flash setup code to RAM

// This includes the following functions: InitFlash();

// The RamfuncsLoadStart, RamfuncsLoadSize, and RamfuncsRunStart

// symbols are created by the linker. Refer to the device .cmd file.

**memcpy**(&RamfuncsRunStart, &RamfuncsLoadStart, (size\_t)&RamfuncsLoadSize);

// Call Flash Initialization to setup flash waitstates

// This function must reside in RAM

**InitFlash**();

**#endif**

// Step 2. Initialize GPIO:

// This example function is found in the F28M35x\_Gpio.c file and

// illustrates how to set the GPIO to it's default state.

// InitGpio(); // Skipped for this example

EALLOW;

GpioG1CtrlRegs.GPADIR.bit.GPIO0 = 1; //Set as output

GpioG1CtrlRegs.GPADIR.bit.GPIO8 = 1; //Set as output

GpioG1CtrlRegs.GPAMUX1.bit.GPIO0 = 1; //Set mux to EPWM1A

GpioG1CtrlRegs.GPAMUX1.bit.GPIO8 = 3; //Set mux to ADCSOCAn

EDIS;

// 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 F28M35x\_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 F28M35x\_DefaultIsr.c.

// This function is found in F28M35x\_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.ADCINT1 = &adc1\_isr;

EDIS; // This is needed to disable write to EALLOW protected registers

// Step 4. Initialize all the Device Peripherals:

// This function is found in F28M35x\_InitPeripherals.c

// InitPeripherals(); // Not required for this example

**InitAdc1**(); // For this example, init the ADC

// Step 5. User specific code, enable interrupts:

// Enable ADCINT2 in PIE

PieCtrlRegs.PIEIER1.bit.INTx1 = 1; // Enable INT 1.1 in the PIE

IER |= M\_INT2; // Enable CPU Interrupt 1

EINT; // Enable Global interrupt INTM

ERTM; // Enable Global realtime interrupt DBGM

LoopCount = 0;

ConversionCount = 0;

// Configure ADC

EALLOW;

Adc1Regs.ADCCTL2.bit.ADCNONOVERLAP = 1; // Enable non-overlap mode i.e.

// conversion and future

// sampling events don't overlap

Adc1Regs.ADCCTL1.bit.INTPULSEPOS = 1; // ADCINT1 trips after

// AdcResults latch

Adc1Regs.INTSEL1N2.bit.INT1E = 1; // Enabled ADCINT1

Adc1Regs.INTSEL1N2.bit.INT1CONT = 0; // Disable ADCINT1 Continuous

// mode

Adc1Regs.INTSEL1N2.bit.INT1SEL = 0; // setup EOC0 to trigger ADCINT1

// to fire

Adc1Regs.ADCSOC0CTL.bit.CHSEL = 0; // set SOC0 channel select to

// ADC1A0

Adc1Regs.ADCSOC1CTL.bit.CHSEL = 2; // set SOC1 channel select to

// ADC1A2

AnalogSysctrlRegs.TRIG1SEL.all = 5; // Assigning EPWM1SOCA to

// ADC TRIGGER 1 of the ADC module

Adc1Regs.ADCSOC0CTL.bit.TRIGSEL = 5; // Set SOC0 start trigger to

// ADC Trigger 1(EPWM1 SOCA) of the

// adc

Adc1Regs.ADCSOC1CTL.bit.TRIGSEL = 5; // set SOC1 start trigger to

// ADC Trigger 1(EPWM1 SOCA) of the

// adc

Adc1Regs.ADCSOC0CTL.bit.ACQPS = 6; // set SOC0 S/H Window to 7 ADC

// Clock Cycles, (6 ACQPS plus

// 1)

Adc1Regs.ADCSOC1CTL.bit.ACQPS = 6; // set SOC1 S/H Window to 7 ADC

// Clock Cycles, (6 ACQPS plus

// 1)

EDIS;

//// Assumes ePWM1 clock is already enabled in InitSysCtrl();

//Set event triggers (SOCA) for ADC SOC1

EPwm1Regs.ETSEL.bit.SOCAEN = 1; // Enable SOC on A group

EPwm1Regs.ETSEL.bit.SOCASEL = ET\_CTRU\_CMPA; // Select SOC from CMPA on

// upcount

EPwm1Regs.ETPS.bit.SOCAPRD = 3; // Generate pulse on every 3rd

// event

// Time-base registers

EPwm1Regs.TBPRD = PERIOD; // Set timer period, PWM

// frequency = 1 / period

EPwm1Regs.TBPHS.all = 0; // Time-Base Phase Register

EPwm1Regs.TBCTR = 0; // Time-Base Counter Register

EPwm1Regs.TBCTL.bit.PRDLD = TB\_IMMEDIATE; // Set Immediate load

EPwm1Regs.TBCTL.bit.CTRMODE = TB\_COUNT\_UP; // Count-up mode: used for

// asymmetric PWM

EPwm1Regs.TBCTL.bit.PHSEN = TB\_DISABLE; // Disable phase loading

EPwm1Regs.TBCTL.bit.SYNCOSEL = TB\_SYNC\_DISABLE;

EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB\_DIV1;

EPwm1Regs.TBCTL.bit.CLKDIV = TB\_DIV1;

// Setup shadow register load on ZERO

EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC\_SHADOW;

EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC\_SHADOW;

EPwm1Regs.CMPCTL.bit.LOADAMODE = CC\_CTR\_ZERO; // load on CTR=Zero

EPwm1Regs.CMPCTL.bit.LOADBMODE = CC\_CTR\_ZERO; // load on CTR=Zero

// Set Compare values

EPwm1Regs.CMPA.half.CMPA = DUTY\_CYCLE\_A; // Set duty 50% initially

EPwm1Regs.CMPB = DUTY\_CYCLE\_B; // Set duty 50% initially

// Set actions

EPwm1Regs.AQCTLA.bit.ZRO = AQ\_SET; // Set PWM2A on Zero

EPwm1Regs.AQCTLA.bit.CAU = AQ\_CLEAR; // Clear PWM2A on event A, up

// count

EPwm1Regs.AQCTLB.bit.ZRO = AQ\_CLEAR; // Set PWM2B on Zero

EPwm1Regs.AQCTLB.bit.CBU = AQ\_SET; // Clear PWM2B on event B, up

// count

// Wait for ADC interrupt

**for**(;;)

{

LoopCount++;

}

}

**\_\_interrupt** **void** **adc1\_isr**(**void**)

{

Voltage1[ConversionCount] = Adc1Result.ADCRESULT0;

Voltage2[ConversionCount] = Adc1Result.ADCRESULT1;

// If 20 conversions have been logged, start over

**if**(ConversionCount == 9)

{

ConversionCount = 0;

}

**else** ConversionCount++;

Adc1Regs.ADCINTFLGCLR.bit.ADCINT1 = 1; //Clear ADCINT1 flag reinitialize

// for next SOC

PieCtrlRegs.PIEACK.all = PIEACK\_GROUP1; // Acknowledge interrupt to PIE

**return**;

}

(Texas Instruments, n.d.)

REFERENCES

Texas Instruments. (n.d.). [online] Available at: http://www.ti.com/tools-software/ccs.html [Accessed 5 Apr. 2019].