AN13569

Dual Servo Motor Demo on LPC553x/LPC55S3x

Rev. 2 — 15 November 2023 Application note

Document information

Information	Content
Keywords	AN13569, LPC553x/LPC55S3x, LPCXpresso55S36, motor control applications, FRDM-MC-PMSM driver boards, 3-phase servo motors
Abstract	This application note describes a dual servo motor demo that uses an NXP board (LPCXpresso55S36) based on the NXP LPC553x/LPC55S3x processor.

1 Introduction

This application note describes a dual servo motor demo that uses an NXP board (LPCXpresso55S36) based on the NXP LPC553x/LPC55S3x processor. It can also be used as a reference to develop motor control applications based on other products.

The LPC553x/LPC55S3x processor uses a single Arm Cortex-M33 core, which operates at speeds of up to 150 MHz. The LPC553x/LPC55S3x processor improves product architecture and integration, reduces power consumption, and provides advanced security features that make it an ideal processor for numerous high-performance applications.

This demo includes one LPCXpresso55S36 board, two FRDM-MC-PMSM driver boards, and two 3-phase servo motors. The LPC553x/LPC55S3x processor samples the currents and voltages of the motors through its Analog-to-Digital Converter (ADC) module. The Quadrature Decoder (ENC) module of the processor receives the encoder signal to obtain the rotor position and speed, and generates pulse width modulation (PWM) based on the field-oriented control (FOC) algorithm to drive the motor. At the same time, the Universal Asynchronous Receiver/Transmitter (UART) module can be used to communicate with the FreeMASTER for command dispatch, variable observation, and other functions that users can debug easily. Finally, precise position control and smooth speed regulation can be achieved.

System structure and software describes the system structure and software of the dual servo motor demo. Key peripherals configuration introduces the peripherals configuration of the dual servo motor demo. Demo operation describes how to operate the dual servo motor demo.

2 System structure and software

This section is dedicated to the specifics of the system and software.

2.1 System structure

Figure 1 presents the system structure block diagram of this dual servo motor demo.

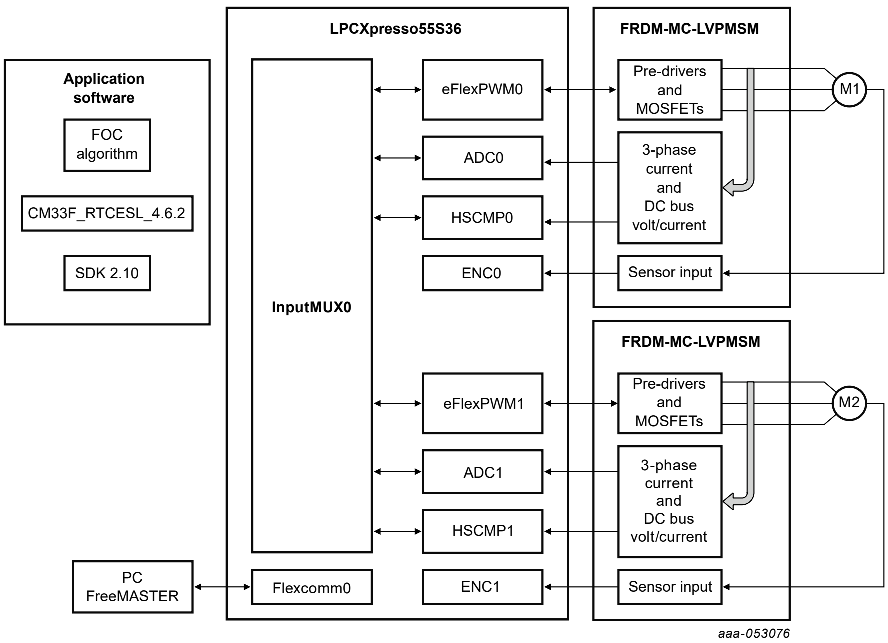

Figure 1. System structure block diagram

- LPCXpresso55S36 designed by NXP contains the LPC553x/LPC55S3x chip and peripheral interfaces.
- FRDM-MC-LVPMSM designed by NXP is a motor driver board that contains driver bridges, analog sampling circuits, and an encoder interface.
- M1 and M2 are servo motors, each with a 1000-line encoder.
- Enhanced Flex Pulse Width Modulator (eFlexPWM), ENC, and ADC are on-chip peripherals used for motor control, encoder signal acquisition, and analog acquisition, respectively.
- Input Multiplexing (INPUTMUX) is an input multiplexing module that can provide different signal path options for the internal peripherals of the chip. In this demo, it is responsible for providing signal connections for PWM synchronization, ADC hardware triggering, and fault protection.
- The application software is running on LPC553x/LPC55S3x that includes the FOC algorithm, CM33_RTCESL_4.6.2 (Real-Time Control Embedded Software Motor Control and Power Conversion Libraries), and SDK 2.14.0.
- Flexible serial communication (Flexcomm) modules provide various peripheral function options that can be configured into the Universal Synchronous/Asynchronous Receiver/Transmitter (USART), Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), and Inter-IC Sound (I2S) functions through software. Here, the USART function is configured to implement the communication between the FreeMASTER debugging tool and LPC553x/LPC55S3x to demonstrate user operation.

2.2 Servo control structure

As shown in Figure 2, the control block diagram of servo control in this demo is a classical three-loop structure.

The innermost loop is the current control loop (fast loop), which contains analog signal sampling, the FOC algorithm, and the PWM duty update.

The middle loop is the speed control loop. The comparison between the desired speed and the measured speed obtained with the speed measurement method generates a speed error. The speed error is input to the speed PI controller, generating a new desired value for the torque-producing component of the stator current.

The outermost loop is the position control loop. The position command is entered from the high-level application layer. The comparison between the actual position speed command and the measured position generates a position error. The position error is input to the position controller, generating a new reference speed.

Note: *For the smooth and stable operation of the system, the position reference must go through path planning. If the position reference signal is discontinuous, increase the ramp and trajectory filter processing before the position controller.*

In addition, a feedforward controller is added to the position control loop. It is a control system that works according to the change of a disturbance or given value. When a disturbance is generated and the controlled variable does not change, the feedforward controller is adjusted according to the disturbance to compensate for the influence on the controlled variable. In this application, the differential of the position desired is used as the feedforward input, and the output is the reference speed. After reasonable debugging, it can achieve a better position tracking effect, reduce position error, and improve response speed.

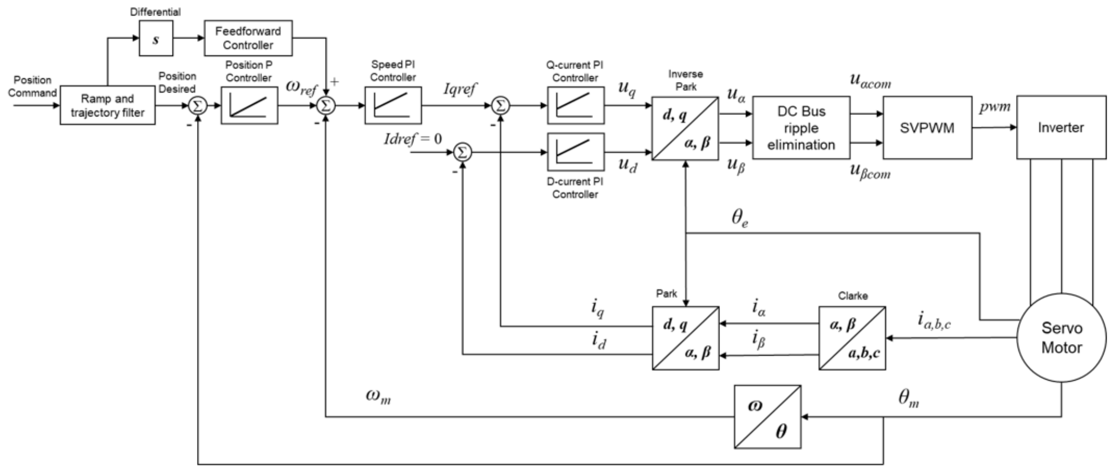

Figure 2. Servo control block diagram

Among them, FOC is a vector control technology based on the stator magnetic field orientation. The basic idea is to decouple the stator current into two components:

- One that controls the magnetic field
- Other that controls the torque

After decoupling, the two current components are controlled independently and do not interfere with each other. Now, the controller structure of the motor is as simple as a separately excited DC motor controller.

