语音识别电子系统综合设计

一.实验目的

通过对在 wujian100 SoC 上集成的语音识别模块的硬件测试，了解并掌握 KWS 的技术原理和集成实现方案。

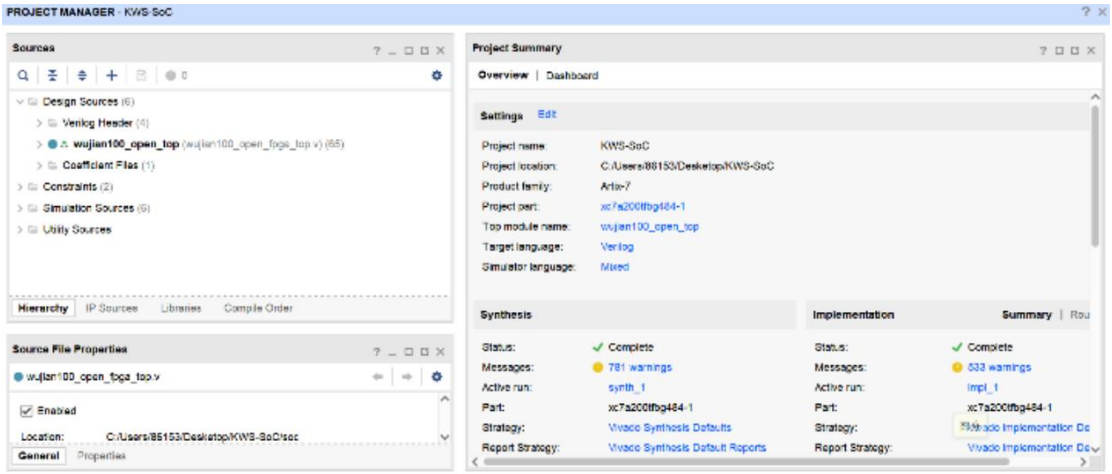
二.实验步骤（包括实验结果，数据记录，截图等）

1.基于示例 vivado 项目，编写 CDK 项目对 KWS IP 核测试

根据实验提供的集成示例，可以综合得到 bit 文件，并将其下载到板子上。为了测试 KWS IP 核在实际硬件上的功能，需要编写嵌入式控制程序（C）和 PC 端辅助脚本（python）。

嵌入式控制程序在 CDK 软件中完成编译等步骤后生成指令流和数据流，通过 CK-LINK 端口将指令和数据烧录到嵌入式处理器的指令存储器和数据存储器，然后在 CDK 软件中进行调试（Debug）。

基于示例的 vivado 项目，综合挂载好 KWS IP 核的 Soc 系统，可以得到 bit 文件。将 bit 文件烧录到开发板中，结果如下：



嵌入式控制程序需要对具体硬件的工作流程进行控制并调度相关的数据流，需要完成的步骤主要如下：

A）初始化 UART 串口，配置好波特率和工作模式如下图所示；

```
void initialize_usart()
{
  // 串口初始化函数，波特率设置为115200

        usart = csi_usart_initialize(0, NULL);
        csi_usart_config(usart, 115200, USART_MODE_ASYNCHRONOUS, USART_PARITY_NONE, USART_STOP_BITS_1, USART_DATA_BITS_8);

}
```

B)　初始化 KWS 模块，向其中写入 cos 函数表值和量化 scale 值、weight 数据、静音帧如下图所示;

```
// 1-从BRAM读取数据后，向写入cos函数表值和量化scale值(1,2,3顺序不能更改)
for(int i = 0; i < 746; i++)
{
        *(volatile float *) KWS_DATA_IN_BADDR = *(volatile float *) (WEIGHT_BADDR + 4 * i);
}
// 2-从BRAM读取数据后，向写入weight数据(1,2，3顺序不能更改)
for(int i = 0; i < 8970; i++)
{
        *(volatile unsigned int *) KWS_WEIGHT_IN_BADDR = *(volatile unsigned int *) (WEIGHT_BADDR + 2984 + 4 * i);
}
// 3-写入初始化静音帧
for(int i = 0; i < 3200; i++)
{
        *(volatile float *) KWS_DATA_IN_BADDR = 0;
}
```

