第 3 章 Intel Galileo 平台例程

通过第 1 章和第 2 章的内容，已经搭建了开发环境，掌握了开发工具和配置方法。本章将进行 Intel Galileo 平台的基础开发，主要包括数字 IO、模拟 IO、PWM 波和串口通信，为后续第 4 章的应用制作做好准备。

3.1 数字 IO 的输出

3.1.1 实例功能

本实例将着重介绍 Galileo 上数字 IO 的输出，主要包括 GPIO 的引脚声明、初始化和电平设置，并以 LED 为例，描述其过程。

3.1.2 硬件电路

本例程需要的元器件包括 LED 灯一个、330 欧姆电阻一个、面包板一块、连接线若干，其连接如图 3-1 所示。

需要注意的是，LED 灯的正极（较长的引脚端）通过电阻接 Galileo 数字引脚 13，负极（较短的引脚端）接 Galileo 的 GND。也就是说，程序中通过改变数字引脚 13 的电平高低来达到控制 LED 亮灭的目的。

3.1.3 程序设计

打开 Visual Studio，新建项目，选择“File→New Project”命令，在打开的 New Project 对话框中展开“Templates→Visual C++→Windows for IoT”节点，选择模板 Galileo Wiring app，将默认的工程名修改为 HelloGalileo，如图 3-2 所示。

Visual Studio 默认生成的程序代码结构如图 3-3 所示。

其中，用户主要编写的 main 函数部分在 Main.cpp 文件中。如果有用 Arduino IDE 写过程序的朋友，看到 Visual Studio 生成的 Main 程序应该会感觉到很熟悉，因为其程序的基本结构类似，包含 setup 和 loop 函数，分别用于初始化和程序的循环。

