MOUSER PSOC 6-WIFI-BT L0：借助PSOC® 6 MCU和WICED® WI-FI/BLUETOOTH技术，打造低功耗云互联物联网设备

BY ALAN KARNS , KAUSHIK, CYPRESS 4823 WIFI-BT, MOUSER 10-24-19 PSOC 6 + 43439 WIFI/BLE, PSOC 6, WICED , 22 OCT 2019

摘要

大家好！这里对应Cypress CY8CKIT-062-WiFi-BT开发套件创建后续系列教程的Session 0（本系列共8个课程）。市场都希望缩短设备互为"智能"PSOC® 6 MCU和WICED® Wi- Fi/BLUETOOTH技术，打造低功耗云互联物联网设备"，全不加思考，但你很快你就了不得我们的。

今天，我将带你从零起步完成系统基础，为大家介绍我工作期间的问题和方，我最后将通过我的最终目的是希望能够为大家提供编程下载的资料和代码，将以可视最终一致不对的情况，请理解如何能够有效利用组，可以直接进行调试的，也可以直接利用我的，我将有效地使用我的功能，我希望工作在其他，也不要，可以直接使用我，这样，我将工作的目标是有效的利用，最后，我会介绍我的复杂。

现在开始整理我要实习那些问题上我的进度，但对我学习，请放心，您还可以按照网络上的资料或技术共享操作。

今天的虚拟网络研讨会我们主要探索，每个课程都有一个目的要点，将会连接到学习和实践。

借助PSOC® 6 MCU和WICED® WI- FI/BLUETOOTH技术，打造低功耗云互联物联网设备

#	课程
0	简介
1	开发者资源
2	WICED Studio与CapSense
3	使用CY8CKIT-028-TFT扩展板
4	视频游戏
5	GoBle与Bluetooth
6	为您的支持GoBle Bluetooth功能
7	实现WiFi和AWS功能
8	为您的支持WiFi和AWS功能

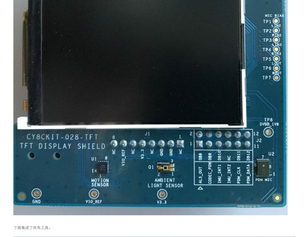
后续我们将探讨，您需要准备：

- CY8CKIT-062-WiFi-BT
- WICED Studio 6.2
- CySmart与LightBlue – BLE应用程序
- GoBle - iOS远程控制应用程序
- Amazon AWS IoT帐户
- 将来将使用的WiFi接入点

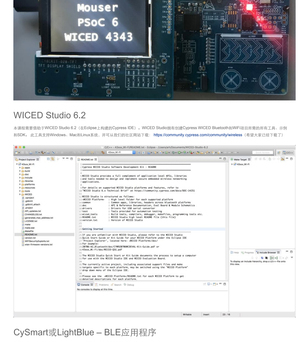
CY8CKIT-062-WiFi-BT

本系统调整的用作项目是新推出的CY8CKIT-062-WiFi-BT评估套件，您可通过[资源专区]来购买此开发套件，套件内容有：内容：

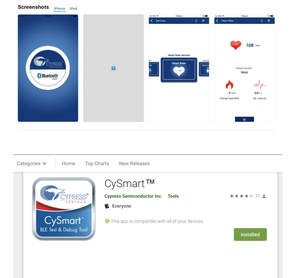
- PSoC 6 + 用于物联网的超低功耗，超安全MCU
- CYW4343W – 802.11 WiFi Bluetooth Combo芯片组
- S25FL512SAGMFB011 – Cypress 512Mb的超级NVSS
- CCG2 – 更便宜高效的Cypress Type C控制器



CY8CKIT-028-TFT是一个具有5777966显示控制器的全彩色240 TFT显示屏。



下面集成了所有工具。

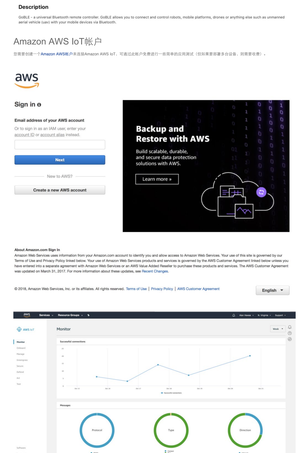


WICED Studio 6.2

本课程需要我的WICED Studio 6.2（在线Wiwi上称为Cypress IDE）。WICED Studio现在包含Cypress WICED Bluetooth和WiFi嵌入应用程序的所有工具，合并到SDK。此工具支持Windows、MacOS,Linux系统，并可以在我们的论坛找到下载：**https://community.cypress.com/community/wireless**（希望大家已经下载了）

CySmart或LightBlue – BLE应用程序

CySmart是一个BLE APP工具包，可从 **iOS应用商店 (Google Play)**免费下载。

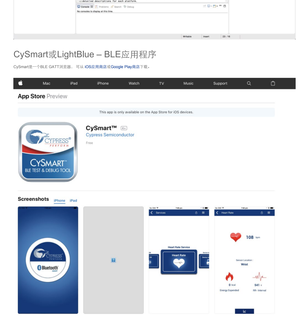


GoBle - iOS远程控制应用程序

GoBle是一款开源入应程序和工具程序，可从**iOS应用商店**下载。

Amazon AWS IoT帐户

您需要创建一个 **Amazon AWS账户**并连接Amazon AWS IoT，可通过此帐户查看进行一些简单的应用测试（但如果要部署多台设备，则需要收费）。



搭载PSOC® 6 MCU和INFINEON® WI- FI或BLUETOOTH®技术，打造低功耗云连接物联网设备

摘要
本指南为 INFINEON 和 CYPRESS 的 WI-FI 和 BLUETOOTH 产品提供了资源概述。

PSOC® 6 产品页面

产品页面网址为 www.cypress.com/psoc6

Cypress 的 PSoC® 6 平台是为 IoT 而构建的，具有以下特点：

- 双核 ARM® Cortex® MCU
- 超低功耗
- 灵活
- 安全

32-bit ARM® Cortex® 6th PSoC® 6

Cypress 的 PSoC® 6 MCU 平台是为 IoT 而构建的，具有以下特点：

- 双核 ARM® Cortex® MCU
- 超低功耗
- 灵活
- 安全

WiFi + Bluetooth Combo产品

PSOC 6 文档

产品页面网址为 www.cypress.com/psoc6

Cypress 的 PSoC® 6 MCU 平台是为 IoT 而构建的，具有以下特点：

- 双核 ARM® Cortex® MCU
- 超低功耗
- 灵活
- 安全

PSOC 6 社区

社区网址为 https://community.cypress.com/psoc-6

Wireless WiFi + Bluetooth Combo芯片

CY8CKIT-062-WIFI-BT开发套件和Web页面

CY8CKIT-062-WIFI-BT开发套件指南

产品页面网址为 www.cypress.com/CY8CKIT-062-WIFI-BT

Doc. # 002-32677 Rev. *B

PSoC 6套接卡册

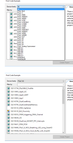
产品页面网址为 www.cypress.com/CY8CKIT-062

CYW43340WiFi芯片手册

PSoC 63技术参考手册

PSoC 62应用笔记

WiFi + Bluetooth应用笔记



PSOC 6代码示例

视频

PSOC 6知识库

外设驱动库文档 (Doxygen)

MOUSER PSOC 6-WIFI-BT L2：WICED STUDIO与CAPSENSE

BY ALAN HAWSE 4343W, BLUETOOTH, CY8CKIT-062-WIFI-BT, MOUSER 10-24-18 PSOC 6 & 4343W WIFI/BLE, PSOC 6, WICED, WIFI 24 OCT 2018

借助PSOC® 6 MCU和WICED® WI- FI/BLUETOOTH技术，打造低功耗云互联物联网设备

#	课程
0	简介
1	开发者资源
2	WICED Studio与CapSense
3	使用CY8CKIT-028-TFT扩展板
4	视频游戏
5	GoBle与Bluetooth
6	为游戏添加GoBle Bluetooth功能
7	实现WiFi和AWS功能
8	为游戏添加WiFi和AWS功能

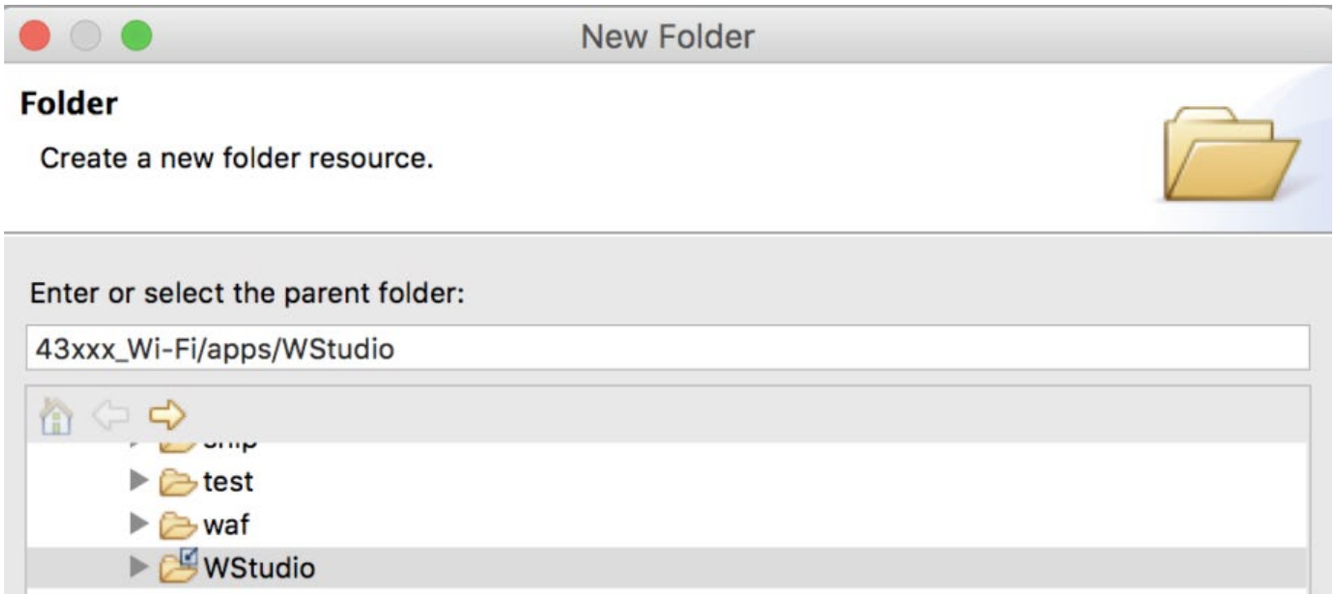
摘要

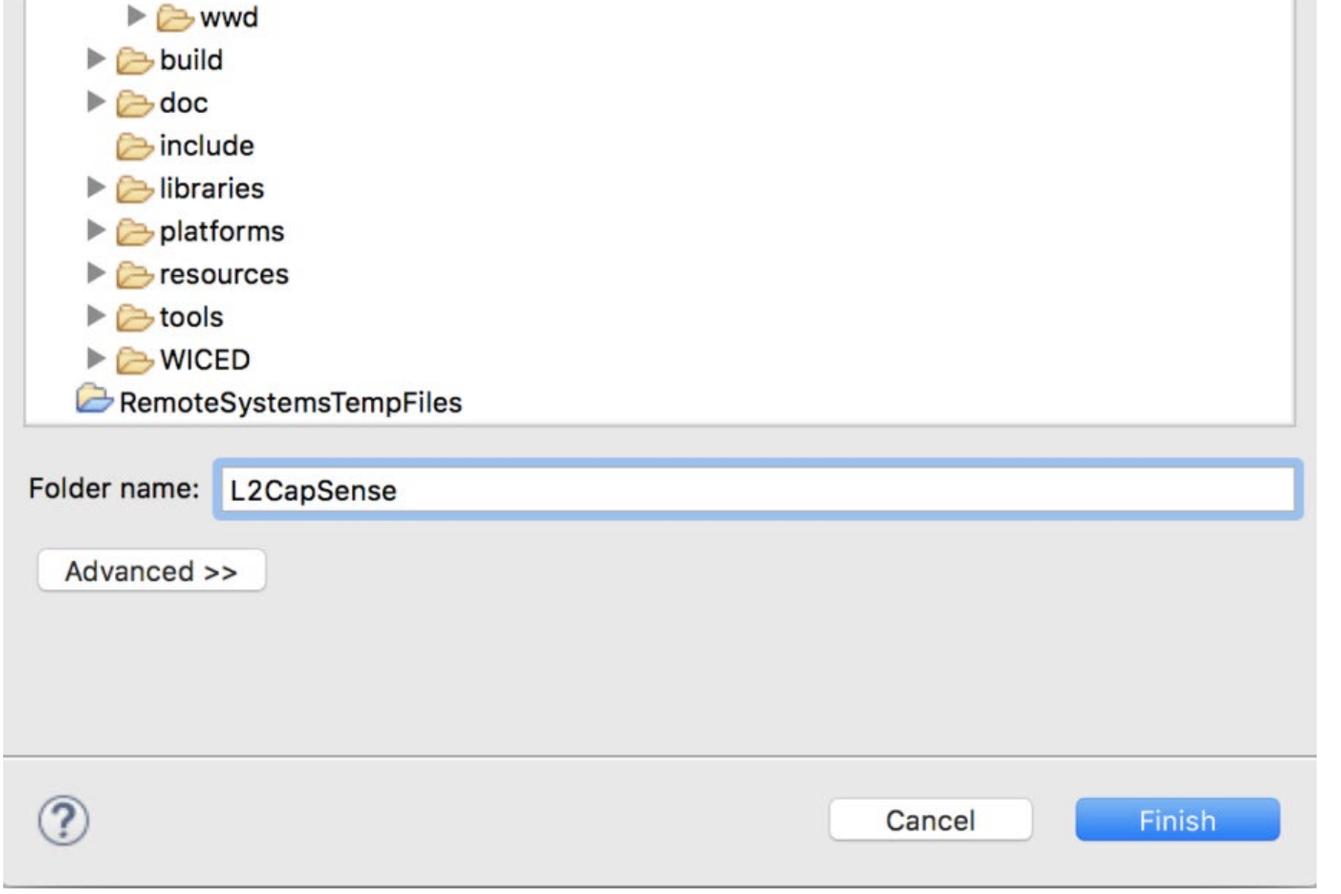
在本课程中，我们将一起创建第一个WICED Studio项目（ blinking LED），并确保您可以对开发套件进行编程。随后将更新项目使之包含一个用于管理CapSense块的线程，此线程将会用于其他项目。

要完成本课程，需要执行以下步骤：

1. 启动WICED Studio 6.2
2. 选择43xxx
3. 创建一个名为L2CapSense的文件夹
4. 创建main.c和blinking LED线程
5. 创建L2CapSense.mk
6. 创建一个make target
7. 编译并测试程序
8. 创建CapSenseThread.c
9. 创建CapSenseThread.h
10. 更新main.c
11. 更新makefile
12. 编译并测试程序

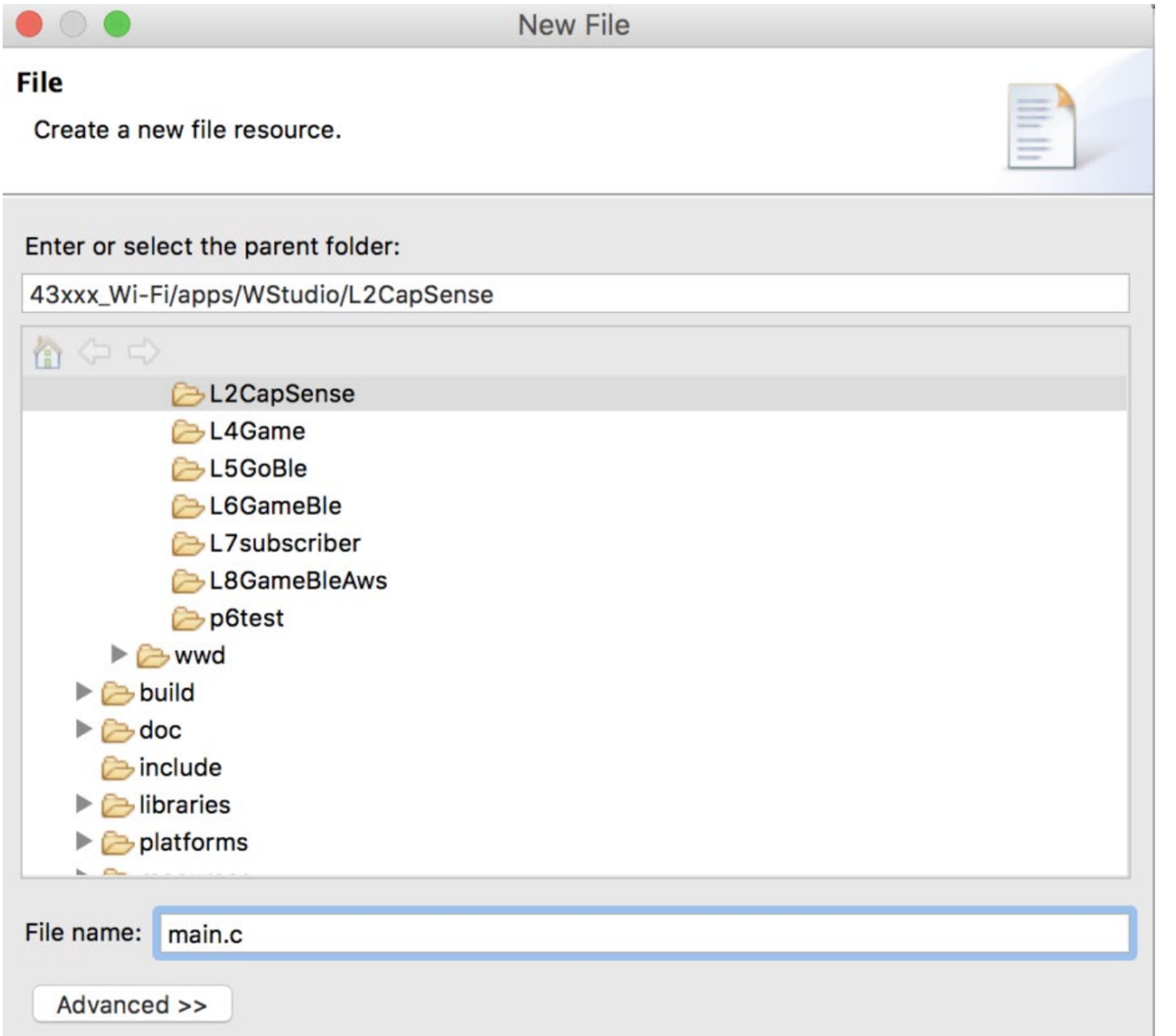
创建L2CapSense文件夹

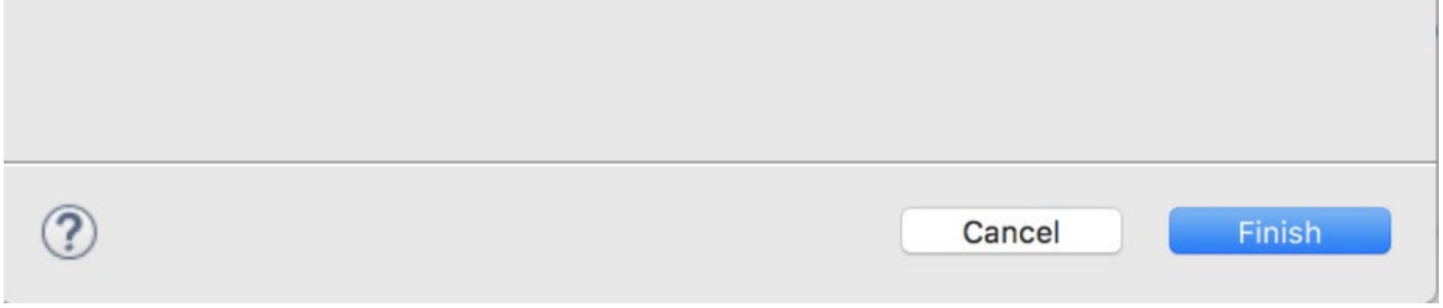




创建main.c

右击文件夹，新建一个名为L2CapSense的文件。



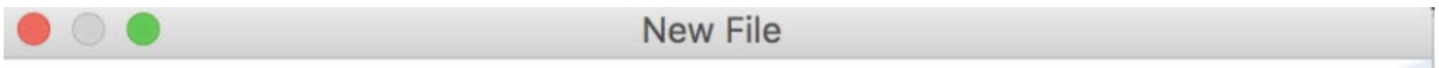


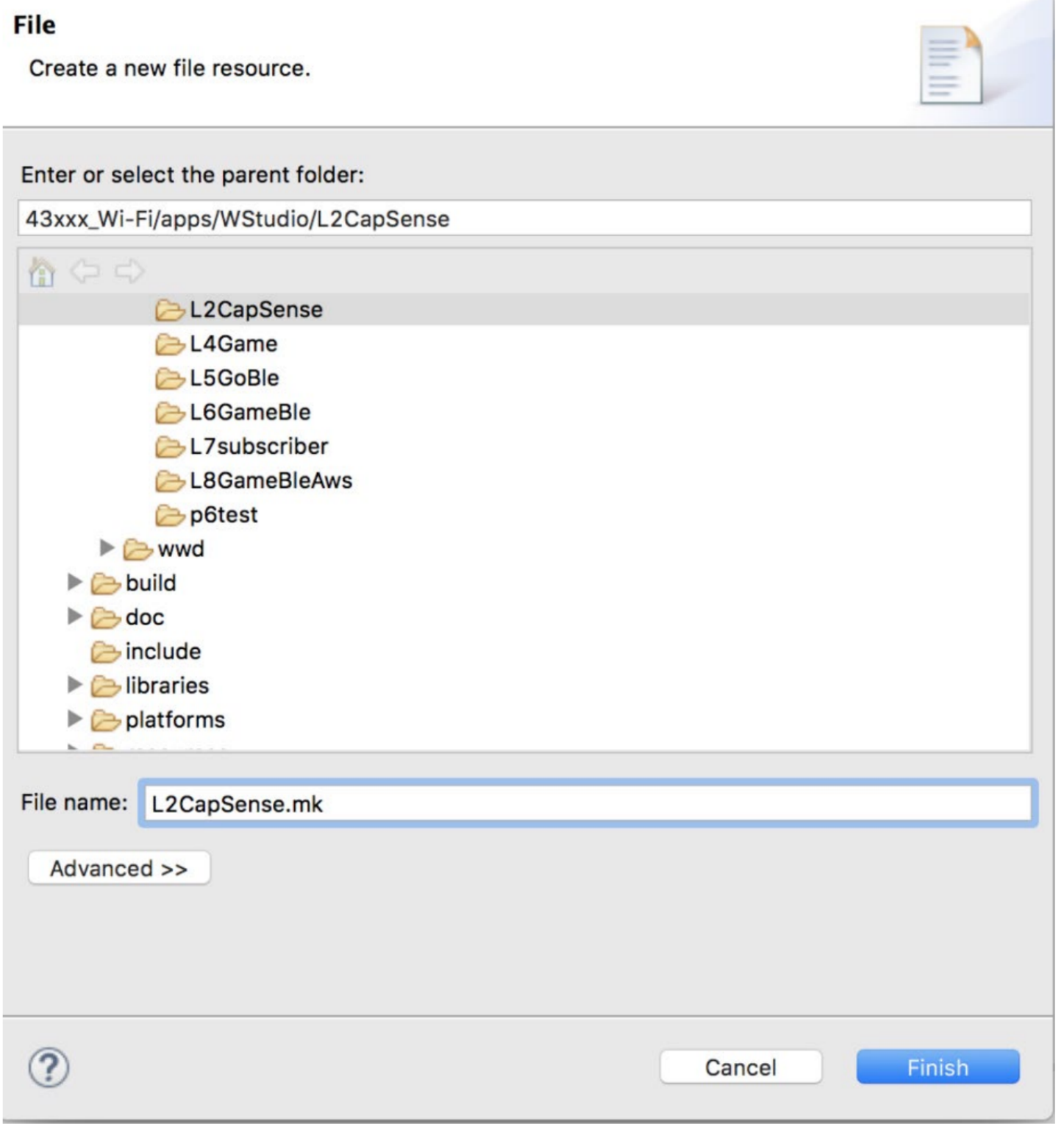
在main.c中插入blinking LED代码。

```
#include "wiced.h"

/*****************************************************
*                    Macros
*****************************************************/

/*****************************************************
*                  Constants
*****************************************************/

/*****************************************************
*                 Enumerations
*****************************************************/

/*****************************************************
*               Type Definitions
*****************************************************/

/*****************************************************
*                  Structures
*****************************************************/


/*****************************************************
*            Static Function Declarations
*****************************************************/


/*****************************************************
*            Variable Definitions
*****************************************************/

wiced_thread_t blinkThreadHandle;

/*****************************************************
*            Function Definitions
*****************************************************/
void pdlBlinkThread(wiced_thread_arg_t arg)
{
    while(1)
    {
        Cy_GPIO_Inv(GPIO_PRT0,3);
        wiced_rtos_delay_milliseconds(500);
    }
}

void application_start( )
{
   WPRINT_APP_INFO(("Started Application\n"));
   wiced_rtos_create_thread(&blinkThreadHandle,7,"Blink Thread",pdlBlinkThread,500,0);
}
```

