Design Guide: TIDA-010948

6-Axis Motor Control With Position Feedback and Industrial Communication Protocols Reference Design

Description

This reference design showcases the ability of using two TI Sitara™ MCU-AM243x devices to handle a Simple Open Real-Time Ethernet Gigabit (SORTE_G) connected motor drive with six-axis control. The design includes one 800MHz R5F core on one AM243x device to perform closed-loop Field Oriented Control with a 62.5µs cycle time for six independent motors with incremental encoders. One programmable real-time unit (PRU) core of this AM243x acts as the SORTE_G controller to send requests and receive the 6-axis motor angles. The PRU core of another AM243x device acts as the SORTE_G device to send 6-axis motor angles which are decoded by other PRU cores on this AM243x to the controller side in every cycle. The AM243x can also support both multiprotocol absolute encoders and multiprotocol industrial Ethernet.

Resources

TIDA-010948, TIDEP-01032	Design Folders
AM243x Motor Control SDK, LP-AM243	Tool Folder
BLDC BoosterPack™ Plug-in Module	Tool Folder
AM2434, TQ-3P-SOM-TQMA243XL	Product Folders
DRV8316, AMC1035	Product Folders
DP83869, THVD1450	Product Folders

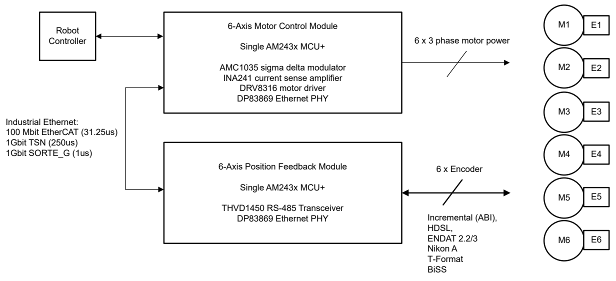
Ask the TI E2E™ support experts

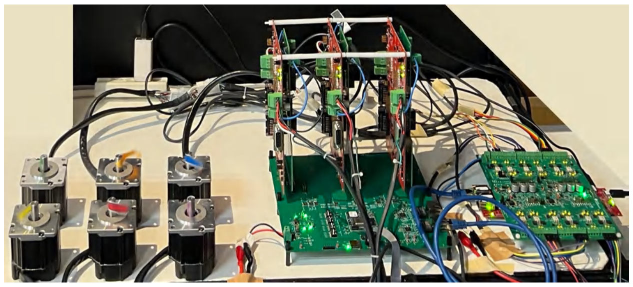
Features

- 18-channel EPWM and 18-channel ICSSG_PRU PWM to generate PWM waveforms based on the output of six FOC loops
- 12-channel Sigma-Delta filtering firmware with continuous sampling and load sharing between PRU cores for phase-current feedback from six directly-connected motors
- SORTE_G firmware implemented by ICSSG. Includes an extend option for EtherCAT®, PROFINET®, or Gigabit TSN
- Six independent closed-loop FOC for current and velocity with 62.5µs cycle time
- Six-channel incremental encoders decoding with an EQEP module and PRU cores
- Extended option for multi-encoder protocols with ICSSG, for example, EnDAT, HDSL, Biss C, Tamagawa, and Nikon A

Applications

- Robot servo drive
- Servo drive position feedback
- Robot position feedback aggregator
- Servo drive control module





1 System Description

This reference design demonstrates the capacity of the AM243x devices to handle 6-axis comprehensive real-time servo motor control and industrial communication protocol. The implementation is done on two TI AM243x SoCs, an AM243x ALV package, and an AM243x ALX package.

The AM243x ALV package with TQ-SOM is placed on the control board to generate 36 complementary PWM signals using both EPWM and ICSSG_PRU PWM peripherals and to measure the phase current from a 12-channel delta-sigma filter module. This action is achieved with ICSSG firmware and also by communication with position feedback board through SORTE_G to send and receive 6-axis motor mechanical angles requests in every 62.5µs cycle time. On the control board, the design uses one 800MHz R5F core to achieve a FOC loop for six independent motors with incremental encoders or can be extended to absolute encoders with multiple protocols. Another R5F core can be used to implement the industrial Ethernet stack as an option. One ICSSG1 is used as a delta-sigma filter module and the other ICSSG0 is used as the SORTE_G controller or EtherCAT secondary controller for an additional option.

The LP_AM243 with ALX package is used for the position feedback board controller. The ICSSG0 on the position feedback board decodes the 4-channel incremental encoders by capturing the rising and falling edge of the ABI signals through the industrial Ethernet peripheral (IEP). The other two channel encoders are decoded with the EQEP module. The ICSSG1 is used as the SORTE_G device to send 6-axis motor mechanical angles data in every 62.5µs cycle time.

1.1 Terminology

SoC	System On Chip
FOC	Field Oriented Control
MCU	Micro-programmed Control Unit
ALV, ALX	Package Drawing of AM243x devices
ABI	An incremental encoder with two quadrature-encoded output A and B, and Index I
RPM	Revolutions Per Minute
LUT	Look-Up Table
EnDAT, HDSL, BissC, Tamagawa, Nikon A	Multi-digital, bidirectional interface protocols for absolute encoders
EtherCAT	Ethernet for Control Automation Technology
Profinet	Portmanteau for Process Field Network
SORTE	Simple Open Real-Time Ethernet
ICSSG	Industrial Communication Subsystem Gigabit
PRU	Programmable Real-time Unit
RTU	Auxiliary Programmable Real-Time Unit
SDFM	Sigma-Delta Filtering Module
SDDF	Sigma-Delta Decimation filtering
SDM	Sigma-Delta Modulator
IEP	Industrial Ethernet Peripheral
CMP	Event Comparator
CAP	Event Capture
ISR	Interrupt Service Routine
EPWM	Enhanced Pulse-Width Modulation
EQEP	Enhanced Quadrature Encoder Pulse
GPIO	General-Purpose Input Output
FIFO	First In, First Out

TSR	Time Sync Routers (Instantiations of Generic interrupt router module in AM243x)
TCM	Tightly Coupled Memory
DRAM	Dynamic Random Access Memory
RGMII	Reduced Gigabit Media Independent Interface
MII_G_RT	Real-time Media Independent Interface Gigabit
MDIO	Management Data Input Output
TQ-SOM	System on module vendor for TI's Arm®-based processors with in-house manufacturing and design services

1.2 Key System Specifications

Table 1-1. Key System Specifications

SUBSYSTEM	SPECIFICATION	COMMENT
SoC EPWM	18-channel complementary PWMs with: • Configurable dead-band • Single- or double update capability • Adjustable switching frequency • Synchronized or phase-shift mode capability • Synchronized with PRU-ICSS PWM and SDFM	For current demonstration, only target on synchronized mode.
ICSSG_PRU PWM	18-channel complementary PWMs with: • Configurable dead-band • Single- or double update capability • Adjustable switching frequency • Synchronized or phase-shift mode capability • Synchronized with EPWM and SDFM	For current demonstration, only target on synchronized mode.
Current feedback – ICSS_SDFM	12-channel SDFM for phase current feedback (2 channels per axis) with: • Normal current OSR: 64 • Load sharing between RTU and PRU cores • Continuous sampling mode • Single- or double update capability	
Position feedback – PRU EQEP	Decode 4-channel motor angles from incremental ABI encoders with PRU cores and IEP_CAP	
Position feedback – SoC EQEP	Decode 2-channel motor angles from incremental ABI encoders with SoC EQEP modules	
Control Algorithm – FOC loop	Achieve 6 independent FOC loops for current and velocity within 62.5µs cycle time (16kHz PWM frequency)	For current demonstration, target on 16kHz double update
Control Algorithm – Time synchronization	Timing synchronization between control loops (position, speed, current, PWM, and Industrial Ethernet)	For current demonstration, using SORTE_G and TSR for the synchronization between the controller and the device.
Industrial communication – SORTE_G	Point-to-point communication between the controller and the device. Controller sends a request for 6 axis motor mechanical angles and receives the data every 62.5µs cycle time through Gigabit Industrial Ethernet protocol	Reduce latency using SORTE_G compares to EtherCAT, Profinet

2 System Overview

Figure 2-1 shows the setup for the TIDA-010948 overall system. The system includes the following:

- TIDA-010948 reference design, a 6-axis control board with an AM2434 ALV SoC as the main servo control MCU
- Three BP-AM2BLDCSERVO boards as the power stage for the servo control and each BP-AM2BLDCSERVO board can support dual axes
- Three adapter boards to route the signals from power stage to the control board
- Six-axis position board to receive the six channel encoder signals
- One LP-AM243x with an AM2434 ALX SoC as the position feedback MCU to decode the motor angle and velocity

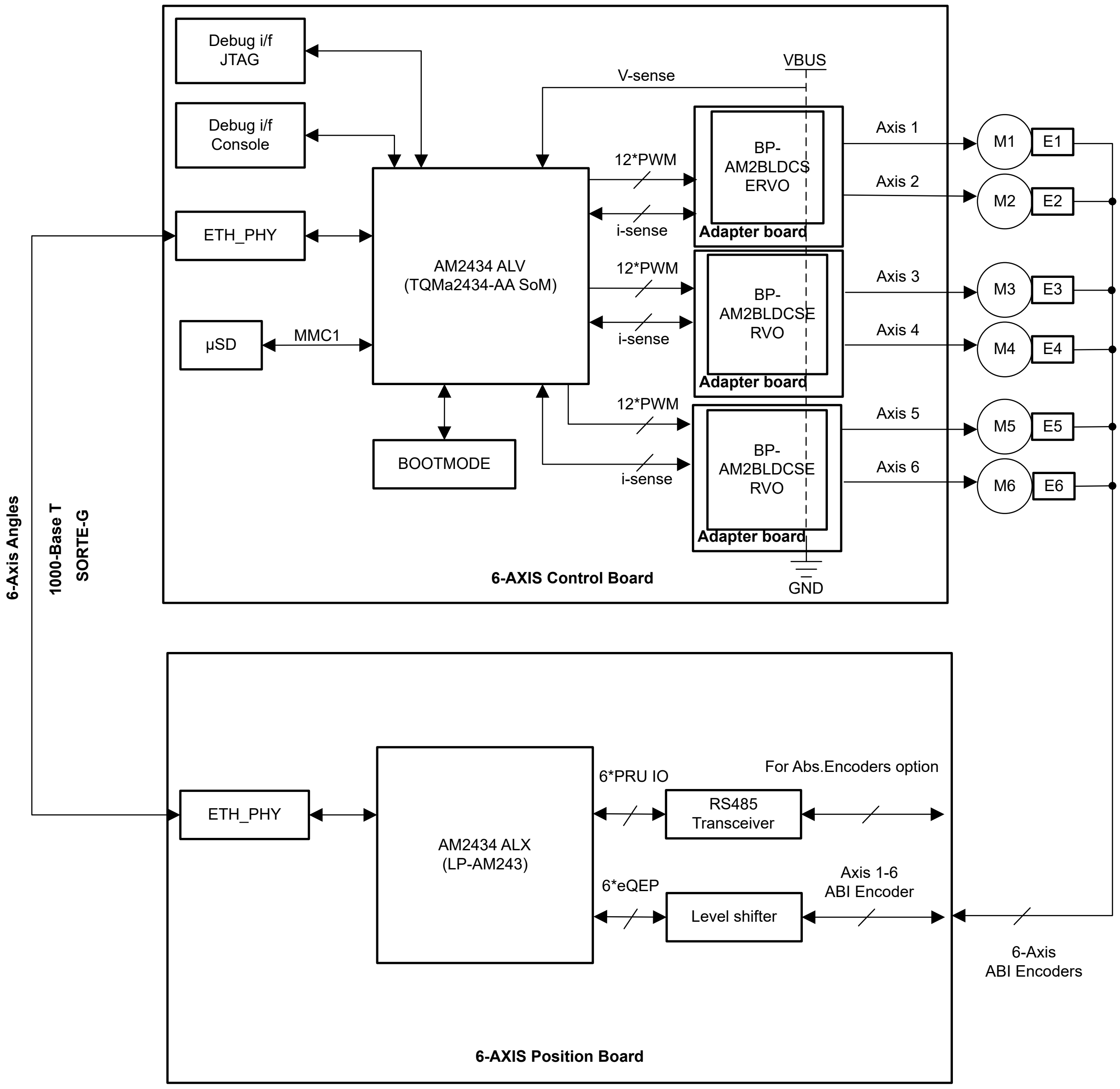


Figure 2-1. System Setup With TIDA-010948

2.1 Block Diagram

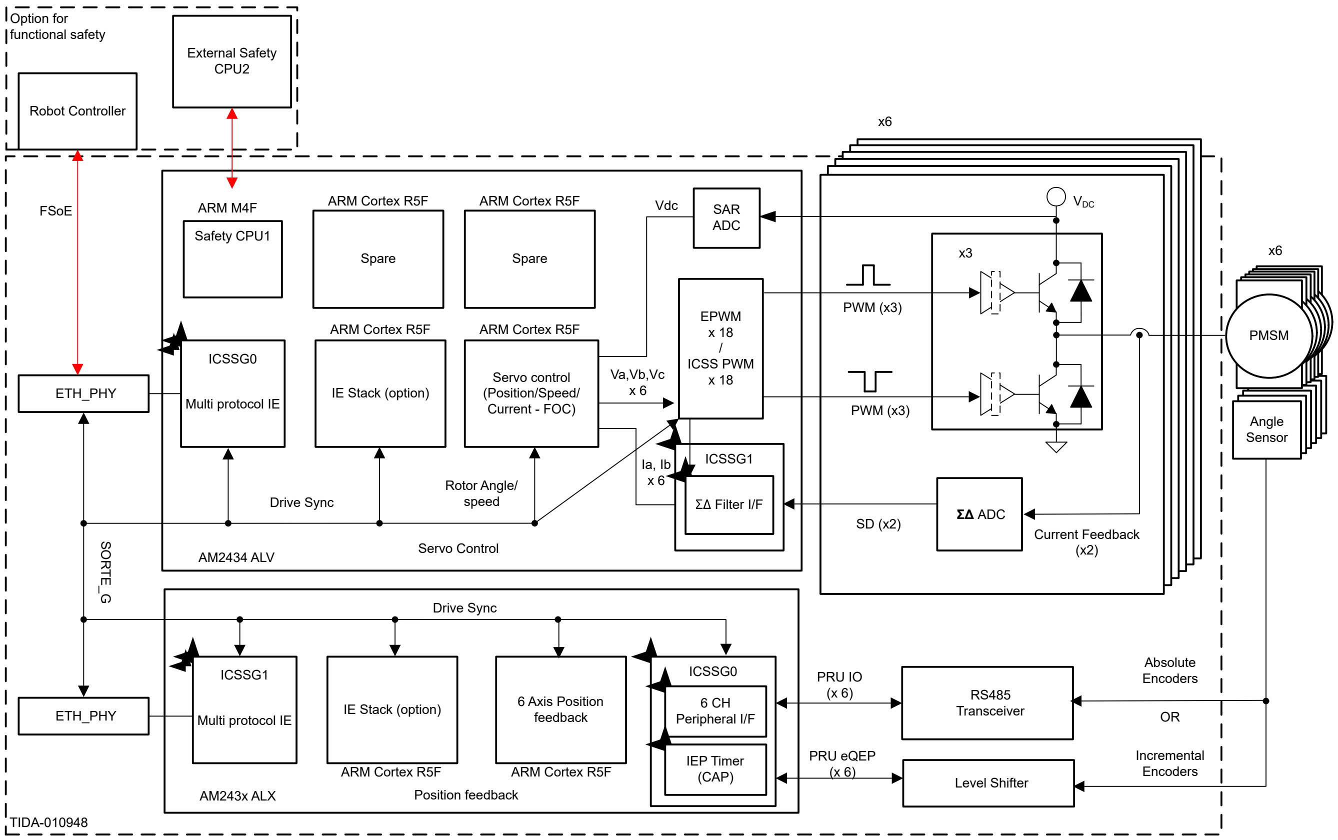


Figure 2-2. System Block Diagram With TIDA-010948

2.2 Design Considerations

The 6-axis servo control implementation was architected around a central real-time path that is made up of two subsystems:

