PODES_M0O

Application User Manual

Ver1.0

Index:

1 概述

当准备一个深嵌入式 SoC 设计时，摆在项目主管面前的首要问题是：

使用什么指令集架构？是否有对应的 MCU IP 可供选择？

相关 MCU IP 的面积、性能及功耗是否满足特定项目的需求？

需要对 MCU IP 重新裁剪吗？

编译器及软件开发环境如何？需要团队重新学习吗？

这个 MCU IP 是否支持在线调试功能？

有无软核提供？如果是硬核，如何才可以做到 FPGA 原型系统上的实时验证？

还有，价格如何？能否在多个项目多次使用？后期的 License 费用？等等

这些问题在 N 年之前都是令人头痛的问题，相信今天的项目主管还是会有一样的感觉。PODES 系列开源 MCU 为项目主管展现了一个新的前景：

	ARM 系列	PowerPC	LEON2/3	PODES
主要特点	主流，商业	非主流，商业	非主流，开源	主流，开源
费用	极高	较高	低或无	低或无
提供源代码	基本不可能	极少	有	有
支持 FPGA 原型验证	否	否	是	是
在线调试能力	支持	支持	弱	支持
编译器及开发环境	主流	非主流	非主流	主流
面积/性能/功耗可裁剪性	无	无	弱	强
代码的可靠性	高	高	低	较高
生态系统成熟度	成熟	小范围	不成熟	成熟
商业投资的持续回报	较好	较弱	较弱	最好

PODES：**P**rocessor **O**ptimization for **D**eep **E**mbedded **S**ystem。包括传统的 51 指令集架构，SparcV8 指令集架构，ARMv6-M 指令集架构以及 PIC-16 指令集架构等一系列 MCU Core。

PODES 目标定位于深嵌入式 SoC 设计应用。

深嵌入式系统具备如下主要特征：单颗芯片集成一个包括 MCU 的完整软件硬件系统，MCU 资源不提供或者少量提供给外部开发，MCU 对用户来说基本不可见。下面是一些内嵌有 MCU 的芯片典型例子。

Bluetooth, Zigbee 控制器芯片
RFID, NFC 阅读器芯片
无线充电 Transmitter 芯片
电源管理芯片
手机 SIM 卡芯片
USB 读卡器芯片
摄像头控制芯片
各种协议转换芯片
Sensor Hub 芯片
USB TypeC 充电芯片

PODES-M0O 是一个极其精简的开源 MCU Core。通过完全开放的源代码，用户可以非常容易地评估该 MCU 的各项性能指标。

PODES-M0O 最大特征是完全兼容 ARMv6-M 指令集。用户可以充分利用 ARM 体系结构现有的生态系统资源，设计基于 PODES-M0O 的 MCU。

ARM 架构基本上已经处于嵌入式应用的统治地位，大量的工程资源和相关知识可供使用。从传统的 8bit MCU 或者非主流的指令架构迁移到 ARM 架构，虽然非时非力，但从长期商业价值考虑，似乎也必不可少。PODES-M0O 可以协助用户以极低成本快速地迁移到 ARM 架构。

PODES 泛指一系列可应用于深嵌入式系统的 MCU Cores。 M0O 表示兼容 Cortex-m0 的 OpenSource 发布。本手册详细地描述 PODES-M0O 应用的方方面面，包括代码集成及设计，软件开发，硬件评估等等。

2 对象和范围

PODES-M0O 是一个经过专门精简优化的开源版本，定位于学习和研究。把他应用在一般的 FPGA 产品中没有问题，用于 ASIC 实现则需要一些额外的设计修改工作。如果用户准备做 ASIC 实现的项目，推荐使用 **PODES-M0A** 版本。**PODES-M0O** 可以用于前期可行性评估。

本手册使用一个工程实例（**AMY_M0O**）来介绍 **PODES-M0O** 的应用及开发过程。主要目标对象为：**个人学习者**。尤其是那些具备一定的基础知识，准备涉足 SoC 设计和应用的人员。比如逻辑设计工程师、在校学生等等。

一个 SoC 设计，无论多么简单，都要涉及到指令集设计、RTL 代码开发、RTL 仿真、FPGA 硬件系统验证、编译器的开发/使用、嵌入式 C 程序开发、甚至操作系统的裁剪。正所谓麻雀虽小，五脏俱全。熟悉或者了解上述知识和相关的工具使用，有助于快速上手。

为了帮助个人学习者尽快上手，**AMY/PODES-M0O** 已经做到了尽量简化。去掉了大量与实际 ASIC 实现相关的代码；在保留核心的前提下优化结构；尽量使用常见和易得的开发工具；编写精简的开发脚本；甚至提供一个完整的 FPGA 评估板。

在如何达到精简易用方面，作者动了不少心思。用剃刀一层一层地刮，直到最后只剩下一堆骨架，再无从下手了。现在你手头的代码已经是数易其稿后的结果。毕竟，SoC 芯片开发在 IC 设计公司一般都经由至少三个不同技术领域的团队协作完成。把这些简化到个人学习者容易接受的程度对我来说确实有一点困难。简单的事情弄复杂一般人都会，复杂的事情弄简单不容易！想到 **AMY/PODES-M0O** 是用于学习和研究的，或许简约而不简单就应该是他的本来面目。

本手册只关注 PODES-M0O 的应用。里面有相当多的地方涉及到 PODES-M0O 具体功能和结构的实现，都没有展开描述。读者若需要详细了解，可以参考下面的 PODES-M0O 设计手册：

PODES-M0O_Implementation_User_Manual_Vxx.doc

本手册只关注 PODES-M0O 的应用。与 FPGA Evaluation board 使用相关的细节没有展开描述。读者若需要详细了解，可以参考下面的 PODES-M0O 评估板用户手册：

