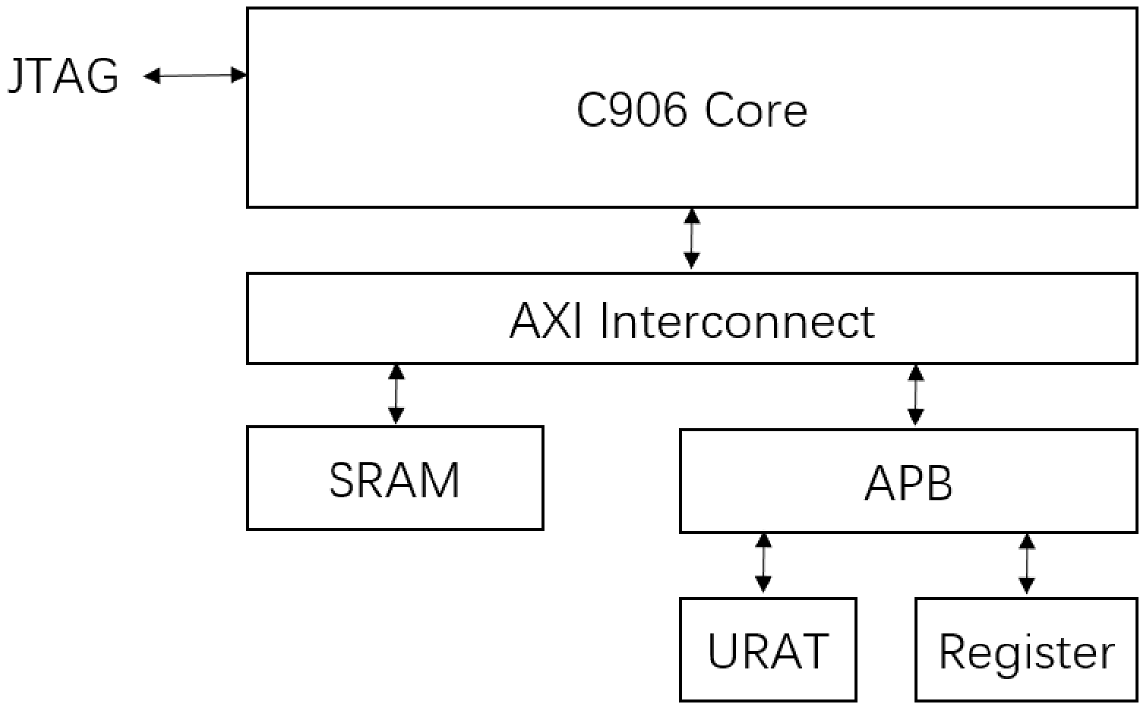
【C906-SOC最小系统搭建】(1)集成

本项目基于玄铁（XuanTie）开源RISC-V核C906，搭建了一个最小化的SoC系统，旨在深入理解SoC开发流程及其关键模块的设计与实现。

1. 总体架构



1.1 总线设计

- 采用AXI4作为高性能总线，负责与SRAM、APB的高速数据交互。
- 通过APB总线管理低速外设，优化系统资源分配与功耗效率。

1.2 存储与外设设计

- **存储模块**：通过AXI4总线挂载了一块256KB的SRAM，作为初始指令存储器（Memory），用于存储启动代码和关键数据。
- **低速外设**：通过APB总线扩展了低速外设接口，包括一个UART模块，用于调试信息的字符串打印。
- **寄存器模块**：在APB总线上挂载了寄存器组，为后续的读写回环测试提供硬件支持。

