MSP430 Launchpad 快速上手手册[1]

宋凌皓 丛昊

[1] 著者申明：本手册适宜 MSP430 Launchpad 初学者使用，以快速上手为目的，故重在使用，而理论较少，建议结合 datasheet、userguide 及其他资料使用。此为手册第一版，错误与不足可能较多，请读者注意。本手册现仅限上海交大电子系中心实验室使用。

0. MSP430G25x3 概述

德州仪器(TI)MSP430 系列超低功耗微控制器包含多种器件，它们特有面向多种应用的不同外设集。这种架构与 5 种低功耗模式相组合，专为在便携式测量应用中延长电池使用寿命而优化。该器件具有一个强大的 16 位 RISC CPU，16 位寄存器和有助于获得最大编码效率的常数发生器。数字控制振荡器(DCO)可在不到 1μs 的时间里完 成从低功耗模式至运行模式的唤醒。

MSP430G2x53 系列是超低功耗混合信号微控制器，具有内置的 16 位定时器、多达 24 个支持 触摸感测的 I/O 引脚、一个多用途模拟比较器以及采用通用串行通信接口的内置通信能力以及一个 10 位模数 (A/D) 转换器。

该微控制器功能如图 0-1 所示。

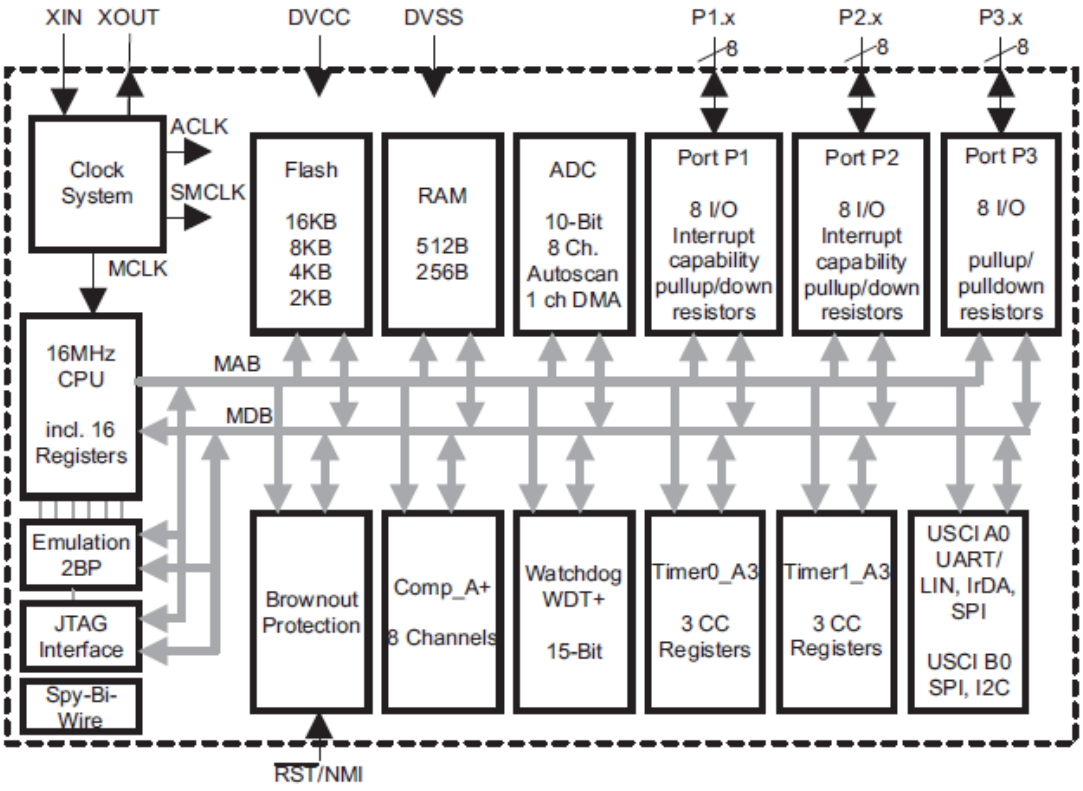


图 0-1

图 0-1 中所示片上外设或功能，与对应的 IO 端口的关系如表 0-1 所示。

表 0-1

端口					
名称	编号			I/O	说明
	PW20，N20	PW28	RHB32		

P1.0/ TA0CLK/ ACLK/ A0 CA0	2	2	31	I/O	通用型数字 I/O 引脚 Timer0_A，时钟信号 TACLK 输入 ACLK 信号输出 ADC10 模拟输入 A0 Comparator_A+，CA0 输入
P1.1/ TA0.0/ UCA0RXD/ UCA0SOMI/ A1/ CA1	3	3	1	I/O	通用型数字 I/O 引脚 Timer0_A，捕捉：CCI0A 输入，比较：Out0 输出 / BSL 发送 USCI_A0 UART 模式：接收数据输入 USCI_A0 SPI 模式：从器件数据输出/主器件数据输入 ADC10 模拟输入 A1 Comparator_A+，CA1 输入
P1.2/ TA0.1/ UCA0TXD/ UCA0SIMO/ A2/ CA2	4	4	2	I/O	通用型数字 I/O 引脚 Timer0_A，捕获：CCI1A 输入，比较：Out1 输出 USCI_A0 UART 模式：发送数据输出 USCI_A0 SPI 模式：从器件数据输入/主器件数据输出 ADC10 模拟输入 A2 Comparator_A+，CA2 输入
P1.3/ ADC10CLK/ A3/ VREF-/VEREF-/ CA3/ CAOUT	5	5	3	I/O	通用型数字 I/O 引脚 ADC10，转换时钟输出 ADC10 模拟输入 A3 ADC10 负基准电压 Comparator_A+，CA3 输入 Comparator_A+，输出
P1.4/ SMCLK/ UCB0STE/ UCA0CLK/ A4/ VREF+/VEREF+/ CA4/ TCK	6	6	4	I/O	通用型数字 I/O 引脚 SMCLK 信号输出 USCI_B0 从器件发送使能 USCI_A0 时钟输入/输出 ADC10 模拟输入 A4 ADC10 正基准电压 Comparator_A+，CA4 输入 JTAG 测试时钟，用于器件编程及测试的输入端口
P1.5/ TA0.0/ UCB0CLK/ UCA0STE/ A5/ CA5/ TMS	7	7	5	I/O	通用型数字 I/O 引脚 Timer0_A，比较：Out0 输出 / BSL 接收 USCI_B0 时钟输入/输出 USCI_A0 从器件发送使能 ADC10 模拟输入 A5 Comparator_A+，CA5 输入 JTAG 测试模式选择，用于器件编程及测试的输入端口
P1.6/ TA0.1/ A6/ CA6/ UCB0SOMI/ UCB0SCL/ TDI/TCLK	14	22	21	I/O	通用型数字 I/O 引脚 Timer0_A，比较：Out1 输出 ADC10 模拟输入 A6 Comparator_A+，CA6 输入 USCI_B0 SPI 模式：从器件输出主器件输入 USCI_B0 I2C 模式：SCL I2C 时钟 编程及测试期间的 JTAG 测试数据输入或测试时钟输入

P1.7/ A7/ CA7/ CAOUT/ UCB0SIMO/ UCB0SDA/ TDO/TDI	15	23	22	I/O	通用型数字 I/O 引脚 ADC10 模拟输入 A7 Comparator_A+，CA7 输入 Comparator_A+，输出 USCI_B0 SPI 模式：从器件输入主器件输出 USCI_B0 I2C 模式：SDA I2C 数据 编程及测试期间的 JTAG 测试数据输出端口或测试数据输入
P2.0/ TA1.0	8	10	9	I/O	通用型数字 I/O 引脚 Timer1_A，捕获：CCI0A 输入，比较：Out0 输出
P2.1/ TA1.1	9	11	10	I/O	通用型数字 I/O 引脚 Timer1_A，捕获：CCI1A 输入，比较：Out1 输出
P2.2/ TA1.1	10	12	11	I/O	通用型数字 I/O 引脚 Timer1_A，捕获：CCI1B 输入，比较：Out1 输出
P2.3/ TA1.0	11	16	15	I/O	通用型数字 I/O 引脚 Timer1_A，捕获：CCI0B 输入，比较：Out0 输出
P2.4/ TA1.2	12	17	16	I/O	通用型数字 I/O 引脚 Timer1_A，捕获：CCI2A 输入，比较：Out2 输出
P2.5/ TA1.2	13	18	17	I/O	通用型数字 I/O 引脚 Timer1_A，捕获：CCI2B 输入，比较：Out2 输出
XIN/ P2.6/ TA0.1	19	27	26	I/O	晶体振荡器的输入端口 通用型数字 I/O 引脚 Timer0_A，比较：Out1 输出
XOUT/ P2.7	18	26	25	I/O	晶体振荡器的输出端口 通用型数字 I/O 引脚
P3.0/ TA0.2	–	9	7	I/O	通用型数字 I/O 引脚 Timer0_A，捕获：CCI2A 输入，比较：Out2 输出
P3.1/ TA1.0	–	8	6	I/O	通用型数字 I/O 引脚 Timer1_A，比较：Out0 输出
P3.2/ TA1.1	–	13	12	I/O	通用型数字 I/O 引脚 Timer1_A，比较：Out1 输出
P3.3/ TA1.2	–	14	13	I/O	通用型数字 I/O Timer1_A，比较：Out2 输出
P3.4/ TA0.0	–	15	14	I/O	通用型数字 I/O Timer0_A，比较：Out0 输出
P3.5/ TA0.1	–	19	18	I/O	通用型数字 I/O Timer0_A，比较：Out1 输出
P3.6/ TA0.2	–	20	19	I/O	通用型数字 I/O Timer0_A，比较：Out2 输出
P3.7/ TA1CLK/ CAOUT	–	21	20	I/O	通用型数字 I/O Timer1_A，时钟信号 TACLK 输入 Comparator_A+，输出
RST/ NMI/ SBWTDIO	16	24	23	I	复位；不可屏蔽中断输入 编程及测试期间的两线制 (Spy-Bi-Wire) 测试数据输入/输出

TEST/ SBWTCK	17	25	24	I	为端口 1 上的 JTAG 引脚选择测试模式。器件保护熔丝被连接至 TEST 上。 编程及测试期间的 Spy-Bi-Wire 测试时钟输入

而 MSP430 具有一种运行模式及 5 种可利用软件来选择的低功耗操作模式。一个中断事件能够将器件从任一低功耗模式唤醒、处理请求、并在接收到来自中断程序的返回信号时恢复至低功耗模式。

以下 6 种操作模式可利用软件来配置：

- 激活模式(AM)
  - 所有时钟处于激活状态
- 低功耗模式 0 (LPM0)
  - CPU 被禁用
  - ACLK 和 SMCLK 仍然有效，MCLK 被禁用
- 低功耗模式 1 (LPM1)
  - CPU 被禁用
  - ACLK 和 SMCLK 仍然有效，MCLK 被禁用
  - 如果 DCO 不是在激活模式下被使用，则 DCO 的 dc 生成器被禁用
- 低功耗模式 2 (LPM2)
  - CPU 被禁用
  - MCLK 和 SMCLK 被禁用
  - DCO 的 dc 生成器保持启用
  - ACLK 保持激活
- 低功耗模式 3 (LPM3)
  - CPU 被禁用
  - MCLK 和 SMCLK 被禁用
  - DCO 的 dc 生成器被禁用
  - ACLK 保持激活
- 低功耗模式 4 (LPM4)
  - CPU 被禁用
  - ACLK 被禁用
  - MCLK 和 SMCLK 被禁用
  - DCO 的 dc 生成器被禁用

- 晶体振荡器被停止

下面，我们将根据 TI 官方例程，分析 MSP430G25x3 单片机的常用片内外设及其功能。

1. GPIO

1.1 GPIO 简述

MSP430G2553 共有两个通用数字端口 P1 和 P2。

端口 P1 和 P2 具有输入/输出\中断和外部模块功能，这些功能可以通过它们各自的 7 个控制寄存器的设置来实现。

1 . PxDIR 输入 / 输出方向寄存器

相互独立的 8 位分别定义了 8 个引脚的输入 / 输出方向。8 位在 PUC 后都被复位。使用时先根据需要定义端口的方向以满足设计者要求。

0 : I/O 引脚被切换成输入模式;

