文档版本	V1.0.0
发布日期	20221026

APT32F110x 基于 CSI 库 ADC 应用指南

目录

1 概述

本文介绍了在APT32F110x中ADC控制器。

2. 适用的硬件

该例程使用于 APT32F110x 系列学习板

3. 应用方案代码说明

基于 APT32F110x CSI 的库文件系统，进行配置 ADC

3.1 ADC 配置

- **硬件配置**

ADC 模块是一个 12 位的逐次逼近电路，将模拟电平转换为一个 12 位的数字值。 参考电压(AVREF)支持选择内部或者外部 、自带固定电压参考源以及温度传感器(INTVREF) 、模拟输入通道有 AIN[18:0]，内部固定电压参考源输入，以及 1/4VDD 输入、最大转换速度: 1MSPS

使用 FVR 做参考时，需要在 GPIO 的配置中使能对应的 AF 功能 (VREF+)， 在 VREF 管脚上增加一个 0.1uF 的电容到地。（输入的模拟电平值必须在 AVREF 和 AVSS 的值之间。）

参考电源选择配置：

1. 正向为内部 VDD，负向为 VSS
2. 正向为外部 VREF+管脚，负向为 VSS
3. 正向为 FVR 2.048V 输出，负向为 VSS
4. 正向为 FVR 4.096V 输出，负向为 VSS
5. 正向为内部 INTVREF 输出，负向为 VSS

6. 正向为内部 VDD，负向为 VREF-
7. 正向为外部 VREF+管脚，负向为 VREF-
8. 正向为 FVR 2.048V 输出，负向为 VREF-
9. 正向为 FVR 4.096V 输出，负向为 VREF-
10. 正向为内部 INTVREF 输出，负向为 VREF-

