模块 4

实验 4：使用 MSP432 进行软件设计

实验 4：使用 MSP432 进行软件设计

4.0 目标

本实验的目标是学会使用机器人上的循线传感器，机器人将用这个传感器来探索周围世界。

1. 您将学习 C 语言的逻辑语句、条件判断和调试方法。
2. 您将编写带有输入和输出参数的函数。
3. 您将学习逻辑和四则运算函数的实现。
4. 您将学习如何进行一致性检测以保证数据真实性。
5. 您将使用名为黑匣子功能测试的自动测试方法来验证您的算法是否正常运行。

小知识：在嵌入式系统中，软件算法的实现是一个十分重要的任务。您在本实验中学到的定义、实现和测试算法的方法可以用来解决许多机器人控制中的问题。

4.1 入门

4.1.1 从下面的软件工程起步

查看以下三个工程：
SineFunction（正弦函数的简单实现）
ProfileSqrt（平方根运算的简单实现）
Lab_SoftwareDesign（本实验的起步工程）

4.1.2 参考资料

GP2Y0A21YK0F_IR_Distance_Sensor.pdf，传感器的数据手册

4.1.3 阅读材料

Volume 1 Chapter 1, Sections 2.8, 5.1, 5.2, 5.3, 5.6, and 5.8
Embedded Systems: Introduction to the MSP432 Microcontroller",
或
Volume 2 Sections 1.4, 1.5, 3.1, 3.2, 3.3, and 3.4
Embedded Systems: Real-Time Interfacing to the MSP432 Microcontroller"

4.1.4 本实验所需组件

Quantity	Description	Manufacturer	Mfg P/N
1	MSP-EXP432P401R LaunchPad	TI	MSP-EXP432P401R

4.1.5 所需实验设备（无）

4.2 系统设计要求

在本课程中，您将学到许多能够帮助您解决机器人控制中的挑战的知识。本实验的目标是构建一些软件组件，使得机器人能够在一个有墙壁存在的世界中进行探索，正如图 1 所示。在本实验中，我们将学到如何构建 C 语言函数来收集信息，以使机器人能自主导航并行进到目标点。在真正的挑战关卡中，机器人将有三个距离传感器，通过收集这些距离传感器的数据，机器人将根据不同情境做出对应的决定。

然而，鉴于您要使用汇编语言编写此函数，您必须遵从一个程序编写标准，名为 ARM 架构过程调用标准（**ARM Architecture Procedure Call Standard**，简称 AAPCS）。这个标准由很多部分组成，但与本实验相关的主要有下列部分：

- 如果有 1 个输入参数，它将被传入 R0
- 如果有 2 个输入参数，它们将被传入 R0 和 R1
- 如果有 3 个输入参数，它们将被传入 R0-R2
- 如果有 4 个输入参数，它们将被传入 R0-R3
- 如果有一个输出参数，它将被返回至 R0
- 函数可以自由修改 R0-R3，还有 R12
- 如果一个函数想要使用 R4-R11，那么它必须使用栈来保存并还原它们
- 如果一个函数调用了另一个函数，那么它必须保存并还原 LR
- 函数必须保证栈平衡

