TI-RSLK MAX

Texas Instruments Robotics System Learning Kit

TEXAS INSTRUMENTS

Module 13

Lab 13: Timers

Lab: Timers

13.0 Objectives

The purpose of this lab is to develop the software needed to spin the motors. The software can vary the **electrical power** delivered to each motor using **pulse width modulation** (PWM). In this module,

1. You will learn the MSP432 Timer_A module.
2. You will configure Timer A0 to create two PWM outputs.
3. You will configure Timer A1 to create an additional periodic interrupt.
4. You will develop low-level robot commands for movement.

Good to Know: PWM is an effective and efficient means for the microcontroller to affect its world. It is effective because setting the timer reload value to 15000 will create an output with essentially 14 bits of precision. It is efficient because the cost of the timer and digital switching circuit (DRV8838) is much less than an equivalent analog amplifier.

13.1 Getting Started

13.1.1 Software Starter Projects

Look at these three projects:
PWMSine (uses a PWM and a timer to create a sine wave output)
PeriodicTimerA0Ints (uses Timer_A0 to create a periodic interrupt)
Lab13_Motors (starter project for this lab)

Note: You will not be able to run the PeriodicTimerA0Ints project on the robot because this project uses Timer_A0, and you need to use Timer_A0 for the robot's two PWM outputs.

13.1.2 Student Resources (in datasheets directory)

Ti-RSLK-chassis-board-v1.0-schematic.pdf Circuit diagram for TI-RSLK board
drv8838.pdf Data sheet for the H-bridge driver

13.1.3 Reading Materials

Chapter 13, "Embedded Systems: Introduction to Robotics"

13.1.4 Components needed for this lab

All components needed in this lab come with the TI-RSLK Max robot kit (TIRSLK-EVM). Batteries will be needed to power your robot.

Quantity	Description	Manufacturer	Mfg P/N
1	TI-RSLK MAX robot	TI	TIRSLK-EVM

13.1.5 Lab equipment needed

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

13.2 System Design Requirements

The first goal of this lab is to write software that can adjust the applied power to the two motors. You will create PWM outputs on the P2.6 and P2.7 pins, which are connected to the PWML and PWMR of the MDPDB (EN input to the DRV8838). The period of both outputs should be fixed at 10 ms (100 Hz). However, the software should be able to independently set the duty cycle of the EN pin to each motor from 0 to 14,998 (0 to 99.99%). At 100 Hz, the motor will not respond to individual highs and lows; rather, the motors will respond to the average level. More specifically, the delivered power will be

$$\mathbf{P} = \mathbf{V} * \mathbf{I} * \mathbf{duty}/15000;\ \ 0 \le \mathbf{duty} \le 14998$$

where **V** is the voltage and **I** the current, as measured previously in Lab 12.

The second goal of this lab is create an additional periodic interrupt using Timer_A1. The high-level main program will initialize this periodic interrupt using a function pointer at run time, providing for abstraction and code reuse.

Similar to Lab 12, the outcome of this lab is a system that drives in a straight line until one of the bump sensors detects a collision. However, contrary to Lab 12, this solution will require very little software overhead.

Lab: Timers

13.3 Experiment set-up

The set-up is same as Lab 12. Refer to the data sheet of the DRV8838 to see how the software output values to these six signals affect motor behavior, Figure 1.