1 : I/O 引脚被切换成输出模式。

2 .PxIN 输入寄存器

输入寄存器是 CPU 扫描 I/O 引脚信号的只读寄存器。通过读取该寄存器的内容获取 I/O 端口的输入信号。此时引脚的方向必须选定为输入。读出时，该引脚的方向寄存器必须设置为输入模式。

3 .PxOUT 输出寄存器

该寄存器为 I/O 端口的输出缓冲寄存器。其内容可以像操作内存数据一样写入，以达到改变 I/O 口状态的目的。在读取时输出缓存的内容与引脚方向定义无关。改变方向寄存器的内容, 输出缓存的内容不受影响。

4 .PxIE 中断使能寄存器

该寄存器的各引脚都有一位用以控制该引脚是否允许中断, 该寄存器中

0 :禁止该位中断;

1 :允许该位中断。

5 . PxIES 中断触发沿选择寄存器

如果允许 Px 口的某个引脚中断，还需定义该引脚的中断触发沿。该寄存器的 8 位分别定义了 Px 口的 8 个引脚的中断触发沿。

0 :上升沿使相应标志置位;

1 :下降沿使相应标志置位。

6 . PxIFG 中断标志寄存器

该寄存器有 8 个标志位, 它们含有相应引脚是否有待处理中断的信息，即

相应引脚是否有中断请求。如果 Px 的某个引脚允许中断，同时选择上升沿，则当该引脚发生由低电平向高电平跳变时，PxIFG 的相应位就会置位，表明在该引脚上有中断事件发生。

0 :没有中断请求；

1 :有中断请求。

7 . PxSEL 功能选择寄存器

P1 和 P2 两端口还有其他片内外设功能，考虑减少引脚，将这些功能与芯片外的联系通过复用 P1 和 P2 引脚的方式来实现。PxSEL 用来选择引脚的 I/O 端口功能与外围模块功能。

0 :选择引脚为 I/O 端口；

1 :选择引脚为外围模块功能。

我们以 msp430g2xx3_1 为例，解析 GPIO 的使用。

1.2 LED 红灯闪亮实验

首先，我们来展示一下如何打开并使用 TI 的例程以及相关资源。

●STEP 1

打开 CCS，我们使用的是 CCS5.4.0。

打开的过程中，会出现图 1.2-1 所示对话框。

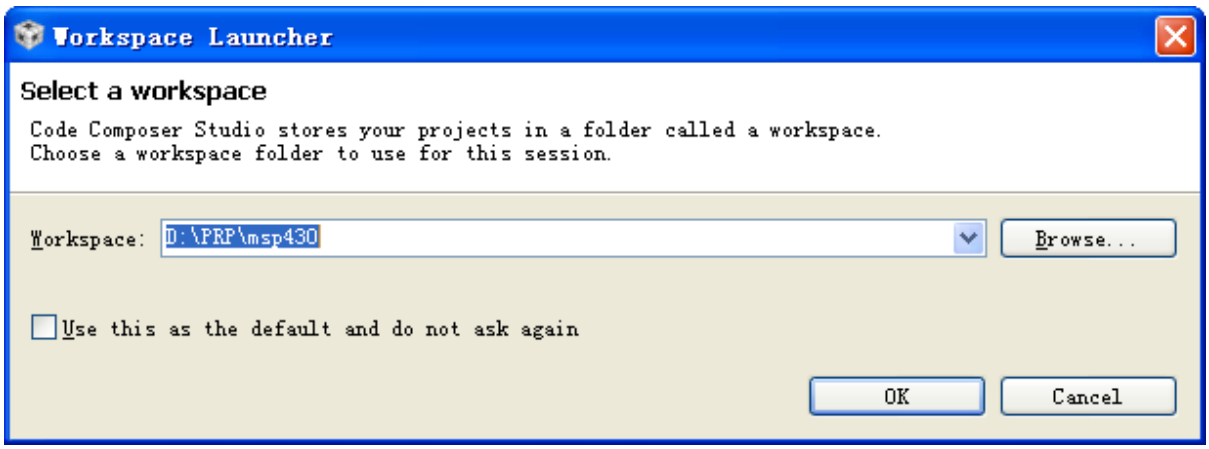


图 1.2-1

此时，需要我们选择工作空间。注意，一般需要放在全英文路径下，也就是说，文件夹需要英文命名。

选定好 Workspace 之后，点击 OK。

进入后，我们可以看到如图 1.2-2 所示的 TI Resource Explorer。

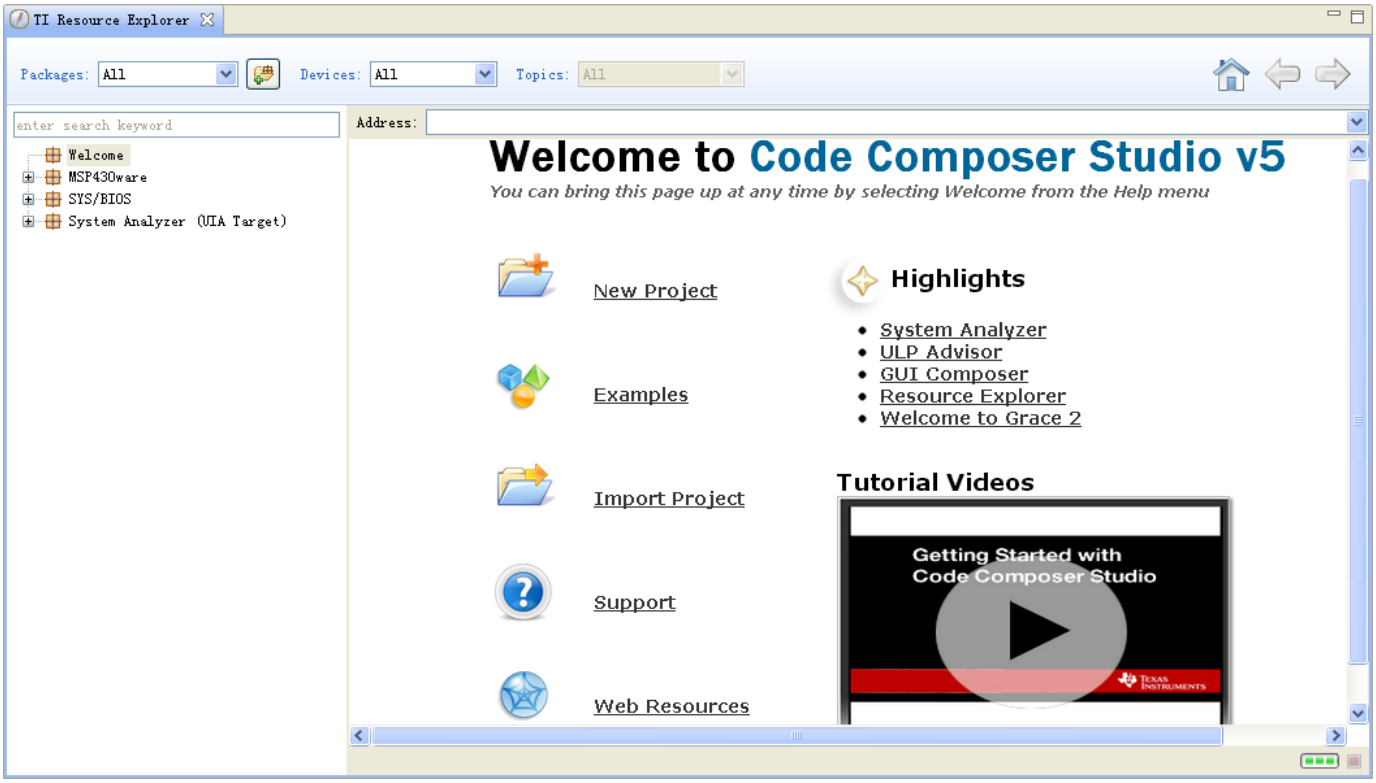


图 1.2-2

在 TI Resource Explorer 中，我们可以点击 New Project，新建工程项目；点击 Examples 加载例程；点击 Import Project 添加已有项目；还可以通过 Support、Web Resources、Tutorial Videos 等，找到各种各样的资源、信息。

