在使用 CMSDK 搭建的 Cortex-M3 SoC 上添加自定义 IP

这篇文档将以挂载一个数码管显示模块为例，介绍如何在使用 CMSDK 搭建的 Cortex-M3 SoC 上添加自定义 IP，并引入键盘中断模块添加外部中断，再结合 UART 实现数码管多模式显示的功能。本次实验将在如图 1 所示的使用 CMSDK 构建的 SoC 的基础上进行，数码管显示模块将挂载到第二级 AHB 总线矩阵上。

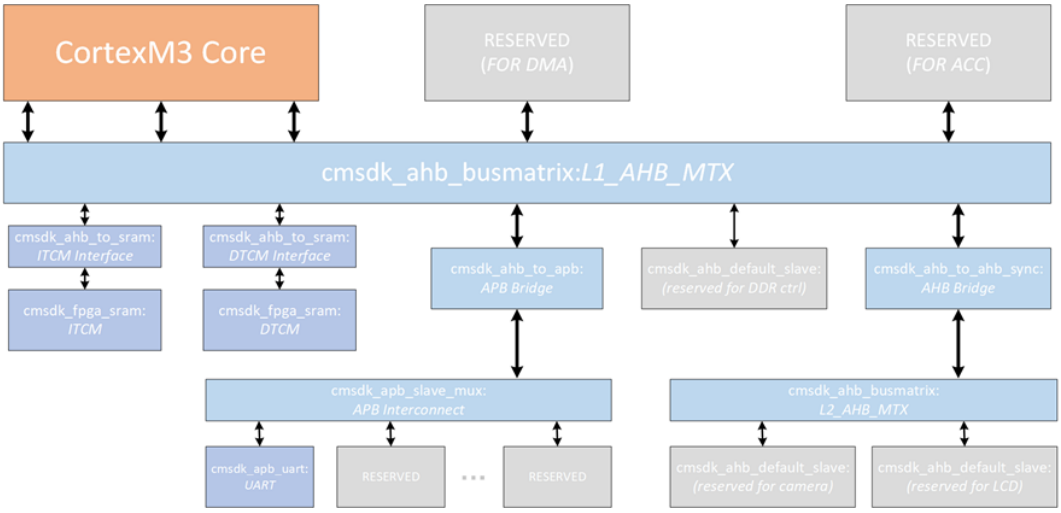

图 1.SOC 架构

- 第一步：由于二级 AHB 总线矩阵目前的两个接口是预留给其他外设的，所以需要进行二级 AHB 总线矩阵的扩展。在 cmsdk_ahb_busmatrix_l2.xml（“CortexM3/cmsdk_anb_busmatrix/xml/cmsdk_ahb_busmatrix_l2.xml”）的基础上修改编写 XML 文件，声明新增的 M 端口名，并编写 M 端口的地址映射，如图 2 所示，新增了 M2 端口，其地址空间为 0x50020000~0x50020fff。此处需注意，不能分配小于 1K Byte 的地址空间，否则后续会报错。

```
<!-- Slave interface definitions -->

<!-- AHBBRIDGE SLAVE -->
<slave_interface name="S0">
  <sparse_connect interface="M0"/>
  <sparse_connect interface="M1"/>
  <sparse_connect interface="M2"/>
  <address_region interface="M0" mem_lo="50000000" mem_hi="5000ffff" remapping='none'/>
  <address_region interface="M1" mem_lo="50010000" mem_hi="5001ffff" remapping='none'/>
  <address_region interface="M2" mem_lo="50020000" mem_hi="50020fff" remapping='none'/>
</slave_interface>

<!-- Master interface definitions -->

<!-- CAMERA SLAVE -->
<master_interface name="M0"/>

<!-- SCREEN SLAVE -->
<master_interface name="M1"/>

<!-- SEG SLAVE -->
<master_interface name="M2"/>

<!- - - - - *** DO NOT MODIFY BELOW THIS LINE *** - - - - - - - - - - - - -->
```