本例中，需要 LED 每隔 1 秒改变一次状态，修改代码如下。

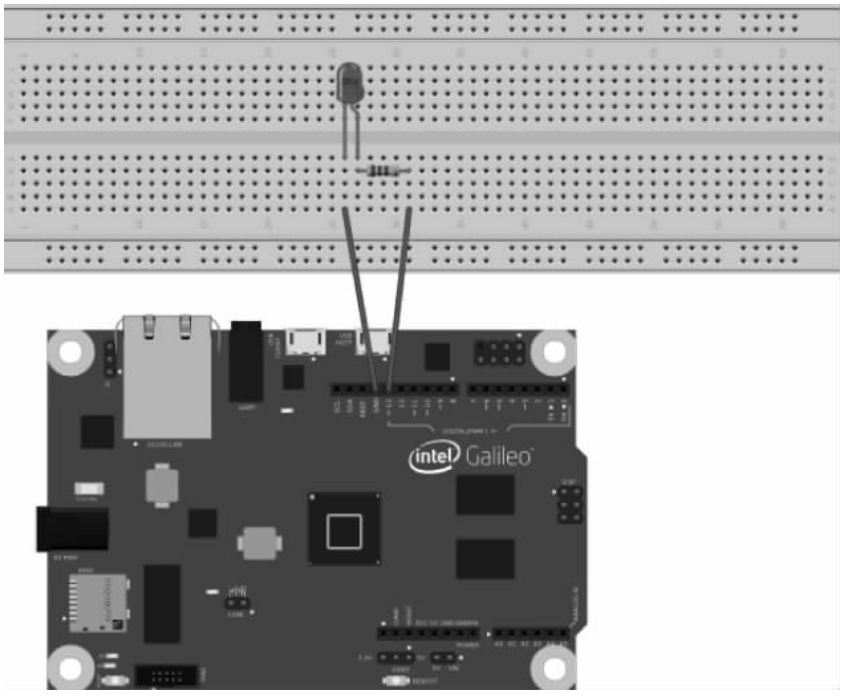


图 3-1　硬件连接图

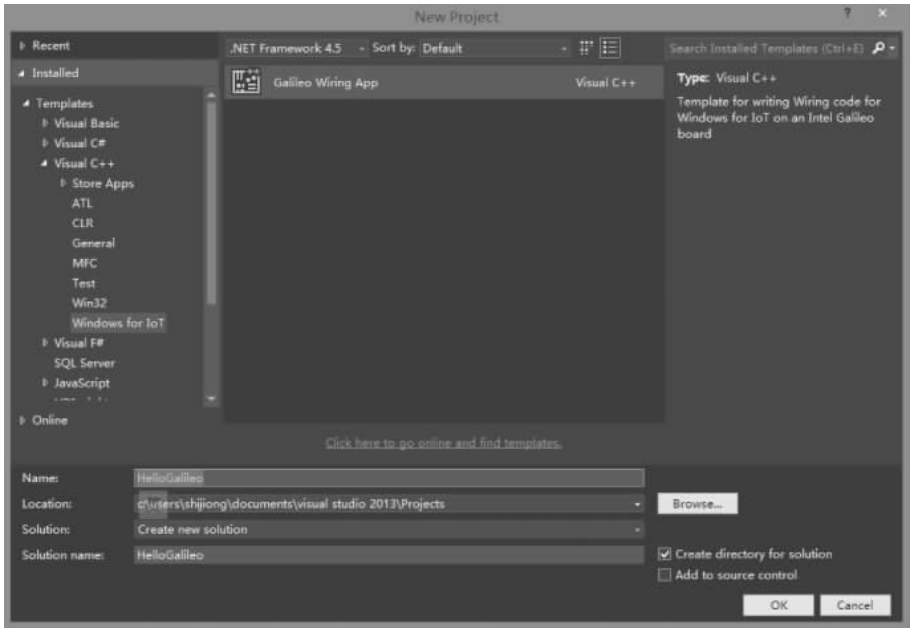


图 3-2　新建工程

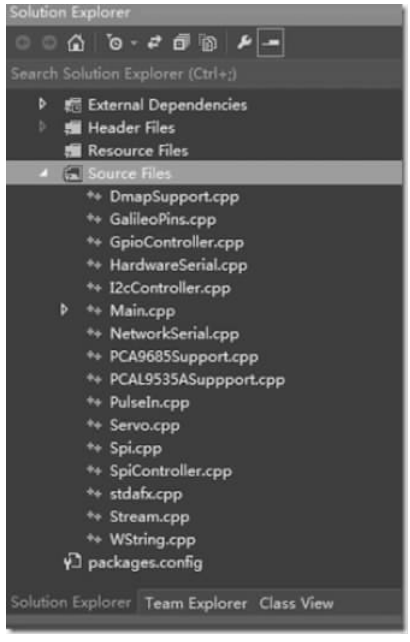


图 3-3 Visual Studio 默认生成的代码结构

代码清单 3-1：数字 IO 的输出

Main.cpp 文件主要代码

```
// ----------------------------------------------------------------------
#include "stdafx.h"
#include "arduino.h"

int _tmain(int argc, _TCHAR * argv[])
{
    return RunArduinoSketch();
}

int led = 13;                    // 申明连接 LED 的引脚号

void setup()
{
    // TODO: Add your code here
    pinMode(led, OUTPUT);        // 配置该 LED 引脚为输出模式
}

// the loop routine runs over and over again forever:
void loop()
{
    // TODO: Add your code here
```

```
    digitalWrite(led, LOW);      // 通过设置低电平来熄灭 LED
    Log(L"LED OFF\n");           // 打印调试信息
    delay(1000);                 // 延时 1 秒
    digitalWrite(led, HIGH);     // 通过设置高电平来点亮 LED
    Log(L"LED ON\n");            // 打印调试信息
    delay(1000);                 // 延时 1 秒
}
```

上述程序中，通过 int led=13 声明了 Galileo 连接 led 的引脚为 13；然后在 setup 函数中通过 pinMode 方法将其设置为输出模式；最后在 loop 函数中，通过 digitalWrite 方法设置引脚的电平高低，每改变一次状态，就使用 delay(1000)来延时 1 秒。

3.1.4 部署与调试

程序编写完成以后，在解决方案浏览器中选中该项目，单击鼠标右键，选择 Property(属性)命令，在弹出的对话框中，选择 Debugging 选项，如图 3-4 所示。

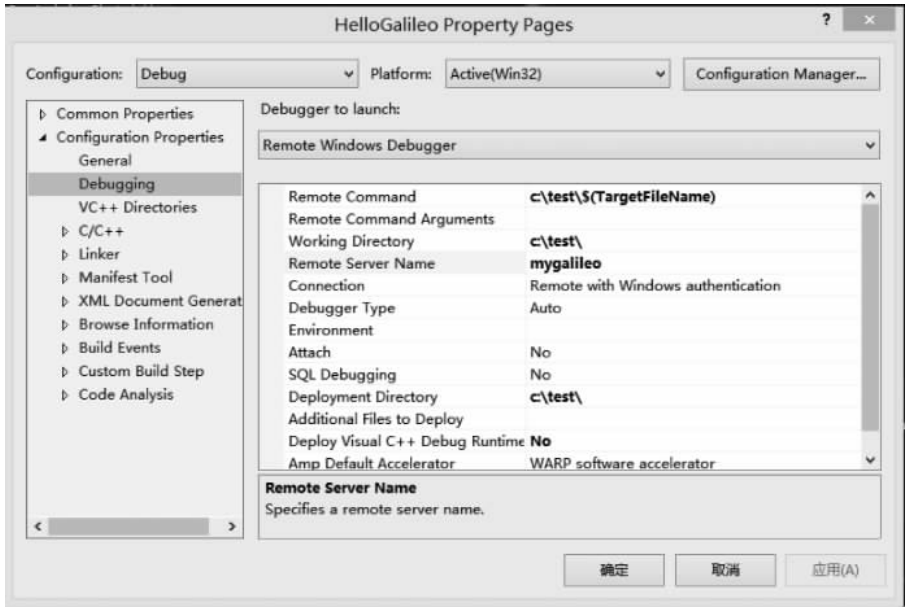


图 3-4 项目属性对话框

列表中的 Working Directory 为应用程序部署到设备上的路径，Remote Server Name 是 Galileo 设备的名称，默认为 mygalileo，这里也可以输入 Galileo 板子的 IP 地址。

然后，按 F5 键进行编译和调试。如果是第一次调试，Visual Studio 会弹出一个安全认证对话框，只需要输入对应的用户名和密码，例如：

```
Username: mygalileo\Administrator
Password: admin
```