C）等待 UART 串口的音频数据（PC 端发送），读取音频数据并将其写入到 KWS 模块的 data_in 端口；

D)KWS 模块处理结束后，将其 data_out 端口的 12 个输出数据从 uart 串口发送到 PC 端，代码如下图所示。

```
// 1-接收串口数据并进行拼接，将接收的byte数据拼接为float数据并写入指定位置
usart_receive_sync(usart, data_in, 8000 * 4);
for(int i = 0; i < 8000; i++)
{
        datain_tmp = data_in[4 * i + 3] + (data_in[4 * i + 2] << 8) + (data_in[4 * i + 1] << 16) + (data_in[4 * i + 0] << 24);
        *(volatile float *) KWS_DATA_IN_BADDR = *(float *)&datain_tmp;
}
// 2-从KWS数据输出端读取结果，并向串口发送结果
for(int i = 0; i < 12; i++)
{
        read_results_kws[i] = *(volatile float *) KWS_DATA_OUT_BADDR;
}
for(int i = 0; i < 12; i++)
{
        printf("%.5f\n", read_results_kws[i]);
}
// 3-单批数据处理并输出结果后，Control端再置1，准备输入下组数据
*(volatile uint32_t *) KWS_CONTROL_IN_BADDR = 0x1;
```

D）一批音频数据处理结束，回到步骤 c

本次实验额外提供一批浮点表示的音频数据（8000 个点），存储在 sample_left.txt 中。在嵌入式控制程序中通过一个 float 数组实例化这批音频数据，等同于从串口读取了一批音频数据，使用这个数组可以先忽略串口传输的控制，专注于程序控制流程的测试。确保程序控制流程没有问题后，再加上串口传输进行调试。

即把 sample_left.txt 中的 8000 个数据用一个 float 的数组存起来，把这个数组当成是从串口中读进来的音频数据送给 kws 的 data_in 端口，然后去读 kws 的 data_out 端口的 12 个 float 的标签置信度数据，把他们打印出来测试 kws 是否正常工作。输出结果如下：

```
Serial Pane
Initialize done
0 is -15.252008
1 is -1.449658
2 is -2.187633
3 is -4.412663
4 is -3.109421
5 is -6.076960
6 is 4.537775
7 is -1.371917
8 is -5.756475
9 is -2.311838
10 is -6.275959
11 is -7.267103
```

PC 端辅助脚本采用 python 编写，脚本解析测试用的音频文件(.wav)得到对应格式的音频数据，将其通过 UART 串口发送到板子；还需要收集板子通过 UART 串口发送回来的结果数据，并统计得到预测结果和预测准确率。

A）读取测试用的音频文件(.wav)，解析得到 16 进制表示的浮点格式的音频数据,代码如下图所示；

```
#------------------------step 2------------------------
# 调用wavread函数解析wav文件，得到float类型数组
# 调用transform函数进行格式转换，得到2进制数据
wav_path_0 = ["down", "go", "left", "no", "off", "on", "right", "stop", "up", "yes"]
wav_path_1 = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
wav_path_2 = ["1", "2", "3", "4", "5"]
wav_num = len(wav_path_0) * len(wav_path_2)
err_num = 0
for i in range(wav_num):
    path = ".%saudio%s%s%s%s%s.wav" % (os.path.sep, os.path.sep, wav_path_0[i // 5], os.path.sep,wav_path_1[i // 5], wav_path_2[i % 5])
    data_in = transform(wavread(path))
```

B）配置好 UART 串口；

```
#------------------------step 3------------------------
# 将译码得到的2进制数据写入UART串口
    result_buffer = []
    for j in range(0, 20):
        ser.write(data_in[j])
        time.sleep(0.015)
```

C）向 UART 串口发送解析后的音频数据，同时并发另一个线程收集来自 UART 串口的结果数据，即向串口发送数据和从串口接收数据是同步工作的。

D）收集得到结果数据后，将其与标签数据进行比对，统计预测结果，进一步可根据参考结果计算预测准确率，如下图所示。

```
#------------------------step 4------------------------
# 等待结果缓冲区(result_buffer)的数据,确保发送过去的音频数据被处理完，结果也已经发送回来
# 根据result_buffer的数据和标签数据得到预测结果，判断预测是否正确
    time.sleep(1)
    with open('./result.txt', "w", newline="") as f:
        for j in range(len(result_buffer) - 1):
            f.write(str(result_buffer[j]))

    count = 0
    result = []
    with open('./result.txt', "r") as f:
        line = f.readline()
        while line:
            count = count + 1
            if(count > 12):
                break
            result.append(float(line.strip('\n')))
            line = f.readline()
    os.remove('./result.txt')
    max = 0
    for j in range(len(result)):
        if(result[j] > result[max]):
            max = j
    print("wav num: %d" % (i))
    print("actual keyword: %s" % (wav_path_0[i // 5]))
    print("confidence list: {}".format(result))
    print("predicted keyword: %s\n" % (word_list[max]))
    if(word_list[max] != wav_path_0[i // 5]):
        err_num = err_num + 1
print("Summary:")
print("total wav num: %d" % (wav_num))
print("total err num: %d" % (err_num))
print("total accuracy: %f" % ((wav_num - err_num) / wav_num))
```