- 6-axis control board with:
  - ICSSG0: SORTE_G controller firmware or EtherCAT secondary controller firmware as extended option.
  - R5FSS0_0: Six independent closed loops capable of current, velocity, or position. Closed loop with FOC for six directly-connected motors with encoders.
  - R5FSS1_0: EtherCAT secondary stack implementing as extended option.
  - EPWM: 18 channels of enhanced PWM peripherals to generate waveforms based on the output of 3 axis FOC loops.
  - ICSSG1: Sigma-Delta filtering firmware with continuous sampling and load sharing between RTU and PRU cores in both slice0 and slice1 for phase-current feedback from six directly connected motors.
  - ICSSG_PRU PWM (ICSSG1): 18 channels of complementary ICSSG PWM signals with dead-band assertion generated based on the output of the other 3 axis FOC loops.
- 6-axis position board with:
  - ICSSG1: SORTE_G device firmware or EtherCAT secondary controller firmware as extended option.
  - R5FSS0_0: System initialization and LUT generation.
  - R5FSS1_0: EtherCAT secondary stack implementing as extended option.
  - ICSSG0: PRU_EQEP firmware for decoding 4-channel encoder data.
  - EQEP: 2-channel encoder data decoding.

The power stage reuses the BP-AM2BLDCSERVO board, see TIDEP-01032 for details. A total of three BoosterPack™ Plug-in Module boards plus three adapter boards to receive the phase current data from all the AMC1035 devices and send PWM signaling from the control board to drive all the DRV8316 devices.

Figure 2-3 shows the time synchronization between the position board and the control board with the Ethernet time-stamping feature of SORTE_G using the IEP timer and TSR module. SORTE_G *IN* packet timing is deterministic and pre-configured to match the sample time plus calculation for position data plus Ethernet delay. At 1Gbps 64 bytes in the packet takes less than 1μs, plus an additional 1μs for Ethernet physical layer delay and line delay. This allows more compute time or a faster PWM cycle time with double update.

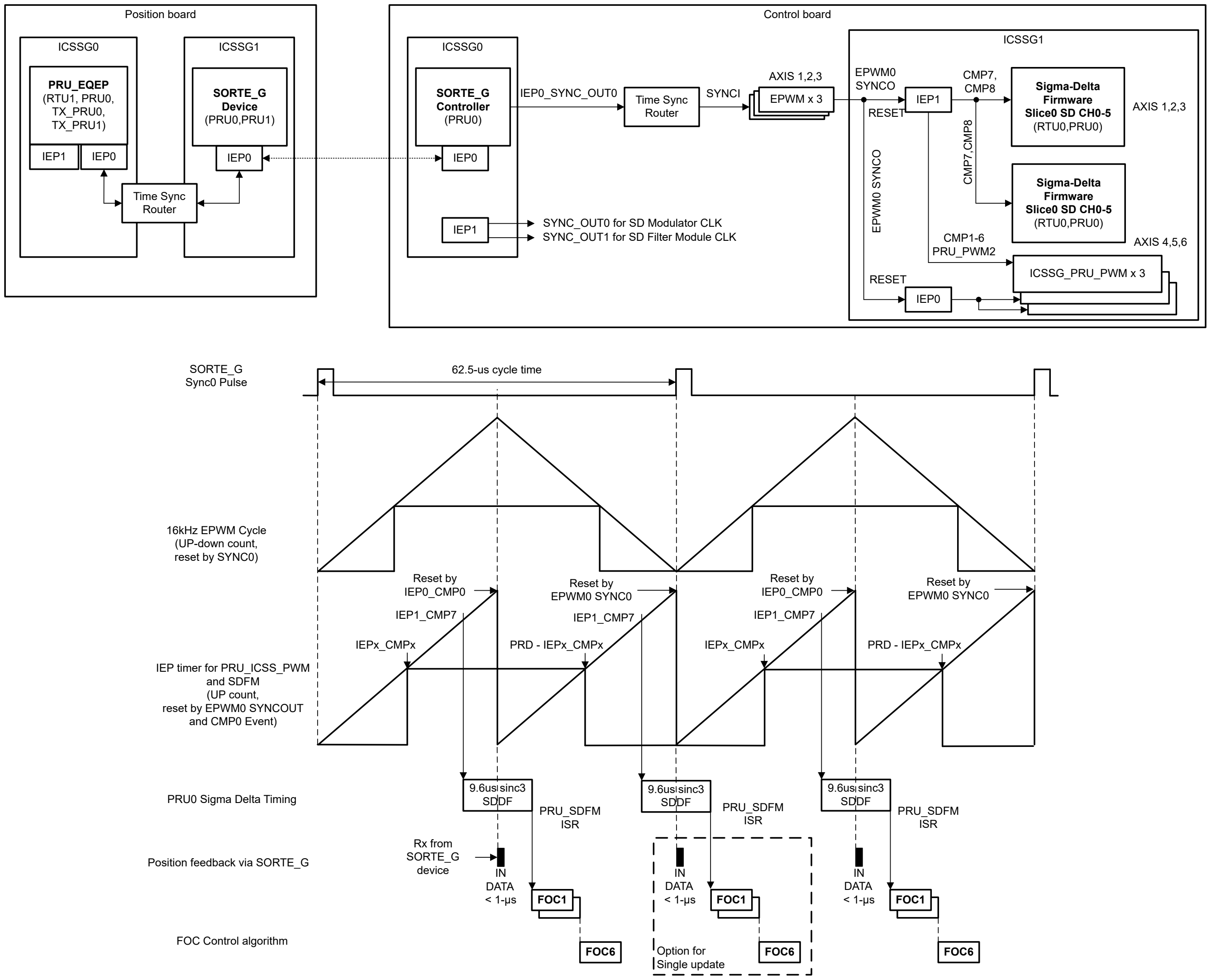


Figure 2-3. Time Synchronization and Trigger FOC Timing – 16kHz

2.3 Highlighted Products - AM243x subsystems

2.3.1 Control Board - SORTE_G Controller Interface

SORTE_G controller firmware:

- `SORTE_g_master_PRU0.h` generated with PRU project `SORTE_g_master`.
- Source code `main_PRU0.asm` in `SORTE_g_master` project contains the function:
  - Schedule the task manager for different states
  - Initialize the ICSSG0 PRU0 register
  - Configure to RGMII interface and IEP clock
  - Check the Ethernet link status
  - Generate the discovery packet, parametrization packet, and calculate and save the delay for synchronization and sends the data packet to the device with TX_L2 FIFO

- Source code `rx.asm` in the `SORTE_g_master` project run receive function with PRU0 broadside interface and register transfer instructions.

SORTE_G configuration and initialization:

- Source code `sorte_g_app_tq_control_board.c` to define the *IN* packet stored address for:
  - Axis 1 received motor angle data in TCM address *0x78000808*
  - Axis 2 received motor angle data in TCM address *0x78000810*
  - Axis 3 received motor angle data in TCM address *0x78000818*
  - Axis 4 received motor angle data in TCM address *0x78000820*
  - Axis 5 received motor angle data in TCM address *0x78000828*
  - Axis 6 received motor angle data in TCM address *0x78000830*
- Configure the cycle time as 62.5μs and the *IN* packet offset as 31.25μs. Initialize ICSSG0, MDIO interface, and IEP timer, then load the firmware and enable PRU cores with the `generic_pruss_init()` function.
- The ICSSG0_IEP0_SYNCOUT0 from SORTE_G client is utilized to synchronize the EPWM block.

2.3.2 Control Board – SDFM Interface

Figure 2-4 illustrates the clock source distribution for both the Sigma-Delta filtering module and the modulators. Table 2-1 shows the SDFM signals.

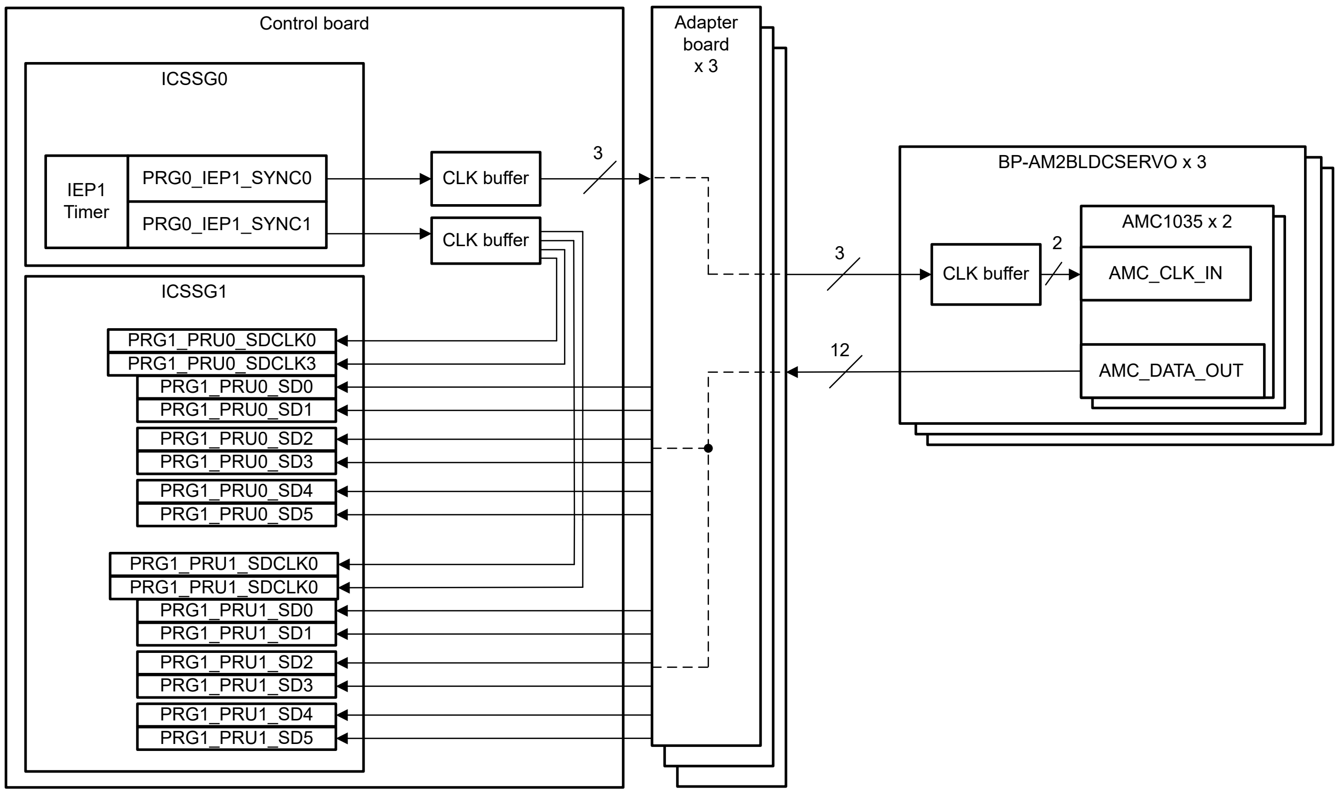


Figure 2-4. Sigma-Delta Clock and Data Distribution

Table 2-1. SDFM Signals

SUBSYSTEM	SIGNAL NAME	PIN NAME	AM243x BALL PINS	TQ_SoM PINS
Axis1-phase A	DOUT_A1	PRG1_PRU0_SD0	U8	F5
Axis1-phase B	DOUT_B1	PRG1_PRU0_SD1	V8	G3
Axis2-phase A	DOUT_A2	PRG1_PRU0_SD2	V13	H1
Axis2-phase B	DOUT_B2	PRG1_PRU0_SD3	U13	H4
Axis3-phase A	DOUT_A3	PRG1_PRU0_SD4	U15	J2
Axis3-phase B	DOUT_B3	PRG1_PRU0_SD5	AA8	J5

Table 2-1. SDFM Signals (continued)

SUBSYSTEM	SIGNAL NAME	PIN NAME	AM243x BALL PINS	TQ_SoM PINS
Slice0-SD0_CLK	Slice0_SD0_CLK	PRG1_PRU0_SD0_CLK	Y7	F4
Slice0-SD3_CLK	Slice0_SD3_CLK	PRG1_PRU0_SD3_CLK	AA7	H3
Axis4-phase A	DOUT_A4	PRG1_PRU1_SD0	V11	M4
Axis4-phase B	DOUT_B4	PRG1_PRU1_SD1	Y12	N2
Axis5-phase A	DOUT_A5	PRG1_PRU1_SD2	AA13	N5
Axis5-phase B	DOUT_B5	PRG1_PRU1_SD3	V15	P2
Axis6-phase A	DOUT_A6	PRG1_PRU1_SD4	V14	P5
Axis6-phase B	DOUT_B6	PRG1_PRU1_SD5	AA10	R3
Slice1-SD0_CLK	Slice1_SD0_CLK	PRG1_PRU1_SD0_CLK	W11	M3
Slice1-SD3_CLK	Slice1_SD3_CLK	PRG1_PRU1_SD3_CLK	U11	P1
SDM_CLK_SOURCE	SDM_CLK	PRG0_IEP1_SYNC0	R2	A2
SDFM_CLK_SHIFT	SDFM_CLK	PRG0_IEP1_SYNC1	V5	B3

The following list describes the SDFM clock:

- The clock source for the Sigma-Delta modulators located on BoosterPack boards is provided by the SYNC0 cyclic output of ICSSG0 IEP1 with 20MHz.
- The clock source for the Sigma-Delta filtering module of ICSSG1 is provided by SYNC1 cyclic output of ICSSG0 IEP1 with 20MHz. The clock selection value is set to option 2 which means the PRG1_PRUx_SD0_CLK is for SD Channel 0, 1, 2 (SD0, SD1, and SD2). PRG1_PRUx_SD3_CLK is for SD Channel 3, 4, 5 (SD3, SD4, and SD5); here x = 0 or 1 for slice0 and slice1.
- The clock phase shift between SDFM and SDM can be configured by setting the delay between IEP SYNC0 and SYNC1 to eliminate. The configuration of IEP is achieved with the `init_IEP1_SYNC()` function in `mclk_iep_sync.c`.