图 2.xml 代码示例

- 第三步：在 cmsdk_ahb_busmatrix/目录下修改编写 makefile 文件，将 xml 文

件名改为上一步修改好的的 cmsdk_ahb_busmatrix_l2.xml，如图 3 所示。

```
all:
    sudo bin/BuildBusMatrix.pl -xmldir xml -cfg cmsdk_ahb_busmatrix_l2.xml -over -verbose
```

图 3.makefile 文件示例

- 第四步：在 cmsdk_ahb_busmatrix/目录下打开 WSL 执行 make，成功执行后便会生成新的总线矩阵并删除旧文件，如图 4 所示。

```
milo@DESKTOP-QGH67GH:/mnt/c/Users/Milo/Desktop/CortexM3_tst/cmsdk_ahb_busmatrix$ make
sudo bin/BuildBusMatrix.pl -xmldir xml -cfg cmsdk_ahb_busmatrix_l2.xml -over -verbose

============================================================
# This confidential and proprietary software may be used only
# as authorised by a licensing agreement from ARM Limited
#    (C) COPYRIGHT 2001-2013 ARM Limited
#          ALL RIGHTS RESERVED
# The entire notice above must be reproduced on all authorised
# copies and copies may only be made to the extent permitted
# by a licensing agreement from ARM Limited
#
# BuildBusMatrix.pl
#
# Run Date : 29/02/2020 16:24:27
============================================================

Warning: This configuration has one slave port and will use 'single' output
 and arbiter stage(s) only.
Script accepted the following parameters:

 - Configuration file      : 'xml/cmsdk_ahb_busmatrix_l2.xml'
 - Top-level name          : 'L2AhbMtx'
 - Slave interfaces        : 1
 - Master interfaces       : 3
 - Architecture type       : 'ahb2'
 - Arbitration scheme      : 'round'
 - Address map             : user defined
 - Connectivity mapping    : S0 -> M0, M1, M2
 - Connectivity type       : full
 - Routing data width      : 32
 - Routing address width   : 32
 - User signal width       : 32
 - Configuration directory : './verilog/built'
 - Source directory        : './verilog/src'
 - Overwrite mode          : enabled

Deleting the './verilog/built/L2AhbMtx/L2AhbArb.v' file...
Deleting the './verilog/built/L2AhbMtx/L2AhbDecS0.v' file...
Deleting the './verilog/built/L2AhbMtx/L2AhbInStg.v' file...
Deleting the './verilog/built/L2AhbMtx/L2AhbMtx.v' file...
Deleting the './verilog/built/L2AhbMtx/L2AhbMtx_default_slave.v' file...
Deleting the './verilog/built/L2AhbMtx/L2AhbMtx_lite.v' file...
Deleting the './verilog/built/L2AhbMtx/L2AhbOutStg.v' file...

Creating the bus matrix variant...

 - Rendering 'L2AhbDecS0.v'
 - Rendering 'L2AhbArb.v'
 - Rendering 'L2AhbMtx_default_slave.v'
 - Rendering 'L2AhbOutStg.v'
 - Rendering 'L2AhbMtx_lite.v'
 - Rendering 'L2AhbInStg.v'
 - Rendering 'L2AhbMtx.v'
```

图 4.在 WSL 成功执行 make 后的编译信息

- 第五步：生成新的总线矩阵后需要在顶层文件 CortexM3.v 中修改其例化的代码，如图 5 所示，并添加新增端口的总线信号声明，如图 6 所示。

```
L2AhbMtx    L2AhbMtx(
    .HCLK                               (clk),
    .HRESETn                            (cpuresetn),

    .REMAP                              (4'b0),

    .HSELS0                             (1'b1),
    .HADDRS0                            (HADDR_AHBL2M),
    .HTRANSS0                           (HTRANS_AHBL2M),
    .HWRITES0                           (HWRITE_AHBL2M),
    .HSIZES0                            (HSIZE_AHBL2M),
```

