Freescale Semiconductor
Application Note

Document Number:AN4976
Rev. 0, 07/2014

Controlling a DC motor using the PWT module on Kinetis E/EA

by: Vicente Gomez Salazar and
Augusto Panecatl Salas

1 Overview

The Pulse Width Timer (PWT) is a versatile timing module present in the KE04/KE06 and KEA8/128 microcontrollers. It works as a 16-bit timer with a variable duty cycle used as a pulse width measurement unit.

The intention of this document is to explain in high detail how to implement a DC motor drive using the PWT module as a flexible pulse width measurement unit in a close loop to maintain the motor speed when a workload is applied. This document includes:

- Annotated code example
- Hardware Schematics
- Oscilloscope captures

The hardware design has been implemented using a TRK-KEA evaluation board featuring a KEAZ128 microcontroller.

This application note assumes the developer is already acquainted with the PWT and FTM modules and fully understands the basics of how the modules work and are configured.

Contents

1 Overview 1
2 DC motor control using Pulse Width Modulation 2
3 Motor speed control using a closed loop system 3
4 Motor speed control using a Kinetis E/EA microcontroller 4
4.1 Software implementation 4
4.2 Hardware implementation 8
5 Running the demo code 11
5.1 Speed control menu 12
5.2 Signal captures 13
6 Conclusion 14
7 References 14
8 Revision history 14

© 2014 Freescale Semiconductor, Inc.

2 DC motor control using Pulse Width Modulation

Pulse Width Modulation (PWM) is an effective method to control the speed on a DC motor, it maintains a constant voltage level delivered to the motor while delivering a variable amount of energy, directly proportional to the geometrical power average below the signal curve which in turns depends on the duty cycle (percentage of the signal's period used to deliver energy) of the modulated signal, ideally ranging from 0 to 100%.

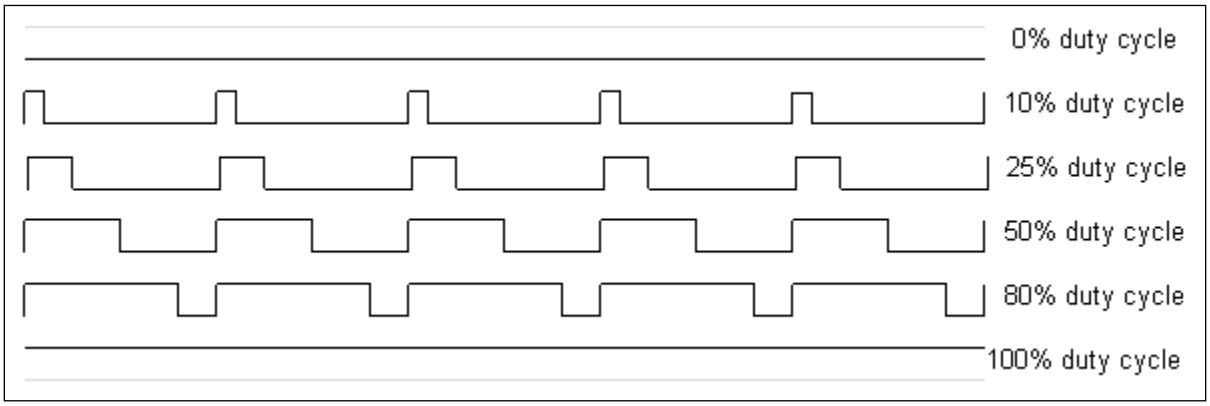

Figure 1 Variable duty cycle signal

This control method is far easier to implement through means of digital circuitry than using analog circuitry which in turn would mean varying the voltage delivered to the motor and compromising its ability to start spinning. PWM motor speed control is achieved through means of injecting a PWM signal into a power stage driver; the PWM signal being generated through a timer or any other mean, in the specific case of this document, a microcontroller with an internal PWM generating module (TPM). The longer its duty cycle, the greater the amount of energy delivered to the motor through the power stage or driver.

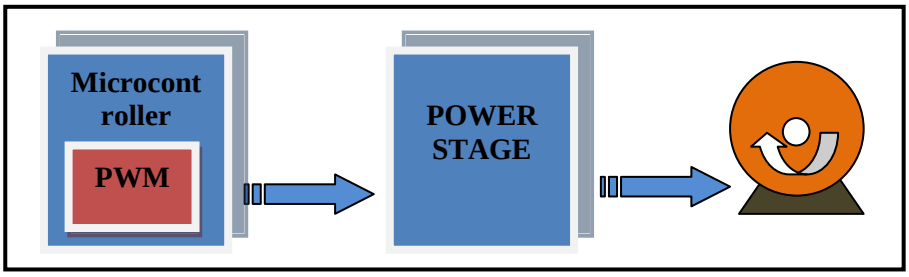

Figure 2 Open loop PWM motor speed control

The effect of the control is physically observed as a variation in the speed attained by the rotor. Thus a variation in its available torque derived from the amount of power delivered by the power stage controlled by the PWM signal coming from the microcontroller's PWM module.