●STEP 2

接下来，我们添加一个例程 msp430g2xx3_1。

依次单击 MSP430ware、Devices、MSP430G2xx，如图 1.2-3。

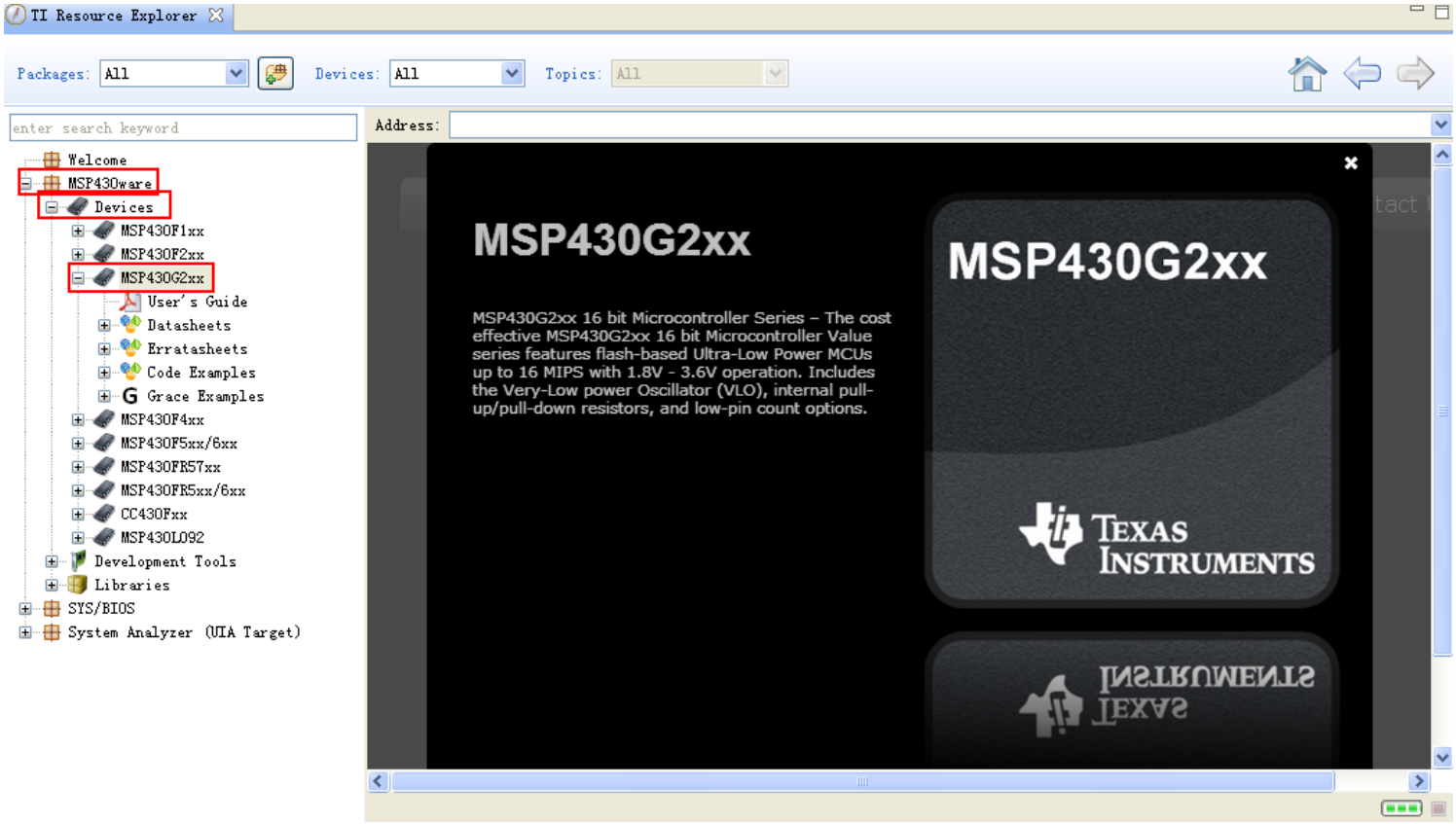


图 1.2-3

在图 1.2-3 中，我们可以看到，在这里，我们可以方便地查看 User's Guide、Datasheets、 Code Examples 等各种资源。

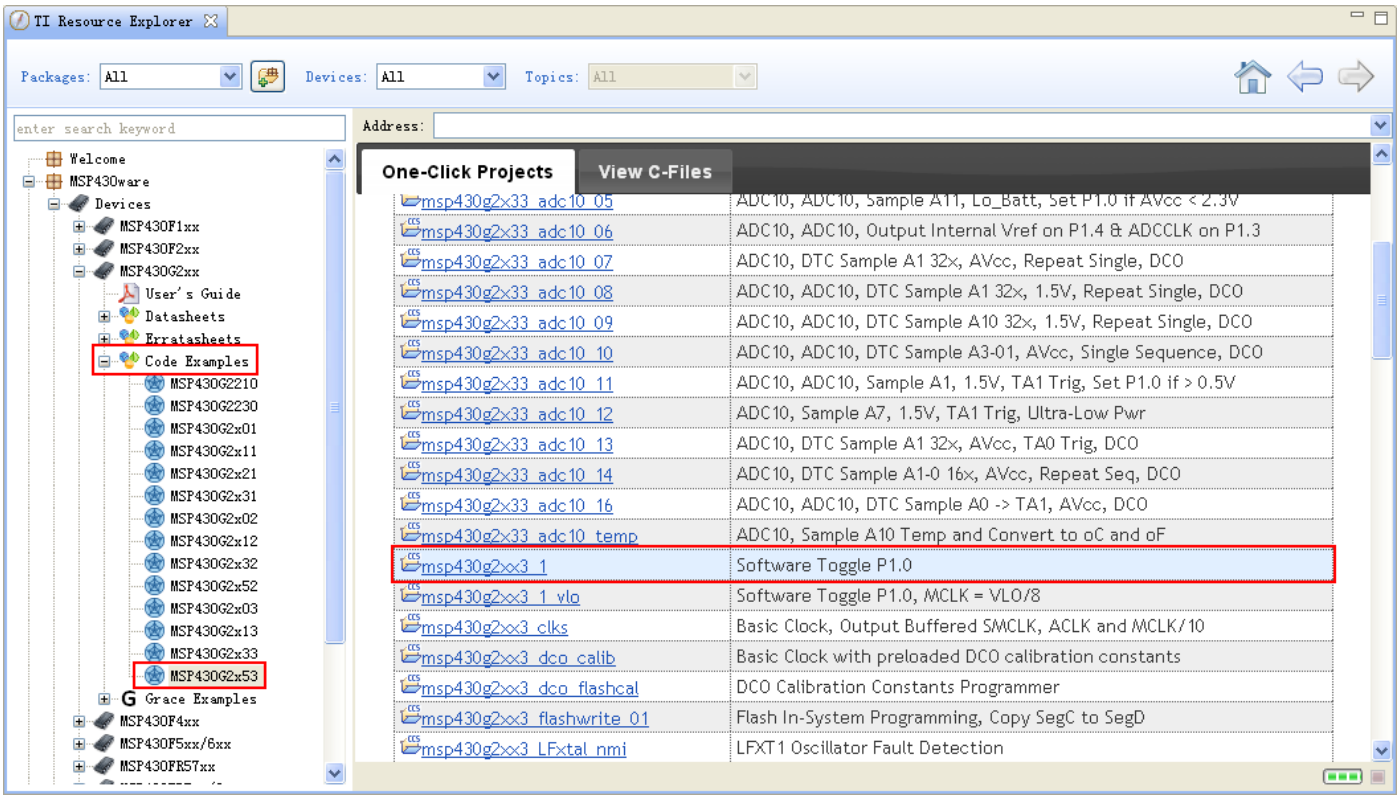


图 1.2-4

接下来，依次单击 Code Examples、MSP430G2x53，在右侧，可以找到我们需要的例程 msp430g2xx3_1。单击之，出现如图 1.2-5 所示对话框。

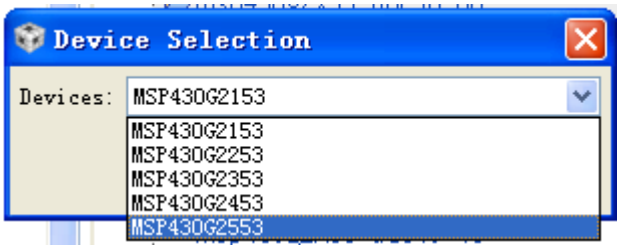


图 1.2-5

选定器件为 MSP430G2553，于是，在右侧，我们可以看到，该例程工程文件已经加载。

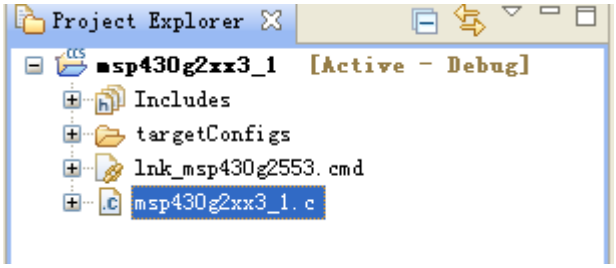


图 1.2-6

点开该文件夹，再双击 msp430g2xx3_1.c，就可以看到程序了。

```
//******************************************************************************
//  MSP430G2xx3 Demo - Software Toggle P1.0
//
//  Description; Toggle P1.0 by xor'ing P1.0 inside of a software loop.
//  ACLK = n/a, MCLK = SMCLK = default DCO
//
//               MSP430G2xx3
//            -----------------
//        /|\|              XIN|-
//         | |                 |
//         --|RST          XOUT|-
//           |                 |
//           |           P1.0|-->LED
//
//  D. Dang
//  Texas Instruments, Inc
//  December 2010
//   Built with CCS Version 4.2.0 and IAR Embedded Workbench Version: 5.10
//******************************************************************************
```

```c
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer
  P1DIR |= 0x01;                            // Set P1.0 to output direction

  for (;;)
  {
    volatile unsigned int i;

    P1OUT ^= 0x01;                          // Toggle P1.0 using exclusive-OR

    i = 50000;                              // Delay
    do (i--);
    while (i != 0);
  }
}
```

●STEP 3

编译与程序下载。

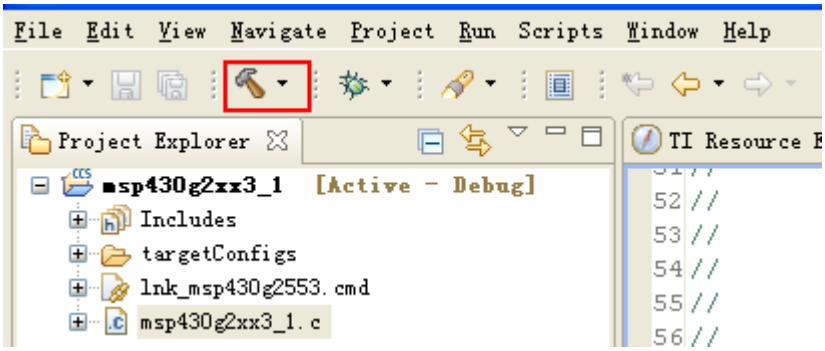


图 1.2-7

单击锤子状图标，开始进行编译。

之后，点击虫子状图标，开始程序下载与 Debug。

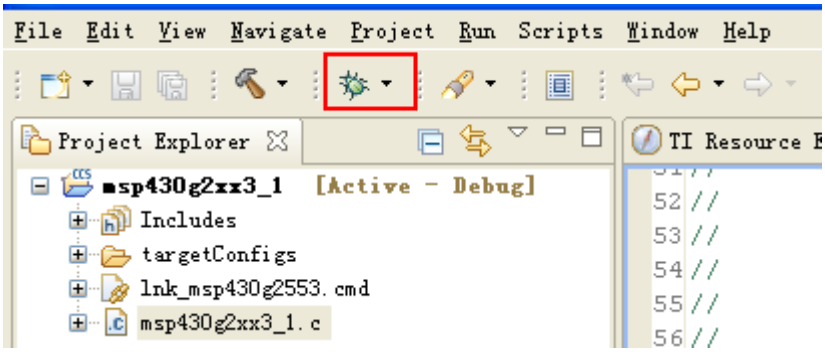


图 1.2-8

之后，在 Debug 对话框中，点击 Resume 按键，即可下载程序。

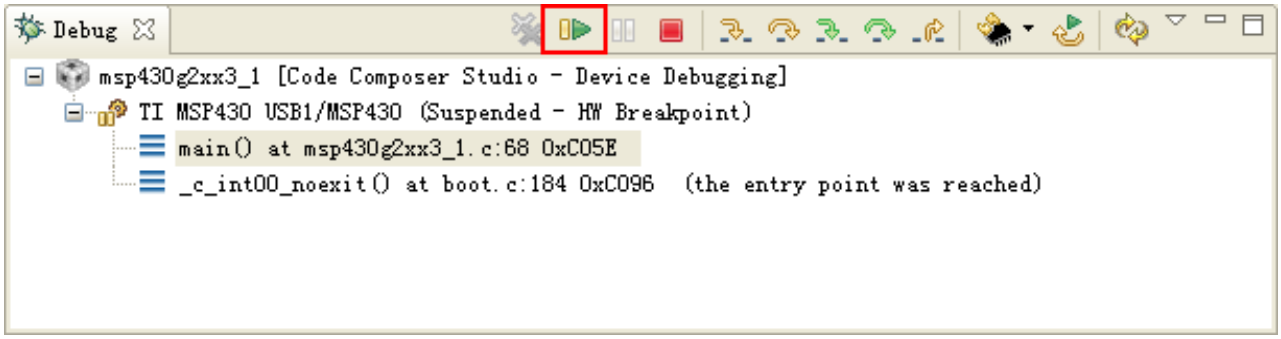


图 1.2-9

我们可以看到，Launchpad 左下角的红色 LED 灯闪亮起来，如图 1.2-10.

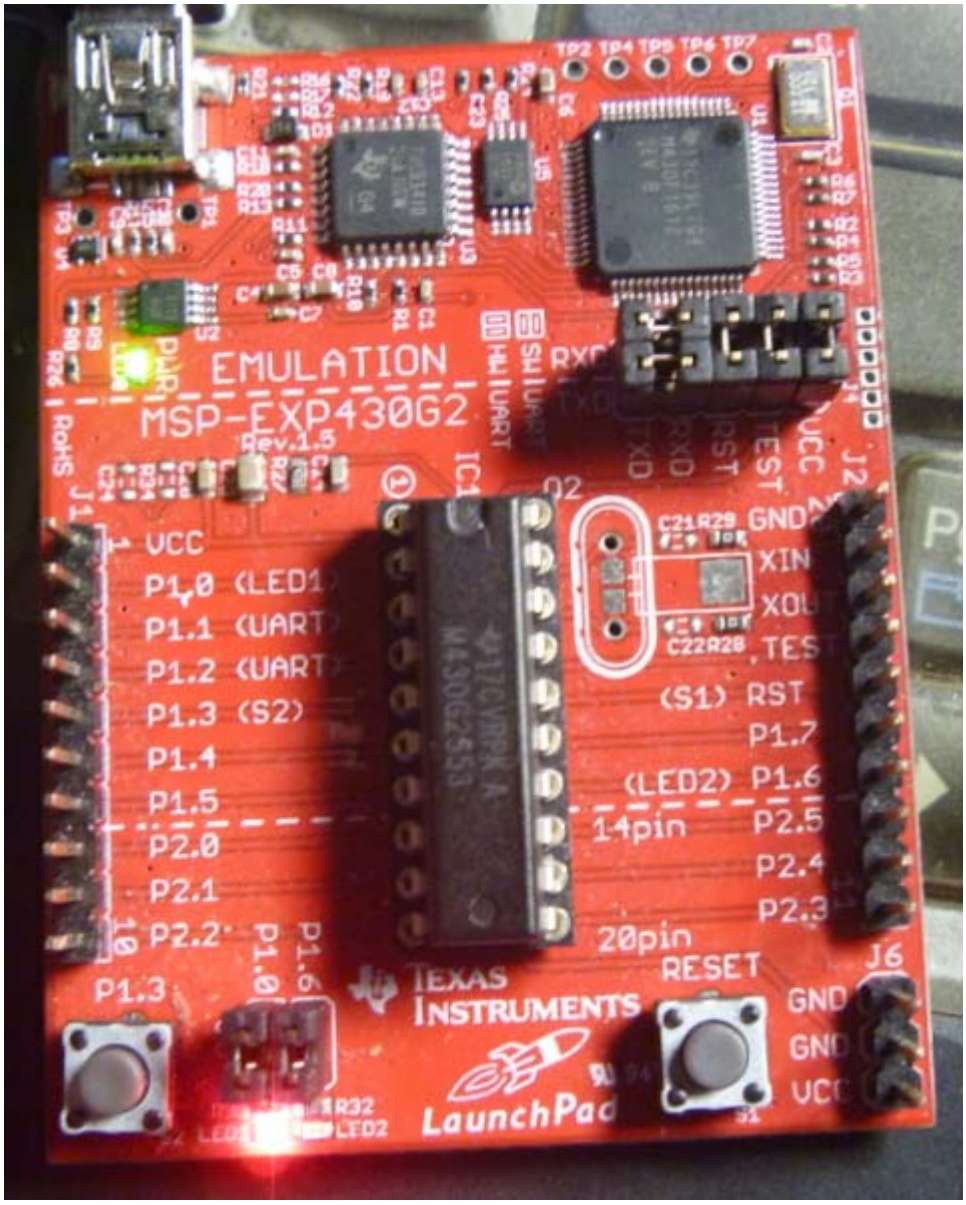


图 1.2-9

我们再看程序主体部分

```c
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer
  P1DIR |= 0x01;                            // Set P1.0 to output direction

  for (;;)
  {
    volatile unsigned int i;

    P1OUT ^= 0x01;                          // Toggle P1.0 using exclusive-OR

    i = 50000;                              // Delay
    do (i--);
    while (i != 0);
  }
}
```