PODES_M0O_Evaluation_Board_User_Manual_Vxx.doc

另外，有关指令集的深入学习则需要研究 Cortex-M0 的相关资料，下面的文档可供参考：

DDI0432C_cortex_m0_r0p0_trm.pdf

DUI0497A_cortex_m0_r0p0_generic_ug.pdf

DDI0419B_arm_architecture_v6m_reference_manual_errata_markup_2_0.pdf

本手册涉及到的相关文档、代码及软件工具可能涉及到不同类型版权问题。因此，用于商业产品开发之前请仔细确认相关版权信息。

3 支持和服务

www.mcucore.club 是 PODES 开源项目的官方维护网站。

立足于保证 PODES 有用，作者会持续地维护这个项目。所有代码和文档资料的最新版本都可以从下面网站获得：
www.mcucore.club

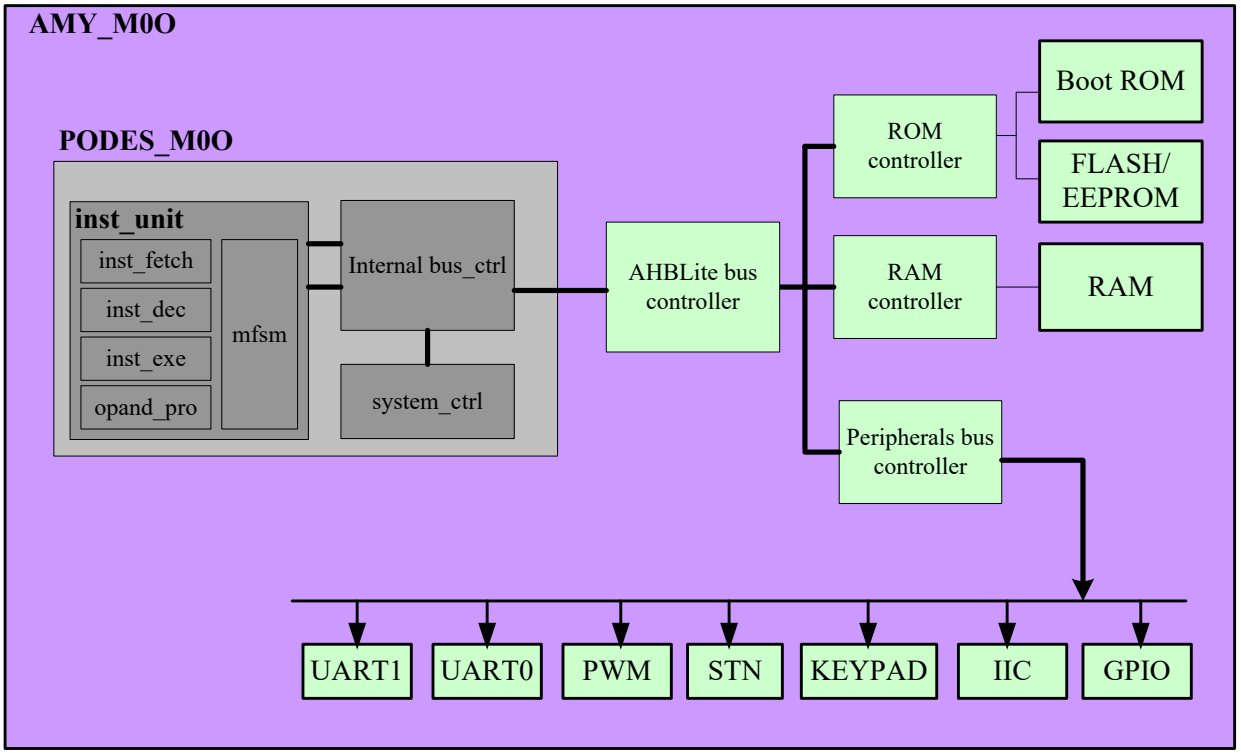
所有的 Issue Report 或者优化建议，请投送到：www.mcucore.club 相关的页面，或者：podes.mcu@qq.com 。

关于捐赠
小额赞助、购买 FPGA 开发板、提供开发支持、甚至是一条建议或者评论，都是鼓舞 PODES 前行的动力。
有意赞助者，可以扫描下方的二维码：

4 AMY—PODES-M0O 应用实例

4.1 AMY 硬件结构框图

最小评估系统结构框图



AMY for M0O 的结构如上图，构成一个 PODES-M0O 处理器内核的最小评估系统（相当于一个简单 MCU 芯片）。

AMY 的外围设备包括 32bit GPIO、2 个 UART、1 个 IIC、一个键盘、1 个 STN、1 个 PWM。应用模式：IIC 连接外部 EEPROM/FLASH 存储芯片；GPIO 扩展应用；STN 显示功能；KEYPAD 输入；PWM 电机驱动控制。

评估系统工作流程为：内建 boot 代码接收串口数据，写入内存或者 IIC 接口的 EEPROM 芯片。硬件自动从内存/EEPROM 芯片中读取代码，存入片内 RAM 然后运行。

FPGA 评估板对 ROM 控制器代码做了进一步简化。ROM 空间使用 FPGA 内部 SRAM 实现，另外提供一个 1Kbyte 的影子 ROM 空间，存放 Boot 代码。Boot 代码直接固化在 FPGA 中，在下载模式时这段 Boot 代码运行，可以接收外部代码下载，转存到 ROM 空间（FPGA SRAM）中。退出 boot 模式，系统复位后程序正常运行。

4.2 AMY 引脚列表

Port name	IO	Description
CLKIN	I	MCU clock input
RSTN	I	MCU reset input. Active low.
BOOT	I	0: Boot mode;1: Normal mode.
RXD0	I	UART0 port
TXD0	O	
RXD1	I	UART1 port
TXD1	O	
SCL	IO	IIC port
SDA	IO	
ROW0	O	5x5 matrix keypad port
ROW1	O	
ROW2	O	
ROW3	O	
ROW4	O	
COL0	I	
COL1	I	
COL2	I	
COL3	I	
COL4	I	
IRIN	I	Infare control input port
GPIO0	IO	8bit GPIO Ports
GPIO1	IO	
GPIO2	IO	
GPIO3	IO	
GPIO4	IO	
GPIO5	IO	
GPIO6	IO	
GPIO7	IO	

注：部分功能引脚没有列出。外围模块功能和引脚依据 FPGA 实现的差异会有变化，特定版本的 FPGA 参考设计，请参见 FPGA 评估板文档。

4.3 AMY Memory Mapping

PODES_M0O 实现下面蓝色字体标注的四个空间，其他保留。

address	Name	Device Type	XN	Cache	Description
0x0000_0000 ~ 0x1FFF_FFFF	Code	Normal	-	WT	Typical ROM or flash memory. Memory required from address 0x0 to support the vector table for system boot code on reset.
0x2000_0000 ~ 0x3FFF_FFFF	SRAM	Normal	-	WBWA	SRAM region typically used for on-chip RAM.
0x4000_0000 ~ 0x5FFF_FFFF	Peripheral	Device	XN	-	On-chip peripheral address space.
0x6000_0000 ~ 0x7FFF_FFFF	RAM	Normal	-	WBWA	Memory with write-back, write allocate cache attribute for L2/L3 cache support.
0x8000_0000 ~ 0x9FFF_FFFF	RAM	Normal	-	WT	Memory with write-through cache attribute.
0xA000_0000 ~ 0xBFFF_FFFF	Device	Device Shareable	XN	-	Shareable device space.
0xC000_0000 ~ 0xDFFF_FFFF	Device	Device	XN	-	Non-shareable device space.
0xE000_0000 ~ 0xFFFF_FFFF	System	-	-	-	System segment including the PPB.

在此基础上 AMY 做进一步的划分，Memory 空间分配如下表：

ROM 空间：0x0000_0000 ~ 0x1FFF_FFFF

address	Description

0x0000_0000 ~0x0000_FFFF	16Kbytes ROM space.
Others	Reserved.

Boot ROM 空间：0x0000_0000 ~ 0x0000_03FF

address	Description
0x0000_0000 ~0x0000_03FF	1Kbytes ROM space.
Others	Reserved.

Boot ROM 空间跟正常的代码 ROM 空间重叠，通过 BOOT 引脚来区分。

RAM 空间：0x2000_0000 ~ 0x3FFF_FFFF

address	Description
0x2000_0000 ~0x2000_1FFF	4Kbytes RAM space.
Others	Reserved.

注： ROM/BootROM/RAM 空间的大小在 PODES-M0O 中可以自由修改。以上只是 AMY Demo 的默认配置。

外设空间：0x4000_0000 ~ 0x5FFF_FFFF

address	Description
0x4000_0000 -- 0x4000_00ff	UART0
0x4000_0100 -- 0x4000_01ff	UART1
0x4000_0200 -- 0x4000_02ff	IIC
0x4000_0300 -- 0x4000_03ff	KEYPAD
0x4000_0400 -- 0x4000_04ff	GPIO
0x4000_0500 -- 0x4000_05ff	STN
0x4000_0600 -- 0x4000_06ff	PWM
0x4000_0700 -- 0x4000_07ff	RSV
others	Reserved.

