Application Report
ZHCA429 - 2011 July 20

16 位 MSP430G 系列微处理器的使用扩展

Hanson He Loops Lu South-China MCU FAE

摘要

在嵌入式设计应用中，一般微处理器都具备 A/D、D/A、UART、EEPROM、I2C、SPI 等功能，但在一般低成本的微控制器中，并不完全具备这些功能。MSP430G 系列处理器是 TI 推出的低成本的 16 位处理器，其集成 16 位的 Timer，I2C/SPI 接口，10 位 A/D 等，但在 G2xx1 和 G2xx2 系列中并不具有 EEPROM，D/A，UART 等外设(G2xx3 具有硬件的 UART)。那么在实际应用中如何实现上述功能呢?本文主要以 MSP430G2231 系列为例，讲述了利用内部定时器来模拟 DAC、软件 UART 与 PC 进行通讯，并通过串口对应用程序进行在线升级的方法。本文给出了实现上述功能的硬件电路以及软件代码。实验证明，通过 MSP430G 系列的 16 位定时器可以容易的实现 8 位分辨率的 DAC；通过软件模拟的 UART 能够与 PC 机进行稳定可靠的通讯；通过 BSL 程序可以对用户程序进行板上在线应用编程。最后结合一个实例讲述 MSP430G 系列在汽车车窗以及工业消费类电子产品的实际应用。

目 录

图

附 录

1 系统硬件概述

本系统的框图如图 1 所示。MSP430G2231 微控制器通过 MAX3232 与上位机进行通信，利用系统内部定时器及外部一阶 RC 滤波电路实现一路 8 位 DAC 输出。本文重点介绍 D/A、定时器软件 UART 通信以及通过 Bootstrap 实现应用程序的在线应用编程功能。

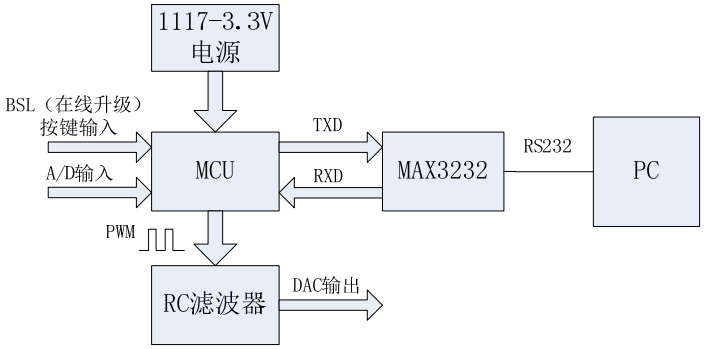

图 1. 系统框图

具体的硬件电路原理图如图 2 所示。由图可见，在 G2231 单片机的最小系统的基础上加入电平转换芯片 MAX3232 用于与 PC 串行通讯；BSL 输入按键用于启动 BSL/用户程序；一阶 RC 对 PWM 进行滤波实现 DAC 功能。使用 MSP430G2231 定时器 A 实现 PWM 波形的输出，PWM 波形从 MSP430G2231 的 P1.2 (第 4 脚) 输出，经 R9 和 C12 对信号进行滤波后，在 C12 正端输出稳定的电压，这样就实现了 8 位精度的 D/A 转换功能。对于 PWM 输出信号，也可在单片机的输出口加一级推挽电路，这样可以实现输出电平的匹配以及加速电容充放电速度。