3 Motor speed control using a closed loop system

This method allows us to achieve a significant degree of control over the motor's speed, however lacks a way through which we are able to monitor its real time behavior in terms of speed or torque, henceforth rendering the system incapable of correcting any variation. Were we in the need of having such type of feedback we would have to add two new components to the existing system, a closed loop in the mode of a sensor capable of acquiring a signal proportional to the rotor's speed and a module capable of comparing the sensor's acquired signal to the PWM control signal.

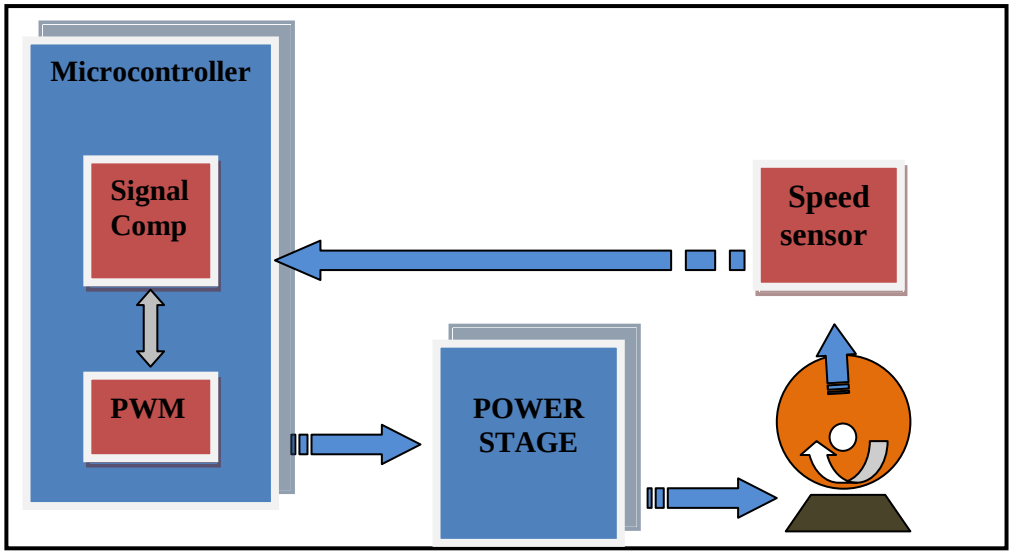

Figure 3 Closed loop PWM motor speed control

There are several types of speed sensor to implement a closed loop motor speed control, ranging from complex and expensive quadrature encoders to simple optical sensors. Once a sensor has been selected and properly included into the feedback loop, it will acquire a signal proportional to the rotor's speed and deliver it to the signal comparator module which in turn will be in charge of comparing the signals period to the PWM control signal's period. The use of having a feedback signal from the motor will allow us to correct such variations in the rotor's speed, obtaining a much more stable system.

By means of comparing the period of both signals, the signal comparator module allows us to know whether the rotor speed is lower, greater or equal and in which magnitude to our PWM control signal; having this knowledge we can then apply corrective means to maintain a constant speed by either increasing or decreasing the PWM control signal's duty cycle and by consequence delivering a greater or lower quantity of energy to the motor through the power stage.

In this document, the feedback channel frequency containing the rotor speed is fed to PWT, which measures the signal period and duty cycle directly.

4 Motor speed control using a Kinetis E/EA microcontroller

As already mentioned before, a basic closed loop motor speed control consists of the following components.

- PWM signal generator
- Power stage / driver
- Motor
- Speed sensor [feedback signal]
- Feedback signal comparator

In the specific case of a Kinetis E/EA device, the microcontroller's equivalent to the PWM signal generator is the Flex Timer (FTM) and the equivalent of the feedback signal comparator is the PWT, which by means of internal connections can be fed the same reference clock signal for an easier and more accurate measurement.