其示意图如图 3-5 所示。



图 3-5 Visual Studio 调试的登录对话框

部署成功以后，就可以看到 LED 灯不停地闪烁，间隔时间为 1 秒，如图 3-6 所示。

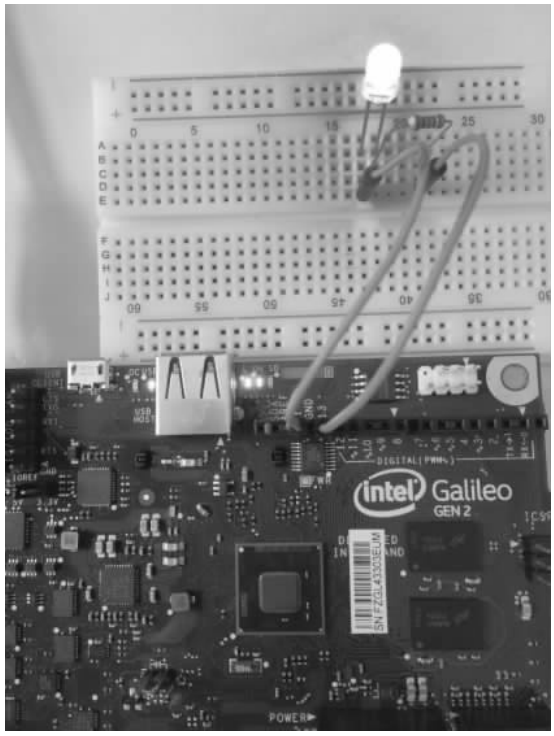


图 3-6 程序运行的实物图

同时，在 Telnet 的客户端上，打印出 LED 实时状态的调试信息，如图 3-7 所示。

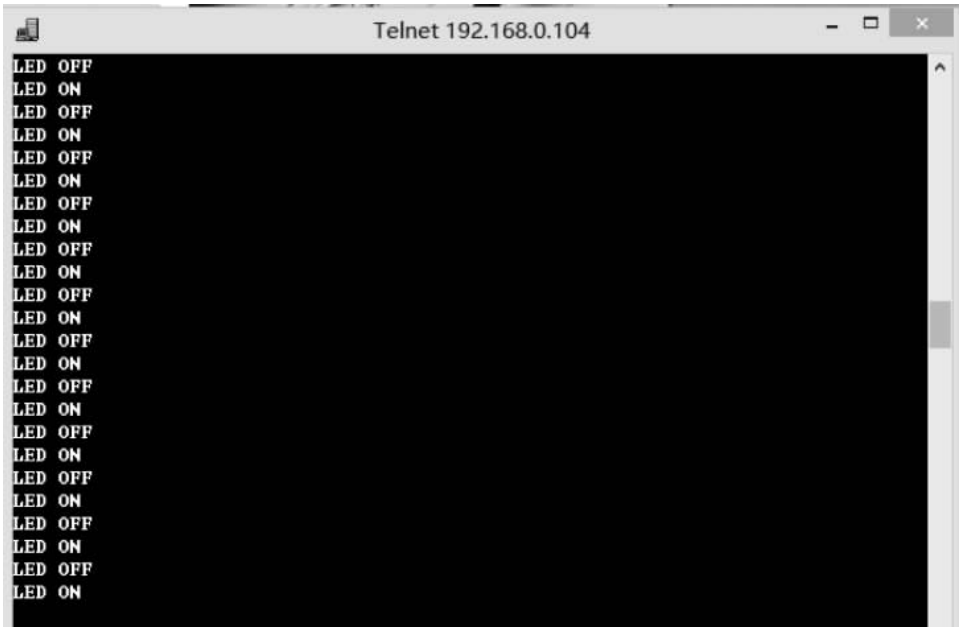


图 3-7　Telenet 调试信息输出

3.2　数字 IO 的输入

3.2.1　实例功能

本实例将着重介绍 Galileo 上数字 IO 的输入，主要包括 GPIO 的引脚声明、初始化和电平设置，并在 3.1 节 LED 操作基础上，添加开关（按钮），利用开关（按钮）来控制 LED 灯的状态。

3.2.2　硬件电路

本实例需要的元器件包括 LED 灯一个、330 欧姆电阻一个、10k 欧姆电阻一个、开关按钮一个、面包板一块、连接线若干，其元器件连接图如图 3-8 所示。

其原理图如图 3-9 所示。

图 3-9 中，R1 为 330 欧姆的限流电阻，R2 为 10k 欧姆的上拉电阻。由图可知，开关按钮 S1 在没有闭合的情况下，D7 引脚能够检测到高电平，在开关按钮 S1 闭合的情况下，D7 检测到的电平为低电平。

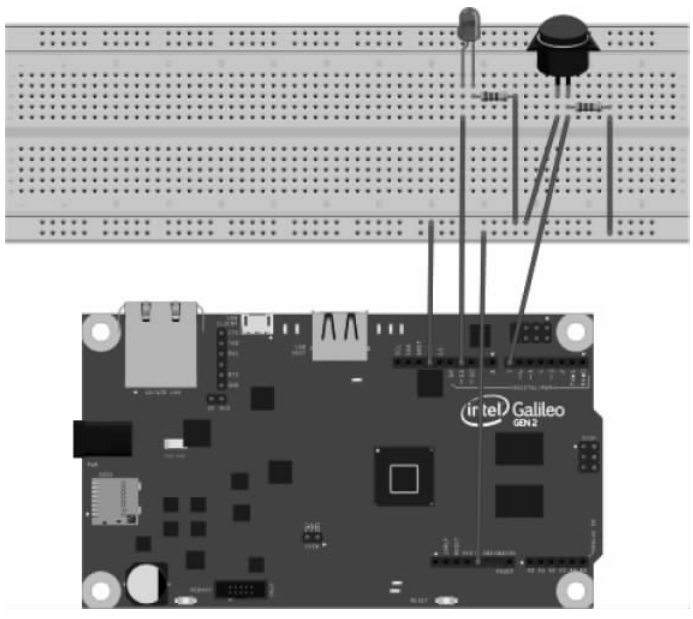


图 3-8　元器件连接图

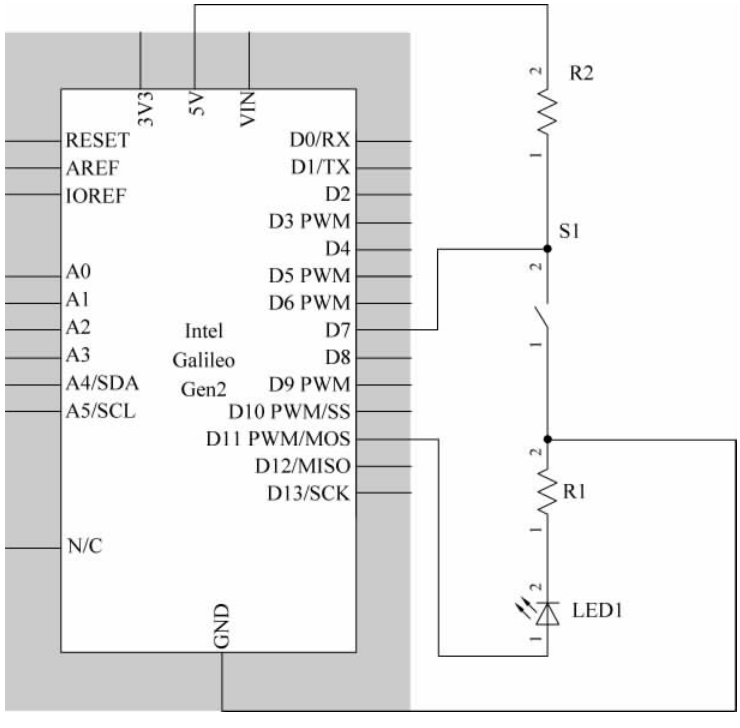


图 3-9　原理图

3.2.3 程序设计

打开 Visual Studio，新建项目，选择“File→New Project”命令，在弹出的 New Project 对话框中展开“Templates→Visual C++→Windows for IoT”节点，选择模板 Galileo Wiring app，将默认的工程名修改为 GalileoButton，如图 3-10 所示。

