**Implementation of Motor Position and Speed Measurement using Cortex-M Microcontroller**

**1. Aim:**

Encoder based measurement of position and speed of DC servomotor.

**2. Introduction:**

Motors are commonly used in several applications such as antenna positioning systems, metal cutting and forming machines, robotic joints etc. Precise position and speed control is necessary for such critical applications. To achieve this precision, several control mechanisms such as PID are applied using a microcontroller which requires position and/or speed measurement as feedback signals. These feedback signals are obtained through various sensor based and sensor less techniques some of which use encoders, hall effect sensors, potentiometers, tachometers etc.

Encoders are a popular type of sensors and are classified into various categories based on the motion translation technology or into linear and rotary, absolute and incremental. Some of the technologies used in encoders to translate motion related data are magnetic, optical, resistive, and mechanical. An optical encoder uses light beams to measure parameters. A rotary encoder, or a shaft encoder, collects data and provides feedback based on the rotation of an object such as a motor. An incremental rotary encoder can measure parameters such as distance, speed, and position whereas absolute encoders measure angular position.

To measure the position and speed of a motor through an **optical, rotary, incremental** encoder is the objective of this experiment.

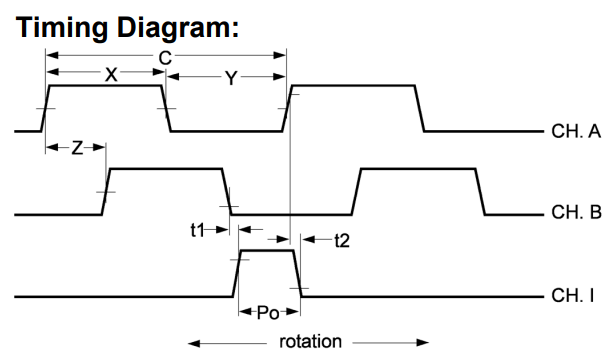
**3. Working Principle of Optical Encoders:**

A picture containing circle, screenshot, diagram, text

Description automatically generated

Above is the image of an optical encoder disk. The disk contains slots through which light passes and is received by a receiver on the other side of the disk. During rotation, the alternating opaque and transparent zones cause light to be transmitted, not continuously, but as pulses. Therefore, the term PPR or Pulses Per Revolution is widely used to describe optical encoders. PPR is equal to the number of slots on the disk and describes how many pulses can pass through the encoder disk to the receiver in one full rotation. The maximum possible amount of encoder pulses determines its resolution. These details are available in the datasheet of the encoder.

The receiver captures all incoming light pulses and generates a digital or analog output signal, sending it to the microcontroller. One of the most used angular sensors are quadrature encoders. They send the signal with results as square waves with usually three channels, A, B and I or Z.



**Channel A** determines the number of pulses.

**Channel B** is identical to Channel A but is used to determine the direction of rotation based on the phase shift of 90 degrees between the channels.

**Channel Z** is needed for notification of the passage of a certain point or zero point. It is also called encoder Z pulse (zero or index pulse). It is necessary when you need a point that defines a certain displacement. For this experiment, only Channels A and B are required.

Another useful term is CPR or Cycles Per Revolution. Here, cycle denotes a complete square wave electrical cycle. In a quadrature encoder, for every cycle on one channel, 4 pulses are generated. This is illustrated below. The definition of CPR varies by manufacturer, hence only PPR is used in this experiment for generality.

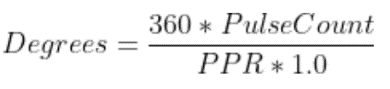
A picture containing text, diagram, font, line

Description automatically generated

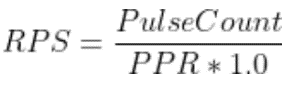
Knowing PPR and obtaining both the channels A and B signals means the position and speed can be measured.

