

32 位微控制器

HC32L15 系列的中断处理机制

适用对象

系列	产品型号
HC32L15	HC32L150KATA
	HC32L150JATA
	HC32L150FAUA
	HC32L156KATA
	HC32L156JATA

目 录

1	摘要	3
2	HC32L15 系列的中断简介	3
3	HC32L15 系列的中断模块	5
3.1	简介	5
3.2	说明	5
3.2.1	寄存器介绍	6
3.2.2	工作流程介绍	7
4	样例代码	9
4.1	代码介绍	9
4.2	代码运行	14
5	总结	16
6	版本信息 & 联系方式	17

1 摘要

本篇应用笔记主要介绍 HC32L15 系列的中断处理并简要说明如何使用 HC32L15 的中断处理配合进行外设的操作。

2 HC32L15 系列的中断简介

什么是中断？

中断技术是一种使 CPU 中止正在执行的程序而转去处理特殊事件的操作技术。

（引自‘互动百科’）

HC32L15 系列 MCU

HC32L15 系列 MCU 是由华大半导体开发的基于 ARM Cortex M0+ Core 的低功耗 MCU，拥有多种定时器和多种通信外设，同时基于 MCU 的外设配合 Core 提供了较为丰富的中断源，为用户开发产品提供了更好的便利性和易用性。

Cortex M0+的中断处理部分特性

Cortex M0+ MCU 在内核中为用户提供了一个异常响应系统，支持各种系统异常和外部中断，编号 1-15 的为对应的系统异常，16 及以后的编号为外设中断。

本文主要介绍和中断相关的控制处理部分。

在 Cortex M0+中的处理中断相关的部分称作为 NVIC（Nested Vectored Interrupt Controller），除了包含控制寄存器和中断处理的控制逻辑之外，NVIC 还包含了 MPU、SysTick 定时器以及调试控制相关的寄存器。

其中 M0+的外设中断最多可以提供 32 个，用户可以通过相关的寄存器对此进行控制，有关寄存器主要如下：

- 使能/除能寄存器
- 挂起/解挂寄存器
- 优先级寄存器

- 活动状态寄存器

另外：

- 异常 Mask 寄存器
- 向量表偏移量寄存器
- 软件触发中断寄存器
- 优先级分组位段

相关寄存器的具体介绍读者可详见‘ARMv6-M-Architecture-Reference-Manual’以及相关的 ARM 介绍资料。

对于中断，Cortex M0+响应时的处理主要分为 3 个步骤：

1. 入栈，把 8 个寄存器的值压栈
2. 取向量，从中断向量表中取得对应中断的服务程序入口地址
3. 更新 SP，更新 PC

在完成了相应的中断处理，启动中断返回序列，主要为：

1. 出栈，将先前入栈的值重新写回寄存器
2. 更新 NVIC 寄存器
3. 更新 PC 并返回到中断前执行点

有关中断的嵌套、咬尾中断、晚到中断、中断延迟以及其他一些中断相关的问题读者可以参考‘ARMv6-M-Architecture-Reference-Manual’或者其他相关的中英文资料。

HC32L15 的中断简要说明

HC32L15 的中断基于 Cortex M0+的 NVIC，多个外设器件的中断信号可以集中并输入到 NVIC 的一个中断源向量，可使用中断请求批量读取寄存器确认中断请求是否发生。

3 HC32L15 系列的中断模块

3.1 简介

华大 HC32L15 系列单片机内部包含了 Cortex M0+ 的 NVIC，同时搭载了 MCU 丰富的外设中断，主要提供：

- 32 个可屏蔽的外设中断通道(不包含 16 个 Cortex-M0 异常中断)
- 4 个可编程的中断优先级(使用 2 位配置中断优先级)
- 实现低延迟异常和中断处理
- 支持不可屏蔽中断(NMI)输入

