Task 1: Motor Current Sensing using ePWM-Triggered ADC

Hotzenblitz EV Re-Engineering Project

Mohamed Ismail

December 19, 2025

1 Objective

The objective of Task 1 is to implement deterministic motor current sensing on the TI C2000 F280039C microcontroller. Two analog current sensors are sampled every 1 ms using the ADC subsystem, with conversions triggered by the ePWM1 module. This approach ensures precise timing and minimal sampling jitter.

2 Hardware and Software Setup

2.1 Hardware

- MCU: TI TMS320F280039C (C2000 Series)
- Development Board: LAUNCHXL-F280039C
- Current Sensors connected to:
  - ADCINA2 (Sensor 1)
  - ADCINA14 (Sensor 2)

2.2 Software

- Code Composer Studio (CCS)
- C2000Ware DriverLib

3 System Working Principle

- ePWM1 is configured in up-down counting mode with a period of 1 ms.
- ePWM1 generates a Start-of-Conversion (SOCA) trigger at the time-base period event.
- ADC SOC0 and SOC1 are triggered simultaneously by ePWM1.

- An ADC interrupt is generated after conversion completion.
- The ISR reads raw ADC values, converts them into current values, and computes total current.

This hardware-triggered approach avoids CPU-timer jitter and guarantees synchronized sampling.

4 CPU Timer vs ePWM Timer Comparison

In embedded systems, periodic tasks such as ADC sampling can be triggered using either a CPU timer or a peripheral-based timer such as ePWM. In this project, ePWM is selected instead of a CPU timer due to its deterministic timing behavior and tight hardware coupling with the ADC.

4.1 CPU Timer (e.g., Timer0)

A CPU timer is a general-purpose timer that generates interrupts at fixed intervals. When a CPU timer expires, the processor must stop its current execution and service the timer interrupt.

Characteristics:

- Interrupt-driven and handled entirely by the CPU
- Susceptible to interrupt latency and jitter
- Timing depends on ISR execution and system load
- ADC triggering requires software execution inside ISR

Limitations:

- Sampling instant varies due to ISR latency
- Not synchronized with PWM signals
- Less suitable for motor-control applications

4.2 ePWM Timer (Used in Task 1)

The ePWM module contains a dedicated hardware time-base counter that can directly trigger ADC conversions without CPU involvement. The ADC Start-of-Conversion (SOC) signal is generated purely in hardware.

Characteristics:

- Hardware-triggered and deterministic timing
- No CPU intervention required for ADC triggering
- Zero jitter between PWM time-base and ADC sampling
- Ideal for synchronized motor-control measurements

Advantages:

- Precise sampling at an exact point in the PWM cycle
- Stable sampling frequency (1 kHz in this project)
- Allows CPU to execute other tasks efficiently

4.3 Why ePWM Timer is Preferred for Motor Control

Motor-control systems require current sampling at well-defined instants relative to PWM switching to avoid noise and measurement errors. Since ePWM can generate ADC triggers at specific counter events, it ensures phase-aligned and repeatable sampling.

4.4 Summary Comparison Table

Feature	**CPU Timer**	**ePWM Timer**
Trigger Type	Software interrupt	Hardware event
Timing Jitter	Present	Negligible
CPU Overhead	High	Low
ADC Synchronization	No	Yes
Motor Control Suitability	Low	High

5 Sampling and Timing

The sampling frequency is defined as:

$$f_s = \frac{1}{T_s} = \frac{1}{1\text{ ms}} = 1\,\text{kHz}$$

The ADC interrupt is executed once per sampling period, ensuring consistent current measurement.

6 Why Up-Down Counter Mode is Used

In Task 1, the ePWM module is configured in **up-down counting mode** instead of simple up-count mode. This choice is motivated by timing accuracy, symmetry, and motor-control best practices.