创建L2CapSense.mk

创建一个名为L2CapSense.mk的makefile

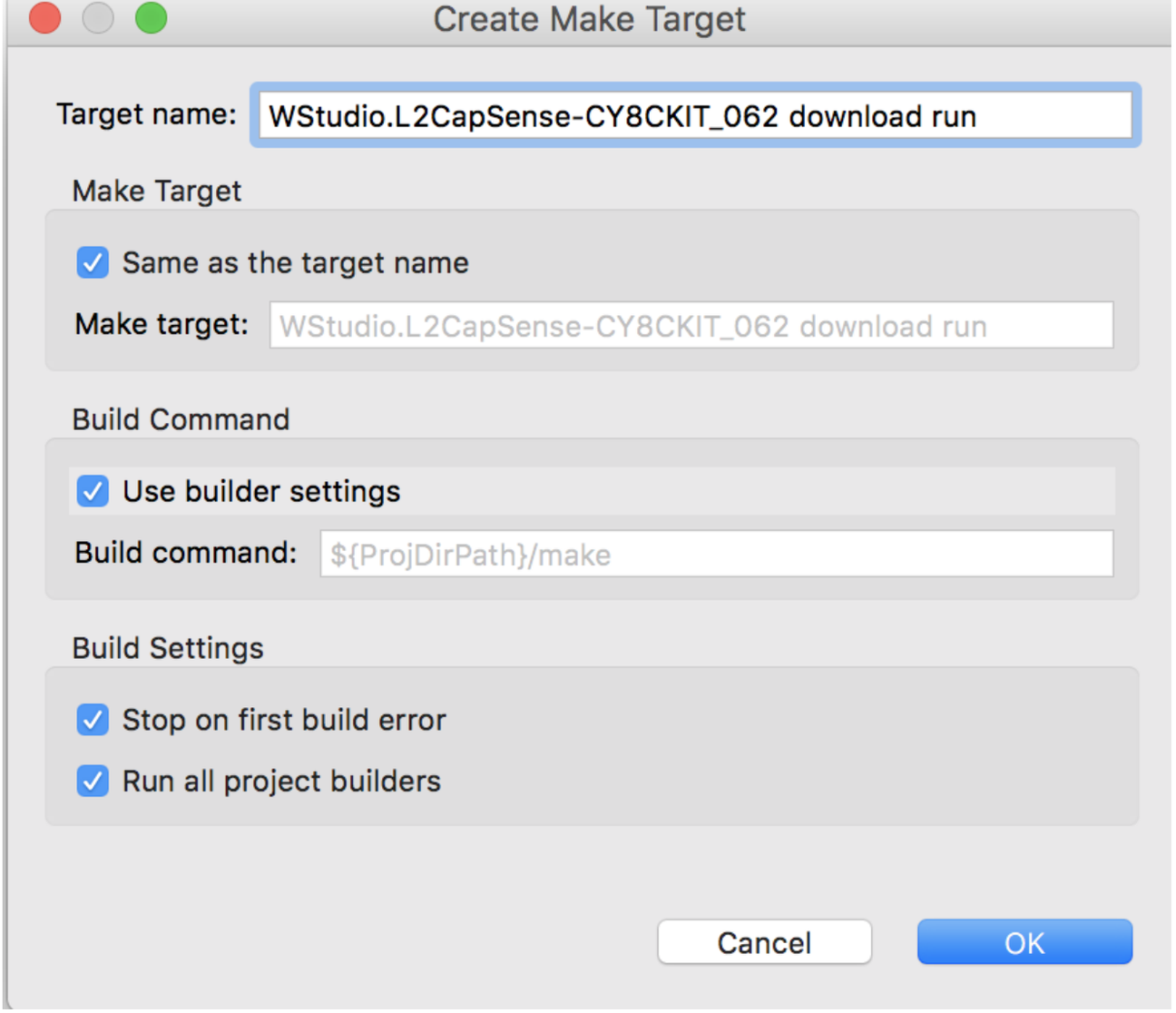




将编译信息添加到L2CapSense.mk中

```
NAME := App_WStudio_L2CapSense

$(NAME)_SOURCES := main.c
```

创建一个Make Target来运行项目



编译并测试Blinking LED

创建/编辑CapSenseThread.c文件

```
#include "wiced.h"

void capSenseThread(wiced_thread_arg_t arg)
{

    CapSense_Start();
    CapSense_ScanAllWidgets();
    while(1)
    {
        if(!CapSense_IsBusy())
        {

            CapSense_ProcessAllWidgets();
            if(CapSense_IsWidgetActive(CapSense_BUTTON0_WDGT_ID))
            {
                WPRINT_APP_INFO(("Button 0 Active\n"));
            }

            if(CapSense_IsWidgetActive(CapSense_BUTTON1_WDGT_ID))
            {
                WPRINT_APP_INFO(("Button 1 Active\n"));
            }

            uint32_t val = CapSense_GetCentroidPos(CapSense_LINEARSLIDER0_WDGT_ID);
            if(val < 0xFFFF)
            {
                WPRINT_APP_INFO(("Slider = %d\n",(int)val));

            }
            CapSense_ScanAllWidgets();
        }
        wiced_rtos_delay_milliseconds(25); // Poll every 25ms (actual scan time ~8ms)
    }
}
```

创建/编辑CapSenseThread.h文件

```
#pragma once
#include "wiced.h"
void capSenseThread(wiced_thread_arg_t arg);
```

更新main.c

```
#include "wiced.h"
#include "CapSenseThread.h"
```

在main.c顶部添加一个变量来存储capSenseThread的句柄

```
wiced_thread_t capSenseThreadHandle;
```

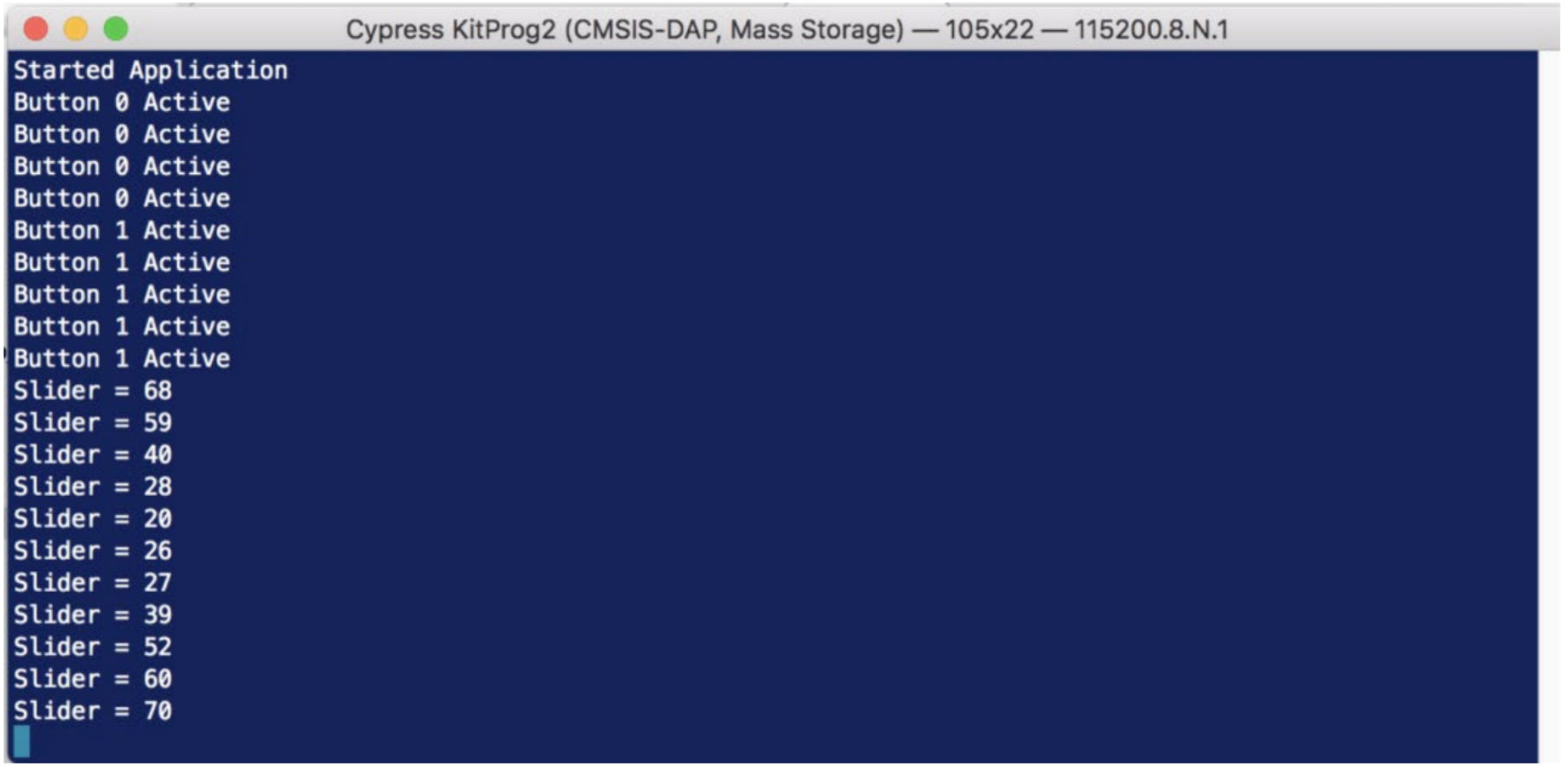
更新main函数以启动CapSenseThread

```
void application_start( )
{
    WPRINT_APP_INFO(("Started Application\n"));
    wiced_rtos_create_thread(&blinkThreadHandle,7,"Blink Thread",pdlBlinkThread,500,0);
    wiced_rtos_create_thread(&capSenseThreadHandle,7,"CapSense Thread",capSenseThread,1024,0);

}
```

更新L2CapsenseThread.mk

```
NAME := App_WStudio_L2CapSense

$(NAME)_SOURCES := main.c \
CapSenseThread.c
```

编译、编程并测试CapSenseThread



MOUSER PSOC 6-WIFI-BT L3：使用CY8CKIT-028-TFT扩展板

BY ALAN HAWSE ▪ 4343W, BLUETOOTH, CY8CKIT-062-WIFI-BT, MOUSER 10-24-18 PSOC 6 & 4343W WIFI/BLE, PSOC 6, WICED, WIFI ▪ 24 OCT 2018

借助PSOC® 6 MCU和WICED® WI- FI/BLUETOOTH技术，打造低功耗云互联物联网设备

#	课程
0	简介
1	开发者资源
2	WICED Studio与CapSense
3	使用CY8CKIT-028-TFT扩展板
4	视频游戏
5	GoBle与Bluetooth
6	为游戏添加GoBle Bluetooth功能
7	实现WiFi和AWS功能
8	为游戏添加WiFi和AWS功能

摘要

本课程将开始编写游戏。首先需要一个显示屏，这次我们选择了CY8CKIT-028-TFT。为了能与显示屏通信，还需要在WICED中创建一个ugui库。此库需要一个驱动配置来复制我们提供的代码示例。最后，将会构建一个“GameThread”线程来实现游戏。

1. 下载CE222494_PSoC6_WICED_WiFi
2. 将L2CapSense复制到L3CapSenseTft中
3. 将cy_tft_display.c/h复制到项目中
4. 创建一个GameThread.h文件
5. 创建一个 GameThread.c文件
6. 将L2CapSense.mk重命名为L3CapSenseTft.mk并进行修改
7. 更新main.c
8. 测试

下载CE222494_PSoC6_WICED_WiFi

单击CY8CKIT-062-WiFi-BT页面，就会看到多个与本开发套件相关的文件，包括CY8CKIT-062-WiFi-BT PSoC® 6 WiFi-BT Pioneer Kit Code Examples.zip。

Related Files

File Title	Language	Size	Last Updated
CY8CKIT-062-WiFi-BT PSoC 6 WiFi-BT Pioneer Kit Guide.pdf	English	13.57 MB	05/07/2018
CY8CKIT-062-WiFi-BT Schematic.pdf	English	3.28 MB	05/03/2018
CY8CKIT-028-TFT Schematic.pdf	English	657.59 KB	05/03/2018
CY8CKIT-062-WiFi-BT PSoC® 6 WiFi-BT Pioneer Kit Hardware.zip	English	13.25 MB	04/03/2018
CY8CKIT-062-WiFi-BT PSoC® 6 WiFi-BT Pioneer Kit Code Examples.zip	English	2.63 MB	04/03/2018
CY8CKIT-062-WiFi-BT PSoC® 6 WiFi-BT Pioneer Kit Release Notes.pdf	English	178.51 KB	04/03/2018

Download CY8CKIT-062-WiFi-BT Kit ISO (Create CD)	English	1.19 GB	03/15/2018
Download CY8CKIT-062-WiFi-BT Kit Only Package (Kit Design Files, Documentation, Examples)	English	33.1 MB	03/15/2018
Download CY8CKIT-062-WiFi-BT Kit Complete Setup (Kit Design Files, PSoC Creator, PSoC Programmer, Documentation, Examples)	English	568.65 MB	03/15/2018
CY8CKIT-062-WiFi-BT PSoC® 6 WiFi-BT Pioneer Kit Quick Start Guide.pdf	English	8.65 MB	02/01/2018

下载此文件夹，并将目录复制到WICED Studio Apps/WStudio文件夹下。

Name	Date Modified
PSoC 6 MCU	Apr 3, 2018 at 2:03 PM
CE222221	Apr 3, 2018 at 2:03 PM
CE222494_PSoC6_WICED_WiFi_Demo	Apr 3, 2018 at 2:03 PM
Hex Files	Apr 3, 2018 at 2:03 PM
ReadMe.txt	Dec 25, 2017 at 7:33 PM

此时文件夹将如下所示：

- 43xxx_Wi-Fi
  - apps
    - demo
    - P6Breakout
    - snip
    - test
    - waf
    - WStudio
      - CE222494_PSoC6_WICED_WiFi
        - CE222494_PSoC6_WICED_WiFi.c
        - CE222494_PSoC6_WICED_WiFi.h
        - CE222494_PSoC6_WICED_WiFi.mk
        - cy_tft_display.c
        - cy_tft_display.h

将L3CapSense复制到L3CapSenseTft中

现在需要将L2CapSense项目复制/粘帖到一个名为L3CapSenseTft的新项目中。

将cy_tft_display.c/h复制到项目中

打开CE222494代码示例目录，将uqui库的驱动程序cy_tft_display.c和cy_tft_display.c复制并粘帖到新项目L3CapSenseTft中。

创建一个GameThread.h文件

创建一个名为GamThread.h的新文件，并定义GameThread，稍后会在main.c中调用以运行游戏线程。

```
#pragma once
#include "wiced.h"

void gameThread(wiced_thread_arg_t arg);
```

创建一个 GameThread.c文件

现在创建一个GameThread.c文件，其中共包含5个函数。下面提供了整个文件便于进行复制/粘帖。请放心，我会逐一介绍每一个函数。

```
#include "GameThread.h"

#include "cy_tft_display.h"

#define SCREEN_X (320)
#define SCREEN_Y (240)

static UG_GUI   gui;


// ARH Function to put text in the center of a point (UG_PutString does upper left)
static void UG_PutStringCenter(uint32_t x, uint32_t y, uint32_t fontx, uint32_t fonty,char *string)
{
    y = y - fonty/2;
    x = x - (strlen(string)/2)*fontx;
    if(strlen(string)%2)
        x = x - fontx/2;
    UG_PutString(x,y,string);
}


// Display the splash screen
static void displaySplashScreen()
{
    UG_FontSelect( &FONT_22X36 );
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5,22,36,"Cypress");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5*2,22,36,"Mouser");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5*3,22,36,"PSoC 6");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5*4,22,36,"WICED 4343");

    wiced_rtos_delay_milliseconds(2000);
}

// This function displays the start button message
static void displayStartButton()
{
    UG_FontSelect(&FONT_12X20);
    UG_PutStringCenter(SCREEN_X/2 , SCREEN_Y - 30 ,12,22,  "Press B0 To Start");
}


// Display the Start Screen
static void  displayStartScreen()
{
    UG_FillScreen( C_BLACK );
    UG_FontSelect( &FONT_22X36 );
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/2 -2 - 18 ,22,36,"Ready");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/2 + 2 + 18 ,22,36,"Player 1");
    displayStartButton();
}

// Main game thread
void gameThread(wiced_thread_arg_t arg)
{

    Cy_TFT_Init();                                  // Init the TFT
    UG_Init( &gui, Cy_TFT_displayDriver, SCREEN_X, SCREEN_Y ); // Connect the driver

    UG_FillScreen( C_BLACK );   // Clear the screen
    UG_SetBackcolor( C_BLACK );
    UG_SetForecolor( C_WHITE );

    displaySplashScreen();
    displayStartScreen();

    while(1)
    {
        wiced_rtos_delay_milliseconds(1000);
    }
}
```

主游戏线程函数是void gameThread(wiced_thread_arg_t arg)，它能够执行以下操作：

1. 初始化TFT
2. 初始化UGUI库
3. 清空屏幕（设置为全黑）
4. 设置颜色以进行黑底白绘
5. 显示初始屏幕（用时2s）
6. 显示启动屏幕
7. 等待结束

```
// Main game thread
void gameThread(wiced_thread_arg_t arg)
{

    Cy_TFT_Init();                                  // Init the TFT
    UG_Init( &gui, Cy_TFT_displayDriver, SCREEN_X, SCREEN_Y ); // Connect the driver

    UG_FillScreen( C_BLACK );   // Clear the screen
    UG_SetBackcolor( C_BLACK );
    UG_SetForecolor( C_WHITE );

    displaySplashScreen();
    displayStartScreen();

    while(1)
    {
        wiced_rtos_delay_milliseconds(1000);
    }
}
```

displaySplashScreen函数简单的设置了字体，绘制了4个文本字符串，等待数秒后继续。

```
// Display the splash screen
static void displaySplashScreen()
{
    UG_FontSelect( &FONT_22X36 );
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5,22,36,"Cypress");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5*2,22,36,"Mouser");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5*3,22,36,"PSoC 6");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5*4,22,36,"WICED 4343");

    wiced_rtos_delay_milliseconds(2000);
}
```

在屏幕上显示“Ready Player 1”，然后提示用户按下B0开始游戏。

```
// This function displays the start button message
static void displayStartButton()
{
    UG_FontSelect(&FONT_12X20);
    UG_PutStringCenter(SCREEN_X/2 , SCREEN_Y - 30 ,12,22,  "Press B0 To Start");
}


// Display the Start Screen
static void  displayStartScreen()
{
    UG_FillScreen( C_BLACK );
    UG_FontSelect( &FONT_22X36 );
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/2 -2 - 18 ,22,36,"Ready");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/2 + 2 + 18 ,22,36,"Player 1");
    displayStartButton();
}
```

U8G_PutString函数使用x和y坐标设置文本左上角位置。但出于格式化考虑，更容易让我想到字符串的中间位置。此函数会根据中间位置坐标 (x,y) 计算左上角坐标 (x,y)。此外，还需要知道字体的 (x,y) 坐标。

static void UG_PutStringCenter(uint32_t x, uint32_t y, uint32_t fontx, uint32_t fonty,char *string)

```
// ARH Function to put text in the center of a point (UG_PutString does upper left)
static void UG_PutStringCenter(uint32_t x, uint32_t y, uint32_t fontx, uint32_t fonty,char *string)
{
    y = y - fonty/2;
    x = x - (strlen(string)/2)*fontx;
    if(strlen(string)%2)
        x = x - fontx/2;
    UG_PutString(x,y,string);
}
```