The SDFM Interrupts are defined in the following list:

- `hwiPrms.intNum = ICSSG_PRU_SDDF_INT_NUM0;`
- `hwiPrms.callback = pruSddfIrqHandler0;`

The following list shows the SDFM output data buffers:

- `gSddfChSamps0[0-5]` (in. `gSddfChSampsRaw0`) in TCMB of R5F_0_0 for axis 1, 2, and 3
- `gSddfChSamps[0-5]` (in. `gSddfChSampsRaw`) in TCMB of R5F_0_0 for axis 4, 5, and 6

The following files are for the SDFM firmware:

- `sdfm_rtu_bin.h` is the PRU firmware for SD0, SD1, and SD2 of slice 0
- `sdfm_pru_bin.h` is the PRU firmware for SD3, SD4, and SD5 of slice 0
- `sdfm_rtu1_bin.h` is the PRU firmware for SD0, SD1, and SD2 of slice 1
- `sdfm_pru1_bin.h` is the PRU firmware for SD3, SD4, and SD5 of slice 1
- The SDDF initialization and firmware load is done with `sddf.c`

The SDFM parameters configuration can be set in the structure `gTestSdfmPrms0` for Axis 1, 2, and 3 and `gTestSdfmPrms` for Axis 4, 5, and 6 including the IEP clock frequency, SD clock frequency, first sample trigger time, SD clock source option, and normal current over-sampling rate (OSR). For this demonstration, the default set is using:

- 250MHz IEP clock
- 20.833333MHz SD clock
- 26.45μs first sample trigger time
- Trigger events are ICSSG1_IEP1_CMP7 for RTU core and ICSSG1_IEP1_CMP8 for PRU core
- Load-sharing between RTU and PRU cores
- Option 2 for SD clock source
- Normal current sampling with OSR 64

For additional details, see AM243x Motor Control SDK: Current Sense.

2.3.3 Control Board - EPWM Interface

Table 2-2 shows the EPWM*0–8* signals for Axis 1, 2, and 3.

Table 2-2. EPWM*0–8* Signals

SUBSYSTEM	SIGNAL NAME	PERIPHERAL	AM243x BALL PINS (ALV)	TQ_SoM PINS
Axis1_PWM	PWM_A1_H	EPWM6	B14	W21
	PWM_A1_L	EPWM6	A15	V19
	PWM_B1_H	EPWM8	V1	B9
	PWM_B1_L	EPWM8	W1	D9
	PWM_C1_H	EPWM7	W20	AA6
	PWM_C1_L	EPWM7	W21	AB6
Axis2_PWM	PWM_A2_H	EPWM5	T19	V9
	PWM_A2_L	EPWM5	W19	U5
	PWM_B2_H	EPWM4	R18	V7
	PWM_B2_L	EPWM4	T21	V6
	PWM_C2_H	EPWM3	V18	W5
	PWM_C2_L	EPWM3	Y21	Y5
Axis3_PWM	PWM_A3_H	EPWM2	V19	Y7
	PWM_A3_L	EPWM2	T17	AA7
	PWM_B3_H	EPWM1	U19	W8
	PWM_B3_L	EPWM1	V20	Y8
	PWM_C3_H	EPWM0	U20	AA9
	PWM_C3_L	EPWM0	U18	AB9

EPWM configuration with the `init_pwms()` function including:

- Configure the SYNCI and SYNCO mapping to tie all the 9 EPWM groups together as daisy-chain connection. Set bit [10–8] of the `CTRLMMR_EPWM0_CTRL` register to 2h, then the EPWM0_SYNCI is triggered by `TIMESYNC_INTRTR0_OUT_38` of the TSR module output. The time sync router input 29 is routed to output 28 which means the ICSSG0_IEP0_SYNC0 from SORTE_G controller triggers the EPWM0. Also, the EPWM0_event out is enabled by *appEpwmCfg.cfgEt = TRUE* to reset the IEP timer which is used for the ICSSG_PRU_PWM and SDFM module. Set bit [10–8] of `CTRLMMR_EPWM3_CTRL`" and `CTRLMMR_EPWM6_CTRL` registers to 1h from default 0h, then the EPWM3_SYNCI and EPWM6_SYNCI is triggered by EPWM2_SYNCO and EPWM5_SYNCO, respectively, as a daisy-chain connection.
- EPWM frequency setting to 16kHz by *appEpwmCfg.epwmOutFreq = gEpwmOutFreq*.
- EPWM Counter mode setting to UP-DOWN mode by *appEpwmCfg.epwmTbCounterDir = EPWM_TB_COUNTER_DIR_UP_DOWN*.
- EPWM dead-band is configured by parameters of `appEpwmCfg.cfgDb` and `appEpwmCfg.dbCfg.x`.
- EPWM period and compare value calculated with the `App_epwmConfig()` function and output data is `gEpwmPrdVal` for all axis.
- EPWM Interrupt configured by *hwiPrms.intNum = PWM_C3_INTR* (EPWM0) and the call back function is *hwiPrms.callback = &App_epwmIntrISR*.

The EPWM compare events are updated according to the FOC calculation results with the `writeCmpA()` function. See also the *EPWM module* section of the *AM64x/AM243x Technical Reference Manual (TRM)*.

2.3.4 Control Board - ICSSG_PRU PWM Interface

Table 2-3 shows the ICSSG_PRU PWM0-2 signals for Axis 4, 5, and 6.

Table 2-3. ICSSG_PRU PWM0-2 Signals

SUBSYSTEM	SIGNAL NAME	PERIPHERAL	IEP_CMP	AM243x BALL PINS (ALV)	TQ_SoM PINS
Axis4_PWM	PWM_A4_H	ICSSG1_PRU_PWM2	IEP1_CMP1	N16	U7
	PWM_A4_L		IEP1_CMP2	N17	U8
	PWM_B4_H		IEP1_CMP3	P17	V10
	PWM_B4_L		IEP1_CMP4	Y18	V4
	PWM_C4_H		IEP1_CMP5	V21	AB8
	PWM_C4_L		IEP1_CMP6	R16	W6
Axis5_PWM	PWM_A5_H	ICSSG1_PRU_PWM1	IEP0_CMP7	V10	R4
	PWM_A5_L		IEP0_CMP8	U10	T2
	PWM_B5_H		IEP0_CMP9	AA11	T3
	PWM_B5_L		IEP0_CMP10	Y11	T5
	PWM_C5_H		IEP0_CMP11	Y10	U1
	PWM_C5_L		IEP0_CMP12	AA14	U2
Axis6_PWM	PWM_A6_H	ICSSG1_PRU_PWM0	IEP0_CMP1	U9	K2
	PWM_A6_L		IEP0_CMP2	W9	K3
	PWM_B6_H		IEP0_CMP3	AA9	K5
	PWM_B6_L		IEP0_CMP4	Y9	L1
	PWM_C6_H		IEP0_CMP5	V9	L3
	PWM_C6_L		IEP0_CMP6	U7	L4

ICSSG_PRU PWM configuration the `init_pruIcssPwm()` function of `app_pruicss_pwm.c` including:

- PWM signals output state including initial, active, and trip state defined by the API function `PRUICSS_PWM_stateInit()`.
- PWM signals enabling with the API function `PRUICSS_PWM_signalEnable()`.
- PWM initial period, duty cycle, and dead-band configured by the API function `PRUICSS_PWM_config()`.
- PWM frequency setting by API function `PRUICSS_PWM_pruIcssPwmFrequencyInit()`.

- IEP timer configured with the `PRUICSS_PWM_IEP_Config()` function to:
  – Enable IEP shadow mode.
  – Calculate the CMP0 value as the PWM period based on IEP clock and PWM frequency.
  – Enable IEP reset with both the EPWM0_SYNCO and the IEP CMP0 event. (IEP CMP0 value is with one clock cycle delay than half of the ICSSG_PRU PWM period so that the IEP CMP0 event resets the IEP only at the period, *not* the zero point of the PWM timer.)

ICSSG_PRU_PWM IEP_CMP0 interrupt `App_pruicssIep1Compare0IrqSet()` is used for setting the software flag for the updating the compare event value of the next rising edge of the PWM signal with the results of the FOC loop calculation:

- `hwiPrms.intNum` = CSLR_R5FSS0_CORE0_INTR_CMP_EVENT_INTROUTER0_OUTP_16
- `hwiPrms.callback` = &App_pruIcssPwmHalfDoneIrq
- Define the source index of the `ICSSG1_IEP0` compare 0 event number `TISCI_PRU_ICSSG1_IEP1_CMP0_SRC_INDEX` as 12U

The compare event of the next falling edge of the PWM signal is updated inside the EPWM0 ISR.

For more details, see *AM243x Motor Control SDK: PRU-ICSS PWM DEADBAND EPWM SYNC*.

2.3.5 Control Board - ICSSG_PRU IEP Timer

The following parameters apply for the IEP timer setup:

- ICSSG0:
  - IEP0 set to 250MHz clock for SORTE_G controller.
  - IEP0 SYNC_OUT0 to synchronize EPWM time-based counter.
  - IEP1 SYNC_OUT0 20MHz as the clock source for SDM.
  - IEP1 SYNC_OUT1 20MHz as the clock source for SDFM.
- ICSSG1:
  - IEP clock set to 250MHz for ICSSG_PRU PWM (same as EPWM time-based counter).
  - IEP0 and IEP1 CMP0 set to 7812 as the PWM period (250000000/16000/2).
  - IEP0 CMP1 to CMP12 as the compare events for Axis 5 and 6 PWM.
  - IEP1 CMP1 to CMP6 as the compare events for Axis 4.
  - IEP1 CMP7 to CMP8 as the compare event (first sample trigger) for SDFM.

2.3.6 Control Board – FOC Loop Control

The six independent FOC loop is called with the `AxisXFocLoopHandlerX()` function, (here X = *1 to 6* for 6 axis) inside the SDFM ISR in every PWM cycle. Figure 2-3 describes the timing for both the PWM update and current feedback. The SDFM ISR is generated twice during each PWM cycles. For a single update, the FOC loop is called by judging both the software flag `gEpwmSyncFlag` set in the EPWM0 ISR and `gUpdateNextRisingEdgeCmpValue` set in the IEP_CMP0 event ISR. As for the double update, the FOC loop can be called twice in each PWM cycle.

The `AxisXFocLoopHandlerX()` function includes:

- Receive the latest mechanical angle decoded with the position feedback board through the industrial Ethernet SORTE_G at the beginning, the raw angle data format is Q23 and needs to be converted to float format as the variable `mechThetaX` (here X = *1 to 6* for 6-axis)
- Calibrate the offset between the mechanical angle and the electric angle used for FOC calculation by:
  - First, spin the motor for a few cycles with a pure open loop to obtain the correct angle with the position board since the *I* pulse triggers the motor angle to the original *0* degree.
  - Second, force the electric angle (variable `elecTheta`) to *0* degrees and force the q-axis current (variable `parkIqOut`) to *0*. Set a constant value for d-axis current (variable `parkIdOut`) then only the torque current without back electromotive force (EMF) generates for the motor. At this moment, the motor does not spin but is forced to the *0* degree electric angle so that the mechanical offset can be known by calculating the remainder between `mechThetaX` and *90.0* degree using the `fmod()` function. Then store the mechanical offset as the variable `mechAngleOffsetX` (here X = *1 to 6* for 6-axis).
- FOC calculation with pure open loop, closed current loop, and closed speed loop options:
  - `elecTheta` used for FOC calculation need to be compensated with `mechAngleOffsetX` then increased by 4 times since the motor is with 4 pole pairs. Make sure `elecTheta` is in the range of 0 to 360 degrees as the input of the `ti_r5fmath_sincos()` function.
  - Clarke and Park are transformed by the `CLARKE_run_twoInput()` and `PARK_run()` function.
  - Inverse Park is transformed with the `IPARK_run()` function and space vector generation is composed with the `SVGEN_runCom()` function.
  - For a pure open loop, the incremental motor angle is given by a fixed value `myMechDeltaX` and the q-axis current is given by `gIq`.
  - For the closed loop, the PI controller is achieved with the `DCL_runPIParallel()` function for both current and velocity. The current and velocity target are defined in the `gIqArray` and `gSpdArray` array. The PI constants can be tuned in the `init_pids()` function.

2.3.7 Position Board – SORTE_G Device Interface

The SORTE_G device software includes the following:

- Firmware `SORTE_g_device_PRU0.h` and `SORTE_g_device_PRU1.h` are generated with the PRU project `SORTE_g_device`. Short J6 pin 59 and J6 pin 60 with a jumper on LP-AM243 to make the board as a device.
- Source code `main.asm` includes the function as:

  - Initialize the PRU register
  - Configure the task manager for the different states
  - Reset and configure the MII_G_RT module
  - Generate MDIO link event to read in current Ethernet link status
  - Execute the control loop which calls interrupt event manager and the SORTE_G states
- Source code `sorte_g_app.c` defines the IO exchange data sent from the device which is loaded from the following addresses:
  - Channel 0 EQEP sends the motor angle to the ICSSG1 DRAM0 address *0x30081504*
  - Channel 1 EQEP sends the motor angle to the ICSSG1 DRAM0 address *0x3008150C*
  - Channel 2 EQEP sends the motor angle to the ICSSG1 DRAM0 address *0x30081514*
  - Channel 3 EQEP sends the motor angle to the ICSSG1 DRAM0 address *0x3008151C*
  - Channel 4 EQEP sends the motor angle to the ICSSG1 DRAM0 address 0x30081524
  - Channel 5 EQEP sends the motor angle to the ICSSG1 DRAM0 address *0x3008152C*

2.3.8 Position Board – PRU_EQEP Interface

Figure 2-5 illustrates the architecture of PRU EQEP firmware for 4-channel position feedback from incremental encoders. The PRU_EQEP design consists of an interface for four incremental encoders with A, B, and I signals. These signals pass through the TXB0106RGYR level shifter to match the encoder signal level to the logic levels of the microcontroller. The GPIO is configured to trigger interrupts on the rising and falling edge for signals A and B, and on the rising edge for signal I. These interrupts are routed to the IEP in ICSSG. The IEP has a 64-bit counter, which captures the counter values to measure the motor RPM and angle. The PRU core has a task manager which configures several tasks that are executed in PRU. These tasks capture the counter values and compute the motor RPM and angle. Moreover, the tasks trigger the *XFR2VBUS* widget to read the status of the GPIOs that basically denotes the status of the encoder signals which are fundamental inputs for the calculations.

Table 2-4 shows the 4-channel ABI signals on the LP-AM243 side which are the outputs of the level-shifter on the position board.