图 5.将 HSELS0 接到高电平上

```
wire    [31:0]  HADDR_AHBL2P2;
wire    [1:0]   HTRANS_AHBL2P2;
wire            HWRITE_AHBL2P2;
wire    [2:0]   HSIZE_AHBL2P2;
wire    [31:0]  HWDATA_AHBL2P2;
wire    [2:0]   HBURST_AHBL2P2;
wire    [3:0]   HPROT_AHBL2P2;
wire            HREADY_AHBL2P2;
wire    [31:0]  HRDATA_AHBL2P2;
wire    [1:0]   HRESP_AHBL2P2;
wire            HREADYOUT_AHBL2P2;
wire            HSEL_AHBL2P2;
wire    [1:0]   HMASTER_AHBL2P2;
wire            HMASTERLOCK_AHBL2P2;
```

图 6.新增端口的 wire 声明

- 第六步：将自己的自定义 IP 和其 AHB 接口的 Verilog HDL 文件加入到工程中，此处以数码管显示模块为例，并在顶层文件中添加例化，如图 7 所示。

```
//------------------------------------------------------------
// AHB SEG
//------------------------------------------------------------

wire [1:0] mode;
wire [6:0] preset_data;
ahb_to_seg ahbseg(
    .HCLK                       (clk),
    .HRESETn                    (cpuresetn),
    .HSEL                       (HSEL_AHBL2P2),
    .HADDR                      (HADDR_AHBL2P2),
    .HPROT                      (HPROT_AHBL2P2),
    .HSIZE                      (HSIZE_AHBL2P2),
    .HTRANS                     (HTRANS_AHBL2P2),
    .HWDATA                     (HWDATA_AHBL2P2),
    .HWRITE                     (HWRITE_AHBL2P2),
    .HRDATA                     (HRDATA_AHBL2P2),
    .HREADY                     (HREADY_AHBL2P2),
    .HREADYOUT                  (HREADYOUT_AHBL2P2),
    .HRESP                      (HRESP_AHBL2P2[0]),
    .mode                       (mode),
    .preset_data                (preset_data)
);

seg_disp seg_disp(
    .clk                        (clk),
    .rstn                       (cpuresetn),
    .preset_data                (preset_data),
    .mode                       (mode),
    .seg                        (seg),
    .an                         (an)
);
```

图 7.数码管显示模块和其 AHB 接口的例化

- 第七步：添加键盘中断模块，引入四个键盘按键的外部中断，如图 8 所示

```
//------------------------------------------------------------------------
// KEYBOARD
//------------------------------------------------------------------------

wire [3:0] key_interrupt;
assign  row = 4'b1110;
Keyboard Keyboard_IRQ(
        .HCLK                   (clk),
        .HRESETn                (RSTn),
        .col                    (~col),
        .key_interrupt          (key_interrupt)
);

//------------------------------------------------------------------------
// INTERRUPT
//------------------------------------------------------------------------

assign  IRQ     =   {233'b0,key_interrupt,TXOVRINT|RXOVRINT,TXINT,RXINT};
```

图 8.键盘模块的例化和中断的添加

- 第八步：在顶层文件的输入输出端口中添加数码管和键盘的输入输出端口的声明，如图 9 所示。到这步为止，硬件代码部分就无需再做其他改动，增加数码管和键盘的管脚约束后便可使用 EDA 工具进行编译仿真综合等并最终下载到 FPGA 开发板上。

```
// SEG
output      wire    [7:0]    seg,
output      wire    [1:0]    an,

// KEYBOARD
input       wire    [3:0]    col,
output      wire    [3:0]    row,
```

图 9.输入输出端口声明

- 第九步：从这一步开始，将对 Keil 工程中的软件代码进行修改以实现预期的功能。首先修改 software 文件夹下的 SoC 启动代码 startup_CM3DS.s，添加键盘的四个按键中断，如图 10 和图 11 所示。图 11 中实现的是按键中断触发时，向数码管模式寄存器直接写入数据从而令数码管切换到相应的模式，四个按键分别对应暂停、向下计数、向上计数和置数模式。

```
; External Interrupts
DCD     UARTRX_Handler          ; UART RX Handler
DCD     UARTTX_Handler          ; UART TX Handler
DCD     UARTOVR_Handler         ; UART RX and TX OVERRIDE Handler
DCD     KEY0_Handler            ; KEY0_Handler
DCD     KEY1_Handler            ; KEY1_Handler
DCD     KEY2_Handler            ; KEY2_Handler
DCD     KEY3_Handler            ; KEY3_Handler
```