1.3 总线空间分配及内存属性

空间分配参考了4.2. BUS — 智显文档中心 v1.0 文档

地址空间定义				CPU SYSMAP 硬件定义 （机器模式内存属性）				
空间段	起始地址	结束地址	大小	变量	ADDR	SO	C	B
BROM（预留）	0x00000000	0x000FFFFF	256MB	SYSMAP_BASE_ADDRO	28'h0010000	0	1	1
SRAM	0x00100000	0x0FFFFFFF	256MB	SYSMAP_BASE_ADDR1	28'h0020000	0	1	1
DEVICE	0x10000000	0x1FFFFFFF	256MB	SYSMAP_BASE_ADDR2	28'h0030000	1	0	0
C906	0x20000000	0x2FFFFFFF	256MB	SYSMAP_BASE_ADDR3	28'h0040000	1	0	0
DRAM（预留）	0x30000000	0x3FFFFFFF	1GB	SYSMAP_BASE_ADDR4	28'h0080000	0	1	1
RESERVE0	0x40000000	0x7FFFFFFF	1GB	SYSMAP_BASE_ADDR5	28'h00C0000	0	1	1
RESERVE1	0x80000000	0xBFFFFFFF	1GB	SYSMAP_BASE_ADDR6	28'h0100000	0	1	1
RESERVE2	0xC0000000	0xFFFFFFFF	1020GB	SYSMAP_BASE_ADDR7	28'hFFFFFFF	0	1	1

还需要修改sysmap.h文件，具体细节参考《玄铁 C906 集成手册-第四章》

```
// ADDR is 28-bit, 4K address
// Flag includes: Strong Order, Cacheable, Bufferable, Shareable, Security

  `define SYSMAP_BASE_ADDR0  28'h0010000 // BROM-256MB
  `define SYSMAP_FLG0        5'b01100

  `define SYSMAP_BASE_ADDR1  28'h0020000 // SRAM-256MB
  `define SYSMAP_FLG1        5'b01100

  `define SYSMAP_BASE_ADDR2  28'h0030000 // DEVICE-256MB
  `define SYSMAP_FLG2        5'b10000

  `define SYSMAP_BASE_ADDR3  28'h0040000 // C906-256MB
  `define SYSMAP_FLG3        5'b10000

  `define SYSMAP_BASE_ADDR4  28'h0080000 // DRAM-1GB
  `define SYSMAP_FLG4        5'b01100

  `define SYSMAP_BASE_ADDR5  28'h00C0000 // RESERVE0-1GB
  `define SYSMAP_FLG5        5'b01100

  `define SYSMAP_BASE_ADDR6  28'h0100000 // RESERVE1-1GB
  `define SYSMAP_FLG6        5'b011000

  `define SYSMAP_BASE_ADDR7  28'hfffffff // RESERVE2-1020GB
  `define SYSMAP_FLG7        5'b01100

//End ct_mmu_sysmap
```

2. SOC集成

RTL文件夹结构如下：

```
tree -L 2 ./rtl/
./rtl/
├── c906_core
│   ├── biu
│   ├── clint
│   ├── clk
│   ├── common
│   ├── cp0
│   ├── cpu
│   ├── dtu
│   ├── filelists
│   ├── fpga
│   ├── idu
│   ├── ifu
│   ├── iu
│   ├── lsu
│   ├── mmu
│   ├── plic
│   ├── pmp
│   ├── pmu
│   ├── rst
│   ├── rtu
│   ├── sram
│   ├── tdt
│   ├── vdiv
│   ├── vdsp
│   ├── vfalu
│   ├── vfdsu
│   ├── vfmau
│   └── vidu
├── peripheral
│   ├── apb
│   └── axi
└── soc
    ├── addr_map.svh
    ├── axi2apb_wrap.sv
    ├── axi_bus.sv
    ├── axi_interconnect_wrap.sv
    ├── axi_slave128_warp.sv
    ├── c906_wrap.sv


    ├── per_clk_gen.v
    └── soc.sv
```