4.4 AMY 中断资源

PODES_M0O 支持 32 个外部中断，AMY 内部功能模块的中断号映射如下表：

INT Number	Description
INT0	UART0 中断
INT1	UART1 中断
INT2	IIC 中断
INT3	keypad 中断
INT4	GPIO 中断
INT5	STN 中断
INT6	PWM 中断
INT7—INT31	保留

4.5 AMY 外围模块

PODES_M0O 内核功能以及寄存器描述参考 ***PODES_M0O_Implementation_User_Manual_Vxx.pdf*** 文档。下面的寄存器描述只涉及到 AMY 实现的外围功能模块。

4.5.1 通用串口 UART

通用串口功能特性：

支持 10bit 帧格式。包括一个起始位，8bit 数据位，一个可选的校验位，一个停止位。
12bit 波特率发生器。
单字节收发，不支持 FIFO 模式。
支持硬件流控。
支持 Rx/Tx 内部环回测试。
支持外部参考时钟。

中断规则：

当发送保持寄存器 thold 的值被移入发送移位寄存器 tshift 后立即产生一个中断，告诉处理器可以写入下一个字节。
每当完成一个字符的接收，产生一个接收中断，告诉处理器可以读出接收的字符。软件需要根据状态寄存器来判断是否接收到一个完整有效的字符。错误状态可能是：frameerr, parerr, break, ovf，只有这些状态为 0 并且 dready 为 1 才表示接收的字符有效。
接收或者发送中断条件满足后，此模块会触发一次中断脉冲。
处理器收到中断后应先读取状态寄存器。区分发送中断、接收中断、以及接收的字符是否有错，然后访问（读或者写）数据寄存器。

状态寄存器的更新：

Frameerr，parerr，break，ovf 在接收数据时自动更新，1 表示收到字符错误。软件对状态寄存器执行一个写操作（不关心写入的值）时这些状态位被清零。

接收到一个完整的字符后 dready 为 1；软件从 rhold 执行读操作后，dready 更新为 0。

软件向 Thold 中写入一个字节时，Thempty 状态为 0；thold 中的数据一旦移入 tshift 寄存器，thempty 变成 1。提示软件可以写入下一个待发送字节。

如果 thold 中已经没有数据，tshift 的字符已经发送完成，tsempty 会变成 1。

波特率计算：

这个串口模块只支持整数分频模式，不能整除的波特率配置使用四舍五入方式取整。因此实际波特率会有一些误差。

Baud Rate = Frequency (Hz) / (scaler +1)*8

Scaler 装载值可以使用下面的公式计算，

Scaler value = Frequency / (8*baud_rate) - 1

常用的波特率和 scaler 值对照表（Freq = 12000000Hz）

Baud Rate	Scaler alue
1200	1249
2400	624
9600	155.25
38400	38.0625
57600	25.0417
115200	12.0208

Scaler 只能装载整数值，上表中的小数部分需要直接舍入。

串口寄存器定义：

Address offset	Description
0x0	UART data register
0x4	UART Status register
0x8	UART Control register
0xc	UART Scaler register

UART Data Register (offset addr: 0x0)

Bits	R/W	Reset	Description
[31:8]	RW	0	Reserved.
[7:0]	RW	0	Receiver holding register (read access) Transmitter holding register (write access) 收发共享数据端口地址。硬件是独立的，彼此互不影响。

UART Status Register (offset addr: 0x4)

Bits	R/W	Reset	Description
[31:7]	RW	0	Reserved.
[6]	RW	0	Frame Error 1：字节帧格式错误。
[5]	RW	0	Parity Error 1：字节校验错误。
[4]	RW	0	Overrun 1：至少一个字节溢出丢失。
[3]	RW	0	Break Received 1：接收到 Break 字符。（00）
[2]	RW	1	Transmitter Holding register empty 1：发送寄存器已经空。

[1]	RW	1	Transmitter shift register empty 1：发送移位寄存器已经空。
[0]	RW	0	Data ready 1：接收 Holding register 有一个新数据。

UART Control Register (offset addr: 0x8)

Bits	R/W	Reset	Description
[31:9]	RW	0	Reserved.
[8]	RW	0	External clock 1：波特率发生器使用外部时钟。
[7]	RW	0	Loopback 1：Rx 和 Tx 环回。
[6]	RW	0	Flow control 1：允许通过 CTS/RTS 实现硬件流控。
[5]	RW	0	Parity enable 允许生成和校验 Parity bit
[4]	RW	0	Parity select 0: even parity; 1: odd parity.
[3]	RW	0	Transmitter interrupt enable.
[2]	RW	0	Receiver interrupt enable.
[1]	RW	0	Transmitter enable.
[0]	RW	0	Receiver enable.

UART Scaler Register (offset addr: 0xC)

Bits	R/W	Reset	Description
[31:12]	RW	0	Reserved
[11:0]	RW	0	Scaler reload value

4.5.2 GPIO 功能

GPIO Output:

在配置成输出功能时，GPIO 引脚的电平即为 GPIO Output 寄存器的值。配置成输入功能时，GPIO Output 寄存器的值不会影响 GPIO 引脚的电平。

GPIO Input:

在配置成输入功能时，输入寄存器的值直接锁存 GPIO 引脚的电平。如果配置成输出，则输入寄存器的值保持为 0。

GPIO 中断:

可以配置成 Low to High 或者 High to Low 边沿中断。中断只在输入模式下有效。
需要使用中断时，必须在 Enable 中断之前清一次中断状态。

GPIO 寄存器定义:

8 个 GPIO 引脚直接映射到 GPIO[31:0]的低 8bit。下面的寄存器低 8bit 有效，每一个 bit 对应一个引脚。相关寄存器如下表。

Address offset	Description
0x0	GPIO Output Value register
0x4	GPIO Direction register
0x8	GPIO input Value register

0xc	GPIO interrupt control register
0x10	GPIO interrupt enable register
0x14	GPIO interrupt status register

GPIO Output Value Register (offset addr: 0x0)

Bits	R/W	Reset	Description
[31:8]	RW	0	Reserved
[7:0]	RW	0	Output Value.

GPIO Direction Register (offset addr: 0x4)

Bits	R/W	Reset	Description
[31:8]	RW	0	Reserved
[7:0]	RW	0	GPIO direction control 1: output; 0 input. If 1, corresponding bit of GPIO Output Value Register will output to Pin. If 0, corresponding pin will be used as input pin. Output Value has on effect on that pin.

GPIO Input Value Register (offset addr: 0x8)

Bits	R/W	Reset	Description
[31:8]	RW	0	Reserved
[7:0]	RW	0	Input Value.

			The values of corresponding pins are latched in this register.

GPIO Interrupt Mode Register (offset addr: 0xC)

Bits	R/W	Reset	Description
[31:8]	RW	0	Reserved
[7:0]	RW	0	GPIO interrupt mode register 1: low to high on GPIO input value will generate interrupt. 0: high to low interrupt.

GPIO Interrupt Enable Register (offset addr: 0x10)

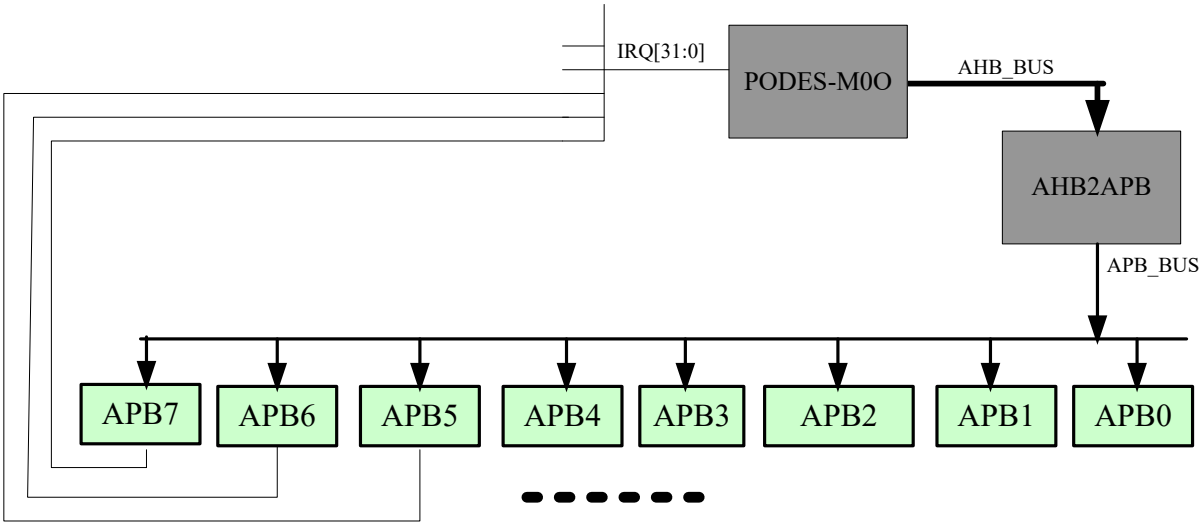
Bits	R/W	Reset	Description
[31:8]	RW	0	Reserved
[7:0]	RW	0	GPIO interrupt enable register 1: Interrupt enable. 0: Interrupt disable.

