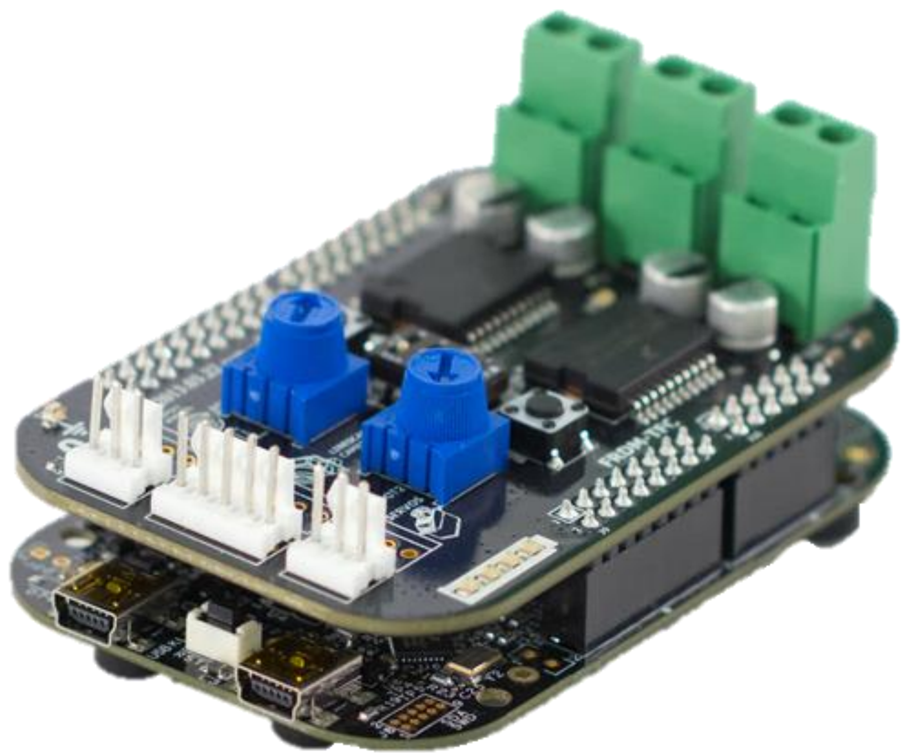
Freescale Semiconductor

Controlling speed in DC motors and position in servomotors with the FRDM-KL25Z and the Kinetis SDK [FTM + GPIO]

By: Technical Information Center

Introduction

This document explains how to control the speed in DC motors and the position in servomotors with the FRDM-KL25Z with the Kinetis SDK. This document is focused on demonstrate the ease of use of the KSDK peripheral drivers applied to control the Freescale Cup smart car.



Freescale Semiconductor

Contents

Freescale Semiconductor

1. About this document

This document is focused in KSDK 1.2.0 with KDS 3.0.0, the document "Create a new KSDK 1.2.0 project in KDS 3.0.0" explains with detail the way to create a new KSDK project.

This document is a continuation of the document "Line scan camera with KSDK [ADC + PIT + GPIO]" where the line scan camera and the FRDM-TFC shield were enabled. Now is time to enable the DC motors and the servo motors which make the car's movement possible.

The application note AN4251 is an auxiliary document to obtain a better understanding of how the DC motors and servo motors work and how can be controlled.