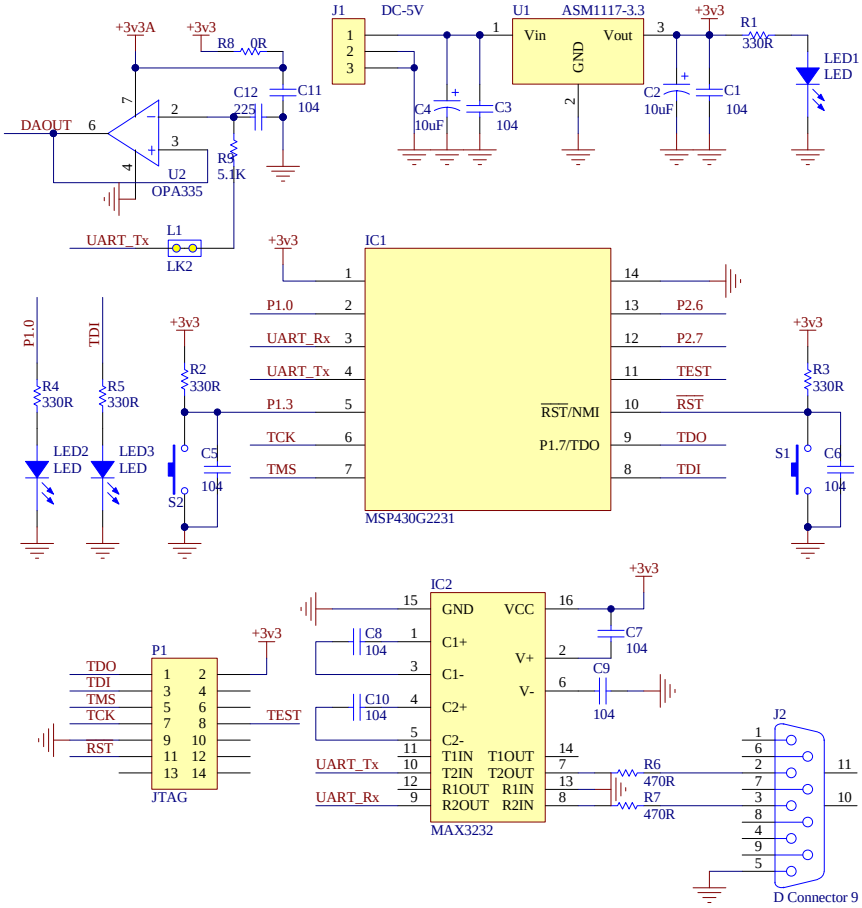

图 2. 硬件电路图

2 TimerA 定时器模拟 DAC 的实现

这一节介绍使用 MSP430G2231 定时器输出的 PWM 来实现 8 位精度的 D/A。利用单片机的 PWM 外接 RC 滤波器实现简单 DAC。由于 DAC 的性能与单片机输出的 PWM 频率，RC 滤波器的参数等相关，这里设定 TimerA 定时器工作在上升模式下，CCR0 控制输出 PWM 频率，CCR1 控制输出 PWM 的占空比。PWM 通过 TimerA 的比较捕获口 TA1 输出。

2. 1 PWM 输出实现 D/A 理论基础

在单片机 PWM 输出口接 RC 滤波器，将 PWM 波形中所含的谐波滤除，只留下 PWM 信号中的直流成分。一般的，D/A 转换输出的电压值可通过公式：输出电压= PWM 波形的占空比*PWM 脉冲峰值。在利用 PWM 实现 DAC 时候，需要注意 PWM 频率的选取和后级滤波器参数的设计对 DAC 的精度以及带宽的影响。PWM 输出滤波器一般可以选择一阶无源滤波器，2 阶无源滤波器，或高阶无源滤波器。这里，根据实际需求，选择一阶 RC 滤波器。

在设计 RC 滤波器时，最关注的是 RC 滤波器的截止频率，这和 PWM 实现的 DAC 的带宽以及输出纹波都有影响。过高的截止频率会使 DAC 输出纹波过大，过低得截止频率会使 DAC 得带宽变窄，响应变慢。因此需要根据实际情况来设置 R 以及 C 的数值，取一个比较好的折中点。

首先设定 PWM 载波的频率。用 MSP430G2231 内部的 DCO，可以获得最高 16MHz 的时钟，这里，要实现 8 位分辨率的 DAC，将 PWM 载波频率设定为 50Khz，那么所需要的最低系统时钟为 50000*256=12800000，即 12.8MHz。由于对带宽没有要求，因此可以将截止频率设置的比较低，这里将截止频率设定为 100Hz，这时候的时间常数 $\tau = R * C = 1.59ms$，通过较低的截止频率可以换取更高的电压输出精度。

2. 2 仿真分析及试验数据

借助 Saber 软件对电路进行模拟仿真分析，可以直观的了解 PWM 实现 D/A 的整个过程。这里，将 PWM 频率设定为 50KHz，将 R 选为 3.3k，C 选择为 0.47uF，这时截止频率为 102Hz，此时 PWM 输出经过 RC 滤波输出如图 3 所示。在 1.59ms 时，它上升至 0.57v，与理论值 0.632*0.90 符合。在输出达到稳定时候，输出的纹波为 0.0089v，符合设计的精度要求。

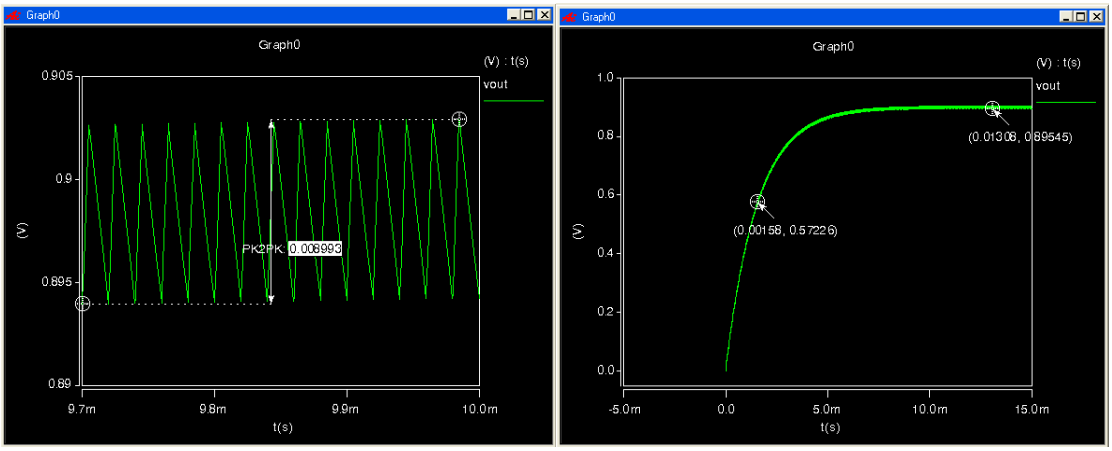

图 3. Saber 电路仿真

这里对上面的仿真进行了验证，测试数据如下所示。实验与仿真证明，当占空比大于 20%时候，理论值与计算值符合，且分辨率可以做到千分之一。

CCR0值	500	510	520	550	560	900	1000
滤波输出电压(V)	1.78	1.8	1.83	1.9	1.94	2.79	3.6
计算输出电压(V)	1.76	1.79	1.83	1.93	1.96	3.16	3.52