To achieve FOC control, perform the following steps:

1. Detect the physical quantities of the motor (phase currents, voltage, and rotor position).
2. Transform the 3-phase stator currents to the two-phase coordinate system (α, β) using the Clarke transform.
3. Use the Park transformation to transform the (α, β) axis stator current rotation to the (d, q) coordinate system.
4. Independently control the torque current component (i_{sq}) and the excitation current component (i_{sd}).
5. Calculate the output stator voltage space vector using the decoupling block.
6. Transform the stator voltage space vector from the (d, q) coordinate system to the (α, β) coordinate system through the inverse Park transformation.
7. Use space vector modulation to generate a 3-phase voltage output.

To decompose the stator current into torque components and magnetic flux components, you must know the position of the excitation magnetic flux of the motor. It requires you to accurately detect the position and speed information of the rotor. In the current example, the rotor position and speed information are obtained by collecting the output signal of the motor encoder through the ENC module in LPC553x/LPC55S3x. For more details, see Section 3.6.

3 Key peripheral configuration

The current dual servo motor demo uses only the essential peripherals for the dual motor control technique implemented in the application code.

3.1 System Configuration (SYSCON)

The SYSCON module has multiple functions, such as system and bus control, clock selection and control, phase-locked loop configuration, and reset wake-up control. The main function in the current design is to generate and control the clock of each submodule and manage its working mode.

The motor control demo has the following two clock sources:

- PLL0: It is sourced from an external crystal oscillator having a clock frequency of 150 MHz. It is obtained after frequency multiplication and division of a phase-locked loop.
- FRO_HF: It is generated by the on-chip crystal oscillator having a clock frequency of 96 MHz. It is frequency-divided.

The Arm core works at a frequency of 150 MHz and the clock source is PLL0. For this setting, the following registers have been configured in `clock_config.c`:

- MAINCLKSELB[SEL]
- AHBCLKDIV[DIV]

The clock sources for different LPC553x/LPC55S3x modules that used for motor control are as follows:

- The ENC module uses the same clock source as the Arm core, and the frequency is 150 MHz.
- Standard counter/timers (CTIMER) modules are clocked by `fro_hf` with a frequency of 96 MHz.
- ADC modules are clocked by `fro_hf` and then they work at the frequency of 48 MHz (divide by 2).
- Flexcomm modules are clocked by `fro_hf_div` with a frequency of 48 MHz.
- Digital-to-Analog Converter (DAC) modules are clocked by `main_clk` and then they work at the frequency of 12.5 MHz (divide by 12).

Figure 3 shows the clock structure diagram used for the motor control peripherals in this application.

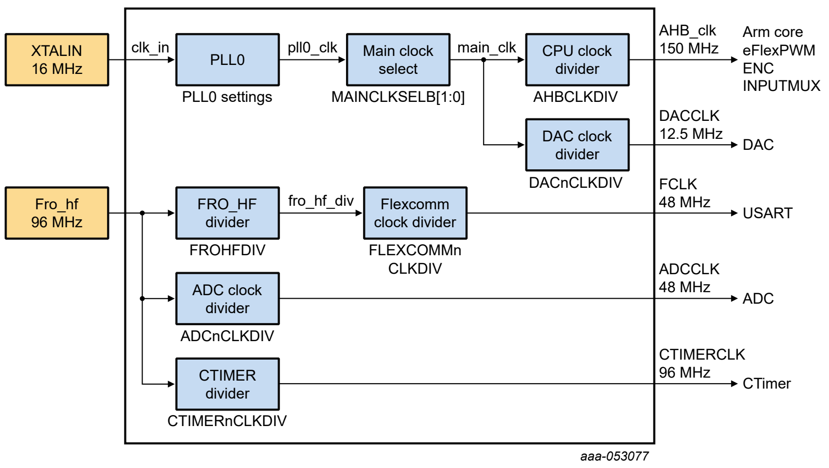

Figure 3. LPC553x/LPC55S3x clock source for motor control peripherals

3.2 Analog sensing (ADC)

ADC0 and ADC1 are used for the motor control analog sensing of currents and DC-bus voltage:

- The clock frequency for ADC0 and ADC1 is 48 MHz. It is taken from `FRO_HF` and is divided by 2.
- The ADCs operate as 16-bit converters with the single-ended conversion and hardware trigger selected.
- The ADC trigger sources are routed through the INPUTMUX module, and the trigger signal comes from the eFlexPWM module. The `TCTRL[HTEN]` register must be set for this function.
- Each ADC module has two independent result FIFOs, each containing 16 entries.

Both ADCs have their own trigger chains. After trigger sampling, the phase current and the bus voltage are collected and stored in the result queue buffer in sequence.

Because the phase current is measured when the bottom transistor is turned on. Therefore, if the duty cycle is high (the voltage value is in the maximum area of the sine curve) as shown in Figure 4, the current can be measured only for a short period of time.

To obtain a stable sampling resistor voltage drop, the bottom transistor must be turned on for at least a critical pulse width. Therefore, for the current example, the dual single-ended sampling mode of the ADC is used to sample the two-phase currents in parallel at the same time. The third-phase current is calculated based on the formula. The channel is selected according to the sector that generates the stator voltage space vector. This assignment is performed at the end of the ADC interrupt service routine. The ADC channel configuration is shown in Table 1.

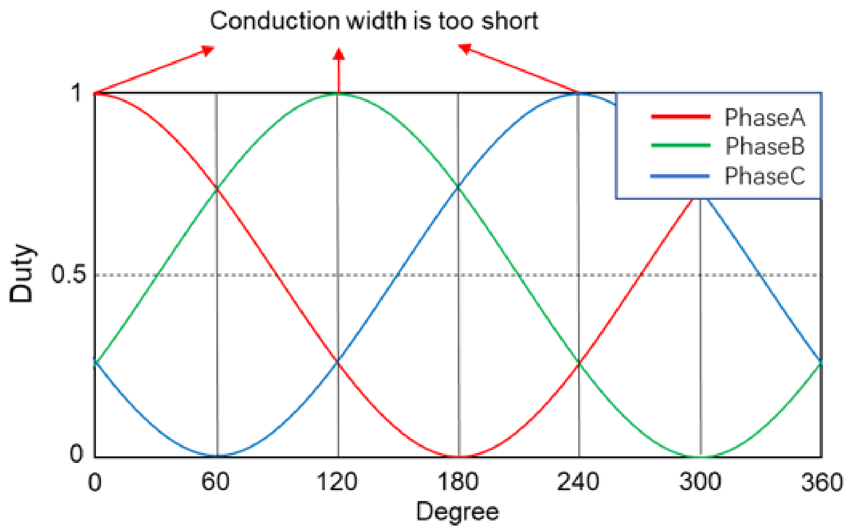

Figure 4. PWM duty and electric degree

Table 1. ADC sample channel configuration

Sector number	ADC channel A	ADC channel B
Sector 1	Phase B	Phase C
Sector 2	Phase A	Phase C
Sector 3	Phase A	Phase C
Sector 4	Phase A	Phase B
Sector 5	Phase A	Phase B
Sector 6	Phase B	Phase C

To implement the function of flexible sampling in dual single-ended mode, you must:

- Enable the alternate channel function in the `CMDLx[ALTBEN]` register.
- Set the required B channel number in the `CMDLx[ALTB_ADCH]` register.

When reading the ADC result register, calculate the complete 3-phase current value according to the sector where the current support vector machine (SVM) is located, using the following formula:

$$i_A + i_B + i_C = 0$$

Each of the two trigger queues has its own conversion completion interrupt. After the ADC0 conversion is complete, the ADC0 interrupt gets triggered and enters `ADC0_IRQHandler` to execute the fast control loop of motor 1. After ADC1 conversion is complete, ADC1 interrupt gets triggered and enters `ADC1_IRQHandler` to execute the fast control loop of motor 2.

3.3 Enhanced flex pulse width modulator (eFlexPWM)

The eFlexPWM contains four PWM submodules. Each submodule is set up to control a single half-bridge power stage. Fault channel support is provided. This PWM module can generate various switching patterns, including highly sophisticated waveforms. It can enable the generation of 3-phase PWM signals connected to the MOSFET H-bridge via predrivers.

Each eFlexPWM module has a 16-bit counter that counts only in the upward direction to the VAL1 value and then resets to the INIT value. During the counting process, the counter value is compared with the value in the VAL2/VAL3 register to control the high and low switching of the output level.

If the count range is a multiple of 2, you can set the INIT and VAL1 values to be 0-centered opposite numbers. Similarly, you can set the VAL2 and VAL3 values to the same number, but with different signs. If the signal edges of all submodules follow the same convention, the signals are center-aligned to one another.