将L2CapSense.mk重命名为L3CapSenseTft.mk并进行修改

要进行编译，需要修改makefile以了解新线程以及tft驱动程序。此外，还需要让链接器链接图形库。

```
NAME := App_WStudio_L3CapSenseTft

$(NAME)_SOURCES := main.c \
CapSenseThread.c \
GameThread.c \
cy_tft_display.c

$(NAME)_COMPONENTS := graphics/ugui
```

更新main.c

需要对main.c执行以下操作：

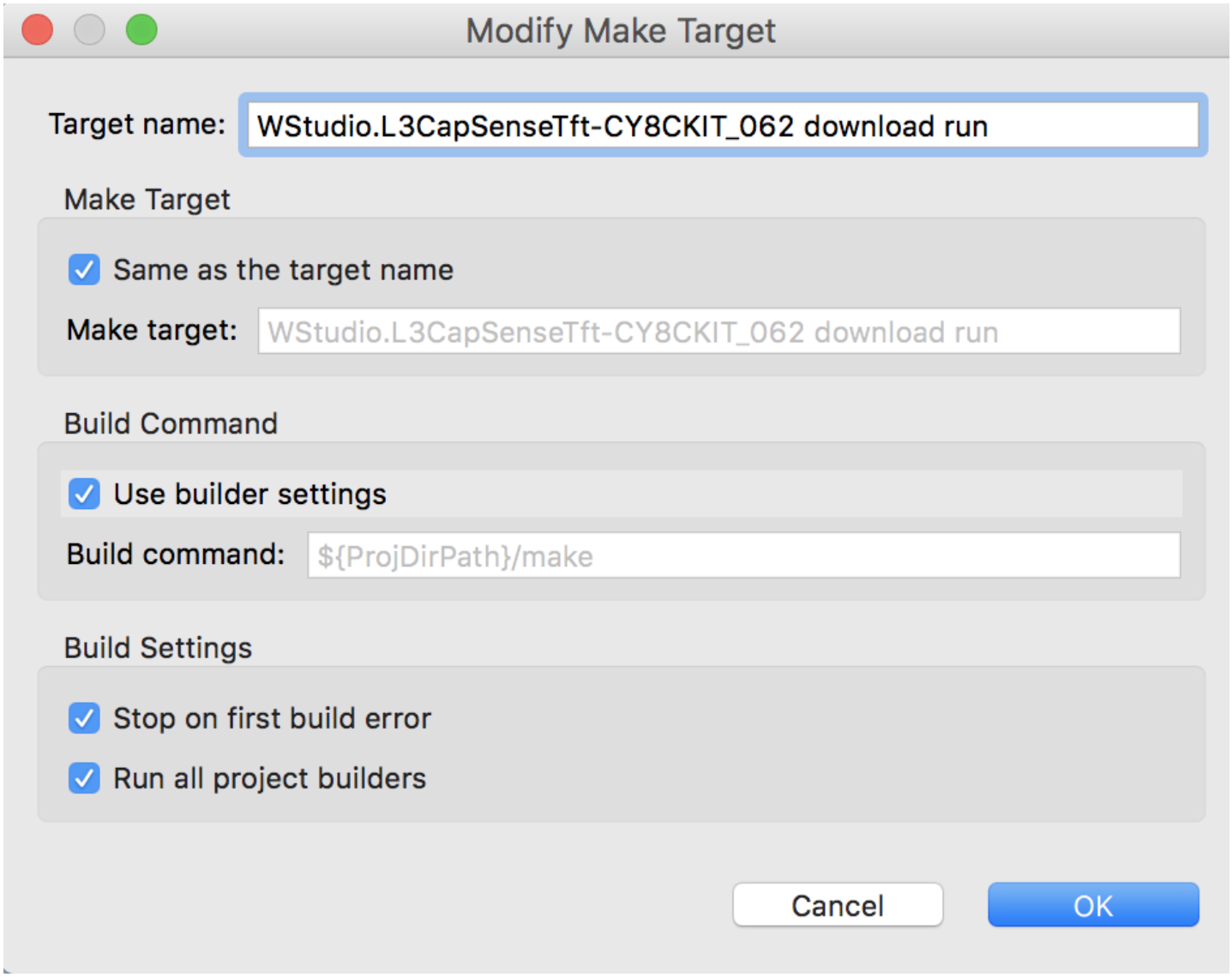
1. 嵌入GameThread.h
2. 设置一个变量来存放gameThreadHandle
3. 启动gameThread

```
#include "wiced.h"
#include "CapSenseThread.h"
#include "GameThread.h"

/******************************************************
*                      Macros
******************************************************/

/******************************************************
*                    Constants
******************************************************/

/******************************************************
*                   Enumerations
******************************************************/

/******************************************************
*                 Type Definitions
******************************************************/

/******************************************************
*                    Structures
******************************************************/


/******************************************************
*               Static Function Declarations
******************************************************/


/******************************************************
*               Variable Definitions
******************************************************/

wiced_thread_t blinkThreadHandle;
wiced_thread_t capSenseThreadHandle;
wiced_thread_t gameThreadHandle;



/******************************************************
*               Function Definitions
******************************************************/
```

```
void pdlBlinkThread(wiced_thread_arg_t arg)
{
    while(1)
    {
        Cy_GPIO_Inv(GPIO_PRT0,3);
        wiced_rtos_delay_milliseconds(500);
    }
}

void application_start( )
{
    wiced_init();
    WPRINT_APP_INFO(("Started Application\n"));
    wiced_rtos_create_thread(&blinkThreadHandle,7,"Blink Thread",pdlBlinkThread,500,0);
    wiced_rtos_create_thread(&capSenseThreadHandle,7,"CapSense Thread",capSenseThread,1024,0);
    wiced_rtos_create_thread(&gameThreadHandle,7,"game Thread",gameThread,4096,0);


}
```

测试

现在可以进行测试了。在此之前需要创建一个Make Target，然后进行编译，编程。一切正常意味着我们成功创建了Ready Player 1。



MOUSER PSOC6-WIFI-BT L4：视频游戏

BY ALAN HAWSE ▪ 4343W, BLUETOOTH, CY8CKIT-062-WIFI-BT, MOUSER 10-24-18 PSOC 6 & 4343W WIFI/BLE, PSOC 6, WICED, WIFI ▪ 24 OCT 2018

借助PSOC® 6 MCU和WICED® WI- FI/BLUETOOTH技术，打造低功耗云互联物联网设备

#	课程
0	简介
1	开发者资源
2	WICED Studio与CapSense
3	使用CY8CKIT-028-TFT扩展板
4	视频游戏
5	GoBle与Bluetooth
6	为游戏添加GoBle Bluetooth功能
7	实现WiFi和AWS功能
8	为游戏添加WiFi和AWS功能

摘要

在本课程中，将通过添加图形等播放游戏功能来完成视频游戏线程。此外，还将完善CapSense线程，使之通过RTOS队列与游戏关联起来。

本游戏有三个主要功能：

1. 用于游戏屏幕（初始屏幕、开始屏幕、运行屏幕以及游戏结束屏幕）的状态机
2. 20ms定时器，在游戏运行时刷新屏幕（移动手柄和球）
3. GUI队列， 便于CapSense、Bluetooth和WiFi等其他系统功能可以发送按钮和手柄消息。

要完成本课程，需要执行以下步骤：

1. 复制L3CapSenseTft来设置项目和makefile
2. 更新gamethread.h以定义GUI队列消息
3. 修改main.c来创建队列
4. 创建SystemGlobal.h让文件的其他部分可以访问gui队列
5. 更新CapSenseThread以发送GUI消息
6. 更新GameThread.c中的includes语句
7. 添加几个#define宏来定义游戏参数
8. 为游戏添加状态机并定义手柄移动方法
9. 对线程函数进行前向声明
10. 设置一些用于维持游戏状态的变量
11. 添加得分和速度显示函数
12. 添加游戏启动和结束函数
13. 添加辅助函数来计算手柄的顶部和底部位置
14. 添加函数来更新并绘制手柄运动路线
15. 添加函数来更新并绘制球移动路线
16. 添加游戏计时器调用函数
17. 更新主游戏线程

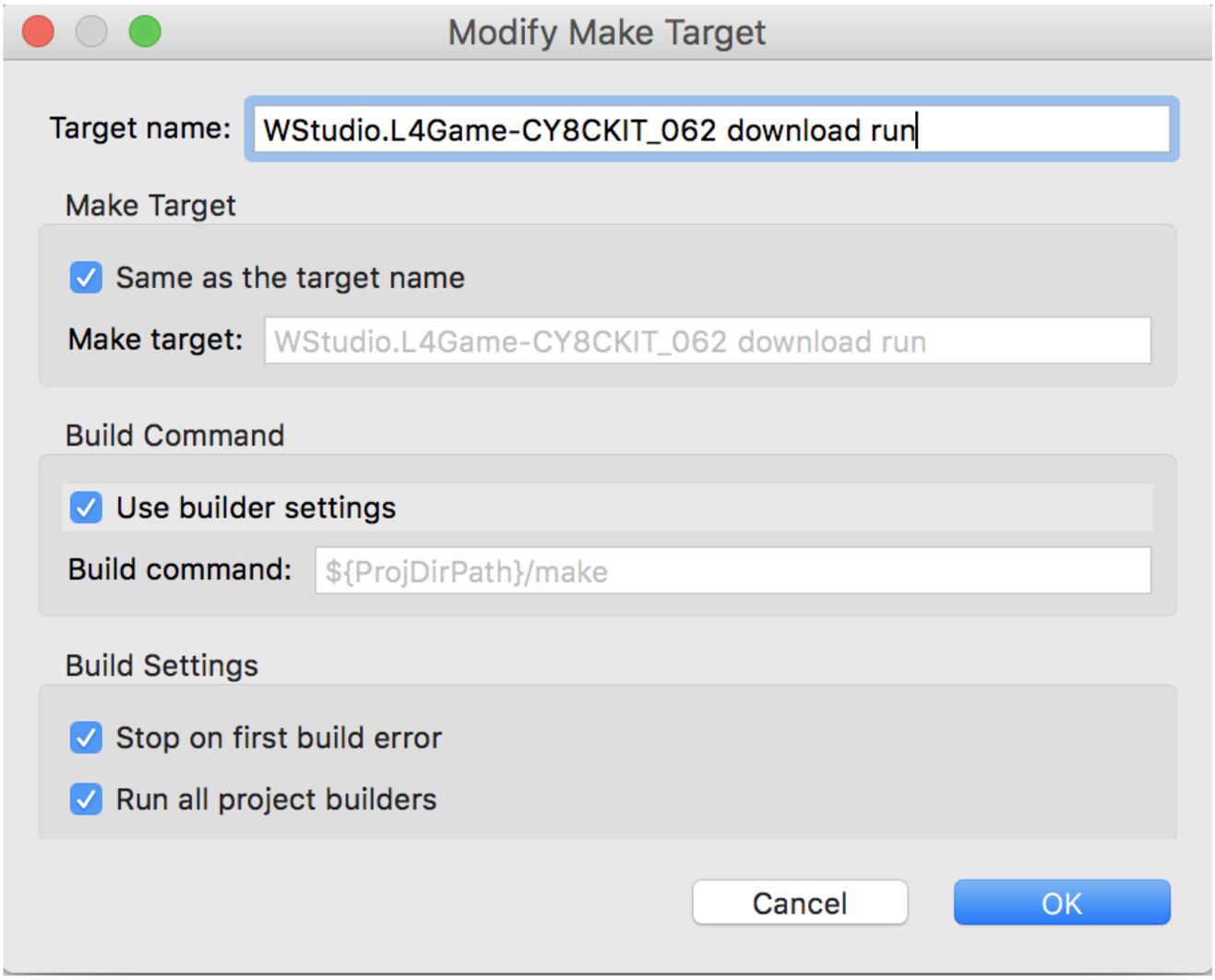
复制L3CapSenseTft来设置项目和makefile

将L3CapSenseTft项目复制/粘帖到L4Game新文件夹中。将makefile重命名为L4Game.mk。

编辑makefile并更改应用程序名。

```
NAME := App_WStudio_L4Game

$(NAME)_SOURCES := main.c \
    CapSenseThread.c \
GameThread.c \
cy_tft_display.c

$(NAME)_COMPONENTS := graphics/ugui
```

为项目创建make target



更新GameThread.h以定义GUI消息

系统中的所有线程（CapSense、Bluetooth和WiFi）都要通过向RTOS队列发送消息来控制手柄和按钮，并且还将在Thread.h中对消息进行定义。此消息由两个值组成 – GUI元素和发送的值。

```
#pragma once
#include "wiced.h"

typedef enum {
    MSG_POSITION,
    MSG_BUTTON0,
    MSG_BUTTON1,
} game_evt_t;

typedef struct {
    game_evt_t evt;
    uint32_t val;
} game_msg_t;

void gameThread(wiced_thread_arg_t arg);
```

修改main.c来创建队列

通常来说RTOS原型应该在main.c中。因此需要编辑main.c并修改includes语句。

```
#include "GameThread.h"
#include "wiced.h"
#include "CapSenseThread.h"
#include "SystemGlobal.h"
```

接下来定义“paddleQueue”队列变量，最初是想命名为“guiQueue”，但现在修改已经来不及了。

```
/*****************************************************
*              Variable Definitions
*****************************************************/

wiced_thread_t blinkThreadHandle;
wiced_thread_t capSenseThreadHandle;
wiced_thread_t gameThreadHandle;
wiced_queue_t paddleQueue;
```

设置队列

```
void application_start( )
{
    wiced_init( );
    wiced_rtos_init_queue(&paddleQueue,"paddleQueue",sizeof(game_msg_t),10);
    wiced_rtos_create_thread(&blinkThreadHandle,7,"Blink Thread",pdlBlinkThread,500,0);
    wiced_rtos_create_thread(&capSenseThreadHandle,7,"CapSense Thread",capSenseThread,1024,0);
    wiced_rtos_create_thread(&gameThreadHandle,7,"game Thread",gameThread,4096,0);
}
```

创建SystemGlobal.h

系统中的所有线程都需要访问paddleQueue变量。为此，需要创建一个SystemGlobal.h文件，并将此变量设置为全局变量以便进行访问。

```
#pragma once

extern wiced_queue_t paddleQueue;
```

更新CapSenseThread以发送GUI消息

之前设置的CapSenseThread只能输出值。 现在要对它进行修改以便输出消息。首先要编辑CapSenseThread.c，

1. 添加消息变量 (line 8)，
2. 修改button0和button 1以生成并发送RTOS消息（lines 20/21和line 26/27）
3. 修改滑块来发送位置消息（lines 33-35）

```
1   #include "wiced.h"
2   #include "GameThread.h"
3   #include "SystemGlobal.h"
4
5   void capSenseThread(wiced_thread_arg_t arg)
6   {
7
8       game_msg_t msg;
9
10      CapSense_Start();
11      CapSense_ScanAllWidgets();
12      while(1)
13      {
14          if(!CapSense_IsBusy())
15          {
16
```

```
            CapSense_ProcessAllWidgets();
            if(CapSense_IsWidgetActive(CapSense_BUTTON0_WDGT_ID))
            {
                msg.evt = MSG_BUTTON0;
                wiced_rtos_push_to_queue(&paddleQueue,&msg,0);
            }

            if(CapSense_IsWidgetActive(CapSense_BUTTON1_WDGT_ID))
            {
                msg.evt = MSG_BUTTON1;
                wiced_rtos_push_to_queue(&paddleQueue,&msg,0);
            }

            uint32_t val = CapSense_GetCentroidPos(CapSense_LINEARSLIDER0_WDGT_ID);
            if(val < 0xFFFF)
            {
                msg.evt = MSG_POSITION;
                msg.val = val;
                wiced_rtos_push_to_queue(&paddleQueue,&msg,0);
            }
            CapSense_ScanAllWidgets();
        }
        wiced_rtos_delay_milliseconds(25); // Poll every 25ms (actual scan time ~8ms)
    }
}
```

更新GameThread.c中的includes语句

现在来修改GameThread.c。需要编辑include语句来添加一个“SystemGlobal.h” 文件，其中包含了GUI队列的全局变量。

```
#include "GameThread.h"
#include "cy_tft_display.h"
#include "SystemGlobal.h"
#include "ugui.h"
```

在GameThread.c中添加#define 宏

在此使用#define宏来定义游戏中需要用到常量。

```
#define UPDATE_SCREEN_TIME (20) // Update the screen every 20ms
#define SPEED (2)
#define SCREEN_X (320)
#define SCREEN_Y (240)
#define TOP_FIELD (21)
#define PD_WIDTH (10)
#define PD_LEN (70)
#define DOTS (3)
#define PADDLE0_COLOR (C_BLUE)
#define BALL_COLOR (C_GREEN)
#define BALL_SIZE (10)
```

为游戏添加状态机并定义手柄的移动

打开GameThread.c – 游戏的所有控制功能都包含在这个文件中。

游戏将有四个屏幕：显示Cypress和Mouser的初始屏幕、Ready Player 1屏、游戏屏和游戏结束屏。

此外，手柄能进行微移（增量式）或直接跳转到相应位置（绝对式）。

```
// States of the game
typedef enum {
    GS_SPLASH,
    GS_START,
    GS_RUNNING,
    GS_OVER
} game_state_t;

// Methods to move the paddle
typedef enum {
    PADDLE_INCREMENT,
    PADDLE_ABSOLUTE
} paddle_update_t;
```

设置gameState状态机

在显示初始屏幕时，需要将状态机设置为GS_SPLASH。

```
static void displaySplashScreen()
{
    gameState = GS_SPLASH;
    UG_FontSelect( &FONT_22X36 );
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5,22,36,"Cypress");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5*2,22,36,"Mouser");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5*3,22,36,"PSoC 6");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5*4,22,36,"WICED 4343");

    wiced_rtos_delay_milliseconds(2000);
}
```

在显示启动屏幕时，需要将状态机设置为GS_START。

```
// Display the Start Screen
static void  displayStartScreen()
{
    gameState = GS_START;
    UG_FillScreen( C_BLACK );
    UG_FontSelect( &FONT_22X36 );
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/2 -2 - 18 ,22,36,"Ready");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/2 + 2 + 18 ,22,36,"Player 1");
    displayStartButton();
}
```

对函数进行前向声明

在使用函数前应先进行定义

```
/*******************************************************
*              Static Function Declarations
*******************************************************/
static void UG_PutStringCenter(uint32_t x, uint32_t y, uint32_t fontx, uint32_t fonty,char *string);
static void displaySplashScreen();
static void displayStartButton();
static void  displayStartScreen();
static void displayScore();
static void displaySpeed();
static void endGame();
static inline uint32_t calcPaddleTop();
static inline uint32_t calcPaddleBottom();
static void updatePaddle(paddle_update_t type);
static void updateBall();
static void updateScreen(void *arg);
```

设置一些用于维持游戏状态的变量

使用updateScreenTimer可实现每隔20ms调用一次updateScreen。变量调用结果也非常直观。

```
/*******************************************************
*              Variable Definitions
*******************************************************/
static UG_GUI   gui;
static wiced_timer_t updateScreenTimer;

static uint32_t gameScore;
static game_state_t gameState;

// position of the paddle
static uint32_t paddle0_desire_pos=0;
static uint32_t paddle0_cur_pos=0;

// Position, direction and speed of the ball
static uint32_t ballx,bally;
static int32_t ballXdir, ballYdir;
static uint32_t ballSpeed;
```

添加得分和速度显示函数

这两个函数将会在屏幕顶部显示速度与得分。

```
// This function displays the score
static void displayScore()
{
    char buff[10];
    sprintf(buff,"%2X",(unsigned int)gameScore);
    UG_FontSelect(&FONT_12X20);
    UG_PutString( 75, 0, buff);
}

// This function displays the speed
static void displaySpeed()
{
    char buff[10];
    sprintf(buff,"%2X",(unsigned int)ballSpeed-1);
    UG_FontSelect(&FONT_12X20);
    UG_PutString( 275, 0, buff);
}
```

添加启动游戏函数

如需启动游戏，请执行以下步骤：：

1. 重置分数
2. 设置手柄位置
3. 将球移动到手柄中间位置
4. 将球移动到右下角
5. 清空屏幕，显示得分与速度
6. 开始游戏

```
// This function initializes everything and starts a new game
static void startGame()
{
    gameScore = 0;

    paddle0_desire_pos = 50; // start the game with the paddle moving
    paddle0_cur_pos = 0;

    ballx = PD_WIDTH ;                  // start the ball on the paddle on the right of the screen
    bally  = calcPaddleTop() + PD_LEN/2; // start the ball in the middle of the paddle

    ballSpeed = SPEED;

    ballXdir = ballSpeed;
    ballYdir = ballSpeed;

    UG_FillScreen( C_BLACK );  // clear screen
    UG_FontSelect(&FONT_12X20);
    UG_PutString( 0, 0,  "Score:");
    displayScore();
    UG_PutString(200,0,"Speed:");
    displaySpeed();
    UG_DrawLine(0,20,SCREEN_X,20,C_RED); // red line under text to represent top of play screen

    gameState = GS_RUNNING;
    wiced_rtos_start_timer(&updateScreenTimer); // Timer to update screen
}
```

添加结束游戏函数

结束游戏时，请执行以下操作：

1. 将游戏状态切换为结束
2. 停止计时
3. 显示游戏结束
4. 提示按button 0以开始游戏

```
// Stop the game
static void endGame()
{
    gameState = GS_OVER;
    wiced_rtos_stop_timer(&updateScreenTimer);
    UG_FontSelect( &FONT_22X36 );
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/2,22,36,"Game Over");
    displayStartButton();
}
```

