AT91SAM7X-EK softpack 1.5 代码解读之 ADC

——基于 SAM7X-EK 和 IAR EWARM

文档编号	MAN2012A_CH				
文档版本	Rev. A				
文档摘要	基于 AT91SAM7X-EK 开发板的代码解读，ADC 代码解读				
关键词	AT91SAM7X256 SAM7X-EK IAR EWARM J-LINK ADC				
创建日期	2010-06-017	创建人员	Cust126	审核人员	Robin
文档类型	公开发布/开发板配套文件				
版权信息	Mcuzone 原创文档，转载请注明出处				

更新历史

版本	时间	更新	作者
Rev. A	2010-06-17	初始创建	Cust126

微控电子 乐微电子
杭州市登云路 639 号 2B143
销售 TEL：86-571-89908193 13957118045
支持 TEL：18913989166 13957118045
FAX：86-571-89908193
www.mcuzone.com www.atarm.com

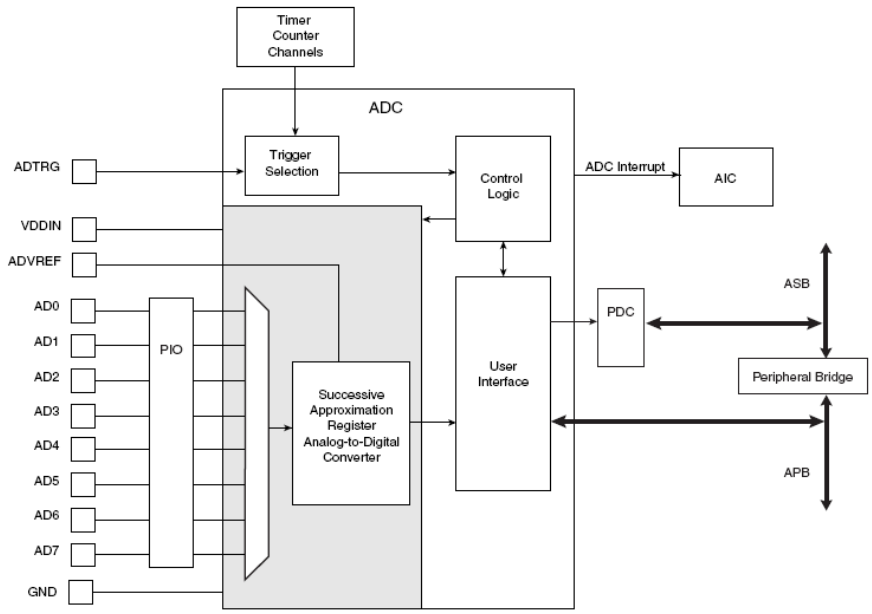
1.概述

本文档以 SAM7X-EK 为硬件平台，IAR EWARM 为编译器平台，使用 J-Link 作为调试工具，演示并解读 AT91SAM7X256 的 ADC 操作流程。

2. ADC 操作

2.1 ADC 操作流程介绍

ADC 主要有 ADC_CR、ADC_MR、ADC_CHER、ADC_CHDR、ADC_CHSR、ADC_SR、ADC_LCDR、ADC_IDR、ADC_IER、ADC_IDR、ADC_IMR、ADC_CDR0-7 寄存器；
ADC 的初始化流程如下：
先初始化时钟，软件复位控制器，再设置 ADC_MR 寄存器，使用 ADC 中断请求前须先对 AIC 编程。
ADC 模数转换器的框图：



2.2 ADC 操作的目的与功能描述

AT91 softpack 1.5 的 SAM7X-EK 包内的 basic-adc-project 工程目的主要是帮助初学者熟悉 ADC 在 AT91SAM7X 系列上的用法。功能是对输入的模拟信号转换成数字信号。