GPIO Interrupt Status Register (offset addr: 0x14)

Bits	R/W	Reset	Description
[31:8]	R/W 1C	0	Reserved
[7:0]	R/W 1C	0	GPIO interrupt enable register 1: An interrupt occurs on the pin. 0: no interrupt.

4.6 AMY 功能扩展

AMY/PODES-M0O 的结构设计为功能扩展做了特别优化，用户只需要将自己设计的 APB 接口模块挂接在系统提供的 APB 总线上即可。



用户可以替换 AMY_M0O.v 中现有的 APB module 或者添加自己设计的 APB 模块。如果新模块实现中断功能，需要为它选择一个中断号。参考下面的代码片段。

AMY_M0O.v

```
…… 

wire [31:0] irq_in = {
                    24'b0,
                    rsv_irq,
                    pwm_irq,
                    stn_irq,
                    gpio_irq,
                    apbkey_irq,
```

```
                iic_irq,
                uart1_irq,
                uart0_irq
                };
……

ahb2apb ahb2apb_u0 (
            .clk        (clk            ),
            .rst_n      (glb_rst_n      ),

            .shready_in (peri_shready_in ),
            .shsel      (peri_shsel      ),
            .shaddr     (peri_shaddr     ),
            .shtrans    (peri_shtrans    ),
            .shwrite    (peri_shwrite    ),
            .shwdata    (peri_shwdata    ),
            .shsize     (peri_shsize     ),
            .shburst    (peri_shburst    ),
            .shprot     (peri_shprot     ),
            .shrdata    (peri_shrdata    ),
            .shready_out(peri_shready_out),
            .shresp     (peri_shresp     ),

            //UART0
            .apb0_psel   (uart0_psel   ),
            .apb0_penable(uart0_penable),
            .apb0_paddr  (uart0_paddr  ),
            .apb0_pwrite (uart0_pwrite ),
            .apb0_pwdata (uart0_pwdata ),
            .apb0_prdata (uart0_prdata ),
            //UART1
            .apb1_psel   (uart1_psel   ),
            .apb1_penable(uart1_penable),
            .apb1_paddr  (uart1_paddr  ),
```

```
        .apb1_pwrite (uart1_pwrite ),
        .apb1_pwdata (uart1_pwdata ),
        .apb1_prdata (uart1_prdata ),
        //IIC
        .apb2_psel   (iic_psel     ),
        .apb2_penable(iic_penable  ),
        .apb2_paddr  (iic_paddr    ),
        .apb2_pwrite (iic_pwrite   ),
        .apb2_pwdata (iic_pwdata   ),
        .apb2_prdata (iic_prdata   ),
        //APBKEY
        .apb3_psel   (apbkey_psel   ),
        .apb3_penable(apbkey_penable),
        .apb3_paddr  (apbkey_paddr  ),
        .apb3_pwrite (apbkey_pwrite ),
        .apb3_pwdata (apbkey_pwdata ),
        .apb3_prdata (apbkey_prdata ),
        //GPIO
        .apb4_psel   (gpio_psel    ),
        .apb4_penable(gpio_penable ),
        .apb4_paddr  (gpio_paddr   ),
        .apb4_pwrite (gpio_pwrite  ),
        .apb4_pwdata (gpio_pwdata  ),
        .apb4_prdata (gpio_prdata  ),
        //STN
        .apb5_psel   (stn_psel     ),
        .apb5_penable(stn_penable  ),
        .apb5_paddr  (stn_paddr    ),
        .apb5_pwrite (stn_pwrite   ),
        .apb5_pwdata (stn_pwdata   ),
        .apb5_prdata (stn_prdata   ),
        //PWM
        .apb6_psel   (pwm_psel     ),
        .apb6_penable(pwm_penable  ),
```

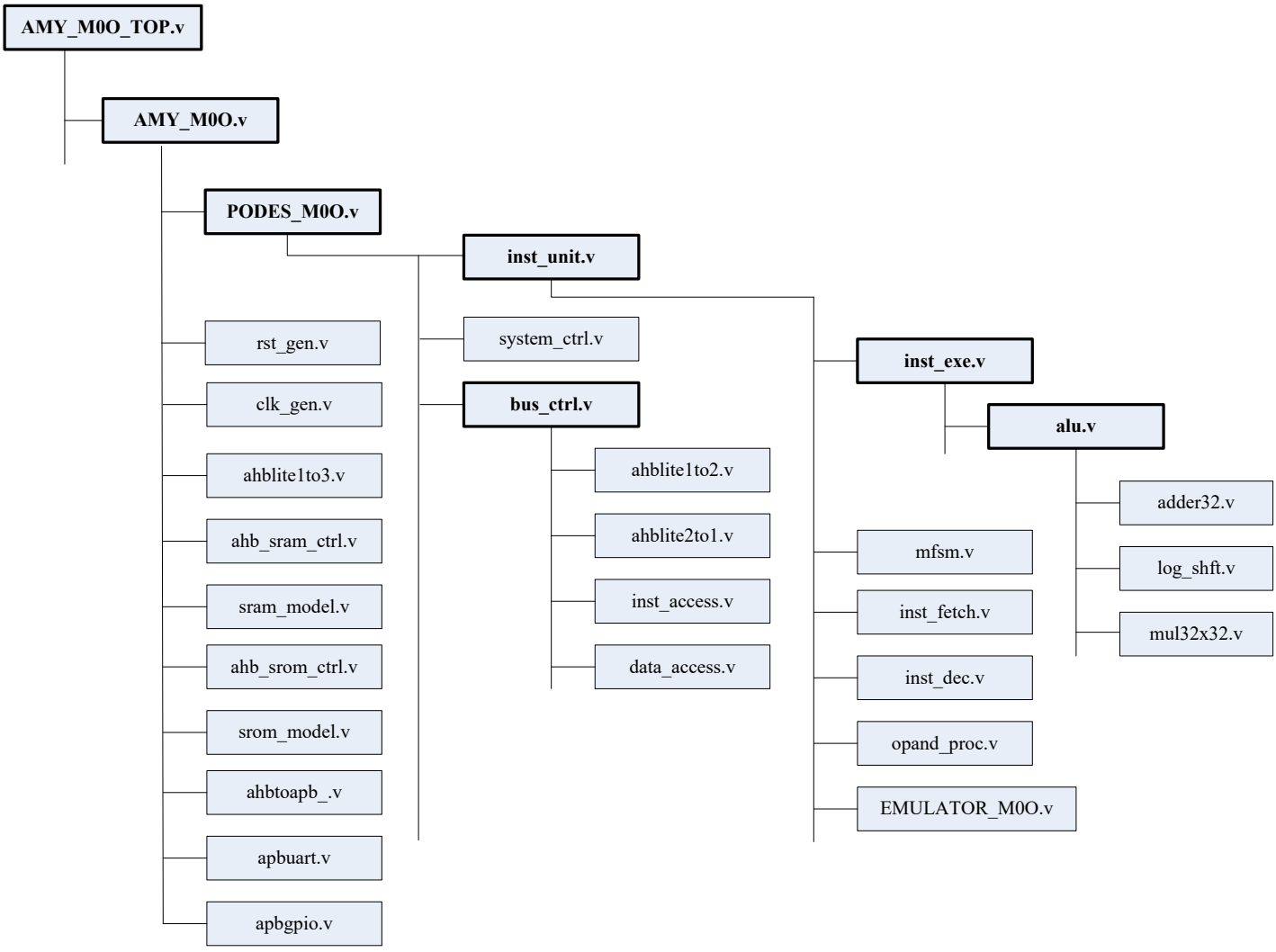
```
        .apb6_paddr  (pwm_paddr    ),
        .apb6_pwrite (pwm_pwrite   ),
        .apb6_pwdata (pwm_pwdata   ),
        .apb6_prdata (pwm_prdata   ),
        //RSVED
        .apb7_psel   (rsv_psel     ),
        .apb7_penable(rsv_penable  ),
        .apb7_paddr  (rsv_paddr    ),
        .apb7_pwrite (rsv_pwrite   ),
        .apb7_pwdata (rsv_pwdata   ),
        .apb7_prdata (rsv_prdata   )

        );
        ……
```