2. 实验数据记录

将 main.c 程序还原为使用串口的版本，接着在 CDK 软件中开启 Debug 模式并执行，令嵌入式程序等待 UART 串口的输入；然后执行 PC 端辅助脚本，向串口发送测试数据并收集结果数据。可以得到 KWS 的运算结果。将结果截图汇总，填表如下：

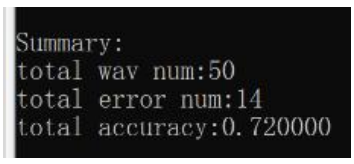
Out\wav num	1	2	3	4	5	6	7	8	9	10
Actual keyword	down	down	down	down	down	go	go	go	go	go
Predicted keyword	no	down	down	down	down	go	go	go	go	go
KWS output confidence list	0.116634，-1.35674 ，-1.236722	-0.790883，-0.034691，-0.556807	-0.868305，0.576562，-1.760847	2.405936，-2.460029，-2.504288	0.53089 ，-0.392112，2.725802	-0.731318，0.173118，1.191306	-0.425867，-0.389982，-1.269473	-0.867801,2.225586,-2.791846	1.915531,-0.835299,-1.401928	-0.792063,-0.916205,4.575005

Out\wav num	11	12	13	14	15	16	17	18	19	20
Actual keyword	left	left	left	left	left	no	no	no	no	no
Predicted keyword	left	left	left	left	left	off	yes	no	no	go
KWS output confidence list	0.951288,2.300205,0.743734	-0.977591,4.716313,-1.100846	4.537775,-1.371917,-1.449658	-0.738593,-1.670111,-1.297732	-1.945448,0.227249,-2.264921	0.29021,-0.960519,-0.504841	2.505998,-0.1505,-0.305609	-0.011438,-0.978777,-1.177637	3.117849,1.164836,-1.226394	0.743213,0.747575,-1.81051
Out\wav num	21	22	23	24	25	26	27	28	29	30
Actual keyword	off	off	off	off	off	on	on	on	on	on
Predicted keywo	off	off	off	off	off	off	on	on	on	on

rd										
KWS output confidence list	3.101857, 0.406952, -1.032353	1.046912, 1.8552, 3.810694	4.936536, 2.207171, -0.267572	-0.039992, 1.665653, -0.375255	2.101816, 0.440906, 6.158479	0.642405, 1.939615, 2.276847	2.598743, 2.303484, -0.40721	5.093901, 1.667503, -0.709846	6.979314, 0.414611, -0.893024	0.28968, 7.131486, -0.301552

Out\wav num	31	32	33	34	35	36	37	38	39	40
Actual keyword	right	right	right	right	right	stop	stop	stop	stop	stop
Predicted keyword	left	right	right	right	right	off	off	stop	stop	stop
KWS output confidence list	0.284648, 0.964737, 0.161669	0.549647, 0.587454, -0.610683	0.362304, 1.159964, -0.679357	1.677891, 2.659493, -0.786511	1.093869, -0.982526, 5.204529	1.654826, 2.792418, 1.405355	1.968368, 1.150927, 0.957622	1.601858, -1.34742, -1.002707	0.393424, -1.605337, -2.324028	-1.635771, -1.016279, 2.303885

Out\wav num	41	42	43	44	45	46	47	48	49	50
Actual keyword	up	up	up	up	up	yes	yes	yes	yes	yes
Predicted keyword	off	up	off	up	up	yes	yes	yes	yes	yes
KWS output confidence list	1.696416, 2.8967, 0.774855	2.834406, -1.310528, -1.844541	1.694917, 1.815174, -1.985948	3.163012, -0.565633, -1.010721	1.806543, -0.500679, -2.283785	0.615658, 3.510835, 1.335709	7.343542, 1.901127, -1.803828	7.751371, 1.095211, -1.379276	7.690877, 0.244878, -1.917129	7.816919, 1.007062, -1.456844