2.3 示例代码

以下代码截取自 AT91 softpack 1.5 的 SAM7X-EK 包内的 basic-adc-project ，基于 IAR EWARM 平台

2.3.1 main 函数代码注释解读

下面对 basic-adc-project 的主要代码进行注释解读，首先是 main 函数内容：

```
//------------------------------------------------------------------------------
//                Global functions
//------------------------------------------------------------------------------
        //主要功能是测试 ADC 通道 0 并通过 DBGU 串口显示结果在终端上
//------------------------------------------------------------------------------
/// Performs measurements on ADC channel 0 and displays the result on the DBGU.
//------------------------------------------------------------------------------
```

```
int main(void)
{
    unsigned int id_channel;
#ifdef PINS_ADC
    PIO_Configure(pinsADC, PIO_LISTSIZE(pinsADC));   //配置 ADC 引脚
#endif

    ADC_Initialize( AT91C_BASE_ADC,                  //ADC 初始化
                    AT91C_ID_ADC,
                    AT91C_ADC_TRGEN_DIS,
                    0,
                    AT91C_ADC_SLEEP_NORMAL_MODE,
                    AT91C_ADC_LOWRES_10_BIT,
                    BOARD_MCK,
                    BOARD_ADC_FREQ,
                    10,
                    1200);

    ADC_EnableChannel(AT91C_BASE_ADC, ADC_NUM_1);   //ADC 通道 1 使能
    ADC_EnableChannel(AT91C_BASE_ADC, ADC_NUM_2);   //ADC 通道 2 使能
    ADC_EnableChannel(AT91C_BASE_ADC, ADC_NUM_3);   //ADC 通道 3 使能
    ADC_EnableChannel(AT91C_BASE_ADC, ADC_NUM_4);   //ADC 通道 4 使能

    AIC_ConfigureIT(AT91C_ID_ADC, 0, ISR_Adc);      //配置 ADC 中断
    AIC_EnableIT(AT91C_ID_ADC);                      //ADC 中断使能

    printf("-I- Press any key to perform a measurement on the ADC ...\n\r");

    // Infinite loop                                 //循环等待
    while (1) {

        // Wait for user input
        DBGU_GetChar();                              //等待用户输入

        conversionDone = 0;

        ADC_EnableIt(AT91C_BASE_ADC, ADC_NUM_1);     //使能 ADC 通道 1 中断
        ADC_EnableIt(AT91C_BASE_ADC, ADC_NUM_2);     //使能 ADC 通道 2 中断
        ADC_EnableIt(AT91C_BASE_ADC, ADC_NUM_3);     //使能 ADC 通道 3 中断
        ADC_EnableIt(AT91C_BASE_ADC, ADC_NUM_4);     //使能 ADC 通道 4 中断

        // Start measurement
        ADC_StartConversion(AT91C_BASE_ADC);         //ADC 开始转换
```

```
        while( conversionDone !=
((1<<ADC_NUM_1)|(1<<ADC_NUM_2)|(1<<ADC_NUM_3)|(1<<ADC_NUM_4)) );

        for(id_channel=ADC_NUM_1;id_channel<=ADC_NUM_4;id_channel++) {

            printf("Channel %d : %d mV\n\r",
                id_channel,
                ConvHex2mV(ADC_GetConvertedData(AT91C_BASE_ADC, id_channel))
                );
        }                           //在终端上输出显示

        printf("\n\r");
    }
}
```