主函数中，首先将看门狗定时器停止，否则看门狗定时器将周期性复位CPU。而接下来，由于要控制 LED1，也就是 P1.0 的输出，因此要将控制该口输入输出的 P1DIR 第 0 位置高。之后，在一个死循环中，每一次，都将 P1OUT 的第 0 位与 1 做异或运算，相当于取反。这样，LED1 一段时间内亮，一段时间内灭，就形成了不断闪亮的效果。

2. Timer

2.1 Timer 概述

定时器是单片机中非常重要的资源，可以用来实现定时控制、延迟、频率测量、脉宽测量、信号产生，作为串行通信接口的可编程波特率发生器，在多任务系统中也可以用来作为中断信号实现程序切换。MSP430 系列单片机的定时器资源非常丰富，包括看门狗定时器(WDT)、定时器 A(Timer_A)和定时器 B (Timer_B)等。器件因系列不同可能包含这些模块的全部或部分。这些模块除了具有定时功能外, 还各自具有一些特殊的用途, 在应用中应根据需要选择合适的定时器模块。MSP430 系列定时器模块功能如下：

看门狗定时器：基本定时；当程序发生错误时执行一个受控的系统重启动。

定时器 A：基本定时;；支持同时进行多种时序控制、多个捕获/比较功能和多种输出波形(PWM)，可以以硬件方式支持串行通信。

定时器 B：基本定时；功能基本同定时器 A，但比定时器 A 更灵活，功能更强大。

下面，以 Timer_A 为例。

Timer_A 分为三部分：主计数器、比较捕获模块和输出单元。主计数器负责定时，时或计数。计数值（TAR 寄存器的值）被送到各比较捕获模块中，它们可以在无需 CPU 干预的情况下根据触发条件与计数器值自动完成某些测量和输出功能。只需定时或计数功能时，可以只使用主计数器部分。在 PWM 调制，利用捕获测量脉宽，周期等应用之中还需要捕获比较模块配合。

与 Timer_A 定时器中的主计数器相关的控制位都位于 TACTL 寄存器中，主计数器的计数数值存放与 TAR 寄存器中。每个比较捕获寄存器 TACCRx（x=0，1，2）。在一般定时器应用中，TACCRx 可提供额外的定时中断触发条件；在 PWM 输出模式之下，TACCRx 可用于设置周期和占空比；在捕获模式下，TACCRx 存放捕获结果。