遵从上述标准将是您能够开发能被 C 语言调用的汇编程序，而且允许您的 C 代码被汇编程序调用。具体来说，编译器在生成机器代码时将遵从这个标准。

实验 4：使用 MSP432 进行软件设计

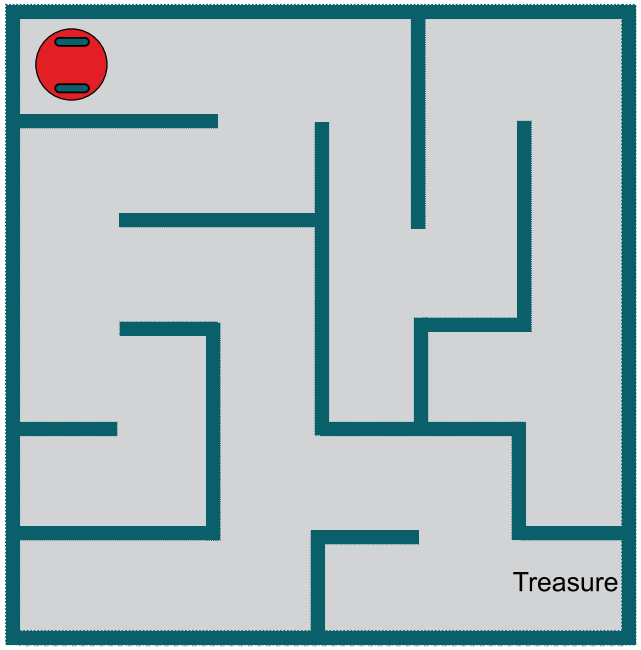

图 1. 机器人探索迷宫的示例

在模块 15 中，我们将把真正的距离传感器与 MSP432 上的模数转换器（ADC）相连。ADC 将 0 到 3.3V 的模拟电压转换为 0 至 16383 之间的数字值。本实验的第一个任务是开发一个 C 语言函数，对 TI LaunchPad 开发板上产生的 ADC 采样值进行**转换**。

注：在机器人挑战环节中您将一种距离传感器，它是由红外传感器和集成的位置探测器构成的。这种传感器也被叫做接近传感器，在机器人上，它一般被用于测量距离。

设 **n** 为 14 位 ADC 的采样值（从 0 到 16383），且 **D** 为距离，单位是 mm。二者之间存在如下的非线性转换关系

D = 1195172/(**n** – 1058)

其中 1195172 和 -1058 是校准系数的经验值，可以通过模块 15 的实验，也就是 ADC 实验取得。您的函数原型为

int32_t Convert (int32_t n);

机器人所需的第二个任务，或者说算法，是利用三个距离值从多种情境中选择机器人目前所处的情境。我们不妨假设机器人的三个距离传感器分别是**左**、**中**、**右**传感器，且每一个传感器都可以测量从机器人中心到墙壁之间的距离，单位是 mm。机器人上将有一个参考点，这三个距离都将基于这个共同的参考点来测量，如图 2 和图 3 所示。这两张图显示了 8 种不同情形，都是机器人接近一个判断点的时候有可能出现的。

软件的最底层是 I/O 口控制。**Convert** 函数将位于最底层。这个软件模块将把细节抽象出来，将做什么（测量距离）和怎么做（非线性、基于 ADC、红外距离传感器）分开。

在更顶层的模块中，软件将决定是该直行、左转、右转还是掉头。同时我们也不希望离墙壁太近。在本实验中，您将不会使用真正的距离传感器来检测距离。而您要做的是利用三个距离值（左、中、右）并判断目前情况与哪一种情形最为接近。