由于AXI信号较多，此工程用Interface对AXI BUS进行了封装（参考：pulpino/rtl/includes/axi_bus.sv at master · pulp-platform/pulpino），方便之后的集成：

```
interface AXI_BUS
#(
    parameter AXI_ADDR_WIDTH = 32,
    parameter AXI_DATA_WIDTH = 64,
    parameter AXI_ID_WIDTH   = 8
);

  localparam AXI_STRB_WIDTH = AXI_DATA_WIDTH/8;

  logic [AXI_ADDR_WIDTH-1:0] aw_addr;
  logic [2:0]                aw_prot;
  logic [7:0]                aw_len;
  logic [2:0]                aw_size;
  logic [1:0]                aw_burst;
  logic                      aw_lock;
  logic [3:0]                aw_cache;
  logic [AXI_ID_WIDTH-1:0]   aw_id;
  logic                      aw_ready;
  logic                      aw_valid;

  logic [AXI_ADDR_WIDTH-1:0] ar_addr;
  logic [2:0]                ar_prot;
  logic [7:0]                ar_len;
  logic [2:0]                ar_size;
  logic [1:0]                ar_burst;
  logic                      ar_lock;
  logic [3:0]                ar_cache;
  logic [AXI_ID_WIDTH-1:0]   ar_id;
  logic                      ar_ready;
  logic                      ar_valid;

  logic                      w_valid;
  logic [AXI_DATA_WIDTH-1:0] w_data;
  logic [AXI_STRB_WIDTH-1:0] w_strb;
  logic                      w_last;
  logic                      w_ready;

  logic [AXI_DATA_WIDTH-1:0] r_data;
  logic [1:0]                r_resp;

  logic                      r_last;
  logic [AXI_ID_WIDTH-1:0]   r_id;
  logic                      r_ready;
  logic                      r_valid;

  logic [1:0]                b_resp;
  logic [AXI_ID_WIDTH-1:0]   b_id;
  logic                      b_ready;
  logic                      b_valid;

  modport Master
  (
    output aw_valid, output aw_addr, output aw_prot,
           output aw_len, output aw_size, output aw_burst, output aw_lock,
           output aw_cache, output aw_id,
    input aw_ready,

    output ar_valid, output ar_addr, output ar_prot,
           output ar_len, output ar_size, output ar_burst, output ar_lock,
           output ar_cache, output ar_id,
    input ar_ready,

    output w_valid, output w_data, output w_strb, output w_last,
    input w_ready,

    input r_valid, input r_data, input r_resp, input r_last, input r_id,
    output r_ready,

    input b_valid, input b_resp, input b_id,
    output b_ready
  );

  modport Slave
  (
    input aw_valid, input aw_addr, input aw_prot,
          input aw_len, input aw_size, input aw_burst, input aw_lock,
          input aw_cache, input aw_id,
    output aw_ready,

    input ar_valid, input ar_addr, input ar_prot,
          input ar_len, input ar_size, input ar_burst, input ar_lock,
          input ar_cache, input ar_id,
    output ar_ready,

    input w_valid, input w_data, input w_strb, input w_last,
    output w_ready,

    output r_valid, output r_data, output r_resp, output r_last, output r_id,
    input r_ready,

    output b_valid, output b_resp, output b_id,
    input b_ready
  );

endinterface
```

2.1 C906 Wrapper

c906_wrap参考了openc906官方工程提供的soc例程，例化了一个openc906核和tdt_dmi_top调试模块。

通过`CPU_RVBA宏定义设置CPU指令启动地址，以下展示了c906_wrap的接口信号。

```
module c906_wrap(
    input               pll_core_cpuclk,
    input               axim_clk_en,
    input               pad_cpu_rst_b,
    input               pad_dtm_jtg_tclk,
    input               pad_dtm_jtg_tdi,
    input               pad_dtm_jtg_tms,
    input               pad_dtm_jtg_trst_b,
    input               pad_yy_scan_enable,
    input               pad_yy_scan_mode,
    input   [39:0]      xx_intc_int,
    output  [1:0]       core0_pad_lpmd_b,
    output              core0_pad_retire0,
    output  [39:0]      core0_pad_retire0_pc,
    output              dtm_pad_jtg_tdo,
    output              dtm_pad_jtg_tdo_en,

    AXI_BUS.Master      axi_mst
);
endmodule
```

2.2 AXI IP

本工程需要axi_interconnect、axi2sram和axi2apb ip。

2.2.1 AXI Interconnect

本工程使用的AXI Interconnect基于开源工程：[alexforencich/verilog-axi: Verilog AXI components for FPGA implementation]