CCR0值	550	551	552	553	554	555	556
滤波输出电压(V)	1.9	1.9	1.91	1.91	1.91	1.91	1.92
计算输出电压(V)	1.93	1.93	1.94	1.94	1.94	1.95	1.95

3 TimerA 定时器模拟 UART

实验中利用 UART 通过 I/O 模拟方式来实现 UART 通信，波特率 2400/4800/9600bps 可选。在 MSP430G2231 系列单片机中，需使用一个定时器来实现这个功能。MSP430G2231 的定时器 A 具有 2 个捕获比较端口，可以利用 TimerA 的捕获端口做输入口，比较口做输出口，根据系统设定的波特率来发送或接收字符。在本实例中，利用 P1.1 作为发送（Transmit）口，P1.2 作为接收（Receive）口。发送程序与接收程序流程图如图 4 和图 5 所示。

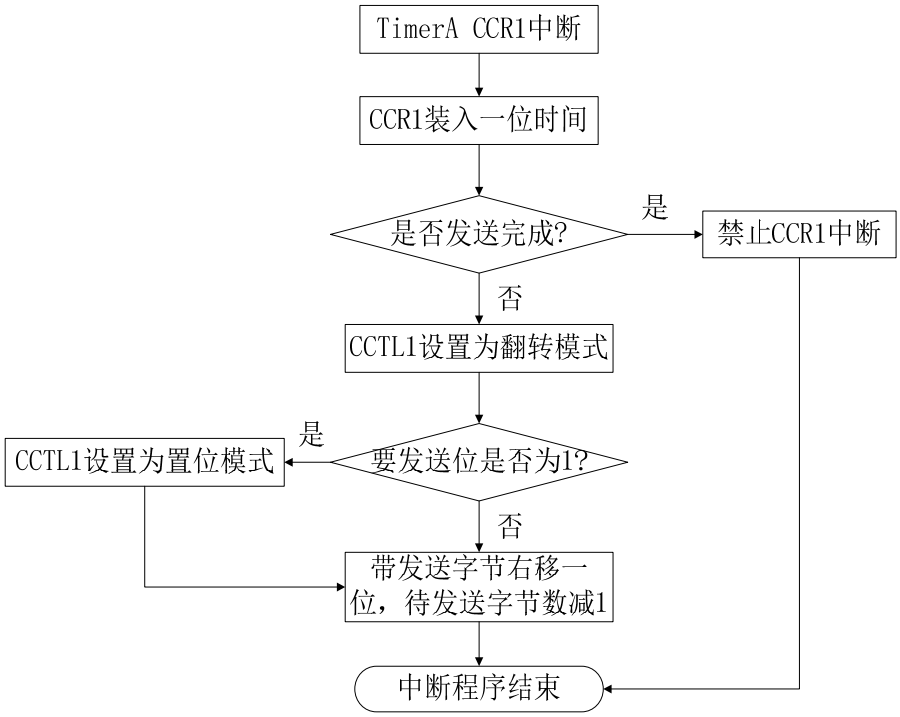

图 4.　发送中断程序流程图

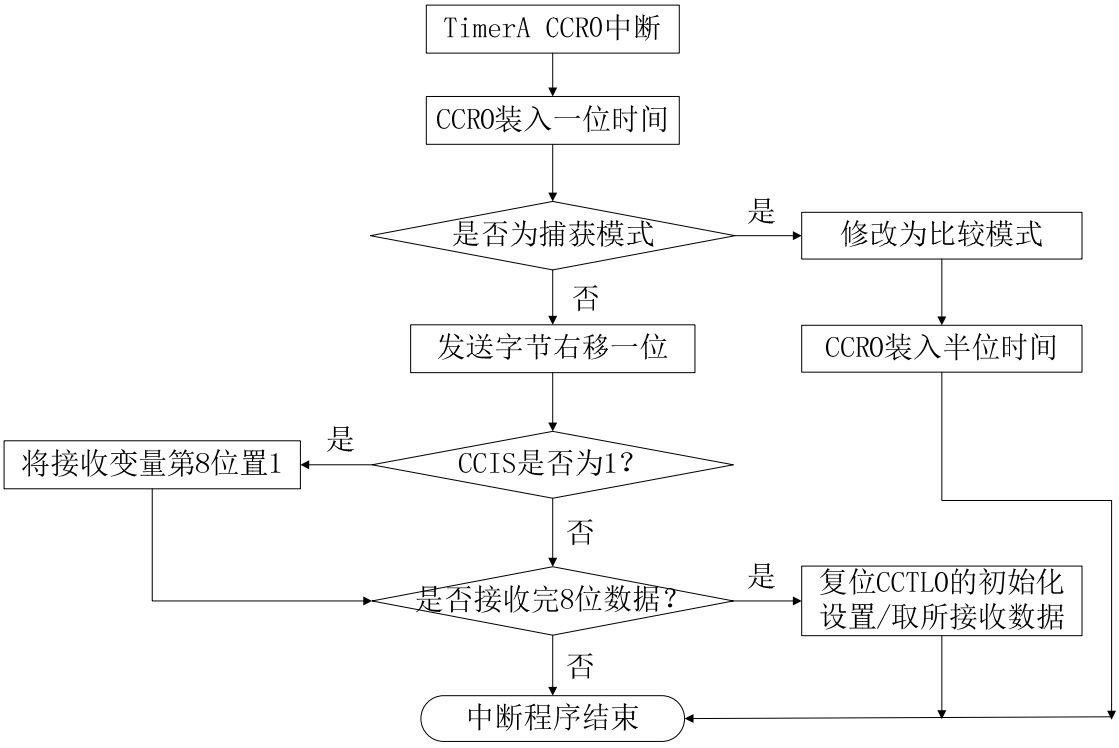

图 5.　接收中断程序流程图

4 Flash 模拟 EEPROM

Flash 模拟 EEPROM 功能通过在线应用编程方法来实现。使用此功能可将重要数据保存到芯片中，作为掉电保护参数和产品 ID 信息标识。Flash 数据读写函数见附录 1。

