NV32F100x CRC 模块编程示例

第一章 库函数简介

1.1 库函数列表

void CRC_Init(CRC_ConfigType *pConfig)

通过 CRC_ConfigType 初始化 CRC

uint32_t CRC_Cal16(uint32_t seed, uint8_t *msg, uint32_t sizeBytes)

16 位模式下 CRC 计算函数

uint32_t CRC_Cal32(uint32_t seed, uint8_t *msg, uint32_t sizeBytes)

32 位模式下 CRC 计算函数

void CRC_DeInit(void)

CRC 恢复默认状态函数

1.2 CRC 模块特性

*使用 16 位或 32 位可编程移位寄存器的硬件 CRC 生成器电路

*可编程初始种子值和多项式

*逐位或逐字节转置输入数据或输出数据（CRC 结果）。某些 CRC 标准要求提供该选项。以 8 位读

*取操作访问 CRC 数据寄存器时，无法执行逐字节转置操作。这种情况下，用户软件必须执行逐字节转置操作。

*提供最终 CRC 结果反转选项

*32 位 CPU 寄存器编程接口

1.3 CRC 计算使用说明

*设置 CRC 协议宽度，选择 16 或 32 位

*按 CRC 计算要求对转置进行编程，并在 CTRL 寄存器中补全选项位。

*将多项式写入 CRC 多项式寄存器对应字段

*进行种子值的编程。将种子值写入数据寄存器对应字段

*写入数据值写入数据寄存器。

*完成所有数值的写入操作后，从 CRC 数据寄存器对应字段读取最终的 CRC 结果。

详细 CRC 计算过程及配置请参阅参考手册

第二章 CRC 模块初始化

2.1 CRC 数据寄存器（CRC_DATA）

CRC 数据寄存器包含种子、数据以及校验和的数值。如果 CTRL[WAS]置位，则对数据寄存器进行的任何写操作都被视为种子值。如果 CTRL[WAS]清零，则对数据寄存器进行的任何写操作都被视为用于一般 CRC 计算的数据。详细请参看参考手册。

位	描述
31－24 HU	CRC 高字段高位字节 在 16 位 CRC 模式（CTRL[TCRC]为 0）下，该字段并不用于种子值的编程。在 32 位 CRC 模式（CTRL[TCRC]为 1）下，当 CTRL[WAS]为 1 时，写入该字段的值是种子值的一部分。当 CTRL[WAS]为 0 时，写入该字段的数据用于在 16 位 CRC 模式和 32 位 CRC 模式下生成 CRC 校验和。
23-16 HL	CRC 高字段低位字节 在 16 位 CRC 模式（CTRL[TCRC]为 0）下，该字段并不用于设定种子值。在 32 位 CRC 模式（CTRL[TCRC] 为 1）下，当 CTRL[WAS]为 1 时，写入该字段的值是种子值的一部分。当 CTRL[WAS]为 0 时，写入该字段的数据用于在 16 位 CRC 模式和 32 位 CRC 模式下生成 CRC 校验和。
15－8 LU	CRC 低字段高位字节 当 CTRL[WAS]为 1 时，写入该字段的数值是种子值的一部分。当 CTRL[WAS]为 0 时，写入该字段的数据用于生成 CRC 校验和。
7-0 LL	CRC 低字段低位字节 当 CTRL[WAS]为 1 时，写入该字段的数值是种子值的一部分。当 CTRL[WAS]为 0 时，写入该字段的数据用于生成 CRC 校验和。

2.2 CRC 多项式寄存（CRC_GPOLY）

该寄存器含有 CRC 计算所需的多项式值，详细请参看参考手册

位	描述
31－16 HIGH	多项式高半字 32 位 CRC 模式下可读写（CTRL[TCRC]为 1），该字段在 16 位 CRC 模式下不可写（CTRL[TCRC]为 0）。
15-0 LOW	多项式低半字 在 32 位 CRC 模式和 16 位 CRC 模式下都可读写。

2.3 CRC 控制寄存器（CRC_CTRL）

寄存器控制 CRC 模块的配置和操作。开始进行新的 CRC 计算前，相应位必须置位。初始化新的 CRC 计算的方法是：使 CTRL[WAS]的电平变为有效，然后将种子写入 CRC 数据寄存器。详细请参考参考手册。