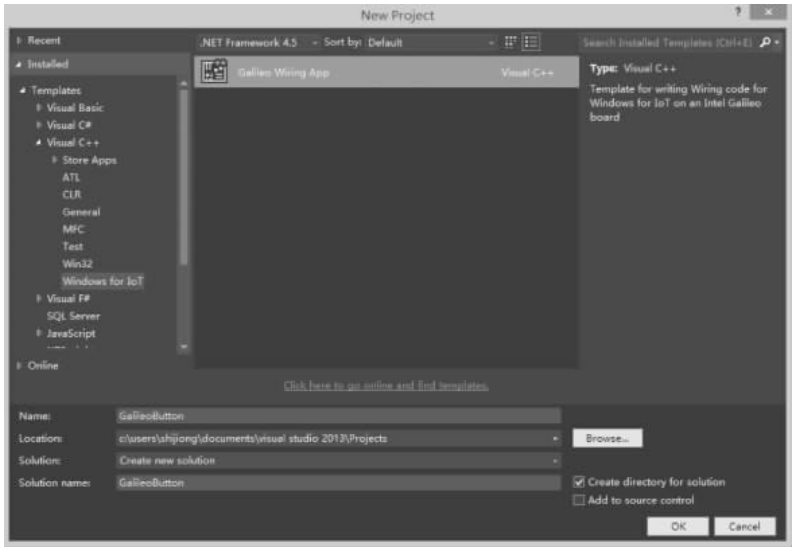


图 3-10 新建工程

打开 Main.cpp 文件，修改 main 函数如下。

代码清单 3-2：数字 IO 的输入

Main.cpp 文件主要代码

```
// ----------------------------------------------------------------------------
#include "stdafx.h"
#include "arduino.h"

int _tmain(int argc, _TCHAR* argv[])
{
    return RunArduinoSketch();
}

int ledpin = 11;                //定义数字 11 接口
int inpin = 7;                  //定义数字 7 接口
int val;                        //定义变量 val
void setup()
{
    pinMode(ledpin, OUTPUT);    //定义小灯接口为输出接口
    pinMode(inpin, INPUT);      //定义按键接口为输入接口
}
```

```
void loop()
{
    val = digitalRead(inpin);   //读取数字 7 口电平值赋给 val
    if (val == LOW)             //检测按键是否按下,按键按下时小灯亮起
    {
        digitalWrite(ledpin, LOW);
    }
    else
    {
        digitalWrite(ledpin, HIGH);
    }
}
```

上述代码中,首先通过 setup 函数,设置数字 7 和 11 引脚的输入输出模式;然后在 loop 函数中,不断地通过 digitalRead 获取数字 7 脚的电平状态,从而根据该状态来改变 LED 灯的点亮和熄灭,其 LED 灯的控制通过 digitalWrite 来实现。

3.2.4 部署与调试

代码编写完成以后,按照 3.1.4 节描述的方法进行部署与调试。如果硬件连接正确,就会看到结果,如果用户按下开关按钮,LED 灯就亮起,如果用户放开开关按钮,LED 灯就熄灭,如图 3-11 所示。

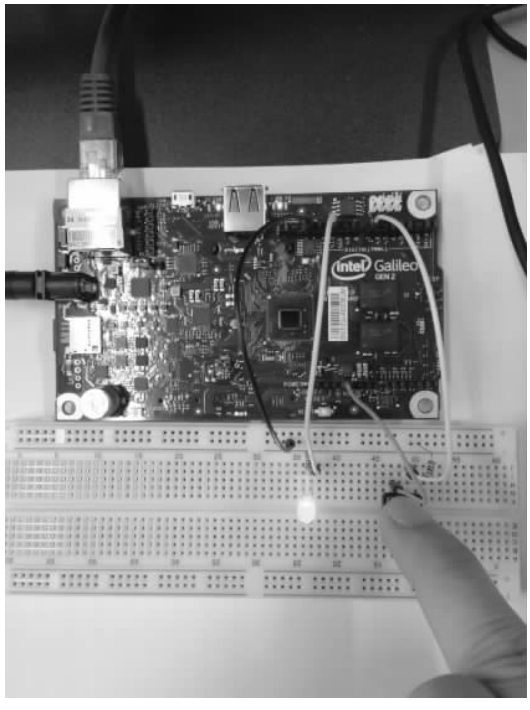


图 3-11 程序运行的实物图

3.3 模拟 IO 的输入

3.3.1 实例功能

本实例将着重介绍 Galileo 上模拟 IO 的使用，主要包括引脚声明、初始化和模数转换，并以温度传感芯片 LM35 为例，描述其获取数据并处理显示的过程。

3.3.2 硬件电路

本实例需要用的元器件包括温度传感器 LM35 一个、面包板一块、连接线若干、330 欧姆电阻一个、0.1μF 瓷片电容一个。其元器件连接如图 3-12 所示。

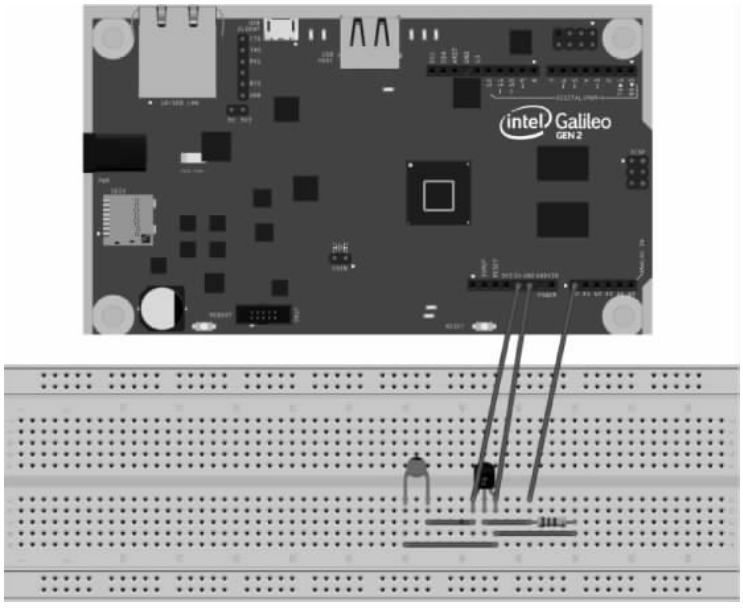


图 3-12 元器件连接图

原理图如图 3-13 所示。

细心的朋友会发现，在基于 Arduino 的方案中，LM35 并不需要加电容和电阻，而这里基于 Galileo 的方案却需要。原因如下：

(1) 在 LM35 的 5V 引脚和 GND 之间加一个小电容能够滤除电源噪声带来的干扰。

(2) 在 LM35 的信号输出引脚和 Galileo 的 A0 引脚之间加一个下拉电阻，可以滤除 LM35 信号引脚的尖峰干扰，因为 Galileo 的运行频率相对于 Arduino 较高，较容易受尖峰的干扰。