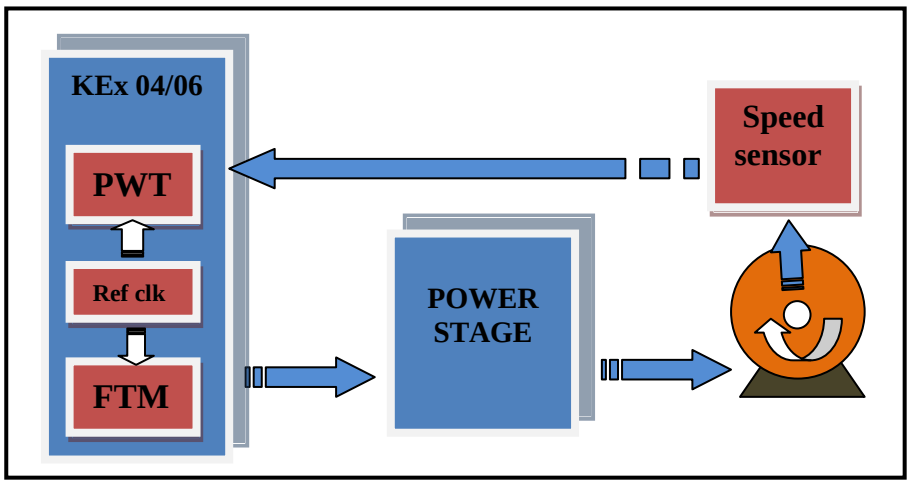

Figure 4 Closed loop PWM motor speed control using the KEx04/06 PWT module

4.1 Software implementation

4.1.1 FTM configuration

This section details the FTM module configuration to provide a PWM signal; channel selection, clock source, frequency, interrupt configuration, interrupt frequency, and PWM signal's period.

The easiest way to clock the FTM module is to use the default clock source, which in this case is the Bus clock (core clock/2), since the core clock is working in FEI at 40 MHz, the FTM clock source runs at 20 MHz; using a prescaler value of four we provide the module with a 5 MHz reference clock:

FTM_Init()

```
SIM_SCGC |= SIM_SCGC_FTM2_MASK;   /* Enable Clock for FTM2 */
FTM2_SC |= FTM_SC_PS(2);          /* Prescaler: 4. 20 Mhz /4 =5 Mhz. */
FTM2_MOD =5000;                   /* PWM frequency= 1 KHz (5 Mhz/5000)*/
```

Now we select the output channel (FTM2_CH0), enable the FTM interrupt and define the PWM frequency as well as the interrupt period :

```
FTM2_C0SC |= FTM_CnSC_MSB_MASK;  /* Channel as Edge aligned PWM */
FTM2_C0SC |= FTM_CnSC_ELSB_MASK; /* High-true pulses */


SIM_PINSEL1_FTM2PS0(0x00);       /*Enable Channel 0 at PTC0*/
FTM2_SC |= FTM_SC_TOIE_MASK;     /*FTM Interrupt Enable */
FTM2_C0V = FTM_CnV_VAL(2500) ;   /*Channel 0 interrupt frequency set to 50%*/
```

4.1.2 PWT configuration

This section details the PWT module configuration to measure the incoming signal from the optical sensor containing the rotor speed information.

As with the FTM module, the PWT clock source is the Bus clock running at 20 MHz. To achieve a higher degree of accuracy, we select a prescaler value of 64 to obtain a frequency of 312.5 KHz.

PWT_Init()

```
SIM_SCGC |= SIM_SCGC_PWT_MASK; /* Enable Clock for PWT module */
PWT_R1 = PWT_R1_PWTSR_MASK;    /* Soft module reset */
PWT_R1 |= PWT_R1_PRE(6);       /* Set Prescaler: 64. 20 Mhz/64= 312.5KHz.*/
```

By default when enabling the PWT module, input channel 0 is set to read an incoming signal, so we simply keep the default configuration to enable PWT input channel 0 (PTD5/PWT_IN0). Now it is necessary to configure the capture mode (falling edges) and enable the module interrupt and the Data Ready interrupt:

```
/*Set the input pin*/
/* Default PWT input is PWTIN0*/
/* PWT clock set, PWTIN0 enable*/

PWT_R1 |= PWT_R1_EDGE(3);     /*falling edge and subsequent falling edges*/
PWT_R1 |= PWT_R1_PRDYIE_MASK; /* PWT Pulse Width Data Ready Int Enable*/
PWT_R1 |= PWT_R1_PWTIE_MASK;  /* PWT Module Interrupt Enable*/
```