2.3.2 重要子函数代码注释解读

以下是对几个重要的子函数进行分析：

```
unsigned char PIO_Configure(const Pin *list, unsigned int size)     //PIO 配置子函数
{
    // Configure pins                                               //配置引脚
    while (size > 0) {

        switch (list->type) {                                       //外设类型

            case PIO_PERIPH_A:                                       //外设 A
                PIO_SetPeripheralA(list->pio,                       //设置 PIO 为外设 A
                                   list->mask,
                                   (list->attribute & PIO_PULLUP) ? 1 : 0);
                break;

            case PIO_PERIPH_B:                                          //设置 PIO 为外设 B
                PIO_SetPeripheralB(list->pio,
                                   list->mask,
                                   (list->attribute & PIO_PULLUP) ? 1 : 0);
                break;

            case PIO_INPUT:                                             //设置 PIO 为输入
                AT91C_BASE_PMC->PMC_PCER = 1 << list->id;
                PIO_SetInput(list->pio,
                             list->mask,
                             (list->attribute & PIO_PULLUP) ? 1 : 0,
```

```
                                    (list->attribute & PIO_DEGLITCH)? 1 : 0);
                break;

            case PIO_OUTPUT_0:                                  //设置 PIO 输出为 0
            case PIO_OUTPUT_1:                                  //设置 PIO 输出为 1

                PIO_SetOutput(list->pio,
                              list->mask,
                              (list->type == PIO_OUTPUT_1),
                              (list->attribute & PIO_OPENDRAIN) ? 1 : 0,
                              (list->attribute & PIO_PULLUP) ? 1 : 0);
                break;

            default: return 0;
        }

        list++;
        size--;
    }

    return 1;
}
```

ADC 初始化子程序：

```
void ADC_Initialize (AT91S_ADC *pAdc,
                     unsigned char idAdc,
                     unsigned char trgEn,
                     unsigned char trgSel,
                     unsigned char sleepMode,
                     unsigned char resolution,
                     unsigned int mckClock,
                     unsigned int adcClock,
                     unsigned int startupTime,
                     unsigned int sampleAndHoldTime)
{
    unsigned int prescal;
    unsigned int startup;
    unsigned int shtim;

    ASSERT(startupTime<=ADC_STARTUP_TIME_MAX, "ADC Bad startupTime\n\r");
    ASSERT(sampleAndHoldTime>=ADC_TRACK_HOLD_TIME_MIN, "ADC Bad sampleAndHoldTime\n\r");


    //PRESCAL: 预分频器速率选择
```

```
//ADC 时钟 = MCK / ( (PRESCAL+1) * 2 )
//STARTUP: 启动时间
//启动时间 = (STARTUP+1) * 8 / ADC 时钟
//SHTIM: 采样与保持时间
//采样与保持时间= (SHTIM+1) / ADC 时钟
//例如 ADC 时钟为 5MHZ，20us 的启动时间，600ns 的采样与保持时间
// PRESCAL: Prescaler Rate Selection ADCClock = MCK / ( (PRESCAL+1) * 2 )
// PRESCAL = [MCK / (ADCClock * 2)] -1 = [48/(5*2)]-1 = 3,8
// PRESCAL =    4 -> 48/((4+1)*2) = 48/10 = 4.8MHz
// 48/((3+1)*2) = 48/8 = 6MHz
// Startup Time = (STARTUP+1) * 8 / ADCClock
// STARTUP = [(Startup Time * ADCClock)/8]-1 = [(20 10e-6 * 5000000)/8]-1 = 11,5
// STARTUP = 11 -> (11+1)*8/48000000 = 96/4800000 = 20us
// Sample & Hold Time = (SHTIM+1) / ADCClock
// SHTIM = (HoldTime * ADCClock)-1 = (600 10e-9 * 5000000)-1 = 2
// SHTIM     =    2 -> (2+1)/4800000 = 1/1200000 = 833ns
prescal = (mckClock / (2*adcClock)) - 1;
startup = ((adcClock/1000000) * startupTime / 8) - 1;
shtim = (((adcClock/1000000) * sampleAndHoldTime)/1000) - 1;

 //检查数据是否正确
ASSERT( (prescal<0x3F), "ADC Bad PRESCAL\n\r");
ASSERT(startup<0x7F, "ADC Bad STARTUP\n\r");
ASSERT(shtim<0xF, "ADC Bad SampleAndHoldTime\n\r");

TRACE_DEBUG("adcClock:%d MasterClock:%d\n\r", (mckClock/((prescal+1)*2)), mckClock);
TRACE_DEBUG("prescal:0x%X startup:0x%X shtim:0x%X\n\r", prescal, startup, shtim);

if( adcClock != (mckClock/((prescal+1)*2)) ) {
      TRACE_WARNING("User and calculated adcClocks are different : user=%d calc=%d\n\r",
            adcClock, (mckClock/((prescal+1)*2)));
}

// Enable peripheral clock                          //使能外设时钟
AT91C_BASE_PMC->PMC_PCER = 1 << idAdc;

// Reset the controller                             //软件复位
ADC_SoftReset(pAdc);

// Write to the MR register                         //设置 ADC_MR 寄存器
ADC_CfgModeReg( pAdc,
                     ( trgEn & AT91C_ADC_TRGEN)
                   | ( trgSel & AT91C_ADC_TRGSEL)
                   | ( resolution & AT91C_ADC_LOWRES)
```

```
                    | ( sleepMode & AT91C_ADC_SLEEP)
                    | ( (prescal<<8) & AT91C_ADC_PRESCAL)
                    | ( (startup<<16) & AT91C_ADC_STARTUP)
                    | ( (shtim<<24) & AT91C_ADC_SHTIM) );
}
```