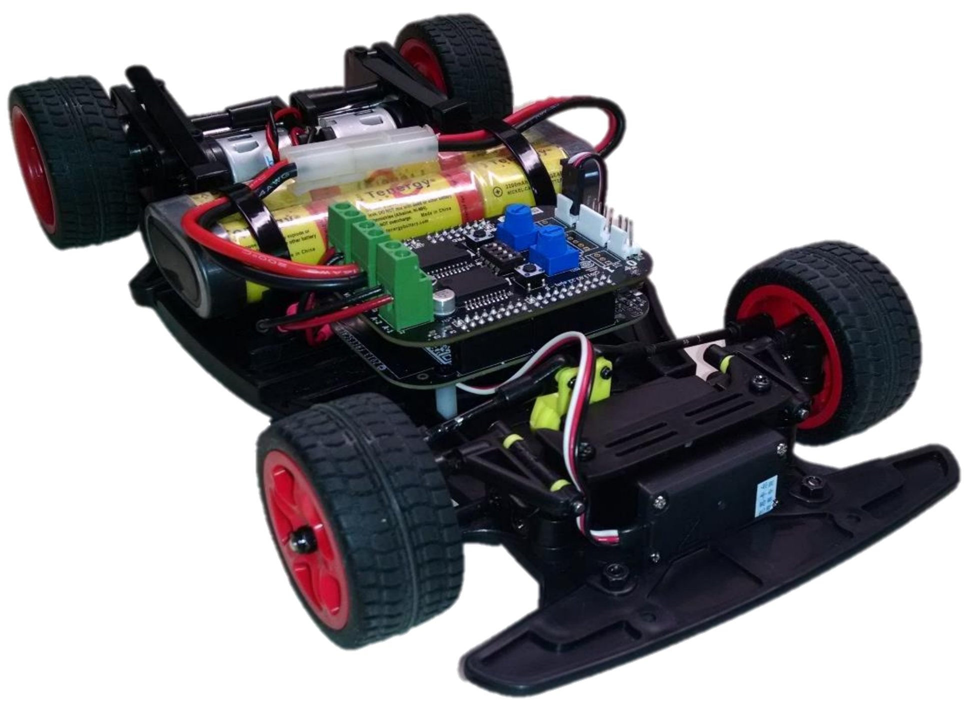


Figure 1. The Freescale Cup smart car used to develop this example.

This application has a structured software that is built as is shown in the image below. The Kinetis SDK peripheral drivers are used to configure the peripherals. This document is focused in controlling the DC motors and the servo motors.

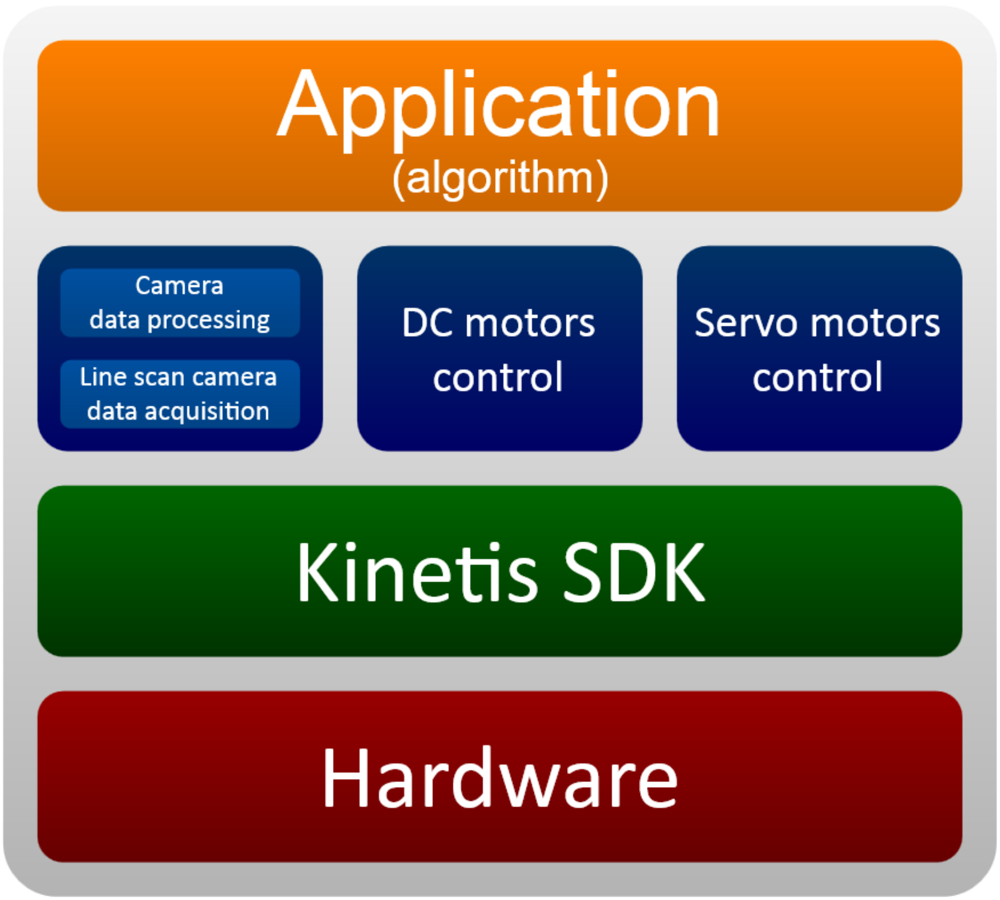


Figure 2. The software architecture that describes this example.

2. Pulse Width Modulation (PWM)

The Pulse Width Modulation (PWM) is a type of digital signal which has a static period but the high pulse duration is changing, i.e. the duty cycle is changing. This is an effective method for adjusting the amount of power delivered to an electrical load.

The term duty cycle describes the portion of ‘on’ time to the regular ‘period’ of time. A low duty cycle corresponds to low power because the power is off for most of the time. This is expressed with a percentage.

The PWM is used for getting analog results with digital means because as the duty cycle is changing the average voltage is changing. This is very useful to control the DC motors speed in an efficient way. Also, it is used to control the direction of a servo motor.