The three PWM submodules of motor 1 used in the current demo are configured as follows:

- PWM0 submodule 0 configuration:
  - The IPBus clock source is 150 MHz.
  - The output frequency of the PWM0 submodule 0 waveform is 16 kHz with 62.5 μs period.
  - The INIT register is set to -4687, and the VAL1 register is set to 4686.
  - Complementary mode with 0.5 μs dead time
  - During every PWM cycle, this PWM submodule sends a reload signal to other submodules for synchronization.
  - Trigger one signal from VAL0 (0) for providing synchronization with PWM1 of motor 2 via the INPUTMUX module.
- PWM0 submodule 1 configuration:
  - PWM0 submodule 0 is the clock source for this submodule.
  - The output frequency of the submodule waveform is 16 kHz with 62.5 μs period.
  - The INIT register is set to -4687, and the VAL1 register is set to 4686.
  - Complementary mode with 0.5 μs dead time
  - During every PWM cycle, a reload signal is received from submodule 0 for synchronization.
  - Trigger one signal from VAL4 (-4687) for providing sample synchronization with the ADC module via the INPUTMUX module.
- PWM0 submodule 2 configuration:
  - PWM0 submodule 0 is the clock source for this submodule.
  - The output frequency of the submodule waveform is 16 kHz with 62.5 μs period.
  - The INIT register is set to -4687, and the VAL1 register is set to 4686.
  - Complementary mode with 0.5 μs dead time
  - During every PWM cycle, a reload signal is received from submodule 0 for synchronization.

The three PWM submodules of motor 2 used in the current demo are configured as follows:

- PWM1 submodule 0 configuration:
  - The IPBus clock source is 150 MHz.
  - The output frequency of PWM1 submodule 0 waveform is 16 kHz with 62.5 μs period.
  - The INIT register is set to -4687, and the VAL1 register is set to 4686.
  - Complementary mode with 0.5 μs dead time

  - The EXT_SYNC signal from PWM0 causes initialization.
  - During every PWM cycle, this PWM submodule sends a reload signal to other submodules for synchronization.
  - Trigger one signal from VAL4 (-4687) for providing sample synchronization with the ADC module via the INPUTMUX module.
- PWM1 submodule 1 / submodule 2 configuration:
  - PWM1 submodule 0 is the clock source for this submodule.
  - The output frequency of the submodule waveform is 16 kHz with 62.5 µs period.
  - The INIT register is set to -4687, and the VAL1 register is set to 4686.
  - Complementary mode with 0.5 µs dead time
  - The EXT_SYNC signal from PWM0 causes initialization.
  - During every PWM cycle, a reload signal is received from submodule 0 for synchronization.

To allocate CPU loading adequately and to avoid two motors consuming energy at the same time, a lag of 180° must be achieved between the PWM waves of the two motors. As shown in Figure 5, the INIT and VAL1 registers are configured reasonably to make the PWM counter run on a regular period. The key to achieve a 180° lag is that whenever the PWM0 counter reaches VAL0, it triggers an EXT_SYNC signal (external synchronization) to PWM1 to initialize the counters of PWM1.

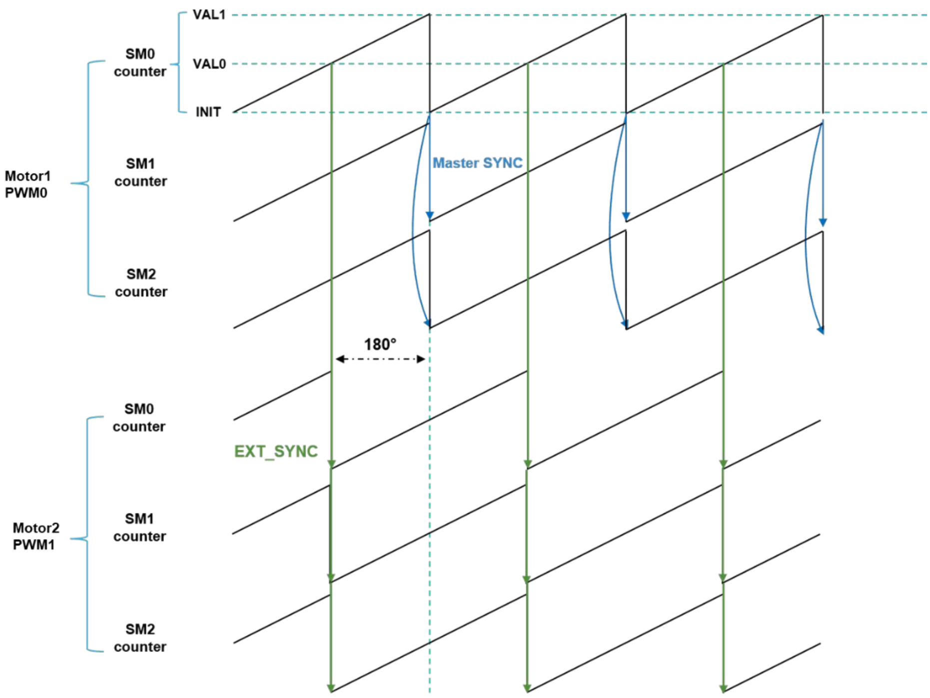

Figure 5. Synchronization between PWMs of dual motor

As mentioned earlier, a 180° phase lag exists between the PWMs of the two motors. Due to this lag, the ADC module uses the triggers from eFlexPWM to implement time division sampling of analog signals of dual motors.

Figure 6 presents the synchronization between ADC and PWMs.

After the eFlexPWM0 counter reaches PWM0SM1VAL4, the ADC is triggered to sample the analog signal of motor 1. After the ADC conversion, the ADC0 conversion completion interrupt gets triggered and enters the `ADC0_IRQHandler` to run the motor 1 control algorithm.

After the eFlexPWM1 counter reaches PWM1SM0VAL4, the ADC is triggered to sample the analog signal of motor 2. After the ADC conversion, the ADC1 conversion completion interrupt gets triggered and enters the `ADC1_IRQHandler` to run the motor 2 control algorithm.

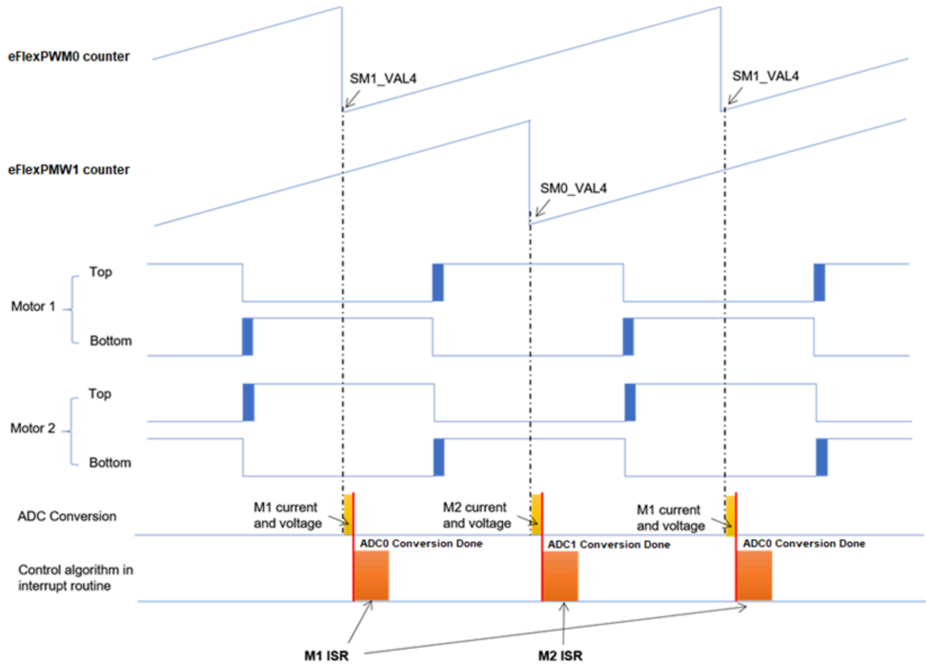

Figure 6. Synchronization between ADC and PWMs

3.4 Standard counter/timers (CTIMER)

CTimer is divided into five submodules: `CTIMER0`, `CTIMER1`, `CTIMER2`, `CTIMER3`, and `CTIMER4`.

Each submodule has a 32-bit counter and programmable frequency divider, which can count with the CTIMER clock or an externally provided clock in a cycle. It can generate interrupts selectively or perform other operations on the specified timer value, according to the contents of the four matching registers.