图 10.在启动代码添加键盘中断

```
KEY0_Handler    PROC
                PUSH    {R0,R1,LR}
                LDR      R0, =0x50020000
                MOVS     R1, #0                 ; Set SEG mode as hold
                STR      R1, [R0]
                POP     {R0,R1,PC}
                ENDP

KEY1_Handler    PROC
                PUSH    {R0,R1,LR}
                LDR      R0, =0x50020000
                MOVS     R1, #1                 ; Set SEG mode as countdown
                STR      R1, [R0]
                POP     {R0,R1,PC}
                ENDP

KEY2_Handler    PROC
                PUSH    {R0,R1,LR}
                LDR      R0, =0x50020000
                MOVS     R1, #2                 ; Set SEG mode as countup
                STR      R1, [R0]
                POP     {R0,R1,PC}
                ENDP

KEY3_Handler    PROC
                PUSH    {R0,R1,LR}
                LDR      R0, =0x50020000
                MOVS     R1, #3                  ; Set SEG mode as preset
                STR      R1, [R0]
                POP     {R0,R1,PC}
                ENDP
```

图 11.在启动代码中添加键盘中断

- 第十步：要使设置的中断起作用，还需要对新增的中断进行使能。在 CortexM3.h 这个头文件中，枚举四个新增按键中断，如图 12 所示。然后在 system.c 文件中修改代码，在 systemInit 函数中利用 NVIC_EnableIRQ 函数完成对四个中断的使能。NVIC_EnableIRQ 函数已在 core_cm3.h 这个头文件中进行了定义和说明。

```
typedef enum IRQn
{
/******  Cortex-M3 Processor Exceptions Numbers ***********************************************/
  NonMaskableInt_IRQn           = -14,    /*!<  2 Cortex-M3 Non Maskable Interrupt                        */
  HardFault_IRQn                = -13,    /*!<  3 Cortex-M3 Hard Fault Interrupt                          */
  MemoryManagement_IRQn         = -12,    /*!<  4 Cortex-M3 Memory Management Interrupt                   */
  BusFault_IRQn                 = -11,    /*!<  5 Cortex-M3 Bus Fault Interrupt                           */
  UsageFault_IRQn               = -10,    /*!<  6 Cortex-M3 Usage Fault Interrupt                         */
  SVCall_IRQn                   = -5,     /*!< 11 Cortex-M3 SV Call Interrupt                           */
  DebugMonitor_IRQn             = -4,     /*!< 12 Cortex-M3 Debug Monitor Interrupt                     */
  PendSV_IRQn                   = -2,     /*!< 14 Cortex-M3 Pend SV Interrupt                           */
  SysTick_IRQn                  = -1,     /*!< 15 Cortex-M3 System Tick Interrupt                       */

/******  CM3DS_MPS2 Specific Interrupt Numbers ************************************************/
  UARTTX_IRQn                   = 0,       /* UART 0 RX and TX Combined Interrupt   */
  UARTRX_IRQn                   = 1,       /* Undefined                             */
  UARTOVR_IRQn                  = 2,       /* UART 1 RX and TX Combined Interrupt   */
  KEY0_Handler                  = 3,
  KEY1_Handler                  = 4,
  KEY2_Handler                  = 5,
  KEY3_Handler                  = 6,
} IRQn_Type;
```

图 12.添加键盘中断

```
void SystemInit (void)
{


  SystemCoreClock = __SYSTEM_CLOCK;

  NVIC_EnableIRQ(KEY0_Handler);

  NVIC_EnableIRQ(KEY1_Handler);

  NVIC_EnableIRQ(KEY2_Handler);

  NVIC_EnableIRQ(KEY3_Handler);

  uart_init ( UART, (SystemCoreClock / 115200), 1,1,1,1,1,1 );


}
```