5 AMY 代码仿真指南

5.1 AMY 代码结构

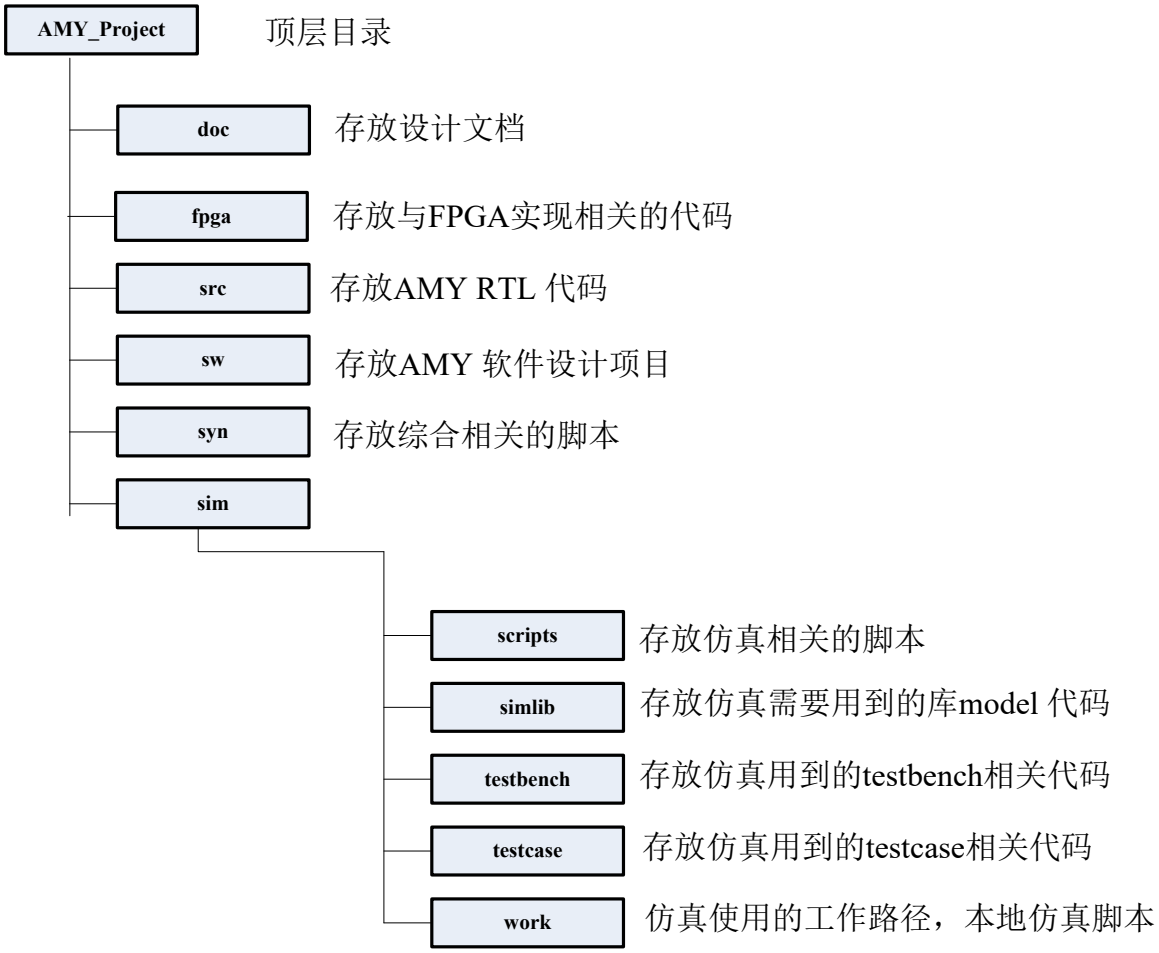
下面图形展示了 AMY 内部 RTL 代码层次结构和模块调用关系。



5.2 AMY 仿真环境

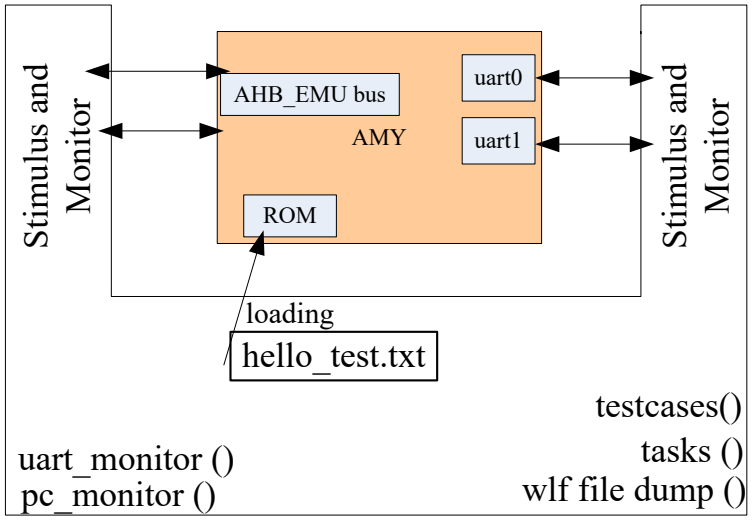
5.2.1 AMY 环境目录结构

仿真的脚本与相关文件存放位置相关联，描述 AMY 的仿真环境必须首先定义 AMY 代码的路径结构。



5.2.2 AMY testbench 结构

Testbench.v 包括测试对象 AMY 模块的端口例化，信号激励和监控，波形文件 dump 等常用功能。



Testbench.v

Testbench 环境支持两种仿真模式：

1. 软件仿真

在仿真启动阶段，测试程序二进制代码（比如 hello_test.txt）会被装载到 AMY 的 ROM 中。Uart_monitor 会实时监控测试程序写入串口中的数据并在仿真过程中显示出来（相当于一个标准输出功能）。

可选的 pc_monitor 可以监控全部已经运行的指令 PC，并 log 到文件中。可选的 emulator 可以生成全部已运行的指令的反汇编代码，并 log 到文件中。这两个 log 文件可以提供简单的仿真状态追踪功能。

每一个仿真的 C 测试程序结尾都固定输出一个字符“~”。如果 uart_monitor 监控到程序结束标记字符“~”，仿真会自动结束。

正常情况下，串口输出字符的速度很慢，程序仿真时需要添加宏定义 SPEEDUP_SIM。这可以屏蔽串行数据输出，程序可以连续向 APBUART 的 DATA 端口写数据。

2. 外围模块 RTL 代码仿真

如果用户开发自己的外围模块，可以直接在这个 testbench 环境中做 RTL 代码仿真。Tasks 中包含 AHB 总线读写相关的 task。如果编译时定义 AHB_EMU 宏，控制外围模块的 AHB Slave 总线接口可以在 testbench 中直接访问。用户编写的 testcase 模块可以通过 ahb task 来读写外围模块。