需要verilog-axi/rtl at master · alexforencich/verilog-axi目录下的axi_interconnect.v、arbiter.v和priority_encoder.v三个文件，

在此基础上用AXI_BUS进行了封装，得到了axi_interconnect_wrap.sv文件，需要注意的是这个ip的rst是高电平有效，接口信号如下：

```
// SLAVES: 1 / MASTERS: 2
module axi_interconnect_wrap #(
    parameter DATA_WIDTH = 32,
    parameter ADDR_WIDTH = 32,
    parameter ID_WIDTH = 8,

    parameter M00_BASE_ADDR = 0,
    parameter M00_ADDR_WIDTH = 32'd24,

    parameter M01_BASE_ADDR = 0,
    parameter M01_ADDR_WIDTH = 32'd24
)
(
    input  wire                         clk,
    input  wire                         rst,
    AXI_BUS.Slave                       s00_axi,
    AXI_BUS.Master                      m00_axi,
    AXI_BUS.Master                      m01_axi
);
endmodule
```

2.2.2 AXI2SRAM

本工程使用的AXI2SRAM基于开源openc906中smart_run-axi_slave128例程：
openc906/smart_run/logical/axi/axi_slave128.v at main · XUANTIE-RV/openc906

使用AXI_BUS封装得到axi_slvae_warp.v文件，接口信号如下：

```
module axi_slave_warp #(
  parameter AXI_ADDR_WIDTH = 32,
  parameter AXI_DATA_WIDTH = 128,
  parameter AXI_ID_WIDTH   = 8
)(
  input           aclk,
  input           arst_n,
  AXI_BUS.Slave   axi_slave_if


);
endmodule
```

在axi_slave128的基础上修改了例化的sram大小，本工程集成了一块256KB的SRAM，shape为为16384x128，带写字节掩码功能，verilog代码如下：

```
module f_spsram_16384x128 (
  A,
  CEN,
  CLK,
  D,
  Q,
  WEN
);

input   [13:0]                          A;
input                                   CEN;
input                                   CLK;
input   [127:0]                         D;
input   [15:0]                          WEN;
output reg      [127:0] Q;

reg [127:0] mem [0:16384-1];
integer i;

always @(posedge CLK) begin
        for (i=0; i<16; i=i+1) begin
                if (~CEN && ~WEN[i])
                        mem[A][i*8 +: 8] <= D[i*8 +: 8];
        end
end

always @(posedge CLK) begin
        if (~CEN && (&WEN))
                Q <= mem[A];
end

endmodule
```

2.2.3 AXI2APB

本工程使用的AXI2APB基于开源工程：[adki/gen_amba_2021: AMBA bus generator including AXI4, AXI3, AHB, and APB]

使用gen_amba_2021生成了带1个axi slave，2个apb master的ip，并在此基础上使用AXI BUS进行封装，接口信号如下：

```verilog
module axi2apb_wrap
    #(parameter AXI_WIDTH_SID  = 4,    // ID width in bits
      parameter AXI_WIDTH_AD   = 32,   // address width
      parameter AXI_WIDTH_DA   = 32,   // data width
      parameter WIDTH_PAD      = 32,   // address width
      parameter WIDTH_PDA      = 32,   // data width
      parameter ADDR_PBASE0    = 32'hC0000000,
      parameter ADDR_PLENGTH0  = 16,
      parameter ADDR_PBASE1    = 32'hC0001000,
      parameter ADDR_PLENGTH1  = 16,
      parameter CLOCK_RATIO    = 2'b00 // 0=1:1, 3=async
     )
(
    input wire                    ARESETn,
    input wire                    ACLK,
    input wire                    PRESETn,
    input wire                    PCLK,
    AXI_BUS.Slave                 AXI_SLAVE,
    output wire [WIDTH_PAD-1:0] M_PADDR,
    output wire                   M_PENABLE,
    output wire                   M_PWRITE,
    output wire [WIDTH_PDA-1:0] M_PWDATA,
    output wire                   M0_PSEL,
    input wire  [WIDTH_PDA-1:0] M0_PRDATA,
    input wire                    M0_PREADY,
    input wire                    M0_PSLVERR,
    output wire                   M1_PSEL,
    input wire  [WIDTH_PDA-1:0] M1_PRDATA,
    input wire                    M1_PREADY,
    input wire                    M1_PSLVERR
);
endmodule
```