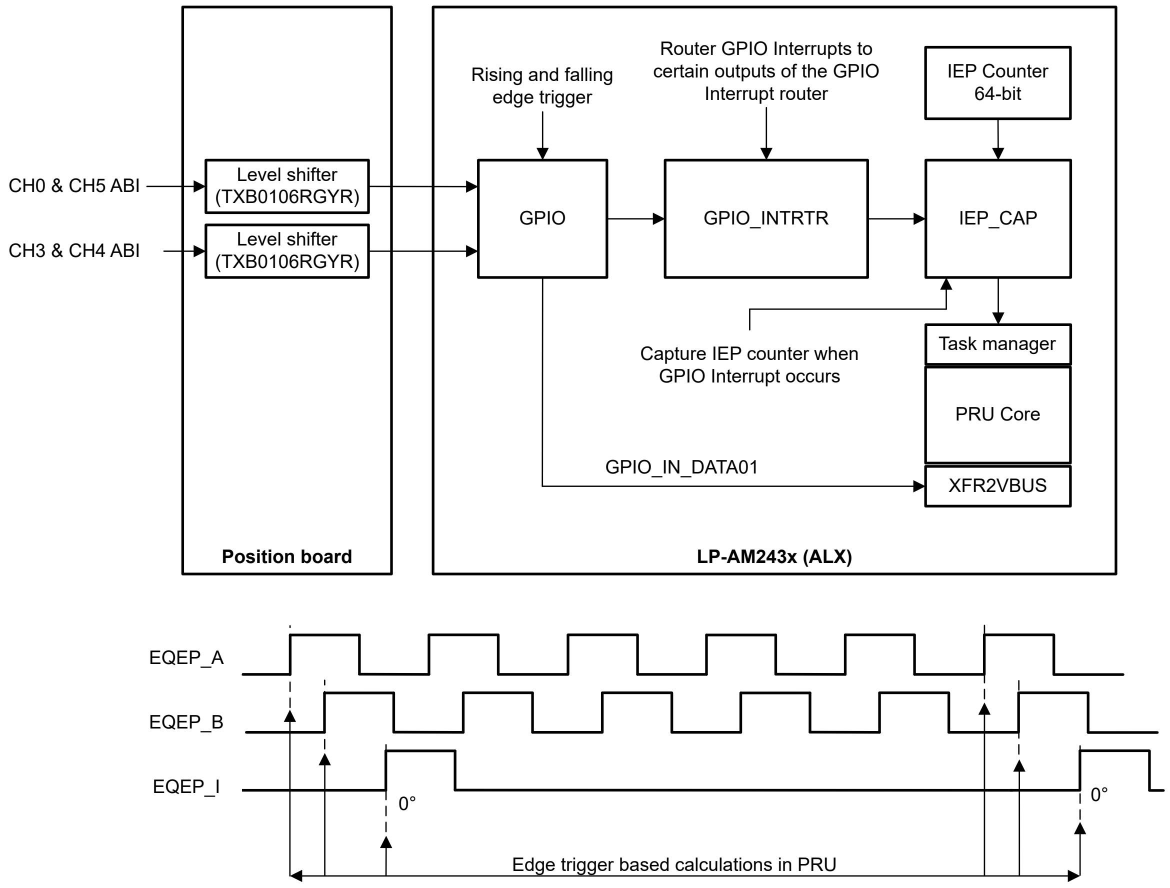

CH0 & CH5 ABI
CH3 & CH4 ABI
Level shifter (TXB0106RGYR)
Level shifter (TXB0106RGYR)
Position board
Rising and falling edge trigger
Router GPIO Interrupts to certain outputs of the GPIO Interrupt router
IEP Counter 64-bit
GPIO
GPIO_INTRTR
IEP_CAP
Capture IEP counter when GPIO Interrupt occurs
Task manager
PRU Core
GPIO_IN_DATA01
XFR2VBUS
LP-AM243x (ALX)
EQEP_A
EQEP_B
EQEP_I
0°
0°
Edge trigger based calculations in PRU

Figure 2-5. PRU_EQEP Subsystem Architecture and Interface

Table 2-4. PRU_EQEP Signals

SUBSYSTEM	SIGNAL NAME	PERIPHERAL	PRU CORE	LP-AM243 HEADER	AM243x BALL PINS (ALX)
PRU_EQEP CH0	EQEP_A_CH0	GPIO1_20	ICSSG0_RTU1	BP.11	L5
	EQEP_B_CH0	GPIO1_21		BP.67	J2
	EQEP_I_CH0	GPIO1_33		BP.71	T4
PRU_EQEP CH3	EQEP_A_CH3	GPIO1_0	ICSSG0_PRU0	BP.33	J3
	EQEP_B_CH3	GPIO1_1		BP.32	J4
	EQEP_I_CH3	GPIO1_6		BP.48	H2
PRU_EQEP CH4	EQEP_A_CH3	GPIO1_2	ICSSG0_TXPRU0	BP.31	G1
	EQEP_B_CH3	GPIO1_3		BP.19	H1
	EQEP_I_CH3	GPIO1_7		BP.44	E2
PRU_EQEP CH5	EQEP_A_CH3	GPIO1_4	ICSSG0_TXPRU1	BP.17	K2
	EQEP_B_CH3	GPIO1_5		BP.13	F2
	EQEP_I_CH3	GPIO1_8		BP.51	H5

PRU_EQEP software includes the following:

- Firmware files:
  - Firmware `EQEP_position_feedback_CH0_ICSS_G0_RTU_PRU1.h` for the Channel 0 encoder
  - Firmware `EQEP_position_feedback_CH3_ICSS_G0_PRU0.h` for the Channel 3 encoder
  - Firmware `EQEP_position_feedback_CH4_ICSS_G0_TX_PRU0.h` for the Channel 4 encoder
  - Firmware `EQEP_position_feedback_CH5_ICSS_G0_TX_PRU1.h` for the Channel 5 encoder
- GPIO configuration:
  - Enable the rising edge triggered interrupt with the `GPIO_SET_RIS_TRIG` register and the falling edge detection by *GPIO_SET_FAL_TRIG* for `EQEP_A` and `EQEP_B`. The *I* signal requires only a rising edge interrupt, as this signal represents a single pulse per revolution used for reference.
  - The GPIO Interrupt Router module serves as an essential intermediary for multiplexing GPIO interrupt signals from the inputs to outputs destinations. Once the GPIO edge-triggered interrupts are configured, these interrupts are routed to the inputs of the GPIO Interrupt Router. The router then maps these inputs to specific outputs. The outputs of the GPIO Interrupt Router are subsequently directed to the IEP module within the ICSSG.

Note

The configuration and routing of interrupts within the GPIO Interrupt Router can only be performed by the SCI client.

- ICSSG_PRU configuration:
  - Set the ICSSG0 IEP timer to 333Mhz, each counter is 1ns.
  - Configure CMP0 to the maximum value with wrap around mode, set CMP5 to 62.5µs to trigger the task manager for speed calculation.
  - `EXT_CAP_EN[5:0]` is set to enable the IEP capture of GPIO interrupt router output.
  - Task `sub_task_TS2_S0` handles the speed calculation. `sub_task_TS2_S1` handles the motor angle calculation with the `XFR2VBUS` widget. `sub_task_TS2_S2` handles the EQEP_A signal detection with the precise time stamping of encoder A transitions. `sub_task_TS2_S3` handles the EQEP_B signal detection with precise time stamping of encoder B transitions. `sub_task_TS2_S4` handles the EQEP_I signal detection to mark the reference position in the rotation cycle of the encoder which provides a reset point for the angular value to zero degrees.

2.3.9 Position Board – SoC EQEP Module Interface

Two channels of position feedback from incremental encoders utilize the SoC EQEP module. Table 2-5 shows the 2-channel ABI signals on the LP-AM243 side.

Table 2-5. SoC EQEP Module Signals

SUBSYSTEM	SIGNAL NAME	PERIPHERAL	LP-AM243 HEADER	AM243x BALL PINS (ALX)
SoC EQEP CH1	EQEP_A_CH1	EQEP1	J12.1	L2
	EQEP_B_CH1		J12.2	L3
	EQEP_I_CH1		J12.3	R5
SoC EQEP CH2	EQEP_A_CH2	EQEP2	J21.1	B14
	EQEP_B_CH2		J21.2	A15
	EQEP_I_CH2		J21.3	B13

The SoC EQEP is configured in the `generic_pruss_init()` function including:

- Configure the QEP period as 16kHz with QEP clock 125MHz.
- Initialize the position counter with zero and set the maximum counter value as 4000 according to the motor specification.
- Configure the QEP position counter source, latch the condition and reset with the index event.
- Configure and enable the interrupt by unit time out:
  - Interrupt number is 144 and interrupt callback function `EQEP1_ISR` for channel 1
  - Interrupt number is 145 and interrupt callback function `EQEP2_ISR` for channel 2

See also the *EQEP module* section of the *AM64x/AM243x Technical Reference Manual (TRM)*.

3 System Design Theory

3.1 Position Board – System Initialization

Use the following steps to initialize the position board as the SORTE_G device and decoding ABI signals for all six encoder channels in R5F_0_0 core with the `generic_pruss_init()` function. Load and run the position board code *prior* to the setting up the control board.

1. Enable System Command Interpreter (SCI Client) by pre-loading the configuration through the secondary boot loader (SBL) using image `sbl_null_sciclient.release.hs_fs.tiimage`.
2. Copy the `include.zip` and `pru_fw_common.zip` files under the workspace folder for the SORTE_G firmware usage.
3. Initialize ICSSG PRU by clearing the data RAM and setting the entry point.
4. Write the motor direction and velocity LUT into the PRU data RAM.
5. Set up GPIO pin mode as input for PRU_EQEP and as EQEP for SoC QEP, set up the GPIO interrupt mode as rising-edge detection.
6. Set up the RGMII interface and MII_G_RT module for the SORTE_G device.
7. Set up the ICSSG0 IEP timer for PRU_EQEP and the ICSSG1 IEP timer for SORTE_G.
8. Set up the SoC QEP module parameters and interrupts.
9. Load and run the SORTE_G device and PRU_EQEP firmware. Then the 6-channel decoded motor angle data is copied into the PRU data memory with pre-defined addresses and is ready for transmitting in every pre-defined PWM cycle.

3.2 Position Board – Interrupts

- Channel 1 EQEP1 interrupt number is 144 and the interrupt callback function `EQEP1_ISR` which is used to calculate the motor angle by reading the number of channel 1 QEP ticks and write the angle data into the pre-defined PRU data memory.
- Channel 2 EQEP2 interrupt number is 145 and the interrupt callback function `EQEP2_ISR` which is used to calculate the motor angle by reading the number of channel 2 QEP ticks and write the angle data into the pre-defined PRU data memory.
- Channel 0 and channel 3 to channel 5 utilize GPIO interrupt when detecting the rising and falling edge of AB signaling on relative IO pins.

3.3 Control Board – System Initialization

Use the following steps to initialize the control board as the SORTE_G controller and to receive six-channel motor angle data for controlling all 6 axis motors in the R5F_0_0 core with the `single_chip_servo_remote_core_start()` function. Load and run the control board code *after* setting up the position board.

1. The control board is using `TQ-SoM`, thus the flash on the SOM needs to be configured manually. The flash has to use the `tq_sbl_uart_uniflash.hs_fs.tiimage` image with the Python® `uart_uniflash.py` (under the `mcu_plus_sdk\tools\` folder) tool to flash the SBL. Copy `tq_sbl_uart_uniflash.hs_fs.tiimage` and `default_sbl_null_tq.cfg` to the SDK folder `mcu_plus_sdk\tools\boot\sbl_prebuilt\am243x-evm`.
2. Set up GPIO pin direction and initial values with `init_gpio_state()` function.
3. Disable PWM by `enable_pwm_buffers(FALSE)` function.
4. Configure ICSSG_PRU_PWM for 3-phase complementary per axis 4,5,6 and set initial duty cycle to 50% with `init_pruIcssPwm()` function.
5. Configure EPWM for 3-phase complementary per axis 1, 2, 3 and set initial duty cycle to 50% with `init_pwms()` function.
6. Configure ICSSG1 RTU0, RTU1, PRU0, PRU1 cores for SDFM for all 6 axis by `init_sddf()` function and load 4 SDFM firmware with load-sharing mode. Initial ICSSG0 IEP1 timer SYNC0 and SYNC1 for SD clocks by `init_IEP1_SYNC()` function. Start ICSSG1 IEP timer with `start_ICSSG1_IEPx()` function.
7. Configure ICSSG0 PRU0 cores for SORTE_G controller and load SORTE_G controller firmware by `generic_pruss_init()` function.
8. Initialize the parameters for FOC control with the `init_pids()` function.
9. Enable the EPWM output buffers for all 6 axes.

3.4 Control Board – Interrupts

- EPWM interrupt (16kHz) – interrupt number is 108 (EPWM0) and the interrupt callback function `App_epwmIntrISR` which is used to reset the IEP timer and update the next ICSSG_PRU_PWM falling edge for Axis 4, Axis 5, and Axis 6 by half the PWM period minus the rising edge. This interrupt compares values as single update scheme or by FOC calculation results as a double update scheme.
- ICSSG_PRU_PWM interrupt (16kHz) – IEP_CMP0 interrupt `App_pruicssIep1Compare0IrqSet()` which is used for setting the software flag `gUpdateNextRisingEdgeCmpValue` to update the compare event value of the next rising edge of the PWM signal with the results of FOC loop calculation.
- SDFM interrupt (32kHz) – interrupt number is 251 (PRU_ICSSG1_PR1_HOST_INTR_PEND_3) and interrupt callback function `pruSddfIrqHandler0` which is used to trigger the FOC loop for 6 axes. From sample 8192 to 16384, compute the SDFM channel offsets `gSddfChOffset[x]` and compensate for the current feedback in the FOC loop.

