**PODES\_M0O**

**Application User Manual**

**Ver1.0**

**Index:**

[1 概述 4](#_Toc375659936)

[2 对象和范围 7](#_Toc375659937)

[3 支持和服务 9](#_Toc375659938)

[4 AMY—PODES-M0O应用实例 10](#_Toc375659939)

[4.1 AMY硬件结构框图 10](#_Toc375659940)

[4.2 AMY引脚列表 12](#_Toc375659941)

[4.3 AMY Memory Mapping 14](#_Toc375659942)

[4.4 AMY 中断资源 16](#_Toc375659943)

[4.5 AMY 外围模块 17](#_Toc375659944)

[4.5.1 通用串口UART 17](#_Toc375659945)

[4.5.2 GPIO 功能 21](#_Toc375659946)

[4.6 AMY 功能扩展 25](#_Toc375659947)

[5 AMY 代码仿真指南 29](#_Toc375659948)

[5.1 AMY代码结构 29](#_Toc375659949)

[5.2 AMY 仿真环境 30](#_Toc375659950)

[5.2.1 AMY 环境目录结构 30](#_Toc375659951)

[5.2.2 AMY testbench结构 30](#_Toc375659952)

[5.2.3 C程序 仿真脚本 32](#_Toc375659953)

[5.2.4 RTL 仿真脚本 35](#_Toc375659954)

[6 AMY 软件开发指南 39](#_Toc375659955)

[6.1 概述 39](#_Toc375659956)

[6.2 测试程序开发环境 40](#_Toc375659957)

[6.3 测试程序代码结构 42](#_Toc375659958)

[6.4 测试程序开发 44](#_Toc375659959)

[6.4.1 指定目标器件 44](#_Toc375659960)

[6.4.2 指定目标器件相关参数 45](#_Toc375659961)

[6.4.3 指定编译相关的参数 45](#_Toc375659962)

[6.4.4 加载内核配置文件 46](#_Toc375659963)

[6.4.5 其他参数配置 46](#_Toc375659964)

[6.4.6 输出测试文件 47](#_Toc375659965)

[7 AMY 评估板 49](#_Toc375659966)

[7.1 概述 49](#_Toc375659967)

[7.2 硬件接口功能 49](#_Toc375659968)

[7.3 软件调试和评估 50](#_Toc375659969)

[7.4 扩展应用 50](#_Toc375659970)

[8 AMY 评估用到的工具和环境 52](#_Toc375659971)

[8.1 Editor and Modelsim 52](#_Toc375659972)

[8.2 Cygwin 52](#_Toc375659973)

[8.3 Keil uVision 52](#_Toc375659974)

[8.4 ISE or QuartusII 53](#_Toc375659975)

[8.5 Synplify 53](#_Toc375659976)

[Change Summary 54](#_Toc375659977)

[CopyLeft 55](#_Toc375659978)



# 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。

# 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。

# 支持和服务

[www.mcucore.club](http://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](http://www.mcucore.club/)

所有的Issue Report或者优化建议，请投送到：[www.mcucore.club](http://www.mcucore.club/) 相关的页面，或者：[podes.mcu@qq.com](mailto: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：



# AMY—PODES-M0O应用实例

## 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。

## 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。

## 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. |

## 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 |
|  |  |

## 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。

### 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 |

### 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/W1C | 0 | Reserved |
| [7:0] | R/W1C | 0 | GPIO interrupt enable register  1: An interrupt occurs on the pin.  0: no interrupt. |

## 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 )    );  …… |

# AMY 代码仿真指南

## 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。



## AMY 仿真环境

### 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。



### 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。

1. 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。

### 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。

### 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.

# AMY 软件开发指南

## 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。

## 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。



## 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);  } |

## 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。

### 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> 。

### 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 |
|  |  |  |

### 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" |

### 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。

### 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。

### 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  **……** |

# 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。

## 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。

## 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。

## 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。

## 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。

# 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。

## 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。

## 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。

## 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。

## 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。

## 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。