3.2 说明

本节介绍 HC32L15 系列的中断模块，包括寄存器和工作流程。

3.2.1 寄存器介绍

针对中断处理部分的寄存器主要包括两个部分：

1) NVIC 寄存器

NVIC_ISER：中断使能寄存器

NVIC_ICER：中断除能寄存器

NVIC_ISPR：中断挂起寄存器

NVIC_ICPR：中断解挂寄存器

NVIC_IPR0 - NVIC_IPR7：中断优先级寄存器

以上寄存器均为 NVIC 标准寄存器，具体含义请参见‘ARMv6-M-Architecture-Reference-Manual’。

针对这些寄存器的操作函数，包括 NVIC_EnableIRQ，NVIC_DisableIRQ，NVIC_GetPendingIRQ，NVIC_SetPendingIRQ，NVIC_ClearPendingIRQ，NVIC_SetPriority，NVIC_GetPriority，NVIC_SystemReset 等都置于 core_cm0plus.h 中，用户可参考函数说明进行操作。

2) HC32L15 中断批量读取寄存器

每个中断批量读取寄存器都对应 NVIC 的一个中断源向量，多个外设的中断可以对应到一个读取寄存器，用户可以读取并判断该寄存器中各个外设中断的中断状态，所有该系列寄存器均为只读寄存器。

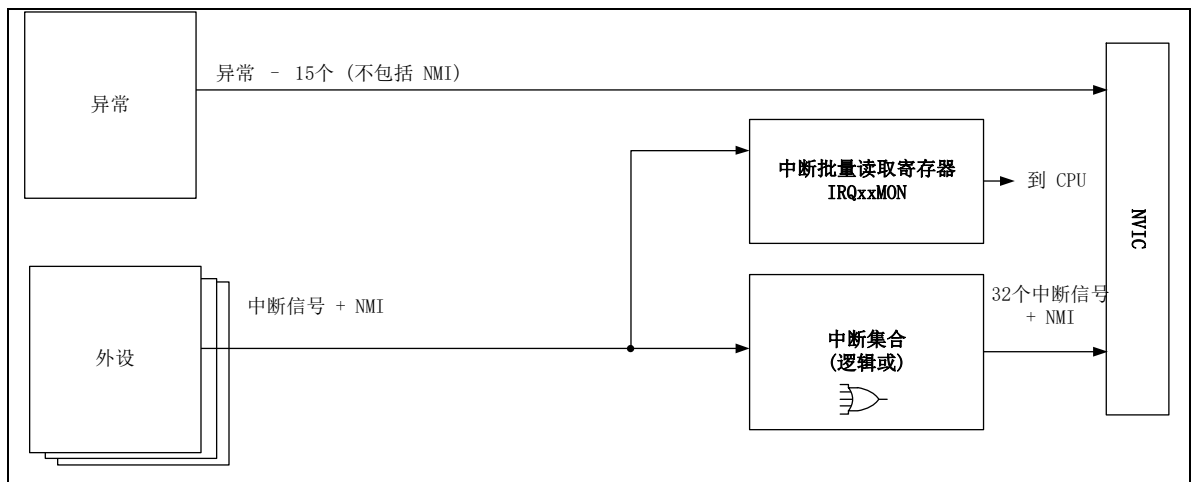
针对每个寄存器的内容说明请详细参见 HC32L15 的用户手册。

3.2.2 工作流程介绍

中断控制器说明

整个异常可分为异常和外设中断两大部分，系统会根据外设的中断（包括 NMI）请求设置中断批量读取寄存器，同时将相应的中断请求发送到 NVIC 以便进行中断处理（挂起/响应 /...）。

中断控制器概图



中断具体操作

设置：

在中断的设置阶段，用户可以根据自己的需要设置中断的优先级分组（寄存器名称为 AIRCR，地址为 0xE000ED0C）；中断的向量表偏移量地址（寄存器名称为 VTOR，地址为 0xE000ED08）。