4 Hardware, Software, Testing Requirements, and Test Results

4.1 Hardware Requirements

The following equipment is required to test this reference design:

- One Control board - TIDA-010948_CB as Figure 4-1 and Figure 4-2 illustrate
- Three adapter boards – TIDA-010948_DB as shown in Figure 4-3 and Figure 4-4
- Three BP-AM2BLDCSERVO — AM2x Brushless-DC (BLDC) Servo Motor BoosterPack™
- One position board – TIDA-010948_PB as shown in Figure 4-5 and Figure 4-6
- One LP-AM243 Evaluation board – AM243x general purpose LaunchPad™ development kit for the Arm®-based MCU
- Six LVSERVOMTR motors - Low Voltage Servo Motor - Low voltage servo (encoder) motor and wiring harness

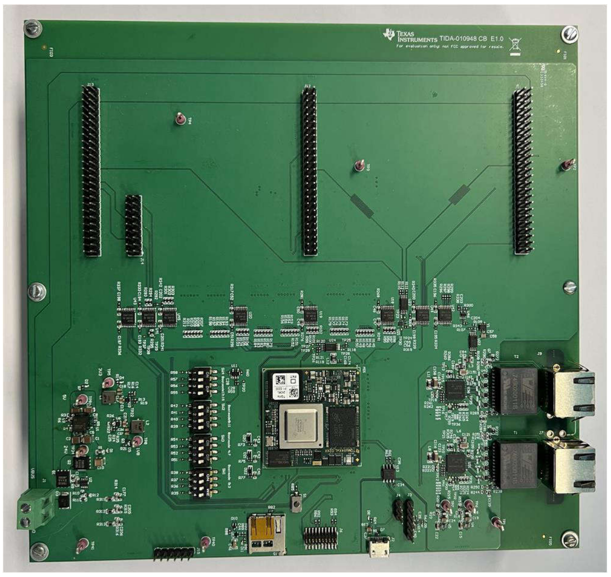


Figure 4-1. TIDA-010948_CB PCB Top Overview

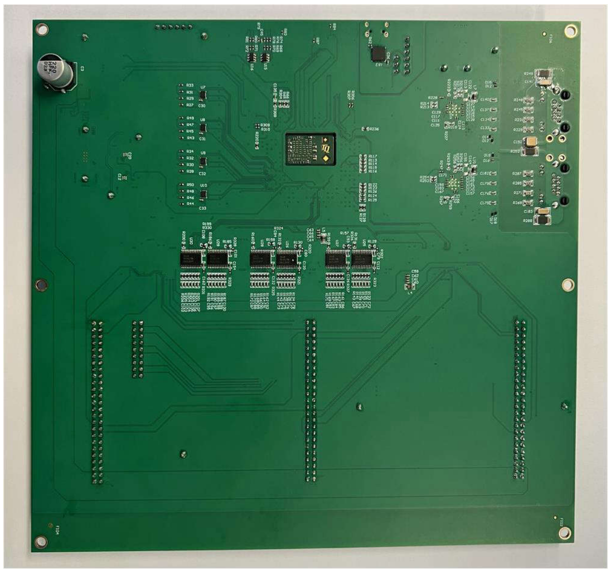


Figure 4-2. TIDA-010948_CB PCB Bottom Overview

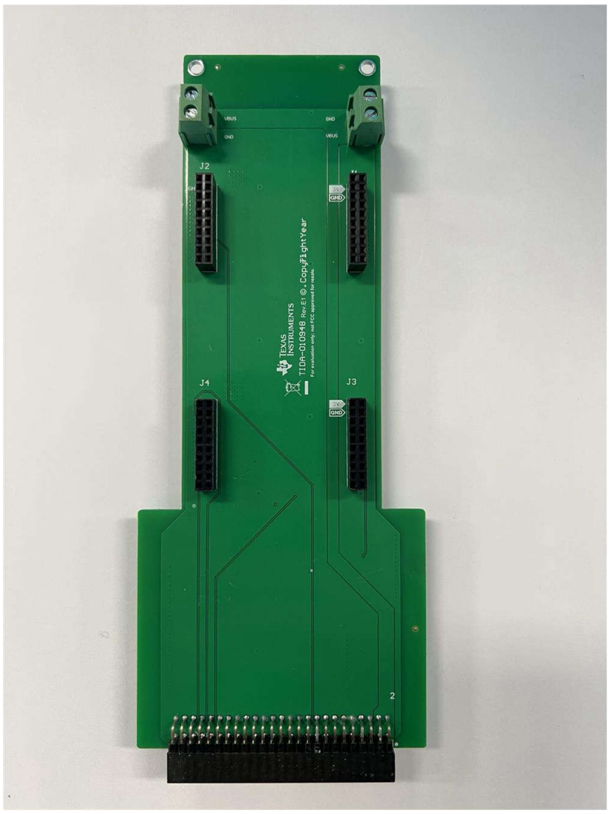


Figure 4-3. TIDA-010948_DB PCB Top Overview

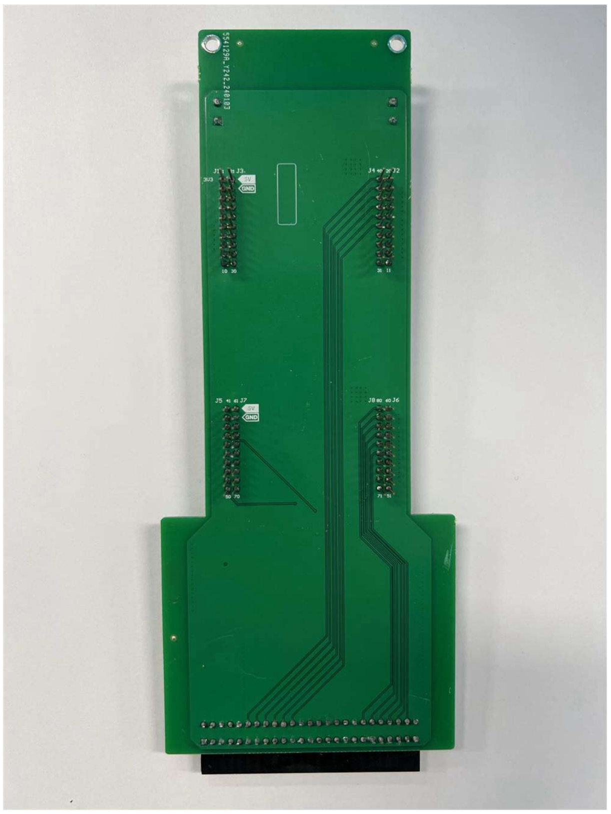


Figure 4-4. TIDA-010948_DB PCB Bottom Overview

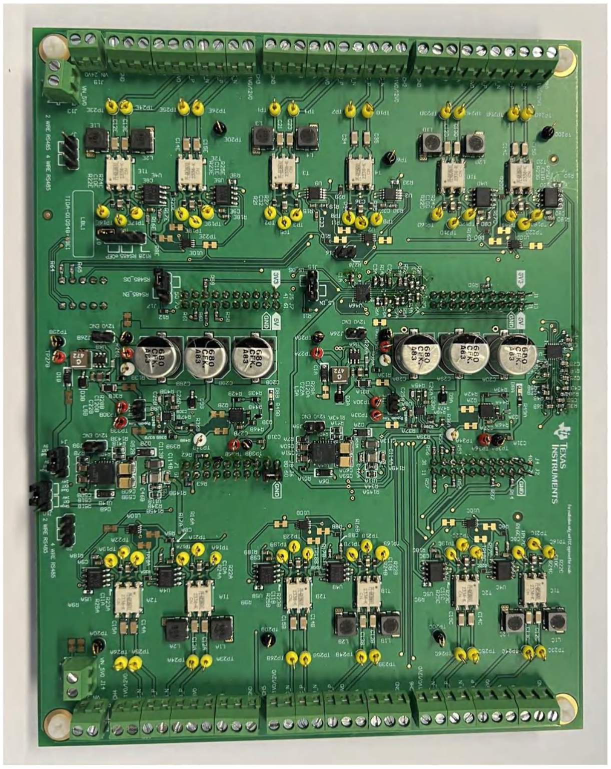


Figure 4-5. TIDA-010948_PB PCB Top Overview

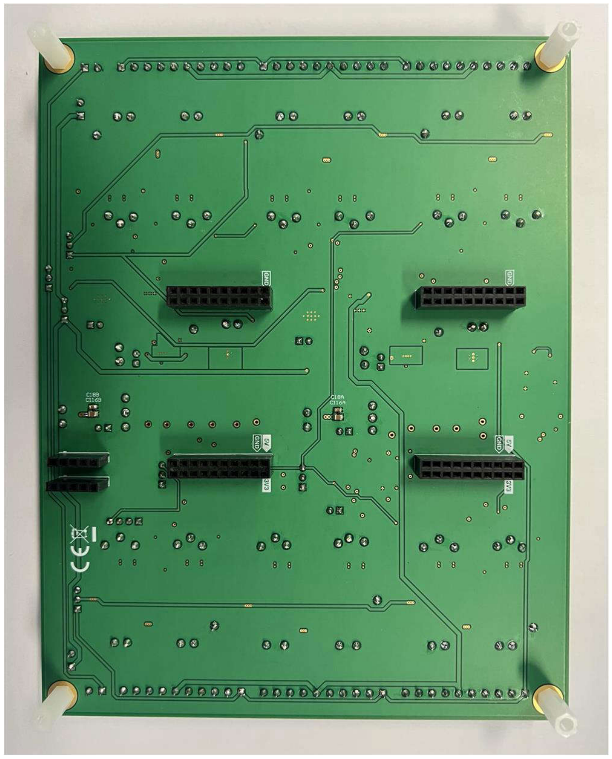


Figure 4-6. TIDA-010948_PB PCB Bottom Overview

Figure 4-7 shows the system demonstration overview.

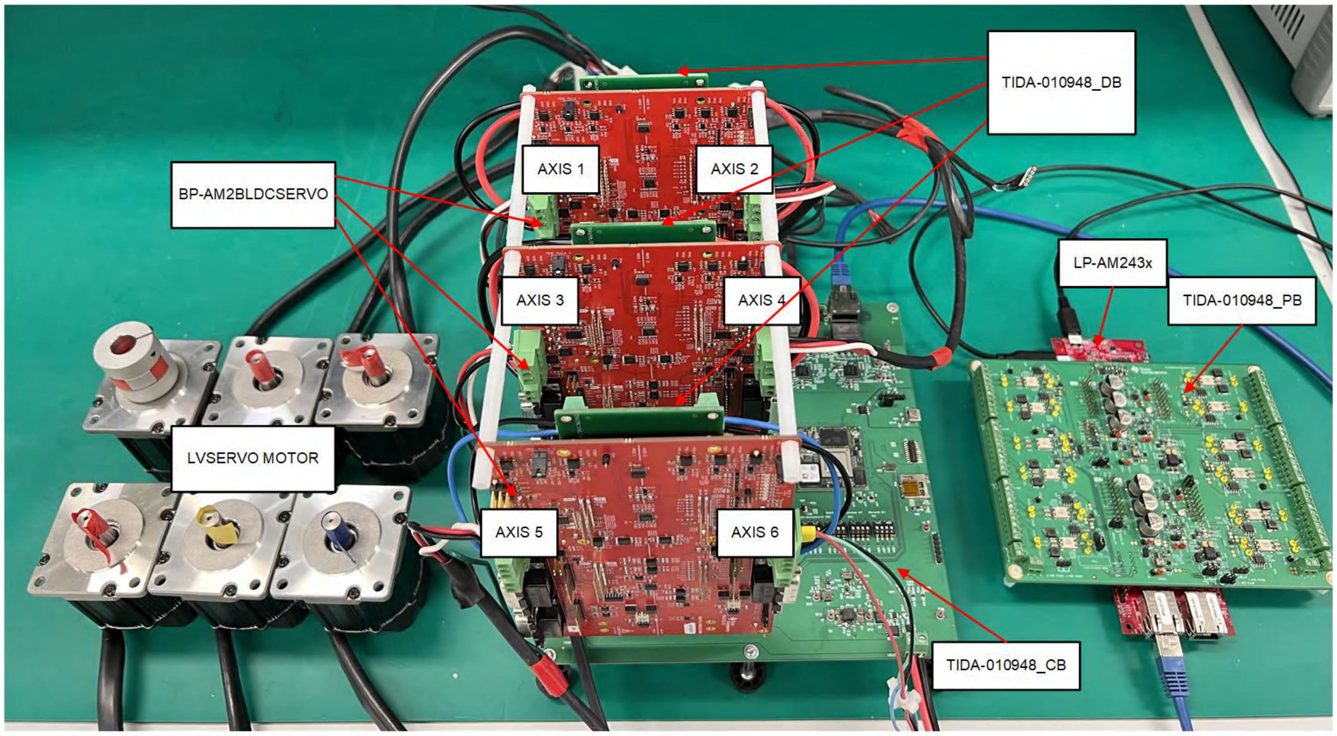


Figure 4-7. TIDA-010948 System Demonstration Overview

4.1.1 System Demonstration Setup

The following steps describe the system demonstration setup:

1. Connect the 3 × adapter boards (TIDA-010948_DB) with 3 × BP-AM2BLDCSERVO boards (BoosterPack board) then connect this setup to the base control board (TIDA-010948_CB). One BoosterPack board plus one adapter board are used as the power stage for two axes (J1–J4 of the adapter board connects to J1–J4 of the BoosterPack board, respectively).
2. Connect J5 of one adapter board to J11 on the control board as the power stage of Axis 1 and Axis 2. J5 of the second adapter board connects to J12 on the control board as the power stage of Axis 3 and Axis 4. J5 of the third adapter board connects to J13 on control board as the power stage of Axis 5 and Axis 6.
3. The J1 connection of the control board is the system input $24V_{DC}$.
4. The J6 connection on the adapter board gets the DC link power from the control board. Make this J6 connection to J5 of the BoosterPack board with cables to pass the power to the BoosterPack board. The J7 connection on the adapter board gets the DC link power from the control board. Make this J7 connection to J6 of the BoosterPack board with cables to pass the power to the BoosterPack board. Complete the same connection of all 3 × adapter boards to power-on all six axes. The distance between two BoosterPack boards is around 83mm, the standoffs are required to fix the installation between the BoosterPack boards plus the adapter boards and the control board. Figure 4-8 and Figure 4-9 shows the connections for the DC link and motor power cables.
5. Control board boot:
   - Short pin 1 and pin 2 of J4 on the control board to supply UART power
   - Connect the USB cable on J2 for the UART terminal
   - Connect the JTAG cable on J6 (default male header is the CM20 pin with 0.05 inch, needs to block pin 6)
   - Set boot mode as UART BOOT, Set SW4 "*0000*", SW2 "*1011*", SW3 "*1100*", SW1 "*1101*"
   - Open the UART terminal, character *C* is printed ever 2 to 3 seconds. Once this process completes, close the UART terminal.
   - Use Python® `uart_uniflash.py` to flash the `SBL_null`. Use the TQ image for the `uniflash`, copy `tq_sbl_uart_uniflash.hs_fs.tiimage` and `default_sbl_null_tq.cfg` to the SDK folder `mcu_plus_sdk\tools\boot\sbl_prebuilt\am243x-evm`

   - Power off the board, set OSPI BOOT MODE by setting SW4 "*0000*", SW2 "*0100*", SW3 "*1110*", SW1 "*1100*" then power on the board. The SBL_NULL information appears in the UART terminal. Figure 4-10 shows the boot mode switch and connections for JTAG and UART.
   - Set the JTAG power voltage to 1.8V in target ccxml file as shown in Figure 4-11.
6. The position board (TIDA-010948_PB) is powered with $5V_{DC}$ on J13. Select the external 5V for the dual channel on J4 and the external 24V for dual channel on J10. Enable the lever shifter on J15 and disable the RS-485 interface on J17. Short J6 Pin 59 to J6 pin 60 (GND) to set the position board as the SORTE_G device. J3A to J3F are used to connect the encoder signals for 6 channels, respectively. J1, J2, J5, J6, J12, and J21 are connected to the LaunchPad LP-AM243x. Figure 4-12 shows the setup of the position board.
7. LP-AM243x is used as the MCU platform for decoding the six axes encoder signals and SORTE_G device. For setup and boot initialization information, see AM243x MCU+ SDK: EVM Setup. As Section 2.3.8 mentions, the LP-AM243x needs to enable the SCI client by pre-loading the configuration through SBL using the `sbl_null_sciclient.release.hs_fs.tiimage` image, copy the image file to the *mcu plus sdk* folder:
`mcu sdk folder\tools\boot\sbl_prebuilt\am243x-lp\`.