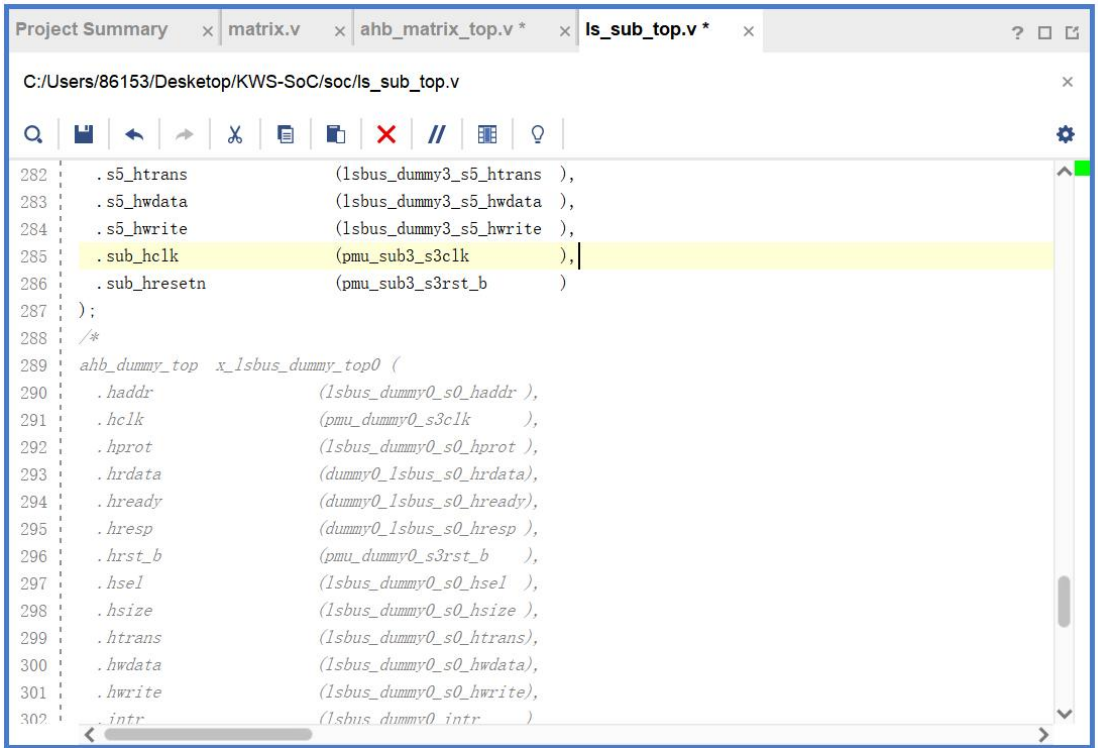
3.重新挂载 KWS IP，修改 CDK 项目再次对 KWS IP 核测试

在实验提供的示例项目中，KWS IP 是集成在 BUS Matrix(HCLK)上的 Dummy0/1/2/3 上的。在这一部分实验内容中，将 KWS IP 核重新挂载到 AHB LS BUS 的 Dummy0/1/2/3 上。

修改 RTL 完成 KWS IP 的重新挂载后，在 vivado 中重新综合、实现生成位流文件，用于之后的硬件测试。重新挂载后的 KWS 模块的各个输入输出端口地址也有变化，需要适当修改 CDK 项目内的嵌入式控制程序。实验截图如下所示：

```
);
/*
ahbm_dummy_top  x_main_mdummy_top0 (
  .hclk                   (pmu_mdummy0_hclk       ),
  .hrst_b                 (pmu_mdummy0_hrst_b     ),
  .mhaddr                 (mdummy0_hmain0_m4_haddr ),
  .mhburst                (mdummy0_hmain0_m4_hburst),
  .mhgrant                (hmain0_mdummy0_m4_hgrant),
  .mhprot                 (mdummy0_hmain0_m4_hprot ),
  .mhrdata                (hmain0_mdummy0_m4_hrdata),
  .mhready                (hmain0_mdummy0_m4_hready),
  .mhresp                 (hmain0_mdummy0_m4_hresp ),
  .mhsize                 (mdummy0_hmain0_m4_hsize ),
  .mhtrans                (mdummy0_hmain0_m4_htrans),
  .mhwdata                (mdummy0_hmain0_m4_hwdata),
  .mhwrite                (mdummy0_hmain0_m4_hwrite)
);
ahbm_dummy_top  x_main_mdummy_top1 (
  .hclk                   (pmu_mdummy1_hclk       ),
  .hrst_b                 (pmu_mdummy1_hrst_b     ),
  .mhaddr                 (mdummy1_hmain0_m5_haddr ),
  .mhburst                (mdummy1_hmain0_m5_hburst),
  .mhgrant                (hmain0_mdummy1_m5_hgrant),
  .mhprot                 (mdummy1_hmain0_m5_hprot ),
  .mhrdata                (hmain0_mdummy1_m5_hrdata),
  .mhready                (hmain0_mdummy1_m5_hready),
```