实验 4：使用 MSP432 进行软件设计

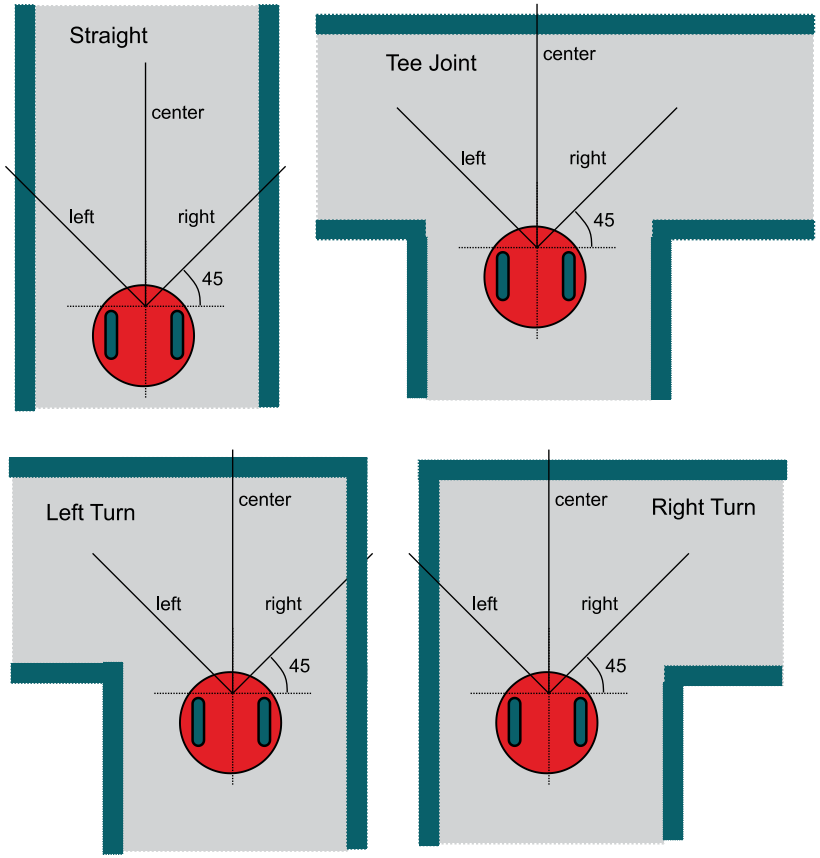

图2. 当机器人接近判断点时可能出现的4 种情形。三个变量（左、中、右）定义为机器人中心点与墙壁之间的距离。

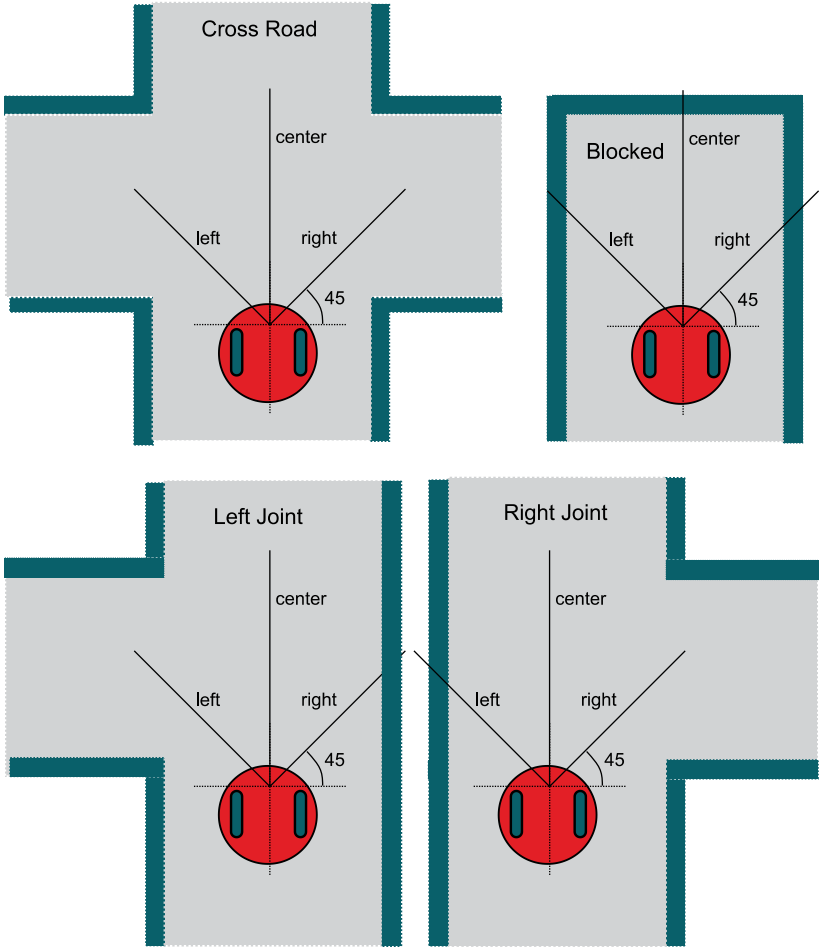

图3. 当机器人接近判断点时可能出现的另外 4 种情形。

我们从最重要的一个类型开始来定义我们的算法，即危险情况。

当左边传感器距离值小于 212 时，算法将会返回一个 **LeftTooClose** (4)错误，当右边传感器距离值小于 212 时，也将返回一个 **RightTooClose** (2)错误。当中间传感器距离值小于 150 时，算法将会返回一个 **CenterTooClose** (1)错误。上述危险

情况有可能同时发生。例如，情况 5 离中心和左边都太近。情形 7 意味着三个方向的距离都太近了。

首先我们来考虑左边和右边两个传感器。在本实验中，我们用 **#define** 语句来声明距离阈值。根据不同的机器人和迷宫搭建方式，这些阈值也需要做出相应调整。在本例中，我们的迷宫道路宽度为 400 mm，两个传感器分别放置在两个 45 度位置。

如果机器人位于道路中间，并处在直路（straight）或是死路（blocked）两种情形之一，左右两个传感器（位于 45 度位置）的数据都将是 283 mm。如果机器人距离道路中心偏差在 ±50mm 之内，那么这两个传感器得到的距离值将处于 212 到 354 mm 范围内。354 将作为阈值用来判断在遇到下一个岔路口时是否能够左转或右转。小于 354 意味着没有路径可以转弯，大于或等于 354 意味着有路径可以转弯。

