1 Introduction

This application note describes the dual servo demo with the NXP LPC55S36 processor. It also can be used as a reference for motor control application developing based on other products.

The LPC55S36 is a processor with a single Arm Cortex-M33 core, which operates at speeds up to 150 MHz. It improves product architecture and integration, improves power consumption, and provides advanced security features which make LPC55S36 ideal for numerous high-performance applications.

This demo includes LPC55S36-EVK, two FRDM-MC-PMSM driver boards, and two three-phase servo motors. The LPC55S36 processor samples the currents and voltages of the motor through the ADC. The ENC module receives the encoder signal to obtain the rotor position and speed, and generates PWM based on the FOC algorithm to drive the motor. At the same time, UART can be used to communicate with the FreeMASTER to achieve command sending, variable observation, and other functions that are convenient for users to debug. Finally, precise position control and smooth speed regulation can be achieved.

Section System structure and software describes the system structure and software of dual servo demo. Section Key peripherals configuration introduces the peripherals configuration of the dual servo demo. Section Demo operation describes how to operate the dual servo 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 demo.
• LPC55S36 evaluation kit (EVK) designed by NXP contains the LPC55S36 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 the servo motors that include 1000 lines encoder.

• eFlexPWM, ENC, ADC are on-chip peripherals, used for motor control, encoder signal acquisition, and analog acquisition respectively.

• 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 LPC55S36 which includes the FOC algorithm, CM33_RTCESL_4.6.2 (Real-Time Control Embedded Software Motor Control and Power Conversion Libraries), and SDK 2.10.

• Flexcomm provides various peripheral function options that can be configured into USART, SPI, I2C, I2S functions through software. Here we configure the USART function to realize the communication between the FreeMASTER debugging tool and LPC55S36 to demonstrate the user's operation.

2.2 Servo control structure

As shown in Figure 2, the control blocks 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 updating.

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.
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 change of disturbance or given value. Its characteristic is that when a disturbance is generated and the controlled variable does not change, it is controlled according to the disturbance to compensate the influence on the controlled variable. In this application, the differential of the position desired is used as 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.

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 a component that controls the magnetic field and a component 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 the one of the separately excited DC motor controller.

To achieve FOC control, perform the following steps:

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

To decompose the stator current into torque components and magnetic flux components, it is necessary to know the position of the excitation magnetic flux of the motor, which requires accurate detection of the position and speed information of the rotor. In this application example, the rotor position and speed information are obtained by collecting the output signal of the motor encoder by the ENC module in LPC55S36. The details of this part are introduced in Section 3.6 Quadrature Decoder (ENC).
3 Key peripherals configuration

This present 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 this design is to generate and control the clock of each submodule and manage its working mode.

There are two clock sources used in this motor control demonstration:

- PLL0, which is input by an external crystal oscillator and obtained after frequency multiplication and division of a phase-locked loop, with a frequency of 150 MHz.

- FRO_HF, which is generated by the on-chip crystal oscillator and is frequency-divided, with a frequency of 96 MHz.

The Arm core works at a frequency of 150 MHz and the clock source is PLL0. For this setting, the following registers have been set in `clock_config.c`: MAINCLKSELB[SEL], AHBCLKDIV[DIV].

The clock source of ENC module is the same as Arm core, and the frequency is 150 MHz.

CTimer modules are clocked by `fro_hf` with a frequency of 96 MHz.

ADC modules are clocked by `fro_hf` and then work at the frequency of 48 MHz (divide by 2).

Flexcomm is clocked by `fro_hf_div` with a frequency of 48 MHz.

DAC modules are clocked by `main_clk` and then 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.

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 divided by 2.
• The ADCs operate as 16-bit with the single-ended conversion and hardware trigger selected.
• ADC Trigger sources get routed through the Input Multiplexer (INPUTMUX). And the trigger signal comes from the eFlexPWM module. The \texttt{TCTRL[HTEN]} register must be set for this function.
• Each ADC module has two independent result FIFOs, and each contains 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.