For a time period of 1 second,

Position in degrees is,



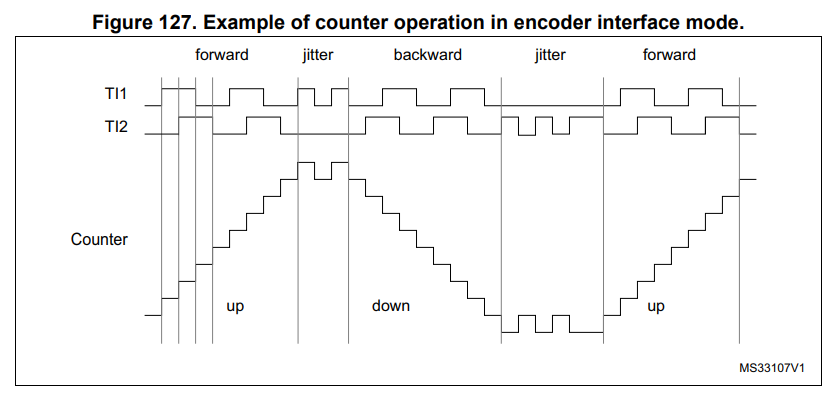
Speed in rps is,

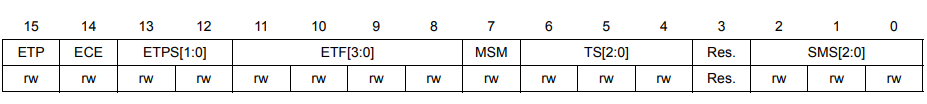


**4.1 Input Capture:**

The input capture method allows us to configure the timer as a recipient to an input which it timestamps. There is a specific “Encoder Interface Mode” in which the timer acts simply as an external clock with direction selection. This is because the square wave signals from the encoder act as an external clock.

For input capture, both channels A and B or either channel can be input with filtering and/or pre-scaling. To decide how many channels are taken as input, SMS or Slave Mode Selection bits in the SMCR or Slave Mode Control Register are configured. Filtering and Prescaling can be configured in CCMR or Capture Compare Mode Registers by manipulating the relevant bits (ICxPSC for prescaling and ICxF for filtering.)





SMCR

A picture containing text, screenshot, line, font

Description automatically generated

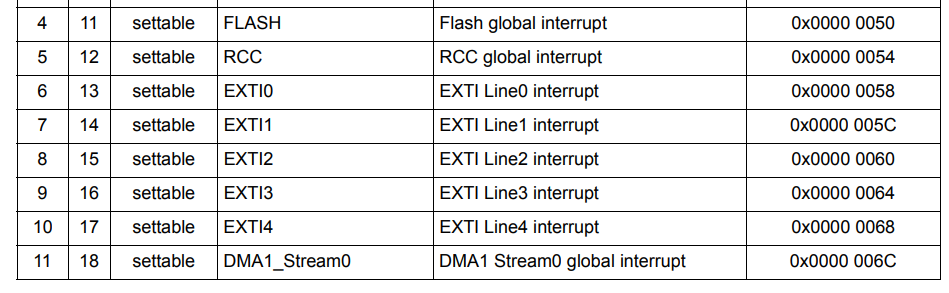
CCMR

**4.2 External Interrupt:**

In this method, for finding the position and speed only one channel input has been taken. However, you may extend the code to configure another GPIO pin and take the other channel input too and by adding flags to the ISR, you can find the direction.

Interrupts are managed by the NVIC or Nested Vector Interrupt Controller. This peripheral needs to be activated using the CPACR i.e., Co-Processor Access Control Register in the SCB or System Control Block of the processor. **Detailed information is found in the “*Cortex™ -M4 Devices Generic User Guid*e” published by ARM.** To execute the ISR, in the NVIC, the ISER or Interrupt Set Enable register needs to be configured by setting the correct bit which corresponds to the peripheral and desired interrupt. This information is found in Table 61 of RM0090 Reference Manual that contains the Interrupt Vector Table.

For example, if an external interrupt needs to be configured, the following excerpt from Table 61 provides the necessary information:



EXTI0 has a position of 6 and a priority of 13. Therefore, the 6th bit of the ISER needs to be set.