最后，让我们来考虑中间的传感器。当机器人接近岔路口时，中间的传感器用来分辨究竟机器人处于以下情形中的哪一种：如果中间传感器距离值小于 600 mm，那么需要从 {Blocked, Right Turn, Left Turn, Tee Joint} 中进行分辨；如果距离值大于或等于 600 mm，那么需要分辨从 {Straight, Right Joint, Left Joint, and Cross Road} 几种情况中进行分辨。由于机器人可能处在一条长的直路上，因此中间的传感器距离值没有上限。

注：本套件中所有的传感器测量范围是 10 到 800 mm。本算法能够测量的最小值为 50 mm，但请注意这个距离是从机器人中心点而不是传感器本身开始计算的。

您需要开发一个算法，使您的机器人能够在迷宫中进行探索，并进行必要的场景分辨。假设您会将机器人上的三个传感器所测量到的距离值作为输入，并把以上所有可能场景中最接近的一个作为返回值。

这个算法一共可能有 16 个不同的输出。为了提高程序可读性，我们定义一个枚举数据类型来表示返回值。在本例中，我们为每一种可能性分配一个整型数值。这将允许我们将 1、2、4 结合起来以表示 7 种不同的危险情况。例如，5 表示左传感器和中间传感器的距离值都太近了。另外如果输入值小于 50 或大于 800，都应返回一个 **Error**。具体而言，我们进行以下定义

```
enum scenario {
  Error = 0,
  LeftTooClose = 1,
  RightTooClose = 2,
  CenterTooClose = 4,
  Straight = 8,
  LeftTurn = 9,
  RightTurn = 10,
  TeeJoint = 11,
  LeftJoint = 12,
  RightJoint = 13,
  CrossRoad = 14,
  Blocked = 15
};
typedef enum scenario scenario_t;
```

我们将使用 **#define** 语句来定义边界值，使算法更加容易理解。

```
#define SIDEMAX 354          // 两侧传感器距离墙壁最大值，单位 mm
#define SIDEMIN 212          //两侧传感器距离墙壁最小值，单位 mm
#define CENTEROPEN 600       // open/blocked 情况下中间传感器距离值
#define CENTERMIN 150        // 中间传感器距离墙壁最小值
```

您的场景归类算法原型为

scenario_t Classify(int32_t Left, int32_t Center, int32_t Right);

4.3 实验准备

本实验将使用 LaunchPad，但不包含板上任何输入输出设备。

4.4 实验步骤

4.4.1 函数和调试

编译并调试 **SineFunction** 例程。在主函数中使用调试器进行单步调试，同时观察输入和输出参数的变化。运行程序并观察数组中的结果。解释 **fsin** 开头处两个 while 循环所起到的作用。解释 **fsin** 中 if-else 语句所起到的作用。证明 **fsin** 函数能够正常工作。

实验 4：使用 MSP432 进行软件设计

编译并调试 **ProfileSqrt** 工程。使用调试器在 **sqrt** 函数的循环中设置一个断点，每执行 **sqrt** 函数一次，**t** 的值将会更新，这时观察 **n**, **s**, **t** 这几个值的变化。判断多少次迭代之后函数会收敛。想想看有哪些方法可以让程序执行的更快。

4.4.2 距离转换

使用 TI LaunchPad 开发板，编写一个 C 语言函数将 ADC 采样得到的 14 位数据转换成单位为 mm 的距离值。请注意每个 GP2Y0A21YK0F 传感器以及 MSP432 都会有轻微的差别，在程序中我们使用 **#define** 语句来封装校准参数。

注：真实的 GP2Y0A21YK0F 距离传感器的使用和校准将会在模块 15 中介绍。

```
#define IRSlope 1195172
#define IROffset  -1058
#define IRMax 2552
```

传感器能测量的最大距离是 800 mm，因此如果 ADC 采样值小于 2552（IRMax），您的函数应当返回 800。您可以使用 **Program4_1** 来测试您的 **Convert** 函数。您可以在本实验的起步工程中找到 **Program4**。这种方法叫做**功能测试**。这个测试程序包含 16 种测试情况（输入和期望输出）。理想的结果如图 4 所示。

