Guidelines for new hardware boards with motor control SDK

Contents

Introduction

The Motor Control Software Development Kit (MC-SDK) for C2000™ microcontrollers (MCU) is a cohesive set of software infrastructure, tools, and documentation designed to minimize C2000 MCU-based motor control system development time targeted for various industry drive, servo, appliances and automotive applications, speeds up the evaluation and development of motor control system for AC motor includes BLDC and PMSM.

1.1 Recommended Reading

- TMS320F28004x Piccolo™ Microcontrollers datasheet
- TMS320F28004x Piccolo Microcontrollers Technical Reference Manual
- TMS320F28004x Piccolo™ Microcontrollers Silicon Errata
- InstaSPIN-FOC™ and InstaSPIN-MOTION™ User's Guide
- C2000WARE: C2000Ware for C2000 Microcontrollers
- MotorControl software development kit (SDK) for C2000™ MCUs
-
- MotorControl SDK InstaSPIN Lab Guide (located in MC-SDK folder, \ti\c2000\C2000Ware_MotorControl_SDK_<version>\solutions\common\sensorless_foc\docs\labs)
- hardware_getting_started_guide (located in MC-SDK folder, \ti\c2000\C2000Ware_MotorControl_SDK_<version>\solutions\common\sensorless_foc\docs)
- gui_quick_start_guide (located in MC-SDK folder, \ti\c2000\C2000Ware_MotorControl_SDK_<version>\solutions\common\sensorless_foc\docs)
- F28004x_DriverLib_Users_Guide and F28004x_DriverLib_API_Guide (located in C2000Ware folder \ti\c2000\C2000Ware_2_01_00_00\device_support\f28004x\docs)

1.2 Supporting Kits

Kits currently available all use the same processor, the controlCARD or the launchPad paired with 3-phase inverters to run the motor.

- Low Voltage / Medium Current: boostxl_drv8320rs for LaunchPad
  - 6-54V operation, 15A continuous output current
  - PN: BOOSTXL-DRV8320RS, BoosterPack, http://www.ti.com/tool/BOOSTXL-DRV8320RS

- PN: LAUNCHXL-F280049C, InstaSPIN-FOC enabled LaunchPad: http://www.ti.com/tool/launchxl-f280049c
- DRV8320RS Integrated 600mA Buck Regulator, external Current Shunt Amplifiers.
- No motor or power supply included
- Support for low voltage Permanent Magnet AC Synchronous, Brushless DC Motor, the motors are available to order:
  - BLDC http://www.ti.com/tool/lvbldcmtr
  - PMSM http://www.ti.com/tool/lvservomtr

- HIgh Voltage: tmdshvmtrinspin kit for controlCARD
  - 220VAC or 350VDC operation, 10A continuous output current
  - PN: TMDSHVMTRINSPIN high voltage motor control evaluation kit http://www.ti.com/tool/TMDSHVMTRINSPIN
  - PN: TMDSCNCD280049C, InstaSPIN enabled controlCARD: http://www.ti.com/tool/tmdscncd280049c
  - PN: TMDSADAP180TO100, 180 to 100 Pin DIMM Adapter: http://www.ti.com/tool/tmdsadap180to100
  - No motor included
  - Support for high voltage AC Induction, Permanent Magnet AC Synchronous, Brushless DC Motor, the motors are available to order:
    - BLDC http://www.ti.com/tool/hvbldcmtr
    - PMSM http://www.ti.com/tool/hvpmsmmtr

1.3 Define the current and voltage parameters in user.h

1). USER_ADC_FULL_SCALE_VOLTAGE_V

This parameter defines the maximum voltage at the input to the AD converter. The value that will be represented by the maximum ADC input (3.3V) and conversion (0FFFh). Hardware dependent, this should be based on the voltage sensing and scaling to the ADC input.

```
//! \brief Defines the maximum voltage at the AD converter
//!        full scale voltage of AD converter
#define USER_ADC_FULL_SCALE_VOLTAGE_V         ((float32_t)(57.528))
```

2). USER_ADC_FULL_SCALE_CURRENT_A

This parameter defines the maximum current at the AD converter. This value will be represented by the maximum ADC input (3.3 V) and conversion (0FFFh). The value is hardware dependent and should be based on the current sensing and scaling to the ADC input.

```
//! \brief Defines the maximum current at the AD converter
//!        BOARD_BSXL8320RS_REVA, Gain=12
#define USER_ADC_FULL_SCALE_CURRENT_A         ((float32_t)(42.843))
```

3). USER_VOLTAGE_FILTER_POLE_Hz

This parameter defines the analog filter pole location, in Hz. The value must match the hardware voltage feedback filter that is calculated based on equation the voltage sensing circuit.

```
//! \brief Defines the analog voltage filter pole location, Hz
//!
#define USER_VOLTAGE_FILTER_POLE_Hz           ((float32_t)(338.357))
```

Refer to Section 5.2 of InstaSPIN User's Guide (SPRUHJ1H) to calculate above three parameters.