5 MSP430G2xx1 系列串口在线升级功能的实现

利用 MSP430G2231 软件定时器模拟 UART 与 PC 机通讯程序和 Flash 读写程序，将程序编译生成的代码通过 RS232 口发送至单片机 MSP430G2231，单片机接收到数据后按照设计将代码存至片内的程序存储器中即可实现程序的在线升级。

5.1 BSL 简介

BSL 全称为启动程序装载器(Bootstrap Loader)。它只需简单的外部电路即可实现单片机程序代码的烧写，完成单片机程序的在线更新。这里，在单片机复位时候通过一个按键来判断是启动 BSL 程序还是主程序。如果启动 BSL 程序，则 UART 口对 Flash 进行在线的编程。

5.2 BSL 的实现

MSP430G2231 具有 2K 的 Flash 以及 256B 的 Information Flash 区(从 0x1000 到 0x10FF)。为了防止 BSL 程序被擦除，可将 BSL 程序置于 Information Flash 区中。系统中的 MSP430G2231 的 Flash 分配如图 6 所示。

地址	用途
0xFFFF to OxFFE0	interrupt vector
0xFC00 to OxFDDF	Flash
0x10FD to Ox1000	BSL
0x027F to Ox0200	RAM
0x01FF to Ox0000	Peripherals

图 6. MSP430G2231 的 Flash 空间分配

控制内存分配的 cmd 文件关键代码如下所示，它详细的分配了 BSL 和 Flash 以及寄存器的空间。

```
    SFR                     : origin = 0x0000,  length = 0x0010
    PERIPHERALS_8BIT        : origin = 0x0010,  length = 0x00F0
    PERIPHERALS_16BIT       : origin = 0x0100,  length = 0x0100
    RAM                     : origin = 0x0200,  length = 0x0080
    BSL                     : origin = 0x1000,  length = 0x00FE
    FLASH                   : origin = 0xFE00,  length = 0x01E0
```

在系统复位时候，由按键 S2(对应单片机引脚 P1.3)是否按下(按下该引脚电平为低电平)来决定执行 Main 程序或 BSL 程序。如果在系统复位时 S2 处于按下状态，则将启动 BSL 程序。BSL 程序流程图如图 7 所示。它首先初始化串口，然后等待 UART 发送同步字符，当接受到同步字符后，单片机即等待 UART 发送程序目标代码，并将其写入 Main Flash 中。

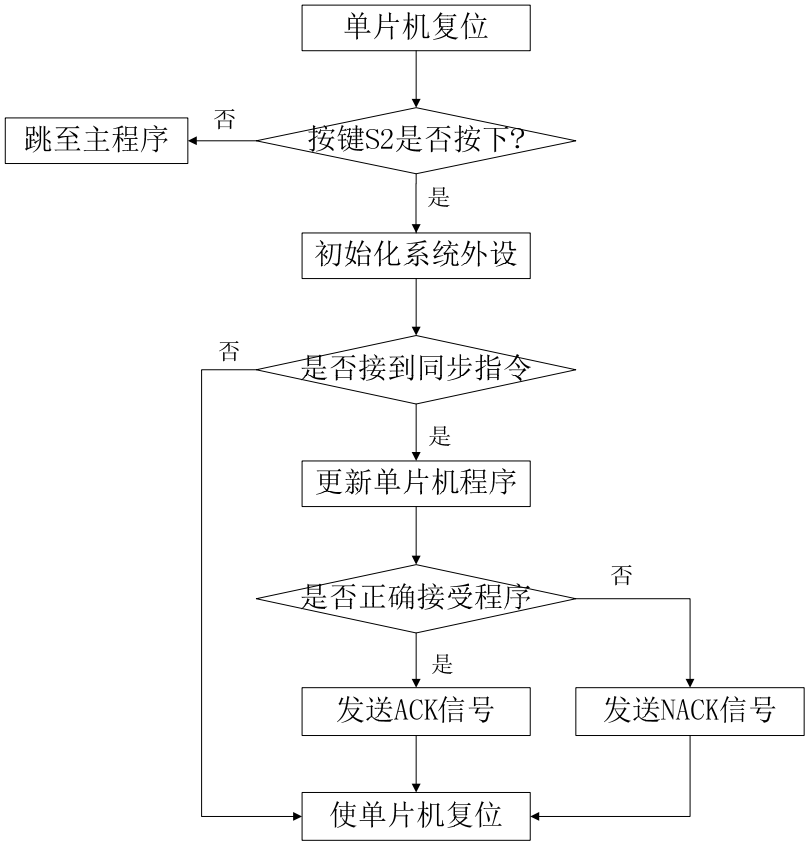

图 7. BSL 程序流程图

6 MSP430G2231 程序代码的编写

6. 1 PWM 功能模拟 D/A 的实现

```
#define               PWMDA_PORT                    BIT2          //定义 PWM 输出端口
#define               PWM_DUTY_RANGE                1024          //定义 PWM 周期
P1SEL |= PWMDA_PORT;                                              //设置 PWM 端口属性
P1DIR |= PWMDA_PORT;                                              //设置 PWM 端口为输出方向
P1OUT = 0;
TACTL = TASSEL_2 + MC_1 + ID_0;                                   //初始化定时器 A
CCR0  = PWM_DUTY_RANGE;
CCR1  = PWM_DUTY_Ctrl;
CCTL1 = OUTMOD_7;
```