Since the phase current is measured at the moment when the bottom transistor is turned on, if the duty cycle is very high (the voltage value is in the maximum area of the sine curve) as shown in Figure 4, the time for which the current can be measured is very short. To obtain a stable sampling resistor voltage drop, the bottom transistor must be turned on for at least a critical pulse width. Therefore, we use the dual single-ended sampling mode of the ADC to sample the two-phase currents in parallel at the same time, and the third-phase current is calculated according to the formula. The channel is selected according to the sector that generates the stator voltage space vector, and this assignment is performed at the end of the ADC interrupt service routine. The ADC channel configuration is shown in Table 1.

![Figure 4. The PWM duty and electric degree](image)

Table 1. ADC sample channel configuration

<table>
<thead>
<tr>
<th>Sector number</th>
<th>ADC Channel A</th>
<th>ADC Channel B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sector 1</td>
<td>Phase B</td>
<td>Phase C</td>
</tr>
<tr>
<td>Sector 2</td>
<td>Phase A</td>
<td>Phase C</td>
</tr>
<tr>
<td>Sector 3</td>
<td>Phase A</td>
<td>Phase C</td>
</tr>
<tr>
<td>Sector 4</td>
<td>Phase A</td>
<td>Phase B</td>
</tr>
<tr>
<td>Sector 5</td>
<td>Phase A</td>
<td>Phase B</td>
</tr>
<tr>
<td>Sector 6</td>
<td>Phase B</td>
<td>Phase C</td>
</tr>
</tbody>
</table>

To realize the function of flexible sampling in dual single-ended mode, it is necessary to enable the alternate channel function in the \texttt{CMDLx[ALTBEN]} register, and set the required B channel number in the \texttt{CMDLx[ALTB_ADCH]} register. And when reading the ADC result register, calculate the complete three-phase current value according to the sector number where the current SVM is located and the formula:

\[ i_A + i_B + i_C = 0 \]
The two trigger queues have their own conversion completion interrupt. When the ADC0 conversion is completed, the ADC0 interrupt is triggered and enters `ADC0_IRQHandler` to execute the fast control loop of motor 1. When ADC1 conversion is completed, trigger ADC1 interrupt, enter `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 of which 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 three-phase PWM signals connected to MOSFET H-bridge via pre-drivers.

Each eFlexPWM module has a 16-bit counter that counts up only in the up direction to the VAL1 value and then resets to the INIT value. During the counting process, the count 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 happens to be a multiple of 2, you can set the INIT value and the VAL1 value to be 0-centered opposite numbers, and similarly 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 each other.

The three PWM submodules of motor 1 used in this demo are configured as listed here:

- **PWM0_Submodule_0**
  - IPBus clock source 150 MHz.
  - Running frequency of 16 kHz with 62.5 $\mu$s period.
  - INIT register –4687, VAL1 4686.
  - Complementary mode with 0.5 $\mu$s dead time.
  - PWM reload and initialization signals generated every opportunity from this submodule to other submodules.
  - Trigger one signal from VAL0(0) for providing synchronization with PWM1 of motor 2 via INPUTMUX module.

- **PWM0_Submodule_1**
  - PWM_0 clock source.
  - Running frequency of 16 kHz with 62.5 $\mu$s period.
  - INIT register –4687, VAL1 4686.
  - Complementary mode with 0.5 $\mu$s dead time.
  - PWM reload and initialization signals generated from submodule 0.
  - Trigger one signal from VAL4(-4687) for providing sample synchronization with ADC module via INPUTMUX module.

- **PWM0_Submodule_2**
  - PWM_0 clock source.
  - Running frequency of 16 kHz with 62.5 $\mu$s period.
  - INIT register –4687, VAL1 4686.
  - Complementary mode with 0.5 $\mu$s dead time.
  - PWM reload and initialization signals generated from submodule 0.

