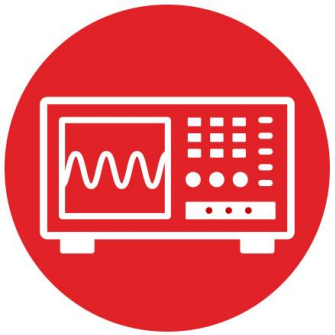

Module 16

Lab 16: Tachometer

Lab: Tachometer

16.0 Objectives

The purpose of this lab is to develop the software needed to measure motor speed. In this module,

1. You will learn more about the MSP432 Timer_A module.
2. You will configure Timer A3 for input capture measurements.
3. You will develop low-level software drivers to measure distance and speed of the two motors on the robot.

Good to Know: A typical application for embedded systems is control. Sensors measure the state of the system (motor speed), and software adjusts the actuator (PWM to motors) in an attempt to control the system in a desired manner (constant speed).

16.1 Getting Started

16.1.1 Software Starter Projects

Look at these two projects:
PeriodMeasure (uses a timer A0 to measure period on P7.3)
Lab16_Tach (starter project for this lab)

Note: You will not be able to run the **PeriodMeasure** project on the robot because this project uses Timer A0, and you are using Timer A0 for the robot's PWM outputs. You will use Timer A3 for the tachometer. Timers A1 and A2 are free to use as periodic interrupts.

16.1.2 Student Resources (in datasheet directory)

MSP432P4xx Technical Reference Manual, Timer_A (SLAU356)
MSP432P401R Datasheet, msp432p401m.pdf (SLAS826)
MotorDriverPowerDistribution.pdf Data sheet for power board
Pololu Romi Chassis User's Guide.pdf How to build the robot

16.1.3 Reading Materials

Volume 1 Sections 4.1, 9.4, and 9.7
"Embedded Systems: Introduction to the MSP432 Microcontroller",
or
Volume 2 Sections 2.2, 5.4, and 6.1
"Embedded Systems: Real-Time Interfacing to the MSP432 Microcontroller"

16.1.4 Components needed for this lab

Quantity	Description	Manufacturer	Mfg P/N
1	MSP-EXP432P401R LaunchPad	TI	MSP-EXP432P401R
1	Romi Chassis Kit - Red	Pololu	3502
1	Motor Driver and Power Distribution Board for Romi	Pololu	3543
1	Romi Encoder Pair Kit, 12 CPR	Pololu	3542
2	Rechargeable Battery, Pack of 4, Metal Hydride 1300 mAh, 1.2V, AA	Energizer	626831
4	1.375in 4-40 Nylon standoff	Keystone	4809
2	0.187in 4-40 metal nut	Keystone	4694
6	0.5in 4-40 Nylon machine screw	Pololu	1962

Lab: Tachometer

16.1.5 Lab equipment needed

Oscilloscope (one or two channels at least 10 kHz sampling)
Logic Analyzer (4 channels at least 10 kHz sampling

Warning: Disconnect the VREG↔+5V wire when the LaunchPad USB cable is connected to the PC. Connect the VREG↔+5V wire when the robot is running on battery power. This way the motors always get power from the batteries, and never get power from the USB.

16.2 System Design Requirements

The first goal of this lab is to write Timer_A software that can measure period from the two encoders. The counter of Timer_A is 16 bits wide, so the period measurement will have a **precision** of 16 bits. This means you can measure about 65536 different periods. The **resolution** is defined as the smallest change in period that the measurement can distinguish. The resolution in input capture mode is equal to the period of the selected clock. If you choose the SMCLK at 12 MHz and a prescale of 1, the period measurement resolution will be 83.33 ns. The maximum period that can be measured is the precision in alternatives times the resolution. At this clock and prescale, the **maximum** period that can be measured is about 5.4 ms.

The second goal is the use the period to determine motor speed. Since there are 360 pulses per rotation, this 5.4-ms maximum means the slowest motor speed that can be measured will be about 30 rpm. If **Period** is the period in 83.33-ns units, then the **Speed** in rpm can be calculated as

Speed (rpm) = (rotation/360pulses)*(1,000,000,000ns/sec)
*(60sec/min)/(**Period***83.33ns/pulse)

or

Speed = 2,000,000/**Period**

The third goal is to use the second input of the encoder to determine which direction the motor is spinning. You will write software that counts the number of pulses observed on each wheel as the robot moves. You will add to a counter as the robot moves forward, and you will subtract from a counter as the robot moves backward.