PWM 模拟 DAC 的主要程序代码如上所示，主要设置 TimerA 的工作模式，使其按照要求输出固定频率固定占空比的 PWM 波。PWM 模拟 DA 的测试程序如附录 2 所示。

6. 2 模拟 UART 的实现

软件模拟 UART 主要使用 TimerA 的 2 个捕获比较口，来产生指定波特率的数据传输信号。软件模拟 UART 主要程序如附录 3 所示。程序中设置波特率为 4800bps，通过串口调试助手显示数据的收发数据如图 8 所示。

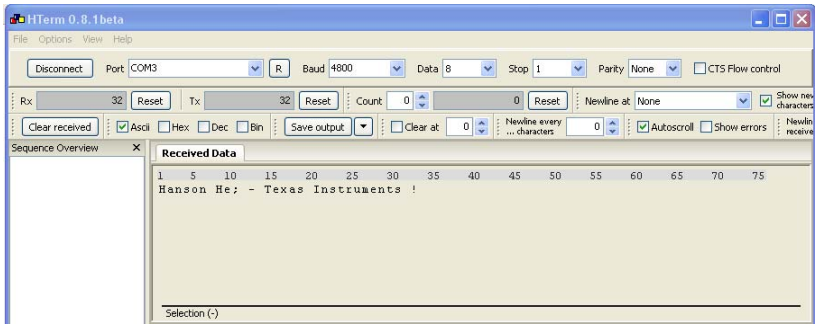

图 8. 软串口收发数据图

6. 3 读写 Flash 的实现

读写 Flash 的实现可以通过 MSP430G2231 中的 BSL 启动程序将从 UART 收到的用户应用程序代码文件写入到 Flash 存储器中，由于 MSP430G2231 的 BOOT 代码要求越小越好，因此，Boot 代码中读写 Flash 函数用汇编来编写，整个的 Bootstrap 程序目前只有 256B，所有的 MSP430G 系列都满足这一基本配置要求。具体汇编代码部分可以参考附录 4 中的代码。当然也可参照附录 1 中的 C 代码来实现 Flash 的读写操作。

7 实际应用

在汽车驾驶系统中，除了车身控制单元等，需要使用一些相对简单的功能来做车窗、车门、雨刮器、车灯的控制等。在实际的案例中，电动车窗控制应用 MSP430G2231 微处理器使用 2400 波特率来实现与主控单元的通信。通过内部 A/D 来检测过流，实现防夹手的功能。使用 I/O 来检测按键的输入，以及控制 TPL9202 输出驱动继电器实现电机正转和反转。在芯片中也保留一段字节用于存储产品的 ID 号，用来识辨产品的信息和版本等。下图是产品简单的应用框图。当然，MSP430G 系列在电动卷帘门、搅拌机、烟感等应用场合也有很多成功的例子。本文中的设计适合于所有 MSP430G 系列，可方便的在 MSP430G 系列中移植。