4.1.3 Main routine

As an optional part of the code, a speed selection menu was implemented to be displayed through a serial terminal:

```
int main(void)
{
      UINT32 counter = 0;

       Clk_Init();                          /* Configure clocks to run at 20 Mhz */
       UART_Init();                         /*Initialize Uart 2 at 9600 bauds */
       put("\r***PWT demo code***\r\n");
       put("Press one option (a,b,c)\r\n");
       put("a)Motor at 3500 rpm\r\n");
       put("b)Motor at 2500 rpm\r\n");
       put("c)Motor at 1500 rpm\r\n");
       FTM_Init();                          /* Initialize the FTM module */
       Enable_Interrupt(INT_FTM2);          /* Enable FTM2 interrupt */
       PWT_Init();
       Enable_Interrupt(INT_PWT);           /* Enable PWT interrupt */

       GPIOA_PDDR |= (1 <<19);              /* PTC 3 as output*/
       GPIOA_PTOR |= (1 << 19);             /*toggle output ptc3 */

      while(!menu)
      {
            ch = Uart_GetChar();
            switch (ch)
                  {
                        case ('a'):
                              rpm_target = 3500;
                              FTM2_SC |= FTM_SC_CLKS(1);  /*FTM2 use system clock*/
                              PWT_R1 |= PWT_R1_PWTEN_MASK; /* Enable PWT module*/
                              menu = 1;
                              break;

                        case ('b'):
                              rpm_target = 2500;
                              FTM2_SC |= FTM_SC_CLKS(1); /*FTM2 use system clock*/
                              PWT_R1 |= PWT_R1_PWTEN_MASK; /* Enable PWT module*/
                              menu = 1;
                              break;

                        case ('c'):
                              rpm_target = 1500;
                              FTM2_SC |= FTM_SC_CLKS(1); /*FTM2 use system clock*/
                              PWT_R1 |= PWT_R1_PWTEN_MASK; /* Enable PWT module*/
                              menu = 1;
                              break;
                        default:
                              put("Not valid Key\r\n");
                              put("Press one option (a,b,c)\r\n");
                              //break;
                  }
            }
```

```
	for (;;)
	{
		printf("\r Motor motor rpm set (%i), real rpm (%i)", rpm_target,
read_rpm);
		some_delay(30000);
	}

	return 0;

}
```

4.1.4 Speed monitoring using the PWT

Once the motor speed has been selected in the terminal and established through the *rpm_target* function, the program inquiries the motor speed by reading the **PWT** measurement coming from the sensor. If the speed matches the target speed selected, no adjust is made, if the speed is lower or greater than the target speed the PWM signal's duty cycle is proportionally adjusted, closing the system loop. The **PWT** is constantly monitoring the motor speed capturing the signal at every falling edge, once the signal is captured, its period is measured, when the measurement is finished the **PWTRDY** flag is raised and an interrupt executed to let the system know the data register can be accessed and contains valid information.

```
void PWT_IRQHandler()
{
	 //Clear interrupt flag
	 pwt_t = (PWT_R2&0x0000FFFF);
	 read_rpm =((312500*60)/pwt_t);                    // 60 to convert to rpm

	if (read_rpm > 10000)
	{
		GPIOA_PTOR |= (1 << 19);                  /*toggle output ptc3 */
	}
	 (void)PWT_R1;
	 PWT_R1 ^= PWT_R1_PWTRDY_MASK;                    /* Clear flag */
}
```

4.1.5 Speed adjustment

The speed adjustment is performed taking into consideration the speed measured by the **PWT** through the *read_rpm* function. When there is a difference between two speeds, the PWM signal provided by the FTM is modified accordingly, either increasing or decreasing the signal duty cycle until both speed figures match, the adjustment is performed every 50 FTM overflow cycles to allow the motor speed a stabilization period; as the speed measurement this was implemented using an interruption, in this case triggered every 50 FTM overflow cycles.

```
void FTM2_IRQHandler()
{
       (void)FTM2_SC;
       FTM2_SC ^= FTM_SC_TOF_MASK;                    /* Clear flag */
       counts--;
         if (!counts)
         {
               if (rpm_target > read_rpm)
               {
                     if (FTM2_C0V <= 4950)
                     FTM2_C0V += 10;                  // increase PWM duty cycle
               }
               else if (rpm_target < read_rpm)
               {
                     if (FTM2_C0V >= 50)
                     FTM2_C0V -= 10;                  // decrease PWM duty cycle
               }
               counts = OVF_COUNTS;
         }

}
```

4.2 Hardware implementation

Having defined the microcontroller's FTM module channel 0 (**PTC0/FTM2_CH0**) will be used to deliver the PWM signal to control the rotor speed through the power stage, the PWT module channel 0 (**PTD5/PWT_IN0**) to read and measure the incoming signal from the optical sensor containing the rotor speed information and having a common internal reference clock for both modules, we can define the rest of the hardware needed to implement the system.

The following table shows the hardware components needed to implement the system.

Table 1 System hardware components

Development board	TRK-KEA (KEAZ128AMLK)
Motor	1Ph 9V DC motor (generic)
Sensor	OMRON EE-SX1137 Photomicrosensor
Power stage	TI L293D H bridge
Power supply	9V DC generic

The following figure shows the TRK-KEA evaluation board.