16.3 Experiment set-up

Refer to the data sheets of the MDPDB and encoder to see how to connect the motors and encoders. See Figure 1. Detailed directions can be found at

https://www.pololu.com/docs/0J68/all

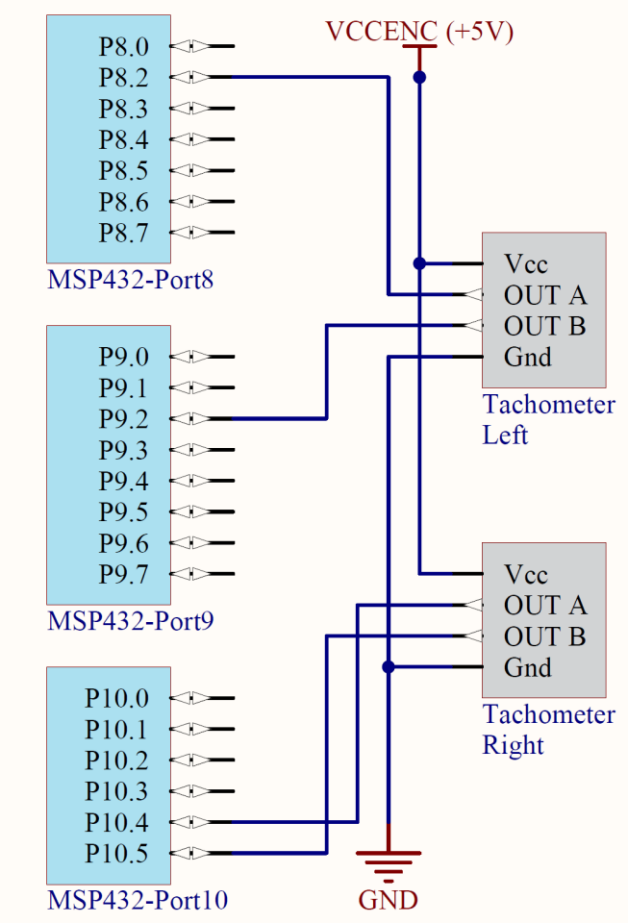

Figure 1. Possible interface for connecting the encoders to the MSP432.

Lab: Tachometer

LaunchPad	MDPDB	Encoder	Description
P8.2/TA3CCP2	ELA	OUT A	Left Encoder A
P9.2/GPIO	ELB	OUT B	Left Encoder B
P10.4/TA3CCP0	ERA	OUT A	Right Encoder A
P10.5/GPIO	ERB	OUT B	Right Encoder B

16.4 System Development Plan

16.4.1 Study the existing input capture

An efficient mechanism for learning a new skill is to first study existing art. The project **PeriodMeasure** will measure the period on P7.3 using Timer A0. You can connect a 0 to 3.3V digital wave to P7.3 using a signal generator, or you can use this main program to create a test wave. To use this program you will need to connect P2.4 output to the P7.3 input.

```
void PeriodMeasure(uint16_t time){
  P2_0 = P2_0^0x01;           // thread profile, P2.0
  Period = (time - First)&0xFFFF; // 16 bits, 83.3 ns
  First = time;                   // setup for next
  Done = 1;
}
#define PERIOD 1000  // must be even
// connect P2.4 output to P7.3
// creates a PERIOD (us) wave out P2.4
int main(void){
  Clock_Init48MHz();  // 48 MHz; 12 MHz Timer A clock
  First = 0;          // first will be wrong
  Done = 0;           // set on subsequent
  TimerA0Capture_Init(&PeriodMeasure);// capture mode
  P2->SEL0 &= ~0x11;
  P2->SEL1 &= ~0x11;  // configure P2.0 and P2.4 as GPIO
  P2->DIR |= 0x11;    // P2.0 and P2.4 outputs
  EnableInterrupts();
  while(1){
    P2_4 ^= 0x01;     // create output
    Clock_Delay1us(PERIOD/2);
  }
}
```

The resolution of the measurement is 1/12MHz = 83.33 ns and the range is about 10 us to 5.44 ms. If the period is 1 ms, then the software will return a result of 12000. This example uses **bit-banding** to access Port 2 in order to eliminate the critical section caused by the read-modify-write access to the shared global (P2->OUT).