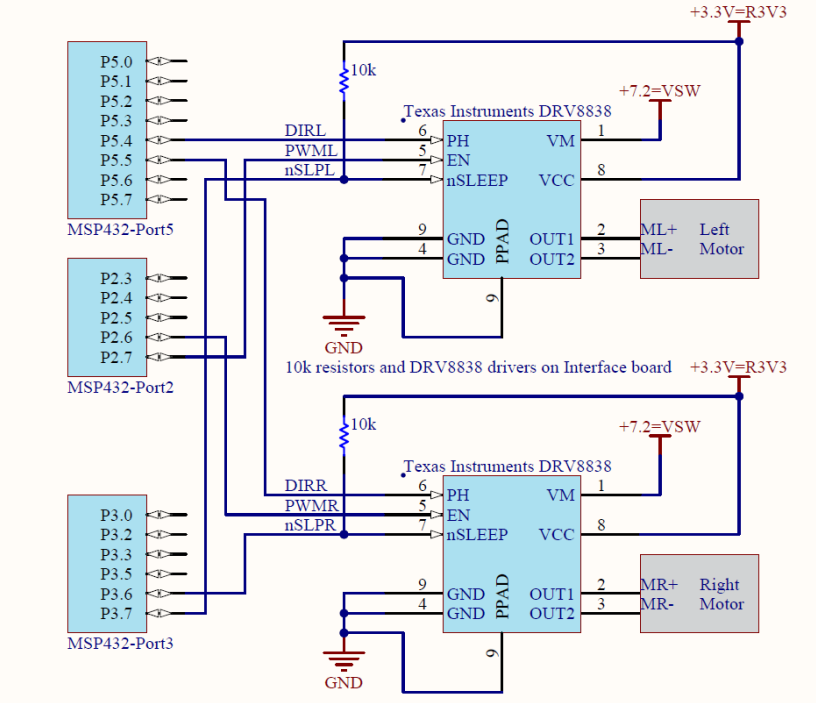

Figure 1. Interface circuit.

Warning: Please ensure the +5V jumper on the MSP432 LaunchPad is disconnected or removed. Not removing this jumper will cause permanent damage to the LaunchPad and the TI-RSLK chassis board.

PH	EN	nSleep	Motor
0	0	1	Stop
1	0	1	Stop
0	1	1	Forward
1	1	1	Back

LaunchPad	MDPDB	DRV8838	Description
P5.5	DIRR	PH	Right Motor Direction
P3.6	nSLPR	nSLEEP	Right Motor Sleep
P2.6	PWMR	EN	Right Motor PWM
P5.4	DIRL	PH	Left Motor Direction
P3.7	nSLPL	nSLEEP	Left Motor Sleep
P2.7	PWML	EN	Left Motor PWM

Safety note: A fully operational TI-RSLK MAX robot (with both motors running, line sensor and bump sensors activated) draws a total current of less than 1A. However if the robot hits a wall and your software does not stop the motor, current can rise to over 2A, which can cause damage to motors and circuits. Therefore, operating robots must periodically read the bump switches and stop the motors on a collision. See all details on the gear motor assembly on www.pololu.com.

13.4 System Development Plan

13.4.1 Low-level software driver

Replace the suite of software functions built in Lab 12 with functions that use Timer_A0 to create the two PWM outputs. This suite of functions will control the two wheels on the robot. When active the PWM to both motors will be 100 Hz (10 ms), but have independent duty cycles. The prototypes for the driver are:

void Motor_Init(void);
Initializes the 6 lines and Timer A0 and puts driver to sleep
Returns right away

void Motor_Stop(void);
Stops both motors, puts driver to sleep
Returns right away

void Motor_Forward(uint16_t leftDuty, uint16_t rightDuty)
Drives left motor forward at **leftDuty** (0 to 14,998)
Drives right motor forward at **rightDuty** (0 to 14,998)
The motors run until software issues another command
Returns right away

void Motor_Backward(uint16_t leftDuty, uint16_t rightDuty)
Drives left motor backward at **leftDuty** (0 to 14,998)
Drives right motor backward at **rightDuty** (0 to 14,998)
The motors run until software issues another command
Returns right away

void Motor_Left(uint16_t leftDuty, uint16_t rightDuty)
Drives left motor backward at **leftDuty** (0 to 14,998)
Drives right motor forward at **rightDuty** (0 to 14,998)
The motors run until software issues another command
Returns right away

void Motor_Right(uint16_t leftDuty, uint16_t rightDuty)
Drives left motor forward at **leftDuty** (0 to 14,998)
Drives right motor backward at **rightDuty** (0 to 14,998)
The motors run until software issues another command
Returns right away

13.4.2 Motor Testing

Place voltmeters on the +BAT line (+7.2) and on +5V of the TI-RSLK MAX chassis board while debugging the motor software. Place the robot on blocks, so the wheels do not touch the ground, and test the low-level motor functions, using a program like **Program13_1**.

```
// Driver test
void TimedPause(uint32_t time){
  Clock_Delay1ms(time);  // run for a while and stop
  Motor_Stop();
  while(LaunchPad_Input()==0);  // wait for touch
  while(LaunchPad_Input());     // wait for release
}
int Program13_1(void){
  Clock_Init48MHz();
  LaunchPad_Init(); // built-in switches and LEDs
  Bump_Init();      // bump switches
  Motor_Init();     // your function
  while(1){
    TimedPause(4000);
    Motor_Forward(7500,7500);  // your function
    TimedPause(2000);
    Motor_Backward(7500,7500); // your function
    TimedPause(3000);
    Motor_Left(5000,5000);     // your function
    TimedPause(3000);
    Motor_Right(5000,5000);    // your function
  }
}
```

Place the robot on the ground and try to adjust each of the 7500 parameters in the calls to ***Motor_Forward*** and ***Motor_Backward*** so robot moves in a straight line. Adjust the 5000 parameters in the calls to ***Motor_Left*** and ***Motor_Right*** and the 3000 parameters to **Pause**, so robot turns 90 degrees.