Timer_A 结构如图 2.1-1 所示。

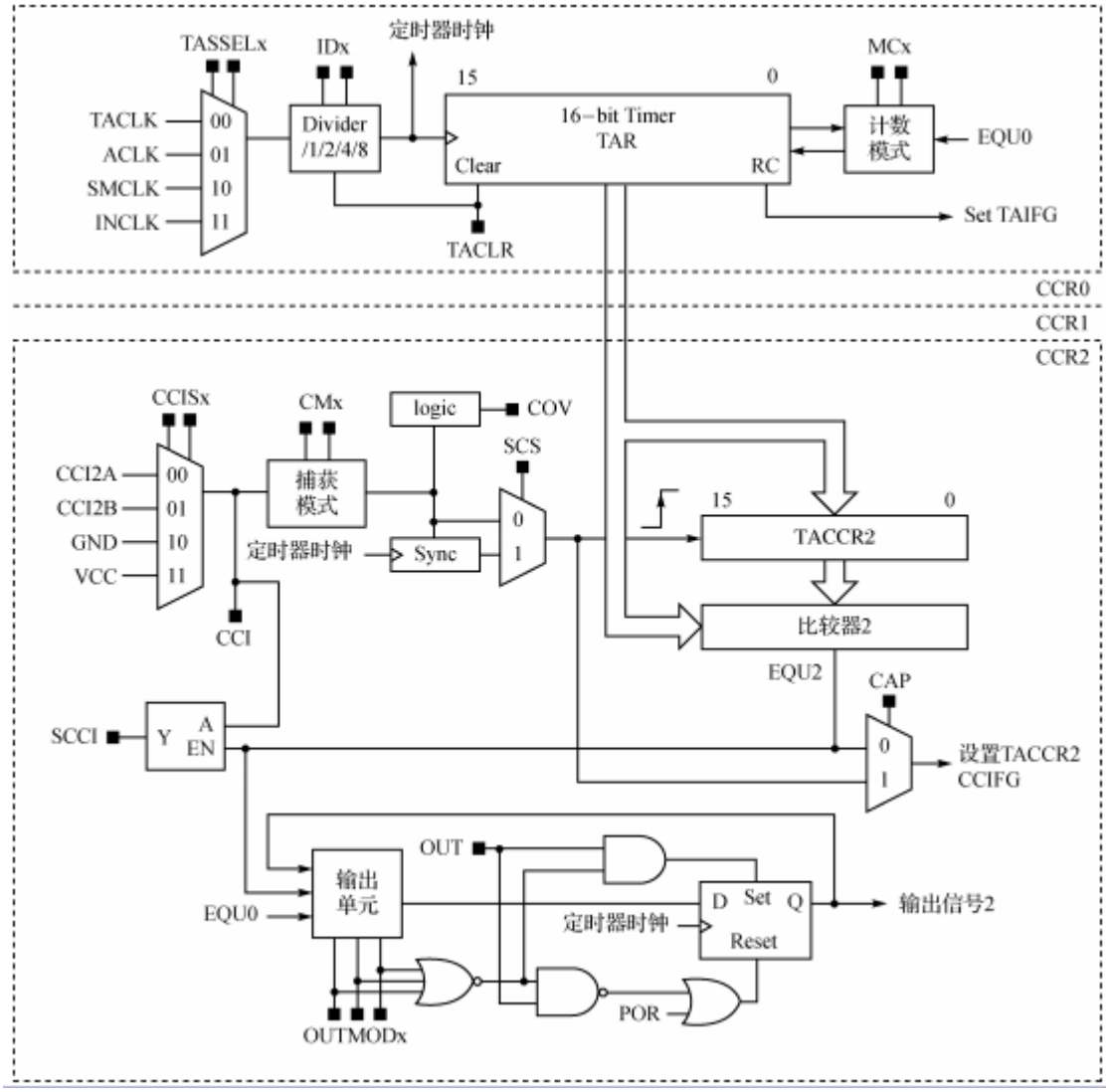


图 2.1-1

而 Timer_A 共有四种工作模式：停止模式、增计数模式、连续计数模式和增减计数模式。

停止模式，MC1=0，MC0=0，定时器暂停工作，但所有寄存器保持当前值，并不复位。

增计数模式，MC1=1，MC0=0，计数器的值从 0 一直到 TACCR0，再如此往复，如图 2.1-2 所示。

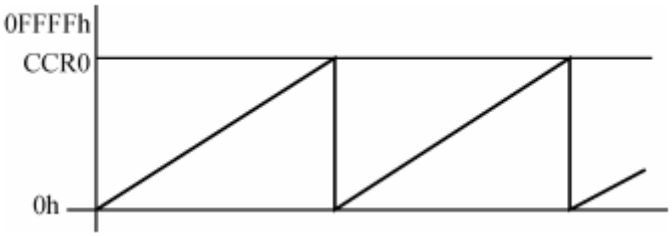


图 2.1-2

连续计数模式，MC1=0，MC0=1，计数器的值从 0 一直到 0FFFFh，再如此往复，如图 2.1-3 所示。

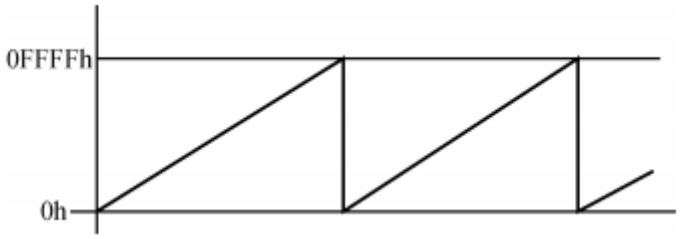


图 2.1-,3

增减计数模式，MC1=1，MC0=1，计数器的值从 0 一直到 TACCR0，再减为 0，如此往复，如图 2.1-4 所示。

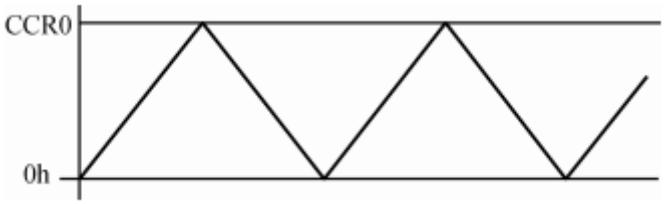


图 2.1-4

而输出单元，可用于输出多种信号。在增计数模式下，七种输出模式波形如图 2.1-5 所示。

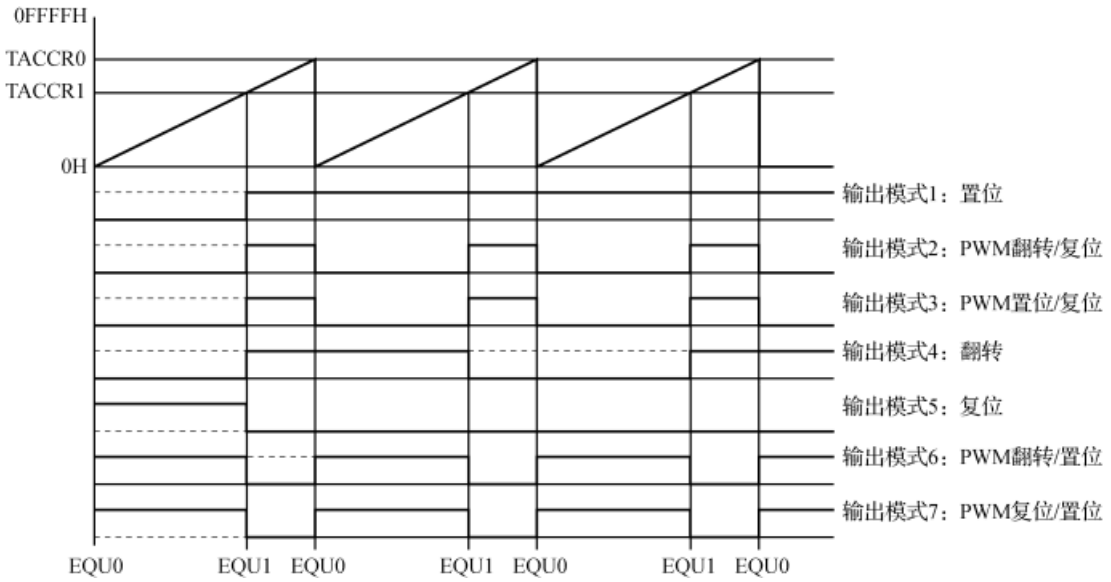


图 2.1-5

我们以 msp430g2xx3_ta_07 和 msp430g2xx3_ta_16 为例，分析定时中断以及 PWM 波的使用。

2.2 Timer_A 定时中断

打开例程 msp430g2xx3_ta_07，程序主要部分如下。

```
//******************************************************************************
//  MSP430G2xx3 Demo - Timer_A, Toggle P1.0-2, Cont. Mode ISR, DCO SMCLK
//
//  Description: Use Timer_A CCRx units and overflow to generate four
//  independent timing intervals. For demonstration, CCR0 and CCR1 output
//  units are optionally selected with port pins P1.1 and P1.2 in toggle
//  mode. As such, these pins will toggle when respective CCRx registers match
//  the TAR counter. Interrupts are also enabled with all CCRx units,
//  software loads offset to next interval only - as long as the interval offset
//  is aded to CCRx, toggle rate is generated in hardware. Timer_A overflow ISR
//  is used to toggle P1.0 with software. Proper use of the TA0IV interrupt
//  vector generator is demonstrated.
//  ACLK = n/a, MCLK = SMCLK = TACLK = default DCO ~1MHz
//  As coded and assuming ~1MHz DCO, toggle rates are:
//  P1.1 = CCR0 ~ 1MHz/(2*200) ~2500Hz
//  P1.2 = CCR1 ~ 1MHz/(2*1000) ~500Hz
//  P1.0 = overflow ~ 1MHz/(2*65536) ~8Hz
//
//            MSP430G2xx3
//         -----------------
//      /|\|            XIN|-
//       | |               |
//       --|RST        XOUT|-
//         |               |
//         |       P1.1/TA0|--> CCR0
//         |       P1.2/TA1|--> CCR1
//         |           P1.0|--> Overflow/software
//
//  D. Dang
//  Texas Instruments Inc.
//  December 2010
//   Built with CCS Version 4.2.0 and IAR Embedded Workbench Version: 5.10
//******************************************************************************

#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P1SEL |= 0x06;                           // P1.1 - P1.2 option select
  P1DIR |= 0x07;                           // P1.0 - P1.2 outputs
```

```
  P1DIR |= 0x07;                            // P1.0 - P1.2 outputs
  CCTL0 = OUTMOD_4 + CCIE;                   // CCR0 toggle, interrupt enabled
  CCTL1 = OUTMOD_4 + CCIE;                   // CCR1 toggle, interrupt enabled
  TACTL = TASSEL_2 +  MC_2 + TAIE;           // SMCLK, Contmode, int enabled

  _BIS_SR(LPM0_bits + GIE);                  // Enter LPM0 w/ interrupt
}

// Timer A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A0 (void)
{
  CCR0 += 200;                              // Add Offset to CCR0
}

// Timer_A2 Interrupt Vector (TA0IV) handler
#pragma vector=TIMER0_A1_VECTOR
__interrupt void Timer_A1(void)
{
  switch( TA0IV )
  {
  case  2: CCR1 += 1000;                    // Add Offset to CCR1
          break;
  case 10: P1OUT ^= 0x01;                   // Timer_A3 overflow
          break;
 }
}
```

本程序中，将 P1.1 和 P1.2 分别设置为 CCR0 和 CCR1 的匹配翻转模式，也就是说，计数值匹配时，这两个脚电平翻转。而在程序中，每次中断时，对 CCR0 加 200，对 CCR1 加 1000，因此 P1.1 和 P1.2 的翻转频率分别为

$$f_1 = \frac{1MHz}{2\times 200} = 2500Hz$$

$$f_2 = \frac{1MHz}{2\times 1000} = 500Hz$$

而对于 P1.0，采用异或运算使之翻转，在程序中，发生溢出中断时才对 P1.0 进行翻转操作，因此，其翻转频率为

$$f_0 = \frac{1MHz}{2\times 65535} = 7.6Hz$$

开始实验后，我们除了看到红色 LED 灯快速闪亮以外，将示波器探头分别放在 P1.0、P1.1 和 P1.2 上，波形分别如图 2.1-6、图 2.1-7、图 2.1-8 所示。示波器测得频率与计算值相近。

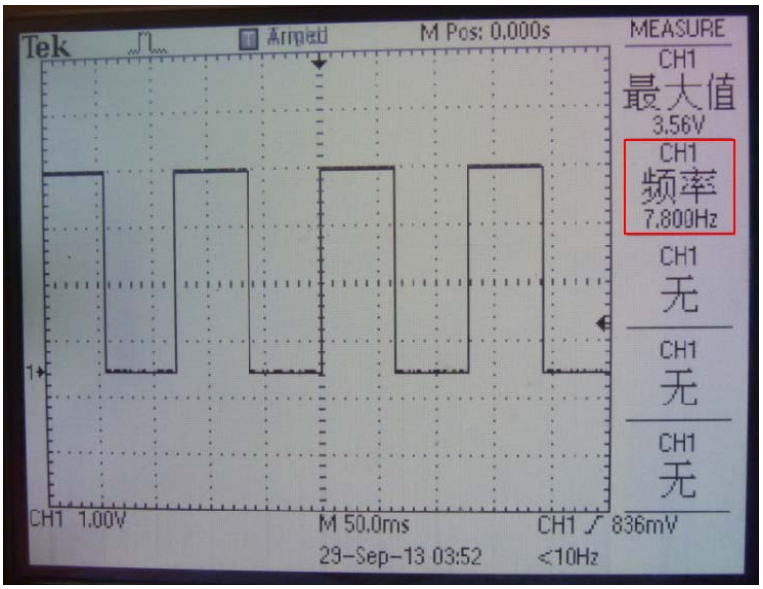


图 2.1-6

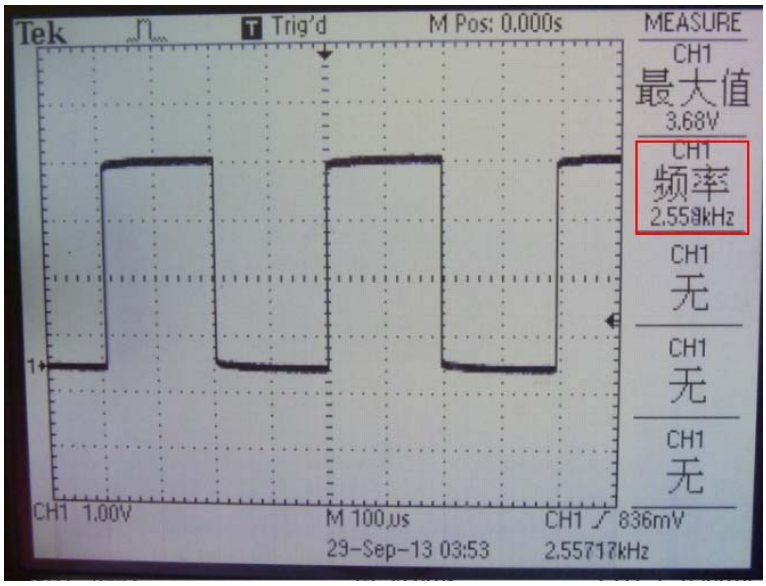


图 2.1-7

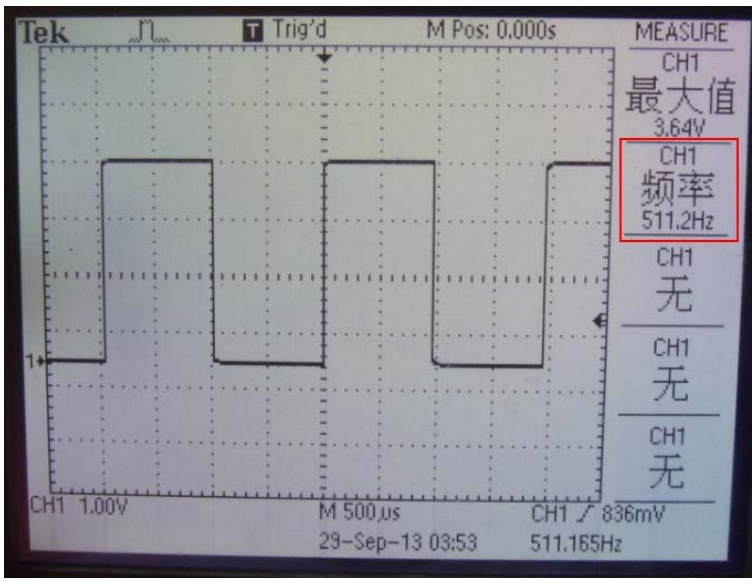


图 2.1-8

2.3 Timer_A 输出 PWM 波

打开例程 msp430g2xx3_ta_16，程序主要部分如下。

```
//******************************************************************************
//  MSP430G2xx3 Demo - Timer_A, PWM TA1-2, Up Mode, DCO SMCLK
//
//  Description: This program generates one PWM output on P1.2 using
//  Timer_A configured for up mode. The value in CCR0, 512-1, defines the PWM
//  period and the value in CCR1 the PWM duty cycles.
//  A 75% duty cycle on P1.2.
//  ACLK = na, SMCLK = MCLK = TACLK = default DCO
//
//            MSP430G2xx3
//          -----------------
//      /|\|              XIN|-
//       | |                 |
//       --|RST          XOUT|-
//         |                 |
//         |       P1.2/TA1|--> CCR1 - 75% PWM
//
//  D. Dang
//  Texas Instruments, Inc
//  December 2010
//   Built with CCS Version 4.2.0 and IAR Embedded Workbench Version: 5.10
//******************************************************************************

#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P1DIR |= 0x0C;                            // P1.2 and P1.3 output
  P1SEL |= 0x0C;                            // P1.2 and P1.3 TA1/2 options
  CCR0 = 512-1;                             // PWM Period
  CCTL1 = OUTMOD_7;                          // CCR1 reset/set
  CCR1 = 384;                               // CCR1 PWM duty cycle
  TACTL = TASSEL_2 + MC_1;                  // SMCLK, up mode

  _BIS_SR(CPUOFF);                          // Enter LPM0
}
```

程序采用增计数模式，上限值为 511，而 CCR1 为 384，模式 7，因此输出 PWM 波占空比为 $D=384/511=75.1\%$，而频率为 $f=1MHz/511=2KHz$。

实验输出波形如图 2.1-9 所示。

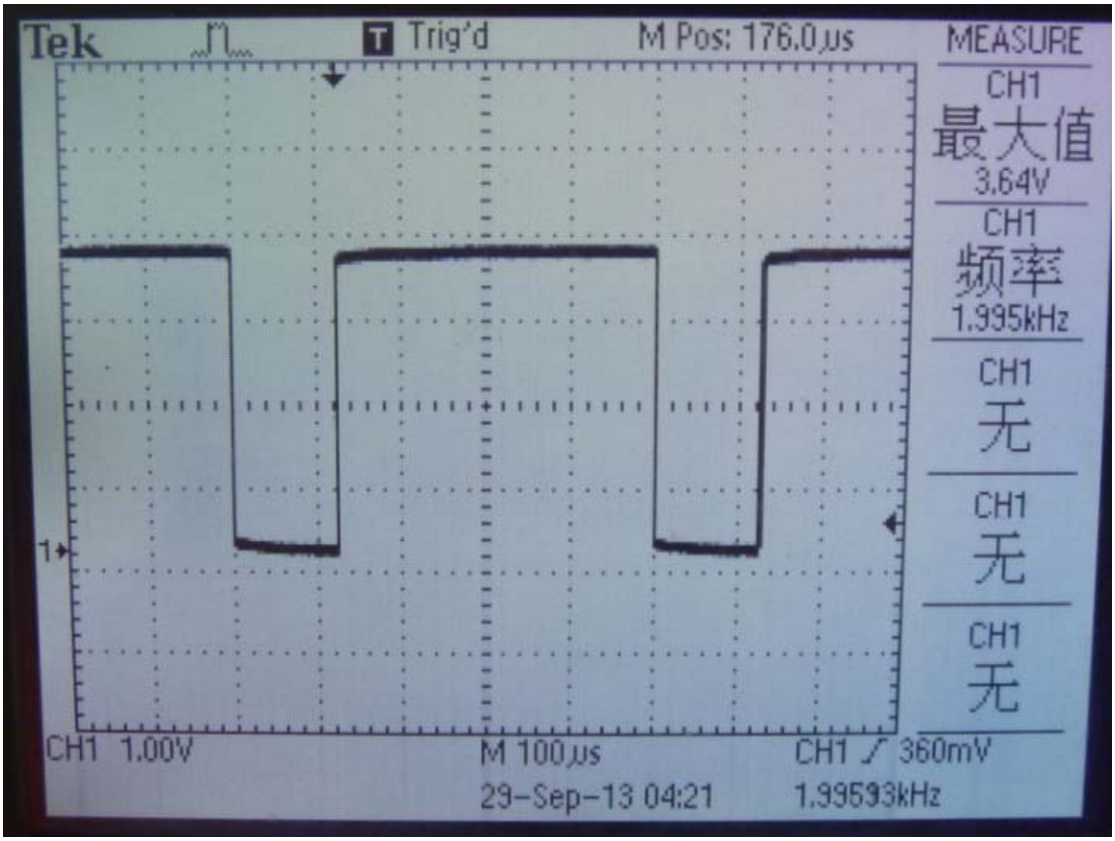

Tek
Trig'd
M Pos: 176.0µs
MEASURE
CH1
最大值
3.64V
CH1
频率
1.995kHz
CH1
无
CH1
无
CH1
无
CH1 1.00V
M 100µs
CH1 ⁄ 360mV
29-Sep-13 04:21
1.99533kHz