6.1 Up-Count vs Up-Down Count

- **Up-count mode**: The counter increments from 0 to TBPRD and then resets to 0.
- **Up-down mode**: The counter increments from 0 to TBPRD and then decrements back to 0, forming a symmetric triangular waveform.

6.2 Advantages of Up-Down Mode

- **Symmetric timing**: The PWM period is symmetric around the center of the cycle, reducing timing distortion.
- **Center-aligned ADC sampling**: ADC triggering at TBCTR = TBPRD occurs at the exact center of the PWM period, minimizing switching noise during current measurement.
- **Reduced jitter**: The sampling instant is deterministic and hardware-controlled, independent of CPU load.
- **Motor-control suitability**: Most motor-control algorithms (BLDC, PMSM, FOC) prefer center-aligned PWM and sampling for accurate current feedback.

6.3 Effect on Sampling Period

In up-down mode, one full PWM period consists of:

$$T = 2 \times TBPRD$$

For Task 1:

$$TBPRD = 60000 \Rightarrow T = 1\,\text{ms}$$

This configuration guarantees precise 1 kHz ADC sampling.

6.4 Summary

Up-down counter mode enables symmetric PWM operation and center-aligned ADC sampling, which improves measurement accuracy and reduces noise. For this reason, it is the preferred counting mode in motor-control applications and is used in Task 1.