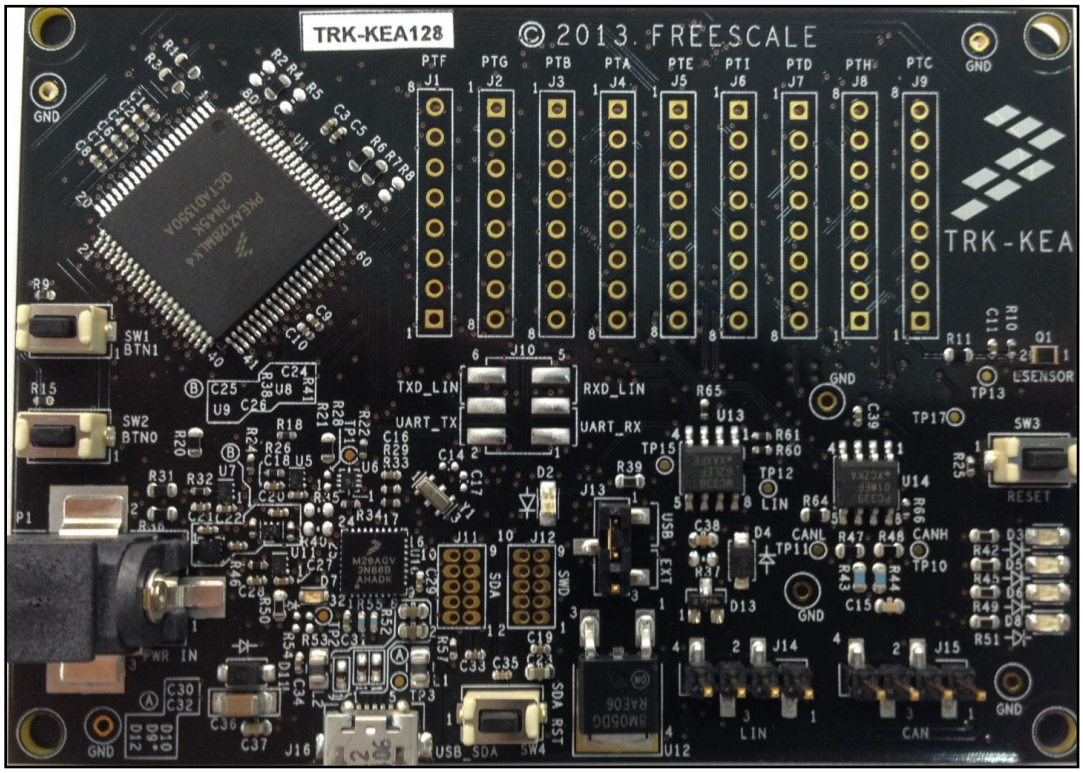

Figure 5 TRK-KEA evaluation board

The following figure shows the motor and speed measurement assembly.

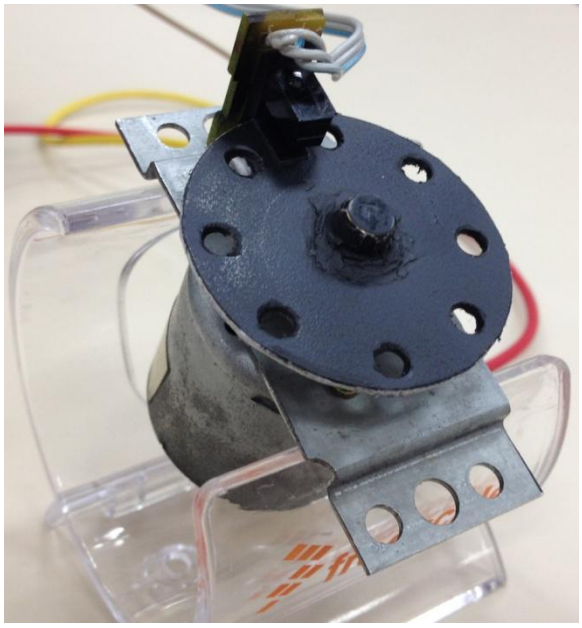

Figure 6 1Ph 9V DC motor and EE-SX1137 optical sensor

The following figure shows the TIL293D and EE-SX1137 circuitry.