1.4 Change ADC Channels

The HAL module configures the ADC channels used according to the board.

1). Assign the address of the ADC modules used in HAL_init().

```
    // initialize the ADC handles
    obj->adcHandle[0] = ADCA_BASE;
    obj->adcHandle[1] = ADCB_BASE;
    obj->adcHandle[2] = ADCC_BASE;

    // initialize the ADC results
    obj->adcResult[0] = ADCARESULT_BASE;
    obj->adcResult[1] = ADCBRESULT_BASE;
    obj->adcResult[2] = ADCCRESULT_BASE;
```

1). The configuration of the ADC in function **`HAL_setupADCs()`** as below (changes marked red) in hal.c

```
    // configure the interrupt sources
    // configure the ample window to 15 system clock cycle wide by assigning 14
    // to the ACQPS of ADCSOCxCTL Register.
    // RB2/B1
    ADC_setInterruptSource(obj->adcHandle[1], ADC_INT_NUMBER1, ADC_SOC_NUMBER2);

    // configure the SOCs for hvkit_rev1p1
    // ISENA - PGA5->A14->RA0
    ADC_setupSOC(obj->adcHandle[0], ADC_SOC_NUMBER0, ADC_TRIGGER_EPWM6_SOCA,
                 ADC_CH_ADCIN14, HAL_ADC_SAMPLE_WINDOW);

    // ISENB - PGA3->C7->RC0
    ADC_setupSOC(obj->adcHandle[2], ADC_SOC_NUMBER0, ADC_TRIGGER_EPWM6_SOCA,
                 ADC_CH_ADCIN7, HAL_ADC_SAMPLE_WINDOW);

    // ISENC - PGA1->B7->RB0
    ADC_setupSOC(obj->adcHandle[1], ADC_SOC_NUMBER0, ADC_TRIGGER_EPWM6_SOCA,
                 ADC_CH_ADCIN7, HAL_ADC_SAMPLE_WINDOW);

    // VSENA - A5->RA1
    ADC_setupSOC(obj->adcHandle[0], ADC_SOC_NUMBER1, ADC_TRIGGER_EPWM6_SOCA,
                 ADC_CH_ADCIN5, HAL_ADC_SAMPLE_WINDOW);

    // VSENB - B0->RB1
    ADC_setupSOC(obj->adcHandle[1], ADC_SOC_NUMBER1, ADC_TRIGGER_EPWM6_SOCA,
                 ADC_CH_ADCIN0, HAL_ADC_SAMPLE_WINDOW);

    // VSENC - C2->RC1
```

```
    ADC_setupSOC(obj->adcHandle[2], ADC_SOC_NUMBER1, ADC_TRIGGER_EPWM6_SOCA,
                 ADC_CH_ADCIN2, HAL_ADC_SAMPLE_WINDOW);

    // VSENVM - B1->RB2. hvkit board has capacitor on Vbus feedback, so
    // the sampling doesn't need to be very long to get an accurate value
    ADC_setupSOC(obj->adcHandle[1], ADC_SOC_NUMBER2, ADC_TRIGGER_EPWM6_SOCA,
                 ADC_CH_ADCIN1, HAL_ADC_SAMPLE_WINDOW);
```

1). Changes in function **HAL_readADCDataWithOffsets()** in hal.h (marked red)

```
    float32_t current_sf = -HAL_getCurrentScaleFactor(handle);
    float32_t voltage_sf = HAL_getVoltageScaleFactor(handle);

  // convert phase A current        ->RA0/A14
    value = (float32_t)ADC_readResult(obj->adcResult[0], ADC_SOC_NUMBER0);
    pADCData->I_A.value[0] = value * current_sf;

    // convert phase B current        ->RC0/C7
    value = (float32_t)ADC_readResult(obj->adcResult[2], ADC_SOC_NUMBER0);
    pADCData->I_A.value[1] = value * current_sf;

    // convert phase C current        ->RB0/B7
    value = (float32_t)ADC_readResult(obj->adcResult[1], ADC_SOC_NUMBER0);
    pADCData->I_A.value[2] = value * current_sf;

    // convert phase A voltage        ->RA1/A5
    value = (float32_t)ADC_readResult(obj->adcResult[0], ADC_SOC_NUMBER1);
    pADCData->V_V.value[0] = value * voltage_sf;

    // convert phase B voltage        ->RB1/B0
    value = (float32_t)ADC_readResult(obj->adcResult[1], ADC_SOC_NUMBER1);
    pADCData->V_V.value[1] = value * voltage_sf;

    // convert phase C voltage        ->RC1/C2
    value = (float32_t)ADC_readResult(obj->adcResult[2], ADC_SOC_NUMBER1);
    pADCData->V_V.value[2] = value * voltage_sf;

    // convert dcBus voltage          ->RB2/B1
    value = (float32_t)ADC_readResult(obj->adcResult[1], ADC_SOC_NUMBER2);
    pADCData->dcBus_V = value * voltage_sf;
```

1.5 Change PWM Channels

The PWM outputs need to be changed to accommodate the board.