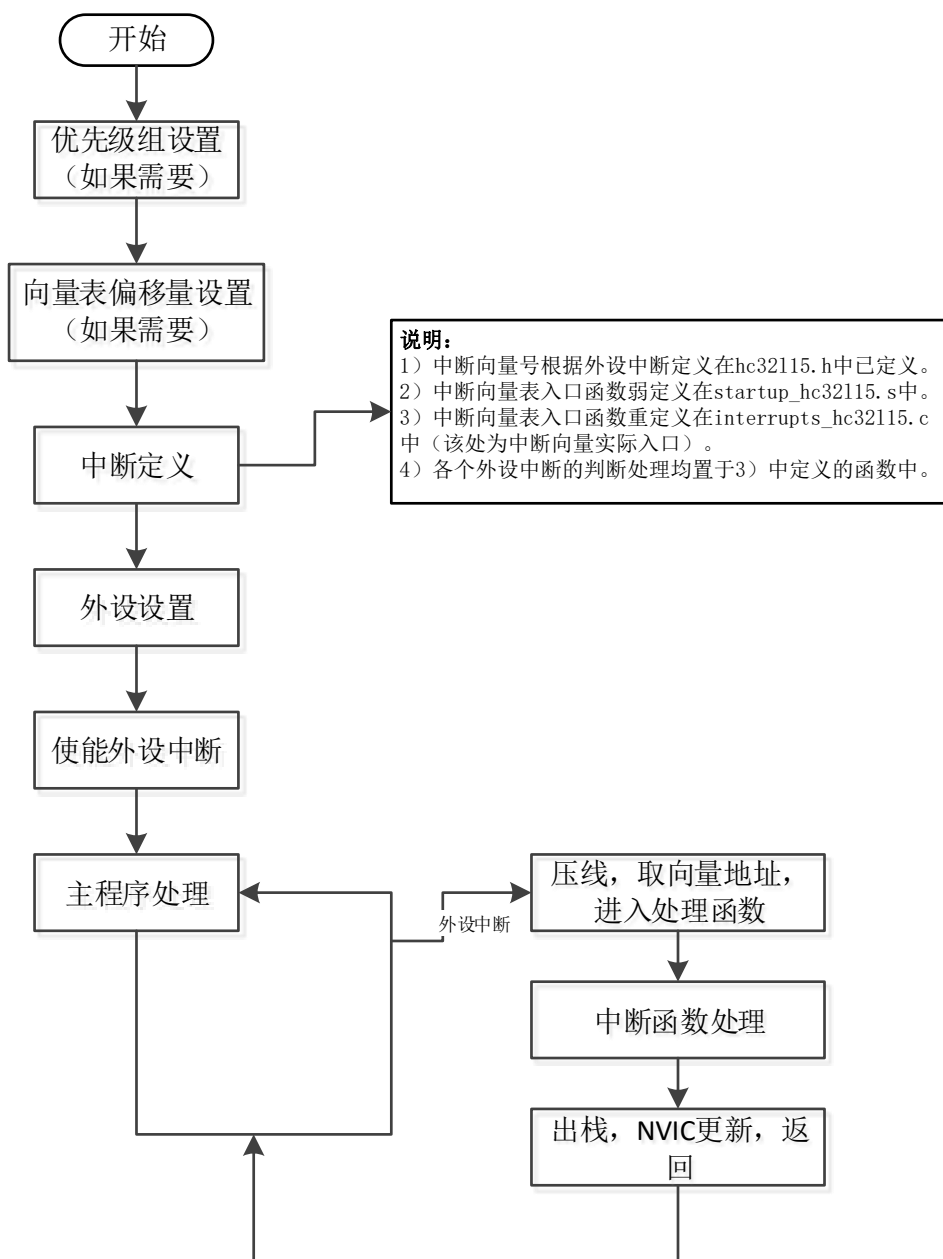
定义：

在工程中中断向量表完成具体定义，包括：hc32l15.h 中的中断号定义和 startup_hc32l15.s 中的中断向量入口定义（该处采用弱定义方法，使该处定义的中断向量入口可在其他函数中重新被定义）；在 interrupts_hc32l15.c 中对中断向量表函数进行了重新定义。

使用：

如果欲使用某外设中断，需要在该外设的相应寄存器中使能中断产生位，并找到该外设的对应中断号，使用 NVIC 设置该中断的中断优先级并使能该中断，在使用中发生中断后就按照本 AN 中第二章所描述的 CM0+中断处理方法进行进入中断和退出中断的处理，另外需要注意的是有的外设中的中断产生后需要手动清除中断标志位以免重复进入。