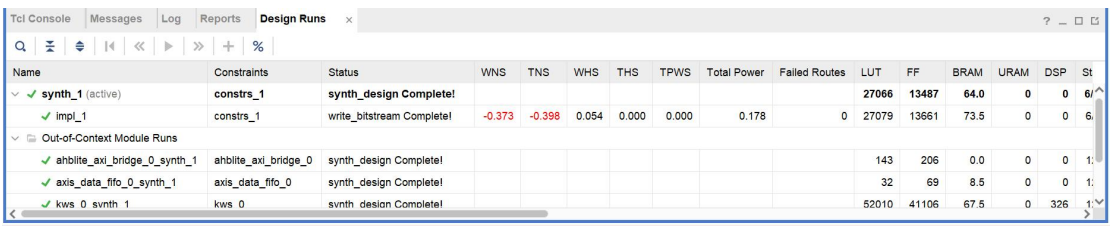
```
ahb_dummy_top  x_main_dmemdummy_top0 (
  .haddr                  (hmain0_dmemdummy0_s7_haddr ),
  .hclk                   (pmu_dmemdummy0_hclk       ),
  .hprot                  (hmain0_dmemdummy0_s7_hprot ),
  .hrdata                 (dmemdummy0_hmain0_s7_hrdata),
  .hready                 (dmemdummy0_hmain0_s7_hready),
  .hresp                  (dmemdummy0_hmain0_s7_hresp ),
  .hrst_b                 (pmu_dmemdummy0_hrst_b      ),
  .hsel                   (hmain0_dmemdummy0_s7_hsel  ),
  .hsize                  (hmain0_dmemdummy0_s7_hsize ),
  .htrans                 (hmain0_dmemdummy0_s7_htrans),
  .hwdata                 (hmain0_dmemdummy0_s7_hwdata),
  .hwrite                 (hmain0_dmemdummy0_s7_hwrite),
  .intr                   (main_dmemdummy0_intr       )
);
ahb_dummy_top  x_main_dmemdummy_top1 (
  .haddr                  (hmain0_dmemdummy0_s8_haddr ),
  .hclk                   (pmu_dmemdummy0_hclk       ),
  .hprot                  (hmain0_dmemdummy0_s8_hprot ),
  .hrdata                 (dmemdummy0_hmain0_s8_hrdata),
  .hready                 (dmemdummy0_hmain0_s8_hready),
  .hresp                  (dmemdummy0_hmain0_s8_hresp ),
  .hrst_b                 (pmu_dmemdummy0_hrst_b      ),
  .hsel                   (hmain0_dmemdummy0_s8_hsel  ),
  .hsize                  (hmain0_dmemdummy0_s8_hsize ),
  .htrans                 (hmain0_dmemdummy0_s8_htrans),
  .hwdata                 (hmain0_dmemdummy0_s8_hwdata),
  .hwrite                 (hmain0_dmemdummy0_s8_hwrite),
  .intr                   (main_dmemdummy0_intr       )
);
```




```
);
*/
ahb_axi_kws  x_ahb_axi_kws (
  .haddr0                 (hmain0_dummy0_s0_haddr ),
  .hclk0                  (pmu_dummy0_hclk        ),
  .hprot0                 (hmain0_dummy0_s0_hprot ),
  .hrdata0                (dummy0_hmain0_s0_hrdata),
  .hready0                (dummy0_hmain0_s0_hready),
  .hresp0                 (dummy0_hmain0_s0_hresp ),
  .hrst_b0                (pmu_dummy0_hrst_b      ),
  .hsel0                  (hmain0_dummy0_s0_hsel  ),
  .hsize0                 (hmain0_dummy0_s0_hsize ),
  .htrans0                (hmain0_dummy0_s0_htrans),
  .hwdata0                (hmain0_dummy0_s0_hwdata),
  .hwrite0                (hmain0_dummy0_s0_hwrite),
  .intr0                  (main_dummy0_intr       ),

  .haddr1                 (hmain0_dummy1_s1_haddr ),
  .hclk1                  (pmu_dummy1_hclk        ),
  .hprot1                 (hmain0_dummy1_s1_hprot ),
  .hrdata1                (dummy1_hmain0_s1_hrdata),
  .hready1                (dummy1_hmain0_s1_hready),
  .hresp1                 (dummy1_hmain0_s1_hresp ),
  .hrst_b1                (pmu_dummy1_hrst_b      ),
  .hsel1                  (hmain0_dummy1_s1_hsel  ),
  .hsize1                 (hmain0_dummy1_s1_hsize ),
  .htrans1                (hmain0_dummy1_s1_htrans),
  .hwdata1                (hmain0_dummy1_s1_hwdata),
  .hwrite1                (hmain0_dummy1_s1_hwrite),
  .intr1                  (main_dummy1_intr       ),

  .haddr2                 (hmain0_dummy2_s4_haddr ),
  .hclk2                  (pmu_dummy2_hclk        ),
  .hprot2                 (hmain0_dummy2_s4_hprot ),
  .hrdata2                (dummy2_hmain0_s4_hrdata),
```