添加Helper函数来计算手柄顶部和底部位置

需要知道两个位置才能确定手柄的位置，尤其是以下情况下：

1. 绘制手柄移动路线
2. 判断球是否碰到手柄

以下两个函数用于根据手柄的当前位置计算其顶部和底部的像素位置

```
// Figure out the y position of the top of the paddle
static inline uint32_t calcPaddleTop()
{
    return (paddle0_cur_pos)*DOTS+TOP_FIELD;
}

// Figure out the y position of the bottom of the paddle
static inline uint32_t calcPaddleBottom()
{
    return (paddle0_cur_pos)*DOTS+PD_LEN+TOP_FIELD;
}
```

添加函数来更新和绘制手柄移动路线

手柄需要在游戏运行时能够移动。有以下两种方法：

1. 绝对式，将手柄从当前位置立即移动到目标位置
2. 增量式，将手柄向目标位置微移。

```
// Move the paddle either to : PADDLE_INCREMENT the next location or PADDLE_ABSOLUTE - final location
static void updatePaddle(paddle_update_t type)
{
    // If the paddle is where it is supposed to be then just return
    if(paddle0_cur_pos == paddle0_desire_pos)
        return;

    // erase the current paddle
    UG_FillFrame(0,calcPaddleTop(),PD_WIDTH,calcPaddleBottom(),C_BLACK);

    switch (type)
    {

    case PADDLE_INCREMENT:

        if(paddle0_cur_pos < paddle0_desire_pos)
            paddle0_cur_pos += SPEED;
        else
            paddle0_cur_pos -= SPEED;

        // If the paddle is within one move of the final spot, put it there
        if(abs((int)paddle0_cur_pos-(int)paddle0_desire_pos) < SPEED)
            paddle0_cur_pos = paddle0_desire_pos;
        break;

```

```
    case PADDLE_ABSOLUTE:
        paddle0_cur_pos = paddle0_desire_pos;
        break;
    }
    // draw the paddle
    UG_FillFrame(0,calcPaddleTop(),PD_WIDTH,calcPaddleBottom(),PADDLE0_COLOR);
}
```

添加函数来更新并绘制球移动路线

需要一个函数来：

1. 移动球
2. 指出球是否运动到屏幕的右侧/左侧/顶部/底部，并执行相应操作

当球碰到屏幕边界时，需要进行增减以改变运动方向。

每碰到手柄一次，分数加1并加速。

如果未碰到手柄，游戏结束。

```
// Move the ball to the next location
static void updateBall()
{
    static const uint32_t BallFudgeFactor=3;

    UG_DrawCircle(ballx,bally,BALL_SIZE,C_BLACK);

    ballx += ballXdir;
    bally += ballYdir;

    // Check to see if the ball hit the far right side
    if(ballx > SCREEN_X - BALL_SIZE)
    {

        ballx = SCREEN_X - BALL_SIZE;
        ballXdir = -ballSpeed;
    }

    // check to see if the ball hit the far left side... or the paddle
    if(ballx < (BALL_SIZE + PD_WIDTH + BallFudgeFactor))
    {
        // See if the ball missed the paddle
        if(bally + BALL_SIZE < calcPaddleTop() || bally - BALL_SIZE > calcPaddleBottom())
        {
            endGame();
            //WPRINT_APP_INFO(("Missed Paddle\r\n"));
        }

        gameScore = gameScore + 1;
        displayScore();
        if(gameScore % 3 == 0) // Speed up every three hits
        {
            ballSpeed +=1;
            displaySpeed();
        }

        ballx = BALL_SIZE + PD_WIDTH + BallFudgeFactor;
        ballXdir = +ballSpeed;
    }
    // Check to see if the ball hit the top or bottom
    if(bally > SCREEN_Y - BALL_SIZE) // bottom
    {
        bally = SCREEN_Y - BALL_SIZE;
        ballYdir = -ballSpeed;
    }

    if(bally < TOP_FIELD+BALL_SIZE) // top
    {
        bally = BALL_SIZE+TOP_FIELD;
        ballYdir = +ballSpeed;
    }
    UG_DrawCircle(ballx,bally,BALL_SIZE,BALL_COLOR);
}
```

创建游戏计时函数

RTOS计时器运行间隔为20ms。此计时器需要一个函数来移动手柄和球。

```
// This function is called every UPADTE_SCREEN_TIME milliseconds by the updateScreenTimer
static void updateScreen(void *arg)
{
    updatePaddle(PADDLE_INCREMENT);
    updateBall();
}
```

更新主游戏线程

此主游戏线程需要获取队列消息并根据游戏状态执行相应操作。

```
// Main game thread
void gameThread(wiced_thread_arg_t arg)
{
    game_msg_t msg;

    Cy_TFT_Init();                                  // Init the TFT
    UG_Init( &gui, Cy_TFT_displayDriver, SCREEN_X, SCREEN_Y ); // Connect the driver

    UG_FillScreen( C_BLACK );   // Clear the screen
    UG_SetBackcolor( C_BLACK );
    UG_SetForecolor( C_WHITE );

    wiced_rtos_init_timer(&updateScreenTimer,UPDATE_SCREEN_TIME,updateScreen,0);
    displaySplashScreen();
    displayStartScreen();

    while(1)
    {
        wiced_rtos_pop_from_queue(&paddleQueue,&msg,WICED_WAIT_FOREVER);
        switch(msg.evt)
        {
        case MSG_POSITION:
            if(gameState == GS_RUNNING)
                paddle0_desire_pos = msg.val/2;
            if(gameState == GS_OVER)
            {
                paddle0_desire_pos = msg.val/2;
                updatePaddle(PADDLE_ABSOLUTE);
            }
            break;

        case MSG_BUTTON0:
            if(gameState == GS_OVER || gameState == GS_START)
                startGame();
            break;
        case MSG_BUTTON1:
            break;
        }
    }
}
```

编程并测试

现在所有函数已完成… 需要进行编程和测试了。

GameThread.c

下面是完整的线程，可以进行复制/粘帖。

```
#include "GameThread.h"
#include "cy_tft_display.h"
#include "SystemGlobal.h"
#include "ugui.h"

/*****************************************************
*                    Macros
*****************************************************/
#define UPDATE_SCREEN_TIME (20) // Update the screen every 20ms
#define SPEED (2)
#define SCREEN_X (320)
#define SCREEN_Y (240)
#define TOP_FIELD (21)
#define PD_WIDTH (10)
#define PD_LEN (70)
#define DOTS (3)
#define PADDLE0_COLOR (C_BLUE)
#define BALL_COLOR (C_GREEN)
#define BALL_SIZE (10)

/*****************************************************
*                   Constants
```

```
*******************************************************/

/*******************************************************
*                    Enumerations
*******************************************************/

/*******************************************************
*                  Type Definitions
*******************************************************/
// States of the game
typedef enum {
    GS_SPLASH,
    GS_START,
    GS_RUNNING,
    GS_OVER
} game_state_t;

// Methods to move the paddle
typedef enum {
    PADDLE_INCREMENT,
    PADDLE_ABSOLUTE
} paddle_update_t;

/*******************************************************
*                     Structures
*******************************************************/


/*******************************************************
*                Static Function Declarations
*******************************************************/
static void UG_PutStringCenter(uint32_t x, uint32_t y, uint32_t fontx, uint32_t fonty,char *string);
static void displaySplashScreen();
static void displayStartButton();
static void  displayStartScreen();
static void displayScore();
static void displaySpeed();
static void endGame();
static inline uint32_t calcPaddleTop();
static inline uint32_t calcPaddleBottom();
static void updatePaddle(paddle_update_t type);
static void updateBall();
static void updateScreen(void *arg);

/*******************************************************
*                Variable Definitions
*******************************************************/
static UG_GUI   gui;
static wiced_timer_t updateScreenTimer;

static uint32_t gameScore;
static game_state_t gameState;

// position of the paddle
static uint32_t paddle0_desire_pos=0;
static uint32_t paddle0_cur_pos=0;

// Position, direction and speed of the ball
static uint32_t ballx,bally;
static int32_t ballXdir, ballYdir;
static uint32_t ballSpeed;


/*******************************************************
*                Functions
*******************************************************/

// ARH Function to put text in the center of a point (UG_PutString does upper left)
static void UG_PutStringCenter(uint32_t x, uint32_t y, uint32_t fontx, uint32_t fonty,char *string)
{
    y = y - fonty/2;
    x = x - (strlen(string)/2)*fontx;
    if(strlen(string)%2)
        x = x - fontx/2;
    UG_PutString(x,y,string);
}


// Display the splash screen
static void displaySplashScreen()
{
    gameState = GS_SPLASH;
    UG_FontSelect( &FONT_22X36 );
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5,22,36,"Cypress");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5*2,22,36,"Mouser");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5*3,22,36,"PSoC 6");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/5*4,22,36,"WICED 4343");

    wiced_rtos_delay_milliseconds(2000);
}

// This function displays the start button message
static void displayStartButton()
{
    UG_FontSelect(&FONT_12X20);
    UG_PutStringCenter(SCREEN_X/2 , SCREEN_Y - 30 ,12,22,  "Press B0 To Start");
}

```

```
// Display the Start Screen
static void  displayStartScreen()
{
    gameState = GS_START;
    UG_FillScreen( C_BLACK );
    UG_FontSelect( &FONT_22X36 );
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/2 -2 - 18 ,22,36,"Ready");
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/2 + 2 + 18 ,22,36,"Player 1");
    displayStartButton();
}


// This function displays the score
static void displayScore()
{
    char buff[10];
    sprintf(buff,"%2X",(unsigned int)gameScore);
    UG_FontSelect(&FONT_12X20);
    UG_PutString( 75, 0, buff);
}

// This function displays the speed
static void displaySpeed()
{
    char buff[10];
    sprintf(buff,"%2X",(unsigned int)ballSpeed-1);
    UG_FontSelect(&FONT_12X20);
    UG_PutString( 275, 0, buff);
}


// This function initializes everything and starts a new game
static void startGame()
{
    gameScore = 0;

    paddle0_desire_pos = 50; // start the game with the paddle moving
    paddle0_cur_pos = 0;

    ballx = PD_WIDTH ;                  // start the ball on the paddle on the right of the screen
    bally  = calcPaddleTop() + PD_LEN/2; // start the ball in the middle of the paddle

    ballSpeed = SPEED;

    ballXdir = ballSpeed;
    ballYdir = ballSpeed;

    UG_FillScreen( C_BLACK );  // clear screen
    UG_FontSelect(&FONT_12X20);
    UG_PutString( 0, 0,  "Score:");
    displayScore();
    UG_PutString(200,0,"Speed:");
    displaySpeed();
    UG_DrawLine(0,20,SCREEN_X,20,C_RED); // red line under text to represent top of play screen

    gameState = GS_RUNNING;
    wiced_rtos_start_timer(&updateScreenTimer); // Timer to update screen
}

// Stop the game
static void endGame()
{
    gameState = GS_OVER;
    wiced_rtos_stop_timer(&updateScreenTimer);
    UG_FontSelect( &FONT_22X36 );
    UG_PutStringCenter(SCREEN_X/2,SCREEN_Y/2,22,36,"Game Over");
    displayStartButton();
}


// Figure out the y position of the top of the paddle
static inline uint32_t calcPaddleTop()
{
    return (paddle0_cur_pos)*DOTS+TOP_FIELD;
}

// Figure out the y position of the bottom of the paddle
static inline uint32_t calcPaddleBottom()
{
    return (paddle0_cur_pos)*DOTS+PD_LEN+TOP_FIELD;
}

// Move the paddle either to : PADDLE_INCREMENT the next location or PADDLE_ABSOLUTE - final location
static void updatePaddle(paddle_update_t type)
{
    // If the paddle is where it is supposed to be then just return
    if(paddle0_cur_pos == paddle0_desire_pos)
        return;

    // erase the current paddle
    UG_FillFrame(0,calcPaddleTop(),PD_WIDTH,calcPaddleBottom(),C_BLACK);

    switch (type)
    {

    case PADDLE_INCREMENT:

        if(paddle0_cur_pos < paddle0_desire_pos)
            paddle0_cur_pos += SPEED;
        else
            paddle0_cur_pos -= SPEED;

```

```
        // If the paddle is within one move of the final spot, put it there
        if(abs((int)paddle0_cur_pos-(int)paddle0_desire_pos) < SPEED)
            paddle0_cur_pos = paddle0_desire_pos;
        break;

    case PADDLE_ABSOLUTE:
        paddle0_cur_pos = paddle0_desire_pos;
        break;
    }
    // draw the paddle
    UG_FillFrame(0,calcPaddleTop(),PD_WIDTH,calcPaddleBottom(),PADDLE0_COLOR);
}

// Move the ball to the next location
static void updateBall()
{
    static const uint32_t BallFudgeFactor=3;

    UG_DrawCircle(ballx,bally,BALL_SIZE,C_BLACK);

    ballx += ballXdir;
    bally += ballYdir;

    // Check to see if the ball hit the far right side
    if(ballx > SCREEN_X - BALL_SIZE)
    {

        ballx = SCREEN_X - BALL_SIZE;
        ballXdir = -ballSpeed;
    }

    // check to see if the ball hit the far left side... or the paddle
    if(ballx < (BALL_SIZE + PD_WIDTH + BallFudgeFactor))
    {
        // See if the ball missed the paddle
        if(bally + BALL_SIZE < calcPaddleTop() || bally - BALL_SIZE > calcPaddleBottom())
        {
            endGame();
            //WPRINT_APP_INFO(("Missed Paddle\r\n"));
        }

        gameScore = gameScore + 1;
        displayScore();
        if(gameScore % 3 == 0) // Speed up every three hits
        {
            ballSpeed +=1;
            displaySpeed();
        }

        ballx = BALL_SIZE + PD_WIDTH + BallFudgeFactor;
        ballXdir = +ballSpeed;
    }
    // Check to see if the ball hit the top or bottom
    if(bally > SCREEN_Y - BALL_SIZE) // bottom
    {
        bally = SCREEN_Y - BALL_SIZE;
        ballYdir = -ballSpeed;
    }

    if(bally < TOP_FIELD+BALL_SIZE) // top
    {
        bally = BALL_SIZE+TOP_FIELD;
        ballYdir = +ballSpeed;
    }
    UG_DrawCircle(ballx,bally,BALL_SIZE,BALL_COLOR);
}

// This function is called every UPADTE_SCREEN_TIME milliseconds by the updateScreenTimer
static void updateScreen(void *arg)
{
    updatePaddle(PADDLE_INCREMENT);
    updateBall();
}


// Main game thread
void gameThread(wiced_thread_arg_t arg)
{
    game_msg_t msg;

    Cy_TFT_Init();                                  // Init the TFT
    UG_Init( &gui, Cy_TFT_displayDriver, SCREEN_X, SCREEN_Y ); // Connect the driver

    UG_FillScreen( C_BLACK );   // Clear the screen
    UG_SetBackcolor( C_BLACK );
    UG_SetForecolor( C_WHITE );

    wiced_rtos_init_timer(&updateScreenTimer,UPDATE_SCREEN_TIME,updateScreen,0);
    displaySplashScreen();
    displayStartScreen();

    while(1)
    {
        wiced_rtos_pop_from_queue(&paddleQueue,&msg,WICED_WAIT_FOREVER);
        switch(msg.evt)
        {
        case MSG_POSITION:
            if(gameState == GS_RUNNING)
                paddle0_desire_pos = msg.val/2;
            if(gameState == GS_OVER)
            {
                paddle0_desire_pos = msg.val/2;
                updatePaddle(PADDLE_ABSOLUTE);
            }
            break;

        case MSG_BUTTON0:
            if(gameState == GS_OVER || gameState == GS_START)
                startGame();
            break;
        case MSG_BUTTON1:
            break;
        }
    }
}
```

BY ALAN HAWSE 4343W, CY8CKIT-062-WIFI-BT, MOUSER 10-24-18 PSOC 6 & 4343W WIFI/BLE, PSOC 6, WICED · 24 OCT 2018

借助PSOC® 6 MCU和WICED® WI- FI/BLUETOOTH技术，打造低功耗云互联物联网设备

摘要

虽然我很喜欢这款视频游戏，但它还缺少了一些功能，也就是现在大热的物联网功能。首先要给这个游戏加一个BLE远程控制功能。为了简化操作，我希望找到一个能从App商店下载的现成远程控制app。在经过一番搜索之后，我找到了GoBle。我也曾经发布过一篇关于GoBle的详细文章，单击 此处即可进行阅读，我也会在本课程中进行详细的解释，让大家了解它如何与本款游戏配合运行。

下面就是一个远程控制截屏。左边是操纵杆，右边是一些按钮，这也是一个典型的远程控制布局。

远程控制作为“中央”单元，知道如何连接外设。在本课程中，我们将启动CYW4343W作为蓝牙外设。

要完成本课程，需要执行以下步骤：

1. 创建“L5GoBle”文件夹
2. 创建L5GoBle.mk makefile
3. 创建GoBle_db.h和GoBle_db.c来设置GATT 数据库
4. 设置wiced_bt_cfg.c
5. 创建GoBleThread.c
6. 创建GoBle.c以启动GoBleThread
7. 编译、编程并测试

创建“L5GoBle”目录

创建L5GoBle.mk makefile

```
NAME := App_WStudio_L5GoBle

$(NAME)_SOURCES    := GoBle.c \
          GoBleThread.c \
                    GoBle_db.c \
                    wiced_bt_cfg.c

$(NAME)_INCLUDES   := .

$(NAME)_COMPONENTS += libraries/drivers/bluetooth/low_energy
```

创建GoBle_db.h和GoBle_db.c来设置GATT 数据库

创建“GoBle_db.h”文件

```
// GoBle_db.h

#ifndef __GATT_DATABASE_H__
#define __GATT_DATABASE_H__

#include "wiced.h"

#define __UUID_GOBLE                    0xfb, 0x34, 0x9b, 0x5f, 0x80, 0x00, 0x00, 0x80, 0x00, 0x10, 0x00, 0x00, 0xb0, 0xdf, 0x00, 0x00
#define __UUID_GOBLE_SERIALPORTID          0xfb, 0x34, 0x9b, 0x5f, 0x80, 0x00, 0x00, 0x80, 0x00, 0x10, 0x00, 0x00, 0xb1, 0xdf,
0x00, 0x00

// ***** Primary Service 'Generic Attribute'
#define HDLS_GENERIC_ATTRIBUTE              0x0001

// ***** Primary Service 'Generic Access'
#define HDLS_GENERIC_ACCESS                 0x0014
// ----- Characteristic 'Device Name'
#define HDLC_GENERIC_ACCESS_DEVICE_NAME           0x0015
#define HDLC_GENERIC_ACCESS_DEVICE_NAME_VALUE       0x0016
// ----- Characteristic 'Appearance'
#define HDLC_GENERIC_ACCESS_APPEARANCE            0x0017
#define HDLC_GENERIC_ACCESS_APPEARANCE_VALUE        0x0018

// ***** Primary Service 'GoBle'
#define HDLS_GOBLE                          0x0028
// ----- Characteristic 'SerialPortId'
#define HDLC_GOBLE_SERIALPORTID                 0x0029
#define HDLC_GOBLE_SERIALPORTID_VALUE             0x002A
// ===== Descriptor 'Client Configuration'
#define HDLD_GOBLE_SERIALPORTID_CLIENT_CONFIGURATION 0x002B


// External definitions
extern const uint8_t  gatt_database[];
extern const uint16_t gatt_database_len;
extern uint8_t BT_LOCAL_NAME[];
extern const uint16_t BT_LOCAL_NAME_CAPACITY;

#endif /* __GATT_DATABASE_H__ */
```

创建“GoBle.c”文件

```
/*
* This file has been automatically generated by the WICED 20719-B1 Designer.
* BLE device's GATT database and device configuration.
*
*/

// GoBle_db.c

#include "GoBle_db.h"

#include "wiced.h"
#include "wiced_bt_uuid.h"
#include "wiced_bt_gatt.h"

/*************************************************************************************
** GATT server definitions
*************************************************************************************/

const uint8_t gatt_database[] = // Define GATT database
{
    /* Primary Service 'Generic Attribute' */
    PRIMARY_SERVICE_UUID16 (HDLS_GENERIC_ATTRIBUTE, UUID_SERVICE_GATT),

    /* Primary Service 'Generic Access' */
    PRIMARY_SERVICE_UUID16 (HDLS_GENERIC_ACCESS, UUID_SERVICE_GAP),

    /* Primary Service 'GoBle' */
    PRIMARY_SERVICE_UUID128 (HDLS_GOBLE, __UUID_GOBLE),

        /* Characteristic 'SerialPortId' */
        CHARACTERISTIC_UUID128_WRITABLE (HDLC_GOBLE_SERIALPORTID, HDLC_GOBLE_SERIALPORTID_VALUE,
            __UUID_GOBLE_SERIALPORTID, LEGATTDB_CHAR_PROP_READ | LEGATTDB_CHAR_PROP_WRITE_NO_RESPONSE |
LEGATTDB_CHAR_PROP_WRITE | LEGATTDB_CHAR_PROP_NOTIFY,
            LEGATTDB_PERM_READABLE | LEGATTDB_PERM_WRITE_CMD | LEGATTDB_PERM_WRITE_REQ),

            /* Descriptor 'Client Characteristic Configuration' */
            CHAR_DESCRIPTOR_UUID16_WRITABLE (HDLD_GOBLE_SERIALPORTID_CLIENT_CONFIGURATION,
                UUID_DESCRIPTOR_CLIENT_CHARACTERISTIC_CONFIGURATION, LEGATTDB_PERM_READABLE |
LEGATTDB_PERM_WRITE_REQ | LEGATTDB_PERM_AUTH_WRITABLE),

};

// Length of the GATT database
const uint16_t gatt_database_len = sizeof(gatt_database);
```