中断定义设置以及处理过程



4 样例代码

4.1 代码介绍

用户可以根据上述的工作流程编写自己的代码来学习验证该模块，也可以直接通过华大半导体的网站下载 HC32L15 系列 MCU 的 DDL 来体验中断的实现和执行。

以下部分主要基于 DDL 的外设 UART 简要介绍中断的使用方法：

首先是定义部分：

1) DDL 中的中断号定义 (hc32l15.h)

```

/*****
* Interrupt Number Definition
*****/
typedef enum IRQn
{
    NMI_IRQn      = -14, /* 2 Non Maskable */
    HardFault_IRQn = -13, /* 3 Hard Fault */
    SVC_IRQn       = -5, /* 11 SV Call */
    PendSV_IRQn    = -2, /* 14 Pend SV */
    SysTick_IRQn   = -1, /* 15 System Tick */

    CSV_IRQn       = 0, /* Clock Super Visor */
    SWDT_IRQn      = 1, /* Software Watchdog Timer */
    LVD_IRQn       = 2, /* Low Voltage Detector */
    /* Reserved    = 3,
    EXINT0_7_IRQn  = 4, /* External Interrupt Request ch.0 to ch.7 */
    EXINT8_15_IRQn = 5, /* External Interrupt Request ch.8 to ch.15 */
    DTIM_IRQn      = 6, /* Dual Timer */
    MSC0RX_IRQn    = 7, /* MultiFunction Serial Reception ch.0 */
    MSC0TX_IRQn    = 8, /* MultiFunction Serial Transmission/Status ch.0 */
    MSC1RX_IRQn    = 9, /* MultiFunction Serial Reception ch.1 */
    MSC1TX_IRQn    = 10, /* MultiFunction Serial Transmission/Status ch.1 */
    MSC2RX_IRQn    = 11, /* MultiFunction Serial Reception ch.2 */
    MSC2TX_IRQn    = 12, /* MultiFunction Serial Transmission/Status ch.2 */
    /* Reserved    = 13
    /* Reserved    = 14
    MSC4RX_IRQn    = 15, /* MultiFunction Serial Reception ch.4 */
    MSC4TX_IRQn    = 16, /* MultiFunction Serial Transmission/Status ch.4 */
    MSC5RX_IRQn    = 17, /* MultiFunction Serial Reception ch.5 */
    MSC5TX_IRQn    = 18, /* MultiFunction Serial Transmission/Status ch.5 */
    MSC6RX_IRQn    = 19, /* MultiFunction Serial Reception ch.6 */
    MSC6TX_IRQn    = 20, /* MultiFunction Serial Transmission/Status ch.6 */
    /* Reserved    = 21
    /* Reserved    = 22
    /* Reserved    = 23
    OSC_WU_RTC_IRQn = 24, /* OSC / MSC Wake Up / RTC */
    ADC0_IRQn      = 25, /* ADC0 */
    VC_IRQn        = 26, /* Voltage Comparator */
    SCI_IRQn       = 27, /* SCI7816 */
    PCNT_IRQn      = 28, /* Pulse Counter Timer */
    /* Reserved    = 29
    /* Reserved    = 30
    CTIM0_7_FLASH_IRQn = 31, /* Composite Timer ch.0 to ch.7 / Flash */
} IRQn_Type;

```

2) DDL 中的中断函数弱定义 (startup_hc32115.s, 以 UART 通道 0 为例) :

```
PUBWEAK IRQ007_Handler
    SECTION .text:CODE:NOROOT:REORDER(1)
IRQ007_Handler
    B    IRQ007_Handler

    PUBWEAK IRQ008_Handler
    SECTION .text:CODE:NOROOT:REORDER(1)
IRQ008_Handler
    B    IRQ008_Handler
```

3) DDL 中的中断函数重命名 (ddl.h), 实际函数入口地址定义:

```
#define MSC0RX_IRQHandler(void)    IRQ007_Handler(void) ///< MSC ch.0, RX
#define MSC0TX_IRQHandler(void)    IRQ008_Handler(void) ///< MSC ch.0, TX
```