5.2.3 C 程序 仿真脚本

此脚本实现从库编译，代码编译到执行仿真批处理的全部过程。ARG 1 为需要运行测试的 C 程序； ARG 2 为需要加入的 testcase。此脚本用于 C 程序仿真测试，testcase 使用 none.v。如下面的例子。

make_simv_sw.csh

```
#!/bin/bash -f

#------------------
#Clear old lib files
#------------------
vdel  -lib podesm0o_lib   -all
vlib  podesm0o_lib
vdel -lib work         -all
vlib  work




#-----------------------
#Complie verilog files to PODESM0O_LIB
#-----------------------
 vlog \
    -work podesm0o_lib   \
```

```
    -sv \
        -novopt  \
    -f ../../src/PODES_M0O/PODES_M0O_filelist.f


#------------------------
#Complie Amy files
#------------------------

 vlog \
    -work work   \
    -timescale 1ps/1ps \
        +define+SPEEDUP_SIM\
    -f ../../src/ahbbus/ahbbus_filelist.f \
    -f ../../src/peri/peri_vlog_filelist.f \
        -v ../simlib/altera/altera_mf.v \
    -f ../../src/amy/AMY_filelist.f




#------------------------
#Load program
#------------------------
rm -f program.txt
cp ../testcase/$1/$1.txt program.txt
#cp ../testcase/$1/$1.txt ram32x4096_init.rif


#------------------------
#Compile testbench
#------------------------
vlog \
    -timescale 1ps/1ps\
```

```
    -sv \
    +incdir+../testbench \
    +incdir+../testbench/header \
    -work work   \
    -novopt \
          ../testbench/uart_monitor.v \
    ../testbench/pc_tracking.v \
    ../testbench/testbench.v \
    ../testcase/$2.v


#------------------------
#Run simulation
#------------------------
 vsim    \
    -c \
    -l simulation.log \
    -novopt +notimingchecks \
    -L work  \
    -L podesm0o_lib \
    work.testbench \
    -do ../testbench/run_finish
```

下面的例子为调用 make_simv_sw.csh 来运行 Hello_test C 测试程序。此脚本文件名为 hello_test，存放在 Work 目录中。

hello_test

```
#!/bin/bash -f
 bash ../scripts/make_simv_sw.csh  hello_test none
```

默认仿真条件下，仿真完成后会输出下列文件：

vsim.wlf：仿真的波形文件，已经把 testbench 及一下所有模块的信号 dump 出来。用户可以使用 vsim -view 后处理来查看信号波形。
pc_tracking.log ：所有运行指令的 PC 全部按顺序 log 出来。如果仿真出错，可以追踪程序运行的位置。
Disassembly.log：所有运行指令全部反汇编并且按执行顺序 log 出来。用户可以获得更多执行信息。

5.2.4 RTL 仿真脚本

此脚本的程序代码直接使用 sw_loop.txt。testcase 为用户自己编写的仿真代码。

make_simv_rtl.csh

```
#!/bin/bash -f

#------------------
#Clear old lib files
#------------------
vdel  -lib podesm0o_lib   -all
vlib  podesm0o_lib
vdel -lib work         -all
vlib  work




#------------------------
#Complie verilog files to PODESM0O_LIB
```

```
#-----------------------
 vlog \
    -work podesm0o_lib   \
    -sv \
          -novopt  \
    -f ../../src/PODES_M0O/PODES_M0O_filelist.f


#-----------------------
#Complie Amy files
#-----------------------

 vlog \
    -work work   \
    -timescale 1ps/1ps \
    +define+SPEEDUP_SIM\
    +define+USE_RIF \
    +define+AHB_EMU \
    -f ../../src/ahbbus/ahbbus_filelist.f \
    -f ../../src/peri/peri_vlog_filelist.f \
   -v ../simlib/altera/altera_mf.v \
    -f ../../src/amy/AMY_filelist.f


#-----------------------
#Load program
#-----------------------
rm -f program.txt
cp ../testcase/sw_loop.txt ram32x4096_init.rif


#-----------------------
#Compile testbench
#-----------------------
```

```
vlog \
    -timescale 1ps/1ps\
    -sv \
    +incdir+../testbench \
    +incdir+../testbench/header \
    -work work   \
    -novopt \
   ../testbench/uart_monitor.v \
    ../testbench/pc_tracking.v \
    ../testbench/testbench.v \
    ../testcase/$1.v


#------------------------
#Run simulation
#------------------------
 vsim    \
    -c \
    -l simulation.log \
    -novopt +notimingchecks \
    -L work  \
    -L podesm0o_lib \
    work.testbench \
    -do ../testbench/run_finish
```

下面的例子为调用 make_simv_rtl.csh 来运行 apbuart_case.v 测试 APBUART 模块。此脚本文件名为 apbuart_case，存放在 Work 目录中。

apbuart_case

```
#!/bin/bash -f
```

```
bash ../scripts/make_simv_rt.csh  apbuart_case
```

Note1：设计中使用到了 Altera FPGA 的 Memory Core，上述脚本涉及到对 Altera 的 RAM model 仿真库的调用。如果使用 General 或者 Xilinx FPGA 的 memory model，仿真脚本需要做相应的处理。

Note2：以上仿真环境涉及的的仿真工具为 Modelsim.

6 AMY 软件开发指南

6.1 概述

PODES-M0O 兼容 Cortex-M0 指令集，处理器内部寄存器和集成的 systemTick，NVIC 模块寄存器也完全兼容 Cortex-M0。因此，用户开发基于 PODES-M0O 的软件并不需要学习专用的编译器和汇编器。现有的 ARM 开发资源可以直接使用，免除学习新开发环境的过程。

AMY 用 PODES-M0O 做 MCU 内核，加入各种外围功能。因此对于 AMY 的软件开发，用户需要做的工作只是针对自己添加的外围功能模块编写驱动程序或者 Firmware 例程。