位	描述
31－30 TOT	写入的转置类型 定义写入 CRC 数据寄存器的数据的转置配置。有关可用的转置选项，请参见转置特性说明。 00 无转置。 01 字节中的位转置；字节不转置。 10 字节中的位和字节均转置。 11 仅字节转置；字节中的位不转置。
29－28 TOTR	读取的转置类型 识别从 CRC 数据寄存器读取的数值的转置配置。有关可用的转置选项，请参见转置特性说明。 00 无转置。 01 字节中的位转置；字节不转置。 10 字节中的位和字节均转置。 11 仅字节转置；字节中的位不转置。
27 保留	此字段为保留字段。 此只读字段为保留字段且值始终为 0。
26 FXOR	CRC 数据寄存器的补充读取 某些 CRC 协议要求最终校验和与 0xFFFFFFFF 或 0xFFFF 进行异或运算。使该位的电平变为有效可使能已读取数据的即时补充。 0 读取时不执行异或运算。 1 反转或补充 CRC 数据寄存器的读取值。
25 WAS	作为种子写入 CRC 数据寄存器 电平变为有效后，写入 CRC 数据寄存器的值被视为种子值。电平变为无效后，写入 CRC 数据寄存器的值用作 CRC 计算中的数据。 0 写入 CRC 数据寄存器的是数据值。 1 写入 CRC 数据寄存器的是种子值。
24 TCRC	CRC 协议宽度。 0 16 位 CRC 协议。 1 32 位 CRC 协议。
23-0 保留	此字段为保留字段。 此只读字段为保留字段且值始终为 0。

函数名	CRC_Init
函数原形	CRC_Init(CRC_ConfigType *pConfig)
功能描述	通过设置配置结构体 pConfig 初始化 CRC
输入参数	CRC 的配置结构体 CRC_ConfigType
输出参数	无
返回值	无
先决条件	无
函数使用实例	先设置 CRC 配置结构体，CRC_Init(pCRC_Config);

```
*****************************************************************************//*!
*
* @ CRC 初始化函数
*
* @ 输入 pConfig  指向 CRC 配置结构体
*
* @ 无返回
*
*****************************************************************************/
    void CRC_Init(CRC_ConfigType *pConfig)
     {
            uint32_t     u32Sc ;
            u32Sc= 0;
            SIM->SCGC |= SIM_SCGC_CRC_MASK; //开启 CRC 模块的总线时钟
            u32Sc|= ((pConfig->bWidth & 0x01)<<24);//设置 CRC 协议宽度
            u32Sc|= CRC_CTRL_TOTR(pConfig->bTransposeReadType & 0x03);//设置读取转置类型
            u32Sc|= CRC_CTRL_TOT(pConfig->bTransposeWriteType & 0x03);//设置写入转置类型
            if (pConfig->bFinalXOR)//设置 CRC 数据寄存器的补充读取
            {
                  u32Sc |= CRC_CTRL_FXOR_MASK;
             }
           CRC0->CTRL= u32Sc;
           if ( pConfig->bWidth )//32 位 CRC 协议
           {
             CRC0->GPOLY= pConfig->u32PolyData;//写 32 位 CRC 计算所需要的多项式
           }
          else
           {
             CRC0->GPOLY_ACCESS16BIT.GPOLYL = pConfig->u32PolyData;//仅允许写 16 位
            }
      }
```

第三章 CRC 计算

在 16 位 CRC 模式和 32 位 CRC 模式下，如果所有字节都是连续的，那么一次可进行 8 位 16 位或 32 位数据值的编程。非连续字节可能导致 CRC 计算错误。

函数名	CRC_Cal16
函数原形	CRC_Cal16(uint32_t seed, uint8_t *msg, uint32_t sizeBytes)
功能描述	16 模式下生成冗余校验码
输入参数	种子值、数据、数据大小
输出参数	无
返回值	校验码

先决条件	先初始化 CRC 函数
函数使用实例	先设置种子值、数据和数据大小， CRC_Cal16(u32SeedValue, &MessageSource[0], (sizeof(MessageSource)-1))

```
/*****************************************************************************//*!
    *
    * @ 16 位模式下 CRC 计算
    *
    * @ 输入 seed 种子值
    * @ 输入 msg 指向数据缓存区
    * @ 输入 sizeBytes 数据大小
    *
    * @返回计算结果
    *
    *****************************************************************************/
    uint32_t CRC_Cal16(uint32_t seed, uint8_t *msg, uint32_t sizeBytes)
    {
          uint32_t   ctrl_reg,data_out,data_in;
          uint8_t   *pCRCBytes;
          uint32_t    sizeWords;
          uint32_t    i,j;
        /* 写入种子值 ， 设置 WaS=1 */
        ctrl_reg= CRC0->CTRL;
        CRC0->CTRL= ctrl_reg | CRC_CTRL_WAS_MASK;//WaS=1
        CRC0->ACCESS16BIT.DATAL = seed;     //写入种子值
        /* 写入数据, Set WaS=0*/
        CRC0->CTRL= ctrl_reg & 0xFD000000;              //设置 WaS=0
      /*等待计算完成*/
      sizeWords = sizeBytes>>1;
       j = 0;
       for(i=0;i<sizeWords;i++){
       data_in = (msg[j] << 8) | (msg[j+1]);
         j += 2;
       CRC0->ACCESS16BIT.DATAL =data_in;
         }
      if (j<sizeBytes)
      {
          pCRCBytes = (uint8_t*)&CRC0->ACCESS8BIT.DATALL;
        *pCRCBytes++ = msg[j];
    }
     data_out=CRC0->ACCESS16BIT.DATAL;
     //读取计算结果
   return(data_out);
    }
```