1). Configure the PWM connections in function **HAL_setupGPIOs()** in hal.c to setup the GPIOs as PWM outputs

```
    // EPWM1A->UH
    GPIO_setMasterCore(0, GPIO_CORE_CPU1);
    GPIO_setPinConfig(GPIO_0_EPWM1A);
    GPIO_setDirectionMode(0, GPIO_DIR_MODE_OUT);
    GPIO_setPadConfig(0, GPIO_PIN_TYPE_STD);

    // EPWM1B->UL
```

```
    GPIO_setMasterCore(1, GPIO_CORE_CPU1);
    GPIO_setPinConfig(GPIO_1_EPWM1B);
    GPIO_setDirectionMode(1, GPIO_DIR_MODE_OUT);
    GPIO_setPadConfig(1, GPIO_PIN_TYPE_STD);

    // EPWM2A->VH
    GPIO_setMasterCore(2, GPIO_CORE_CPU1);
    GPIO_setPinConfig(GPIO_2_EPWM2A);
    GPIO_setDirectionMode(2, GPIO_DIR_MODE_OUT);
    GPIO_setPadConfig(2, GPIO_PIN_TYPE_STD);

    // EPWM2B->VL
    GPIO_setMasterCore(3, GPIO_CORE_CPU1);
    GPIO_setPinConfig(GPIO_3_EPWM2B);
    GPIO_setDirectionMode(3, GPIO_DIR_MODE_OUT);
    GPIO_setPadConfig(3, GPIO_PIN_TYPE_STD);

    // EPWM3A->WH
    GPIO_setMasterCore(4, GPIO_CORE_CPU1);
    GPIO_setPinConfig(GPIO_4_EPWM3A);
    GPIO_setDirectionMode(4, GPIO_DIR_MODE_OUT);
    GPIO_setPadConfig(4, GPIO_PIN_TYPE_STD);

    // EPWM3B->WL
    GPIO_setMasterCore(5, GPIO_CORE_CPU1);
    GPIO_setPinConfig(GPIO_5_EPWM3B);
    GPIO_setDirectionMode(5, GPIO_DIR_MODE_OUT);
    GPIO_setPadConfig(5, GPIO_PIN_TYPE_STD);
```

2). Assign the address of the PWM modules used in function HAL_init() in hal.c

```
    // initialize PWM handles for Motor
    obj->pwmHandle[0] = EPWM1_BASE;        //!< the PWM handle
    obj->pwmHandle[1] = EPWM2_BASE;        //!< the PWM handle
    obj->pwmHandle[2] = EPWM3_BASE;        //!< the PWM handle
```

3) Configure the PWM modules in function **`HAL_setupPWMs()`** in hal.c

1.6 Change Interrupt configuration

1). Initializes the interrupt vector table in function **`HAL_initIntVectorTable()`** in hal.h

```
static inline void HAL_initIntVectorTable(HAL_Handle handle)
 {

    Interrupt_register(INT_ADCB1, &mainISR);

    return;
 } // end of HAL_initIntVectorTable() function
```

2). Enables the ADC interrupts in function **`HAL_enableADCInts()`** in hal.c

```
void HAL_enableADCInts(HAL_Handle handle)
{
    HAL_Obj *obj = (HAL_Obj *)handle;

    // enable the PIE interrupts associated with the ADC interrupts
```

```
    Interrupt_enable(INT_ADCB1);        //RB2

    // enable the ADC interrupts
    ADC_enableInterrupt(obj->adcHandle[1], ADC_INT_NUMBER1);

    // enable the cpu interrupt for ADC interrupts
    Interrupt_enableInCPU(INTERRUPT_CPU_INT1);

    return;
} // end of HAL_enableADCInts() function
```

3). Change interrupt acknowledge in **HAL_ackADCInt()** in hal.h

```
static inline void
HAL_ackADCInt(HAL_Handle handle,const ADC_IntNumber adcIntNum)
{
    HAL_Obj *obj = (HAL_Obj *)handle;

    // clear the ADC interrupt flag
    ADC_clearInterruptStatus(obj->adcHandle[1], adcIntNum);          // ADCB/RB2

    // Acknowledge interrupt from PIE group 1
    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP1);

    return;
} // end of HAL_ackADCInt() function
```

4). Change argument in **HAL_ackADCInt()** in is0x_xx.c

```
    //
    // acknowledge the ADC interrupt
    //
    HAL_ackADCInt(halHandle, ADC_INT_NUMBER1);
```

Notes 1: Need to change the PGA and CMPSS configuration in the functions `HAL_setupPGAs()` and `HAL_setupCMPSSs()` if used both modules

Notes 2: Change the function `HAL_setupGPIOs()` for other assigned GPIOs according to the hardware board.

Notes 3: Change Fault Protection PWM Configuration in HAL_setupFaults() in hal.c

Notes 4: Change the related functions for other used peripherals in hal.c.