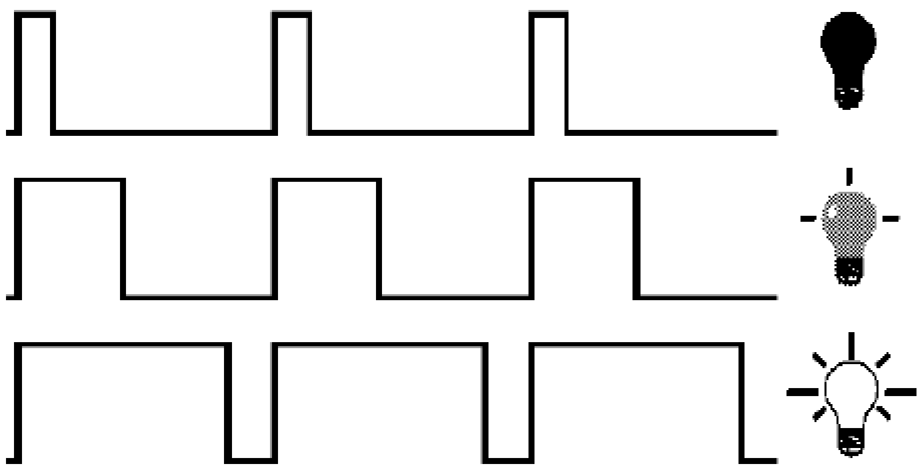


Figure 3. The duty cycle of a PWM signal impacts in the energy transferred to the load.

3. DC motors

A DC motor is an electrical device that converts energy into rotational movement. The motor can rotate in any direction, this depends on the polarity of the source power. The speed in a DC motor can be modified through PWM as is shown in the figure below.

The voltage and the current that must be delivered to the motor to work is too much for a microcontroller output port so an intermediary device must be used. This intermediary is an H-Bridge MC33887 for each of the two DC motors which are contained in the FRDM-TFC shield.

The chosen frequency for the DC motors included in the Freescale smart car is 5000 KHz.

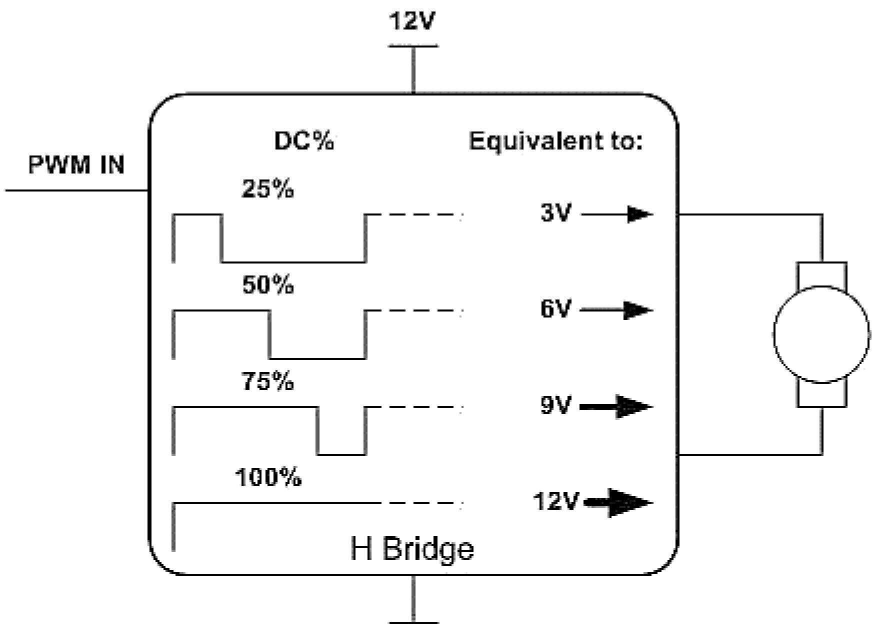


Figure 4. Example of how the PWM impacts on the DC motor speed.

4. Servo motors

The servo motors are specialized DC motors that controls the steering of the smart car. The Futaba S3010 is included in the Freescale Cup kit. The servo motors require a control line to specify the position of the axis. This control is made with PWM.

The chosen frequency for the DC motors included in the Freescale smart car is 50 Hz.