三. 实验分析和总结

1.对于 SoC 系统流程的分析

将 KWS IP 核挂载到相应串口后，通过 PC 端数据收发脚本，同时在 CPU 程序控制下，才能完成语音识别的过程。

通过编写 C 程序来负责嵌入式中的数据流。C 代码编译完毕后以指令的形式存储到 Soc 中。CPU 负责执行相关代码控制硬件的工作。如向 KWS 发送控制信号，将初始化数据写入 KWS 和等待 KWS 输出结果并通过串口发送至 PC 端等。

通过 python 脚本对音频数据进行加包处理并对串口收到的数据进行解包处理，同时得到预测结果和准确率。

2.对于嵌入式 C 代码的分析

C 代码的编写包括 KWS 初始化和数据流的控制。

在 KWS 初始化过程中，读绝对地址的方法为：先将绝对地址的数值强制转换为 float 指针或 uint32 指针，再对强制转换后的指针值解引用，即可得到绝对地址内存储的数据。把它作为赋值语句的右值。此时寻址的变量与原本的指针略有不同，其需要一次增加 4。原因是 float 以及 uint32 都以 4 字节存储，而传统指针加 1，会根据其数据类型的大小，使地址增加相应的值。

整个 KWS 的初始化过程为先装载 KWS 运行所需的常数，从 BRAM 相应地址中读数并赋值给 KWS 的 data_in 端口。后写入初始化镜音帧，即向 data_in 端口写入足够数量的 0。

在数据流的控制中，需要确定 PC 端的存储方式是否为小端模式。同时，需要对 union 转换及定义联合体 Converter 进行讨论。首先，对比 C 语言中的强制转换，union 转换只在语法层面改变数据形式，维持数据的存储形式不变。而普通的转换会转换数据的存储形式，在精度上维持数据不变。然后联合体 Converter 包含两个成员 data_float 以及 data_byte[4]。在声明联合体变量 cvt 之后，我们将串口接收的 data_in 数据按照字节赋值给 uint8 数组 cvt.data_byte。此时 cvt.data_float 会自动转换成已经拼接好的 float 型数据。再用 cvt.data_float 为 KWS 的 data_in 端口赋值，可以实现单次的数据传输。最后利用 for 循环完成全部传输。传输完成后，KWS 输出数据的有效信号是对软件不可见的，因此我们直接在 KWS 的 data_in 赋值语句后编写结果导出的代码，等到 KWS 输出有效便将数据传出。

讨论 PC 端存储方式是否为小端模式。可以编写一个简单的程序进行验证。程序中要求打印地址。具体形式为从低地址到高地址逐字节取数并打印。结果显示低位在前，高位在后。小地址对应数据低位，说明 PC 端为小端存储，与 wujian 平台一致，不必对串口接收数据的顺序做出调整。

四. 实验收获、存在问题、改进措施或建议等

本次实验通过编写 C 程序代码已经 python 脚本，实现并验证了 KWS IP 核在实际硬件上的功能并大致了解了 Soc 系统的流程。