**4.3 Extra Timer:**

With either of the above methods, a different timer must be configured to generate an interrupt whose ISR performs the calculations of position/speed at regular intervals. After calculation, the value of the pulses needs to be reset to 0 so that a fresh count can begin before the next ISR.

**5. Components Required:**

1. 1 Cortex-M4 STM32F407 Discovery board
2. Motor with Encoder
3. Analog Discovery or logic analyzer for encoder signal visualization
4. Connecting wires

**6. Embedded C Codes:**

**6.1 Input Capture:**

**6.1.1 Problem Statement:**

1. Encoder channels A and B are connected to GPIO PE9 and PE11.
2. TIM1 is configured in encoder interface mode.
3. TIM2 is configured for interrupting every 1 second to compute position and speed.
4. TIM2 ISR resets the count value to take a fresh reading in the next iteration.
5. Please read the comments and accordingly comment out the portion of the code depending on whether you want both channels to be used or the rotation is clockwise or anticlockwise.

**6.1.2 Input Capture Code:**

#include "stm32f4xx.h"

#include <stdio.h>

#include<math.h>

#define ARM\_MATH\_CM4

#define PPR 1024 //Modify this for different encoders based on value given in encoder datasheet

float speedRad; // in rad/sec

uint16\_t ticks;

/\*Enabling Clocks\*/

void clocksEnable(void)

{

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOEEN; //GPIO Port E for PE9, PE11

RCC->APB2ENR |= RCC\_APB2ENR\_TIM1EN; //TIM1

RCC->APB1ENR |= RCC\_APB1ENR\_TIM2EN; //TIM2

}

/\*Configuring GPIO for input capture\*/

void configGPIO(void)

{

GPIOE->MODER |= ((2<<18)|(2<<22)); //AF for PE9, PE11

GPIOE->AFR[1] |= ((1<<4)|(1<<12)); //AF1 for TIM1\_CH1, TIM1\_CH2

}

/\*Configuring TIM1 in Encoder Interface Mode\*/

void configTIM1(void)

{

TIM1->PSC = 0x0000; //Prescalar = 0

TIM1->ARR = 0xFFFF; //Period = 65536

//Decide this based on how many channels are taken as input

//TIM1->SMCR |= (3<<0); //Encoder mode 3, both TI1 and TI2 i.e. channel A and B are captured

TIM1->SMCR |= (1<<0); //Encoder mode 1, only TI1 i.e. channel A is captured

/\*

\* IC1F=IC2F = 0b0000, IC1PSC=IC2PSC = 0b00, CC1S=CC2S = 0b01

\* TIM1->CCMR1 = 0b0000000100000001;

\*/

TIM1->CCMR1 |= ((0x1<<0)|(0x1<<8)); //CC1 and CC2 are input. CC1 = TI1, CC2 = TI2

TIM1->CCMR1 &= ~((0x3<<2)|(0x3<<10)); //Prescalars are 0 for IC1PSC and IC2PSC

TIM1->CCMR1 &= ~((0xF<<4)|(0xF<<12)); //Filters are 0 for IC1F and IC2F

TIM1->CCER |= ((0x1<<0)|(0x1<<4)); //CC1E AND CC2E ARE ENABLED

}

/\*Configuring TIM2 for an interrupt every 1s\*/

void configTIM2(void)

{

TIM2->CR1 &= ~(0x0010); //Set the mode to Count up

TIM2->PSC = 16000-1; //Set the Prescalar

TIM2->ARR = 1000-1; //Set period (Auto reload) to 1000

TIM2->SR &= ~(0x1<<0); //Clear Update interrupt flag

}

int main(void)

{

SCB->CPACR|=(0xF<<20);

clocksEnable();

configGPIO();

configTIM1();

configTIM2();

NVIC->ISER[0] |= 1<<28;

TIM2->DIER |=(1<<0);

TIM2->CR1 |= (1<<0);

TIM1->CR1 |= (1<<0);

while(1)

{

//Nothing here

}

}

void TIM2\_IRQHandler()

{

//Select which code to execute based on direction of rotation and no. of inputs.