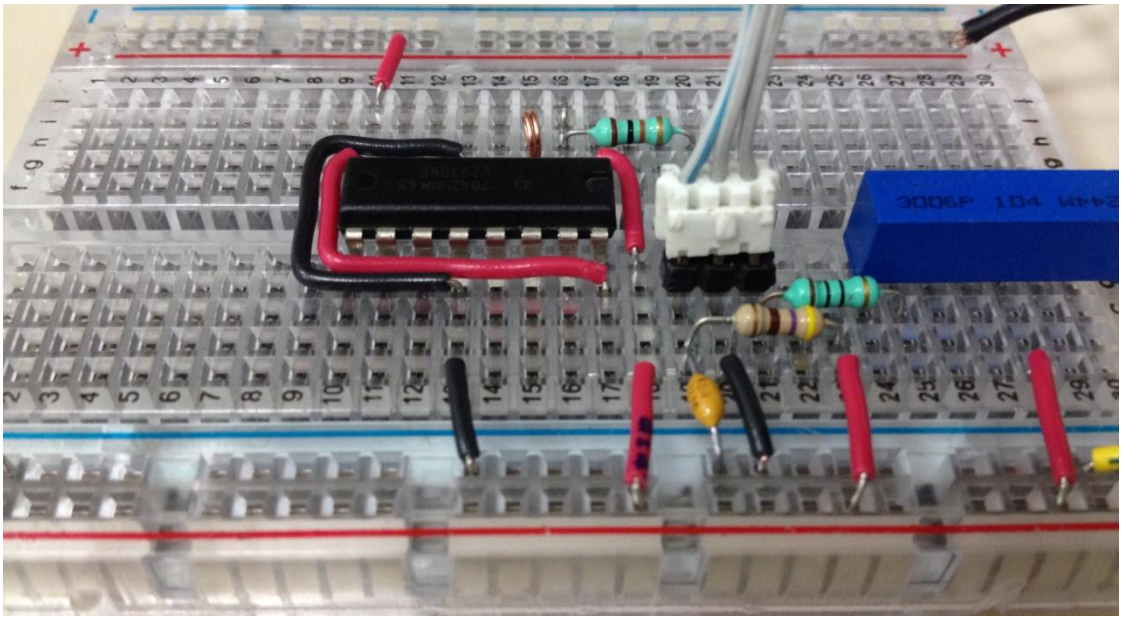

Figure 7 TIL293D and EE-SX1137 circuitry

4.2.1 Electrical diagrams

According to the corresponding technical specifications, the power stage and sensor circuitry were designed as given in the following figure.

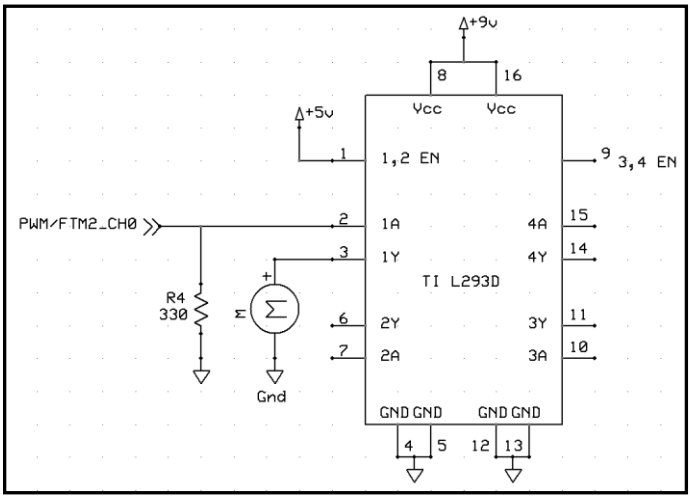

Figure 8 Power stage electrical diagram

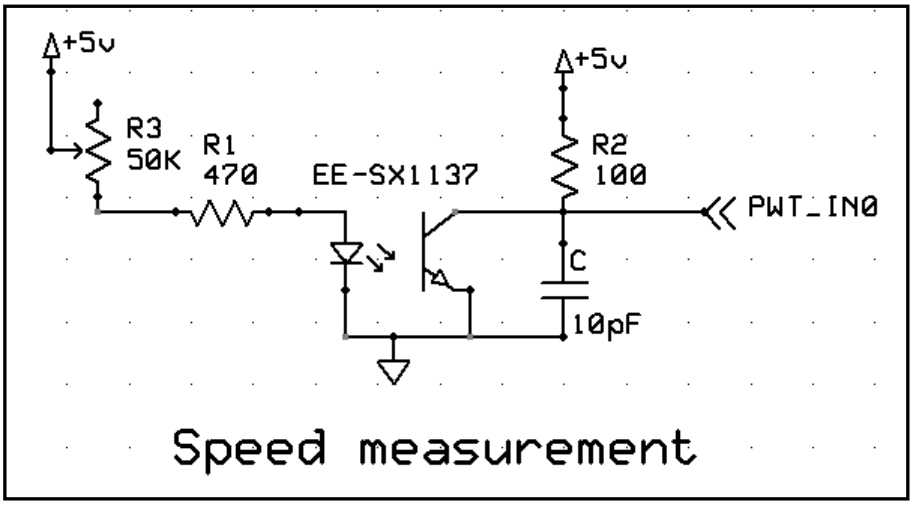

Figure 9 Optical sensor electrical diagram

According to the **TRK-KEA** schematics, the **PWT_IN0** channel is located on connector **J7** and the **FTM2_CH0** channel is located on connector **J9**.