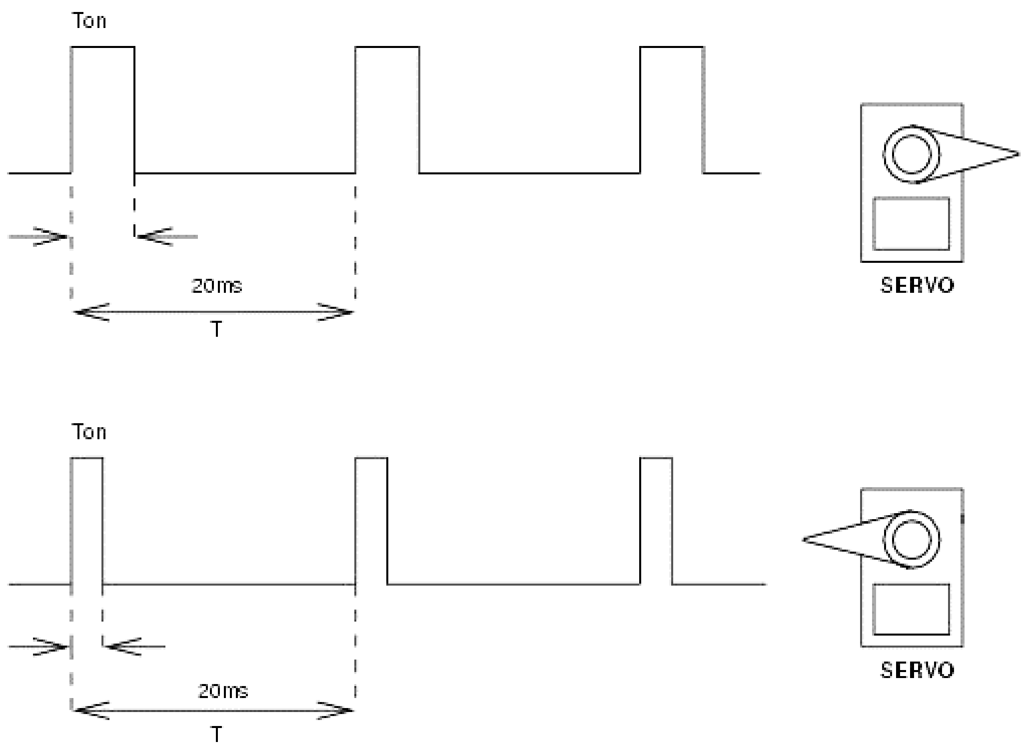


Figure 5. The servo's position depends on the Duty Cycle.

5. Modify the board.c file

The board.c file contains structures with the configurations about the system such the clock. The configurations about the TPM channels must be added in this file. The FRDM-TFC shield schematics show where the motors are routed. The schematics below show the analog channels that will be used to enable the motors in the FRDM-TFC shield.

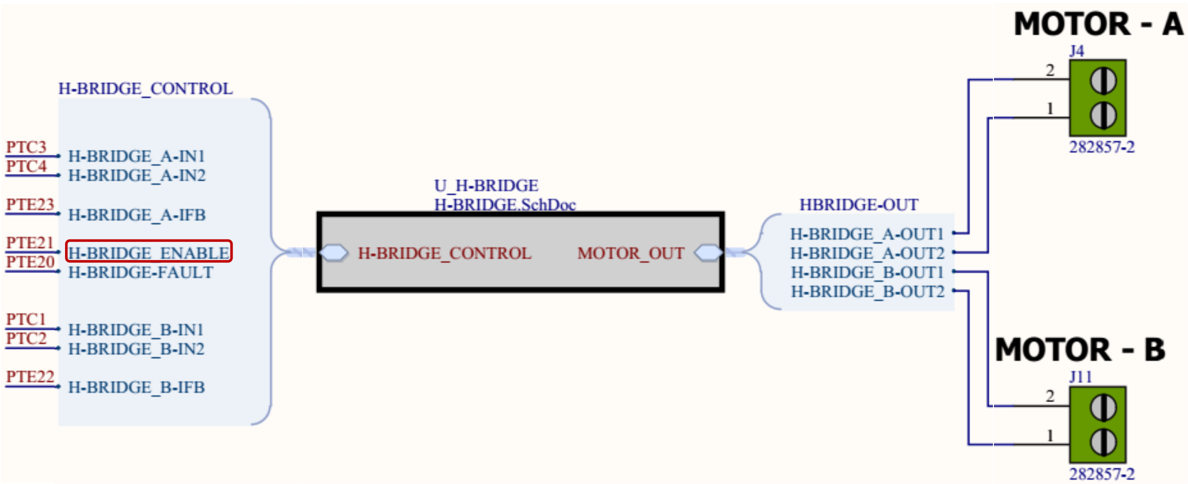


Figure 6. The H-Bridge schematic on the FRDM-TFC shield.

A TPM channel is configured through the structure `tpm_pwm_param_t`, the required configuration fields are:

- mode. The PWM operation mode, this can be center aligned or edge aligned.
- edgeMode. This can be high true or low true. When the edge mode is low true the pulse is negated.
- uFrequencyHZ. The frequency of the PWM signal in Hertz.
- uDutyCyclePercent. The duty cycle for the signal when the channel is started.