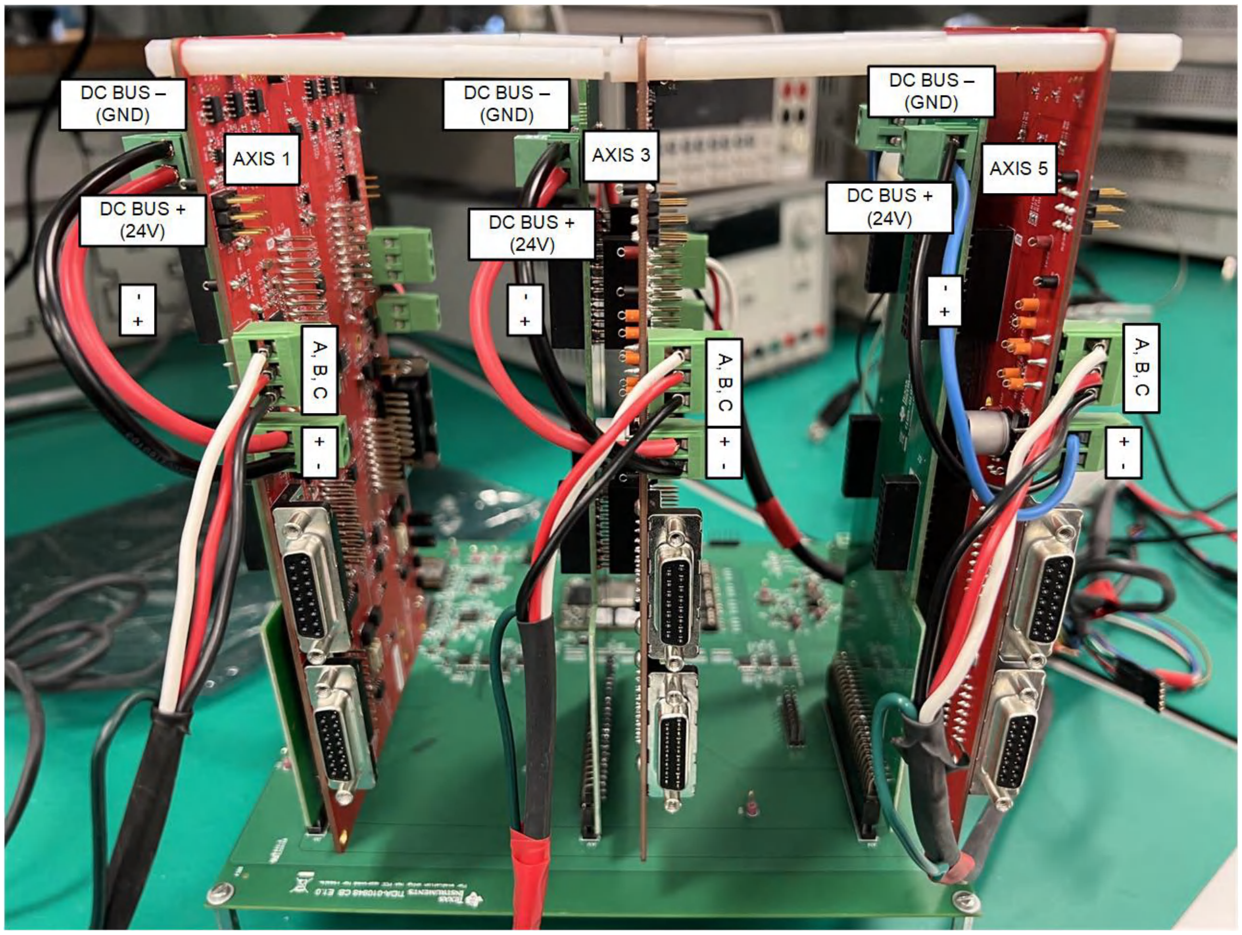


Figure 4-8. DC Link Connection and Motor Phase Power – Axis 1, 3, 5

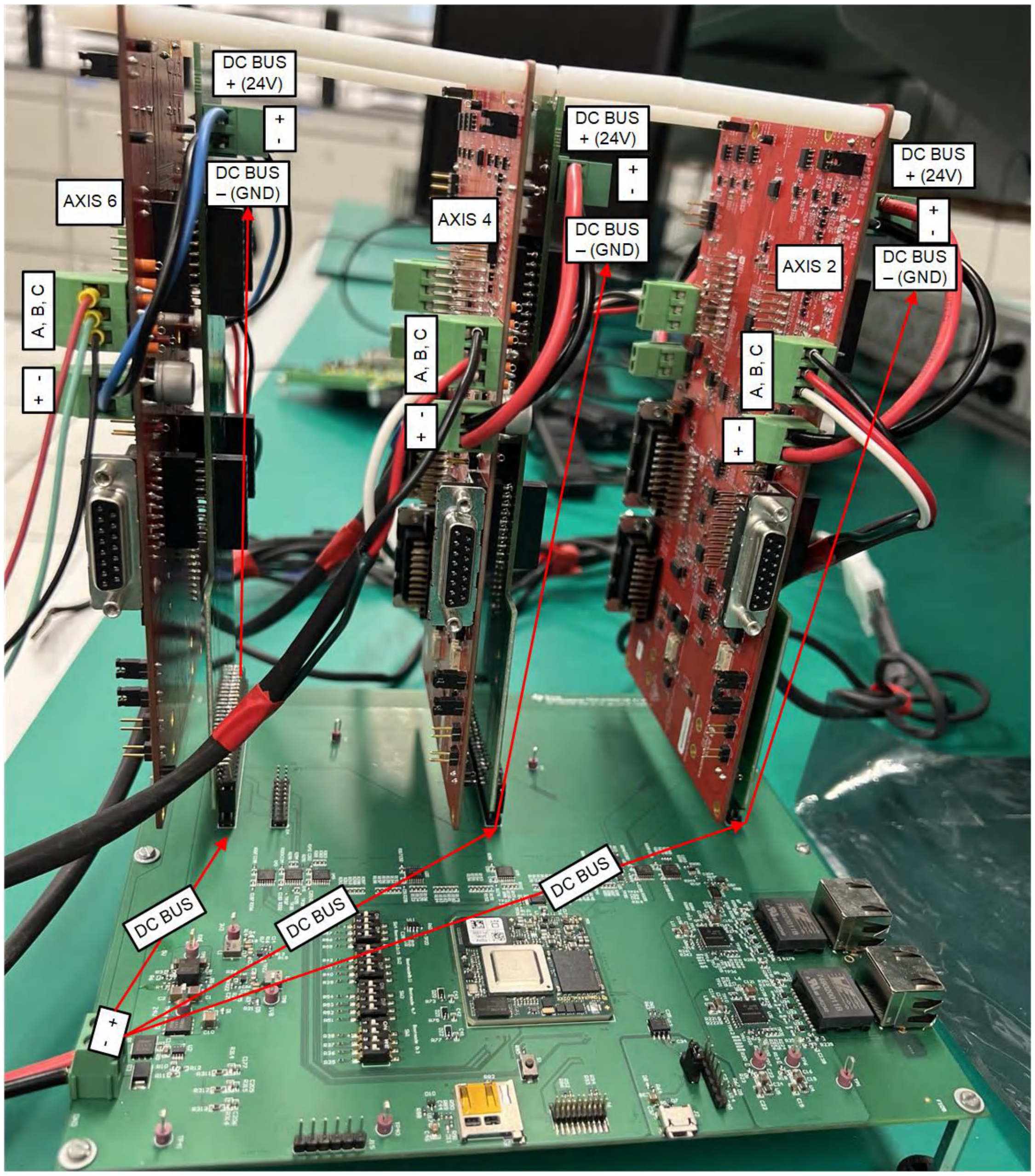


Figure 4-9. DC Link Connection and Motor Phase Power – Axis 2, 4, 6

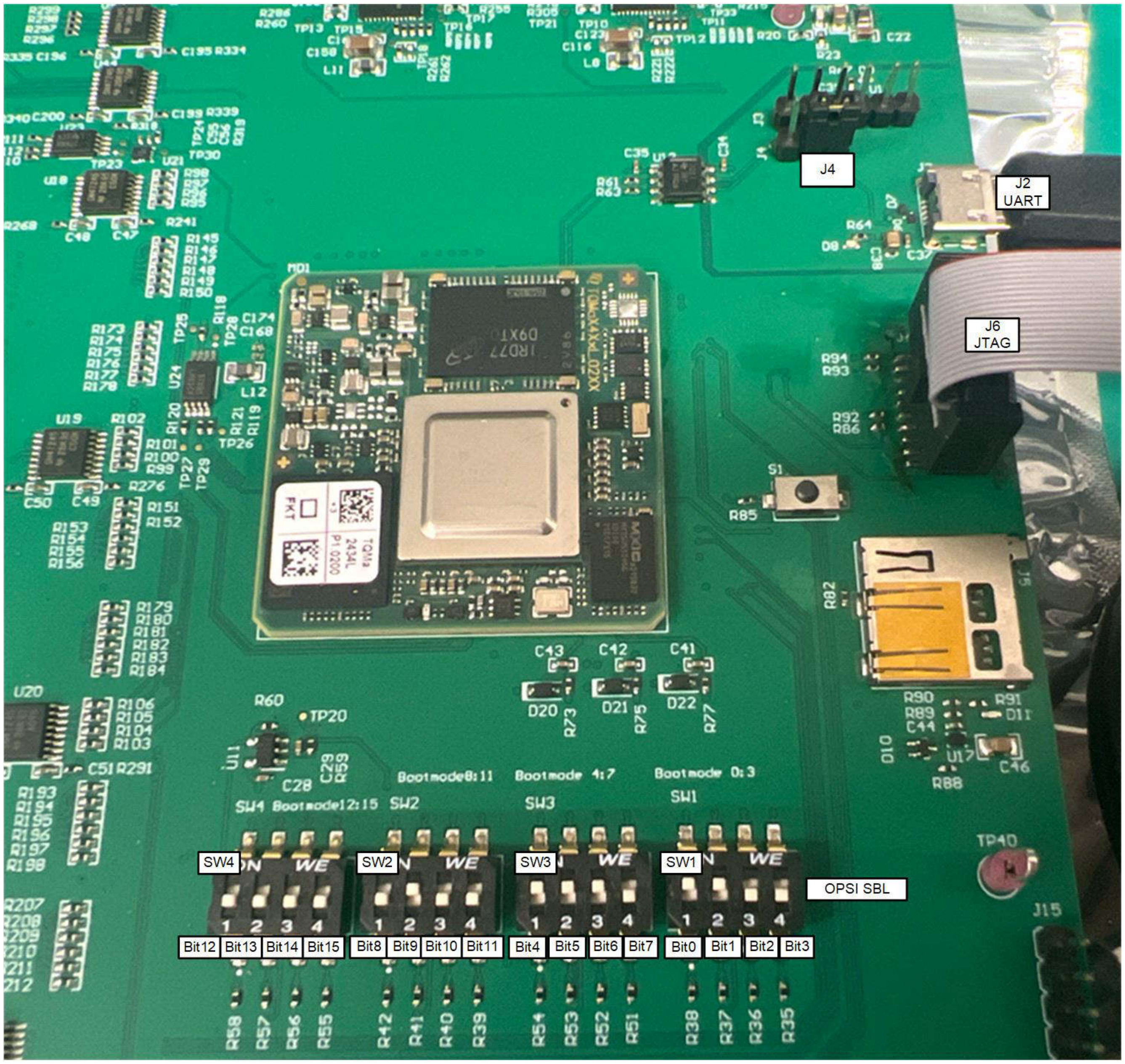
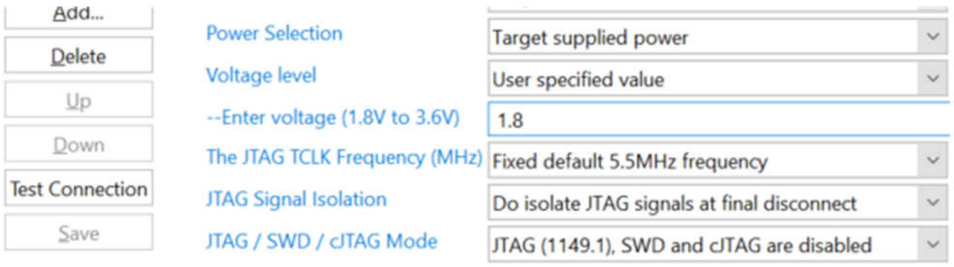


Figure 4-10. Boot Mode Switch and Connection for JTAG and UART on Control Board



Power Selection: Target supplied power
Voltage level: User specified value
--Enter voltage (1.8V to 3.6V): 1.8
The JTAG TCLK Frequency (MHz): Fixed default 5.5MHz frequency
JTAG Signal Isolation: Do isolate JTAG signals at final disconnect
JTAG / SWD / cJTAG Mode: JTAG (1149.1), SWD and cJTAG are disabled