The three PWM submodules of motor 2 used in this demo are configured as listed here:

- **PWM1_Submodule_0**
  - IPBus clock source 150 MHz.
  - Running frequency of 16 kHz with 62.5 $\mu$s period.
  - INIT register –4687, VAL1 4686.
  - Complementary mode with 0.5 $\mu$s dead time.
  - EXT_SYNC signal from PWM0 causes initialization.
— PWM reload signals generated every opportunity from this submodule to other submodules.
— Trigger one signal from VAL4(-4687) for providing sample synchronization with ADC module via INPUTMUX module.

• PWM1_Submodule_1 & Submodule_2
  — PWM_0 clock source.
  — Running frequency of 16 kHz with 62.5 μs period.
  — INIT register –4687, VAL1 4686.
  — Complementary mode with 0.5 μs dead time.
  — EXT_SYNC signal from PWM0 causes initialization.
  — PWM reload signals generated from submodule 0.

To allocate CPU loading reasonably and to avoid two motors consuming energy at the same time, it is necessary to achieve a 180 degrees lag between the PWM waves of the two motors. As shown in Figure 6, the INIT and VAL are configured reasonably to make PWM counter run on a regular period. The key to achieve a 180 degrees lag is that whenever the PWM0 counter reaches VAL0, it triggers a signal as EXT_SYNC (external synchronization) to PWM1 to initial the counters of PWM1.

![Figure 5. The synchronization between PWMs of dual motor](image)

As we discussed, there is a 180 degrees phase lag between the PWMs of dual motors, and on this basis 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.

Once eFlexPWM0 counter reaches PWM0SM1VAL4, ADC is triggered to sampling analog signal of motor 1. After the ADC conversion, the ADC0 conversion completion interrupt is triggered and enters the ADC0_IRQHandler to run the motor 1 control algorithm.
Once eFlexPWM1 counter reaches PWM1SM0VAL4, ADC is triggered to sampling analog signal of motor 2. After the ADC conversion, the ADC1 conversion completion interrupt is triggered and enter the ADC1_IRQHandler to run the motor 2 control algorithm.

Figure 6. Synchronization between ADC and PWMS

3.4 Standard counter/timers (CTIMER)

CTimer contains five submodules: CTIMER0, CTIMER1, CTIMER2, CTIMER3, CTIMER4.

Each submodule has a 32-bit counter and programmable frequency divider, which can count with the CTIMER clock or an externally provided clock as the cycle and can selectively generate interrupts 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 count value is set to 48000 and the interrupt is enabled, so the interrupt frequency is 2 kHz.

3.5 Input multiplexing (INPUTMUX)

The input multiplexing module (INPUTMUX) 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, which can be external pins, interrupts, output signals of other peripherals or other internal signal.

In this dual-servo motor control demonstration, Figure 7 shows all the signals transmitted between modules through INPUTMUX.
Figure 7. Input multiplexing connection

- **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.

3.6 Quadrature Decoder (ENC)

ENC contains two submodules ENC0 and ENC1.

They have a 32-bit counter/timer group, which is suitable for the decoding of encoder signals. Each counter/timer group includes prescaler, filter, position counter, revolution counter, position deviation counter, holding register, watchdog clock, pulse accumulator, and so on.

In this application, the ENC module is used for obtaining motor rotor position information and speed measurement.

**Figure 8** shows how to use an ENC module to realize the counting of motor encoder signals.
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, whose count value is proportional to position. The direction of the count is determined by 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 motion is in the positive direction.
- If PHASEA trails PHASEB, then motion is in the negative direction.

In order to realize the motor speed measurement function, there are several key registers in the ENC module: POSD, POSDH, POSDPERH, LASTEDGEH.