The channel configurations are shown below.

```
/* Configuration of TPM channels for FRDM-TFC motors. */
tpm_pwm_param_t TFC_Motors_pwm_param[] =
{
            /* kTFCMotorsServoSpare */
            {
                        .mode = kTpmEdgeAlignedPWM,
                        .edgeMode = kTpmHighTrue,
                        .uFrequencyHZ = TFC_MOTORS_SERVO_FREQUENCY,
```

```
                    .uDutyCyclePercent = (TFC_MOTORS_SERVO_MAXIMUM_DUTY_CYCLE
+ TFC_MOTORS_SERVO_MINIMUM_DUTY_CYCLE) / 2,
          },
          /* kTFCMotorsServoSteering */
          {
                    .mode = kTpmEdgeAlignedPWM,
                    .edgeMode = kTpmHighTrue,
                    .uFrequencyHZ = TFC_MOTORS_SERVO_FREQUENCY,
                    .uDutyCyclePercent = (TFC_MOTORS_SERVO_MAXIMUM_DUTY_CYCLE
+ TFC_MOTORS_SERVO_MINIMUM_DUTY_CYCLE) / 2,
          },
          /* kTFCMotorsDCB1*/
          {
                    .mode = kTpmEdgeAlignedPWM,
                    .edgeMode = kTpmHighTrue,
                    .uFrequencyHZ = TFC_MOTORS_DC_FREQUENCY,
                    .uDutyCyclePercent = 0
          },
          /* kTFCMotorsDCB2*/
          {
                    .mode = kTpmEdgeAlignedPWM,
                    .edgeMode = kTpmHighTrue,
                    .uFrequencyHZ = TFC_MOTORS_DC_FREQUENCY,
                    .uDutyCyclePercent = 0
          },
          /* kTFCMotorsDCA1*/
          {
                    .mode = kTpmEdgeAlignedPWM,
                    .edgeMode = kTpmHighTrue,
                    .uFrequencyHZ = TFC_MOTORS_DC_FREQUENCY,
                    .uDutyCyclePercent = 0
          },
          /* kTFCMotorsDCA2*/
          {
                    .mode = kTpmEdgeAlignedPWM,
                    .edgeMode = kTpmHighTrue,
                    .uFrequencyHZ = TFC_MOTORS_DC_FREQUENCY,
                    .uDutyCyclePercent = 0
          }
};
```

6. Modify the board.h file

The board.h file contains macros used for the GPIOs (General-Purpose Input/Output) which let the application read or write the pins though a simple one-line call. Each H-Bridge contained in the FRDM-TFC shield contains an enable signal to start or stop the motors, this is controlled trough a GPIO pin.

```
#define TFC_MOTORS_ENABLE_EN (GPIO_DRV_OutputPinInit(&ledPins[9]))
             /*!< Enable target TFC_MOTORS_ENABLE */

#define TFC_MOTORS_ENABLE_DIS (PORT_HAL_SetMuxMode(PORTE,  21, kPortMuxAsGpio))
      /*!< Disable target TFC_MOTORS_ENABLE */

#define TFC_MOTORS_ENABLE_OFF (GPIO_DRV_WritePinOutput(ledPins[9].pinName, 0))
      /*!< Turn off target TFC_MOTORS_ENABLE */

#define TFC_MOTORS_ENABLE_ON (GPIO_DRV_WritePinOutput(ledPins[9].pinName, 1))
      /*!< Turn on target TFC_MOTORS_ENABLE */

#define TFC_MOTORS_ENABLE_TOGGLE (GPIO_DRV_TogglePinOutput(ledPins[9].pinName)) /*!<
Toggle on target TFC_MOTORS_ENABLE */
```

7. Modify the gpio_pins.c file

The gpio_pins.c file contains two structures, one for the GPIO input pins and one for the output pins. The output structure `gpio_output_pin_user_config_t` have the following configuration fields:

- pinName (pin name). This value is later explained and generated in the file gpio_pins.h.
- config (the GPIO output pin configuration `gpio_output_pin_t`). This structure defines the pin specific hardware configurations and is later explained.

The `gpio_output_pin_t` structure have the following fields:

- outputLogic (default output value). This is the default value after the GPIO initialization.
- slewRate (slew rate select). Selects between the slow and the fast slew rate.
- driveStrength (drive strength select). Selects between the low and high drive strength.

This structure needs to be modified to add the enable signal for the DC motors. The added field is shown below.

```
/* Declare Output GPIO pins */
gpio_output_pin_user_config_t ledPins[] =
{
                          .
                          .
                          .
             /* DC motors enable signal. */
             {
                          .pinName = kGpioTFC_MotorsEnable,
                          .config.outputLogic = 0,
                          .config.slewRate = kPortSlowSlewRate,
                          .config.driveStrength = kPortLowDriveStrength,
             },
             /* End of signals.*/
             {
                          .pinName = GPIO_PINS_OUT_OF_RANGE,
             }
};
```

8. Modify the gpio_pins.h file

The pinName field in the GPIO configurations structures in the file gpio_pins.c are generated in the file gpio_pins.h. This is an enumeration which contains the information about the port and the pin number and it is generated through the macro GPIO_MAKE_PIN. The added field is shown to the enumeration below.

```
enum _gpio_pins
{
      kGpioTFC_MotorsEnable = GPIO_MAKE_PIN(GPIOE_IDX, 21),/*FRDM-TFC motors enable*/
};
```