4) DDL 中的中断函数重定义 (interrupts_hc32115.c), 实际函数入口:

```
/*
*****
***** MSC0 - RX *****
*****
*/
void MSC0RX_IRQHandler(void)
{
    uint8_t u8Mode = 0;
    u8Mode = M0P_MSC0_UART->UART_MR_f.MODE;
    MSCRX_IRQHander2(0,u8Mode);
}

/*
*****
***** MSC0 - TX *****
*****
*/
void MSC0TX_IRQHandler(void)
{
    uint8_t u8Mode = 0;
    uint32_t u32Val = 0;

    u8Mode = M0P_MSC0_UART->UART_MR_f.MODE;
    u32Val = M0P_INTREQ->IRQ08PEND;

    if (0u != (u32Val & 0x2u))
    {
        MSCSTAT_IRQHander2(0,u8Mode);
    }
    if (0u != (u32Val & 0x1u))
    {
        MSCTX_IRQHander2(0,u8Mode);
    }
}
```

- 5) DDL 中 UART 相关中断的具体分类处理部分 (interrupts_hc32l15.c)，中断分类处理部分：

```
void MSCRX_IRQHandler2(uint8_t u8ch,uint8_t u8mode)
{
    switch (u8mode)
    {
        case 0:
            Uart_Rx_IrqHandler(u8ch);
            break;
        case 1:
            ///< \todo
            break;
        case 2:
            Spi_Rx_IrqHandler(u8ch);
            break;
        case 4:
            I2c_Rx_IrqHandler(u8ch);
            break;
        default:
            break;
    }
}

void MSCTX_IRQHandler2(uint8_t u8ch,uint8_t u8mode)
{
    switch (u8mode)
    {
        case 0:
            Uart_Tx_IrqHandler(u8ch);
            break;
        case 1:
            ///< \todo
            break;
        case 2:
            Spi_Tx_IrqHandler(u8ch);
            break;
        case 4:
            I2c_Tx_IrqHandler(u8ch);
            break;
        default:
            break;
    }
}
```

- 6) DDL 中的 UART 驱动代码中中断处理函数 (uart.c)，外设中断按位判断并处理：

```
void Uart_Rx_IrqHandler(uint8_t u8Idx)
{
    stc_uart_instance_data_t *pstcData = NULL;

    ASSERT(IS_VALID_CH(u8Idx));

    pstcData = UartGetInternDataPtr(u8Idx);
    if (NULL == pstcData)
    {
        return;
    }

    //Exception handling
    if((1u == pstcData->pstcInstance->UART_SR_f.ORF) ||
        (1u == pstcData->pstcInstance->UART_SR_f.FEF) ||
        (1u == pstcData->pstcInstance->UART_SR_f.PEF))
```

```
{
    if(NULL != pstcData->stcUartInternIrqCb.pfnRxErrIrqCb)
    {
        pstcData->stcUartInternIrqCb.pfnRxErrIrqCb();
    }
}

//Normal case
if(1u == pstcData->pstcInstance->UART_SR_f.RDFF)
{
    if(NULL != pstcData->stcUartInternIrqCb.pfnRxIrqCb)
    {
        pstcData->stcUartInternIrqCb.pfnRxIrqCb();
    }
}
if(1u == pstcData->pstcInstance->UART_CR2_f.CTSF)
{
    if(NULL != pstcData->stcUartInternIrqCb.pfnCtsCb)
    {
        pstcData->stcUartInternIrqCb.pfnRxIrqCb();
    }
}
if(1u == pstcData->pstcInstance->UART_CR2_f.SINF)
{
    if(NULL != pstcData->stcUartInternIrqCb.pfnSinCb)
    {
        pstcData->stcUartInternIrqCb.pfnSinCb();
    }
}
} /* UartIrqHandlerRx */

void Uart_Tx_IrqHandler(uint8_t u8Idx)
{
    stc_uart_instance_data_t *pstcData = NULL;

    ASSERT(IS_VALID_CH(u8Idx));

    pstcData = UartGetInternDataPtr(u8Idx);
    if (NULL == pstcData)
    {
        return;
    }

    if(1u == pstcData->pstcInstance->UART_SR_f.TDEF)
    {
        if(NULL != pstcData->stcUartInternIrqCb.pfnTxIrqCb)
        {
            pstcData->stcUartInternIrqCb.pfnTxIrqCb();
        }
    }

    if(1u == pstcData->pstcInstance->UART_SR_f.TBIF)
    {
        if(NULL != pstcData->stcUartInternIrqCb.pfnTxIdleCb)
        {
            pstcData->stcUartInternIrqCb.pfnTxIdleCb();
        }
    }
} /* UartIrqHandlerTx */
```