FVR 固定电压： 2.048V /4.096V

INTVREF 电压：1.0V

- ADC 模块图

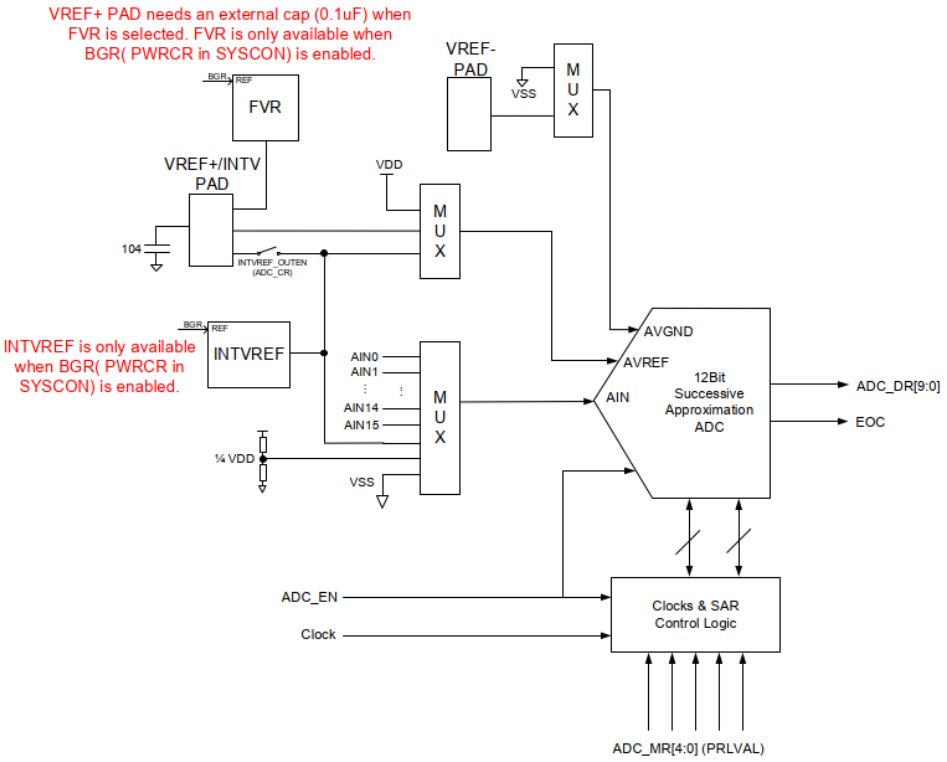

图 3.1.1ADC 框图

3.2 ADC 单次转换模式

系统主频 48MHZ，参考电压选择 VDD，ADC 通道 PIN39-PA0.1

```
/************************************************************/
const csi_adc_seq_t adcSeqCfg[] =
{   {ADCIN1,          ADC_CV_COUNT_1,          ADC_AVG_COF_1,          ADCSYNC_NONE},
};
//采样序列的通道数
volatile uint8_t     adcChnlNum = sizeof(adcSeqCfg)/sizeof(adcSeqCfg[0]);
volatile uint16_t    g_hwAdcBuf[16];
```

```
void adc_config(void)
{
    csi_adc_config_t tAdcConfig;
    csi_pin_set_mux(PA01, PA01_ADC_AIN1);

    tAdcConfig.byClkDiv = 0x02;
    tAdcConfig.bySampHold = 0x06;
    tAdcConfig.byConvMode = ADC_CONV_ONESHOT;
    tAdcConfig.byVrefSrc = ADCVERF_VDD_VSS;
    tAdcConfig.wInt = ADC_INTSRC_NONE;
    tAdcConfig.ptSeqCfg = (csi_adc_seq_t *)adcSeqCfg;

    csi_adc_init(ADC0, &tAdcConfig);
    csi_adc_set_seqx(ADC0, tAdcConfig.ptSeqCfg, adcChnlNum);
    csi_adc_start(ADC0);
    //
}
int main()
{
    mdelay(3000);
    //
    uint8_t i;
    system_init();
    board_init();
    //
    adc_config ();
     while(1)
     {
            g_hwAdcBuf[0] = csi_adc_read_channel(ADC0, 0);
            my_printf("ADC channel 0 value of seq: %d \n", g_hwAdcBuf[0]);
            //
            csi_adc_start(ADC0);
     }
 }
```

- **代码说明：**

  csi_pin_set_mux(); ---- 用于配置 ADC 管脚

  csi_adc_init(); ---- 用于配置 ADC 参数

  csi_adc_set_seqx(); ---- 用于配置 ADC 序列

csi_adc_start(); ---- 用于启动 ADC

- **函数参数说明：**

csi_pin_set_mux(PA01, PA01_ADC_AIN1);

csi_adc_init(ADC0, &tAdcConfig);

通过配置 tAdcConfig 此结构体来对 ADC 进行初始化配置。

tAdcConfig.byClkDiv //ADC clk 分频

tAdcConfig.bySampHold //ADC 采样时间

tAdcConfig.byConvMode //ADC 转换模式： 单次转换/连续转换

tAdcConfig.byVrefSrc //ADC 参考电压

tAdcConfig.wInt //ADC 中断配置

tAdcConfig.ptSeqCfg //ADC 采样序列

采样结果图：

g_hwAdcBuf[0]	2794	volatile uint16_t

图 3.2.1 采样结果

3.3 ADC 中断接收

配置 ADC 模块的 EOC 中断。

```
/************************************************************/
const csi_adc_seq_t adcSeqCfg[] =
{   {ADCIN1,        ADC_CV_COUNT_1,        ADC_AVG_COF_1,        ADCSYNC_NONE},
};
//采样序列的通道数
volatile uint8_t    adcChnlNum = sizeof(adcSeqCfg)/sizeof(adcSeqCfg[0]);
volatile uint16_t   g_hwAdcBuf[16];
```

```
    void adc_config(void)
    {
        csi_adc_config_t tAdcConfig;
        csi_pin_set_mux(PA01, PA01_ADC_AIN1);

        tAdcConfig.byClkDiv = 0x02;
        tAdcConfig.bySampHold = 0x06;
        tAdcConfig.byConvMode = ADC_CONV_ONESHOT;
        tAdcConfig.byVrefSrc = ADCVERF_VDD_VSS;
        tAdcConfig.wInt = ADC_INTSRC_EOC;
        tAdcConfig.ptSeqCfg = (csi_adc_seq_t *)adcSeqCfg;

        csi_adc_init(ADC0, &tAdcConfig);
        csi_adc_set_seqx(ADC0, tAdcConfig.ptSeqCfg, adcChnlNum);
        csi_adc_start(ADC0);
        //
    }
    int main()
    {
        mdelay(3000);
        //
        uint8_t i;
        system_init();
        board_init();
        //
        adc_config ();
         while(1)
         {

         }
}

__attribute__((weak)) void adc_irqhandler(csp_adc_t *ptAdcBase)
{
        uint32_t wIntStat = csp_adc_get_sr(ptAdcBase) & csp_adc_get_isr(ptAdcBase);
        if(wIntStat == ADC_INTSRC_EOC)
        {
                g_hwAdcBuf[0] = csi_adc_read_channel(ADC0, 0);
                //my_printf("ADC channel 0 value of seq: %d \n", g_hwAdcBuf[0]);
                //重新启动 ADC
                csi_adc_start(ADC0);
        }
}
```