9. Modify the pin_mux.c file

The pin_mux.c file contains functions to configure the signal multiplexing for the used pins. For this project, a new GPIO has been added. It will be used to enable the DC motors. This function must be called in the GPIO initialization. The new code in this file is shown below.

```
void configure_gpio_pins(uint32_t instance)
{
	switch(instance) {
	case 4:                             /* PTE */
		/* PORTE_PCR21 FRDM-TFC DC_MOTORS_ENABLE */
		PORT_HAL_SetMuxMode(PORTE,21u,kPortMuxAsGpio);
		break;
	default:
		break;
	}
}
```

10. Modify the pin_mux.h file

The pin_mux.h file contains the prototypes of the functions defined in the file pin_mux.c. For this project no other function is required so this file was not modified.

11. Initializing the TPM instances

Since the DC motors and servo motors work at different frequencies, it is needed to use two TPM instances. The information about the frequencies is stored in the array `TFC_Motors_pwm_param`. This initialization is called in the FRDM-TFC initialization function.

```
void TFC_Motors_Init()
{
	const tpm_general_config_t TFC_Motors_TPM_Config = {0};

	/* Configure the signal multiplexing. */
	configure_tpm_pins(TFC_MOTORS_DC_INSTANCE);
	configure_tpm_pins(TFC_MOTORS_SERVO_INSTANCE);

	/* Initialize the TPM instances. */
	TPM_DRV_Init(TFC_MOTORS_DC_INSTANCE, &TFC_Motors_TPM_Config);
	TPM_DRV_Init(TFC_MOTORS_SERVO_INSTANCE, &TFC_Motors_TPM_Config);

	/* Set the TPM clock. Both instances work with the same clock. */
	TPM_DRV_SetClock(TFC_MOTORS_SERVO_INSTANCE, kTpmClockSourceModuleHighFreq,
kTpmDividedBy16);

	/* Servo motors PWM. */
	TPM_DRV_PwmStart(TFC_MOTORS_SERVO_INSTANCE,
&TFC_Motors_pwm_param[kTFCMotorsServoSpare], TFC_MOTORS_SERVO_SPARE_CHANNEL);
	TPM_DRV_PwmStart(TFC_MOTORS_SERVO_INSTANCE,
&TFC_Motors_pwm_param[kTFCMotorsServoSteering], TFC_MOTORS_SERVO_STEERING_CHANNEL);

	/* DC motors PWM. */
	TPM_DRV_PwmStart(TFC_MOTORS_DC_INSTANCE,
&TFC_Motors_pwm_param[kTFCMotorsDCA1], TFC_MOTORS_DC_A1_CHANNEL);
	TPM_DRV_PwmStart(TFC_MOTORS_DC_INSTANCE,
&TFC_Motors_pwm_param[kTFCMotorsDCA2], TFC_MOTORS_DC_A2_CHANNEL);
	TPM_DRV_PwmStart(TFC_MOTORS_DC_INSTANCE,
&TFC_Motors_pwm_param[kTFCMotorsDCB1], TFC_MOTORS_DC_B1_CHANNEL);
	TPM_DRV_PwmStart(TFC_MOTORS_DC_INSTANCE,
&TFC_Motors_pwm_param[kTFCMotorsDCB2], TFC_MOTORS_DC_B2_CHANNEL);
}
```

12. Setting a new speed for the car

As mentioned before, the car's speed depends on the duty cycle applied to the DC motors. Once the TPM is initialized as PWM it is time to change only the duty cycle for each channel.

Each motor has two PWM signals, this allows to move the motors in both ways (clockwise and counterclockwise). There is a function for each motor to set the speed from -100 to 100, a negative speed means that the car will be running in a reverse way. Those functions are shown below.

```
void TFC_Motors_SetSpeedA(int8_t speed)
{
	TFC_Motors_SetSpeed(speed, TFC_MOTORS_DC_A1_CHANNEL,
TFC_MOTORS_DC_A2_CHANNEL);
}
void TFC_Motors_SetSpeedB(int8_t speed)
{
	TFC_Motors_SetSpeed(speed, TFC_MOTORS_DC_B1_CHANNEL,
TFC_MOTORS_DC_B2_CHANNEL);
}
```