设置wiced_bt_cfg.c

WICED Bluetooth配置文件名为 “wiced_bt_config.c”，我们只需要更改广播时间，其他设置均保持不变，可直接复制到你的文件中。

```
/*
* This file has been automatically generated by the WICED 20719-B1 Designer.
* Device Configuration.
*
*/

/** wiced_bt_cfg.c
*
* Runtime Bluetooth stack configuration parameters
*
*/

#include "wiced_bt_dev.h"
#include "wiced_bt_ble.h"
#include "wiced_bt_uuid.h"
#include "wiced_bt_gatt.h"
#include "wiced_bt_avrc.h"
#include "wiced_bt_cfg.h"

/* Null-Terminated Local Device Name */
uint8_t BT_LOCAL_NAME[] = { 'G','o','B','l','e','T','e','s','t','\0' };
const uint16_t BT_LOCAL_NAME_CAPACITY = sizeof(BT_LOCAL_NAME);


/*******************************************************************
* wiced_bt core stack configuration
*******************************************************************/
const wiced_bt_cfg_settings_t wiced_bt_cfg_settings =
{
    .device_name =              (uint8_t*)BT_LOCAL_NAME,                 /**< Local device name (NULL terminated) */
    .device_class =             {0x00, 0x00, 0x00},                 /**< Local device class */
    .security_requirement_mask =      BTM_SEC_NONE,                   /**< Security requirements mask (BTM_SEC_NONE,
or combination of BTM_SEC_IN_AUTHENTICATE, BTM_SEC_OUT_AUTHENTICATE, BTM_SEC_ENCRYPT */
    .max_simultaneous_links =        1,                         /**< Maximum number of simultaneous links to different devices
*/

    /* BR/EDR Scan Configuration */
    .br_edr_scan_cfg = {
        .inquiry_scan_type =         BTM_SCAN_TYPE_STANDARD,                /**< Inquiry Scan Type
(BTM_SCAN_TYPE_STANDARD or BTM_SCAN_TYPE_INTERLACED) */
        .inquiry_scan_interval =       0x0000,                      /**< Inquiry Scan Interval (0 to use default) */
        .inquiry_scan_window =        0x0000,                      /**< Inquiry Scan Window (0 to use default) */

        .page_scan_type =           BTM_SCAN_TYPE_STANDARD,                /**< Page Scan Type
(BTM_SCAN_TYPE_STANDARD or BTM_SCAN_TYPE_INTERLACED) */
        .page_scan_interval =         0x0000,                      /**< Page Scan Interval (0 to use default) */
        .page_scan_window =          0x0000,                      /**< Page Scan Window (0 to use default) */
    },

    /* BLE Scan Settings */
    .ble_scan_cfg = {
        .scan_mode =              BTM_BLE_SCAN_MODE_PASSIVE,               /**< BLE Scan Mode
(BTM_BLE_SCAN_MODE_PASSIVE or BTM_BLE_SCAN_MODE_ACTIVE) */

        /* Advertisement Scan Configuration */
        .high_duty_scan_interval =     WICED_BT_CFG_DEFAULT_HIGH_DUTY_SCAN_INTERVAL,      /**< High Duty Scan Interval */
        .high_duty_scan_window =      WICED_BT_CFG_DEFAULT_HIGH_DUTY_SCAN_WINDOW,       /**< High Duty Scan
Window */
        .high_duty_scan_duration =     5,                         /**< High Duty Scan Duration in seconds (0 for infinite) */

        .low_duty_scan_interval =      WICED_BT_CFG_DEFAULT_LOW_DUTY_SCAN_INTERVAL,       /**< Low Duty Scan Interval */
        .low_duty_scan_window =       WICED_BT_CFG_DEFAULT_LOW_DUTY_SCAN_WINDOW,        /**< Low Duty Scan Window
*/
        .low_duty_scan_duration =      5,                         /**< Low Duty Scan Duration in seconds (0 for infinite) */

        /* Connection Scan Configuration */
        .high_duty_conn_scan_interval =   WICED_BT_CFG_DEFAULT_HIGH_DUTY_CONN_SCAN_INTERVAL,     /**< High Duty
Connection Cycle Connection Scan Interval */
        .high_duty_conn_scan_window =    WICED_BT_CFG_DEFAULT_HIGH_DUTY_CONN_SCAN_WINDOW,      /**< High Duty
Connection Cycle Connection Scan Window */
        .high_duty_conn_duration =     30,                        /**< High Duty Connection Cycle Connection Duration in
seconds (0 for infinite) */

        .low_duty_conn_scan_interval =   WICED_BT_CFG_DEFAULT_LOW_DUTY_CONN_SCAN_INTERVAL,     /**< Low Duty
Connection Cycle Connection Scan Interval */
        .low_duty_conn_scan_window =    WICED_BT_CFG_DEFAULT_LOW_DUTY_CONN_SCAN_WINDOW,      /**< Low Duty
Connection Cycle Connection Scan Window */
        .low_duty_conn_duration =      30,                        /**< Low Duty Connection Cycle Connection Duration in
seconds (0 for infinite) */

        /* Connection Configuration */
        .conn_min_interval =         WICED_BT_CFG_DEFAULT_CONN_MIN_INTERVAL,           /**< Minimum Connection Interval */
        .conn_max_interval =         WICED_BT_CFG_DEFAULT_CONN_MAX_INTERVAL,           /**< Maximum Connection Interval
*/
        .conn_latency =            WICED_BT_CFG_DEFAULT_CONN_LATENCY,              /**< Connection Latency */
        .conn_supervision_timeout =     WICED_BT_CFG_DEFAULT_CONN_SUPERVISION_TIMEOUT,       /**< Connection Link
Supervision Timeout */
    },

    /* BLE Advertisement Settings */
    .ble_advert_cfg = {
        .channel_map =             BTM_BLE_ADVERT_CHNL_37 |                /**< Advertising Channel Map (mask of
BTM_BLE_ADVERT_CHNL_37, BTM_BLE_ADVERT_CHNL_38, BTM_BLE_ADVERT_CHNL_39) */
                           BTM_BLE_ADVERT_CHNL_38 |
                           BTM_BLE_ADVERT_CHNL_39,

        .high_duty_min_interval =      WICED_BT_CFG_DEFAULT_HIGH_DUTY_ADV_MIN_INTERVAL,      /**< High Duty Undirected
Connectable Minimum Advertising Interval */
        .high_duty_max_interval =      WICED_BT_CFG_DEFAULT_HIGH_DUTY_ADV_MAX_INTERVAL,      /**< High Duty Undirected
Connectable Maximum Advertising Interval */
        .high_duty_duration =        0,                        /**< High Duty Undirected Connectable Advertising Duration in
seconds (0 for infinite) */

        .low_duty_min_interval =      WICED_BT_CFG_DEFAULT_LOW_DUTY_ADV_MIN_INTERVAL,      /**< Low Duty Undirected
Connectable Minimum Advertising Interval */
        .low_duty_max_interval =      WICED_BT_CFG_DEFAULT_LOW_DUTY_ADV_MAX_INTERVAL,      /**< Low Duty Undirected
Connectable Maximum Advertising Interval */
        .low_duty_duration =         60,                        /**< Low Duty Undirected Connectable Advertising Duration in
seconds (0 for infinite) */

        .high_duty_directed_min_interval =  WICED_BT_CFG_DEFAULT_HIGH_DUTY_DIRECTED_ADV_MIN_INTERVAL,   /**< High Duty
Directed Minimum Advertising Interval */
        .high_duty_directed_max_interval =  WICED_BT_CFG_DEFAULT_HIGH_DUTY_DIRECTED_ADV_MAX_INTERVAL,   /**< High Duty
Directed Maximum Advertising Interval */

        .low_duty_directed_min_interval =  WICED_BT_CFG_DEFAULT_LOW_DUTY_DIRECTED_ADV_MIN_INTERVAL,   /**< Low Duty
Directed Minimum Advertising Interval */
        .low_duty_directed_max_interval =  WICED_BT_CFG_DEFAULT_LOW_DUTY_DIRECTED_ADV_MAX_INTERVAL,   /**< Low Duty
Directed Maximum Advertising Interval */
        .low_duty_directed_duration =    30,                        /**< Low Duty Directed Advertising Duration in seconds (0 for
infinite) */

        .high_duty_nonconn_min_interval =  WICED_BT_CFG_DEFAULT_HIGH_DUTY_NONCONN_ADV_MIN_INTERVAL,   /**< High Duty
Non-Connectable Minimum Advertising Interval */
        .high_duty_nonconn_max_interval =  WICED_BT_CFG_DEFAULT_HIGH_DUTY_NONCONN_ADV_MAX_INTERVAL,   /**< High Duty
Non-Connectable Maximum Advertising Interval */
        .high_duty_nonconn_duration =    30,                        /**< High Duty Non-Connectable Advertising Duration in
seconds (0 for infinite) */

        .low_duty_nonconn_min_interval =  WICED_BT_CFG_DEFAULT_LOW_DUTY_NONCONN_ADV_MIN_INTERVAL,   /**< Low Duty
Non-Connectable Minimum Advertising Interval */
        .low_duty_nonconn_max_interval =  WICED_BT_CFG_DEFAULT_LOW_DUTY_NONCONN_ADV_MAX_INTERVAL,   /**< Low Duty
Non-Connectable Maximum Advertising Interval */
        .low_duty_nonconn_duration =    0,                         /**< Low Duty Non-Connectable Advertising Duration in
seconds (0 for infinite) */
    },

    /* GATT Configuration */
    .gatt_cfg = {
        .appearance =             0x0000,                      /**< GATT Appearance */
        .client_max_links =          1,                         /**< Client Config: Maximum number of servers that local client can
connect to */
        .server_max_links =          1,                         /**< Server Config: Maximum number of remote client connections
allowed by local server */
        .max_attr_len =            512,                        /**< Maximum attribute length; wiced_bt_cfg must have a
corresponding buffer pool that can hold this length */
        .max_mtu_size =            515,                        /**< Maximum MTU size for GATT connections, should be
between 23 and (max_attr_len + 5) */
    },

    /* RFCOMM Configuration */
    .rfcomm_cfg = {
        .max_links =              0,                         /**< Maximum number of simultaneous connected remote devices */
        .max_ports =              0,                         /**< Maximum Number of simultaneous RFCOMM ports */
    },

    /* Application-Managed L2CAP Protocol Configuration */
    .l2cap_application = {
        .max_links =              0,                         /**< Maximum Number of Application-Managed L2CAP Links (BR/EDR
and BLE) */
        .max_psm =               0,                         /**< Maximum Number of Application-Managed BR/EDR PSMs */
        .max_channels =            0,                         /**< Maximum Number of Application-Managed BR/EDR Channels
*/
        .max_le_psm =             0,                         /**< Maximum Number of Application-Managed LE PSMs */
        .max_le_channels =          0,                         /**< Maximum Number of Application-Managed LE Channels */
        .max_le_l2cap_fixed_channels =   0,                         /**< Maximum Number of Application-Managed LE L2CAP
Fixed Channnels supported (in addition to mandatory channels 4, 5, and 6 */
    },

    /* Audio/Video Distribution Configuration */
    .avdt_cfg = {
        .max_links =              0,                         /**< Maximum Number of simultaneous Audio/Video links */
        .max_seps =               0,                         /**< Maximum Number of stream end points */
    },

    /* AVRC Configuration */
    .avrc_cfg = {
        .roles =                0,                         /**< Local Roles supported (AVRC_CONN_INITIATOR or
AVRC_CONN_ACCEPTOR) */
        .max_links =              0,                         /**< Maximum simultaneous Remote Control links */
    },


    /* LE Address Resolution Database Settings */
    .addr_resolution_db_size =        10,                        /**< LE Address Resolution Database Size - Effective only for
pre-4.2 controller */
    .max_number_of_buffer_pools =      4,                         /**< Maximum number of buffer pools in
p_btm_cfg_buf_pools and by wiced_create_pool */
    .rpa_refresh_timeout =          0,     /**< Interval of random address refreshing - secs */
};

/*******************************************************************
* wiced_bt_stack buffer pool configuration
*
* Configure buffer pools used by the stack
*
* Pools must be ordered in increasing buf_size.
* If a pools runs out of buffers, the next pool will be used.
*******************************************************************/
const wiced_bt_cfg_buf_pool_t wiced_bt_cfg_buf_pools[WICED_BT_CFG_NUM_BUF_POOLS] =
{
/*  { buf_size, buf_count, }, */
    { 64,     12,    }, /* Small Buffer Pool */
    { 360,    4,     }, /* Medium Buffer Pool (used for HCI & RFCOMM control messages, min recommended size is 360) */
    { 512,    4,     }, /* Large Buffer Pool  (used for HCI ACL messages) */
    { 1024,   2,     }, /* Extra Large Buffer Pool (used for AVDT media packets and miscellaneous; if not needed, set buf_count to 0) */
};
```

创建GoBleThread.c

要使GoBleThread正常运行，需要执行以下步骤：

1. 添加GATT数据库和WICED蓝牙协议栈includes语句。
2. 对GATT 数据库进行外部引用。
3. 添加一些函数原型。
4. 添加“GoBleThread_start”函数，以设置蓝牙协议栈并运行程序。这包括提供蓝牙管理和GATT事件处理程序函数。

```
#include "GoBle_db.h"
#include "wiced.h"
#include "wiced_bt_ble.h"
#include "wiced_bt_gatt.h"
#include "wiced_bt_stack.h"

/*******************************************************************
* Variable Definitions
*******************************************************************/
extern const wiced_bt_cfg_settings_t wiced_bt_cfg_settings;
extern const wiced_bt_cfg_buf_pool_t wiced_bt_cfg_buf_pools[WICED_BT_CFG_NUM_BUF_POOLS];

/*******************************************************************
* Function Prototypes
*******************************************************************/
static wiced_bt_dev_status_t  goble_management_callback    ( wiced_bt_management_evt_t event, wiced_bt_management_evt_data_t
*p_event_data );
static void                   goble_set_advertisement_data ( void );
static wiced_bt_gatt_status_t goble_event_handler          ( wiced_bt_gatt_evt_t  event, wiced_bt_gatt_event_data_t *p_event_data );

/*******************************************************************
* Function Definitions
*******************************************************************/
void GoBleThread_start(void)
{
    wiced_bt_stack_init(goble_management_callback, &wiced_bt_cfg_settings, wiced_bt_cfg_buf_pools);
}
```

创建一个设置广播数据的函数。GoBle iOS应用程序会查找广播GoBLE服务UUID的外设。

```
/* Set Advertisement Data */
void goble_set_advertisement_data( void )
{
    wiced_bt_ble_advert_elem_t adv_elem[3] = { 0 };
    uint8_t adv_flag = BTM_BLE_GENERAL_DISCOVERABLE_FLAG | BTM_BLE_BREDR_NOT_SUPPORTED;
    uint8_t num_elem = 0;

    /* Advertisement Element for Flags */
    adv_elem[num_elem].advert_type = BTM_BLE_ADVERT_TYPE_FLAG;
    adv_elem[num_elem].len = sizeof(uint8_t);
    adv_elem[num_elem].p_data = &adv_flag;
    num_elem++;

    uint8_t gobleuuid[] = {__UUID_GOBLE};

    /* Advertisement Element for Name */
    adv_elem[num_elem].advert_type = BTM_BLE_ADVERT_TYPE_128SRV_COMPLETE;
    adv_elem[num_elem].len = 16;
    adv_elem[num_elem].p_data = gobleuuid;
    num_elem++;

    /* Set Raw Advertisement Data */
    wiced_bt_ble_set_raw_advertisement_data(num_elem, adv_elem);
}
```

在协议栈启动或发生某一配对事件时，蓝牙管理事件处理程序就需要采取一些措施。

```
/* Bluetooth Management Event Handler */
wiced_bt_dev_status_t goble_management_callback( wiced_bt_management_evt_t event, wiced_bt_management_evt_data_t *p_event_data )
{
    wiced_bt_dev_status_t status = WICED_BT_SUCCESS;

    switch (event)
    {
    case BTM_ENABLED_EVT:
        goble_set_advertisement_data();
        wiced_bt_gatt_register( goble_event_handler );
        wiced_bt_gatt_db_init( gatt_database, gatt_database_len );
        wiced_bt_start_advertisements(BTM_BLE_ADVERT_UNDIRECTED_HIGH, 0, NULL);
        break;

    case BTM_SECURITY_REQUEST_EVT:
        wiced_bt_ble_security_grant(p_event_data->security_request.bd_addr, WICED_BT_SUCCESS);
        break;

    case BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT:
        p_event_data->pairing_io_capabilities_ble_request.local_io_cap = BTM_IO_CAPABILITIES_NONE;
        p_event_data->pairing_io_capabilities_ble_request.oob_data = BTM_OOB_NONE;
        p_event_data->pairing_io_capabilities_ble_request.auth_req = BTM_LE_AUTH_REQ_NO_BOND;
        break;


    case BTM_USER_CONFIRMATION_REQUEST_EVT: // Just confirm
        wiced_bt_dev_confirm_req_reply( WICED_BT_SUCCESS , p_event_data->user_confirmation_request.bd_addr);
        break;

    case BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT:
        WPRINT_APP_INFO(("Paired linke keys\n"));
        status = WICED_BT_ERROR;
        break;

    case BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT:
    case BTM_PAIRING_COMPLETE_EVT:
    case BTM_ENCRYPTION_STATUS_EVT:
    case BTM_BLE_ADVERT_STATE_CHANGED_EVT:
    case BTM_LPM_STATE_LOW_POWER:
        break;

    default:
        WPRINT_APP_INFO(("Unhandled Bluetooth Management Event: 0x%x (%d)\n", event, event));
        break;
    }

    return status;
}
```

在发生以下事件时将调用GATT事件处理程序：

1. 建立或中断连接时
2. GoBle app写入GATT数据库时， 这时只需要显示写入的数值。

```
/* GATT Event Handler */
wiced_bt_gatt_status_t goble_event_handler( wiced_bt_gatt_evt_t event, wiced_bt_gatt_event_data_t *p_event_data )
{
    wiced_bt_gatt_status_t status = WICED_BT_GATT_ERROR;
    wiced_bt_gatt_attribute_request_t *p_attr_req = NULL;

    switch ( event )
    {
    case GATT_CONNECTION_STATUS_EVT:
        if(!p_event_data->connection_status.connected)
        {
            WPRINT_APP_INFO(("Disconnected\n"));
            wiced_bt_start_advertisements(BTM_BLE_ADVERT_UNDIRECTED_HIGH, 0, NULL);
        }
        else
            WPRINT_APP_INFO(("Connected\n"));
        break;
    case GATT_ATTRIBUTE_REQUEST_EVT:

        p_attr_req = &p_event_data->attribute_request;
        if( p_attr_req->request_type == GATTS_REQ_TYPE_WRITE  && p_attr_req->data.handle == HDLC_GOBLE_SERIALPORTID_VALUE)
        {
            uint32_t numButtons = p_attr_req->data.write_req.p_val[3];
            uint32_t sliderY = p_attr_req->data.write_req.p_val[5+numButtons];
            uint32_t sliderX = p_attr_req->data.write_req.p_val[6+numButtons];
            uint32_t buttonMask = 0x00;
            for(int i=0;i<numButtons;i++)
            {
                buttonMask |= (1<<p_attr_req->data.write_req.p_val[5+i]);
            }

            WPRINT_APP_INFO(("# Buttons = %d ButtonMask=%02X Slider x=%02X Slider Y=%02X Raw=",(int)numButtons,(unsigned
int)buttonMask,(unsigned int)sliderX,(unsigned int)sliderY));

            for(int i=0;i<p_attr_req->data.write_req.val_len;i++)
            {
                WPRINT_APP_INFO(("%02X ",p_attr_req->data.write_req.p_val[i]));
            }
            WPRINT_APP_INFO(("\n"));
            status = WICED_BT_GATT_SUCCESS;
        }
        break;
    default:
        status = WICED_BT_GATT_SUCCESS;
        break;
    }

    return status;
}
```

Create GoBleThread.h

```
#pragma once

extern void GoBleThread_start(void);
```

创建GoBle.c以启动GoBleThread

```
#include "GoBleThread.h"
#include "wiced.h"


/*******************************************************************
* Function Definitions
*******************************************************************/
void application_start(void)
{
    wiced_init();
    GoBleThread_start();

}
```

编译、编程并测试

MOUSER PSOC6-WIFI-BT L6为游戏添加GOBLE BLUETOOTH功能

BY ALAN HAWSE . 4343W, BLUETOOTH, CY8CKIT-062-WIFI-BT, MOUSER 10-24-18 PSOC 6 & 4343W WIFI/BLE, PSOC 6, WICED, WIFI . 24 OCT 2018

借助PSOC® 6 MCU和WICED® WI- FI/BLUETOOTH技术，打造低功耗云互联物联网设备

摘要

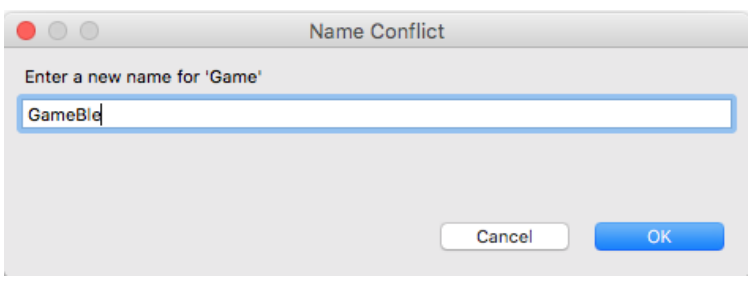
现在我们已经设置好了BLE远程控制和游戏。本课程就是要把这两个项目合并起来，让GoBle远程控制能够控制游戏手柄。我们需要创建消息（就像CapSense消息一样）并通过paddleQueue将它们发送给游戏线程来完成此操作。此外，在按下“Button A”时会发送“Button0”消息。

要完成本课程，需要执行以下步骤：

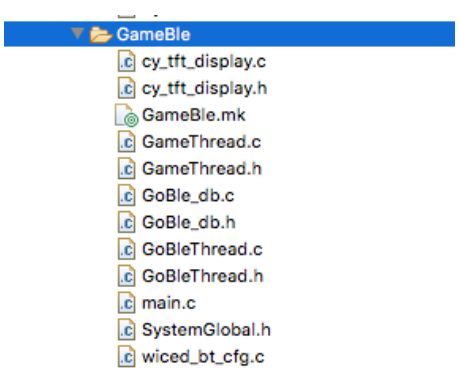
1. 将所有代码复制到一个新项目中
2. 修改Makefile
3. 修改main.c
4. 创建一个新的make target
5. 进行测试以确保所有代码都能正常运行
6. 修改GoBleThread.c
7. 编程并测试

将L4Game复制到L6GameBle新项目中

使用“复制”、“粘帖”来创建一个新项目，在粘帖时输入名称“L6GameBle”（截屏中显示有误）。



将GoBle_db.c/h, GoBleThread.c/h, and wiced_bt_cfg.c文件从GoBle项目复制到新的GoBle项目。复制完成后，应如下所示：