Figure 4-11. Target File Setting for JTAG Power

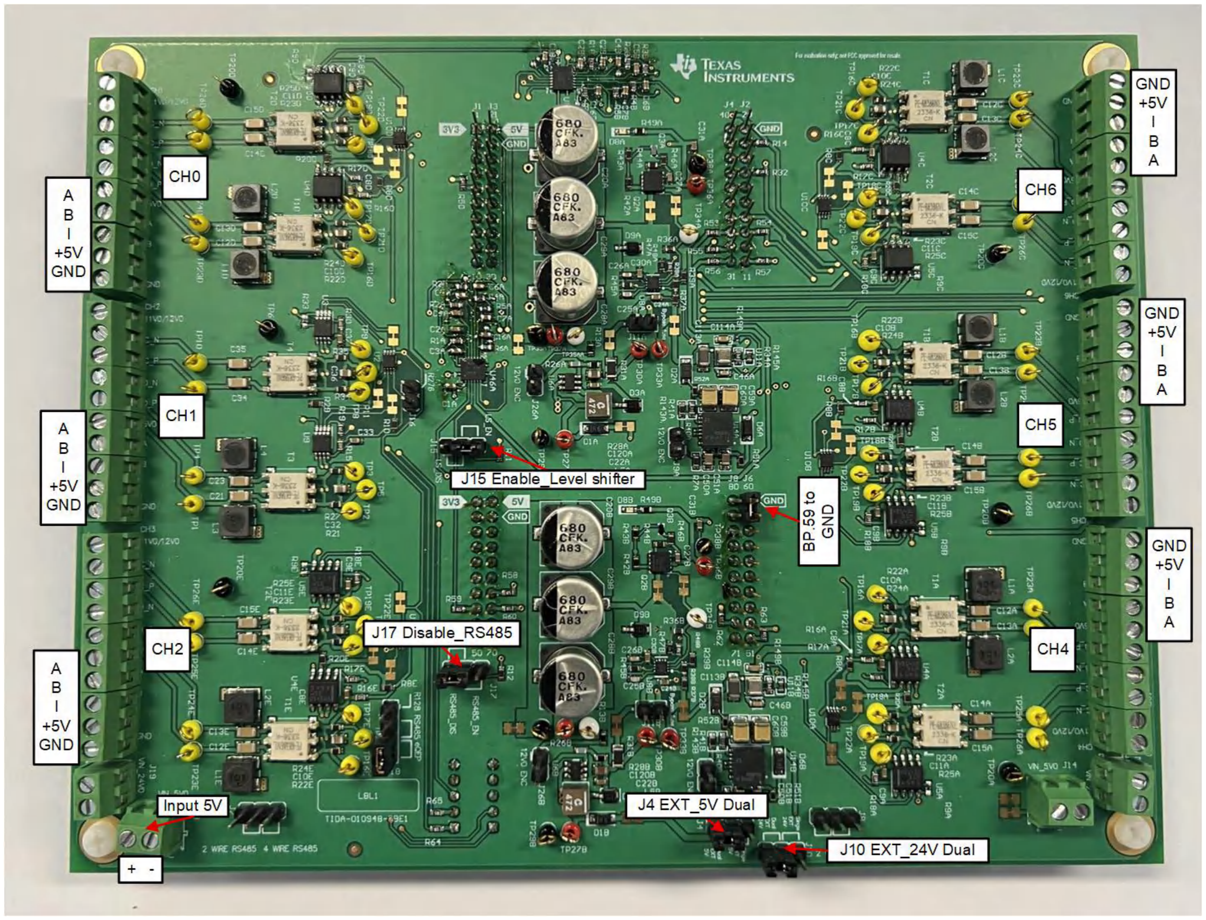


Figure 4-12. Position Board Setup

4.2 Software Requirements

To validate this reference design, a TI internal test software was developed with both the AM243x ALV and ALX package using the `motor_control_sdk_am243x_09_02_00_09` software development kit for AM243x MCU. This software is not available for public use. For AM243x software support, see also MCU-PLUS-SDK-AM243X Software development kit (SDK) and the Arm-based microcontrollers forum - Arm-based microcontrollers - TI E2E support forums.

Table 4-1. Key Software Configuration

SUBSYSTEM	SPECIFICATION	VALUE
SoC EPWM and ICSSG_PRU PWM	Frequency	16kHz
	Sync or phase shift mode	Sync mode
	Count mode	Up-down count
	Deadband	200ns
Current feedback – ICSS SDFM	Normal current OSR	OSR 64
	Sampling mode	Continuous sampling
	Single- or double update capability	Yes
Position feedback – PRU EQEP and SoC QEP	Maximum channels for QEP	6
Control Algorithm – FOC loop	Cycle time	62.5µs or 31.25µs as option for double update
Industrial communication – SORTE_G	Cycle time	62.5µs
	Real-time control	Deterministic network time

4.3 Test Setup and Results

4.3.1 Current Feedback – SDFM

Figure 4-13 shows the SDFM clock and data on the modulator AMC1035 side. Channel 1 with the blue curve is the 20MHz clock generated by the control board ICSSG0_IEP1_SYNC0 after the clock buffer LMK1C1104. Channel 2 with the red curve is the SD data generated by AMC1035.

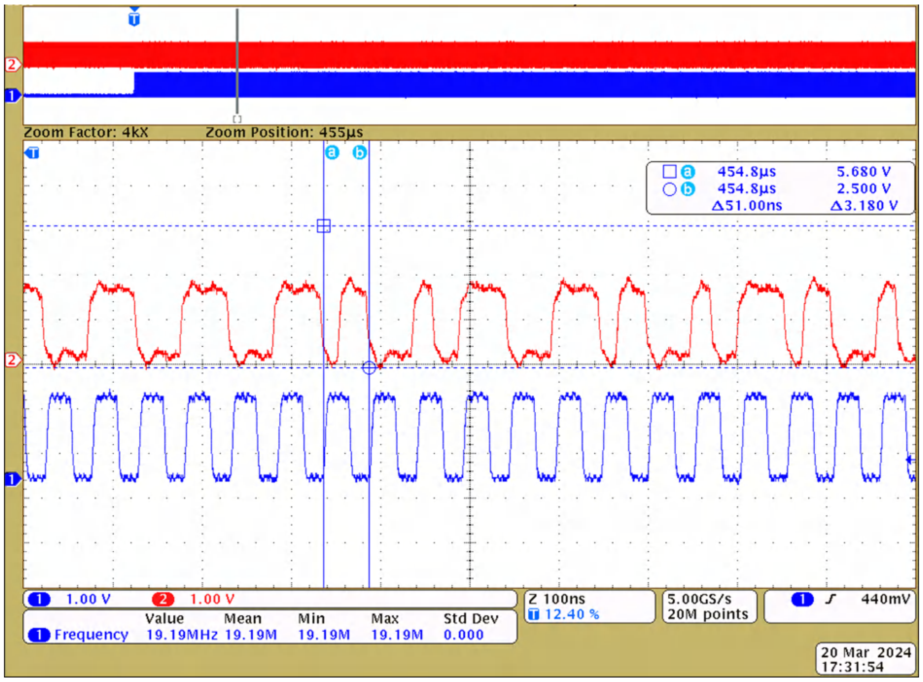


Figure 4-13. SDM Clock and Data Line Signals

Figure 4-14 and Figure 4-15 show the test setup and 2-phase current tested using the current probe with open loop control.

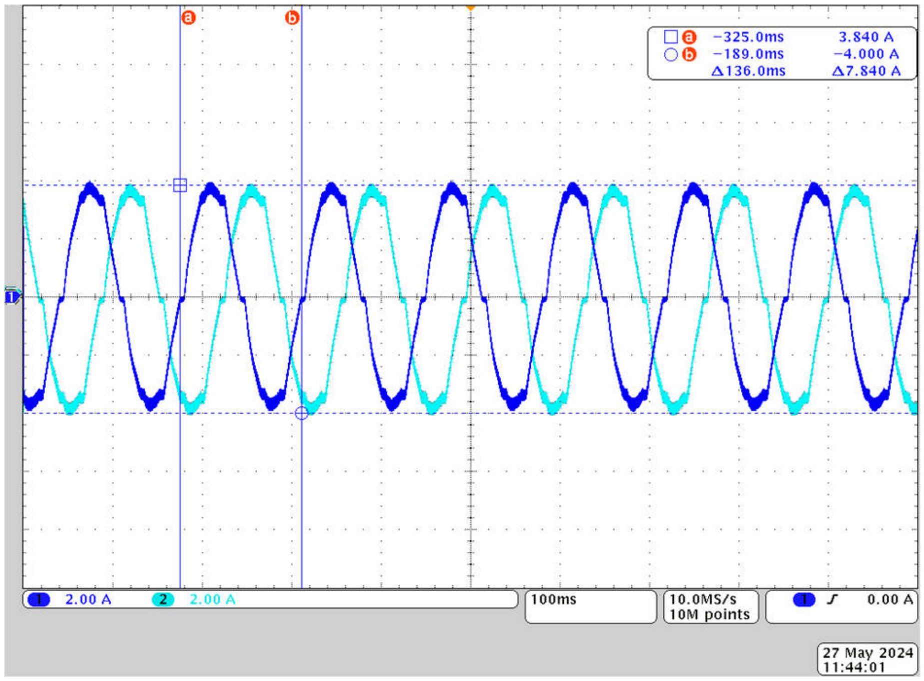


Figure 4-14. Phase A and Phase B Current

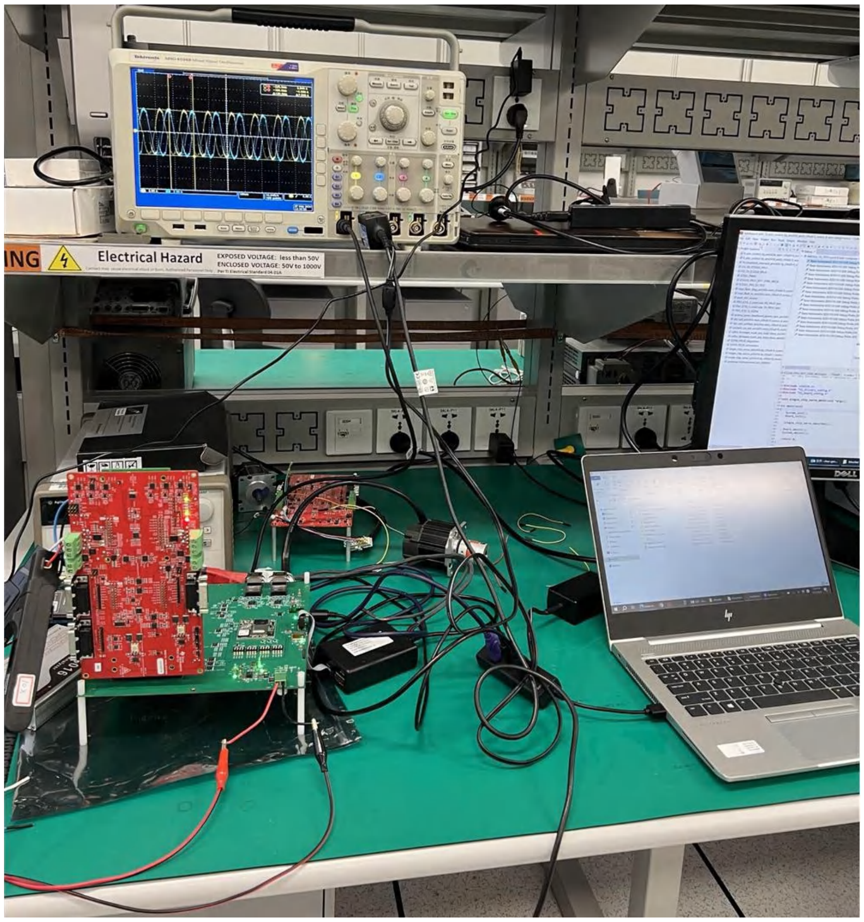
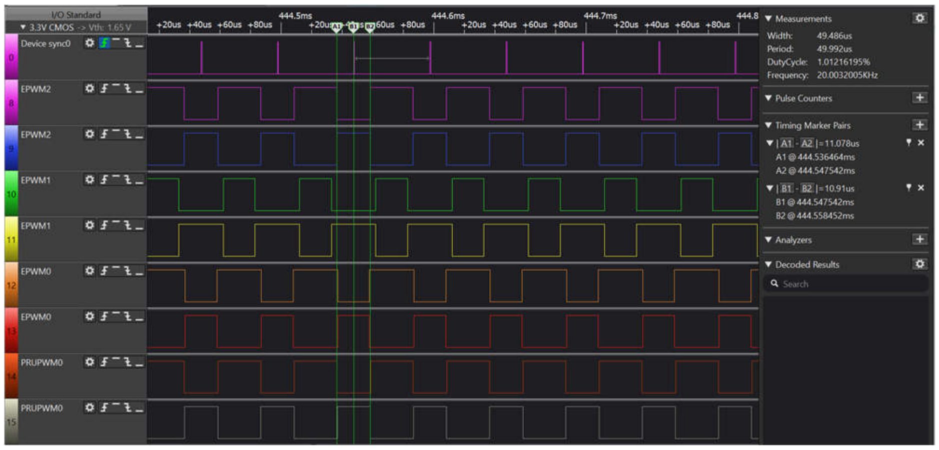


Figure 4-15. SDFM Test Setup

4.3.2 Time Synchronization Between Industrial Ethernet (SORTE_G) and PWM Interface

Figure 4-16 shows the synchronized pulse generated by the ICSSG1_IEP0_SYNCOUT0 on the position board (SORTE_G device), SoC EPWM signals, and ICSSG_PRU_PWM signals. Channel 0 is the Sync pulse on the position board, Channel 8 to 13 are the EPWM0, 1, 2 signals on the control board and Channel 14 to 15 are the ICSSG_PRU_PWM signals on the control board. Figure 2-3 shows the time synchronization flow. The SORTE_G generated the cyclic pulse with pre-defined cycle times to synchronize with EPWM0 through the TSR module. The IEP timer used for ICSSG_PRU_PWM is reset by the EPWM0 SYNC0.



Measurements
Width: 49.486us
Period: 49.992us
DutyCycle: 1.01216195%
Frequency: 20.0032005KHz

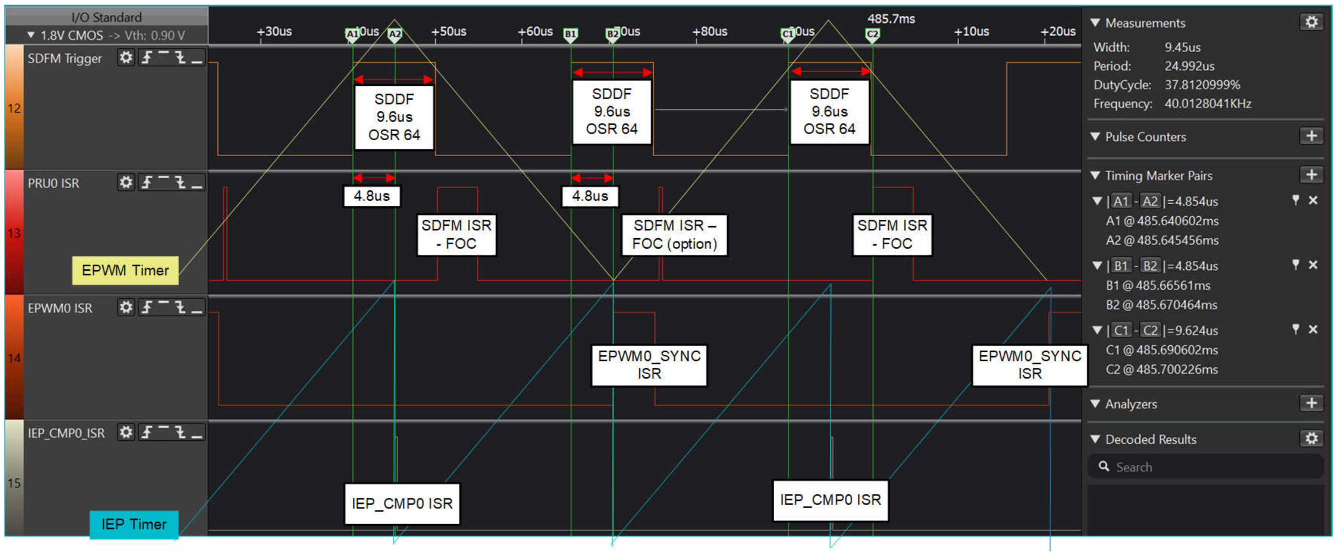
Timing Marker Pairs
| A1 - A2 |=11.078us
A1 @ 444.536464ms
A2 @ 444.547542ms
| B1 - B2 |=10.91us
B1 @ 444.547542ms
B2 @ 444.558452ms

Figure 4-16. Time Synchronization Between Industrial Ethernet (SORTE_G) and PWM Interface

4.3.3 FOC Loop Verification

4.3.3.1 FOC Loop Timing

Figure 4-17 shows the testing data of the timing for the FOC loop (20kHz with double update) with PWM and SDFM interface.