上述代码提供了完整的中断定义（针对 UART 通道 0 外设部分）。

其次为使用部分：

- 1) 首先需要定义中断处理程序中会使用到的回调函数变量（main.c）：

```
stc_uart_irq_cb_t stcUart0IrqCb, stcUart1IrqCb;
```

- 2) 定义回调函数地址（main.c）：

```
//Init the UART
stcUart0IrqCb.pfnTxIrqCb = Uart0TxIntCallback;
stcUart1IrqCb.pfnRxIrqCb = Uart1RxIntCallback;
```

- 3) 初始化 UART 外设并使能中断（main.c）：

```
Uart_Init(UARTCH0, &stcUart0Config);
...
Uart_Init(UARTCH1, &stcUart1Config);
...
/* Configure interrupt */
Uart_EnableIrq(UARTCH1, UartRxIrq);
Uart_EnableIrq(UARTCH0, UartTxIrq);
```

- 4) 定义回调函数具体内容（main.c），该部分为针对该 UART 外设的中断的用户处理程序：

```
void Uart0TxIntCallback(void)
{
    Uart_SendData(UARTCH0, u8TxData[u8TxCnt]);
    u8TxCnt++;
    if(u8TxCnt > 9)
    {
        /* Disable transfer interrupt of UART0 */
        Uart_DisableIrq(UARTCH0, UartTxIrq);
        return;
    }
}
...
void Uart1RxIntCallback(void)
{
    u8RxData[u8RxCnt] = Uart_ReceiveData(UARTCH1);
    u8RxCnt++;
    if(u8RxCnt > 9)
    {
        /* Disable transfer interrupt of UART0 */
        Uart_DisableIrq(UARTCH1, UartRxIrq);
        return;
    }
}
```