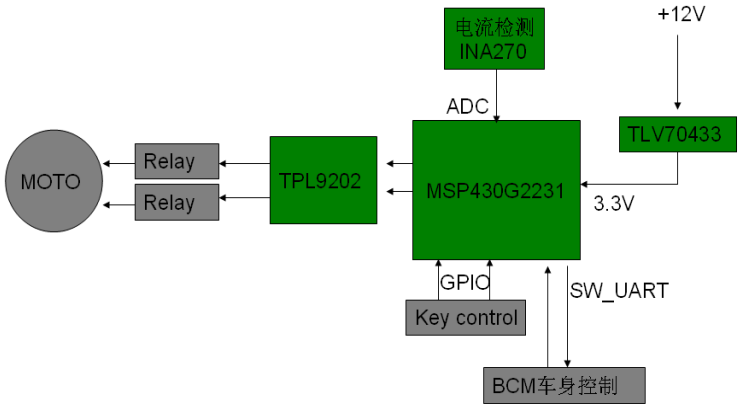

图 9. MSP430G2231 在电动车窗中的应用框图

附录 1. Flash 读写程序关键代码

```
void            Flash_Config()
{
              FCTL2 = FWKEY + FSSEL0 + FN1;
}
void            Flash_EraseSeg(unsigned int SegID)
{
              char * SegAddr= (char *)SegID;          //取得段开始地址
              FCTL1         = FWKEY + ERASE;
              FCTL3         = FWKEY ;
              *SegAddr      = 0;
              FCTL1 = FWKEY;
              FCTL3 = FWKEY + LOCK;
}
void            Flash_WRByte(unsigned int ByteID, unsigned char wrValue)
{
              char * ByteAddr;
              ByteAddr=(char *)ByteID;
              FCTL1 = FWKEY + WRT;
              while( FCTL3 & BUSY);
               *ByteAddr++ = wrValue;                  // 写字节到 flash
              while( FCTL3 & BUSY);
              FCTL1 = FWKEY;                          // 清除 WRT 位
              FCTL3 = FWKEY + LOCK;                   // 锁定 Flash
}
void            Flash_WRSeg(unsigned int SegID, unsigned char wrValue)
{
              char * SegAddr;
              char          i;
              SegAddr=(char *)SegID;
              FCTL1 = FWKEY + WRT;
              while( FCTL3 & BUSY);
              for (i=0; i<64; i++)
              {
                            *SegAddr++ = wrValue;       //写字节到 Flash
                            while( FCTL3 & BUSY);
              }
              FCTL1 = FWKEY;                          // 清除 WRT 位
              FCTL3 = FWKEY + LOCK;                   // 锁定 Flash
}
char            Flash_RDByte(unsigned int ByteID)
{
              FCTL1 = FWKEY;                          // 清除 WRT 位
              FCTL3 = FWKEY + LOCK;                   // 锁定 Flash
              char * ByteAddr;
              ByteAddr=(char *)ByteID;
              return (*ByteAddr);
}
```

附录 2. PWM 模拟 DAC 程序

```
#define       PWMDA_PORT                  BIT2
#define PWM_DUTY_RANGE                    1024          // PWM 占空比变化范围，也决定 PWM 频率
volatile unsigned int                     PWM_DUTY_Ctrl = PWM_DUTY_RANGE/2;
void          PWMDA_Config()
{
              // Port Config
              P1SEL |= PWMDA_PORT;
              P1DIR |= PWMDA_PORT;
              P1OUT = 0;
              // TimerA 配置
              TACTL = TASSEL_2 + MC_1 + ID_0;          // SMCLK/2，  连续模式
              CCR0  = PWM_DUTY_RANGE;
              CCR1  = PWM_DUTY_Ctrl;
              CCTL1 = OUTMOD_7;
}
void main()
{
              WDTCTL = WDTPW+ WDTHOLD;
              Set_DCO(DELTA_12MHZ);
              PWMDA_Config();
              while(1)
              {
                 //Change PWM_DUTY_Ctrl to change the PWM Duty
                 PWM_DUTY_Ctrl = 400;
                 CCR0 = PWM_DUTY_Ctrl;
                 _bis_SR_register(LPM1_bits + GIE);    // 开启中断和 LPM3
              }
}
```

附录 3. 软件模拟 UART 程序关键代码

```
void          SfUart_Config()                          //软 UART 初始配置函数
{
              #ifdef SOFTUART_ENABLE
              //软 UART 的引脚配置
              P1SEL |= SOFT_TXD + SOFT_RXD;            //设置 Timer 的引脚作用是外围引脚
              P1DIR = ~ SOFT_RXD;                      //设置 RXD  为输入放下
              P1OUT = 0;                               //输出 全部为 0
              //软 UART 的 Timer 模块配置
              CCTL0 = OUT;
              CCTL1 = SCS + CM_2 + CAP + CCIE;
              TACTL = TASSEL_2 + MC_2 + ID_3;          //SMCLK/8,  连续模式
              #endif
}
void          SfUart_PrintByte(unsigned int Byte)      //发送字节函数
{
              TXByte = Byte;
              BitCnt = 0xA;                       //装载位计数器
              while (CCR0 != TAR)                 //防止非同步捕获
                            CCR0 = TAR;           //装载新的 TA 计数器数值
              CCR0 += ISR_PERIOD;                 //向发送数据添加停止位
              TXByte |= 0x100;
              TXByte = TXByte << 1;               //向发送数据添加空起始位
              CCTL0 =  CCIS0 + OUTMOD0 + CCIE;    //设置定时器工作模式
              while ( CCTL0 & CCIE );             //等待 TX 发送完成
}
void          SfUart_PrintStr(char * StrPt)      //发送字符串函数
{
              char * TxPtr=StrPt;
              while( *TxPtr != '\0')
                            {
                                          SfUart_PrintByte(*TxPtr);
                                          TxPtr++;
                            }
}
// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
              CCR0 += ISR_PERIOD;                       //CCR0 加一位时间
              if (CCTL0 & CCIS0)                        //TX 端口是否正确
              {
                            if ( BitCnt == 0)
                            {
                               CCTL0 &= ~ CCIE ;    //数据传输完毕，封锁中断
                            }
                            else
                            {
                               CCTL0 |=  OUTMOD2;
                               if (TXByte & 0x01)
                                  CCTL0 &= ~ OUTMOD2;
                                  TXByte = TXByte >> 1;
                                  BitCnt --;
                            }
              }
}
#pragma vector=TIMERA1_VECTOR
```

```
__interrupt void Timer_A1 (void)
{
              static char RecPtr=0;
              switch (TAIV)
              {
              case TAIV_TACCR1:
                            CCR1 += ISR_PERIOD;         //CCR0 加一位时间
                    if(CCTL1 & CAP)                     //是否捕获模式?
                    {
                            CCTL1 &= ~CAP;
                            CCR1  += ISR_PERIOD_HALF;
                    }
                    else
                    {
                            RXByte = RXByte >>1;
                            if(CCTL1 & SCCI)
                            RXByte |= 0x80;
                            RBitCnt -- ;
                            if(RBitCnt == 0)
                            {
                                 RBitCnt  = 8;
                                 CCTL1 |= CAP ;
                                 RxBuffer = RXByte;
                                 if(RxBuffer == ';')
                                 {
                                          RxMessage[RecPtr]='\0';
                                          if(RECEIVEBACK == 1)
                                          TxMessage=RxMessage;
                                          RecPtr = 0;
                                          SfUart_PrintStr(TxMessage);
                                 }
                                 else
                                 {
                                          RxMessage[RecPtr++]=RxBuffer;
                                 }
                            }
                     }
                        break;
              case 4:  break;
              case 10: break;
              }
}
```