2.3.3 运行结果

注意：实验用的是 ADC 转换通道 5，使用的是外部 3.3V 基准。

代码在终端上运行的结果为：输入电压为 1.7V 时

```
-- Basic ADC Project 1.5 --

-- AT91SAM7X-EK
-- Compiled: Jun 29 2010 10:08:00 --
-I- Press any key to perform a measurement on the ADC ...
Channel 4 : 1106 mV
Channel 5 : 1700 mV
Channel 6 : 1825 mV
Channel 7 : 1570 mV

Channel 4 : 1100 mV
Channel 5 : 1700 mV
Channel 6 : 1851 mV
Channel 7 : 1609 mV

Channel 4 : 1106 mV
Channel 5 : 1700 mV
Channel 6 : 1880 mV
Channel 7 : 1651 mV

Channel 4 : 1093 mV
Channel 5 : 1700 mV
Channel 6 : 1796 mV
Channel 7 : 1535 mV

_
```

输入电压为 2.770V 时：

```
Channel 6 : 2319 mV
Channel 7 : 1825 mV

Channel 4 : 1180 mV
Channel 5 : 2774 mV
Channel 6 : 2367 mV
Channel 7 : 1896 mV

Channel 4 : 1087 mV
Channel 5 : 2770 mV
Channel 6 : 2287 mV
Channel 7 : 1780 mV

Channel 4 : 1096 mV
Channel 5 : 2774 mV
Channel 6 : 2293 mV
Channel 7 : 1787 mV

Channel 4 : 1074 mV
Channel 5 : 2774 mV
Channel 6 : 2306 mV
Channel 7 : 1803 mV
```

输入电压为 3.3V 时：

```
Channel 6 : 2535 mV
Channel 7 : 1903 mV

Channel 4 : 1041 mV
Channel 5 : 3300 mV
Channel 6 : 2583 mV
Channel 7 : 1948 mV

Channel 4 : 1009 mV
Channel 5 : 3300 mV
Channel 6 : 2658 mV
Channel 7 : 2038 mV

Channel 4 : 932 mV
Channel 5 : 3300 mV
Channel 6 : 2606 mV
Channel 7 : 1958 mV

Channel 4 : 887 mV
Channel 5 : 3300 mV
Channel 6 : 2587 mV
Channel 7 : 1919 mV
```

输入电压为 0V 时：

```
Channel 4 : 743 mV
Channel 5 : 0 mV
Channel 6 : 600 mV
Channel 7 : 1041 mV

Channel 4 : 741 mV
Channel 5 : 0 mV
Channel 6 : 583 mV
Channel 7 : 1016 mV

Channel 4 : 735 mV
Channel 5 : 0 mV
Channel 6 : 570 mV
Channel 7 : 990 mV

Channel 4 : 751 mV
Channel 5 : 0 mV
Channel 6 : 606 mV
Channel 7 : 1041 mV
```

输入信号电压使用校准过的万用表进行测试，误差在 5mV 内，电压基准采用 SAM7X-EK 板载 3.3V 系统电源。从测试结果来看，AT91SAM7X256 的轨对轨特性和 ADC 精度都还是很不错的。