通过以上代码即可完成 UART 两个之间的中断方式收发。

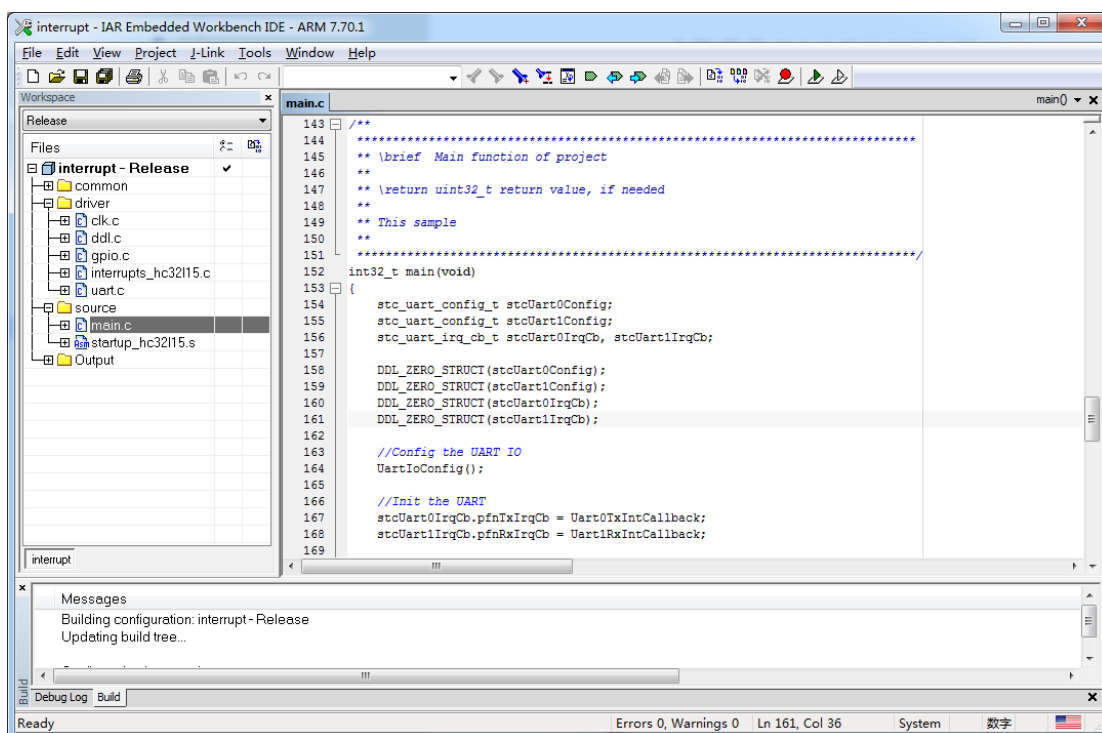
4.2 代码运行

用户可以通过华大半导体的网站下载到 HC32L15 的 DDL 的样例代码（UART），并配合学习板（比如‘SK-HC32L156-64L V10’）运行相关代码学习使用 UART 外设的中断收发。

以下部分主要介绍如何在‘SK-HC32L156-64L V10’学习板上运行 UART 样例代码并观察结果：

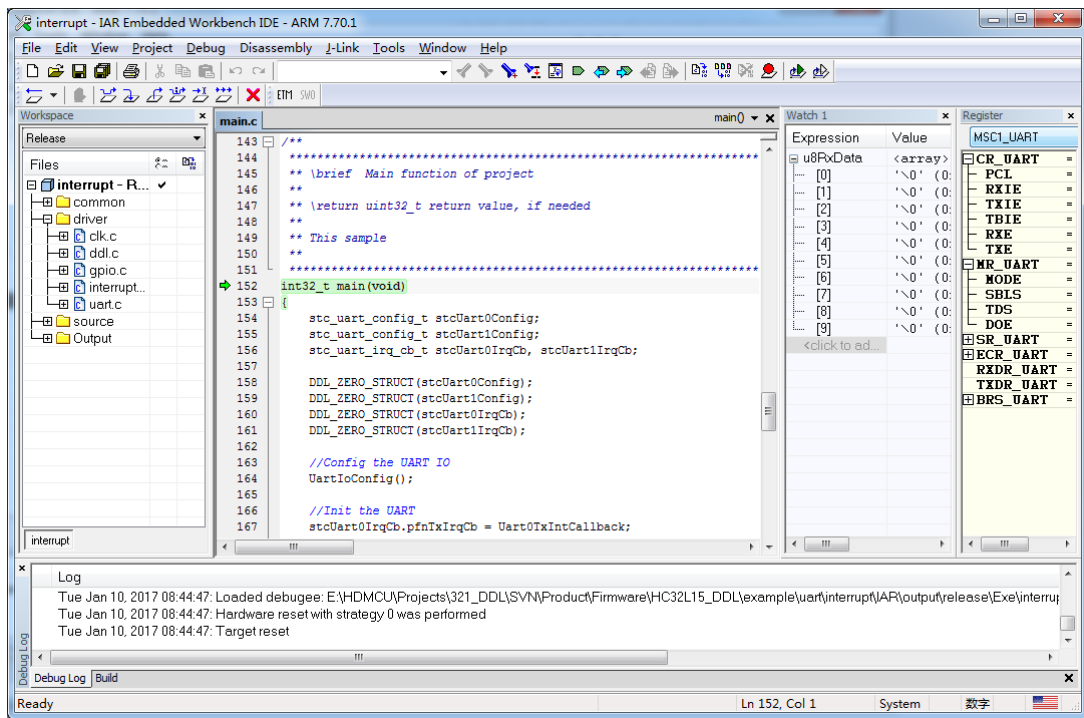
- 确认安装正确的 IAR EWARM v7.7 工具（请从 IAR 官方网站下载相应的安装包并参考用户手册进行安装。
- 获取‘SK-HC32L156-64L V10’学习板。
- 从华大半导体网站下载 HC32L15 DDL 代码。
- 下载并运行 uart\interrupt\中的项目文件。