#if 0

//For 2 channel input,

ticks = TIM1->CNT;

//Comment out speedRAD formula based on whether it's clockwise or anti-clockwise

//speedRad = (2\*3.14\*(65535-ticks))/(4\*PPR\*1.0); //rad/s for anti-clockwise as seen from ceiling

//speedRad = (2\*3.14\*(ticks))/(4\*PPR\*1.0); //rad/s for clockwise as seen from ceiling

TIM1->CNT=0;

TIM2->SR &= ~(0x1<<0);

#endif

#if 0

//For 1 channel input,

ticks = (TIM1->CNT)\*2;

//Comment out speedRAD formula based on whether it's clockwise or anti-clockwise

//speedRad = (2\*3.14\*(65535-ticks))/(4\*PPR\*1.0); //rad/s for anti-clockwise as seen from ceiling

//speedRad = (2\*3.14\*(ticks))/(4\*PPR\*1.0); //rad/s for clockwise as seen from ceiling

TIM1->CNT=0;

TIM2->SR &= ~(0x1<<0);

#endif

}

**6.2 External Interrupt:**

**6.2.1 Problem Statement:**

1. Encoder single channel signal is given to GPIO PA0.
2. TIM2 is configured for interrupting every 1 second to compute position and speed.
3. TIM2 ISR resets the count value to take a fresh reading in the next iteration.

**6.2.2 External Interrupt Code:**

/\*

\* This code computes speed of the motor using just channel A of the encoder.

\* Hence, for every cycle of signal A, there are 2 pulses generated. PPR = CPR = 1024.

\* Using external interrupts, the number of cycles is determined as pulseCount.

\* Every 1s, pulseCount is read, speed is computed and pulseCount is reset by TIM2's ISR.

\* Give input from encoder to PA0

\*/

#include "stm32f4xx.h"

#define ARM\_MATH\_CM4

#define PPR 1024

#define PI 3.14

void EXT\_Init(void);

void External\_Interrupt\_Enable(void);

unsigned int pulseCount = 0;

float positionDEG;

float positionRAD;

float speedRPS;

float speedRADS;

void EXT\_Init(void)

{

RCC->APB1ENR |= (1<<0); // enable PORTA clock

EXTI->RTSR |=(1<<0);

EXTI->IMR |=(1<<0);

}

/\*Configuring TIM2 for an interrupt every 1s\*/

void configTIM2(void)

{

RCC->APB1ENR |= RCC\_APB1ENR\_TIM2EN; //TIM2 clock enable

TIM2->CR1 &= ~(0x0010); //Set the mode to Count up

TIM2->PSC = 16000-1; //Set the Prescalar

TIM2->ARR = 1000-1; //Set period (Auto reload)

TIM2->SR &= ~(0x0001); //Clear Update interrupt flag

NVIC->ISER[0] |= 1<<28;

TIM2->DIER |=(1<<0);

}

void External\_Interrupt\_Enable(void)

{

NVIC->ISER[0] |= 1<<6;

}

int main ()

{

SCB->CPACR|=(0xF<<20); //Co-processor (core peripheral) access control register

EXT\_Init();

External\_Interrupt\_Enable();

configTIM2();

TIM2->CR1 |= (1<<0); //Enable TIM2

while(1)

{

}

}

void EXTI0\_IRQHandler( )

{

pulseCount++;

EXTI->PR |= (1<<0);

}

void TIM2\_IRQHandler( )

{

positionDEG = 360\*pulseCount/(PPR\*1.0);

positionRAD = 2\*PI\*pulseCount/(PPR\*1.0);

speedRPS = pulseCount/(PPR\*1.0);

speedRADS = 2\*PI\*pulseCount/(PPR\*1.0);

pulseCount = 0;

TIM2->SR &= ~(0x0001);

}

**7. Logic Analyzer Output:**

\*Insert images from logic analyzer here\*

**8. Conclusion:**

In this experiment, the position and speed of a motor have been measured using an encoder through both input capture and external interrupt methods.