Reading the POSD register can load the value of each register into the corresponding hold register to achieve data synchronization;

POSDH is the position value change that occurs between two reads of the position register;

POSDPERH uses the peripheral clock prescaled by CTRL3[PRSC] as the reference, and stores the change of the counter value between two encoder pulses, reflecting the time difference between the two encoder pulses;

LASTEDGEH reflects the time since the last encoder pulse;

\( T_{\text{timer}} \) is the clock cycle of ENC module;

\( Line \) is the line number of motor encoder.
For motor speed measurement, the M method and the T method are suitable for high-speed and low-speed conditions, respectively, so it is necessary to realize the flexible switching of the two speed measurement methods under different rotational speed conditions. For high-speed situations, the speed can be calculated by the M method:

\[
\text{Speed} = \left( \frac{\text{POS DH}}{\text{POS PER PH (in line)}} \times \frac{60}{4 \times \text{timer}} \right) \text{RPM}
\]

For low speed conditions, the speed can be calculated by the T method:

\[
\text{Speed} = \left( \frac{1}{\text{LASTED GEH (in line)}} \times \frac{60}{4 \times \text{timer}} \right) \text{RPM}
\]

With the ENC module, the automatic switching between the two methods can be easily realized to achieve accurate speed measurement. For the specific algorithm details, refer to the ENC chapter of LPC55S3xRM.

### 3.7 Fault protection (HSCMP and DAC)

The fault protection function is required in the operation of the motor, so that the motor can turn off the PWM output in time to protect the equipment when an overcurrent fault occurs. LPC55S36 has three HSCMP and three DAC modules to deal with the overcurrent fault.

In this demo, we use HSCMP0 and HSCMP1 to compare the bus current and then generate fault signals when overcurrent happens, DAC0 and DAC1 to provide our desired reference voltage.

- HSCMP0 input minus is DAC0_OUT, input plus is HSCMP0_IN3.
- HSCMP1 input minus is DAC1_OUT, input plus is HSCMP1_IN3.
- Enable HSCMP high power/high speed mode.
- DAC use VDDA supply as voltage reference.
- Attach main_clk to DAC, and set DACnCLKDIV[DIV] to 11, so the clock frequency is 150/(11+1) = 12.5 MHz.

### 3.8 FreeMASTER communication—Flexcomm0

Flexcomm (Flexcomm Serial Communication) is used for the FreeMASTER communication between the LPC55S36 and the PC.

- Flexcomm0 is configured for UART.
- The baud rate is set to 115200 bit/s.
- The receiver and transmitter are both enabled.
- The other settings are set to default.
- In the freemaster_cfg.h file, add the serial communication module 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 sources (*.c) and header files (*.h) in the project is larger. Therefore, only the key project files are described in detail, and the rest will be described in groups.

The main project folder is divided into seven directories:

- \boards\dual_servo — contains initialization configuration files for hardware board.
- \boards\dual_servo\iar — contains compiler necessary files.
- \boards\dual_servo\mc_drivers — contains driver files of each module
- \boards\dual_servo\motor_control — contains motor control algorithm files and state machine files.
• \boards\dual_servo\parameter — contains parameter header files and configuration file.
• \CMSIS — Cortex Microcontroller Software Interface Standard.
• \devices\LPC55S36 — LPC55S36 Software Development Kit.
• \FM_ControlPage — FreeMASTER control page files.
• \middleware\freemaster— FreeMASTER support files.
• \middleware\CM33_RTCESL_4.6.2_IAR— Real-Time Control Embedded Software Motor Control and Power Conversion Libraries.

Files in the folders:

• M1_statemachine.c and M1_statemachine.h contain the software routines executed when the application is in a particular state or state transition.
• State_machine.c and state_machine.h contain the application state machine structure definition and manage switching between the application states and application state transitions.
• Motor_structure.c and motor_structure.h contain the structure definitions and subroutines dedicated for execution of the motor control algorithm (vector control algorithm, position and speed estimation algorithm, speed control loop).
• Motor_def.h contains the main control and fault structure definition.

4.2 Motor parameters

The motor we used is DC brushless servo motor. Table 2 provides the motor specifications.