In this dual servo example, `CTimer0` and `CTimer1` are used for the synchronization of the slow control loops of the two motors. The counter value is set to 48000 and the interrupt is enabled, with the interrupt frequency as 2 kHz.

3.5 Input Multiplexing (INPUTMUX)

The Input Multiplexing (INPUTMUX) module can provide different signal path options for the internal peripherals of the chip. The input signals of the peripherals can be multiplexed to multiple input sources. The input sources can be external pins, interrupts, output signals of other peripherals, or other internal signals.

Figure 7 shows all the signals transmitted between different modules of the current demo through INPUTMUX:

- `PWM0SM1_OUT_TRIG0` is used as the sampling trigger signal of ADC0.
- `PWM1SM0_OUT_TRIG0` is used as the sampling trigger signal of ADC1.
- `PWM0SM0_OUT_TRIG0` is used as the external synchronization signal of PWM1 for phase synchronization control.
- `EXTTRIG_IN3` is used as the A-phase signal of ENC0.
- `EXTTRIG_IN2` is used as the B-phase signal of ENC0.

- `EXTTRIG_IN4` is used as the A-phase signal of ENC1.
- `EXTTRIG_IN1` is used as the B-phase signal of ENC1.
- `HSCMP0_OUT` is used as the PWM0FaultTrigger.
- `HSCMP1_OUT` is used as the PWM1FaultTrigger.

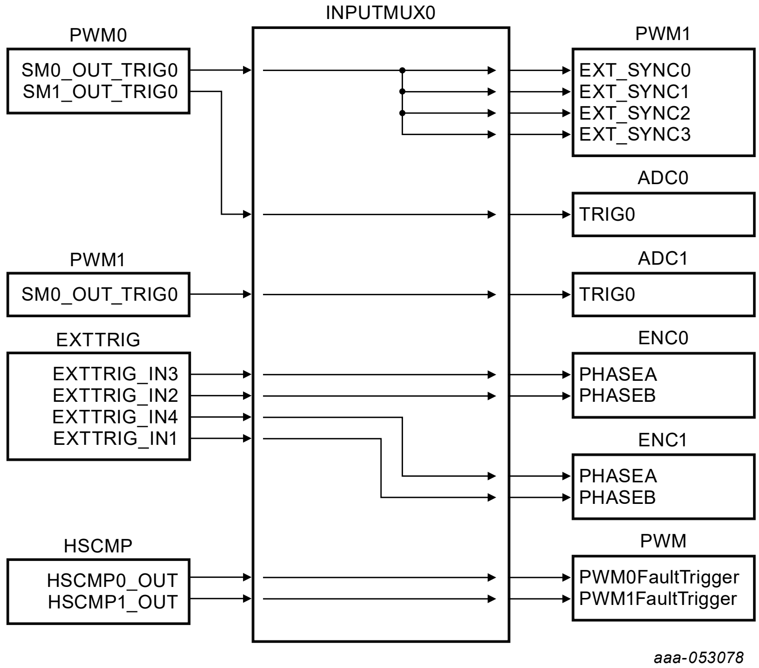

Figure 7. Input multiplexing connections

3.6 Quadrature decoder (ENC)

ENC is divided into two submodules: ENC0 and ENC1. Each submodule has a 32-bit counter/timer group, suitable for decoding the encoder signals. A counter/timer group includes:

- Prescaler
- Filter
- Position counter
- Revolution counter
- Position deviation counter
- Holding register
- Watchdog clock
- Pulse accumulator

In the current demo, the ENC module is used for obtaining motor rotor position information and for performing speed measurement.

Figure 8 shows how to use the ENC module to implement the counting of motor encoder signals.

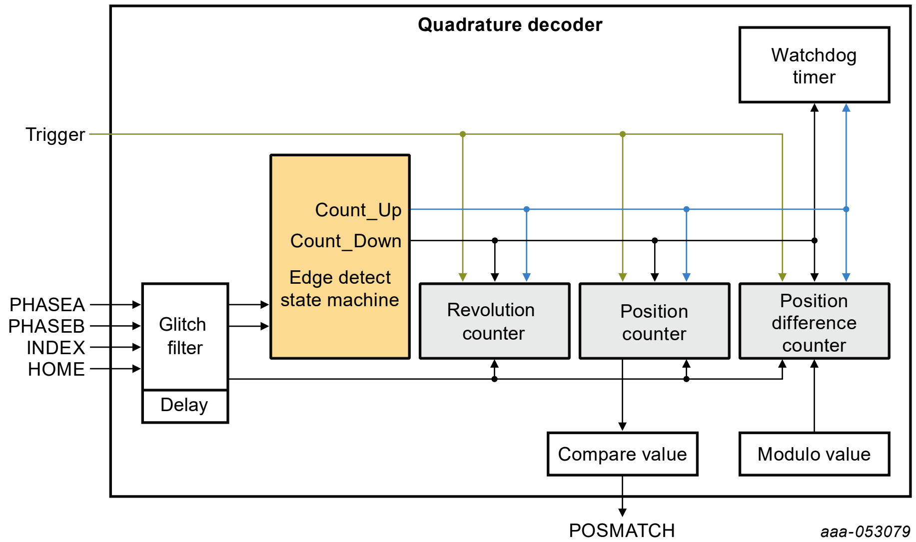

Figure 8. ENC structure diagram

The 32-bit position counter calculates up or down on every count pulse generated by the difference of PHASEA and PHASEB. The position counter acts as an integration information, and its value is proportional to position. The direction of the count is determined through the Count_Up and Count_Down signals.

Figure 9 shows the basic operation of a quadrature incremental position quad decoder:

- If PHASEA leads PHASEB, then the motion is in the positive direction.
- If PHASEA trails PHASEB, then the motion is in the negative direction.

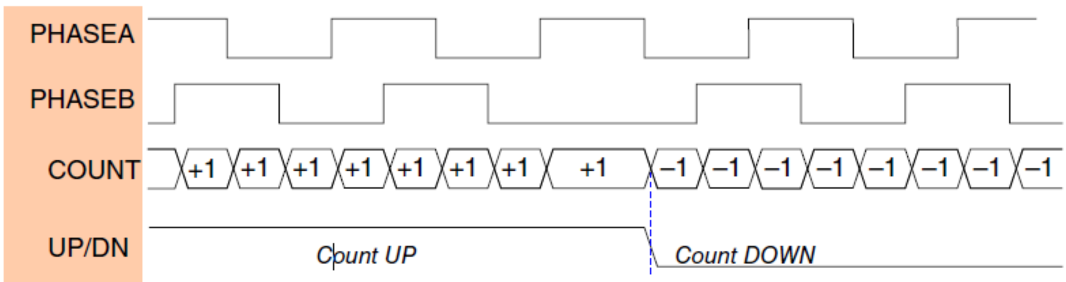

Figure 9. Quad decoder signals

To implement the motor speed measurement function, the ENC module has the following key registers:

- POSD: Reading this register can load the value of each register into the corresponding hold register to implement data synchronization.
- POSDH: This register indicates the position value change that occurs between two reads of the position register.
- POSDPERH: This register uses the peripheral clock prescaled by CTRL3[PRSC] as the reference. It stores the change of the counter value between the two encoder pulses, reflecting the time difference between the two encoder pulses.
- LASTEDGEH: This register indicates the time since the last encoder pulse.

T_{timer} is the clock cycle of the ENC module.

Line is the line number of the motor encoder.

For motor speed measurement, the M method and the T method are suitable for high-speed and low-speed conditions, respectively. Therefore, it is necessary to implement the flexible switching of the two speed measurement methods under different rotational speed conditions. For high-speed scenarios, the speed can be calculated using the M method:

$$Speed\ =\left(\frac{\mathrm{POSDH}}{\mathrm{POSDPERH}*T_{timer}}*\frac{60}{4*Line}\right)\mathrm{RPM}$$

For low-speed scenarios, the speed can be calculated using the T method:

$$Speed\ =\left(\frac{1}{\mathrm{LASTEDGEH}*T_{timer}}*\frac{60}{4*Line}\right)\mathrm{RPM}$$

With the ENC module, automatic switching between the two methods can be easily implemented to achieve accurate speed measurement. For algorithm related details, see the "Quadrature Decoder (ENC)" chapter of *LPC553x Reference Manual*.

3.7 Fault protection (HSCMP and DAC)

The fault protection function is necessary in the operation of a motor as it helps to protect the motor from any damage during overcurrent scenarios. It notifies the motor if an overcurrent fault has occurred so that the motor can turn off the PWM output in time.