1) 打开 uart\interrupt\项目，并打开‘main.c’如下视图：

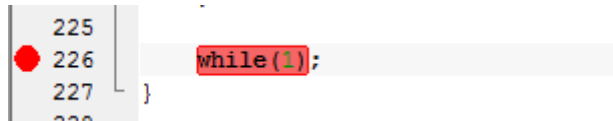



2) 点击  重新编译整个项目并将代码下载到学习板上；

3) 可以看见类似如下的视图:

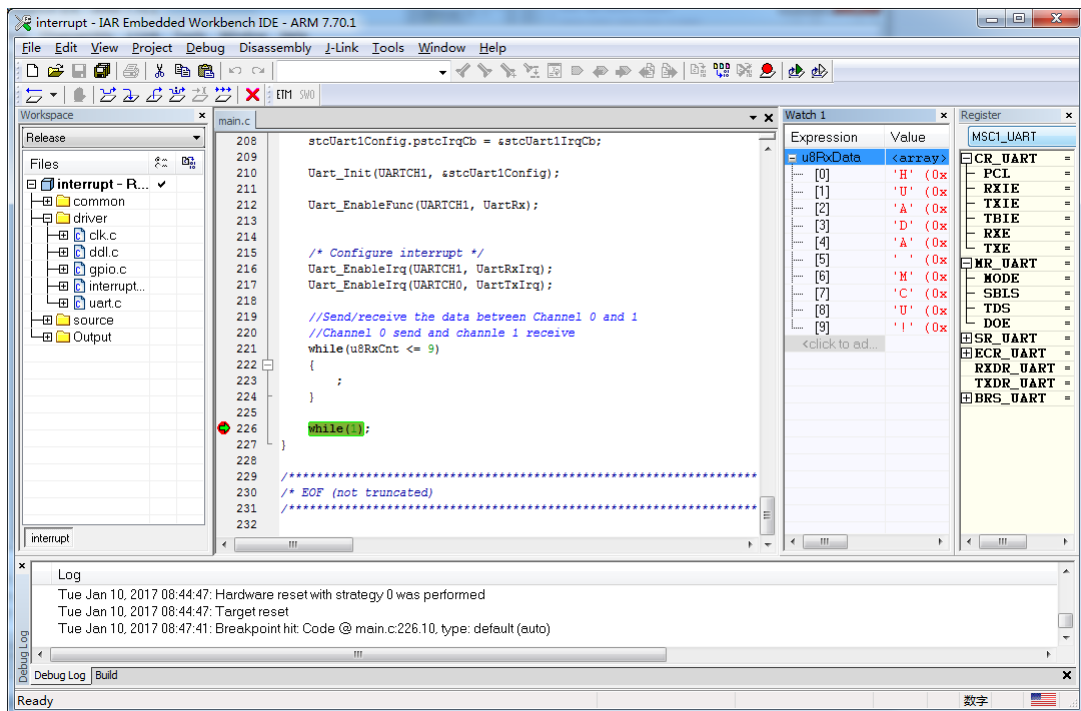


4) 在'main.c'的第 226 行设置断点, 如下图所示:



5) 点击  运行。

6) 代码运行并会停止在‘main.c’的断点处，观察并记录‘u8RxData’的值。



7) 该样例代码通过 UART 的中断方式将从通道 0 发送的数据通过通道 1 进行接收。

8) 用户也可以重新运行代码并在中断回调函数（Uart0TxIntCallback，Uart1RxIntCallback）中设置断点观察中断的发生。

9) 用户可以阅读代码进行具体的中断相关的设定使用。

5 总结

以上章节简要介绍了中断处理，详细说明了 HC32L15 的中断模块并且演示了如何使用相关的 UART 样例代码学习中断的设定使用，在开发中用户可以根据自己的实际需要使用该中断模块。

6 版本信息 & 联系方式

日期	版本	修改记录
2018/8/9	Rev1.0	初版发布。



如果您在购买与使用过程中有任何意见或建议，请随时与我们联系。

Email: mcu@hdsc.com.cn

网址: www.hdsc.com.cn

通信地址: 上海市张江高科园区碧波路 572 弄 39 号

邮编: 201203