Both functions use the static function shown below.

```
static void TFC_Motors_SetSpeed(int8_t speed, uint8_t channelMotor1, uint8_t
channelMotor2)
{
	uint32_t freq;
	uint16_t uMod;
	uint16_t uCnV1;
	uint16_t uCnV2;

	/* Get the TPM frequency. */
	freq = TPM_DRV_GetClock(TFC_MOTORS_DC_INSTANCE);

	/* Verify if the speed is between -100 and 100. If the speed is negative means
that the motor will run in the reverse way. */
	if(-100 > speed)
	{
		speed = 100;
	}
	else if(100 < speed)
	{
		speed = 100;
	}

	/* Set the speed since the speed sign. */
	/* For PWM edge aligned, calculate the module value. */
	uMod = freq / TFC_MOTORS_DC_FREQUENCY - 1;
```

```
        /* For PWM edge aligned, calculate the match value. */
        if(0 < speed)
        {
                uCnV1 = uMod * speed / 100;
                uCnV2 = 0;
        }
        else
        {
                uCnV1 = 0;
                uCnV2 = uMod * (-speed) / 100;
        }

        /* For 100% duty cycle */
        if(uCnV1 >= uMod)
        {
                uCnV1 = uMod + 1;
        }

        /* For 100% duty cycle */
        if(uCnV2 >= uMod)
        {
                uCnV2 = uMod + 1;
        }

        /* Set the new speeds. */
        TPM_HAL_SetChnCountVal(g_tpmBase[TFC_MOTORS_DC_INSTANCE], channelMotor1,
uCnV1);
        TPM_HAL_SetChnCountVal(g_tpmBase[TFC_MOTORS_DC_INSTANCE], channelMotor2,
uCnV2);
}
```

This function calculates and set the match value for the two channels involved in a DC motor to run as the application asks for.

13. Setting a new steer for the car

In comparison with the DC motors, the duty cycle used for control the servo motor used for the steering needs to be in a bounded range. This depends on the hardware because there is a mechanical limitation as is shown below.

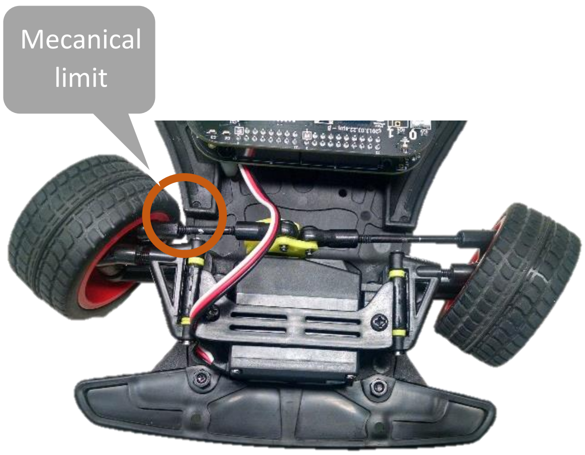


Figure 7. The mechanical limit when turning to the right.

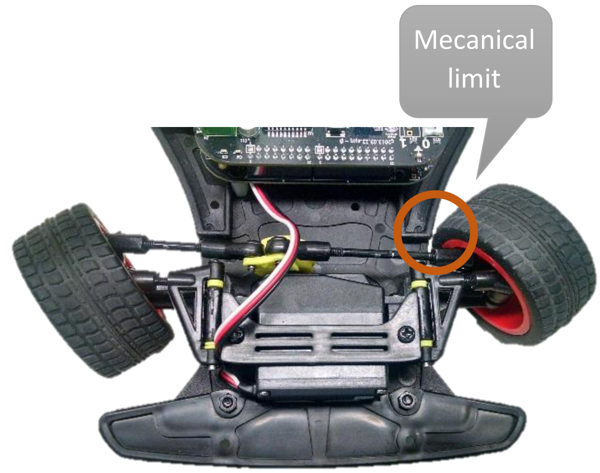


Figure 8. The mechanical limit when turning to the left.

Since it depends on the hardware, two macros are defined to calibrate the minimum and the maximum duty cycle for the servo PWM.

```
/*Minimum duty cycle for the servo. It depends on each car so it must be calibrated*/
#define TFC_MOTORS_SERVO_MINIMUM_DUTY_CYCLE   (5.5)

/*Maximum duty cycle for the servo. It depends on each car so it must be calibrated*/
#define TFC_MOTORS_SERVO_MAXIMUM_DUTY_CYCLE   (8.5)
```

Once the TPM is initialized as PWM it is time to change only the duty cycle for each channel. The functions shown below are used to define a servo position from in a range from -100 to 100.

```
void TFC_Motors_SetPositionSteering(int8_t position)
{
	TFC_Motors_SetPosition(position, TFC_MOTORS_SERVO_STEERING_CHANNEL);
}

void TFC_Motors_SetPositionSpare(int8_t position)
{
	TFC_Motors_SetPosition(position, TFC_MOTORS_SERVO_SPARE_CHANNEL);
}
```

