【C906-SOC最小系统搭建】(2)仿真

> 例程：使用uart重定向printf函数，打印"Hello World!"文本，参考文章。

0. 环境需求

本工程使用**VCS**和**Verdi**作为verilog仿真工具，仿真目录位于./smart_run，结构如下：

```tcl
├── export.sh
├── filelists
│   ├── C906_asic_rtl.fl
│   ├── sim.fl
│   ├── soc.fl
│   └── tdt_dmi_top_rtl.fl
├── Makefile
├── setup
│   ├── env_check.mk
│   └── smart_cfg.mk
├── tb
│   ├── tb.sv
│   └── uart_mnt.v
├── tests
│   ├── bin
│   ├── cases
│   ├── lib
│   └── regress
└── work
```

1. Testbench

相关文件位于./smart_run/tb目录下，tb.sv基于openc906的testbench进行了部分修改，记录下关键修改部分。

1）宏定义

时钟周期为10ns，对应频率为100MHz：

```
`define NOISA

`timescale 1ns/100ps
`define CLK_PERIOD          10

```

```
`define TCLK_PERIOD          40
`define MAX_RUN_TIME         700000000

`define SOC_TOP              tb.x_soc
`define RTL_MEM              tb.x_soc.x_axi_slave_warp.x_axi_slave128.x_f_spsram_16384x128



`define CPU_TOP              tb.x_soc.x_c906_wrap.x_cpu_top
`define tb_retire0           `CPU_TOP.core0_pad_retire
`define retire0_pc           `CPU_TOP.core0_pad_retire_pc[39:0]
`define CPU_CLK              `CPU_TOP.pll_core_cpuclk
`define CPU_RST              `CPU_TOP.pad_cpu_rst_b
```

2）指令Memory初始化

inst.pat和data.pat由编译过程生成，分别读到mem_inst_temp和mem_data_temp，再赋值给挂载的SRAM，注意是小端对齐。

```
integer i;
  bit [31:0] mem_inst_temp [65536];
  bit [31:0] mem_data_temp [65536];
  bit [127:0] temp128;
  integer j;
  initial
  begin
    $display("\t********* Init Program *********");
    $display("\t********* Wipe memory to 0 *********");
    for(i=0; i < 32'h16384; i=i+1)
    begin
      `RTL_MEM.mem[i] = 128'h0;
    end

    $display("\t********* Read program *********");
    $readmemh("inst.pat", mem_inst_temp);
    $readmemh("data.pat", mem_data_temp);

    $display("\t********* Load program to memory *********");
    for (i=0; i<('h10000/16); i=i+1) begin
      temp128 = {mem_inst_temp[i*4+0], mem_inst_temp[i*4+1], mem_inst_temp[i*4+2], mem_inst_temp[i*4+3]};
      for (j=0; j<16; j=j+1)
        `RTL_MEM.mem[i][j*8 +: 8] = temp128[(15-j)*8 +: 8];
    end

    for (i=0; i<('h30000/16); i=i+1) begin
      temp128 = {mem_data_temp[i*4+0], mem_data_temp[i*4+1], mem_data_temp[i*4+2], mem_data_temp[i*4+3]};
      for (j=0; j<16; j=j+1)
        `RTL_MEM.mem[i+('h10000/16)][j*8 +: 8] = temp128[(15-j)*8 +: 8];
    end
    i=0;
  end
```

3) UART Monitor

在testbench中例化了一个uart monitor模块用于打印字符串，uart_mnt.v文件基于 opene906/smart_run/logical/tb/uart_mnt.v at main · XUANTIE-RV/opene906，波特率BAUD设置为19200

```
`define sout          tb.uart0_sout
`define sin           tb.uart0_sin
`define clk           tb.clk
`define rst_b         tb.rst_b

module uart_mnt();

//#############################################
//            Create baud_clk
//#############################################
parameter FCPU = 1000000000/`CLK_PERIOD;
parameter BAUD = 19200;
parameter CYCLE_CUNT=FCPU/(BAUD*16) - 1;
```

4) Filelist

./smart_run/filelists目录下存放了soc和tb的filelist文件

```
├── C906_asic_rtl.fl
├── sim.fl
├── soc.fl
└── tdt_dmi_top_rtl.fl
```

C906_asic_rtl.fl设置了C906核相关代码路径；

tdt_dmi_top_rtl.fl设置了tdt_dmi调试模块相关代码路径;

soc.fl设置了整个SOC相关代码路径;

sim.fl设置了整个testbench相关代码路径;

2. C代码

header文件包括: **config.h**, **datatype.h**和**uart.h**; source文件包括**uart.c**, **printf.c**和**hello.c**。

config.h内定义了CPU和APB频率，均设置为了100MHz。

uart.h定义了波特率BUAD，设置为了19200。

其中printf函数最终要调用fputc函数输出字符，所以只需要用ck_uart_putc函数重定向fputc。

主函数位于hello.c，代码如下:

```c
#include "datatype.h"
#include "stdio.h"
#include "uart.h"

t_ck_uart_device uart0 = {0xFFFF};

int fputc(int ch, FILE *stream)
{
  ck_uart_putc(&uart0, (char)ch);
}

int main (void)
{
  //----------------------------------------------------------
  // setup uart
  //----------------------------------------------------------
  t_ck_uart_cfig   uart_cfig;

  uart_cfig.baudrate = BAUD;       // any integer value is allowed
  uart_cfig.parity = PARITY_NONE;     // PARITY_NONE / PARITY_ODD / PARITY_EVEN
  uart_cfig.stopbit = STOPBIT_1;      // STOPBIT_1 / STOPBIT_2
  uart_cfig.wordsize = WORDSIZE_8;    // from WORDSIZE_5 to WORDSIZE_8
  uart_cfig.txmode = ENABLE;          // ENABLE or DISABLE
  // open UART device with id = 0 (UART0)
  ck_uart_open(&uart0, 0);
  // initialize uart using uart_cfig structure

  ck_uart_init(&uart0, &uart_cfig);

  for (int i=0; i<4; i++) {
    printf("Hello World!\n");
  }
  ck_uart_close(&uart0);
  return 0;
}
```