图 2.1-9

3. USART

3.1 USART 概述

MSP430 的 USART 模块配置为异步模式时的结构如图 3.1-1 所示，该模块主要包含 4 个部分：（1）用来控制串行通信数据接收和发送速率的波特率部分。（2）用来接收串行输入的接收部分，在接收时，当移位寄存器将接收来的数据流组合满一个字节后，就保存到接收缓存 UxRXBUF 中。（3）用来发送串行输出数据的发送部分，在发送时，将发送缓存 UxTXBUF 内的数据逐位送到发送端口。发送和接收两个移位寄存的移位时钟都是波特率发生器产生的时钟信号 BITCLK。（4）用来完成并-串转换和串-并转换的接口部分。

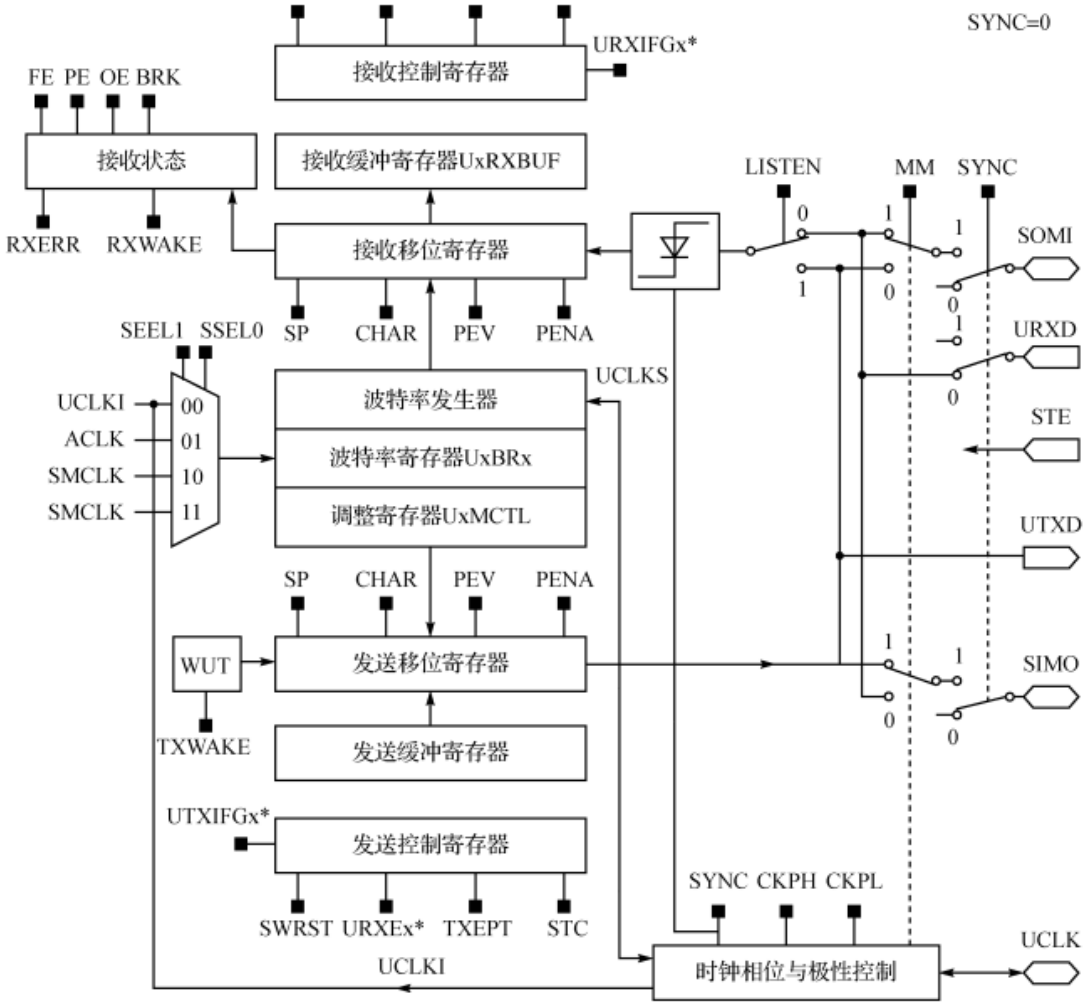


图 3.1-1

以 msp430g2xx3_uscia0_uart_01_9600 为例，分析 USART 的使用。

3.2 USART 实验

例程 msp430g2xx3_uscia0_uart_01_9600 代码如下

```
//******************************************************************************
//   MSP430G2xx3 Demo - USCI_A0, 9600 UART Echo ISR, DCO SMCLK
//
//   Description: Echo a received character, RX ISR used. Normal mode is LPM0.
//   USCI_A0 RX interrupt triggers TX Echo.
//   Baud rate divider with 1MHz = 1MHz/9600 = ~104.2
//   ACLK = n/a, MCLK = SMCLK = CALxxx_1MHZ = 1MHz
//
//             MSP430G2xx3
//          -----------------
//       /|\|              XIN|-
//        | |                 |
//        --|RST          XOUT|-
//          |                 |
//          |    P1.2/UCA0TXD|------------>
//          |                 | 9600 - 8N1
//          |    P1.1/UCA0RXD|<------------
//
//   D. Dang
//   Texas Instruments Inc.
//   February 2011
//   Built with CCS Version 4.2.0 and IAR Embedded Workbench Version: 5.10
//******************************************************************************
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  if (CALBC1_1MHZ==0xFF)                    // If calibration constant erased
  {
    while(1);                               // do not load, trap CPU!!
  }
  DCOCTL = 0;                               // Select lowest DCOx and MODx settings
  BCSCTL1 = CALBC1_1MHZ;                    // Set DCO
  DCOCTL = CALDCO_1MHZ;
  P1SEL = BIT1 + BIT2 ;                     // P1.1 = RXD, P1.2=TXD
  P1SEL2 = BIT1 + BIT2 ;                    // P1.1 = RXD, P1.2=TXD
  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
  UCA0BR0 = 104;                            // 1MHz 9600
  UCA0BR1 = 0;                              // 1MHz 9600
  UCA0MCTL = UCBRS0;                        // Modulation UCBRSx = 1
```

```
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  IE2 |= UCA0RXIE;                          // Enable USCI_A0 RX interrupt

  __bis_SR_register(LPM0_bits + GIE);       // Enter LPM0, interrupts enabled
}

//  Echo back RXed character, confirm TX buffer is ready first
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
  while (!(IFG2&UCA0TXIFG));                // USCI_A0 TX buffer ready?
  UCA0TXBUF = UCA0RXBUF;                    // TX -> RXed character
}
```

由于系统频率为 1MHz，而波特率为 9600，因此波特率因子为 $1M/9600=104$，而 104 小于 256，则只需要在波特率寄存器的低位写入即可。而程序设置了接收中断，在收到一个字符后，立即将之发回去。

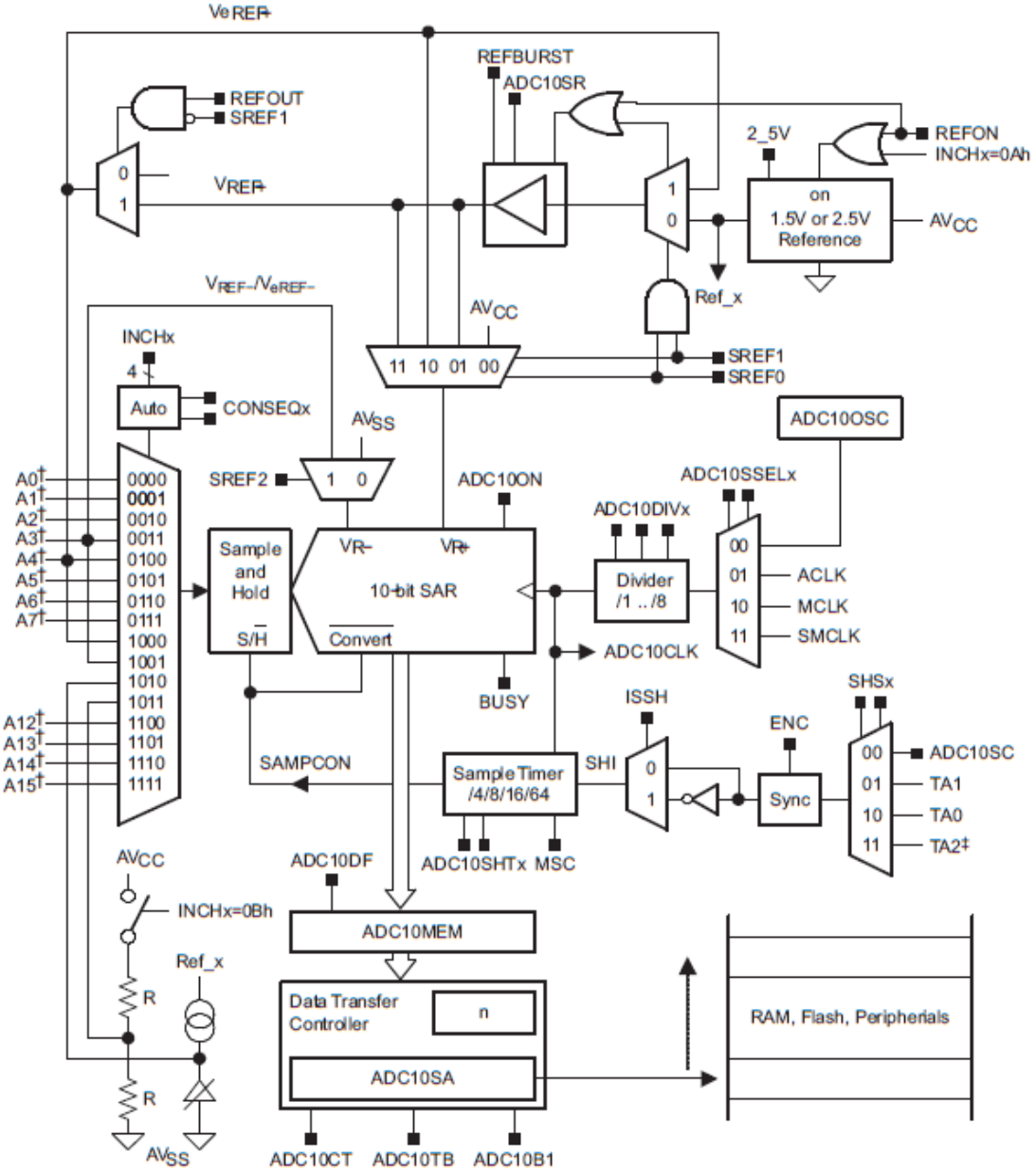
我们使用 sscom4.2 来调试。结果如图 3.1-2 所示。

图 3.1-2

4. ADC

4.1 ADC 概述

ADC10 是 MSP430 单片机的片上模数转换器，其转换位数为 10 比特，该模块内部是一个 SAR 型的 AD 内核，可以在片内产生参考电压，并且具有数据传输控制器。数据传输控制器能够在 CPU 不参与的情况下，完成 AD 数据向内存任意位置的传输。其结构如图 4.1-1 所示。