To deal with overcurrent faults, LPC553x/LPC55S3x has three High-Speed Comparator (HSCMP) and three DAC modules.

In the current demo, DAC0 and DAC1 are used to provide reference voltages for motor 1 and motor 2, respectively. HSCMP0 and HSCMP1 are used for comparing motor 1 DC bus current and motor 2 DC bus current with DAC0 output and DAC1 output, respectively. HSCMP0/HSCMP1 generates a fault signal when an overcurrent condition arises, as explained below:

- If motor 1 / motor 2 DC bus current (plus input) - DAC0/DAC1 output (minus input) < 0, then the motor current is within the normal range.
- If motor 1 / motor 2 DC bus current (plus input) - DAC0/DAC1 output (minus input) > 0, then the motor current is too high, and overcurrent protection is needed.

Note: *For HSCMP0,* `HSCMP0_IN3` *is plus input and* `DAC0_OUT` *is minus input. For HSCMP1,* `HSCMP1_IN3` *is plus input and* `DAC1_OUT` *is minus input.*

DAC uses the VDDA supply as a voltage reference. To implement fault protection:

1. Enable HSCMP high power / high speed mode.

2. Attach `main_clk` to DAC, and set `DACnCLKDIV[DIV]` to 11.

3. Calculate the clock frequency as follows:

Frequency = 150/(11+1) MHz = 12.5 MHz

3.8 FreeMASTER communication (Flexcomm0)

Flexcomm0 is used for the FreeMASTER communication between the LPC553x/LPC55S3x and the PC:

- Flexcomm0 is configured for UART.
- Both the receiver and transmitter are enabled.
- The baud rate is set to 115200 bit/s.
- Other settings are kept as default.
- In the `freemaster_cfg.h` file, add the serial communication module being used and the base address of the register, such as UART0.

4 Demo operation

This section demonstrates the operation of the dual servo motor.

4.1 Project file structure

The total number of source files (`*.c`) and header files (`*.h`) in the project is large. Therefore, only the key project files are described in detail, and the rest are described in groups.

The main project folder is divided into the following directories:

- `\boards\dual_servo`: Contains the initialization configuration files for the hardware board.
- `\boards\dual_servo\iar`: Contains necessary compiler files.
- `\boards\dual_servo\mc_drivers`: Contains the driver files of each module.
- `\boards\dual_servo\motor_control`: Contains the motor control algorithm files and state machine files.
- `\boards\dual_servo\parameter`: Contains the parameter header files and configuration file.
- `\CMSIS`: Provides details related to Cortex Microcontroller Software Interface Standard (CMSIS).
- `\devices\LPC55S36`: Contains details of LPC553x/LPC55S3x Software Development Kit (SDK).
- `\FM_ControlPage`: Contains FreeMASTER control page files.
- `\middleware\freemaster`: Contains FreeMASTER support files.
- `\middlewareCM33F_RTCESL_4.6.2_IAR`: Contains Real-Time Control Embedded Software Libraries (RTCESL) for motor control and power conversion.

Files in the folders:

- `M1_statemachine.c` and `M1_statemachine.h`: These files contain the software routines executed when the application is in a particular state or state transition.
- `State_machine.c` and `state_machine.h`: These files contain the application state machine structure definition and they manage the switching between the application states and application state transitions.
- `Motor_structure.c` and `motor_structure.h`: These files contain structure definitions and subroutines meant for executing motor control algorithms. The algorithms include vector control algorithm, position and speed estimation algorithm, and speed control loop.
- `Motor_def.h`: Contains the main control and fault structure definition.

4.2 Motor parameters

The motor used in the current example is a brushless DC servo motor. Table 2 provides the motor specifications.

Table 2. Servo motor specifications

Manufacturer name	Just Motion Control	**Stator resistor/Ohm**	0.58
Model	42JSF630AS	**Stator winding inductance d-axis/μH**	308
Rated speed/rpm	3000	**Stator winding inductance q-axis/μH**	330
Rated line voltage/V	24	**Pole pairs**	4
Rated power/W	64	**Line number**	1000

The application parameters (position, speed, and current controller) are set for a motor with a plastic ring (included in the board kit) mounted on the shaft; otherwise, speed oscillations may occur.

4.3 Setting up dual servo motor demo

Figure 10 shows the dual servo motor demo setup.

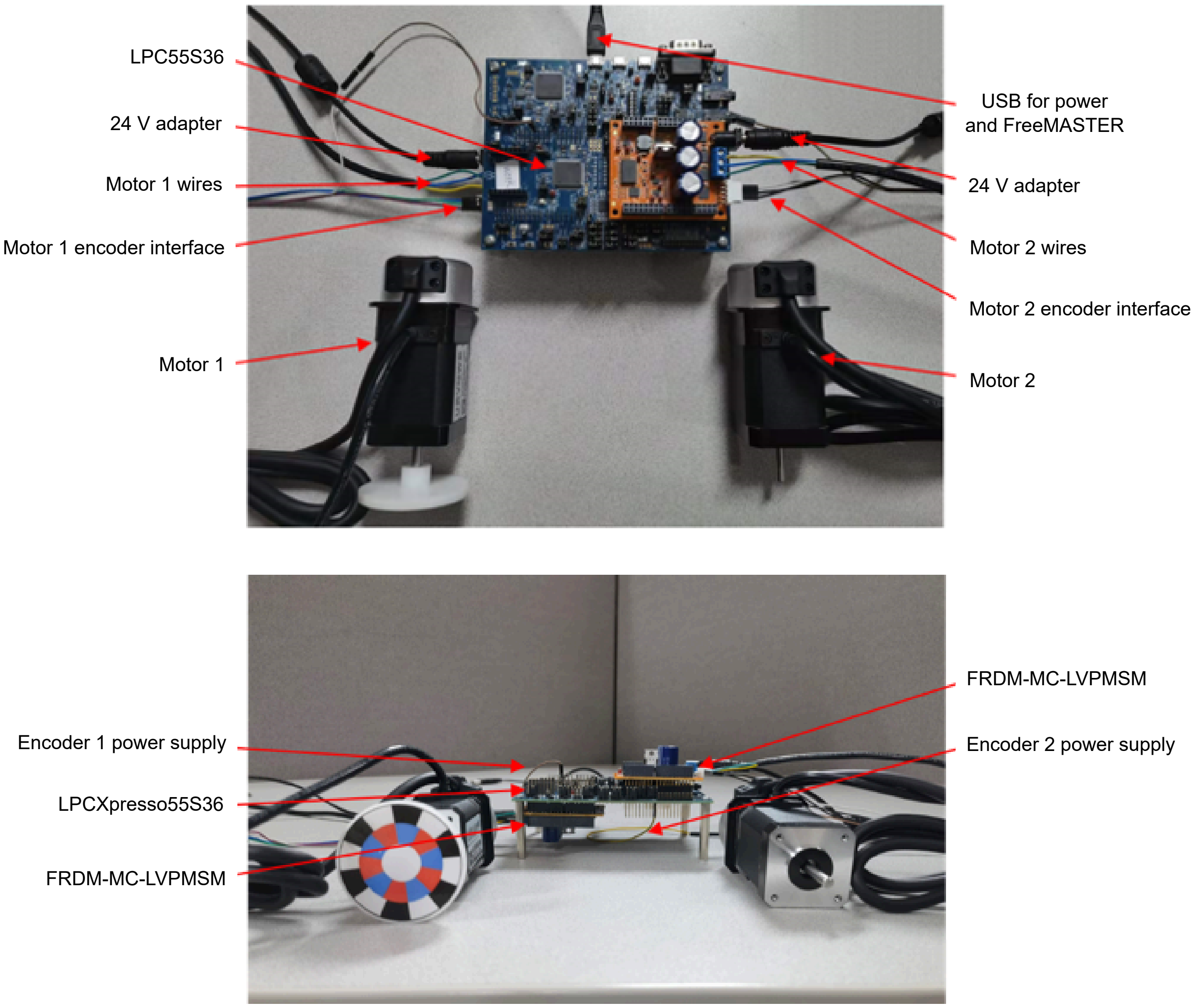

Figure 10. Dual servo motor demo setup

Major hardware components used in the demo are listed below:

- LPCXpresso55S36 board
- Two FRDM-MC-LVPMSM boards
- Two 24 V servo motors
- Micro-USB cable

To set up the dual servo motor demo, perform these steps:

Note: *Using the TPS54060 DC-DC converter on the FRDM-MC-LVPMSM board as a power supply for the LPCXpresso55S36 board causes a voltage glitch. To avoid this issue, open jumper JP71 on the LPCXpresso55S36 board and use the board power supply, SYS_5V0, for powering the motor encoder. Before starting the process, ensure that the adapters are powered off.*

1. As shown in Figure 10, plug the LPCXpresso55S36 and FRDM-MC-LVPMSM boards together via Arduino interface, and connect motor wires and encoder interface.
2. Supply power to the FRDM-MC-LVPMSM board through a 24 V adapter.
3. Connect LPCXpresso55S36 and PC via USB interface.
4. Open `FM_DualServo.pmp` available in the software package (FreeMASTER version must be no lower than 3.1.2).
5. Click the **GO** button to enable communication between PC and LPC553x/LPC55S3x, as shown in Figure 11.
6. Click the **DualServo** page.
7. Click the **Start** button to enable the demo.
8. Operate the demo by clicking other buttons on the control page.