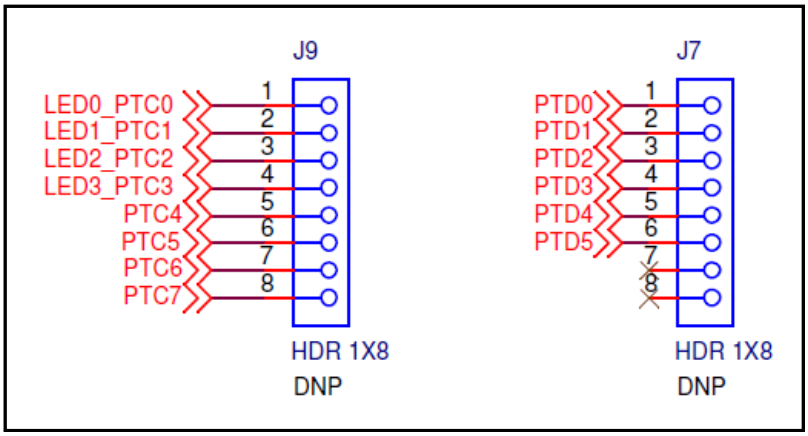

Figure 10 FTM2 and PWT pin assignment

5 Running the demo code

Now the hardware has been implemented, the next step is to test the code. This section depicts screenshots of both the terminal showing the start up menu and measurements as well as oscilloscope captures displaying the PWM and PWT signals at the various implemented speeds.

5.1 Speed control menu

As defined in the code, a serial terminal is necessary to display the speed control menu, the terminal speed is set to 9600 bauds, once the code has been compiled and a debug session started, the serial terminal should show as given in the following figure.

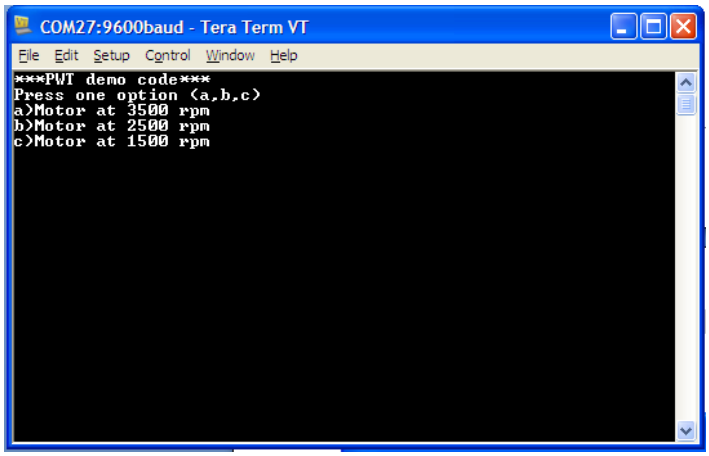

Figure 11 Speed control menu displayed at the serial port terminal

After selecting the motor speed, the microcontroller will start feeding the power stage with a PWM signal and it will also start measuring the rotor speed, both the target speed and the rotor speed will be displayed at the terminal.

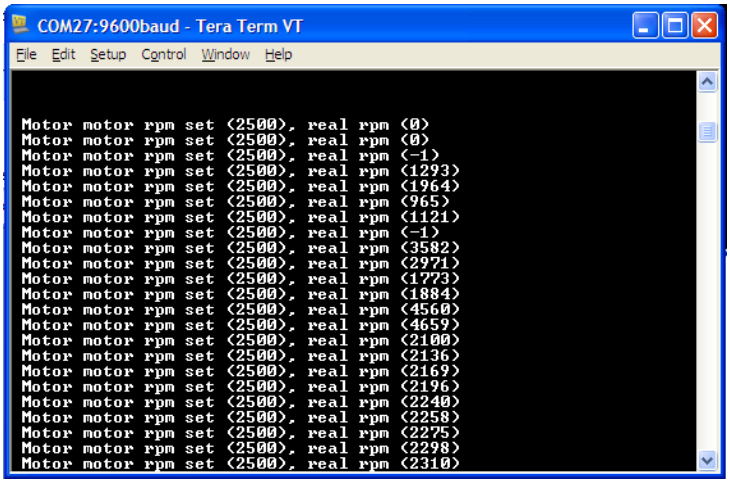

Figure 12 Rotor speed measurements in the serial port terminal

5.2 Signal captures

The following figures depict an oscilloscope capture displaying the PWM and PWT signals with their frequencies.