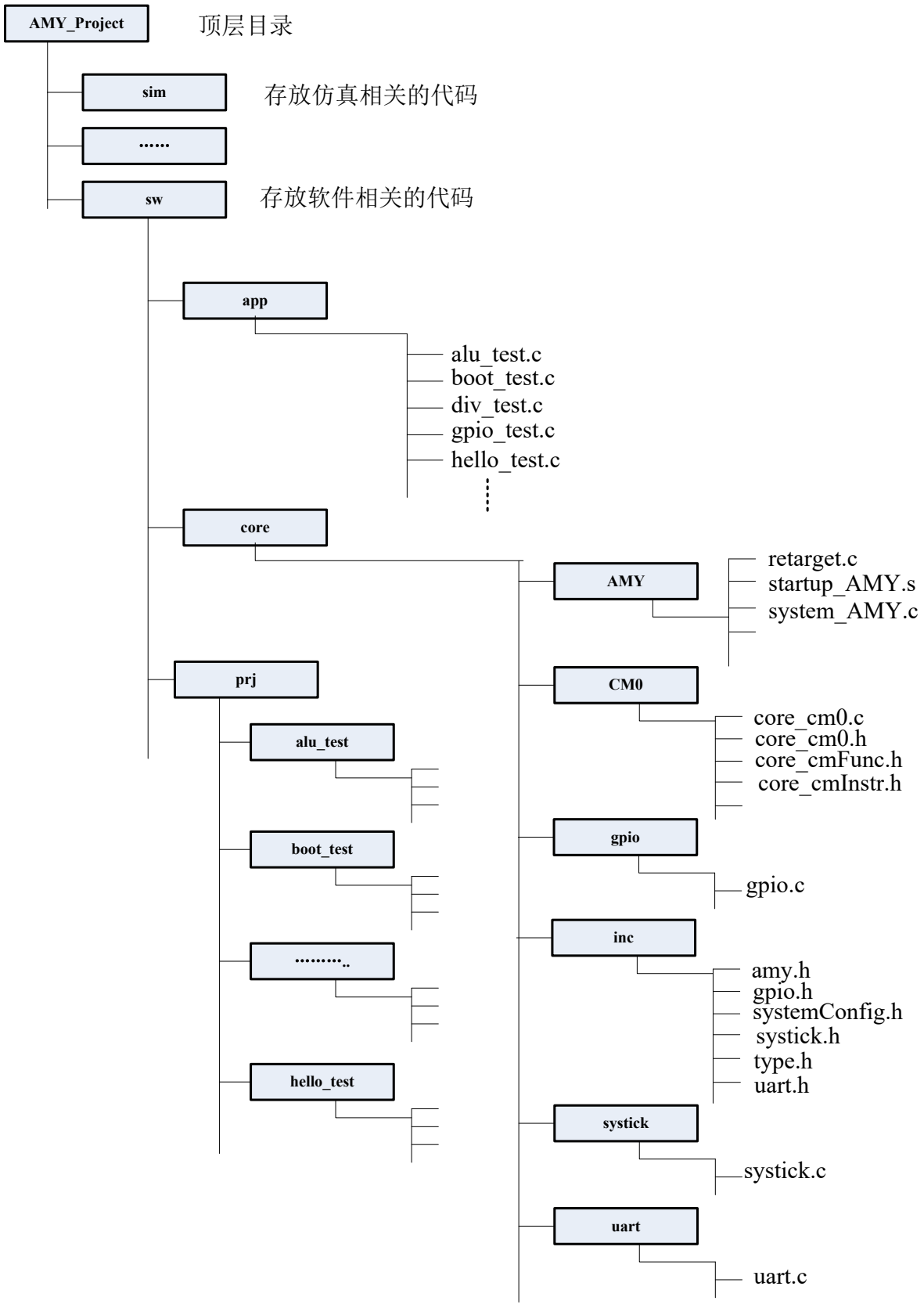
开发用于仿真的测试程序和实际应用程序的方法基本相同。但是测试程序需要与 RTL 仿真的 Testbench 环境相配合，需要在代码结构上做一点特别处理。另外，应用程序和测试程序的 load 方式也不同。下面的段落通过例子说明测试程序的开发和 Load 方式。应用程序的使用方法会在硬件使用手册中说明。

6.2 测试程序开发环境

下面的图例展示了软件相关的目录结构及其包含的基本文件。

name	Dir/files	description
/app	目录	存放 AMY 测试应用程序。比如 Hello_test.c
/core	目录	存放通用 PODES_M0O 内核代码和 AMY 功能模块驱动程序。
/prj	目录	存放应用程序的工程文件。
/core/AMY	目录	存放 MCU 的 startup 和 system 启动代码。
/core/cm0	目录	存放 Cortex-m0 内核抽象层代码。
/core/inc	目录	存放测试应用程序共同的 Include 文件。
/core/uart	目录	存放功能模块的硬件抽象层程序。
/core/AMY /retarget.c	文件	标准输入输出重定向代码。需要下载到目标机的程序不能使用半主机模式。需要重定向输入输出。
/core/AMY /startup.c	文件	目标系统初始化代码存放在这里。比如配置 PLL 等硬件环境。根据系统的复杂度而定，简单的系统比如 AMY 可以不需要这个程序。
/core/AMY /startup.s	文件	汇编代码。构造中断向量表，堆栈指针和空间分配。指导 Link 程序确定应用代码存放的绝对地址。ARM 系统必备。
/core/systick/systick.c	文件	PODES_M0O 内嵌 SysTick 模块的 Firmware Routine。
/core/uart/uart.c	文件	AMY UART 模块的 Firmware Routine。

注：以上只是部分示例，不同源代码版本包含的内容会有少许差异。 具体内容请以最新版本源代码为准。


AMY_Project
顶层目录
sim
存放仿真相关的代码
.....
sw
存放软件相关的代码
app
alu_test.c
boot_test.c
div_test.c
gpio_test.c
hello_test.c
core
AMY
retarget.c
startup_AMY.s
system_AMY.c
CM0
core_cm0.c
core_cm0.h
core_cmFunc.h
core_cmInstr.h
gpio
gpio.c
inc
amy.h
gpio.h
systemConfig.h
systick.h
type.h
uart.h
systick
systick.c
uart
uart.c
prj
alu_test
boot_test
.....
hello_test

6.3 测试程序代码结构

开发 C 程序用于仿真时，主体测试代码结束后需要在末尾输出一个 ASCII 字符“~”（0x7e）。在 testcase 中的 uart_monitor.v task 会监控 UART 输出的字符，如果发现“~”则立即停止运行，退出仿真。参考下面的例子：

hello_test.c

```
#include <stdio.h>
#include <string.h>

#include "amy.h"

 int main (void)
{
  int i = 1;

//initialize the UART
   uartInit(CFG_UART_BAUDRATE);

//output a string
   printf ("%x \n", i);
          printf ("start the test program: hello_test!\n");

          while (i<5) {
          printf ("current i = %x \n", i);
          i++;
          }

//stop the program.
```

```
        uartSendByte('~');
        while(1);
}
```

6.4 测试程序开发

用户可以使用 Linux 下的 ARM Tool Chain 环境或者直接在 Windows 环境下使用 Keil uVision 工具来开发和调试代码。
下面以 Keil uVision4.0 为例，概述测试程序开发需要的配置和注意事项。有经验者可忽略。

6.4.1 指定目标器件

PODES_M0O 并不是一款量产出货的真实芯片，Keil Uvision 的默认 device Database 中无法支持它。我们需要为它创建工程开发的基础环境。

开发环境最关键的配置包括：MCU 类型，编译工具参数，程序代码空间分配规划等。

具体到 Cortex-M0 系列 MCU，ARM 已经提供完整的 CMSIS 相关文件 core_cm0.h，core_cmFunc.h，core_cmInstr.h， core_cm0.c。这些可以不加修改直接使用（因为 PODES_M0O 完全兼容 cortex-M0）。但是内核配置文件 system_device.c 和 startup_device.s 以及 device.h 是跟随具体的 MCU 芯片变化的。我们必须针对 PODES_M0O 的实现做特别的修改。

在 sw/core/AMY 中已经提供了 system_AMY.c 和 startup_AMY.s 以及 AMY.h 三个文件。用户可以直接加载这两个文件到自己的工程中。

有三种方式用于设定目标器件。

PODES_M0O 完全兼容 Cortex-M0，因此可以在创建新项目时直接选择任何一家公司的 Cortex-M0 MCU 芯片，然后在此基础上做相关参数的修改。注意：使用这种方式不要拷贝 startup_xxx.s，system_xxx.c 文件到工程中。这两个文件由用户手动加载。Section5.4.2 和 Section5.4.3 描述相关参数的修改。

也可以不用选定具体的 MCU 芯片，直接在器件列表中选择 ARM--> Cortex-M0 processor。创建新工程项目。使用这种方式不会拷贝 startup_xxx.s, system_xxx.c 文件到工程中。这种方式需要修改的参数类同上例。

还有一种办法就是自己创建一个新器件加到 Keil uVision 软件中去。这样创建新工程项目时可以直接选择 PODES_M0O 的类别。具体做法参见 http://www.keil.com/support/docs/1421.htm 。

6.4.2 指定目标器件相关参数

PODES_M0O 完全兼容 Cortex-M0，因此选定任何一家公司的 Cortex-M0 MCU 器件都可以。这里选择 NXP LPC1111。目标器件的 On-chip ROM/RAM 起始地址以及空间大小需要根据 AMY 的设计修改。

On-chip ROM/RAM	Start Address	Size
AMY ROM	0x0000_0000	0x8000
AMY RAM	0x2000_0000	0x800