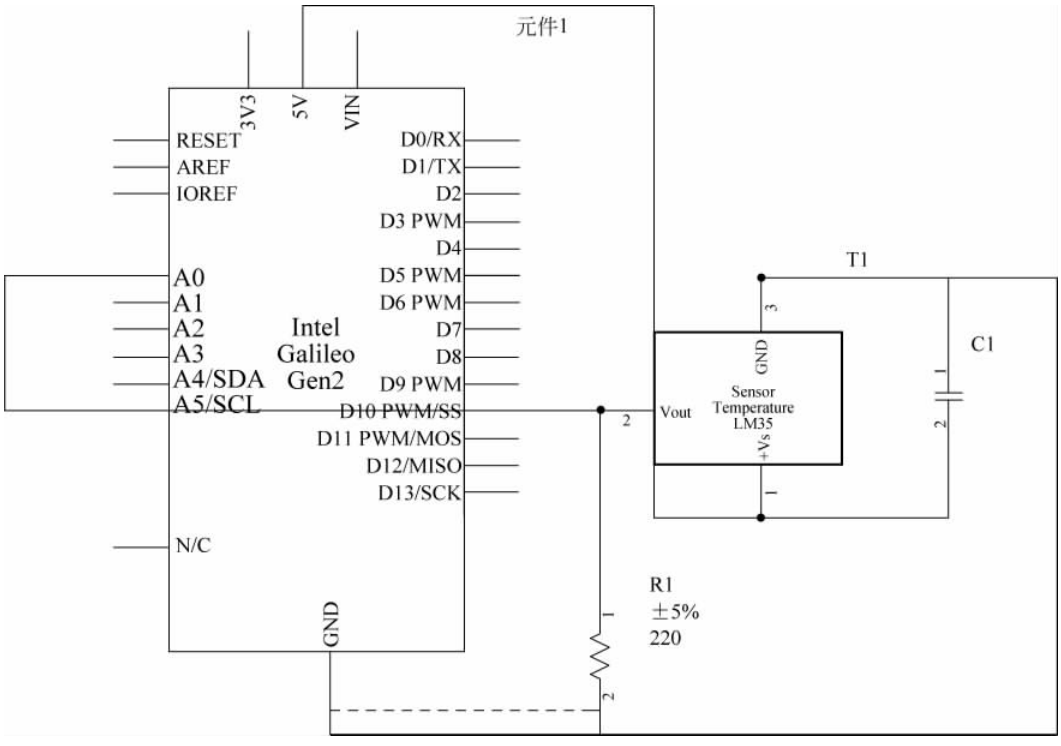


图 3-13 原理图

3.3.3 程序设计

新建工程，选择模板(即展开“Templates→Visual C++→Windows for IoT”节点，选择模板 Galileo Wiring app)，将默认的工程名修改为 GalileoTempture，如图 3-14 所示。

打开 Main.cpp 文件，修改 main 函数如下。

代码清单 3-3：模拟 IO 的输入

Main.cpp 文件主要代码

```
// ------------------------------------------------------------------------
#include "stdafx.h"
#include "arduino.h"

int _tmain(int argc, _TCHAR* argv[])
{
    return RunArduinoSketch();
}

int potPin = A0;                    //定义模拟接口 0,连接 LM35 温度传感器

void setup()
```

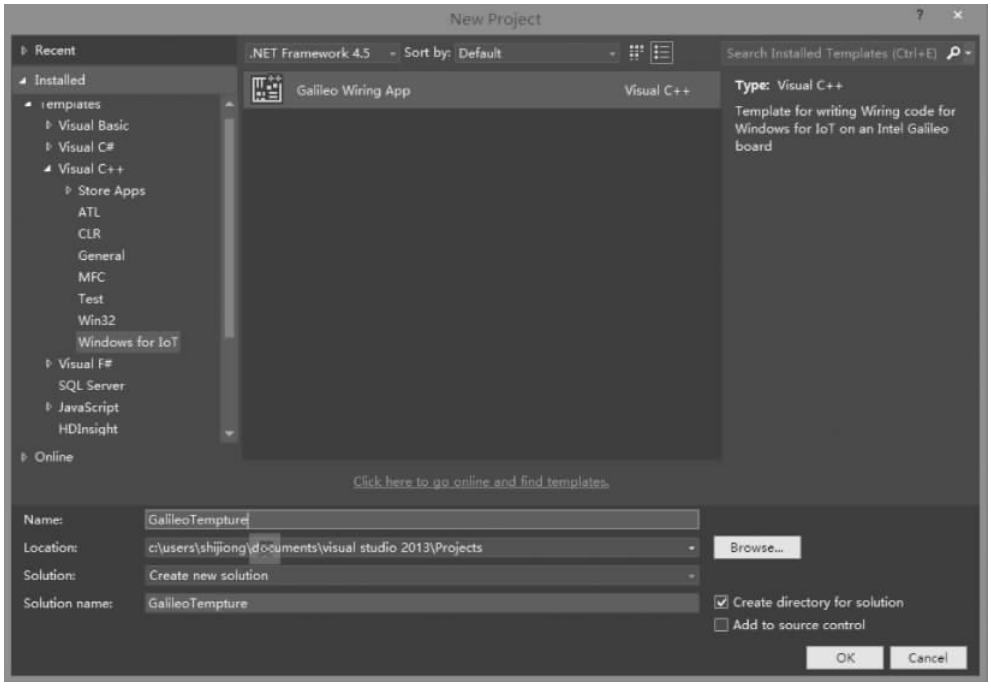


图 3-14　新建工程

```
{
}
void loop()
{
    int val;                    //定义变量
    int dat;                    //定义变量
    val = analogRead(potPin);   //读取传感器的模拟值并赋值给 val
    dat = (125 * val) >> 8;     //温度计算公式
    Log(L"Tep:");
    Log(L"%d", dat);            //以十进制显示 dat 变量数值
    Log(L"C\r\n");              //打印回车换行
    delay(1000);                //延时 1 秒
}
```