2.3 APB IP

2.3.1 APB2REG

本工程使用gpt生成了一个简单的apb2reg模块，带有16个32bit的寄存器，verilog代码如下：

```verilog
module apb_to_reg #(
    parameter ADDR_WIDTH = 6,
    parameter DATA_WIDTH = 32
) (
    input  wire                      pclk,
    input  wire                      presetn,
    input  wire [ADDR_WIDTH-1:0]     paddr,
    input  wire                      psel,
    input  wire                      penable,
    input  wire                      pwrite,
    input  wire [DATA_WIDTH-1:0]     pwdata,
    output reg  [DATA_WIDTH-1:0]     prdata,
    output reg                       pready,
    output reg                       pslverr
);

    localparam NUM_REGS = 16;
    reg [DATA_WIDTH-1:0] registers[0:NUM_REGS-1];

    localparam IDLE = 1'b0;
    localparam ACCESS = 1'b1;
    reg state;

    wire [3:0] reg_addr = paddr[5:2];

    always @(posedge pclk or negedge presetn) begin
        if (!presetn) begin
            state <= IDLE;
            pready <= 1'b0;
            pslverr <= 1'b0;
            prdata <= {DATA_WIDTH{1'b0}};
            for (integer j = 0; j < NUM_REGS; j = j + 1) begin
                registers[j] <= {DATA_WIDTH{1'b0}};
            end
        end else begin
            pslverr <= 1'b0;
            case (state)
                IDLE: begin
                    if (psel && !penable) begin
                        state <= ACCESS;
                    end
                end
                ACCESS: begin
                    pready <= 1'b1;
                    if (psel && penable) begin
                        if (pwrite) begin
                            registers[reg_addr] <= pwdata;
                        end
                        else begin
                            prdata <= registers[reg_addr];
                        end
                        state <= IDLE;
                    end
                end
                default: state <= IDLE;
            endcase
            if (!psel) begin
                pready <= 1'b0;
            end
        end
    end
endmodule
```

2.3.2 APB2URAT

本工程使用的AXI2SRAM基于开源openc906中smart_run-uart例程：[openc906/smart_run/logical/uart at main · XUANTIE-RV/openc906]()

接口信号如下：

```
module uart(
  apb_uart_paddr,
  apb_uart_penable,
  apb_uart_psel,
  apb_uart_pwdata,
  apb_uart_pwrite,
  rst_b,
  s_in,
  s_out,
  sys_clk,
  uart_apb_prdata,
  uart_vic_int
);
endmodule
```

2.3.3 APB SUB-SYSTEM

apb_sub_system模块用于例化apb相关ip，包括了apb2reg和apb2uart

```
module apb_sub_system (
    input                   pclk,
    input                   prst_n,
    input           [31:0]  m_paddr,

    input                 m_penable,
    input                 m_pwrite,
    input         [31:0]  m_pwdata,
    // REG
    input                 m0_psel,
    output        [31:0]  m0_prdata,
    output                m0_pready,
    output                m0_pslverr,
    // URAT0
    input                 m1_psel,
    output        [31:0]  m1_prdata,
    output                m1_pready,
    output                m1_pslverr,
    // URAT0
    input                 uart0_rx,
    output                urat0_tx,
    output                uart0_int
);
endmodule
```