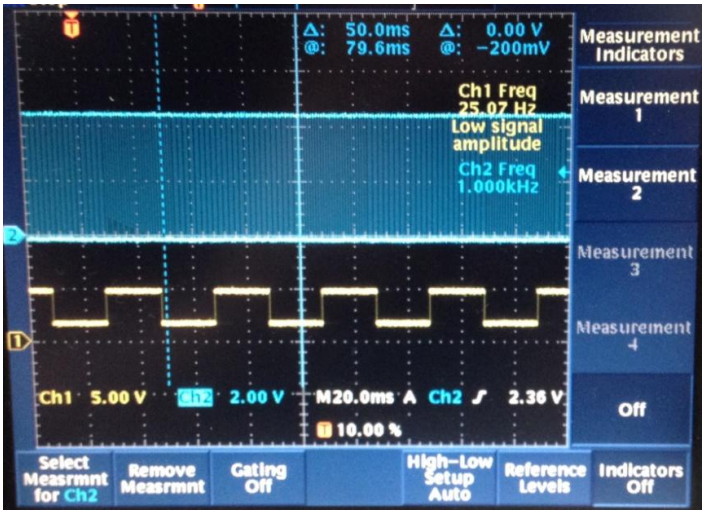

Figure 13 PWM and PWT signals

The previous figure shows PWM in channel 2 (blue) and PWT in channel 1 (yellow) when option “c” is selected (1500 rpm), confirming with the PWM frequency (25 Hz), we can reduce the rotor speed by multiplying the signals frequency 60 times.

$$RPM = 60 f_{PWM} = 60 * 25Hz = 1{,}500$$

If we increase the PWM duty cycle by selecting the b option, the rotor speed increases proportionally, as shown in the following figure.

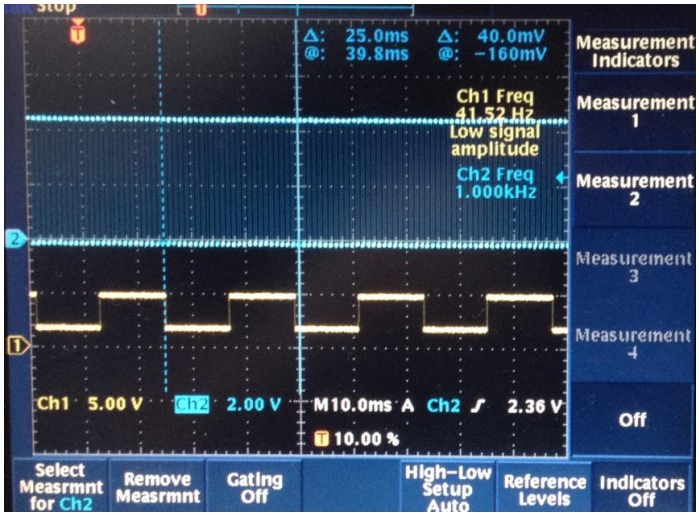

Figure 14 PWM and PWT signals with rotor speed increased

As it can be observed, the PWM frequency remains the same, 1 KHz; however the duty cycle has increased thus increasing the rotor speed, the PWT signal shows its frequency is now 41.25 Hz, if we use the previous equation we obtain.

$$RPM = 60 f_{PWM} = 60 * 41.25 Hz = 2{,}475$$

Which is close to the target speed of 2,500 rpm, the speed of course is being constantly monitored through the PWT signal and adjusted accordingly by means of modifying the PWM duty cycle provided by the FTM module to maintain a rotor speed as close to the target speed as possible.

6 Conclusion

The Pulse Width Timer provides an accurate signal frequency measurement for both the positive and negative portions of a periodic signal, useful for applications such as motor control, in conjunction with a Pulse Width Modulated signal it can effectively be used to implement a highly accurate closed loop motor control system, or any other system in which it might be necessary to measure a periodic signal frequency and duty cycle, providing not only accuracy but high flexibility.

7 References

The following references are available at freescale.com.

- *AN4839: How to Use PWT Module on Kinetis E Series*
- *AN3729: Using FlexTimer in ACIM/PMSM Motor Control Applications*

8 Revision history

Revision number	Date	Substantive change(s)
0	07/2014	Initial release

How to Reach Us:

Home Page:

freescale.com

Web Support:

freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale 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. Freescale reserves the right to make changes without further notice to any products herein.

Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale 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 Freescale 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. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, and Kinetis are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners. ARM and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.
© 2014 Freescale Semiconductor, Inc.

Document Number AN4976
Revision 0, 07/2014