图 13.使能中断

- 第十一步：修改 main 函数，如图 14 所示。通过这一步在原来的基础上新增了如下功能：通过 PC 键盘输入数据，将数据通过 UART 传入 SoC 以控制数码管预置数模式下置数的数值。

```
#include "CortexM3.h"

int main(void) {
    printf("******************\n");
    printf("cortex-m3 startup!\n");
    printf("******************\n");
    int buf;
    int *p;
    p = (int *)0x50020004;
    while(1) {
        scanf("%d",&buf);
        if (buf>=0 && buf<=99) {
          printf("The SEG preset value is : %d\n",buf);
          *p = buf;
        }
        else
            printf("ERROR! The SEG preset data can only be integers ranging from 0 to 99\nPlease retry\n");

    }
}
```

图 14.main 函数代码

最后，使用 EDA 工具将硬件代码综合并下载到 FPGA 开发板上，将板载的 CMSIS-DAP 调试器连接到 PC 机上，通过 Keil 运行程序进行测试。图 15 中的橘黄色方框中的四个按键从左到右分别对应数码管的四种显示模式：暂停、向上计数、向下计数和预置数模式。按一下某个按键便可触发对应的中断。

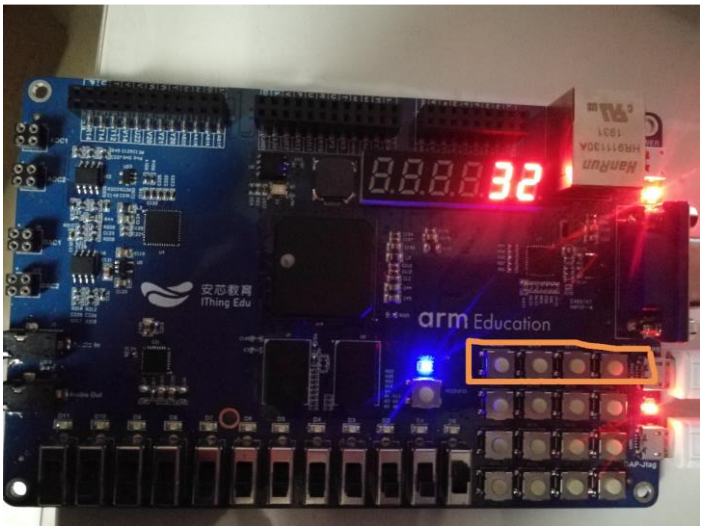

图 15.上板示意图

打开 WSL，运行 minicom 测试 UART 功能，结果如图 16 所示。在 PC 机通过键盘输入数据后，若输入的数据无效则会提示 Error，若为有效数据则会通过 UART 传给 SoC 设置数码管预置数模式的置数值，按下 FPGA 开发板上对应置数模式的按键便可触发中断，数码管会显示此前通过 PC 机键盘输入的数值。

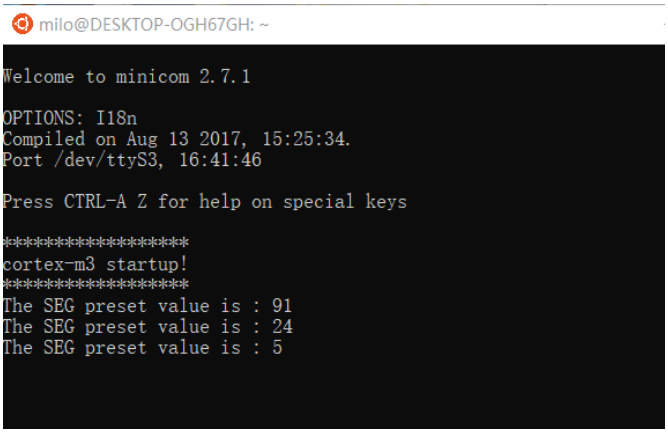

图 16 .minicom 测试结果