第四章 样例程序

CRC_demo

```
/*****************************************************************************************************
*
*CRC 样例程序,给定一个数据，分别对其进行 16 位模式下和 32 位模式下 CRC 计算
*
*****************************************************************************************************/
    #include "common.h"
    #include "crc.h"
    #include "uart.h"
    #include "sysinit.h"
    int main (void);
    int main (void)
    {
        uint8_t
        u8Ch;
        uint32_t
      u32Crc_ConverterResult;
      uint32_t
      u32SeedValue;
      CRC_ConfigType
      sCRC_ConfigType = {0};
      CRC_ConfigType
      pCRC_Config=&sCRC_ConfigType;
      uint8_t MessageSource[] = {"123456789"} ;
      /*!<数据*/
      /* 系统初始化*/
      sysinit();
      printf("\nRunning the CRC_demo project.\n");
      /* 初始化 CRC 寄存器，使其工作在 16 位 CRC 模式下 */
      /*初始化 CRC 配置结构体*/
      pCRC_Config->u32PolyData= 0x1021;    /*!< CRC 计算多项式的值 */
        u32SeedValue= 0xFFFF;          /*!< 设置 CRC 种子值 */
```

```
    pCRC_Config->bWidth= CRC_WIDTH_16BIT;     //16 位 CRC 协议
    pCRC_Config->bTransposeReadType= CRC_READ_TRANSPOSE_NONE;   /*!< 读取时转置*/
    pCRC_Config->bTransposeWriteType= CRC_WRITE_TRANSPOSE_NONE; /*!< 写入时无转置 */
     /* 开始 CRC-CCITT 计算
       */
    CRC_Init(pCRC_Config);       /*!< 初始化 CRC 模块 */
    printf("CRC0->GPOLY=0x%x, CRC0->CTRL=0x%x\n",CRC0->GPOLY,CRC0->CTRL);
    u32Crc_ConverterResult = CRC_Cal16(u32SeedValue, &MessageSource[0], (sizeof(MessageSource)-1));
     //16 位模式下 CRC 计算
 printf("CRC-CCITT function calculation result = 0x%x @seed = 0x%x .\n", u32Crc_ConverterResult,
u32SeedValue );
  /* 将 CRC 模块恢复到默认状态*/
  CRC_DeInit();
   /*初始化 CRC 配置结构体*/
  /* 初始化 CRC 寄存器，使其工作在 16 位 CRC 模式下 */
  pCRC_Config->u32PolyData= 0x04C11DB7; /*!< 设置 CRC32 多项式的值*/
  u32SeedValue= 0xFFFFFFFF;       /*!< 设置 CRC32 种子值 */
  pCRC_Config->bWidth= CRC_WIDTH_32BIT; //32 位 CRC 协议
  pCRC_Config->bTransposeReadType= CRC_READ_TRANSPOSE_ALL; /*!< 读取时无转置 */
  pCRC_Config->bTransposeWriteType= CRC_WRITE_TRANSPOSE_BIT; /*!< 写入时无转置*/
  pCRC_Config->bFinalXOR= TRUE; /*!< 反转或补充 CRC 数据寄存器的读取值 */
  /* 开始 CRC-CCITT 计算 */
  CRC_Init(pCRC_Config); /*!< 初始 CRC 模块*/
  u32Crc_ConverterResult = CRC_Cal32(u32SeedValue, &MessageSource[0], (sizeof(MessageSource)-1));
  //32 位模式下 CRC 计算
  printf("CRC32 function calculation result = 0x%x @seed = 0x%x .\n", u32Crc_ConverterResult,
  u32SeedValue );
  while(1)
  {
  u8Ch = UART_GetChar(TERM_PORT);
  UART_PutChar(TERM_PORT, u8Ch);
  }
  }
```