Table 2. Servo motor specifications

<table>
<thead>
<tr>
<th>Manufacturer's name</th>
<th>Just Motion Control</th>
<th>Stator resistor /Ohm</th>
<th>Stator winding inductance d-axis/μH</th>
<th>Stator winding inductance q-axis/μH</th>
<th>Pole pairs</th>
<th>Line number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>42JSF630AS</td>
<td>0.58</td>
<td>308</td>
<td>330</td>
<td>4</td>
<td>1000</td>
</tr>
<tr>
<td>Rated speed/rpm</td>
<td>3000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rated line voltage/V</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rated power/W</td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The application parameters (position, speed, and current controller) are set for a motor with a plastic ring (part of the kit) mounted on the shaft, otherwise speed oscillations may occur.
4.3 Setup dual servo demo

Figure 10. Setup dual servo demo

Hardware requirements:
- LPC55S36 EVK board REV.B
- Two FRDM-MC-LVPMSM boards
- Two 24 V servo motors
- Micro USB cable

Follow the steps below to set up the dual servo demo:

NOTE
Using the TPS54060 DC-DC converter on the FRDM-MC-LVPMSM board as a power supply for LPC55S36 EVK causes voltage glitch, so it is recommended to disconnect jumper JP71 on the EVK board and use pin 5V on the EVK board (instead of the pin on the FRDM board) as a power supply of the motor encoder. Before starting the process, make sure that adapters are powered off.

1. According to Figure 10, plug the LPC55S36 EVK and FRDM-MC-LVPMSM board together via Arduino interface, connect motor wires and encoder interface.
2. Power on 24 V adapter to power on the FRDM-MC-LVPMSM board.
3. Connect LPC55S36-EVK and PC via USB interface.
4. Open FM_DualServo.pmp in the software package. (FREEMASTER version must be not lower than 3.1.2)
5. Click the GO! button to enable the communication between PC and LPC55S36, as shown in Figure 11.
6. Click the DualServo page.
7. Click the Start button to enable demo.
8. Operate the demo by clicking other buttons on the control page.
4.4 Parameter configuration

If the parameters of the users' servo motor are different from those of the default motor in this demo, the parameters must be reconfigured for matching the different motor.

To do the reconfiguration, follow the steps below:

1. Open the header file `M1_Params.h` or `M2_Params.h` and input the basic parameters of the motor body to the corresponding position.

```
/* Motor basic parameters */
/**
 * @brief Motor parameters
 */
#define M1_MOTOR_R 0.50 /* [Ohm] */
#define M1_LQ_INDUCTANCE 0.00032 /* [Henry] */
#define M1_LQ_INDUCTANCE 0.00032 /* [Henry] */
#define M1_LQ_PAIRS 4 /* [pairs] */

/* Bandwidth */
/**
 * @brief Motor bandwidth
 */
#define M1_U_DCR_MAX 40.1 /* [V] */
#define M1_U_FOC_MAX /* [V] */
#define M1_U_OVERVOLTAGE 30 /* [V] */
#define M1_U_UNDERVOLTAGE 10 /* [V] */
#define M1_I_MAX 8.25 /* [A] */
#define M1_E_MAX 12 /* [V] */
#define M1_L_MAX 3000 /* [rpm] */
#define M1_OVERVOLTAGE_LIMIT 25 /* [V] */
#define M1_UNDERVOLTAGE_LIMIT 25 /* [V] */
#define M1_CURRENT_CYCLE_LIMIT 6.9 /* [rpm] */

/* Time scaling */
/**
 * @brief Motor time scaling
 */
#define M1_MC_CONTROL_FREQ 16000 /* [Hz], PWM frequency */
#define M1_MC_SLOW_CONTROL_LOOP_FREQ 2000 /* [Hz], slow loop control frequency */
#define M1_SPEED_LOOP_FREQ (M1_CONTROL_FREQ/M1_MC_SLOW_CONTROL_LOOP_FREQ)
```

Figure 12. Motor parameters file

2. Fill in 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 will be calculated by the formula and assigned to the relevant structure when the program is running.