linker.lcf链接脚本修改，指令和数据都放在SRAM上，MEM1为指令空间(inst.pat)，MEM2为数据空间(data.pat)

```
MEMORY
{
MEM1(RWX)  : ORIGIN = 0x10000000,  LENGTH = 0x10000
MEM2(RWX)  : ORIGIN = 0x10010000,  LENGTH = 0x30000
}
__kernel_stack = 0x10030000 ;

ENTRY(__start)

SECTIONS {
    .text :
    {
        crt0.o (.text)
        *(.text*)
    } >MEM1

    .rodata :  {   *( .rodata* )
                   *(.srodata)
                   *(.srodata.*)
                   *(.srodata.cst4*)
                   *(.srodata.cst8*)
                  } >MEM1

    .data :
    {
        *(.data*)
        *(.sdata*)
    } >MEM2
    .bss :
    {
        *(.bss) *.(COMMON) *(.sbss)
    } >MEM2


    end = .;
}
```

3. 编译

主Makefile位于./smart_run下，能够完成不同例程的C程序和testbench编译，Makefile基于openc906的Makefile进行了修改，只支持使用vcs和verdi仿真。

Makefile导入了./setup/smart_cfg.mk文件，在smart_cfg添加用户自定义的例程，本工程添加了hello例程。

```
CASE_LIST := \
      ISA_THEAD \
      ISA_INT \
      ISA_LS \
      ISA_FP \
      coremark \
      MMU \
      interrupt \
      exception \
      debug \
      csr \
      cache \
      hello \

hello_build:
        @cp ./tests/cases/hello/* ./work
        @find ./tests/lib/ -maxdepth 1 -type f -exec cp {} ./work/ \;
        @cp ./tests/lib/clib/* ./work
        @cp ./tests/lib/newlib_wrap/* ./work
        @cd ./work && make -s clean && make -s all CPU_ARCH_FLAG_0=c906fd  ENDIAN_MODE=little-endian CASENAME=hello FILE=hello >& hello_build.case.log
```

4. 运行

0）cd ./smart_run

1）source export.sh, export.sh脚本设置了RTL代码路径和RISCV编译器路径, **需要修改**:

```
#!/bin/bash

export CODE_BASE_PATH=$(readlink -f ../rtl)
```

```
echo "Root of code base has been specified as:"
echo "    $CODE_BASE_PATH"

export TOOL_EXTENSION=$(realpath /data/Xuantie-900-gcc-elf-newlib-x86_64-V3.0.1/bin)
echo 'Toolchain path($TOOL_EXTENSION):'
echo "    $TOOL_EXTENSION"
```

2）chmod +x ./tests/bin/Srec2vmem, 编译过程中会执行Srec2vmem，需要给Srec2vmem赋运行权限。

3）make runcase CASE=hello, 部分log如下：

```
../simv up to date
CPU time: 13.008 seconds to compile + 1.000 seconds to elab + .412 seconds to link
Verdi KDB elaboration done and the database successfully generated: 0 error(s), 0 warning(s)
  Toolchain path: /data/Xuantie-900-gcc-elf-newlib-x86_64-V3.0.1/bin
cd ./work && ./simv -l run.vcs.log
Notice: timing checks disabled with +notimingcheck at compile-time
Chronologic VCS simulator copyright 1991-2018
Contains Synopsys proprietary information.
Compiler version O-2018.09-SP2_Full64; Runtime version O-2018.09-SP2_Full64; Apr  7 15:54 2025
Addressing configuration for axi_interconnect instance tb.x_soc.x_axi_interconnect_wrap.x_axi_interconnect
 0 ( 0): 0010000000 / 28 -- 0010000000-001fffffff
 1 ( 0): 0020000000 / 28 -- 0020000000-002fffffff
        ********* Init Program *********
        ********* Wipe memory to 0 *********
        ********* Read program *********
        ********* Load program to memory *********
######time:                   0, Dump start######
*Verdi* Loading libsscore_vcs201809.so
FSDB Dumper for VCS, Release Verdi_O-2018.09-SP2, Linux x86_64/64bit, 02/21/2019
(C) 1996 - 2019 by Synopsys, Inc.
*Verdi* : Create FSDB file 'novas.fsdb'
*Verdi* : Begin traversing the scopes, layer (0).
*Verdi* : End of traversing.
Hello World!
Hello World!
Hello World!
Hello World

```

```
**********************************************
*    simulation finished successfully        *
**********************************************
$finish called from file "../tb/tb.sv", line 195.
$finish at simulation time         261334750000
           V C S   S i m u l a t i o n   R e p o r t
Time: 26133475000000 fs
CPU Time:     66.590 seconds;       Data structure size:   2.7Mb
Mon Apr  7 15:55:21 2025
```

由于仿真早于uart_mnt完全接受uart信号，所以有一个"!"和换行符没有显示，暂时不管。