上述代码在 loop 循环中，每隔 1 秒，通过 analogRead 获取 LM35 的信号引脚的电压，转换成数字信号，然后通过 dat＝(125 * val)＞＞8 计算出实际温度，最后显示在调试窗口中。

3.3.4　部署与调试

代码编写完成以后，按照 3.1.4 节描述的方法进行部署与调试。如果硬件连接正确，就会在 Visual Studio 的 Output 窗口中显示 LM35 检测到的温度信息，如图 3-15 所示。



图 3-15　温度调试信息输出页面

实物连接和测试如图 3-16 所示。

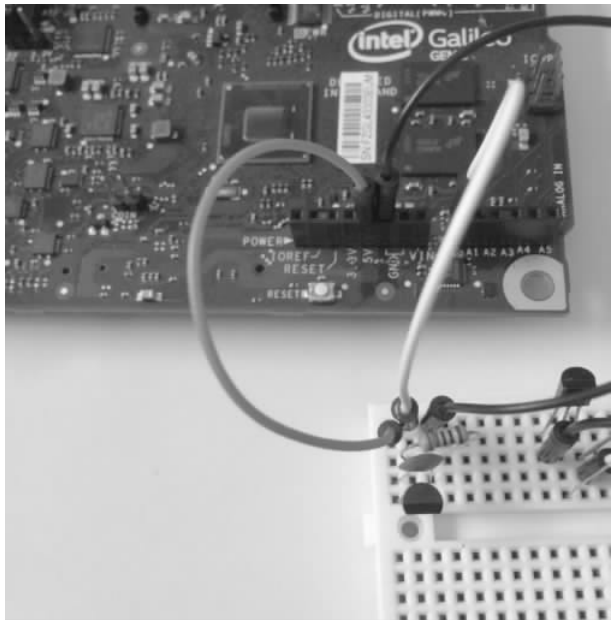


图 3-16　程序运行的实物图

3.4　PWM 波的输出

3.4.1　实例功能

PWM (Pulse Width Modulation)就是通常所说的脉冲宽度调制，简称脉宽调制，广泛应用于灯具调光、电机调速和声音制作等领域。PWM 是一种对模拟信号电平进行数字编码的方法，由于计算机无法输出模拟电压，只能输出 0 或 5V 的数字电压值，就通过使用高分辨率计数器，利用方波的占空比被调制的方法来对一个具体模拟信号的电平进行编码。与 Arduino 类似，Galileo 提供了 6 个 PWM 功能的引脚，分别是 3、5、6、9、10 和 11。

本实例演示了如何利用 Galileo 输出 PWM 波，包括引脚的声明、初始化和 PWM 的输出操作。

3.4.2 硬件电路

本例程需要的元器件包括 LED 灯一个、330 欧姆电阻一个、面包板一块、连接线若干，其连接和 3.1 节的硬件电路类似，只是 Galileo 的控制引脚由 13 引脚变为 11 引脚，其他不变。

3.4.3 程序设计

打开 Visual Studio，新建项目（即选择“File→New Project”命令），选择模板（即展开“Templates→Visual C++→Windows for IoT”节点，选中 Galileo Wiring app），将默认的工程名修改为 GalileoPWM，如图 3-17 所示。

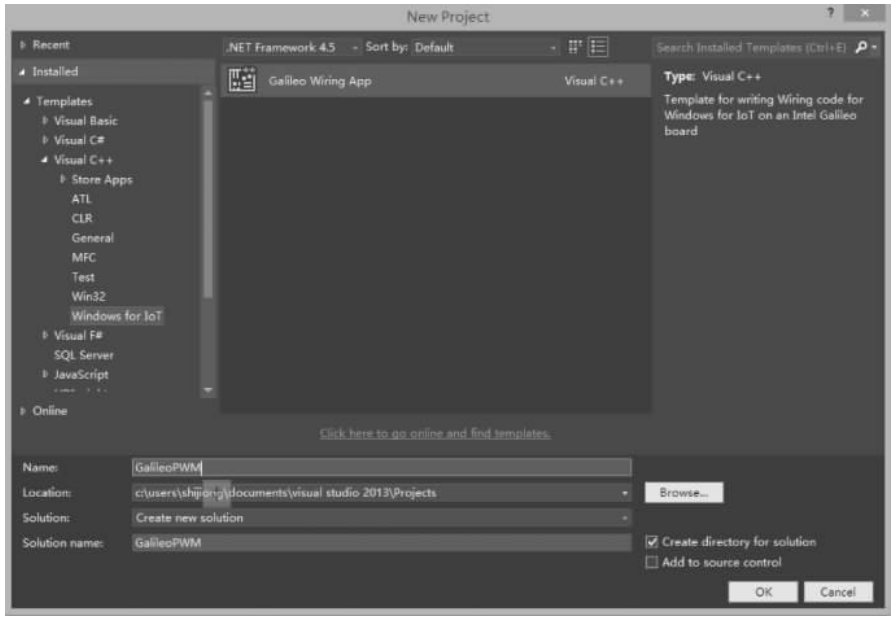


图 3-17 新建工程

打开 Main.cpp 文件，修改 main 函数如下。

代码清单 3-4：PWM 波的输出

Main.cpp 文件主要代码

```
-------------------------------------------------------------------------------
#include "stdafx.h"
#include "arduino.h"

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    return RunArduinoSketch();
}

int pwmpin = 11;                    //定义数字接口 11(PWM 输出)
int val = 0;

void setup()
{
    pinMode(pwmpin, OUTPUT);        //定义数字接口 11 为输出
}

void loop()
{
    for (val = 0; val < 255; val++)
    {
        Log(L"val: %d\r\n", val);   //输出调试信息
        analogWrite(pwmpin, val);   //打开 LED 并设置亮度(PWM 输出最大值为 255)
        delay(10);                  //延时 0.01 秒
    }
}
```