3. The control parameters of the speed loop and position loop must be manually input and debugged in the file as shown in Figure 13. While the current loop is equivalent to a second-order control system, and the corresponding PI control coefficients can be automatically generated by setting the attenuation and bandwidth frequency as shown in Figure 14. For speed and voltage using IIR filter, you can manually input the cut-off frequency of the filter for debugging as shown in Figure 15.
in Figure 15. The specific controller and filter parameters will be calculated by the formula and assigned to the relevant structure for execution when the program is running.

4.5 Demo experiment performance

All the following experimental results are tested when the motor is loaded with a light plastic ring. And all the figures come from the FreeMASTER.
Figure 16. Speed and current response

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

Figure 17. Sinusoidal position response

Figure 17 shows the position response when position requirement is 10 Hz sinusoidal, and the range of motion is 180° mechanical angles. We can see that the rotor position (green line) can well track the change of the given value (red line), and the maximum error (blue line) is about 2°.
As shown in Figure 18, the top waveforms show the speed response and the bottom waveforms show the position response. The red line is the requirement, the green line is the actual value, and the blue line shows the error between them. After setting the 180° position requirement, it takes about 0.1 seconds to reach the desired position. We can see that the error of dynamic response is small and the static response is very stable.
If we set a variable $x$ that changes periodically, the positions of the two motors are set to $\sin(x)$ and $\cos(x)$ respectively, and 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, and 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 experimental results are shown in Figure 19.

### 4.6 CPU load and memory usage

The following information applies to the demo application built using the IAR Embedded Workbench.

IDE v8.50.9 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.

In this case, it applies to the fast loop frequency of 16 kHz and the slow loop (speed and position loop) frequency of 2 kHz.

Table 3. LPC55S36 dual servo demo CPU load and memory usage

<table>
<thead>
<tr>
<th></th>
<th>Fast loop (Flash)</th>
<th>Slow loop (Flash)</th>
<th>Fast loop (RAM)</th>
<th>Slow loop (RAM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM code memory</td>
<td>22996</td>
<td>22996</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>RAM code memory</td>
<td>—</td>
<td>—</td>
<td>22996</td>
<td>22996</td>
</tr>
<tr>
<td>ROM data memory</td>
<td>3940</td>
<td>3940</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Table continues on the next page...
Table 3. LPC55S36 dual servo demo CPU load and memory usage (continued)

<table>
<thead>
<tr>
<th></th>
<th>Fast loop (Flash)</th>
<th>Slow loop (Flash)</th>
<th>Fast loop (RAM)</th>
<th>Slow loop (RAM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM data memory [bytes]</td>
<td>15510</td>
<td>15510</td>
<td>19450</td>
<td>19450</td>
</tr>
<tr>
<td>CPU cycles (single motor)</td>
<td>2323</td>
<td>1056</td>
<td>1149</td>
<td>405</td>
</tr>
<tr>
<td>CPU load</td>
<td>52.37 %</td>
<td></td>
<td></td>
<td>25.59 %</td>
</tr>
</tbody>
</table>

5 References

The following documents may offer further reference.

- LPC553xRM
- MCUXpresso SDK 3-Phase PMSM Control (LPC). (3PPMSMCLPCUG)

6 Revision history

Table 4. Revision history

<table>
<thead>
<tr>
<th>Revision number</th>
<th>Date</th>
<th>Substantive changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14 February 2022</td>
<td>Initial release</td>
</tr>
</tbody>
</table>
Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

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.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. 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, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFAREMOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCMOS, UCODE, Freescale, the Freescale logo, AltiiVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetics, Layerscape, MagniIV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org. M, M Mobileye and other Mobileye trademarks or logos appearing herein are trademarks of Mobileye Vision Technologies Ltd. in the United States, the EU and/or other jurisdictions.

© NXP B.V. 2022. All rights reserved.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 14 February 2022
Document identifier: AN13569