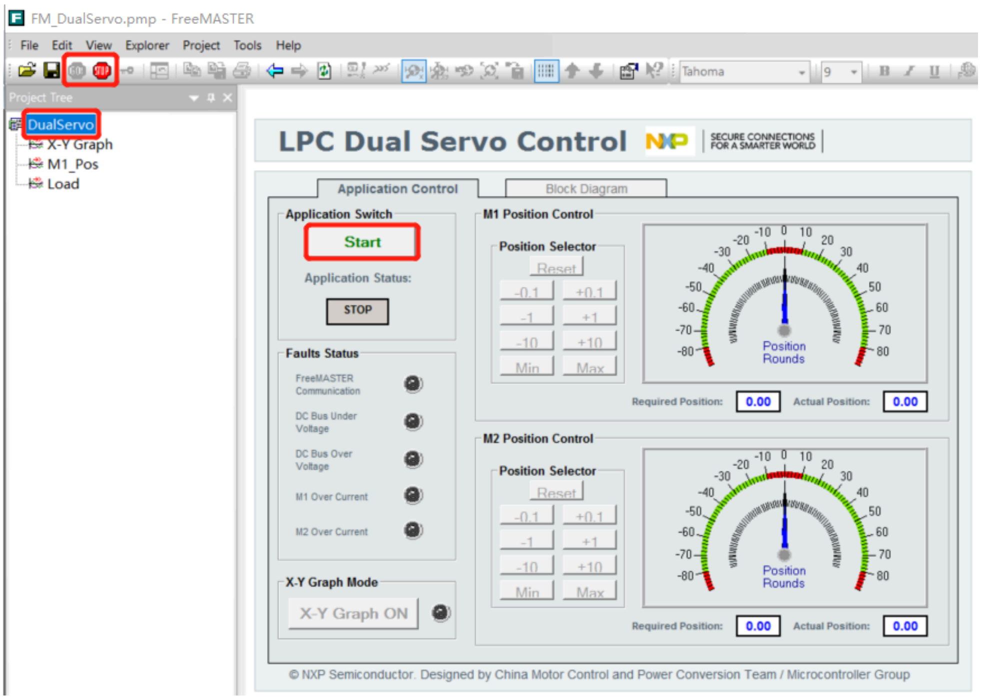

Figure 11. FreeMASTER control page

4.4 Configuring parameters

If the parameters of your servo motor are configured differently than the default parameter configuration for the servo motor used in the current demo, then you must reconfigure your motor parameters accordingly.

To do the reconfiguration, follow the steps below:

1. Open the header file `M1_Params.h` or `M2_Params.h` as needed and add the basic parameters of the motor body at the corresponding position.

```
/* Motor basic parameters */
//-----------------------------------------------------------------------------------------------
#define M1_STATOR_R           0.58         /* [ohm] */
#define M1_LD_INDUCTANCE      0.00032     /* [Henry] */
#define M1_LQ_INDUCTANCE      0.00032     /* [Henry] */
#define M1_POLE_PAIRS         4            /* [pairs] */

/* Based value */
//-----------------------------------------------------------------------------------------------
#define M1_U_DCB_MAX          60.8                        /* [V] */
#define M1_U_FOC_MAX          (M1_U_DCB_MAX/1.732)        /* [V] */
#define M1_U_DCB_OVERVOLTAGE  30                         /* [V] */
#define M1_U_DCB_UNDERVOLTAGE 18                         /* [V] */
#define M1_I_MAX              8.25                        /* [A] */
#define M1_E_MAX              12                          /* [V] */
#define M1_N_MAX              3000                        /* [RPM] */
#define M1_OVERVOLT_LIMIT     FRAC16(1.0*M1_U_DCB_OVERVOLTAGE/M1_U_DCB_MAX)
#define M1_UNDERVOLT_LIMIT    FRAC16(1.0*M1_U_DCB_UNDERVOLTAGE/M1_U_DCB_MAX)
#define M1_DUTY_CYCLE_LIMIT      0.9                /* [ ] */

/* Time scaling */
//-----------------------------------------------------------------------------------------------
#define      M1_MC_PWM_CLK_FREQ       150000000                /* [Hz], PWM clock frequency */
#define      M1_CONTROL_FREQ          16000                    /* [Hz], PWM frequency */
#define      M1_MC_SLOW_CONTROL_LOOP_FREQ        2000          /* [Hz], slow loop control frequency */
#define      M1_SPEED_LOOP_CNTR       (M1_CONTROL_FREQ/M1_MC_SLOW_CONTROL_LOOP_FREQ)
```

Figure 12. Motor parameter file

2. Add the required bandwidth and other parameters in the corresponding current loop, speed loop, position loop, filter, and so on. The specific controller and filter parameters are calculated based on the formula and are assigned to the relevant structure, when you run the program.
   The control parameters of the speed loop and position loop must be provided manually and debugged in the file, as shown in Figure 13.

```
/*Speed  Loop */
#define M1_Spd_Ramp                  20000            /* [RPM/s], mechanical speed accelerating ra
#define M1_Spd_Ctrl_AW_Limit         1                /* [A], current output limitation */
#define M1_SPEED_PI_PROP_GAIN        ACC32(2.5)      /* proportional gain */
#define M1_SPEED_PI_INTEG_GAIN       ACC32(0.12)     /* integral gain */
//------------------------------------------------------------------------------------------
#define M1_RAMP_INC_SPD_CL  FRAC32(1.0F*M1_Spd_Ramp/M1_MC_SLOW_CONTROL_LOOP_FREQ/M1_N_MAX)
#define M1_SPEED_LOOP_UPPER_LIMIT FRAC16(M1_Spd_Ctrl_AW_Limit/M1_I_MAX)
#define M1_SPEED_LOOP_LOWER_LIMIT -FRAC16(M1_Spd_Ctrl_AW_Limit/M1_I_MAX)

/* Positon Loop */
#define M1_Pos_Speed_Limit_Up           1200          /* [RPM/s], position ramping rate for a p
#define M1_Pos_Speed_Limit_Down         1200          /* [RPM/s], position ramping rate for a p
#define M1_Pos_Ctrl_AW_Limit            2000          /* [RPM], mechanical speed - position con
#define M1_Pos_Ctrl_PropGain            0.08          /* proportional gain */
#define M1_POS_CTRL_PROP_GAIN_SHIFT     0             /* proportional gain shift */
//------------------------------------------------------------------------------------------
#define M1_POS_BASE                  180
#define M1_POS_RAMP_UP               FRAC32(M1_Pos_Speed_Limit_Up/60.0/M1_MC_SLOW_CONTROL_LOOP_FREQ)
#define M1_POS_RAMP_DOWN             FRAC32(M1_Pos_Speed_Limit_Down/60.0/M1_MC_SLOW_CONTROL_LOOP_FREQ)
#define M1_POS_CTRL_PROP_GAIN_BASE      100
#define M1_POS_CTRL_PROP_GAIN        FRAC32(M1_Pos_Ctrl_PropGain)
#define M1_POS_CTRL_UPPER_LIMIT FRAC32(1.0*M1_Pos_Ctrl_AW_Limit/M1_N_MAX)
#define M1_POS_CTRL_LOWER_LIMIT FRAC32(-1.0*M1_Pos_Ctrl_AW_Limit/M1_N_MAX)
```