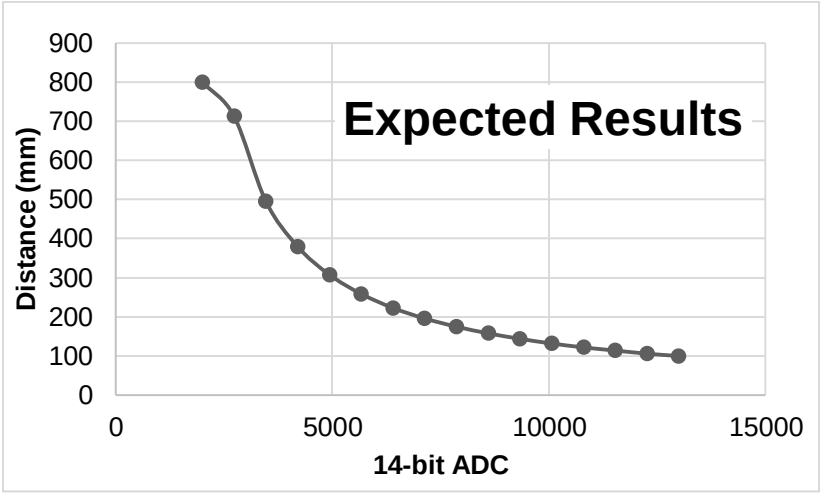

图 4. GP2Y0A21YK0F 传感器的理想转换结果

```
// Program 4_1 used to test the Convert function
int32_t const ADCBuffer[16]={2000,2733,3466,4199,4932,
 5665, 6398, 7131, 7864, 8597, 9330, 10063, 10796,
 11529, 12262, 12995};
int32_t const DistanceBuffer[16]={800,713,496,380,
 308,259,223,196,175,158,144,132,122,114,106,100};
void Program4_1(void){int i;
int32_t adc,distance,errors,diff;
  errors = 0;
  for(i=0; i<16; i++){
    adc = ADCBuffer[i];
    distance = Convert(adc); // call to your function
    diff = distance-DistanceBuffer[i];
    if((diff<-1)||(diff>1)){
      errors++;
    }
  }
  while(1){};
}
```

实验 4：使用 MSP432 进行软件设计

为了运行这个测试程序，请将 **Program4_1** 的函数名重命名为 **main**，并将原来的 **main** 函数重命名为 **main2**。运行 **Program4_1** 并将运行结果与理想结果做比较。如果您的结果误差在之内是可以接受的（这个误差可能是由四舍五入造成的）。

4.4.3 场景归类算法

要解决一个复杂问题，第一步就是将它分解为几个简单的步骤。让我们从创建 11 个不同的归类算法开始，其中每一个算法有 15 个场景。使用流程图或伪代码来定义每一个算法。例如您可以定义

CenterTooClose if (Center < CENTERMIN)

或者

Blocked if (SIDEMIN ≤ Left < SIDEMAX)
and (SIDEMIN ≤ Right < SIDEMAX)
and (CENTERMIN ≤ Center < CENTEROPEN)

对于 4.2 节中所描述的情况，一个比较好的方法是先解决 **Error** 这个情形。接下来，考虑危险情况，当出现任何危险情况时通过返回 1-7 来表示。

下一步，考虑三个距离传感器可能出现的其他值。如果出现任何输入值的组合不符合图 2 和图 3 所述的 8 种情形之一，那么请扩大选择标准来满足最接近的可能情形。如果有某个输入可能导致同时产生两种不同的输出结果，收紧选择标准来移除重叠的情况。再强调一遍，我们想要选择的是最合理的可能性。

对 **convert** 函数我们使用一组包含 16 种测试条件的测试方法，其中包含带有理想输出的、从 2000 到 12995 线性分布的输入值。对 **Classify** 函数，三个输入值均可能从 50 到 800 之间变化。因此，一共有 751^3 (423,564,751) 种可能的输入组合。我们可以用穷举测试来评估所有这些输入。然而，基于问题的本身特性，我们可以利用阈值将输入范围缩小到一个子集中。这种利用系统工作原理来有策略的选择测试值的方法叫做**边界情况**。具体而言，我们可以将被测值的数量从 751 减小到 18，而基本不影响测试准确度。特别地，我们将只测试位于阈值 ±1 的值，这些阈值包括 50, 150, 212, 354, 600 和 800。

int32_t const CornerCases[18]={49,50,51,149,150,151,211,212,213,353,
354,355,599,600,601,799,800,801};

使用边界情况可以将需要测试的值从 751^3 减小到 18^3 (5832)。
对本函数进行测试的另一种方法是利用已经能够正常工作的解决方案来进行对比。本实验中提供了一个已经能够正常运行的归类算法，但不提供源代码，这个算法在 ***Solution.obj*** 文件中。不过您可以调用这个函数来观察正常归类算法在某个输入下的输出。该算法函数的原型为

scenario_t Solution(int32_t Left, int32_t Center, int32_t Right);