7 Task 1 Firmware Implementation

Listing 1: Task 1: ePWM-triggered ADC current sensing code

```
#include "driverlib.h"
#include "driverlib/adc.h"
#include "device.h"

//*****************************************
// Sensor Pin / SOC Configuration
//*****************************************
#define SENSOR1_ADC_MODULE      ADCA_BASE
#define SENSOR1_ADC_CHANNEL     ADC_CH_ADCIN2
#define SENSOR1_RESULT_BASE     ADCARESULT_BASE
#define SENSOR1_SOC_NUMBER      ADC_SOC_NUMBER0

#define SENSOR2_ADC_MODULE      ADCA_BASE
#define SENSOR2_ADC_CHANNEL     ADC_CH_ADCIN14
#define SENSOR2_RESULT_BASE     ADCARESULT_BASE
```

```
16 #define SENSOR2_SOC_NUMBER      ADC_SOC_NUMBER1
17
18 //*****************************************
19 // Global Variables
20 //*****************************************
21 uint16_t sensor1_raw = 0;
22 uint16_t sensor2_raw = 0;
23
24 float current_phase_a = 0.0f;
25 float current_phase_b = 0.0f;
26 float total_current = 0.0f;
27
28 uint32_t sample_count = 0;
29
30 //*****************************************
31 // Function Prototypes
32 //*****************************************
33 void InitEPWM1(void);
34 void InitADC(void);
35 void InitADCSOC(void);
36 __interrupt void ADCA1_ISR(void);
37
38 //*****************************************
39 // Main Function
40 //*****************************************
41 void main(void)
42 {
43     Device_init();
44     Device_initGPIO();
45     Interrupt_initModule();
46     Interrupt_initVectorTable();
47
48     InitADC();
49     InitEPWM1();
50     InitADCSOC();
51
52     Interrupt_register(INT_ADCA1, &ADCA1_ISR);
53     Interrupt_enable(INT_ADCA1);
54
55     EINT;
56     ERTM;
57
58     while(1)
59     {
60         NOP;
61     }
62 }
63
64 //*****************************************
65 // InitEPWM1 - Configure ePWM1 for triggering ADC
66 //*****************************************
```

```
67  void InitEPWM1(void)
68  {
69      SysCtl_enablePeripheral(SYSCTL_PERIPH_CLK_EPWM1);
70      SysCtl_disablePeripheral(SYSCTL_PERIPH_CLK_TBCLKSYNC);
71
72      EPWM_setTimeBasePeriod(EPWM1_BASE, 60000);
73      EPWM_setTimeBaseCounter(EPWM1_BASE, 0);
74
75      EPWM_setClockPrescaler(EPWM1_BASE,
76                             EPWM_CLOCK_DIVIDER_1,
77                             EPWM_HSCLOCK_DIVIDER_1);
78
79      EPWM_setTimeBaseCounterMode(EPWM1_BASE,
            EPWM_COUNTER_MODE_UP_DOWN);
80      EPWM_setPhaseShift(EPWM1_BASE, 0);
81      EPWM_disablePhaseShiftLoad(EPWM1_BASE);
82      EPWM_setEmulationMode(EPWM1_BASE, EPWM_EMULATION_FREE_RUN);
83
84      EPWM_setADCTriggerSource(EPWM1_BASE,
85                               EPWM_SOC_A,
86                               EPWM_SOC_TBCTR_PERIOD);
87
88      EPWM_setADCTriggerEventPrescale(EPWM1_BASE, EPWM_SOC_A, 1);
89      EPWM_enableADCTrigger(EPWM1_BASE, EPWM_SOC_A);
90
91      SysCtl_enablePeripheral(SYSCTL_PERIPH_CLK_TBCLKSYNC);
92  }
93
94  //*****************************************
95  // InitADC - Configure ADC module
96  //*****************************************
97  void InitADC(void)
98  {
99      ADC_setPrescaler(ADCA_BASE, ADC_CLK_DIV_4_0);
100     ADC_setInterruptPulseMode(ADCA_BASE, ADC_PULSE_END_OF_CONV);
101     ADC_enableConverter(ADCA_BASE);
102     DEVICE_DELAY_US(1000);
103 }
104
105 //*****************************************
106 // InitADCSOC - Configure Start Of Conversion
107 //*****************************************
108 void InitADCSOC(void)
109 {
110     ADC_setupSOC(SENSOR1_ADC_MODULE,
111                  SENSOR1_SOC_NUMBER,
112                  ADC_TRIGGER_EPWM1_SOCA,
113                  SENSOR1_ADC_CHANNEL,
114                  15);
115
116     ADC_setupSOC(SENSOR2_ADC_MODULE,
```

```
                    SENSOR2_SOC_NUMBER,
                    ADC_TRIGGER_EPWM1_SOCA,
                    SENSOR2_ADC_CHANNEL,
                    15);

    ADC_setInterruptSource(ADCA_BASE, ADC_INT_NUMBER1,
       SENSOR2_SOC_NUMBER);
    ADC_enableContinuousMode(ADCA_BASE, ADC_INT_NUMBER1);
    ADC_enableInterrupt(ADCA_BASE, ADC_INT_NUMBER1);
    ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1);
}

//*****************************************
// ADCA1_ISR - ADC Interrupt Service Routine
//*****************************************
__interrupt void ADCA1_ISR(void)
{
    sensor1_raw = ADC_readResult(SENSOR1_RESULT_BASE,
       SENSOR1_SOC_NUMBER);
    sensor2_raw = ADC_readResult(SENSOR2_RESULT_BASE,
       SENSOR2_SOC_NUMBER);

    current_phase_a = ((float)sensor1_raw / 4095.0f) * 100.0f;
    current_phase_b = ((float)sensor2_raw / 4095.0f) * 100.0f;
    total_current = current_phase_a + current_phase_b;

    sample_count++;

    ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1);
    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP1);
}
```

8 Results

- ADC sampling occurs exactly every 1 ms.
- Both current sensors are sampled synchronously.
- Current values are calculated in real time inside the ISR.
- The system is fully interrupt-driven with no CPU polling.

9 Conclusion

Task 1 successfully demonstrates precise and deterministic current sensing using ePWM-triggered ADC on the F280039C. This implementation forms a reliable foundation for advanced motor-control tasks such as PWM generation, protection logic, and field-oriented control (FOC).

Reference

Texas Instruments, *C2000™ F28003x Series LaunchPad™ Development Kit User's Guide*, SPRUJ31.