上述程序中,通过 int pwmpin = 11 声明了数字接口 11 为 PWM 输出接口,然后在 setup 函数中通过 pinMode 方法将其设置为输出模式,最后在 loop 函数中,通过 analogWrite 方法来输出 PWM 波形。需要注意的是,该方法的第二个参数的最大值是 255。

3.4.4 部署与调试

代码编写完成以后,按照 3.1.4 节描述的方法进行部署与调试。如果硬件连接正确,就会看到结果,即 LED 灯一直重复逐渐由暗变亮的过程,每个周期约为 2.5 秒左右,如图 3-18 所示。

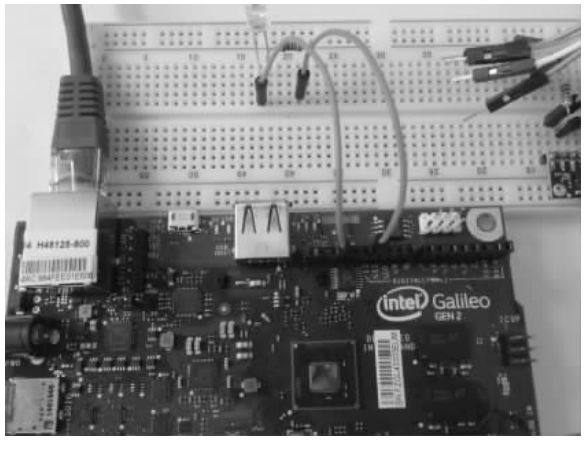


图 3-18 程序运行的实物图

3.5 串口通信

3.5.1 实例功能

串口通信操作简单灵活，在嵌入式设备的调试过程中经常用到。本实例将在 3.4 节的基础上，结合串口，演示如何在 Galileo 上使用 HardwareSerial 进行串口的读写操作，即通过串口接收数据，控制连接的 LED 灯亮度，同时通过串口回复设置成功信息。

3.5.2 硬件电路

本例程需要的元器件包括 LED 灯一个、330 欧姆电阻一个、USB 转 TTL UART 模块一个、面包板一块、杜邦线若干。关于 USB 转 TTL UART 模块，选择一款 PC 机能够安装驱动、可以顺利通信即可，市场上常见的包括使用 CP、FTDI、CH 等系列主芯片均可以使用。其电路原理图如图 3-19 所示。

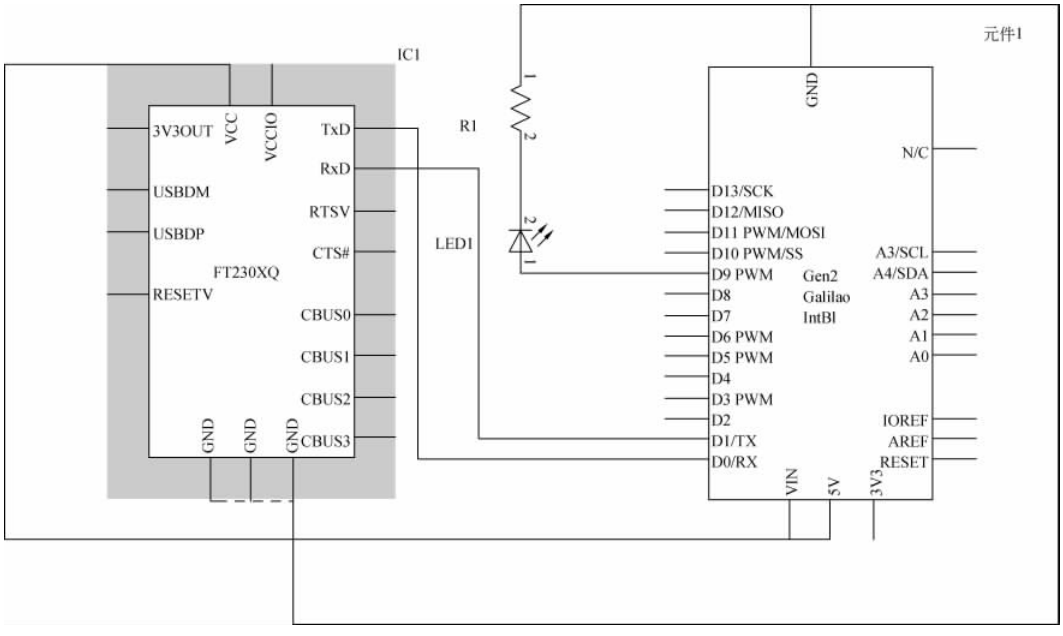


图 3-19 原理图

对应的元器件连接如图 3-20 所示。

需要注意的是，USB 转 TTL UART 模块一般只使用 4 个引脚，即 VCC、GND、Tx 和 Rx。其中，VCC 接 Galileo 的 5V 电源，GND 接 Galileo 的 GND，Tx 接 Galileo 的 RxD，Rx 接 Galileo 的 TxD，即 USB 转 TTL UART 模块的数据收发正好和 Galileo UART 的数据收发进行交叉，如图 3-21 所示。

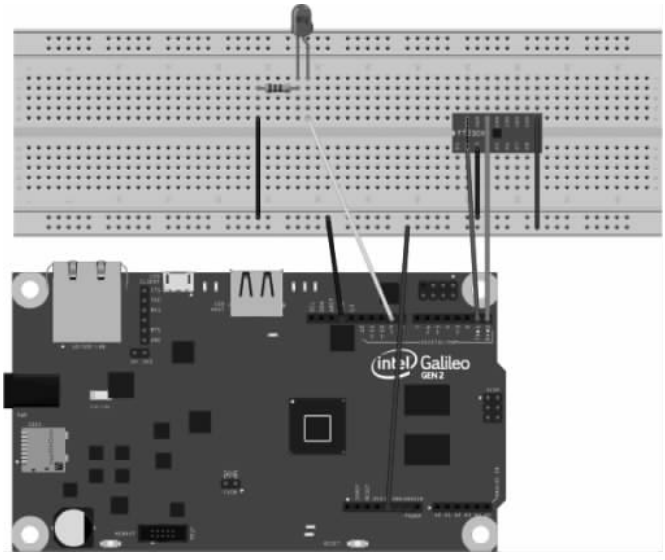


图 3-20 元器件连接图



图 3-21 USB转 UART 模块和 Galileo 的 UART 需要数据收发交叉

3.5.3 程序设计

新建工程，选择模板（即，展开“Templates→Visual C++→Windows for IoT”节点，选中Galileo Wiring app），将默认的工程名修改为 GalileoUART。