Note: You will not be able to complete this lab without reading the MSP432 data sheet. Look at the chapter on Timer_A, and go line by line through the existing **TimerA1_Init** and **TA1_0_IRQHandler** functions within the **PeriodMeasure** project. This measurement works, but you need to understand each line, by looking up each of the registers it accesses. Once you understand each line, you will be able to convert it from measuring on P7.3 using Timer A0 to measuring both P10.4 and P8.2 using Timer A3.

16.4.2 Low-level software driver

Write the low-level driver to handle input capture on P10.4 and P8.2 using Timer A3. The prototype for the low-level driver is:

void TimerA3Capture_Init(void(*task0)(uint16_t time),
void(*task2)(uint16_t time));

This is an example of a vectored interrupt. The rising edge of P10.4 will cause an interrupt on **TA3_0_IRQHandler**, and the rising edge of P8.2 will cause an interrupt on **TA3_N_IRQHandler**. The **TA3_0_IRQHandler** ISR will call the user function passed in via the **task0** parameter, and the **TA3_N_IRQHandler** ISR will call the user function passed in via the **task2** parameter. The captured time of the edge is passed from the ISR to the user function in a manner similar to the **PeriodMeasure** project. You can use Program16_1 to test the low-level driver. Place the robot on blocks so the wheels do not touch the ground while performing initial testing.

Lab: Tachometer

```
uint16_t Period0;   // (1/SMCLK) units = 83.3 ns units
uint16_t First0;    // Timer A3 first edge, P10.4
int Done0;          // set each rising
void PeriodMeasure0(uint16_t time){
  P2_0 = P2_0^0x01; // thread profile, P2.0
  Period0 = (time-First0)&0xFFFF; // 16 bits, 83.3 ns
  First0 = time;    // setup for next
  Done0 = 1;
}
uint16_t Period2;   // (1/SMCLK) units = 83.3 ns units
uint16_t First2;    // Timer A3 first edge, P8.2
int Done2;          // set each rising

  P2_4 = P2_4^0x01; // thread profile, P2.4
  Period2 = (time-First2)&0xFFFF; // 16 bits, 83.3 ns
  First2 = time;    // setup for next
  Done2 = 1;
}
int Program16_1(void){
  Clock_Init48MHz();   // 48 MHz; 12 MHz Timer A
  P2->SEL0 &= ~0x11;
  P2->SEL1 &= ~0x11;   // P2.0 and P2.4 as GPIO
  P2->DIR |= 0x11;     // P2.0 and P2.4 outputs
  First0 = First2 = 0; // first will be wrong
  Done0 = Done2 = 0;   // set on subsequent
  Motor_Init();        // activate Lab 13 software
  TimerA3Capture_Init(&PeriodMeasure0,&PeriodMeasure2);
  Motor_Forward(7500,7500); // 50%
  EnableInterrupts();
  while(1){
    WaitForInterrupt();
   }
}
```

Note: Feel free to modify any of the details of how it works, as long as the overall system can measure motor speed for both wheels.

Adjust the period measurement resolution so that the system can measure period for a range of motor duty cycles from 25 to 100%

16.4.3 Mid-level software driver

Write the software to convert the period measurements into motor speed in rpm. Perform a static motor test while the robot is still on the blocks. For duty cycles {25, 50, 75, and 100%}, measure the motor speed of each motor in RPM.

Write a test program that periodically collects motor speeds versus time using a 100 Hz periodic interrupt. Include the bumper driver from Lab 10 or Lab 14 so the robot stops on a collision. Dump power (duty cycle) and speed data into buffers similar to Lab 10. For very long tests, you can dump into flash ROM. For shorter tests, you can dump into RAM. In the main program, perform these steps running the robot for 10 seconds.

1. Run forward at 25% duty cycle for 2 seconds
2. Run forward at 50% duty cycle for 2 seconds
3. Run forward at 75% duty cycle for 2 seconds
4. Run forward at 100% duty cycle for 2 seconds
5. Run forward at 25% duty cycle for 2 seconds
6. Stop the motors and stop the recording

Run this motor test on blocks and on a flat surface. We define the **time constant**, τ, of the motor as the time it takes to achieve $(1\text{-}e^{-1}) = 0.63$ of the final speed, given a step change in power to the motor. Fit the speed versus time data to an exponential to estimate the time-constant of your motors.

$$y(t) = S_0 + \Delta S\, e^{-t/\tau}$$

where S_0, ΔS, and τ are least squares fit of the y(t) data verses time. Initial time is defined at the point the duty cycle was changed.