附录 4. BSL 程序关键代码

```
RESET
            bit.w   #BSLPIN，&P1IN            ;BSL 按键低电平启动 BSL 程序
BslEntry?   jz      InvokeBsl
            br      #main                     ;BSL 按键高电平退出 BSL 程序

;-------------------------------------------------------------------------------
;           BSL 唤醒
;-------------------------------------------------------------------------------
InvokeBsl   mov.w   #280h，SP                 ;初始化 RAM 堆栈指针
                                              ; RAM: 0x27f - 0x200
UnlockFlash mov.w   #FWKEY+00h，&FCTL3        ;解锁 Flash
StopWDT     mov.w   #WDTPW+WDTHOLD，&WDTCTL   ;停止看门狗时钟
SetupDCO    ; 设置 DCO 时钟为 1 MHz:
            clr.b   &DCOCTL                   ;
            mov.b   #0afh，&DCOCTL            ;
            mov.b   #086h，&BCSCTL1           ;
SetupPins   bis.b   #RXD，&P1SEL              ;
            bis.b   #TXD，&P1DIR
            bis.b   #TXD，&P1OUT
SetupTA0    ;设置定时器 0 工作模式
            mov.w   #CM_2+CCIS_0+SCS+CAP+CCIE，&TACCTL0
            ;定时器工作在连续模式，时钟源为 SMCLK
            bis.w   #TASSEL_2+MC_2+TACLR，&TACTL
InitRx      mov.w   #RX_Count，BitCnt         ;设置 UART 接收标志
            clr     rxCount                   ;接收数据计数器清零
            eint                              ;使能全局中断
;-------------------------------------------------------------------------------
MainBsl             ; BSL 主循环
;-------------------------------------------------------------------------------
Wait4sync   bit.b   #BIT0，  rxCount          ;等待 UART 接收标志
            jz      Wait4sync                 ;等待直到字节接收完成
            dec     rxCount
SyncCmd?    cmp     #CMD_SYNC，rxData         ;是否接收到同步字符?
            jne     BSLEnd                    ;没有接收到同步字符则退出 BSL 程序
;-------------------------------------------------------------------------------
CmdFct_Erase        ;擦除主存储器，恢复中断向量地址
;-------------------------------------------------------------------------------
            mov     &FWKEY+ERASE，&FCTL1      ; ERASE=1
            mov.b   #0h，&0FE00h
            mov     #FWKEY+WRT，&FCTL1        ; WRT=1
            mov     #RESET，&0FFFEh           ;
            mov     #TA0ISR，&0FFF2h          ;
;-------------------------------------------------------------------------------
CmdFct_Write        ; 写主存储器
            mov     #0FE00h，rPoint           ;指向 FLASH 的第一个字节
            clr     rCHKSUM                   ;初始化校验和
CFW_w4d     bit.b   #BIT0，  rxCount          ;循环执行，直到数据接收完成
            jz      CFW_w4d
            dec     rxCount
CFW_Range   cmp     #0FFF2h，rPoint           ;指向 TA0 向量?
            jeq     CFW_COMM                  ;
            cmp     #0FFF3h，  rPoint         ;指向 TA0 向量?
            jeq     CFW_COMM                  ;
            cmp     #0FFFEh，  rPoint         ;指向复位向量?
```

```
            jeq     CFW_Done                ;退出写操作
            mov.b   rxData，0(rPoint)        ;将 8 位数据写入 Flash
CFW_COMM
CFW_Xor     xor.b   @rPoint+，rCHKSUM       ;
            ;inc    rPoint                  ;
            jmp     CFW_w4d                 ; 等待下一个字节
CFW_Done
            cmp.b   rxData，  rCHKSUM        ; 计算校验和
            jeq     LoadACK                 ;  如果匹配则为零

LoadNACK    mov.w   #(NACKCYCL/3)，rTemp     ;
            jmp     SendACK
LoadACK     mov.w   #(ACKCYCL/3)，rTemp      ; 3 个 CPU 周期的等待

SendACK     bic.b   #TXD，&P1OUT             ; 给 PC 机发送应答
DelayACK    dec.w   rTemp                   ;
            jnz     DelayACK                ; 超时?
            bis.b   #TXD，&P1OUT

BSLEnd      ;jmp     MainBsl                ; 启动 BSL
            clr     WDTCTL                  ;
;-------------------------------------------------------------------------------
TA0ISR      ; CCR0/UART 中断服务程序， RXTXData 缓存区用于保存 UART 数据
;-------------------------------------------------------------------------------
            add.w   #BITTIME，&TACCR0        ;
            bic.w   #CCIFG，  &TACCTL0       ; 清除 IFG 标志
            mov.b   @BitCnt+，rTemp          ;
            add     rTemp，PC                ;
RX_Edge     bic.w   #CAP+CCIFG，&TACCTL0     ; 切换到比较模式
            add.w   #BITTIME_5，&TACCR0      ; 第一个 1.5 位的数据
            reti                            ;
RX_Bit      bit.w   #SCCI，&TACCTL0          ; 在 SCCI 中等待接收位数据
            rrc.b   RXTXData                ;存储接收数据
            reti                            ;
RX_Comp     ;字节接收完成，使能下次接收
            bis.w   #CAP+CCIE，&TACCTL0      ;切换至捕获模式
            mov.w   #RX_Count，BitCnt        ;装载接收位标志计数器
            mov.b   RXTXData，rxData         ;保存接收到的数据
            inc     rxCount                 ;主程序接收新的数据
            reti                            ;
```