†Channels A12-A15 are available in MSP430F22xx devices only. Channels A12-A15 tied to channel A11 in other devices. Not all channels are available in all devices.
‡TA1 on MSP430F20x2, MSP430G2x31, and MSP430G2x30 devices

图 4.1-1

它具有如下特点：

●最大转换速率大于 200ksps

●转换精度为 10 位

●采样保持器的采样周期可通过编程设置

●可利用软件或者 TimerA 设置转换初始化

●编程选择片上电压参考源，选择 1.5V 或者 2.5V

●编程选择内部或者外部电压参考源

●8 个外部输入通道

●具备对内部温度传感器、供电电压 VCC 和外部参考源的转换通道

●转换时钟源可选择

●多种采样模式：单通道采样、序列通道采样、单通道重复采样、序列通道重复采样

●提供自动数据传输方法

●ADC 的内核和参考源可分别单独关闭

ADC10 模块工作的核心是 ADC10 的核，即图中的 10-bit SAR。ADC10 的核将模拟量转换成 10 位数字量并储存在 ADC10MEM 寄存器里。这个核使用 VR+和 VR-来决定转换模拟值的高低门限。当输入电压超过 VR+时它会停在 03FFh 上，当输入门限低于 VR-时它会停在 0 上。采样值的计算公式为：

$$N_{ADC} = 1023 \times \frac{V_{IN} - V_{R^-}}{V_{R^+} - V_{R^-}}$$

4.2ADC 实验

我们使用例程 msp430g2x33_adc10_02 代码如下

```
//******************************************************************************
//  MSP430G2x33/G2x53 Demo - ADC10, Sample A1, 1.5V Ref, Set P1.0 if > 0.2V
//
//  Description: A single sample is made on A1 with reference to internal
//  1.5V Vref. Software sets ADC10SC to start sample and conversion - ADC10SC
//  automatically cleared at EOC. ADC10 internal oscillator times sample (16x)
//  and conversion. In Mainloop MSP430 waits in LPM0 to save power until ADC10
//  conversion complete, ADC10_ISR will force exit from any LPMx in Mainloop on
//  reti. If A1 > 0.2V, P1.0 set, else reset.
//
//              MSP430G2x33/G2x53
//           -----------------
//       /|\|              XIN|-
//        | |                 |
//        --|RST          XOUT|-
//          |                 |
//      >---|P1.1/A1      P1.0|-->LED
//
//  D. Dang
//  Texas Instruments Inc.
//  December 2010
//   Built with CCS Version 4.2.0 and IAR Embedded Workbench Version: 5.10
//******************************************************************************
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE;
  __enable_interrupt();                     // Enable interrupts.
  TACCR0 = 30;                              // Delay to allow Ref to settle
  TACCTL0 |= CCIE;                          // Compare-mode interrupt.
  TACTL = TASSEL_2 | MC_1;                  // TACLK = SMCLK, Up mode.
  LPM0;                                     // Wait for delay.
  TACCTL0 &= ~CCIE;                         // Disable timer Interrupt
  __disable_interrupt();
  ADC10CTL1 = INCH_1;                       // input A1
  ADC10AE0 |= 0x02;                         // PA.1 ADC option select
  P1DIR |= 0x01;                            // Set P1.0 to output direction

  for (;;)
```

```
  {
    ADC10CTL0 |= ENC + ADC10SC;           // Sampling and conversion start
    __bis_SR_register(CPUOFF + GIE);       // LPM0, ADC10_ISR will force exi
    if (ADC10MEM < 0x88)                  // ADC10MEM = A1 > 0.2V?
      P1OUT &= ~0x01;                     // Clear P1.0 LED off
    else
      P1OUT |= 0x01;                      // Set P1.0 LED on
  }
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
  __bic_SR_register_on_exit(CPUOFF);       // Clear CPUOFF bit from 0(SR)
}

#pragma vector=TIMER0_A0_VECTOR
__interrupt void ta0_isr(void)
{
  TACTL = 0;
  LPM0_EXIT;                              // Exit LPM0 on return
}
```

对 ADC10CTL0 寄存器写入的几个常量，我们在定义中可以看到

```
#define SREF_1              (1*0x2000u)    /* VR+ = VREF+ and VR- = AVSS */
#define ADC10SHT_2           (2*0x800u)     /* 16 x ADC10CLKs */
#define REFON               (0x020)       /* ADC10 Reference on */
#define ADC10ON             (0x010)        /* ADC10 On/Enable */
#define ADC10IE             (0x008)        /* ADC10 Interrupt Enalbe */
```

可见，ADC 采用内部 2.5V 参考电压，满量程也是 2.5V，而同时开启了 ADC 中断。在该程序中，$0.2/1.5\times1023=136D=0x88$，所以，将结果与 0x88 比较，

为了测试该程序是否正常工作，我们利用变阻器搭建一个分压电路，将变阻器分压接到 P1.1 口，调整变阻器，使得红色 LED 灯由暗刚好变亮时，如图 4.1-2 所示。而使得红色 LED 灯由亮刚好变暗时，如图 4.1-3。


HEWLETT PACKARD
974A
0.2022
10A
nA·µA
Com
V·Ω·⇥

图 4.1-2


HEWLETT PACKARD
974A
0.1932
10A
nA·µA
Com
V·Ω·⇥

图 4.1-3

5. 程序模板

为了便于读者使用 MSP430 Launchpad 进行实验，笔者整理了一些常用外设的驱动、应用程序。

5.1 系统初始化函数

```
/***系统初始化************************************************/
void deviceinit(void)
{
    WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer
    if (CALBC1_8MHZ ==0xFF || CALDCO_8MHZ == 0xFF)
    {while(1);}
    //振荡源为片内RC振荡器，DCO=8MHz,供CPU;      SMCLK=4MHz,供定时器A0、串行口UART
    BCSCTL1 = CALBC1_8MHZ;   // Set range
    DCOCTL = CALDCO_8MHZ;    // Set DCO step + modulation，DCO=8MHz
    BCSCTL3 |= LFXT1S_2;      // LFXT1 = VLO
    IFG1 &= ~OFIFG;          // Clear OSCFault flag
    BCSCTL2 |= DIVS_1;      //  SMCLK = DCO/2 = 4MHz
}
```

在该函数中，我们主要是使用片内 RC 振荡源，将 DCO 设为 8MHz，而系统时钟为 4MHz。

5.2 GPIO 初始化函数

```
/*************************************************************/
//普通IO口设置
void GPIOinit(void)
{
    P1DIR |= 0x41; //P1.0和P1.6设为输出
    P1OUT &= ~0x41;
}
```

在该函数中，我们将 P1.0 和 P1.6 设为输出。

5.3 TimerA 初始化函数

```
/*************************************************************/
//timer初始化
void timerAinit(void)
{
    TA0CTL = TASSEL_2 + MC_1 ;          // 定时器0时钟: SMCLK=4MHz, UP mode
    TA0CCR0 = 20000;                  //计满 20000 一次中断   ，5 ms,200Hz
    CCTL0 = CCIE;                      //CCR0 interrupt enabled
    //PWM初始化，输出脚为1.6
    P1DIR |= 0x40;                            //P1.6 output
    P1SEL |= 0x40;                            //P1.6 TA1/2 options
    CCTL1 = OUTMOD_7;                          // CCR1 reset/set
    CCR1 = 20000;                             // CCR1 PWM duty cycle
}
```

在该函数中，我们设置了 5ms 定时中断，同时将 P1.6 配置为 PWM 波输出端口。

5.4 USART 初始化函数

```
/*************************************************************/
//UART初始化
void usartinit(void)
{
    P1SEL |= BIT1 + BIT2 ;                    // P1.1 = RXD, P1.2=TXD
    P1SEL2 |= BIT1 + BIT2 ;                    // P1.1 = RXD, P1.2=TXD
    UCA0CTL1 |= UCSSEL_2;                    // uart时钟: SMCLK=4MHz
    UCA0BR0 = 161;                           // 设置波特率:9600
    UCA0BR1 = 1;                             // 设置波特率:9600
    UCA0MCTL = UCBRS0;                        // Modulation UCBRSx = 1
    UCA0CTL1 &= ~UCSWRST;                      // **Initialize USCI state machine**
    IE2 |= UCA0RXIE;                         // Enable USCI_A0 RX interrupt
}
```

在该函数中，我们设置将 USART 波特率设置为 9600，同时使能了接收中断。

5.5 ADC 初始化函数

```
/*****************************************************************/
//ADC初始化，输入脚为1.7
void ADCinit(void)
{
    ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE + REF2_5V;
    __delay_cycles(3000);
    ADC10CTL1 = INCH_7;                       // input A7 P1.7为采样输入端
    ADC10AE0 |= 0x80;                         // A7使能
}
```

在该函数中，我们设置 P1.7 配置为 ADC 输入端，采用单端单次采样，内部 2.5V 参考电压。

5.6 USART 接收中断函数

```
//uart串口RX接收到一个字符,产生中断处理
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
    //此处写代码
}
```

5.7 TimerA 定时中断函数

```
//timer中断处理
#pragma vector=TIMER0_A0_VECTOR
__interrupt void ta0_isr(void)
{
    //此处写代码
}
```

5.8 ADC 中断函数

```
// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}
```

5.9 ADC 辅助函数

ADC 读取函数：

```
//ADC读取函数
unsigned int GetADC(void)
{
  ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
  __bis_SR_register(CPUOFF + GIE);        // LPM0, ADC10_ISR will force exi
  return ADC10MEM;
}
```

ADC 数据存储函数：

```
//存储adc数据，必须放在timer中断中自动执行
void StoreADC(void)
{
  ++loop;
  loop = ((loop>19)? 0:loop);
  adcvalue[loop] = GetADC();
}
```

5.10 USART 辅助函数

单字符发送函数：

```
//单个字符发送函数
void putchar(char ch)
{
  while (!(IFG2&UCA0TXIFG));              // USCI_A0 TX buffer ready?
  UCA0TXBUF = ch;
}
```

字符串发送函数：

```
//字符串送函数
void putstring(char *str)
{
    while(*str != '\0')
    {
        putchar(*(str++));
    }
}
```