修改Makefile

1. 将App名改为“App_WStudio_L6GameBle”
2. 添加新的源文件
3. 添加BLE库“libraries/drivers/bluetooth/low_energy”

```
1   NAME := App_WStudio_L6GameBle
2
3   $(NAME)_SOURCES := main.c \
4       CapSenseThread.c \
5   GameThread.c \
6   cy_tft_display.c \
7   GoBleThread.c \
8   GoBle_db.c \
9   wiced_bt_cfg.c
10
11  $(NAME)_COMPONENTS := graphics/ugui \
12                        libraries/drivers/bluetooth/low_energy
```

修改main.c

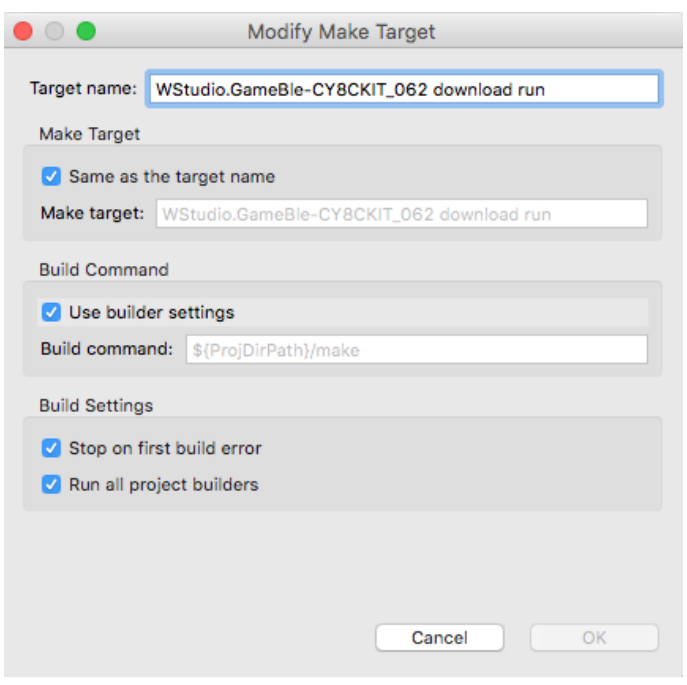
添加GeBleThread的 include语句。

```
1   #include "GameThread.h"
2   #include "GoBleThread.h"
3   #include "CapSenseThread.h"
4   #include "wiced.h"
```

在application_start 函数中，添加GoBleThread_start()调用，以运行GoBle线程。

```
93  void application_start( )
94  {
95      wiced_init( );
96      wiced_rtos_init_queue(&paddleQueue,"paddleQueue",sizeof(game_msg_t),10);
97      wiced_rtos_create_thread(&blinkThreadHandle,7,"Blink Thread",pdlBlinkThread,500,0);
98      wiced_rtos_create_thread(&capSenseThreadHandle,7,"CapSense Thread",capSenseThread,1024,0);
99      wiced_rtos_create_thread(&gameThreadHandle,7,"game Thread",gameThread,4096,0);
100     GoBleThread_start();
101 }
```

创建一个新的make target



进行测试以确保所有代码都能正常运行

运行make target，确保游戏仍能运行并能获取远程控制消息。

修改GoBleThread.c

添加GameThread的include语句（以访问游戏消息结构）和SystemGlobal的include语句（以访问队列）。

```
8   #include "GameThread.h"
9   #include "SystemGlobal.h"
```

我不需要（或想要）按钮/滑块打印信息，所以我将删除goble_event_handler中的原有按钮代码（删除此代码段）。(delete this stuff)

```
137         WPRINT_APP_INFO(("# Buttons = %d ButtonMask=%02X Slider x=%02X Slider Y=%02X Raw=",(int)numButtons,(unsigned
138 int)buttonMask,(unsigned int)sliderX,(unsigned int)sliderY));
139
140         for(int i=0;i<p_attr_req->data.write_req.val_len;i++)
141         {
142             WPRINT_APP_INFO(("%02X ",p_attr_req->data.write_req.p_val[i]));
143         }
            WPRINT_APP_INFO(("\n"));
```

现在，需要更新代码将滑块和按钮信息发送给GameThread（通过paddleQueue）。我们会发现GoBle上的滑块值为0->0xFF，方向与游戏设置的相反，因此需要通过第143行将此值缩小到100，并反转反向，以便控制器能够正确移动手柄。此消息格式与CapSense的完全相同，因此CapSense滑块将与GoBle控制器同步运行。

```
127     case GATT_ATTRIBUTE_REQUEST_EVT:
128
129         p_attr_req = &p_event_data->attribute_request;
130         if( p_attr_req->request_type == GATTS_REQ_TYPE_WRITE  && p_attr_req->data.handle == HDLC_GOBLE_SERIALPORTID_VALUE)
131         {
132             uint32_t numButtons = p_attr_req->data.write_req.p_val[3];
133             uint32_t sliderY = p_attr_req->data.write_req.p_val[5+numButtons];
134             uint32_t sliderX = p_attr_req->data.write_req.p_val[6+numButtons];
135             uint32_t buttonMask = 0x00;
136             for(int i=0;i<numButtons;i++)
137             {
138                 buttonMask |= (1<<p_attr_req->data.write_req.p_val[5+i]);
139             }
140
141             game_msg_t msg;
142             msg.evt = MSG_POSITION;
143             msg.val = 100 - (100*sliderY/255);
144             wiced_rtos_push_to_queue(&paddleQueue,&msg,0);
145             if(buttonMask & 0x10)
146             {
147                 msg.evt = MSG_BUTTON0;
148                 msg.val = 1;
149                 wiced_result_t result=wiced_rtos_push_to_queue(&paddleQueue,&msg,0);
150             }
151             status = WICED_BT_GATT_SUCCESS;
152         }
153         break;
```

编程并测试

MOUSER PSOC 6-WIFI-BT L7：实现WIFI和AWS功能

BY ALAN HAWSE ▪ 4343W, BLUETOOTH, CY8CKIT-062-WIFI-BT, MOUSER 10-24-18 PSOC 6 & 4343W WIFI/BLE, PSOC 6, WICED, WIFI ▪ 24 OCT 2018

借助PSOC® 6 MCU和WICED® WI- FI/BLUETOOTH技术，打造低功耗云互联物联网设备

#	课程
0	简介
1	开发者资源
2	WICED Studio与CapSense
3	使用CY8CKIT-028-TFT扩展板
4	视频游戏
5	GoBle与Bluetooth
6	为游戏添加GoBle Bluetooth功能
7	实现WiFi和AWS功能
8	为游戏添加WiFi和AWS功能

摘要

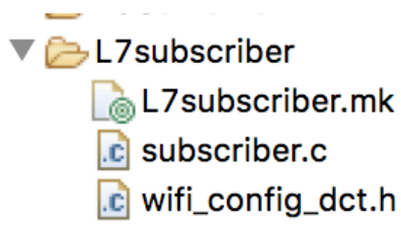
本课程中我们将要更新WICED Studio中的一个预定义应用程序，它定义了如何连接AWS IoT云以便进行网络操作。此外，我们还将展示WiFi和MQTT功能，并修改应用程序使之拥有一个称为“PADDLE”的新MQTT主题，用于发送移动游戏的消息。

要完成本项目，需要执行以下步骤：

1. 复制demo/aws/iot/pub_sub/subscriber中的subscriber
2. 在AWS中设置“thing”，并下载证书
3. 更新WICED安装程序中的证书
4. 通过AWS配置更新项目
5. 测试
6. 更新项目使之与游戏兼容
7. 测试

复制Subscriber并修改Makefile

首先需要将demo/aws/iot/pub_sub/subscriber中的subscriber应用程序复制/粘帖到我的文件夹中，并将项目重命名为L7subscriber，如下所示：

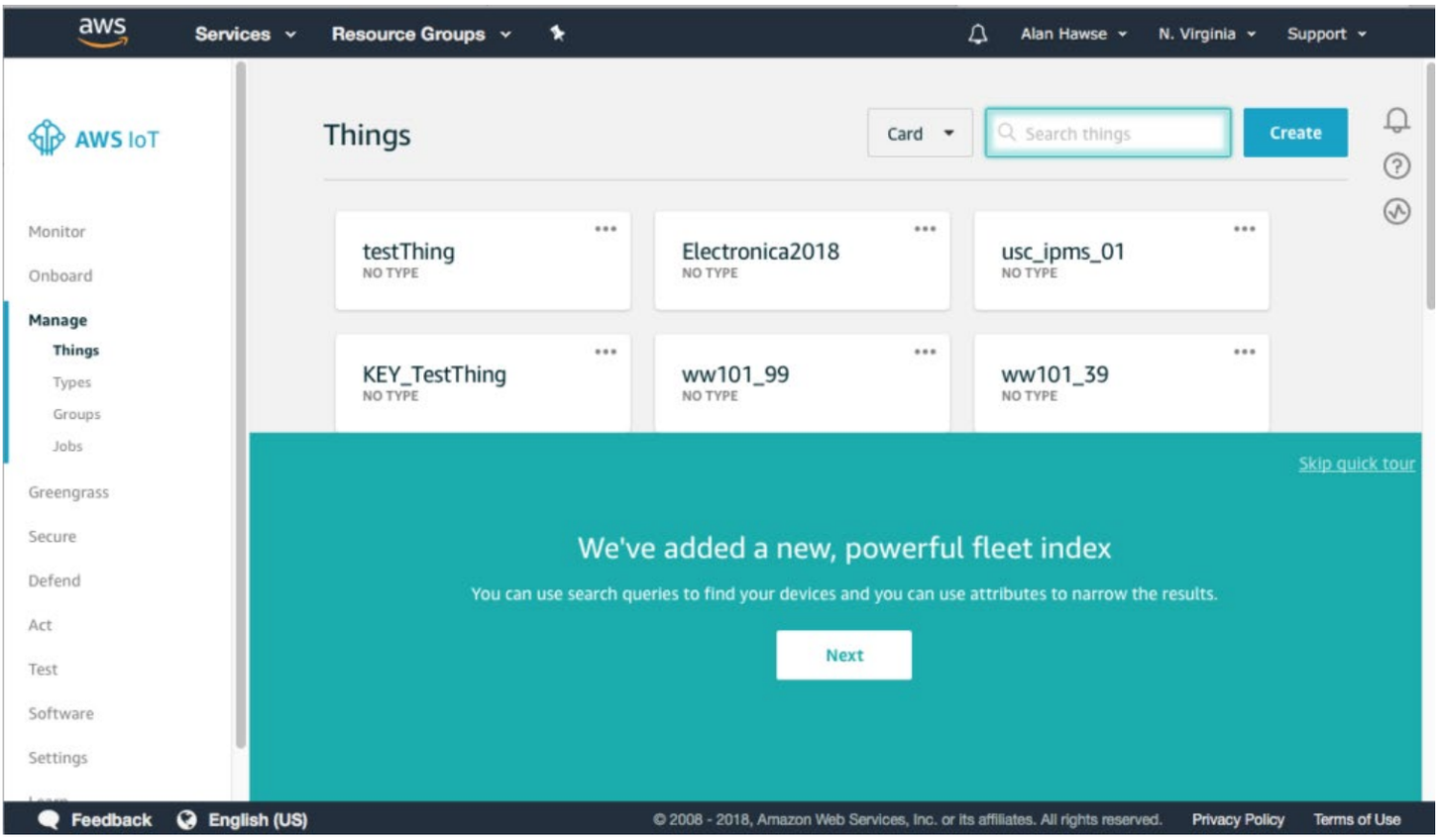


然后需要编辑Makefile来更新项目名。

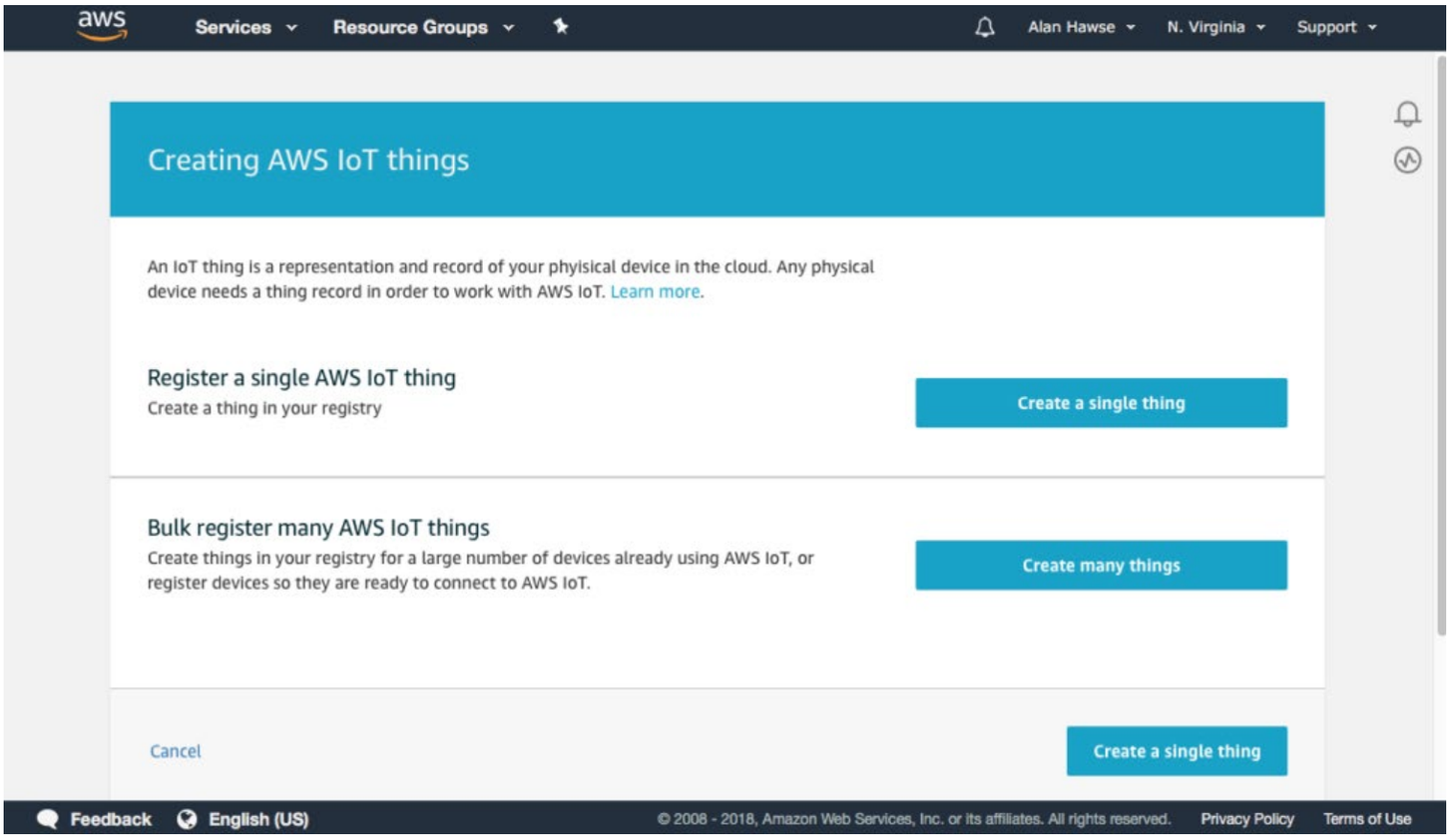
```
NAME := App_WStudio_L7subscriber
<a href="https://iotexpert.com/?attachment_id=5975" rel="attachment wp-att-5975"></a>
```

转至Amazon.com并创建“thing”

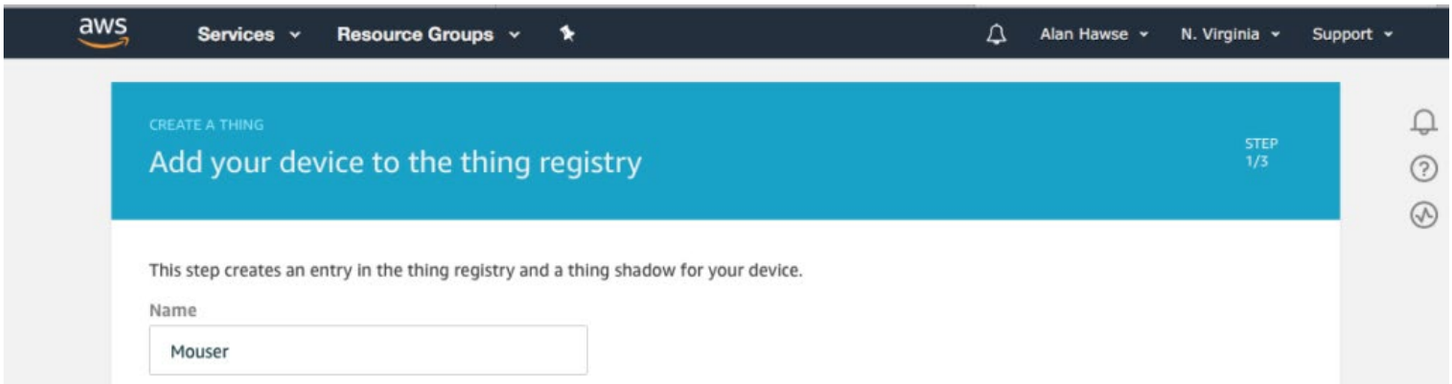
要连接AWS云，需创建一个“thing”。操作如下：转至AWS控制台，单击“Create”。