您可以使用下方这个 **Program4_2** 函数来测试您的 **Classify** 函数。这个函数将测试所有 5832 种边界情况。理想的输出结果可以通过调用 Solution.obj 来查看。

```
// Program 4_2 tests the corner cases
int32_t errors;
void Program4_2(void){
  scenario_t result,truth;
  int i,j,k;
  int32_t left, right, center; // sensor readings
  errors = 0;
  for(i=0; i<18; i++){
    left = CornerCases[i];
    for(j=0; j<18; j++){
      center = CornerCases[j];
      for(k=0; k<18; k++){
        right = CornerCases[k];
        result = Classify(left,center,right); // yours
        truth = Solution(left,center,right);  // correct
        if(result != truth){
          errors++;
        }
      }
    }
  }
  while(1){
  }
}
```

实验 4：使用 MSP432 进行软件设计

4.5 疑难解答

转换（convert）函数不能正常工作：

- 使用 **Program 4_1** 找出无法转换的输入，写一个主函数调用您的转换函数来测试这个输入值，并利用单步调试来比较函数内部的计算结果和理想输出。
- 如果仍有问题，我们建议您将计算过程分解成多个步骤（每一行 C 代码只执行一次四则运算），这样您可以用单步调试来观察每一个计算步骤。

归类（classify）函数不能正常工作：

- 使用 **Program 4_2** 找到无法正常归类的输入，将您的输出与理想输出做比较。利用图 2 和图 3 来思考哪一种情景更符合这个输入。写一个主函数并在其中调用您的归类函数来计算这个输入，使用单步调试找到您的函数与理想结果之间的区别。
- 如果仍有问题，请于您的老师或同学们讨论。在讨论过程中您有可能得到解决问题的其他思路。

4.6 请思考

本环节中将列出一些问题，您可以在完成本实验后思考它们。这些问题是为了检验您对于本实验中所涉及概念的理解。

- 软件是如何处理图 3 所示的距离传感器的非线性响应的？
- 尽管所有距离值都是无符号的，但我们都用有符号数来表示它们。如果您尝试使用无符号数来实现转换过程，您将看到一个编译器的 warning（代码仍将正常工作）。为什么编译器会提示这个 warning？
- 测试一个程序往往比写程序本身更加困难。请列出本实验中所介绍的测试方法。
- 为什么我们允许 **Convert** 函数出现 ±1 的误差？
- 什么极端情况会导致机器人处于情境 7 中？

4.7 其他挑战

本环节中将列出一些问题，供您在完成实验之后思考。这些问题是为了检验您对于本实验中所涉及到的概念的理解。您可以扩展本系统或提出其他完全不同的观点。例如：

- 考虑使用 Program 4_3 来进行穷举测试，您可以在起步工程中找到这个函数。这个测试可能需要花费 16 个小时来完成。穷举测试方法有什么好处？
- 使用两个距离传感器和一个前置碰撞开关来重新设计归类算法。
- 使用四个距离传感器来重新设计归类算法。
- 如果有五个距离传感器，您还可以计算机器人和左右墙壁之间的角度。
- 如果没有已经正常工作的解决方案，请思考您可以如何测试 **Classify** 函数。例如，假设您没办法看到源代码，有什么方式能够将全班同学的解决方案结合到仪器？

4.8 接下来是哪些模块？

我们将利用接下来几个实验来创建控制机器人所需要的其它软件代码。输入/输出是嵌入式系统中的一个重要部分。接下来的模块中我们将学习这个模块：

模块 5)　开始搭建机器人，包括电池和电压转换
模块 6)　学习如何控制微控制器的输入输出引脚
模块 7)　学习利用有限状态机来控制机器人
模块 8)　使用微控制器与真实开关及 LED 进行交互
　　这将允许更多输入输出设备加入系统，以处理更复杂的问题
模块 9)　开发一个简单的 PWM 输出程序，并调整占空比

实验 4：使用 MSP432 进行软件设计

4.9 您应该已经学会

本环节中我们来回顾一下本模块中您应该已经学会的重要概念：

- 利用函数进行软件抽象
- 利用 AND 和 OR 来编写逻辑判断函数
- 利用加减乘除进行四则运算
- 利用 #define 语句来增强程序可读性
- 利用 enum 和 typedef
- 用 if 语句进行判断
- 如何处理错误情况
- 使用调试器进行单步调试并查看变量
- 进行函数测试
- 利用边界情况来缩短测试时间

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
Copyright © 2018, Texas Instruments Incorporated