参考文献

1. Using PWM Output as a Digital-to-Analog Converter on a TMS320F280x Digital Signal Controller SPRAA88A – September 2008
2. MSP430x2xx Family User's Guide - SLAU144H
3. MSP-EXP430G2 LaunchPad Experimenter Board User's Guide
4. Implementing a UART Function With the 8-Bit Interval Timer/Counter SLAA083 - December 1999

重要声明

德州仪器(TI) 及其下属子公司有权在不事先通知的情况下, 随时对所提供的产品和服务进行更正、修改、增强、改进或其它更改，并有权随时中止提供任何产品和服务。客户在下订单前应获取最新的相关信息，并验证这些信息是否完整且是最新的。所有产品的销售都遵循在订单确认时所提供的TI 销售条款与条件。

TI 保证其所销售的硬件产品的性能符合TI 标准保修的适用规范。仅在TI 保证的范围内，且TI 认为有必要时才会使用测试或其它质量控制技术。除非政府做出了硬性规定，否则没有必要对每种产品的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。客户应对其使用TI 组件的产品和应用自行负责。为尽量减小与客户产品和应用相关的风险，客户应提供充分的设计与操作安全措施。

TI 不对任何TI 专利权、版权、屏蔽作品权或其它与使用了TI 产品或服务的组合设备、机器、流程相关的TI 知识产权中授予的直接或隐含权限作出任何保证或解释。TI 所发布的与第三方产品或服务有关的信息，不能构成从TI 获得使用这些产品或服务的许可、授权、或认可。使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可，或是TI 的专利权或其它知识产权方面的许可。

对于TI 的产品手册或数据表，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况下才允许进行复制。在复制信息的过程中对内容的篡改属于非法的、欺诈性商业行为。TI 对此类篡改过的文件不承担任何责任。

在转售TI 产品或服务时，如果存在对产品或服务参数的虚假陈述，则会失去相关TI 产品或服务的明示或暗示授权，且这是非法的、欺诈性商业行为。TI 对此类虚假陈述不承担任何责任。

TI 产品未获得用于关键的安全应用中的授权，例如生命支持应用（在该类应用中一旦TI 产品故障将预计造成重大的人员伤亡），除非各方官员已经达成了专门管控此类使用的协议。购买者的购买行为即表示，他们具备有关其应用安全以及规章衍生所需的所有专业技术和知识，并且认可和同意，尽管任何应用相关信息或支持仍可能由TI 提供，但他们将独力负责满足在关键安全应用中使用其产 品及TI 产品所需的所有法律、法规和安全相关要求。此外，购买者必须全额赔偿因在此类关键安全应用中使用TI 产品而对TI 及其 代表造成的损失。

TI 产品并非设计或专门用于军事/航空应用，以及环境方面的产品，除非TI 特别注明该产品属于“军用”或“增强型塑料”产品。只 有TI 指定的军用产品才满足军用规格。购买者认可并同意，对TI 未指定军用的产品进行军事方面的应用，风险由购买者单独承担，并且独力负责在此类相关使用中满足所有法律和法规要求。

TI 产品并非设计或专门用于汽车应用以及环境方面的产品，除非TI 特别注明该产品符合ISO/TS 16949 要求。购买者认可并同意，如果他们在汽车应用中使用任何未被指定的产品，TI 对未能满足应用所需要求不承担任何责任。

可访问以下URL 地址以获取有关其它TI 产品和应用解决方案的信息：

	产品		应用
数字音频	www.ti.com.cn/audio	通信与电信	www.ti.com.cn/telecom
放大器和线性器件	www.ti.com.cn/amplifiers	计算机及周边	www.ti.com.cn/computer
数据转换器	www.ti.com.cn/dataconverters	消费电子	www.ti.com/consumer-apps
DLP® 产品	www.dlp.com	能源	www.ti.com/energy
DSP - 数字信号处理器	www.ti.com.cn/dsp	工业应用	www.ti.com.cn/industrial
时钟和计时器	www.ti.com.cn/clockandtimers	医疗电子	www.ti.com.cn/medical
接口	www.ti.com.cn/interface	安防应用	www.ti.com.cn/security
逻辑	www.ti.com.cn/logic	汽车电子	www.ti.com.cn/automotive
电源管理	www.ti.com.cn/power	视频和影像	www.ti.com.cn/video
微控制器 (MCU)	www.ti.com.cn/microcontrollers		
RFID 系统	www.ti.com.cn/rfidsys		
OMAP 机动性处理器	www.ti.com/omap		
无线连通性	www.ti.com.cn/wirelessconnectivity		
	德州仪器在线技术支持社区	www.deyisupport.com	

邮寄地址： 上海市浦东新区世纪大道 1568 号，中建大厦 32 楼 邮政编码： 200122