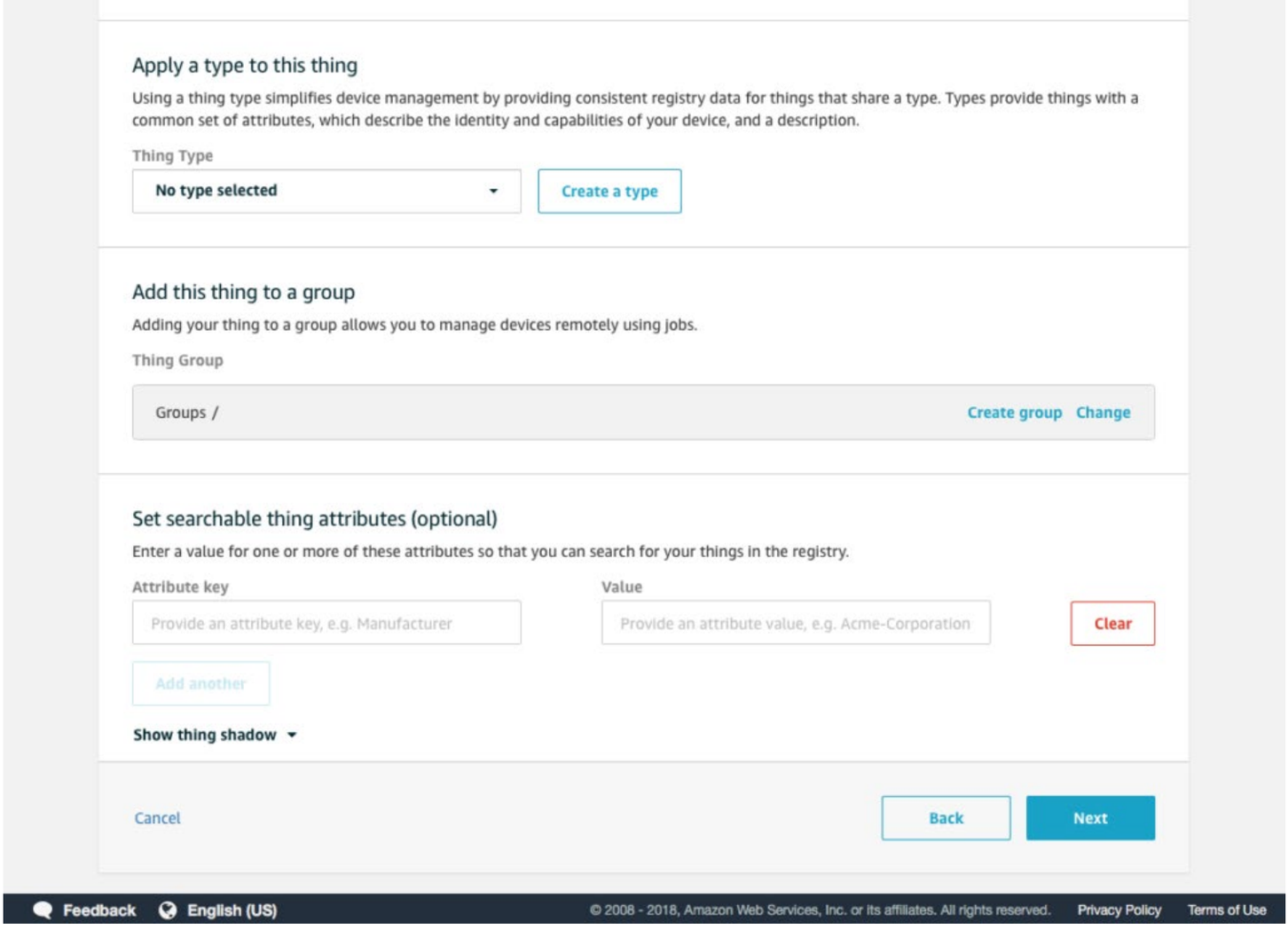


单击“Create a single thing”，因为我们只需创建一个thing。

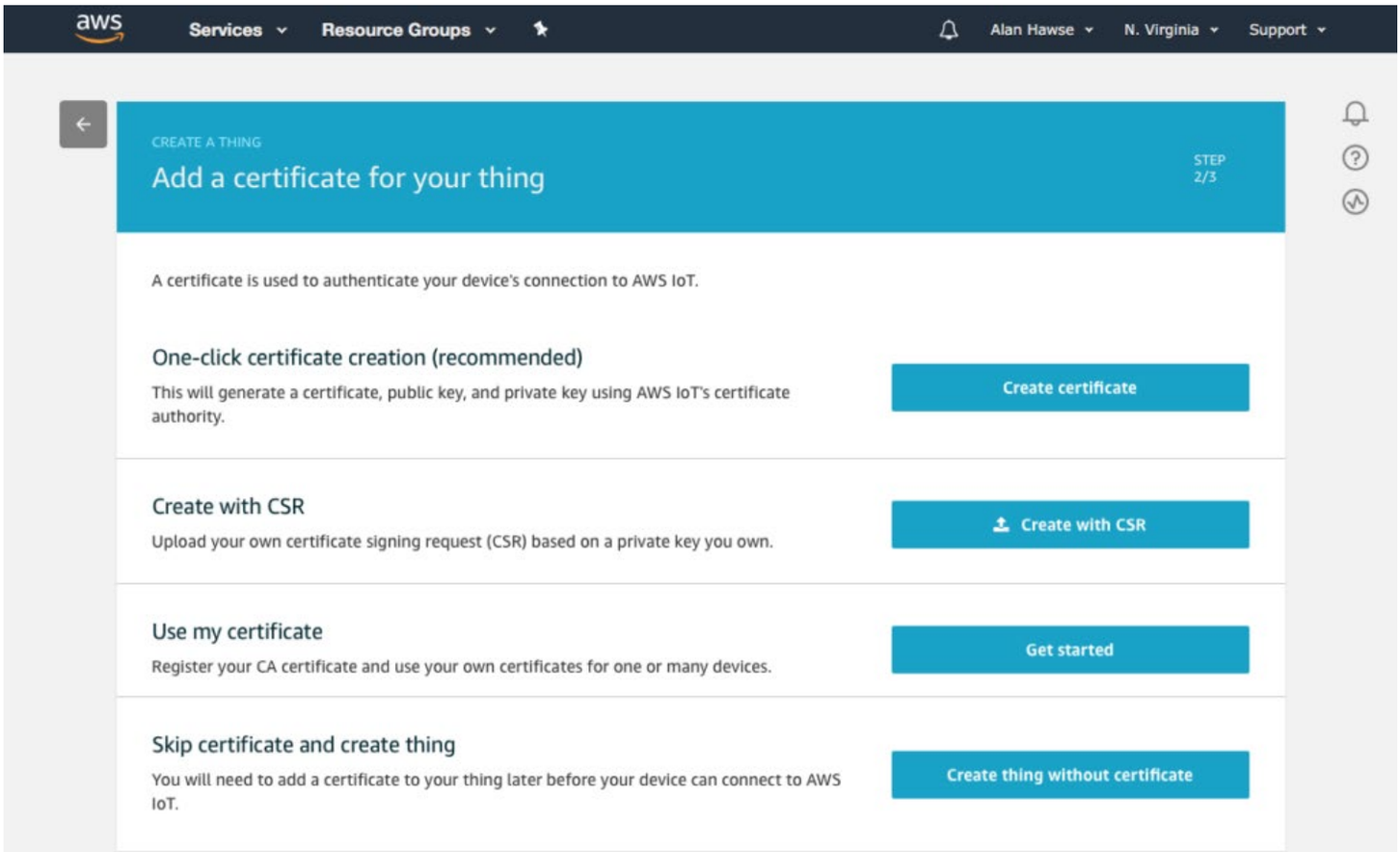


在Name框中输入名字，本例中我们将其命名为Mouser。

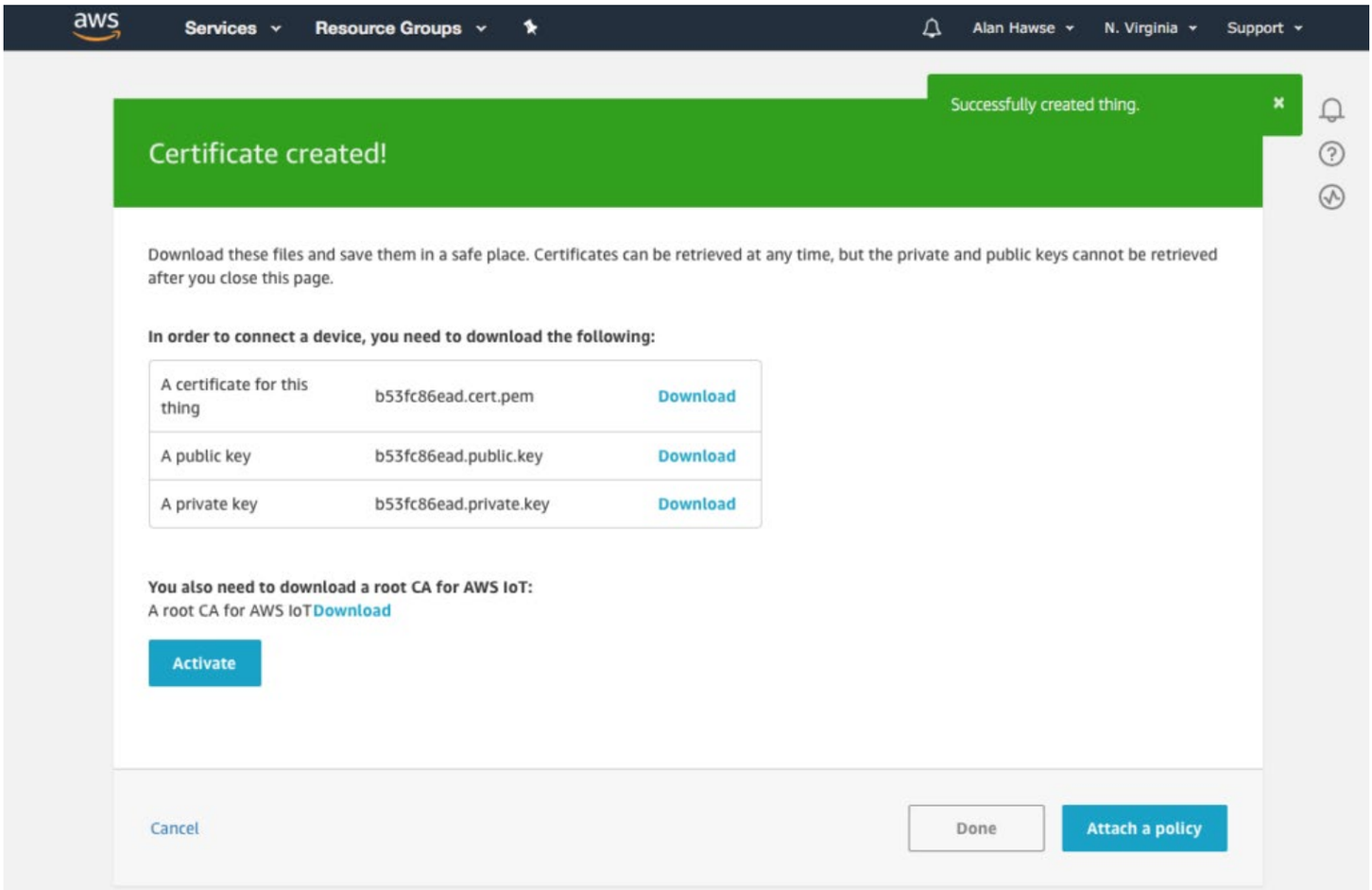




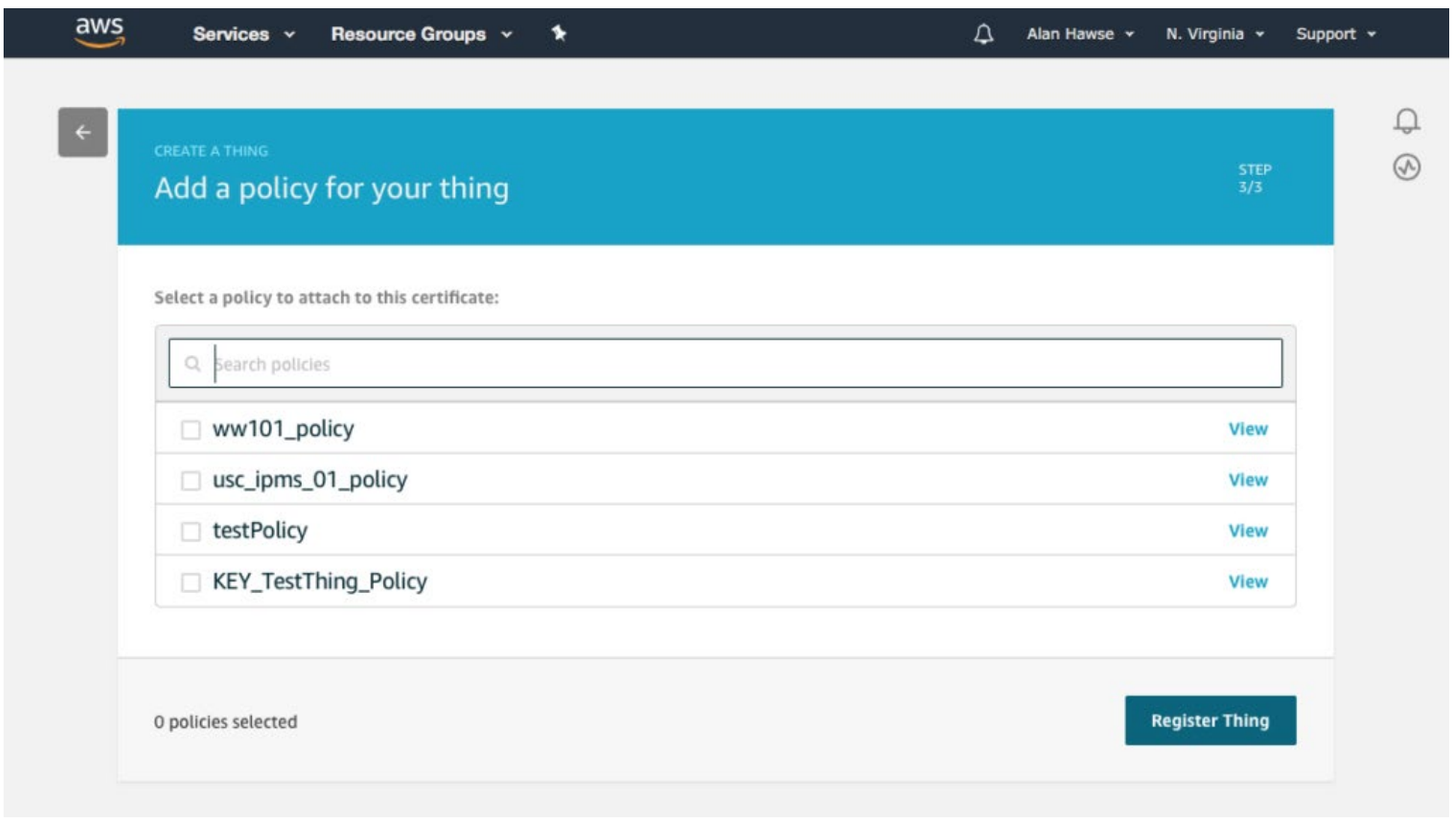
AWS拥有设备级安全性，可通过RSA证书针对每个设备进行设置。 因此，需要AWS创建一个证书。



完成后，应该且必须立即下载设备证书和两个密钥，后面将无法再进行下载，并应在获得证书和密钥后立即激活证书。



现在为thing添加策略。我已经创建了一个“ww101_policy”策略。



此策略如下所示。基本上来说，这些thing都是公开的。

X.509 Certificates and AWS IoT - AWS IoT | Edit Post ‹ IoT Expert — WordPress | AWS IoT

aws Services Resource Groups Alan Hawse N. Virginia Support

es > ww101_policy

POLICY

ww101_policy

Actions

Overview
Certificates
Versions
Groups
Non-compliance

Policy ARN

A policy ARN uniquely identifies this policy. **Learn more**

```
arn:aws:iot:us-east-1:478080436345:policy/ww101_policy
```

Policy document

The policy document defines the privileges of the request. **Learn more**

Version 1 updated Mar 21, 2018 9:31:31 AM -0400 **Edit policy document**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*"
    }
  ]
}
```

Feedback English (US) © 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

更新证书

最后还需要一份最新的AWS证书。WICED推荐使用RSA2048位密钥。

X.509 Certificates and AWS IoT - AWS IoT | Edit Post ‹ IoT Expert — WordPress | Mouser VTW - Lesson 6 : Integrate GoBle Blueto... | AWS IoT

aws English Sign In to the Console

AWS IoT
Developer Guide

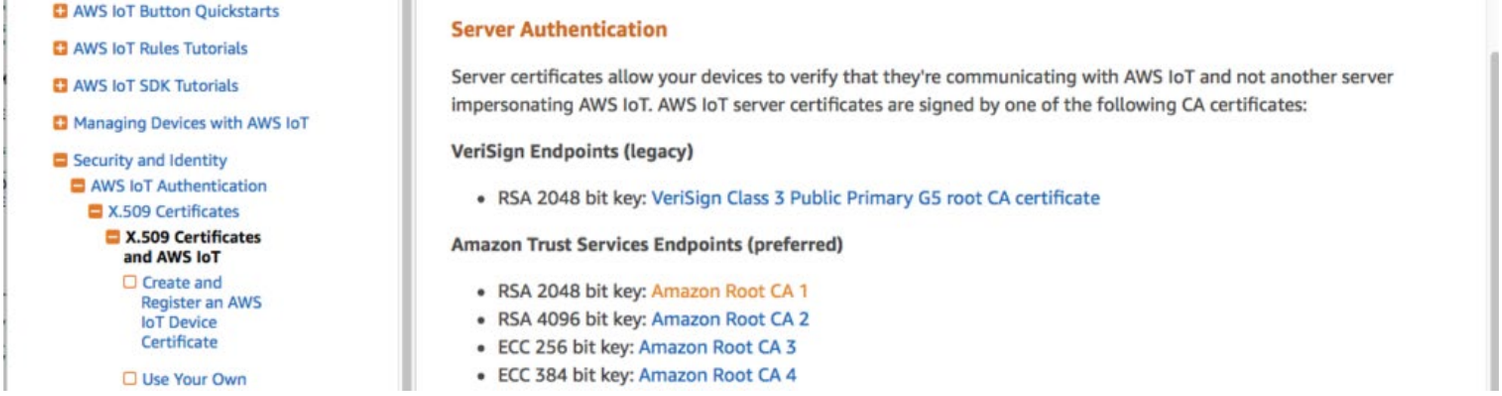
Documentation - This Guide
Search

What Is AWS IoT?
Getting Started with AWS IoT

- Revoke a device certificate.
- Transfer a device certificate to another AWS account.
- List all CA certificates registered to your AWS account.
- List all device certificates registered to your AWS account.

For more information about the CLI commands to use to perform these operations, see AWS IoT CLI Reference.

For more information about using the AWS IoT console to create certificates, see Create and Activate a Device Certificate.

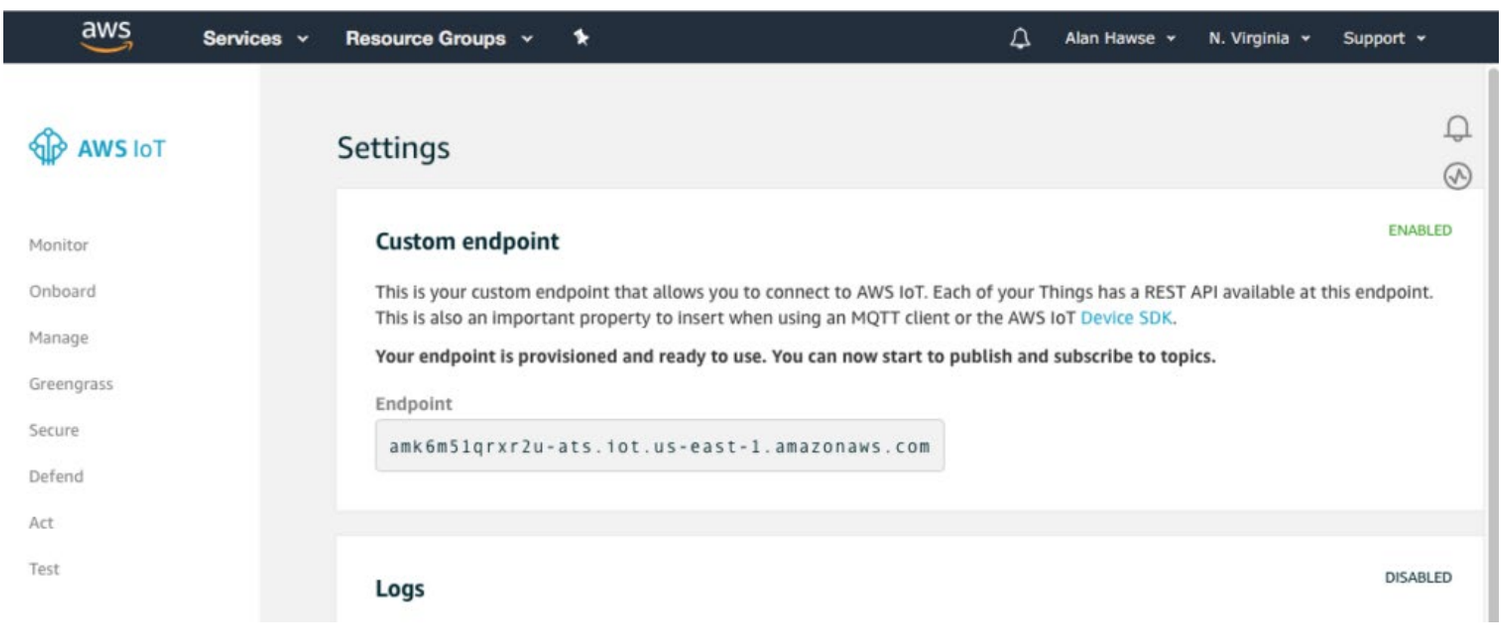


现在，我们有了四个安全文档，需要将它们集成到WICED安装程序中。为此，应把目录更改为~/Documents/WICED-Studio-6.2/43xxx_Wi-Fi/resources/apps/aws/iot，复制文件，并创建符号链接。如果使用的不是Mac或Linux系统，则需要复制文件并进行重命名。