Figure 13. Position and speed loop parameters

While the current loop is equivalent to a second-order control system, the corresponding PI control coefficients can be generated automatically by setting the attenuation and bandwidth frequency, as shown in Figure 14.

```
/*ACR parameter*/
#define M1_CLOOP_ATT              (0.85F)    /* Attenuation */
#define M1_CLOOP_FREQ             (1000.0F)  /* [Hz], Current loop bandwidth frequency */
#define M1_CLOOP_LIMIT           (0.5F)    /* Voltage output limitation, based on real tim
//----------------------------------------------------------------------------------------
/*Damping Coefficient_D&Q_ACR=  0.85    */
/*bandWidth=    1000    [Hz]*/
#define M1_D_KP_GAIN_A32    ACC32((2.0*M1_CLOOP_ATT*2*PI*M1_CLOOP_FREQ*M1_LD_INDUCTANCE -
#define M1_D_KI_GAIN_A32    ACC32(2*PI*M1_CLOOP_FREQ*2*PI*M1_CLOOP_FREQ*M1_LD_INDUCTANCE/
#define M1_D_POS_LIMIT      FRAC16(M1_CLOOP_LIMIT)
#define M1_D_NEG_LIMIT      -FRAC16(M1_CLOOP_LIMIT)

#define M1_Q_KP_GAIN_A32    ACC32((2.0*M1_CLOOP_ATT*2*PI*M1_CLOOP_FREQ*M1_LQ_INDUCTANCE -
#define M1_Q_KI_GAIN_A32    ACC32(2*PI*M1_CLOOP_FREQ*2*PI*M1_CLOOP_FREQ*M1_LQ_INDUCTANCE/
#define M1_Q_POS_LIMIT      FRAC16(M1_CLOOP_LIMIT)
#define M1_Q_NEG_LIMIT      -FRAC16(M1_CLOOP_LIMIT)
```

Figure 14. Current loop parameter calculation

For speed and voltage using the infinite impulse response (IIR) filter, you can manually add the cut-off frequency of the filter for debugging as shown in Figure 15.

```
/* Filters */
#define M1_UDCBUS_FILTER_CUTOFF_FREQ     100     /* [Hz] */
//-------------------------------------------------------------------------------------
// Udc bus IIR, cutoff freq = 100 [Hz]    Ts =    0.0000625   [s]
#define M1_FILTER_UDCBUS_B1         WARP(M1_UDCBUS_FILTER_CUTOFF_FREQ, M1_CONTROL_FREQ)/(
#define M1_FILTER_UDCBUS_B2         WARP(M1_UDCBUS_FILTER_CUTOFF_FREQ, M1_CONTROL_FREQ)/(
#define M1_FILTER_UDCBUS_A2         (M1_FILTER_UDCBUS_B1 + M1_FILTER_UDCBUS_B2 - 1.0F)
```

Figure 15. DC bus voltage filter parameter calculation

The specific controller and filter parameters are calculated based on the formula and are assigned to the relevant structure for execution, when you run the program.

4.5 Demo experiment performance

All the measurement results mentioned in this section are taken when the motor is loaded with a light plastic ring, and all the graphs are generated from the FreeMASTER tool.

Figure 16 shows the speed and current waveforms when the motor startup is at 2500 revolutions per minute (RPM). The red line is speed requirement, the green line is actual speed, and the blue line is torque current. As you can see that the motor can accelerate to 2500 RPM within 0.13 s, and the overshoot is very small.

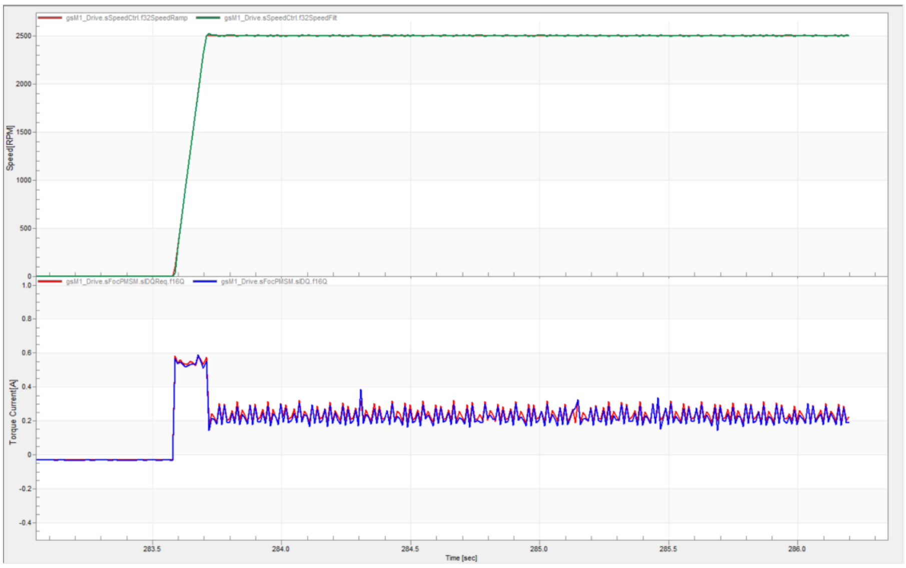

Figure 16. Speed and current response

[Figure 17](#) shows the position response when position requirement is 10 Hz sinusoidal, and the range of motion is 180° mechanical angles. You can see that the rotor position (green line) can track well the change of the given value (red line), and the maximum error (blue line) is about 2°.

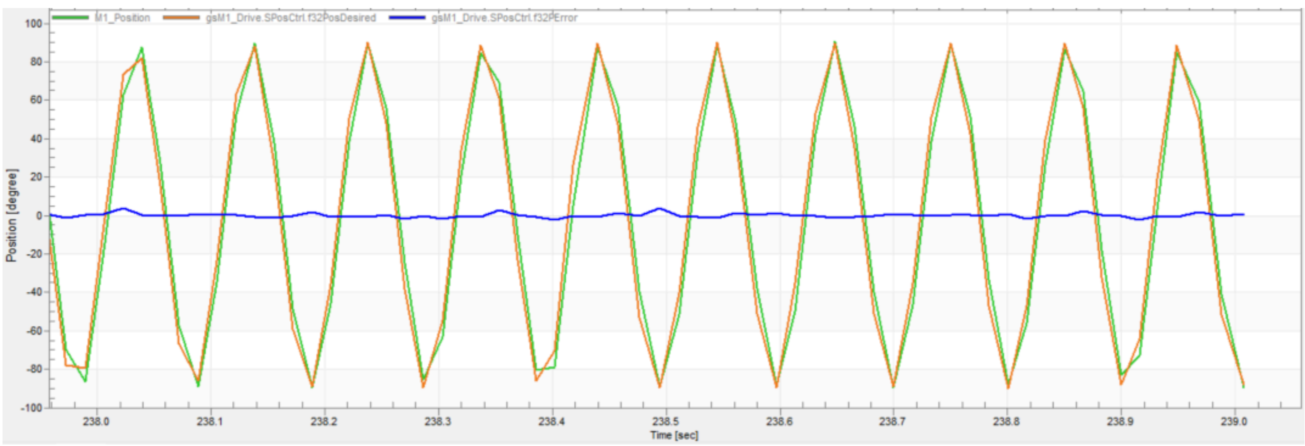

Figure 17. Sinusoidal position response

In [Figure 18](#), the waveforms on the upper side show the speed response, and the waveforms on the lower side show the position response. The red line indicates the requirement, the green line indicates the actual value, and the blue line shows the error between them. After setting the 180° position requirement, the motor takes about 0.1 seconds to reach the desired position. You can notice that the error of the dynamic response is small and the static response is stable.