打开 Main.cpp 文件，修改 main 函数如下。

代码清单 3-5：串口通信

Main.cpp 文件主要代码

```
#include "stdafx.h"
#include "arduino.h"

int _tmain(int argc, _TCHAR* argv[])
{
    return RunArduinoSketch();
}

int ledpin = 9;
void setup()
{
    Serial.begin(CBR_9600, Serial.SERIAL_8N1);
    pinMode(ledpin,OUTPUT);
}

void loop()
{
    int brightness;
    //查询串口是否收到数据
    if (Serial.available())
    {
        //获取数据
        brightness = Serial.read();
        //设置 LED 亮度
        analogWrite(ledpin,brightness);
        //回复串口信息
        Serial.write("Set LED OK");
    }
}
```

上述程序中，设置数字 9 引脚为控制 LED 的 PWM 波输出引脚，在 setup 函数中，通过 Serial.begin 方法初始化串口，其中的两个参数分别表示 9600 波特率和 8 位数据位、1 位停止位，同时，将 LED 引脚设置为输出模式。在 loop 函数中，不断查询串口是否有数据收到，如果有数据就接收，放在 brightness 变量中，然后使用 analogWrite 设置 PWM 波形的占空比，达到调节 LED 灯亮度的目的，最后，使用 Serial.write 方法回复串口数据，提示 LED 设置成功。需要注意的是，用户发送的串口数据不能大于 255，因为 analogWrite 第二个参数的最大值就是 255。

3.5.4　部署与调试

代码编写完成以后，按照 3.1.4 节描述的方法进行部署与调试。如果硬件连接正确，就可以在 PC 上打开串口调试工具。需要注意的是，在设备管理器中查询的 USB－Serial 模块后面的 COM 号需要和串口调试工具中打开的串口号一致，如图 3-22 所示。



图 3-22　查看并设置 USB 转 UART 模块串口号

可以在串口调试助手的发送数据框中输入需要发送的数据。注意，数据范围为 0～255，且发送的方式选择 HEX 发送，接收采用字符串方式显示，如图 3-23 所示，发送十六进制数 F1 以后，Galileo 的数字引脚 9 变亮，同时在串口调试助手的接收界面收到了数据“Set LED OK”。

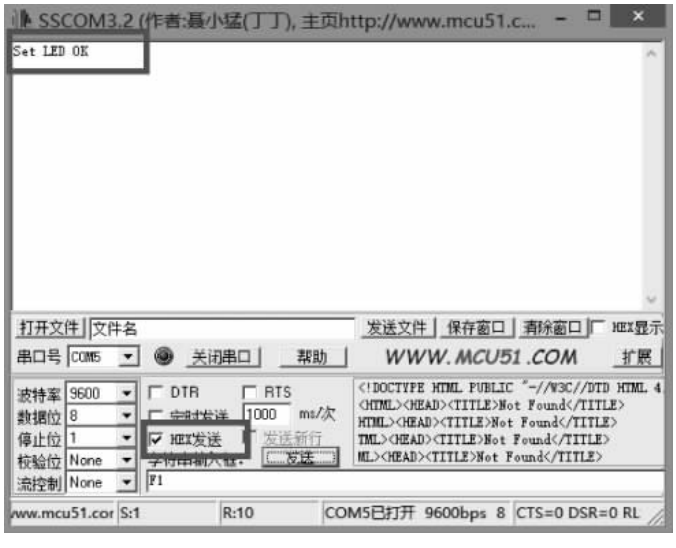


图 3-23　串口调试助手运行界面

实物运行图如图 3-24 所示。

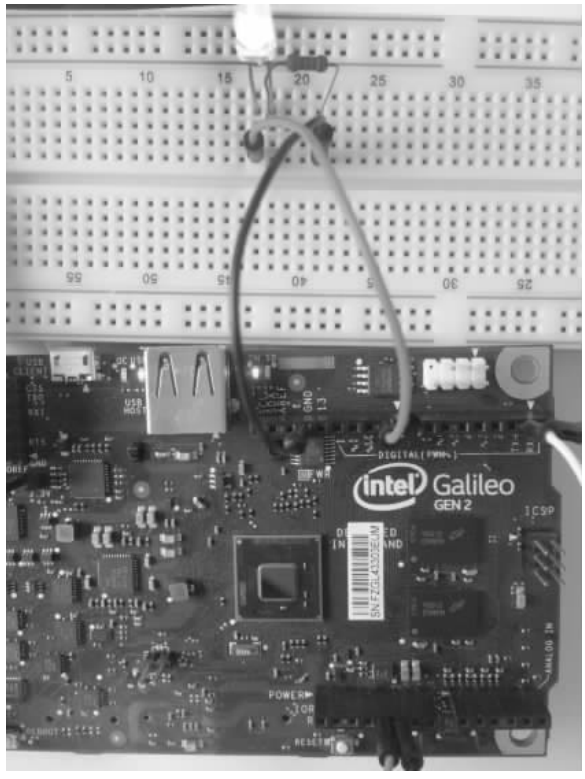


图 3-24　实物运行图

3.6　动手练习

1. 在 3.1 节和 3.2 节内容的基础上，制作一个简单的抢答器，具体要求为：加入两个按钮和两个不同颜色的 LED 灯，如果程序检测到任何一个按钮为低电平，就点亮该按钮对应的 LED，持续时间为 3 秒，3 秒后熄灭，并且通过 Log()方法打印信息，提示是哪个按钮按下。

2. 在 3.1 节和 3.3 节内容的基础上，制作一个简单的温度指示灯，具体要求为：加入一个 LM35 温度传感器和一个 LED 灯，程序中不断检测 LM35 的温度，当温度超过或者低于设定的阈值时，点亮 LED，并且通过 Log()方法打印信息，提示温度已经超过或者低于某个阈值。

3. 在动手练习 1 和动手练习 2 的基础上，结合本章 3.5 节串口通信的内容，将原来程序中通过 Log()方法打印的提示信息改为通过串口输出，设定串口参数为：9600 波特率、8 位数据位、1 位停止位。