```
arh iot $ ls
aws_config.html mesh            publisher       rootca.cer      subscriber
arh iot $ mv rootca.cer rootca.cer.old
arh iot $ cp ~/Downloads/AmazonRootCA1.pem.txt .
arh iot $ ln -s AmazonRootCA1.pem.txt rootca.cer
arh iot $ cd subscriber/
arh subscriber $ ls
b53fc86ead-certificate.pem.crt  b53fc86ead-private.pem.key      b53fc86ead-public.pem.key       client.cer
                        privkey.cer
arh subscriber $ rm client.cer
arh subscriber $ ln -s b53fc86ead-certificate.pem.crt client.cer
arh subscriber $ rm privkey.cer
arh subscriber $ ln -s b53fc86ead-private.pem.key  privkey.cer
arh subscriber $
```

更新项目

AWS创建一个虚拟机，并在其上运行MQTT服务器，还能在设置屏幕上看到此虚拟机的DNS名。在此屏幕中可以看到端点设置，需要配置WICED项目使其连接服务器。



可以通过Line 119更改AWS云中MQTT服务器的实际端点。

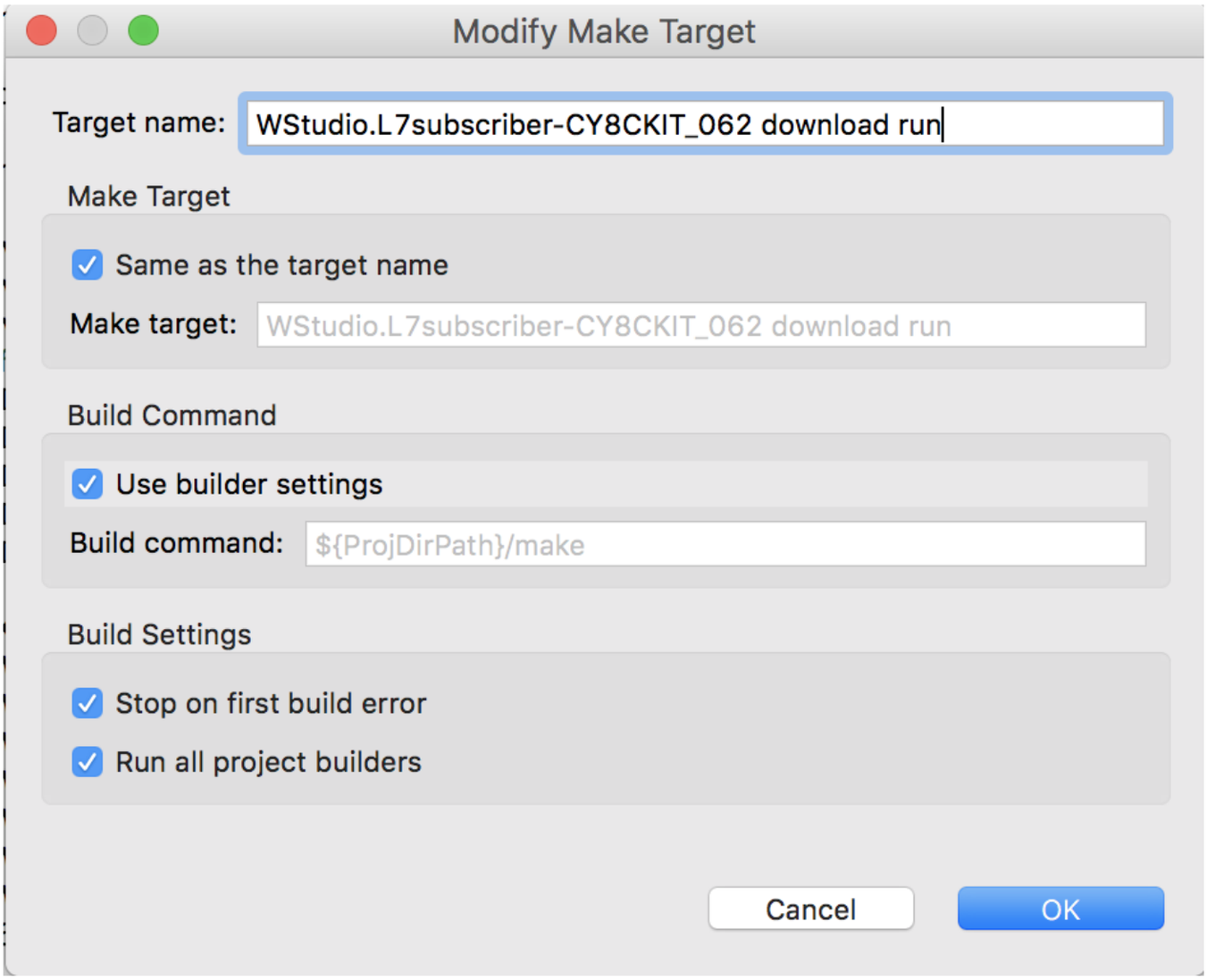
```
117  static wiced_aws_endpoint_info_t my_subscriber_aws_iot_endpoint = {
118      .transport          = WICED_AWS_TRANSPORT_MQTT_NATIVE,
119      .uri                = "amk6m51qrxr2u-ats.iot.us-east-1.amazonaws.com",
120      .peer_common_name   = NULL,
121      .ip_addr            = {0},
122      .port               = WICED_AWS_IOT_DEFAULT_MQTT_PORT,
123      .root_ca_certificate = NULL,
124      .root_ca_length     = 0,
125  };
```

在wifi_config_dct.h文件中，需要根据你的网络更改CLIENT_AP_SSID和CLIENT_AP_PASSPHRASE。

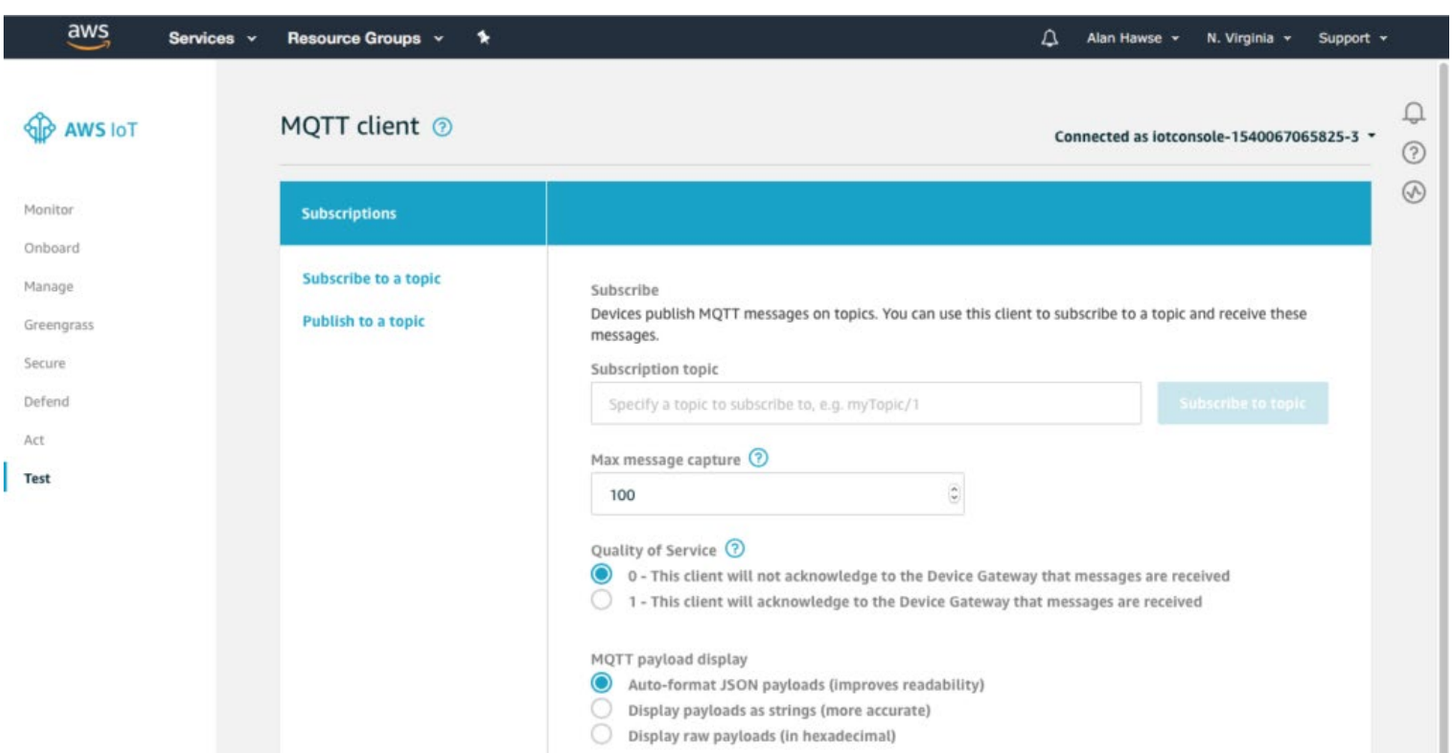
```
/* This is the default AP the device will connect to (as a client)*/
#define CLIENT_AP_SSID       "WW101WPA"
#define CLIENT_AP_PASSPHRASE "yourpassword"
#define CLIENT_AP_BSS_TYPE   WICED_BSS_TYPE_INFRASTRUCTURE
#define CLIENT_AP_SECURITY   WICED_SECURITY_WPA2_MIXED_PSK
#define CLIENT_AP_CHANNEL    1
#define CLIENT_AP_BAND       WICED_802_11_BAND_2_4GHZ
```

测试

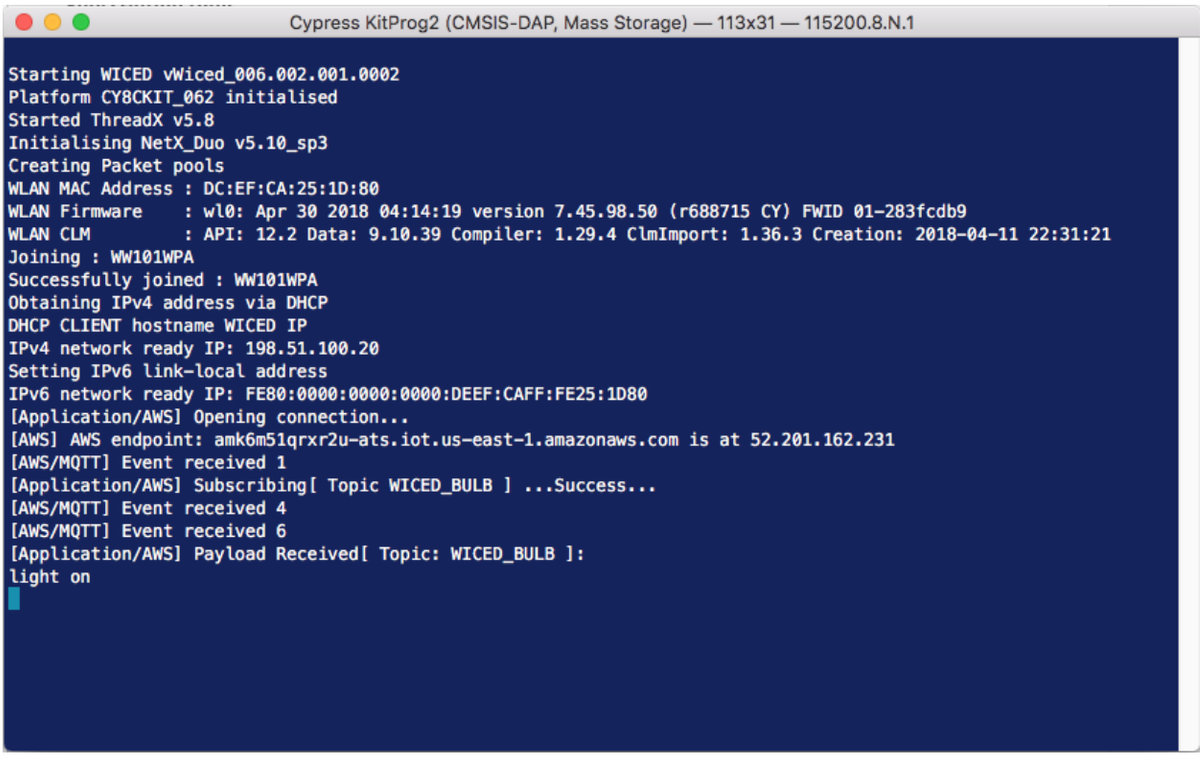
要测试项目，需要创建一个make target，并编程开发套件。



完成之后，可以转至Amazon.com测试控制台并向相应主题发送MQTT消息。



此时，可以看到连网的项目，并成功接收LIGHT ON消息。



针对游戏更新项目

我们知道此游戏并不需要执行LIGHT主题，所以我们需要更改为PADDLE。

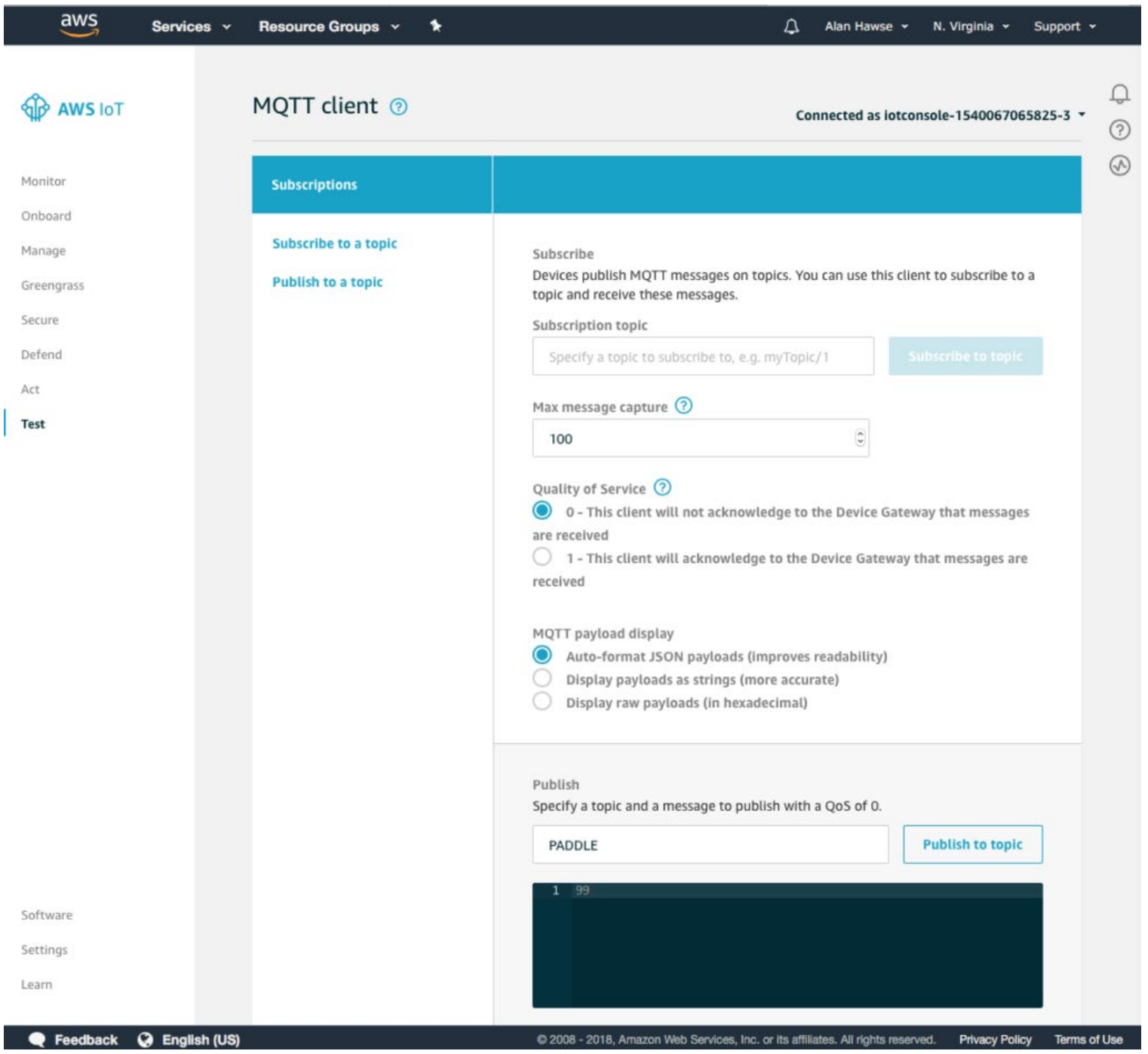
```
#define WICED_TOPIC                    "PADDLE"
```

我们假设发送给手柄的消息ASCII值<100。因此，可以解析此消息并输出。请注意，这里使用sscanf来解析消息。如果使用不慎，会带来很大的麻烦。

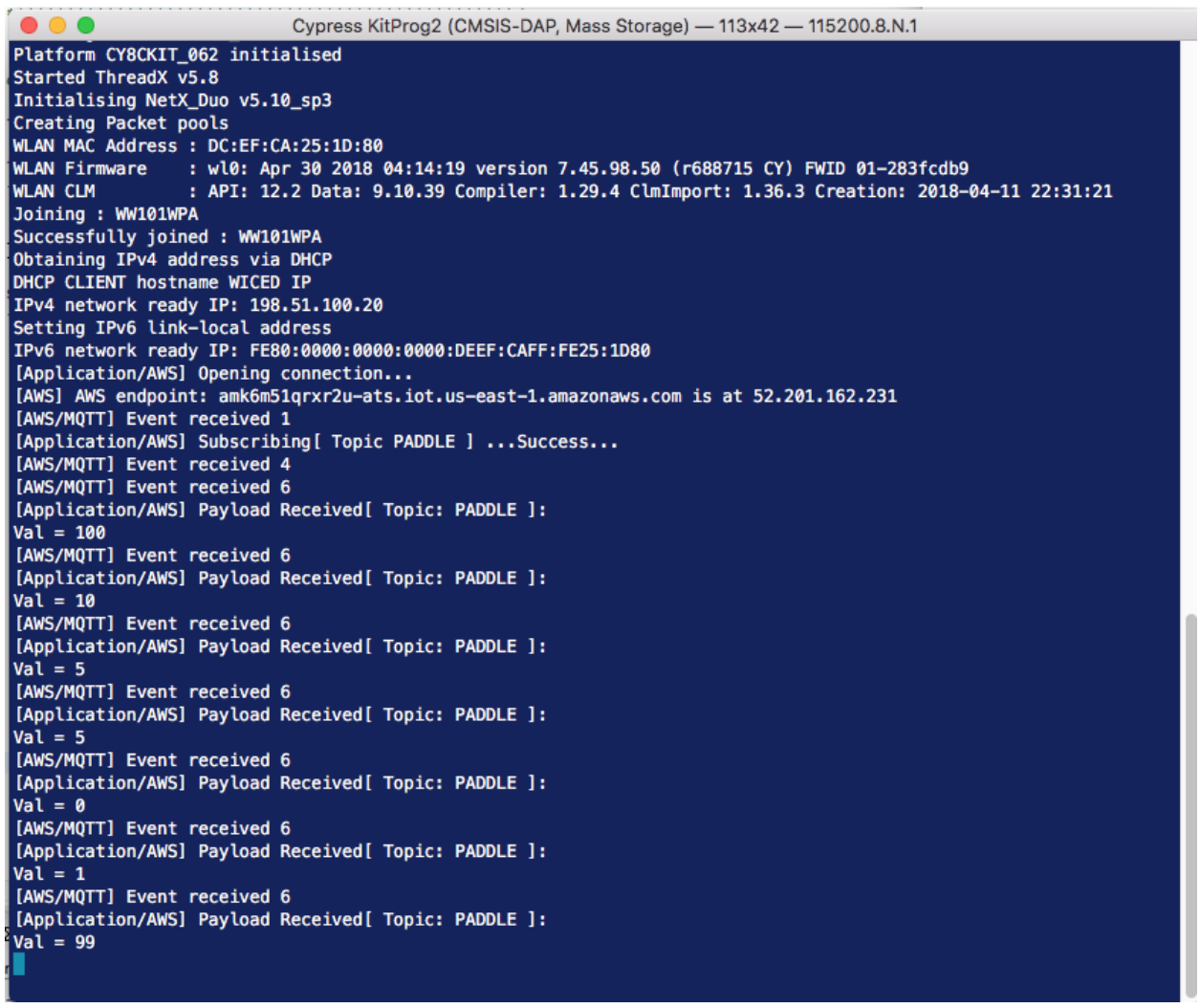
```
        case WICED_AWS_EVENT_PAYLOAD_RECEIVED:
        {
            char buff[10];
            uint32_t val;
            WPRINT_APP_INFO( ("[Application/AWS] Payload Received[ Topic: %.*s ]:\n", (int)data->message.topic_length, data->message.topic
) );

            if(strncmp(WICED_TOPIC,data->message.topic,strlen(WICED_TOPIC)) == 0 && data->message.data_length < 4)
            {
                sscanf(data->message.data,"%d",&val);
                WPRINT_APP_INFO(("Val = %d\n",val));
            }
        }
        break;
```

测试

在进行更新后，需要重新对整个代码进行一次编程。然后通过向PADDLE主题发送消息来进行测试。



如下图所示，表示一切运行正常。



借助PSOC® 6 MCU和WICED® WI- FI/BLUETOOTH技术，打造低功耗云互联物联网设备

#	课程
0	简介
1	开发者资源
2	WICED Studio与CapSense
3	使用CY8CKIT-028-TFT扩展板
4	视频游戏
5	GoBle与Bluetooth
6	为游戏添加GoBle Bluetooth功能
7	实现WiFi和AWS功能
8	为游戏添加WiFi和AWS功能

摘要

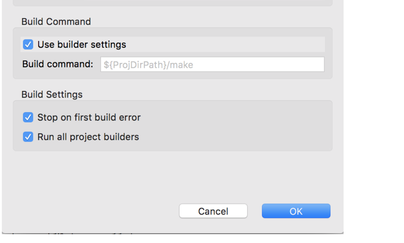
本课程中，我们会将subscriber应用程序功能移到GameBle主项目（现在是GameBleAws）中。为此，需要将subscriber转换为线程并进行修改，将发送到PADDLE主题的消息发送到paddleQueue。

要完成本课程，需要执行以下步骤：

1. 创建一个名为L6GameBle的新项目
2. 复制subscriber.c和wifi_config_dct.h
3. 更新Makefile
4. 创建subscriber.h
5. 更新main.c
6. 更新subscriber.c
7. 测试

创建新项目

将L6GameBle项目复制、粘贴到L8GameBleAws新项目中。



为GameBleAws项目创建一个新的Make target。

复制subscriber.c和wifi_config_dct.h

将subscriber应用程序复制并粘贴到GameBleAws项目中。之后，文件夹应如下所示：

更新Makefile

```
1   NAME := App_WStudio_L8GameBleAws
2
3   $(NAME)_SOURCES := main.c \
4                      CapSenseThread.c \
5   GameThread.c \
6   cy_tft_display.c \
7   GoBleThread.c \
8   GoBle_db.c \
9   wiced_bt_cfg.c \
10  subscriber.c
11
12  $(NAME)_COMPONENTS  := graphics/ugui \
13                         libraries/drivers/bluetooth/low_energy \
14                         protocols/AWS
15
16  WIFI_CONFIG_DCT_H := wifi_config_dct.h
17
18  $(NAME)_RESOURCES  := apps/aws/iot/rootca.cer \
19                        apps/aws/iot/subscriber/client.cer \
20                        apps/aws/iot/subscriber/privkey.cer
21
22  # To support Low memory platforms, disabling components which are not required
23  GLOBAL_DEFINES += WICED_CONFIG_DISABLE_C_SSL_SERVER \
24                    WICED_CONFIG_DISABLE_DTLS \
25                    WICED_CONFIG_DISABLE_ENTERPRISE_SECURITY \
26                    WICED_CONFIG_DISABLE_DES \
27                    WICED_CONFIG_DISABLE_ADVANCED_SECURITY_CURVES
```

创建subscriber.h

要让main.c创建awsThread（subscriber.c的初始application_start），应创建一个subscriber.h文件。然后需要定义一个awsThread函数来匹配wiced_thread_t函数原型（一个接收wiced_thread_arg并返回void的函数）。

```
1   #pragma once
2   extern void awsThread( wiced_thread_arg_t arg );
```

更新main.c

要在主项目中集成awsThread线程，需要添加"subscriber.h"的include语句：

```
1   #include "GameThread.h"
2   #include "GoBleThread.h"
3   #include "CapSenseThread.h"
4   #include "subscriber.h"
5   #include "wiced.h"
```

添加新变量以存放 awsThreadHandle。

```
39  wiced_thread_t awsThreadHandle;
```

最后，通过wiced_rtos_create_thread创建AWS线程，并启动该线程。

```
96  void application_start( )
97  {
98      wiced_init( );
99      wiced_rtos_init_queue(&paddleQueue, "paddleQueue", sizeof(game_msg_t),10);
100     wiced_rtos_create_thread(&blinkThreadHandle,7,"Blink Thread",pdBlinkThread,500,0);
101     wiced_rtos_create_thread(&capsenseThreadHandle,7,"CapSense Thread",capSenseThread,1024,0);
102     wiced_rtos_create_thread(&gameThreadHandle,7,"game Thread",gameThread,4096,0);
103     GoBleThread_start();
104     wiced_rtos_create_thread(&awsThreadHandle,7,"AWS Thread",awsThread,4096,0);
105 }
```

更新subscriber.c

要使用subscriber.c，需要修改include语句，从PADDLE主题向游戏线程发送消息，并将application_start加入线程中。首先要修改include语句（如第6课中的BLE示例所示）

```
88  #include "SystemGlobal.h"
89  #include "GameThread.h"
```

在收到PADDLE主题消息后，应进行解析，然后将其发送到游戏线程以移动手柄，与BLE项目一样。请注意，通过发送一个小于100的值来保护游戏线程。

```
176     case WICED_AWS_EVENT_PAYLOAD_RECEIVED:
177     {
178         uint32_t val;
179         WPRINT_APP_INFO( ("[ApplicationAWS] Payload Received[ Topic: %.*s ]\n", (int)data->message.topic_length, data->message.topic
180 ));
181
182         if(strncmp(WICED_TOPIC,(const char *)data->message.topic,strlen(WICED_TOPIC)) == 0 && data->message.data_length < 4)
183         {
184             sscanf((const char *)data->message.data,"%d",(int *)&val);
185             if(val>100)
186                 val = 100;
187
188             WPRINT_APP_INFO(("Val = %d\n",(int)val));
189             game_msg_t msg;
190             msg.evt = MSG_POSITION;
191             msg.val = val;
192             wiced_rtos_push_to_queue(&paddleQueue,&msg,0);
193         }
194     }
195     break;
```

测试

并不需要一个很好的程序（比GoBle）来测试手柄的移动，但可以看到当游戏结束时，仍可以通过AWS控制台移动手柄。在下面截屏中，可以看到手柄移动到了位置0，并且BLE与AWS同时启动，得分！。