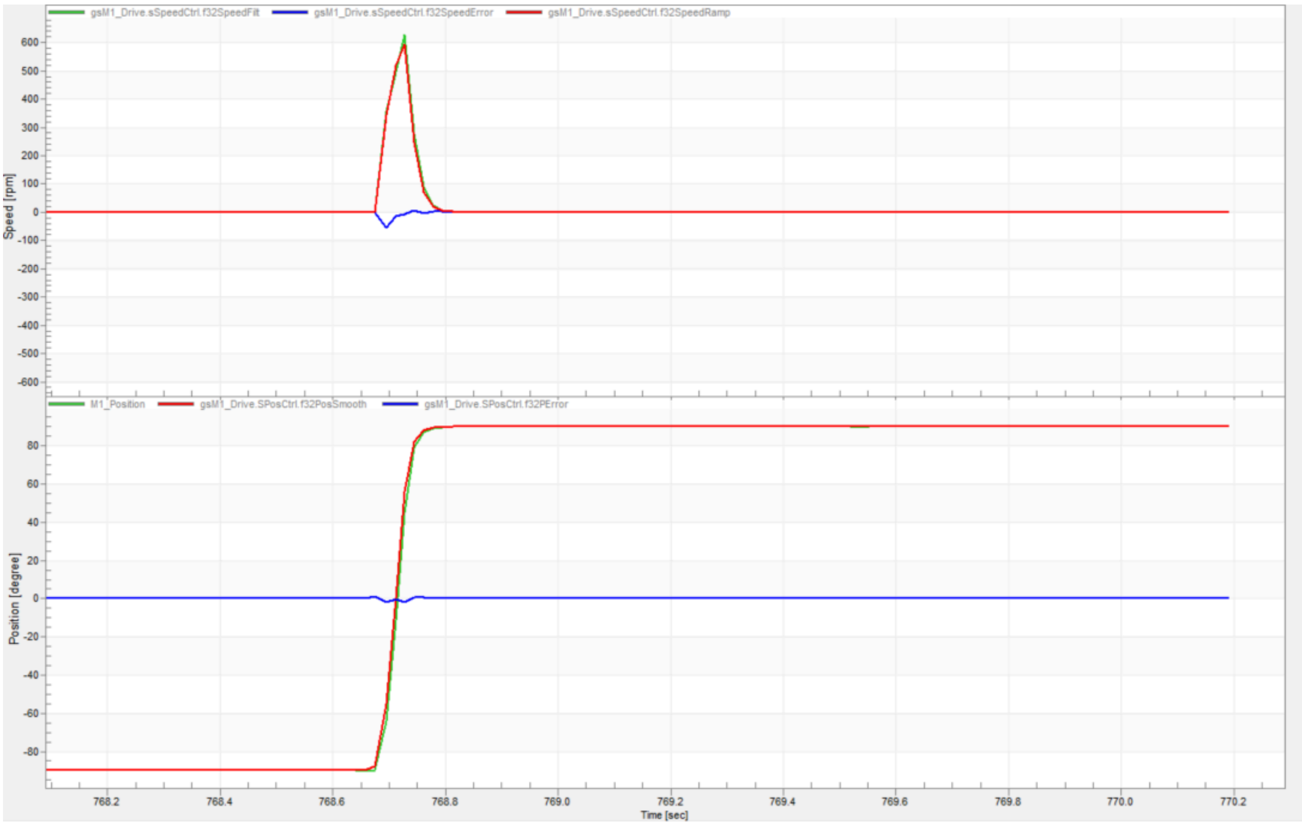

Figure 18. Position and speed response

If you set a variable x that changes periodically, the positions of the two motors are set to sin(x) and cos(x), respectively. The rotor positions of the two motors are used as the abscissa and ordinate, respectively. The ideal trajectory of the coordinate point is a circle. The smoother the edge of the circle is, the more precise the position control is. Click the **X-Y Graph ON** button on the FreeMASTER control page to start the demo. The measurement results are shown in Figure 19.

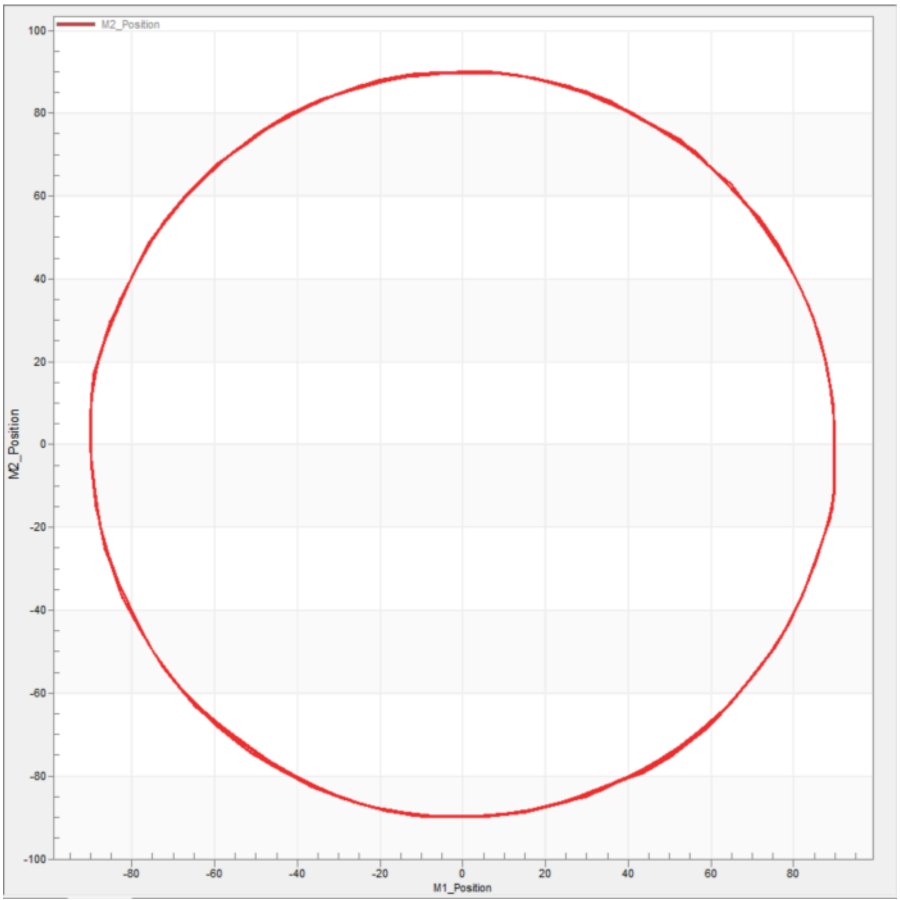

Figure 19. Dual motor position trajectory

4.6 CPU load and memory usage

The information provided in this section is based on the demo application built using IAR Embedded Workbench IDE v9.40.1. The demo application is in the debug RAM and flash configurations, and the optimization level is set to high.

Table 3 shows the memory usage and CPU load. The memory usage is calculated from the linker `.map file` (IAR IDE), including the 4 KB FreeMASTER recorder buffer allocated in RAM. The CPU load is measured using the SysTick timer.

For the current demo, the fast loop frequency is 16 kHz and the slow loop (speed and position loop) frequency is 2 kHz.

Table 3. LPC553x/LPC55S3x dual servo motor demo CPU load and memory usage

—	Fast loop (Flash)	Slow loop (Flash)	Fast loop (RAM)	Slow loop (RAM)
ROM code memory (in bytes)	22996	22996	—	—
RAM code memory (in bytes)	—	—	22996	22996
ROM data memory (in bytes)	3940	3940	—	—

Table 3. LPC553x/LPC55S3x dual servo motor demo CPU load and memory usage*...continued*

—	Fast loop (Flash)	Slow loop (Flash)	Fast loop (RAM)	Slow loop (RAM)
RAM data memory (in bytes)	15510	15510	19450	19450
CPU cycles (single motor)	2323	1056	1149	405
CPU load	52.37 %		25.59 %	

5 References

Table 4 lists additional documents that may be required while working on the dual servo motor demo.

Table 4. Reference documentation

Document	Link
LPC553x Reference Manual (LPC553xRM)	LPC553xRM.pdf
MCUXpresso SDK 3-Phase PMSM Control (LPC) (3PPMSMCLPCUG)	3PPMSMCLPCUG.pdf

6 Acronyms

Table 5 lists the acronyms used in this document.

Table 5. Acronyms

Acronym	Description
ADC	Analog-to-Digital Converter
CMSIS	Cortex Microcontroller Software Interface Standard
CTIMER	Standard counter/timers
DAC	Digital-to-Analog Converter
eFlexPWM	Enhanced Flex Pulse Width Modulator
ENC	Quadrature Decoder
Flexcomm	Flexible serial communication
FOC	Field-oriented control
HSCMP	High-Speed Comparator
I2C	Inter-Integrated Circuit
I2S	Inter-IC Sound
INPUTMUX	Input Multiplexing
PWM	Pulse width modulation
SDK	Software Development Kit
SPI	Serial Peripheral Interface
SVM	Support vector machine
SYSCON	System Configuration
RPM	Revolutions per minute

Table 5. Acronyms...*continued*

Acronym	Description
RTCESL	Real-Time Control Embedded Software Libraries
UART	Universal Asynchronous Receiver/Transmitter

7 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

8 Revision history

Table 6 summarizes the revisions to this document.

Table 6. Revision history

Revision number	Release date	Description
2	15 November 2023	Changed SDK version to 2.14.0.
		Changed IAR Embedded Workbench IDE version to 9.40.1.
1	26 May 2022	Replaced LPC55(S)3x with LPC553x/LPC55S3x
0	23 February 2022	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

Date of release: 15 November 2023
Document identifier: AN13569