16.4.4 High-level software driver

Extend the measurement to initialize the other two input pins. Create two global signed 32-bit counters, one for each motor. In addition to measuring period and motor speed, count the number of edges on each encoder. On each edge add one if moving forward and subtract one if moving backward.

16.5 Troubleshooting

Input capture interrupts do not occur:

- Check to see if the edges are occurring on P8.2 and P10.4
- Check to see if the trigger flags are being set. Bit 0 of the register TIMER_A3->CCTL[0] should be set by edge of P10.4, and bit 0 of the register TIMER_A3->CCTL[2] should be set by edge of P8.4.
- Check to see if the arm bits are set in Timer A3. Bit 4 of the register TIMER_A3->CCTL[0] arms P10.4, and bit 4 of the register TIMER_A3->CCTL[2] arms P8.4.
- Check to see if the enable bits are set in the NVIC for Timer A3. Bit 14 of the register NVIC->ISER[0] enables T3_0 (P10.4) and bit 15 enables T3_N (P8.2).
- Check to see if the I-bit in the processor is clear.

Interrupts occur over and over:

- Check the hardware with a scope or logic analyzer to make sure the sensor is operating properly
- Make sure you clear the trigger flag (acknowledge) in the ISR. Bit 0 of the register TIMER_A3->CCTL[0] should be cleared by software in the ISR of P10.4, and bit 0 of the register TIMER_A3->CCTL[2] should be cleared by software in the ISR for P8.4

16.6 Things to think about

In this section, we list thought questions to consider after completing this lab. These questions are meant to test your understanding of the concepts in this lab. The goal of this module is for you to understand Timer_A and its use for measuring period.

- What does the prescaler do for Timer_A? Why is the prescaler important (i.e., what happens when you change the prescale?)
- What is the precision of the period measurement mean and how is it determined?
- What happens if the motor spins too slowly, e.g., less than 30 RPM?
- What happens if the motor stops, e.g., does not spin at all?
- How do we debug this system if the robot is moving along the ground?
- Why is the time constant of the motor differ if the robot is on blocks versus on the ground?

16.7 Additional challenges

In this section, we list additional activities you could do to further explore the concepts of this module. For example,

- If you completed Lab 11, add LCD outputs for each of the test functions. Remember to perform LCD output only in the main program and not during an ISR.
- Add software to detect if the motor has stopped or moving less than 30 PRM. Deploy a periodic interrupt that counts the time with the semaphore clear. If 10ms has elapsed and the semaphore is still clear, you can assume the motor is moving slowly or has stopped.
- You could configure the measurement to interrupt on rising and falling edges of all four encoder pins. For each encoder define period as the time from one edge to the next edge, see Figure 1. This means there will be 4*360 (1440) edges per one rotation. In this approach, there are four times as many interrupts. This results in four times the resolution and four times the rate at which measurements are obtained. With the SMCLK at 12 MHz and prescale at 1, the maximum time that can be measured is still 5.4 ms. Consequently, this means the slowest motor speed that can be measured will be about 7.5 rpm.
- If you consider how the speed measurement will be used, you will find a new speed measurement will be needed every 10 ms. During this 10-ms time, there could be multiple input capture events. If the data is needed only once every 10 ms, you can see some data is collected and never used. We learned in previous modules that averaging can improve SNR. Consider this period measurement algorithm that averages all measurements in one 10-ms interval:

Initially, set **count** equal to zero. During an input capture interrupt

1. If **count** is 0, set **first** = time from TIMER_A3->CCTL[]
2. If **count** > 0, set **last** = time from TIMER_A3->CCTL[]
3. Increment **count**

During 10-ms periodic interrupt

1. If **count** < 2, set **period** = max value (too slow)
2. If **count** >= 2, set **period** = (**last**-**first**)/(**count**-1)
3. Set **count** equal to zero
4. Calculate **speed** from **period**

Lab: Tachometer

16.8 Which modules are next?

Module 17) Combine modules 12, 13, and 16 to create a control system that does spin the motors at a desired speed.

16.9 Things you should have learned

In this section, we review the important concepts you should have learned in this module:

- Understand the relationship between duty cycle and speed, experiencing the effect of friction.
- Be able to use input capture to measure speed.
- Know how to use interrupts to build complex real-time systems.
- Know how to write and test a low-level software driver.

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265