6. MSP430 与 Labview 协同工作

由于 MSP430 Launchpad 小板过于精简，板上没有数码管、液晶等显示部件，而且能用的按键，除了 RESET 之外，只有一个，其人机交互不方便。

为了更好地与处理器进行信息交流，我们示范一种经济、简单，使用串口进行 MSP430 处理器与上位机之间进行通信的方法——使用 Labview。

本实验中，我们使用 Labview 搭建虚拟仪器平台，在上位机上，对红色 LED 灯，绿色 LED 灯以及 ADC 转换进行控制，并显示 ADC 转换结果。

6.1 MSP430 程序状态机

我们定义如下几种状态：

状态 0：红色 LED 灯、绿色 LED 灯保持上一次状态，单片机不发送任何信息，而等待上位机信息；

状态 L：即将根据指令，决定红色 LED 是熄灭还是点亮；

状态 L1：点亮红色 LED 灯；

状态 L0：熄灭红色 LED 灯；

状态 P：即将根据指令，决定绿色 LED 是熄灭还是渐亮；

状态 P1：绿色 LED 灯渐亮；

状态 P0：熄灭绿色 LED 灯；

状态 D：即将根据指令，决定 ADC 动作；

状态 D1：ADC 活跃，不断返回新值；

状态 D0：ADC 停止返回新值。

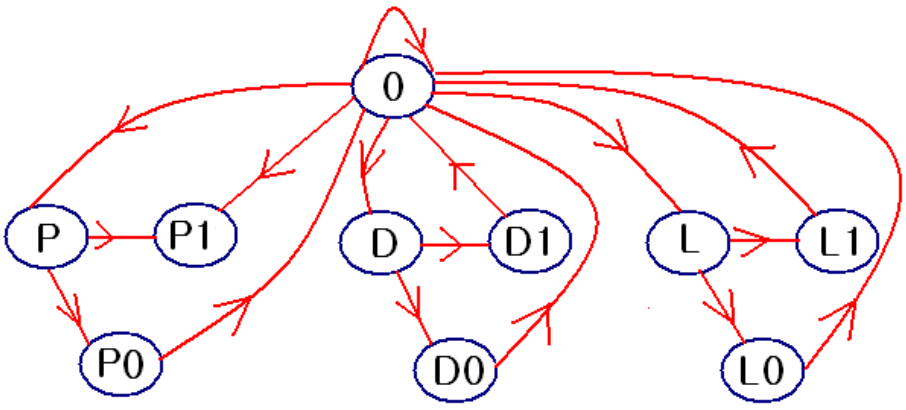


图 6.1-1

而状态转换图如图 6.1-1 所示

6.2 MSP430 程序

```
#include <msp430.h>
/***全局变量***/
unsigned int adcvalue[40] = {0}; //ADC滑动平均
unsigned char loop = 0;   //ADC位置，0-19
unsigned char state=0;
unsigned char inttimes=0;
unsigned char str[6];
char tmpch=0;


/***函数***/
void deviceinit(void);//系统初始化
void usartinit(void);
void timerAinit(void);
void ADCinit(void);
void GPIOinit(void);
unsigned int GetADC(void);//ADC读取函数
void StoreADC(void);//存储adc数据，必须放在timer中断中自动执行
void putchar(char ch);//单个字符发送函数
void putstring(char *str);
void process_s0(char rch);
void process_sL(char rch);
void process_sL1(char rch);
void process_sL0(char rch);
void process_sP(char rch);
void process_sP1(char rch);
void process_sP0(char rch);
void process_sD(char rch);
void process_sD1(char rch);
void process_sD0(char rch);

void main(void)
{
    deviceinit();
    timerAinit();
    ADCinit();
    GPIOinit();
    usartinit();
    _BIS_SR(GIE); //开全局中断

    while(1)
```

```
    {
        switch(state)
        {
        case 2:{if(CCR1<20000) CCR1+=50;else CCR1=0;}break;
        case 3:CCR1=0;break;
        case 5:P1OUT |=0x01;break;
        case 6:P1OUT &=~0x01;break;
        case 8:
        {
            if(inttimes>=100)
            {
                inttimes=0;
                unsigned int tmp=0;
                unsigned char itmp;
                for(itmp=0;itmp<40;++itmp) {tmp += adcvalue[itmp];}
                tmp /= 40;
                putstring("DR");
                str[3] = (tmp/1000)%10 +'0';
                str[2] = (tmp/100)%10 +'0';
                str[1] = (tmp/10)%10 +'0';
                str[0] = tmp%10 +'0';
                putchar(str[3]);
                putchar(str[2]);
                putchar(str[1]);
                putchar(str[0]);
            }
        }break;
        default:break;
        }
    }
}


/***系统初始化************************************************/
void deviceinit(void)
{
    WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer
    if (CALBC1_8MHZ ==0xFF || CALDCO_8MHZ == 0xFF)
    {while(1);}
    //振荡源为片内RC振荡器，DCO=8MHz,供CPU;      SMCLK=4MHz,供定时器A0、串行口UART
    BCSCTL1 = CALBC1_8MHZ;   // Set range
    DCOCTL = CALDCO_8MHZ;    // Set DCO step + modulation，DCO=8MHz
    BCSCTL3 |= LFXT1S_2;      // LFXT1 = VLO
    IFG1 &= ~OFIFG;          // Clear OSCFault flag
    BCSCTL2 |= DIVS_1;      //  SMCLK = DCO/2 = 4MHz
}
```

```
/*****************************************************************/
//UART初始化
void usartinit(void)
{
    P1SEL |= BIT1 + BIT2 ;                    // P1.1 = RXD, P1.2=TXD
    P1SEL2 |= BIT1 + BIT2 ;                   // P1.1 = RXD, P1.2=TXD
    UCA0CTL1 |= UCSSEL_2;                     // uart时钟: SMCLK=4MHz
    UCA0BR0 = 161;                            // 设置波特率:9600
    UCA0BR1 = 1;                              // 设置波特率:9600
    UCA0MCTL = UCBRS0;                        // Modulation UCBRSx = 1
    UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
    IE2 |= UCA0RXIE;                          // Enable USCI_A0 RX interrupt
}

/*****************************************************************/
//timer初始化
void timerAinit(void)
{
    TA0CTL = TASSEL_2 + MC_1 ;          // 定时器0时钟: SMCLK=4MHz, UP mode
    TA0CCR0 = 20000;                  //计满 20000 一次中断   , 5 ms,200Hz
    CCTL0 = CCIE;                      //CCR0 interrupt enabled
    //PWM初始化，输出脚为1.6
    P1DIR |= 0x40;                            //P1.6 output
    P1SEL |= 0x40;                            //P1.6 TA1/2 options
    CCTL1 = OUTMOD_7;                         // CCR1 reset/set
    CCR1 = 0;                               // CCR1 PWM duty cycle
}

/*****************************************************************/
//ADC初始化，输入脚为1.7
void ADCinit(void)
{
    ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE + REF2_5V;
    __delay_cycles(3000);
    ADC10CTL1 = INCH_7;                       // input A7 P1.7为采样输入端
    ADC10AE0 |= 0x80;                         // A7使能
}

/*****************************************************************/
//普通IO口设置
void GPIOinit(void)
{
    P1DIR |= 0x41; //P1.0和P1.6设为输出
    P1OUT &= ~0x41;
}
```

```
/***中断函数*************************************************************/
//uart串口RX接收到一个字符,产生中断处理
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{

    tmpch = UCA0RXBUF;

    switch (state)
    {
    case 0 : process_s0(tmpch); break;
    case 1 : process_sP(tmpch); break;
    case 2 : process_sP1(tmpch); break;
    case 3 : process_sP0(tmpch); break;
    case 4 : process_sL(tmpch); break;
    case 5 : process_sL1(tmpch); break;
    case 6 : process_sL0(tmpch); break;
    case 7 : process_sD(tmpch); break;
    case 8 : process_sD1(tmpch); break;
    case 9 : process_sD0(tmpch); break;
    default:break;
    }
}

//timer中断处理
#pragma vector=TIMER0_A0_VECTOR
__interrupt void ta0_isr(void)
{
    StoreADC();
    ++inttimes;
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
  __bic_SR_register_on_exit(CPUOFF);       // Clear CPUOFF bit from 0(SR)
}

/**辅助函数*************************************************/
//ADC读取函数
unsigned int GetADC(void)
{
  ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
  __bis_SR_register(CPUOFF + GIE);        // LPM0, ADC10_ISR will force exi
  return ADC10MEM;
```

```
}

//存储adc数据，必须放在timer中断中自动执行
void StoreADC(void)
{
  ++loop;
  loop = ((loop>39)? 0:loop);
  adcvalue[loop] = GetADC();
}

//单个字符发送函数
void putchar(char ch)
{
  while (!(IFG2&UCA0TXIFG));              // USCI_A0 TX buffer ready?
  UCA0TXBUF = ch;
}

void putstring(char *str)
{
    while(*str != '\0')
    {
        putchar(*(str++));
    }
}

void process_s0(char rch)
{
    switch(rch)
    {
    case 'P':state=1;break;
    case 'L':state=4;break;
    case 'D':state=7;break;
    default:state=0;break;
    }
}
void process_sL(char rch)
{
    switch(rch)
    {
    case '1':state=5;break;
    case '0':state=6;break;
    default:state=0;break;
    }
}
void process_sL1(char rch)
{
```

```
    switch(rch)
    {
    case 'P':state=1;break;
    case 'L':state=4;break;
    case 'D':state=7;break;
    default:state=0;break;
    }
}
void process_sL0(char rch)
{
    switch(rch)
    {
    case 'P':state=1;break;
    case 'L':state=4;break;
    case 'D':state=7;break;
    default:state=0;break;
    }
}
void process_sP(char rch)
{
    switch(rch)
    {
    case '1':state=2;break;
    case '0':state=3;break;
    default:state=0;break;
    }
}
void process_sP1(char rch)
{
    switch(rch)
    {
    case 'P':state=1;break;
    case 'L':state=4;break;
    case 'D':state=7;break;
    default:state=0;break;
    }
}
void process_sP0(char rch)
{
    switch(rch)
    {
    case 'P':state=1;break;
    case 'L':state=4;break;
    case 'D':state=7;break;
    default:state=0;break;
    }
```

```
}
void process_sD(char rch)
{
    switch(rch)
    {
    case '1':state=8;break;
    case '0':state=9;break;
    default:state=0;break;
    }
}
void process_sD1(char rch)
{
    switch(rch)
    {
    case 'P':state=1;break;
    case 'L':state=4;break;
    case 'D':state=7;break;
    default:state=0;break;
    }
}
void process_sD0(char rch)
{
    switch(rch)
    {
    case 'P':state=1;break;
    case 'L':state=4;break;
    case 'D':state=7;break;
    default:state=0;break;
    }
}
```

6.3 Labview 上位机界面与程序设计

Labview 前面板如图 6.3-1 所示。

图 6.3-1

前面板上，主要有串口调试部分和 MSP430 Launchpad 控制部分构成。

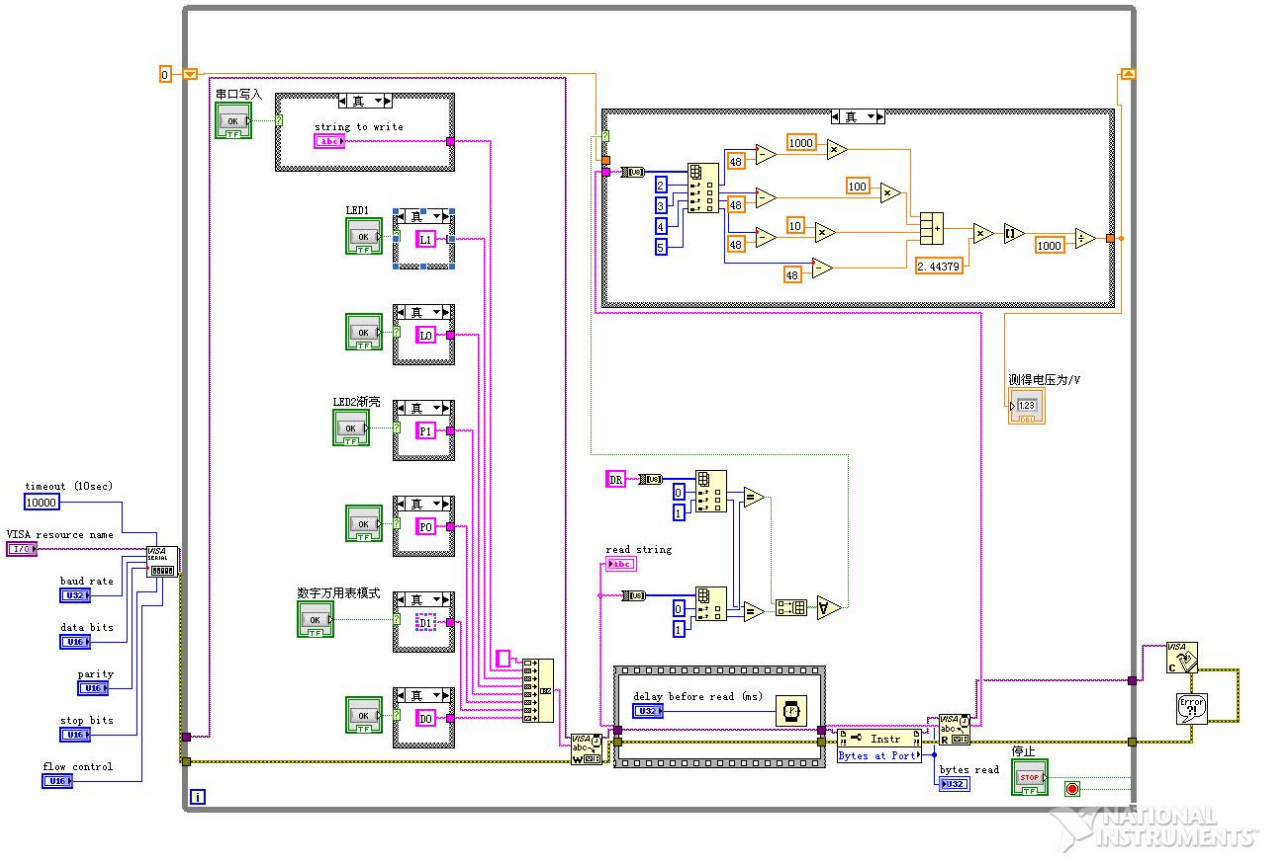


图 6.3-2

程序框图如图 6.3-2 所示。