2.4 集成

通过之前的封装，SOC顶层集成十分清晰，verilog代码如下：

```
module soc (
        input   i_pad_clk,
        input   i_pad_rst_b,
        input   i_pad_jtg_nrst_b,
        input   i_pad_jtg_tclk,
        input   i_pad_jtg_tdi,
        input   i_pad_jtg_tms,
        input   i_pad_jtg_trst_b,
        output  o_pad_jtg_tdo,
        input   i_pad_uart_rx,
        output  o_pad_uart_tx
);

wire pad_cpu_rst_b = i_pad_rst_b & i_pad_jtg_nrst_b;
wire aclk;
wire aclk_en;
wire arst_n = pad_cpu_rst_b;
per_clk_gen x_per_clk_gen (
        .ckl_i      (i_pad_clk),
        .per_clk    (aclk),
        .axi_clk_en (aclk_en)
);
```

```verilog
//----------------------------------------------------------------
//  SLAVE0: BROM
//  SLAVE1: SRAM
//  SLAVE2: APB
//  SLAVE3: DRAM
//----------------------------------------------------------------
localparam AXI_ADDR_WIDTH = 40;
localparam AXI_DATA_WIDTH = 128;
AXI_BUS #(
    .AXI_ADDR_WIDTH(AXI_ADDR_WIDTH),
    .AXI_DATA_WIDTH(AXI_DATA_WIDTH),
    .AXI_ID_WIDTH(8)
) axi_node_in[0:0]();
AXI_BUS #(
    .AXI_ADDR_WIDTH(AXI_ADDR_WIDTH),
    .AXI_DATA_WIDTH(AXI_DATA_WIDTH),
    .AXI_ID_WIDTH(8)
) axi_node_out[1:0]();

axi_interconnect_wrap #(
        .DATA_WIDTH     (AXI_DATA_WIDTH),
        .ADDR_WIDTH     (AXI_ADDR_WIDTH),
        .ID_WIDTH       (8),
        .M00_BASE_ADDR  (`SRAM_BASE_BASE),
        .M00_ADDR_WIDTH (`SRAM_ADDR_LEN),
        .M01_BASE_ADDR  (`APB_BASE_BASE),
        .M01_ADDR_WIDTH (`APB_ADDR_LEN)
) x_axi_interconnect_wrap (
        .clk     (aclk),
        .rst     (~arst_n),
        .s00_axi (axi_node_in[0]),
        .m00_axi (axi_node_out[0]),
        .m01_axi (axi_node_out[1])
);

//----------------------------------------------------------------
//  SRAM
//----------------------------------------------------------------
axi_slave_warp #(
        .AXI_ADDR_WIDTH (AXI_ADDR_WIDTH),
        .AXI_DATA_WIDTH (AXI_DATA_WIDTH),
        .AXI_ID_WIDTH   (8)
) x_axi_slave_warp (
        .aclk           (aclk),
        .arst_n         (arst_n),
        .axi_slave_if   (axi_node_out[0])
);