Note: Adjusting these parameters to run the robot open loop will be virtually impossible. Asking you to try to solve an impossible problem will motivate the need for inputs and create a closed loop controller.

13.4.3 Periodic Interrupt

Write the software to create an additional periodic interrupt using Timer_A1. If you use the 12 MHz SMCLK and divide by 24, the 16-bit timer will clock at 500 kHz. At this clock rate, the slowest interrupt that can be created is about 130 ms (65535*2µs). You can use a program like **Program13_2** to test this driver. Notice the use of bit-banding to remove the critical section that would normally occur with a read-modify-write sequence on a shared global.

Lab: Timers

```
// Test of Periodic interrupt
#define REDLED (*((volatile uint8_t *)(0x42098060)))
#define BLUELED (*((volatile uint8_t *)(0x42098068)))
uint32_t Time;
void Task(void){
  REDLED ^= 0x01;       // toggle P2.0
  REDLED ^= 0x01;       // toggle P2.0
  Time = Time + 1;
  REDLED ^= 0x01;       // toggle P2.0
}
int Program13_2(void){
  Clock_Init48MHz();
  LaunchPad_Init();  // built-in switches and LEDs
  TimerA1_Init(&Task,50000);  // 10 Hz
  EnableInterrupts();
  while(1){
    BLUELED ^= 0x01; // toggle P2.1
  }
}
```

Use a dual trace scope to observe both P2.0 (interrupt thread) and P2.1 (main thread). Trigger on the interrupt signal and use the gap in the oscillations on P2.1 to estimate the time required to service the Timer A1 interrupt.

After testing the PWM and Timer A1 separately, combine them into one software system that runs the robot like Program 13.1, but uses the periodic interrupt to check the bump switches, stopping the robot on a collision.

13.5 Troubleshooting

PWM or interrupts are the incorrect period:

- Check the source of the timer clock.
- Make sure the processor is running at 48 MHz.

PWM output does not occur:

- Run the **PWMSine** project to see if the hardware is ok.
- Use the debugger to make sure the Timer_A0 registers are set.

Interrupts do not occur:

- Run the **PeriodicTimerA0Ints** project and use the debugger to observe the Timer A0 registers. Run your program and observe the Timer A1 registers
- Use the debugger to observe the registers in the NVIC
- Make sure the I-bit is clear, by calling **EnableInterrupts()**;
-

13.6 Things to think about

In this section, we list 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 PWM and periodic interrupts.

- How does the software select the input clock for Timer_A?
- 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 PWM generated in this lab?
- What would happen if the main program in **Program13_2** while loop executed **P2->OUT ^= 0x04;** instead?
- How could you use Timer A1 to perform periodic tasks once a second?
- What is a function pointer? Why are function pointers used in this lab?

13.7 Additional challenges

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

- If you do not have the TI-RSLK MAX board, you will have to change the way your software operates. Luckily, it is possible to create PWM outputs on any of the P2.4, P2.5, P2.6 or P2.7.
- It is now possible to combine Lab 7 (FSM), Lab 12 (motors) and this lab to create a robot that follows a line.

13.8 Which modules are next?

The major limitation to the robot conceived in this lab is the speed of the motors depends on many factors most of which cannot be predicted in advance. Therefore the system must deploy sensors to determine its state. Over the remaining labs we will solve these limitations.

Module 15) Use the ADC to interface distance sensors. Two distance sensors can be used to drive the robot at a fixed distance and fixed angle to the wall.
Module 16) Interface tachometers (Romi Encoder Pair Kit) and use timer capture to measure the speeds of each wheel directly.
Module 17) Combine modules 12, 13, and 16 to create a control system that spins the motors at a desired speed.

13.9 Things you should have learned

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

- Understand voltage, current, and power to a motor.
- Be able to use PWM output to adjust power to the motors.
- Understand basic operation and purpose of an H-bridge.
- Know how to write and test a low-level software driver.

ti.com/rslk

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

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