6.4.3 指定编译相关的参数

下面这些参数基本上是开发环境默认配置。用户可以根据需要核对，或者修改。

C/C++ 控制参数:
-c --cpu Cortex-M0 -D__EVAL -g -O0 --apcs=interwork -I..\include -I..\..\core\include -I "C:\Keil\ARM\CMSIS\Include" -I "C:\Keil\ARM\INC\NXP\LPC11xx" -o "*.o" --omf_browse "*.crf" --depend "*.d"

ASM 控制参数：
--cpu Cortex-M0 --pd "__EVAL SETA 1" -g --apcs=interwork -I..\include -I..\..\core\include -I "C:\Keil\ARM\CMSIS\Include" -I "C:\Keil\ARM\INC\NXP\LPC11xx" --list "*.lst" --xref -o "*.o" --depend "*.d"

Linker 控制参数：
--cpu Cortex-M0 *.o --ro-base 0x00000000 --entry 0x00000000 --rw-base 0x20000000 --entry Reset_Handler --first __Vectors --strict --autoat --summary_stderr --info summarysizes --map --xref --callgraph --symbols --info sizes --info totals --info unused --info veneers --list ".\hello_test.map" -o "hello_test.axf"

6.4.4 加载内核配置文件

sw/core/src 中已经提供了 system_AMY.c 和 startup_AMY.s 以及 AMY.h 三个文件。用户需要在新建的项目中使用这三个文件。

6.4.5 其他参数配置

其他诸如 include 路径指定，List 文件内容指定，输出文件名指定等都是常规处理。用户可以根据需要来配置。

6.4.6 输出测试文件

编译出来的 elf 文件需要做一下格式转换，如下例。产生 list 文件和 hex 文件可以 copy 到/sim/testcase/目录下用于仿真。

```
C:\Keil\ARM\BIN40\fromelf.exe --vhx --32x1 --output=hello_test.txt hello_test.axf
C:\Keil\ARM\BIN40\fromelf.exe --text -c -s --output=hello_test.lst hello_test.axf
```

Hello_test.txt 文件包含以 word （32bit）为单位的 16 进制代码数据，这个代码在仿真时会被 Load 到 ROM 中。

Hello_test.txt

```
20000470
00000221
00000229
0000022B
00000000
00000000
00000000
00000000
00000000
00000000
00000000
……
```

Hello_test.lst 文件包含指令代码的反汇编程序，和符号表。可以用于跟踪代码执行过程。

Hello_test.lst

```
......

$d.realdata
   RESET
   __Vectors
      0x00000000:   20000470   p..    DCD    536872048
      0x00000004:   00000221   !...   DCD    545
      0x00000008:   00000229   )...   DCD    553
      0x0000000c:   0000022b   +...   DCD    555
      0x00000010:   00000000   ....   DCD    0
......

main
      0x00000178:   2401       .$     MOVS    r4,#1
      0x0000017a:   20e1       .      MOVS    r0,#0xe1
      0x0000017c:   0240       @.     LSLS    r0,r0,#9
      0x0000017e:   f000f86b   ..k.   BL      uartInit ; 0x258
      0x00000182:   4621       !F     MOV     r1,r4
......
```

7 AMY 评估板

这一节简要介绍评估板的功能，更详细的资料可以参考硬件使用手册。

7.1 概述

实际上，如果你手头有一块 FPGA 开发板，你可以直接把 PODES-M0O/AMY-M0O 移植到你的板子上做评估测试。

PODES-M0O 评估板是专为评估 PODES 系列 MCUCore 定制的开发板。

在硬件方面 PODES-M0O 的 FPGA 评估板使用容量合适的 FPGA (预留 50%左右的资源供扩展功能)，提供简单纯粹的外部接口。够用就行是 FPGA 评估板发行的原则。

在软件方面提供基于 IC 设计行业规范编写的 MCU Core 代码和应用实例。并且准备了详尽的应用手册文档。使用户可以专注于代码设计层面的创造性工作。

7.2 硬件接口功能

深嵌入式 MCU 芯片除了处理器内核外就是专用的外围功能模块。从评估 PODES-M0O 的角度看，只需要在其 APB 总线上连接一个简单的外围模块就可以实现等效模拟而不影响通用性。

一个 USB/UART 接口
这个评估板只用 UART 做为调试输入和输出终端，全部操控都通过串口终端完成。

一个 IIC 接口存储芯片
可以用来评估 IIC Master 和 Slave 接口功能。

4 个 LED 灯
可以用于状态指示。

Ardunio 兼容接口
可以将各种 Ardunio shield 模块连接到 FPGA 板，做扩展开发。

FPGA 保留 IO
提供剩余全部 FPGA IO 接口。

一个 Cortex-M0 MCU
用户可以在这个 MCU 上调试 C 程序，然后将调试好的程序下载到 FPGA 做对比运行。

7.3 软件调试和评估

AMY/PODES-M0O 评估板提供一个完整的 UCOSIII 多任务操作系统环境的参考实例。

PODES-M0O 的版本不提供硬件在线调试的功能，可以使用下面的流程做评估。
先将自己的源代码在板上提供的 Cortex-M0 芯片上运行，调试正常。然后将代码移植编译成 PODES-M0O 的代码。下载到 FPGA，对比双方运行的结果。

7.4 扩展应用

PODES-M0O 评估板默认的显示输出是 UART 终端。如果想使输出显示更直观，可以使用可选的 STN 显示模块配件。

PODES-M0O 评估板提供 Ardunio 兼容接口。市面上种类繁多的 Ardunio Shield 模块可以被连接到此评估板，实现用户特定的功能。

PODES-M0O 评估板开放了全部剩余 FPGA IO 引脚。方便用户将这个 FPGA 用于其他项目。

8 AMY 评估用到的工具和环境

简单和方便起见，AMY 评估尽量在 Windows 环境下完成，能不用 Linux 环境尽量不用。工具也选择常见的和容易获得的。下面列出 AMY 评估用到的工具和环境，相关工具和环境的使用方法的介绍超出本手册的范围，略过不提。
另外，IC 及 FPGA 设计相关的工具和环境有很多。用户可以使用自己熟悉的工具和环境来评估 AMY/PODES-M0O，不限于下列工具和环境。

8.1 Editor and Modelsim

代码设计使用任何一款文本编辑器都行。代码仿真使用 Modelsim SE 6.6 以上版本。全部仿真脚本都在 Windows 环境下的 Modelsim SE6.6 下测试通过。

8.2 Cygwin

全部仿真脚本都 Cygwin 和 Linux 环境下测试通过。如果用户没有 Linux 环境，最好在 Windows 机器上安装典型的 Cygwin 环境。并设置 Cygwin 的 Home 路径。当然也可以不用 Cygwin 环境，需要把仿真脚本修改成 DOS 环境可以运行的命令行。

8.3 Keil uVision

全部测试软件开发都基于 Keil uVision4.0 版本。用户需要在机器上安装相应版本的软件。当然如果用户使用 Linux 环境，也可以使用 Linux 环境下安装的 ARM ToolChain 来做软件开发。

8.4 ISE or QuartusII

FPGA 评估板提供 Altera 或者 Xilinx 的 FPGA。用户需要安装相应公司的工具。Xilinx ISE SUITE 12.1 以上版本或者 Altera QuartusII 10.1 以上版本。

8.5 Synplify

FPGA 综合可以使用 Snyplify。当然，AMY 的规模不大，直接使用 ISE 或者 QuartusII 完全没有问题。

Change Summary

REVISION HISTORY

Revision No.	Description of change	Release Date
1.0	Initial release	20200101

CopyLeft🄯

除非明确声明，PODES 项目的软件代码都以 LGPL 方式发行。所有文档则以 CC-BY-SA-4.0 方式发行。PODES 项目中涉及到第三方软件和工具遵守第三方版权规定。

分发开源软件代码时请保留原始 file header 注释。分发开源文档时请完整保留本文档第一节至三节信息。