- 参数说明

tAdcConfig.wInt = ADC_INTSRC_EOC; //使能 ADC 的 EOC 中断

- **测试结果：**

```
28  extern volatile uint16_t    g_hwAdcBuf[16];
29  /** \brief adc interrupt handle function
30   *
31   *  \param[in] ptAdcBase: pointer of adc register structure
32   *  \return none
33   */
34  __attribute__((weak)) void adc_irqhandler(csp_adc_t *ptAdcBase)
35  {
36      uint32_t wIntStat = csp_adc_get_sr(ptAdcBase) & csp_adc_get_isr(ptAdcBase);
37      if(wIntStat == ADC_INTSRC_EOC)
38      {
39          g_hwAdcBuf[0] = csi_adc_read_channel(ADC0, 0);
40          //my_printf("ADC channel 0 value of seq: %d \n", g_hwAdcBuf[0]);
41          //重新启动ADC
42          csi_adc_start(ADC0);
43      }
```

wCnt	<cannot evaluate
g_hwAdcBuf[0]	2818

图 3.3.1 测试结果

3.4 ADC 连续模式

```
/************************************************************/
const csi_adc_seq_t adcSeqCfg[] =
{   {ADCIN1,            ADC_CV_COUNT_1,             ADC_AVG_COF_1,          ADCSYNC_NONE},
};
//采样序列的通道数
volatile uint8_t    adcChnlNum = sizeof(adcSeqCfg)/sizeof(adcSeqCfg[0]);
volatile uint16_t   g_hwAdcBuf[16];

void adc_config(void)
{
    csi_adc_config_t tAdcConfig;
    csi_pin_set_mux(PA01, PA01_ADC_AIN1);

    tAdcConfig.byClkDiv = 0x02;
    tAdcConfig.bySampHold = 0x06;
    tAdcConfig.byConvMode =ADC_CONV_CONTINU;
    tAdcConfig.byVrefSrc = ADCVERF_VDD_VSS;
    tAdcConfig.wInt = ADC_INTSRC_NONE;
    tAdcConfig.ptSeqCfg = (csi_adc_seq_t *)adcSeqCfg;

    csi_adc_init(ADC0, &tAdcConfig);
    csi_adc_set_seqx(ADC0, tAdcConfig.ptSeqCfg, adcChnlNum);
    csi_adc_start(ADC0);
    //
}
int main()
```

```
{
    mdelay(3000);
    //
    uint8_t i;
    system_init();
    board_init();
    //
    adc_config ();
     while(1)
     {
            g_hwAdcBuf[0] = csi_adc_read_channel(ADC0, 0);
            my_printf("ADC channel 0 value of seq: %d \n", g_hwAdcBuf[0]);
     }
}
```

- **参数说明**

  tAdcConfig.byConvMode =ADC_CONV_CONTINU; //配置 ADC 模式为连续模式

- **测试结果**

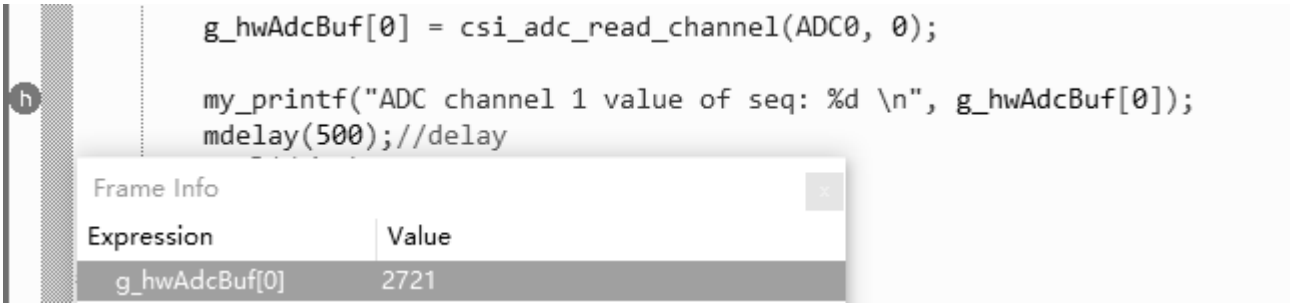

图 3.4.1 测试结果

4. 程序下载和运行

1．将目标板与仿真器连接，分别为 VDD SCLK SWIO GND

2. 程序编译后仿真运行

3. 通过 110x 学习 Demo 板查看变化数据。