Both functions use the static function shown below.

```
static void TFC_Motors_SetPosition(int8_t position, uint8_t channelMotor)
{
	uint32_t freq;
	uint16_t uMod;
	uint16_t uCnV;
	uint16_t minCnV;
	uint16_t maxCnV;

	/* Get the TPM frequency. */
	freq = TPM_DRV_GetClock(TFC_MOTORS_SERVO_INSTANCE);

	/* Verify if the position is between -100 and 100. If the position is negative 
means that the motor be charged to the left. */
	if(TFC_MOTORS_SERVO_POSITION_MIN > position)
	{
		position = TFC_MOTORS_SERVO_POSITION_MIN;
	}
	else if(TFC_MOTORS_SERVO_POSITION_MAX < position)
	{
		position = TFC_MOTORS_SERVO_POSITION_MAX;
	}

	/* Set the position since the position sign. */
	/* For PWM edge aligned, calculate the module value. */
	uMod = freq / TFC_MOTORS_SERVO_FREQUENCY - 1;

	/* Calculate the minimum and the maximum CnV values. */
	minCnV = uMod * TFC_MOTORS_SERVO_MINIMUM_DUTY_CYCLE / 100;
	maxCnV = uMod * TFC_MOTORS_SERVO_MAXIMUM_DUTY_CYCLE / 100;

	/* For PWM edge aligned, calculate the match value. */
	uCnV = (maxCnV - minCnV) * (position + (TFC_MOTORS_SERVO_POSITION_MAX - 
TFC_MOTORS_SERVO_POSITION_MIN) / 2) / (TFC_MOTORS_SERVO_POSITION_MAX - 
TFC_MOTORS_SERVO_POSITION_MIN) + minCnV;

	/* Set the new position. */
	TPM_HAL_SetChnCountVal(g_tpmBase[TFC_MOTORS_SERVO_INSTANCE], channelMotor, 
uCnV);
    }
```

This function calculates and set the match value for the channel involved in a servo motor to set a steer as the application asks for.

14. Results

The application built demonstrates the ease of use of the new APIs for the car's movement. This application uses the potentiometers from the FRDM-TFC shield. The POT1 controls the DC motors while the POT2 controls the steering.

```
/* FRDM-TFC initialization. */
TFC_Shield_Init();

/* Wait until the SW button is pressed. This will prevent that the motors start when
the board is initially powered on. */
while(!TFC_SHIELD_SW1_READ)
{
	/* Blink the four TFC-SHIELD LEDs to show that the smart car is waiting for a
push in the SW1. */
	static uint16_t counter = 0;
	if(!--counter)
	{
		TFC_SHIELD_LED1_TOGGLE;
		TFC_SHIELD_LED2_TOGGLE;
		TFC_SHIELD_LED3_TOGGLE;
		TFC_SHIELD_LED4_TOGGLE;
	}
}

/* Enable the H-bridge for the motors. */
TFC_MOTORS_ENABLE_ON;

for (;;)
{
	/* Update the speed and the steering. THe POT1 controls the CD motors and the
POT2 the servomotors. */
	TFC_Motors_SetSpeedA(TFC_Shield_ADC_ReadValues[kTFCChnPot1] / 327 - 100);
	TFC_Motors_SetSpeedB(TFC_Shield_ADC_ReadValues[kTFCChnPot1] / 327 - 100);
	TFC_Motors_SetPositionSteering(TFC_Shield_ADC_ReadValues[kTFCChnPot2]/327-100);
	TFC_Motors_SetPositionSpare(TFC_Shield_ADC_ReadValues[kTFCChnPot2]/327-100);

	/* Enable and disable the motors. */
	if(TFC_SHIELD_SW1_READ)
	{
		/* Enable the H-bridge for the motors. */
		TFC_MOTORS_ENABLE_ON;
	}

	if(TFC_SHIELD_SW2_READ)
	{
		/* Disable the H-bridge for the motors. */
		TFC_MOTORS_ENABLE_OFF;
	}
```

```
    /* Dummy code to use the GPIOs. */
    if(TFC_SHIELD_DIP1_READ)
    {
           TFC_SHIELD_LED1_ON;
    }
    else
    {
           TFC_SHIELD_LED1_OFF;
    }

    if(TFC_SHIELD_DIP2_READ)
    {
           TFC_SHIELD_LED2_ON;
    }
    else
    {
           TFC_SHIELD_LED2_OFF;
    }

    if(TFC_SHIELD_DIP3_READ)
    {
           TFC_SHIELD_LED3_ON;
    }
    else
    {
           TFC_SHIELD_LED3_OFF;
    }

    if(TFC_SHIELD_DIP4_READ)
    {
           TFC_SHIELD_LED4_ON;
    }
    else
    {
           TFC_SHIELD_LED4_OFF;
    }
}
```

15. Conclusions

This document has demonstrated the ease of use of the FlexTimer peripheral with the Kinetis SDK, a few lines of code have been enough to enable the motors provided in the Freescale Cup Kit. This project is a good starting point to create an application with the Freescale Cup smart car.