```

```verilog
//------------------------------------------------------------------
//  C906
//------------------------------------------------------------------
wire [39:0] xx_intc_int;
c906_wrap x_c906_wrap(
        .pll_core_cpuclk      (i_pad_clk),
        .axim_clk_en          (aclk_en),
        .pad_cpu_rst_b        (pad_cpu_rst_b),
        .pad_dtm_jtg_tclk     (i_pad_jtg_tclk),
        .pad_dtm_jtg_tdi      (i_pad_jtg_tdi),
        .pad_dtm_jtg_tms      (i_pad_jtg_tms),
        .pad_dtm_jtg_trst_b   (i_pad_jtg_trst_b),
        .pad_yy_scan_enable   (1'b0),
        .pad_yy_scan_mode     (1'b0),
        .xx_intc_int          (xx_intc_int),
        .core0_pad_lpmd_b     (),
        .core0_pad_retire0    (),
        .core0_pad_retire0_pc (),
        .dtm_pad_jtg_tdo      (o_pad_jtg_tdo),
        .dtm_pad_jtg_tdo_en   (),
        .axi_mst              (axi_node_in[0])
);

//------------------------------------------------------------------
//  AXI TO APB BRIDGE
//------------------------------------------------------------------
wire pclk   = aclk;
wire prst_n = arst_n;

wire    [31:0]  m_paddr;
wire            m_penable;
wire            m_pwrite;
wire    [31:0]  m_pwdata;
wire            m0_psel;
wire    [31:0]  m0_prdata;
wire            m0_pready;
wire            m0_pslverr;
wire            m1_psel;
wire    [31:0]  m1_prdata;
wire            m1_pready;
wire                    m1_pslverr;


axi2apb_wrap #(
        .AXI_WIDTH_SID (8),
        .AXI_WIDTH_AD  (AXI_ADDR_WIDTH),
        .AXI_WIDTH_DA  (AXI_DATA_WIDTH),
        .WIDTH_PAD     (32),
        .WIDTH_PDA     (32),
        .ADDR_PBASE0   (`NPU_BASE_ADDR),
        .ADDR_PLENGTH0 (`NPU_ADDR_LEN),
        .ADDR_PBASE1   (`UART0_BASE_ADDR),
        .ADDR_PLENGTH1 (`UART0_ADDR_LEN),
        .CLOCK_RATIO   (2'b00)
) x_axi2apb_wrap (
        .ARESETn       (arst_n),
        .ACLK          (aclk),
        .PRESETn       (prst_n),
        .PCLK          (pclk),
        .AXI_SLAVE     (axi_node_out[1]),
        .M_PADDR       (m_paddr),
        .M_PENABLE     (m_penable),
        .M_PWRITE      (m_pwrite),
        .M_PWDATA      (m_pwdata),
        .M0_PSEL       (m0_psel),
        .M0_PRDATA     (m0_prdata),
        .M0_PREADY     (m0_pready),
        .M0_PSLVERR    (m0_pslverr),
        .M1_PSEL       (m1_psel),
        .M1_PRDATA     (m1_prdata),
        .M1_PREADY     (m1_pready),
        .M1_PSLVERR    (m1_pslverr)
);

wire uart0_int;
apb_sub_system x_apb_sub_system(
        .pclk       (pclk),
        .prst_n     (prst_n),
        .m_paddr    (m_paddr),
        .m_penable  (m_penable),
        .m_pwrite   (m_pwrite),
        .m_pwdata   (m_pwdata),
        .m0_psel    (m0_psel),
        .m0_prdata  (m0_prdata),
        .m0_pready  (m0_pready),
        .m0_pslverr (m0_pslverr),
        .m1_psel    (m1_psel),
        .m1_prdata  (m1_prdata),
        .m1_pready  (m1_pready),
        .m1_pslverr (m1_pslverr),
        .uart0_rx   (i_pad_uart_rx),
        .urat0_tx   (o_pad_uart_tx),
        .uart0_int  (uart0_int)
);
assign xx_intc_int = {39'd0, uart0_int};

```

```
endmodule
```

其中的宏定义在addr_map.svh文件中定义：

```
//-----------------------------------------------------------------
//  AXI MEMORY MAP
//-----------------------------------------------------------------
`define BROM_ADDR_BASE  40'H00_0000_0000
`define BROM_ADDR_LEN   32'd28
`define SRAM_ADDR_BASE  40'H00_1000_0000
`define SRAM_ADDR_LEN   32'd28
`define APB_ADDR_BASE   40'H00_2000_0000
`define APB_ADDR_LEN    32'd28
`define C906_ADDR_BASE  40'H00_3000_0000
`define C906_ADDR_LEN   32'd28
`define DRAM_ADDR_BASE  40'H00_4000_0000
`define DRAM_ADDR_LEN   32'd30

//-----------------------------------------------------------------
//  APB SUB-SYSTEM MEMORY MAP
//-----------------------------------------------------------------
`define NPU_BASE_ADDR   40'H00_2000_0000
`define NPU_ADDR_LEN    12 // 4KB
`define UART0_BASE_ADDR 40'H00_2000_2000
`define UART0_ADDR_LEN  12 // 4KB


//-----------------------------------------------------------------
//  CPU RESET BOOT ADDR
//-----------------------------------------------------------------
`define CPU_RVBA `SRAM_ADDR_BASE
`define C906_PLIC `C906_ADDR_BASE
```

CPU_RVBA定义了CPU核复位后指令开始运行的地址，本工程设置为SRAM的BASE地址，即从SRAM启动程序。