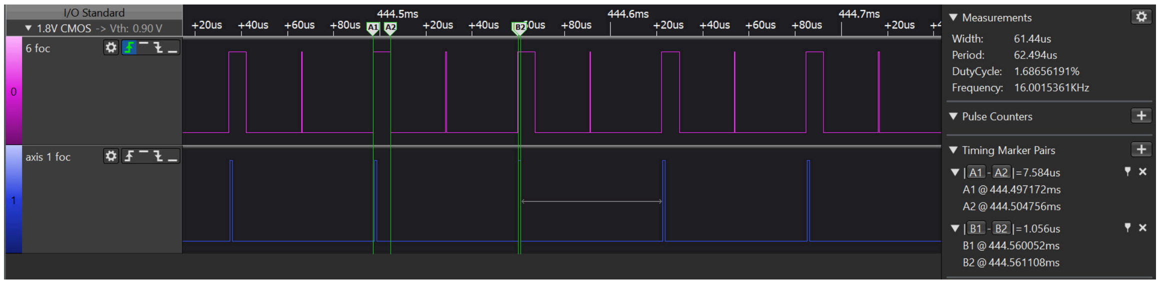
Measurements
Width: 9.45us
Period: 24.992us
DutyCycle: 37.8120999%
Frequency: 40.0128041KHz

Timing Marker Pairs
| A1 - A2 |=4.854us
A1 @ 485.640602ms
A2 @ 485.645456ms
| B1 - B2 |=4.854us
B1 @ 485.66561ms
B2 @ 485.670464ms
| C1 - C2 |=9.624us
C1 @ 485.690602ms
C2 @ 485.700226ms

Figure 4-17. FOC Timing Verification

4.3.3.2 FOC Loop Processing Time Verification

Figure 4-18 shows the testing data of the processing time for the FOC loop with closed-speed loop. One FOC loop calculation time is around 1.056µs and 6 axes closed-speed loop takes around 7.584µs with 62.5µs cycle time.



Measurements
Width: 61.44us
Period: 62.494us
DutyCycle: 1.68656191%
Frequency: 16.0015361KHz

Timing Marker Pairs
| A1 - A2 |=7.584us
A1 @ 444.497172ms
A2 @ 444.504756ms
| B1 - B2 |=1.056us
B1 @ 444.560052ms
B2 @ 444.561108ms

Figure 4-18. 6-Axis FOC Loop Processing Time

4.3.4 Verification for Closed-Loop Control With PI Controller

The PI controller is achieved with the `DCL_runPIParallel()` function for both current and speed loop. The PI constants can be tuned by the `init_pids()` function.

Figure 4-19 shows the verification for the closed-loop FOC block diagram. During closed-speed loop, the I_Q reference is the output of the speed PI controller. But to verify the PI controller, the `gIqRef` is given as a constant target and `parkIqMeasured` is the motor phase current value after the Park and Clarke transform as the feedback value after the PI adjusting. Now, to record the `parkIqMeasured` to see if the variable can follow with the target value `gIqRef`.

Figure 4-20 shows the motor phase A feedback waveform and the step response of the current loop using the PI controller with the following parameters :

- `gPiIq.Kp` = 0.245
- `gPiIq.Ki` = 0.09
- `gPiIq.Umax` = 0.2

- `gPiIq.Umin` = 0.2

Here, the current-loop PI control output is limited to 0.2 according to the DRV8316 capability. For the verification, take `gIqRef` = 0.3 as the current target for the closed current loop. The PI controller with the function is implemented with the following line:

- *parkIqOut = DCL_runPIParallel(&gPiIq, gIqRef, parkIqMeasured);*

The results show that phase current is well controlled by the PI (peak value is about 0.3A which is equals to the target I_Q reference) and the current loop step response time is around 90µs with 32kHz control frequency. So, the current loop bandwidth is around 3.54kHz with Equation 1:

$$\text{Bandwidth} = \frac{1}{\pi \times t_r} \tag{1}$$

where

- t_r = response time

The motor is using LVSERVO with following parameters:

- Resistance, phase to phase = 0.72Ω
- Inductance, phase to phase = 0.4mH
- Electrical Time Constant = 0.56
- Back EMF (V_{peak} / K_{rpm}) = 4.64

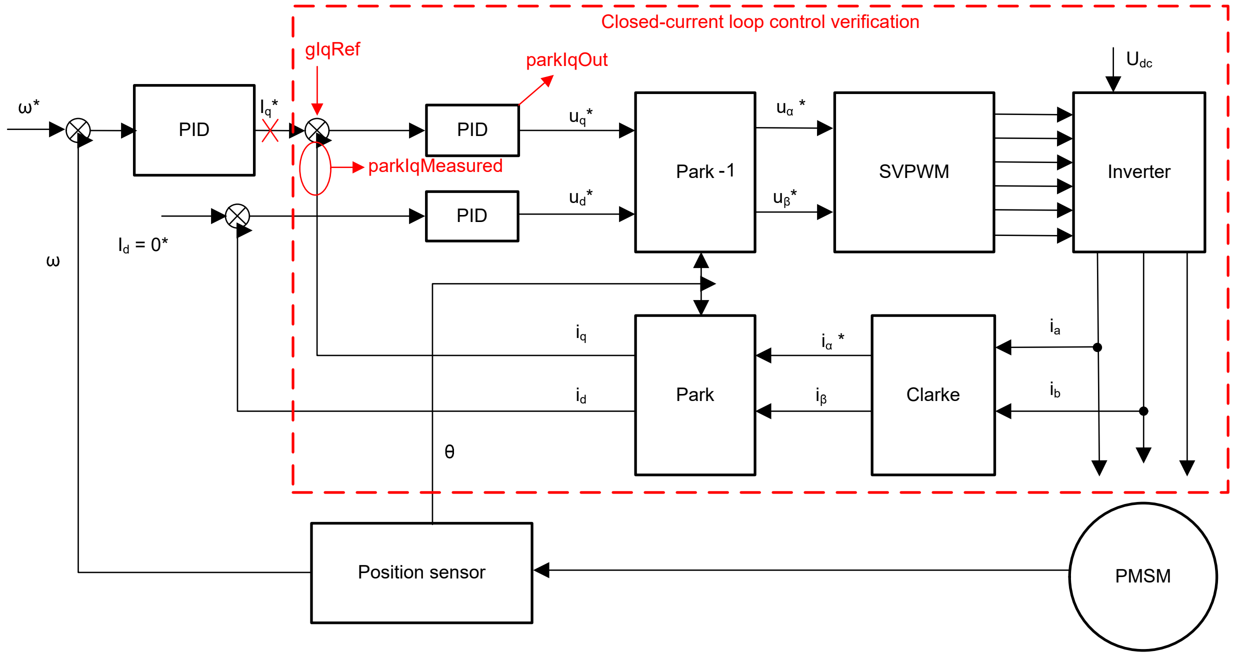


Figure 4-19. Closed-Current Loop With PI Controller Verification Diagram

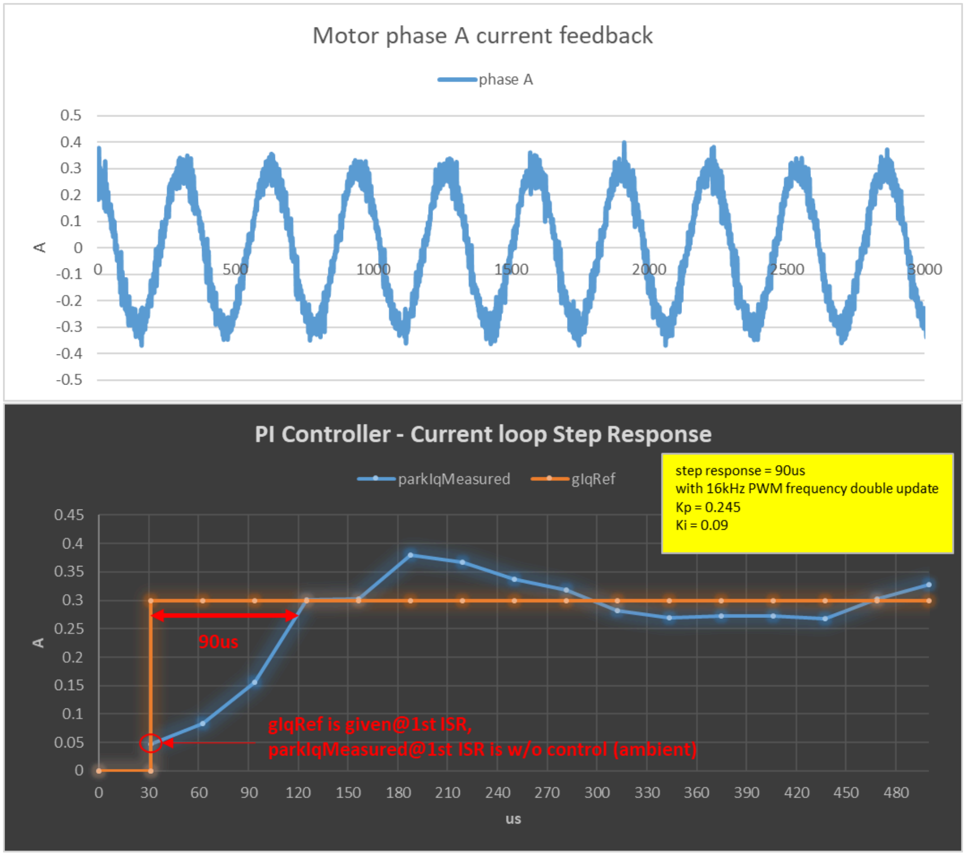


Figure 4-20. Motor Phase Current Feedback and PI Controller Step Time Response

5 Design and Documentation Support

5.1 Design Files

5.1.1 Schematics

To download the schematics, see the design files at TIDA-010948.

To download the BLDC BoosterPack schematics, see the design files at BP-AM2BLDCSERVO Design Package.

To download the AM243x LaunchPad schematics, see the design files at LP-AM243 Design Package.

5.1.2 BOM

To download the bill of materials (BOM), see the design files at TIDA-010948.

To download the BLDC BoosterPack BOM, see the design files at BP-AM2BLDCSERVO Design Package.

To download the AM243x LaunchPad BOM, see the design files at LP-AM243 Design Package.

5.1.3 Layer Plots

To download the layer plots, see the design files at TIDA-010948.

To download the BLDC BoosterPack layer plots, see the design files at BP-AM2BLDCSERVO Design Package.

To download the AM243x LaunchPad layer plots, see the design files at LP-AM243 Design Package.

5.1.4 Altium Project

To download the Altium Designer® project files, see the design files at TIDA-010948.

To download the BLDC BoosterPack Altium project files, see the design files at BP-AM2BLDCSERVO Design Package.

To download the AM243x LaunchPad Altium project files, see the design files at LP-AM243 Design Package.

5.1.5 Gerber Files

To download the Gerber files, see the design files at TIDA-010948.

To download the BLDC BoosterPack Gerber files, see the design files at BP-AM2BLDCSERVO Design Package.

To download the AM243x LaunchPad Gerber files, see the design files at LP-AM243 Design Package.

5.1.6 Assembly Drawings

To download the assembly drawings, see the design files at TIDA-010948.

To download the BLDC BoosterPack assembly drawings, see the design files at BP-AM2BLDCSERVO Design Package.

To download the AM243x LaunchPad assembly drawings, see the design files at LP-AM243 Design Package.

5.2 Tools and Software

Tools

CCSTTUDIO	Code Composer Studio™ integrated development environment (IDE): download CCS 20.0.0 version for Microsoft® Windows® or Linux®
ARM-CGT-CLANG	Arm® code generation tools - compiler: download TI ARM CLANG 3.2.0.LTS for Microsoft Windows or Linux
SYSCONFIG	Standalone desktop version of SysConfig: download SysConfig 1.22.0 for Microsoft Windows or Linux

Software

AM243x Motor Control SDK	Motor Control SDK Microsoft Windows Installer
AM243x Industrial Communication SDK	Industrial Communications SDK Microsoft Windows Installer
AM243x MCU+ SDK	MCU PLUS SDK Microsoft Windows Installer

5.3 Documentation Support

1. Texas Instruments, *AM64x /AM243x Processors Silicon Technical Reference Manual*
2. Texas Instruments, *AM2x BLDC Servo Motor BoosterPack (BPAM2BLDCSERVO) EVM User's Guide*
3. Texas Instruments, *TIDEP-01032 EtherCAT® connected, single-chip, dual-servo motor drive reference design*
4. Texas Instruments, *TIDEP-01032 Example under AM243x Motor control SDK 09.02.00*

5.4 Support Resources

TI E2E™ support forums are an engineer's go-to source for fast, verified answers and design help — straight from the experts. Search existing answers or ask your own question to get the quick design help you need.

Linked content is provided "AS IS" by the respective contributors. They do not constitute TI specifications and do not necessarily reflect TI's views; see TI's Terms of Use.

5.5 Trademarks

Sitara™, E2E™, BoosterPack™, LaunchPad™, Code Composer Studio™, and TI E2E™ are trademarks of Texas Instruments.
EtherCAT® is a registered trademark of Beckhoff Automation GmbH.
PROFINET® is a registered trademark of PROFIBUS Nutzerorganisation e.V..
Arm® is a registered trademark of Arm Limited.
Python® is a registered trademark of Python Software Foundation.
Altium Designer® is a registered trademark of Altium LLC.
Microsoft® and Windows® are registered trademarks of Microsoft Corporation.
Linux® is a registered trademark of Linus Torvalds.
All trademarks are the property of their respective owners.

6 About the Author

CHEN GAO is a System Engineer in the Industrial Systems Motor Drive team at Texas Instruments and is responsible for specifying and developing reference designs for industrial drives and robotics

SABARI KANNAN MUTHALAGU is a System Engineer in the Industrial Systems Robotics team at Texas Instruments and is responsible for specifying and developing reference designs for robotics

THOMAS LEYRER is a System Architect and Distinguished Member of Technical Staff in the Industrial Systems Factory Automation Control team at Texas instruments, where he is responsible for specifying and developing reference designs for industrial drives, robotics and factory automation

Recognition:

The authors recognize the excellent contribution from **Pratheesh Gangadhar TK**, **Dhaval Khandla**, **Achala Ram**, and **Manoj Koppolu** to support the software development of the TIDA-010948 reference design