CSI-RTOS Smart SDK(v1.7.1) User Guide

2021 年 03 月 22 日

平头哥半导体有限公司 T-HEAD Semiconductor Co.,LTD

地址: 杭州市余杭区向往街 1122 号欧美金融城 (EFC) 英国中心西楼 T6

邮编: 311121

网址: www.t-head.cn

版本历史

版本	日期	描述	作者
1.0	07/06/2017	1. 第一版	平头哥半导体有限公司
1.1	09/08/2018	1. 修改 CPU 子系统描述	平头哥半导体有限公司
1.2	11/07/2018	1. 增加 AXI 总线架构描述	平头哥半导体有限公司
1.3	12/29/2018	1. 增加 E902 产品描述	平头哥半导体有限公司
1.4	15/11/2019	1. 增加 E906 产品描述	平头哥半导体有限公司
1.6	10/11/2020	1. 增加 Smart SOC 相关信息	平头哥半导体有限公司
1.7	16/03/2021	1. 增加 E907 产品描述	平头哥半导体有限公司

CSI-RTOS Smart SDK(v1.7.1) User Guide

第一章　介绍

1.1　概述

CSI 是 Chip Software Interface 的简称，这是平头哥定义的一套嵌入式软件接口标准，该接口包含了 CPU 核、芯片驱动程序、操作系统内核、DSP 库、NN 库等部分的接口抽象。

CSI-RTOS Smart SDK 是基于 Smart（包含 SmartL 和 SmartH）平台的软件开发工具包，软件遵循 CSI 接口规范。通过该 SDK，用户可以快速基于 Smart 平台对平头哥 CPU 进行测试与评估，用户也可参考 SDK 中集成的各种常用组件以及示例程序进行应用开发，快速形成产品方案。

SDK 具体内容包括：

- CSI-Core：定义了 CPU 及与 CPU 紧耦合外设的标准接口规范以及实现。
- CSI-Driver：定义外围设备驱动的标准接口规范，以及相关外围接口在 SmartL 和 SmartH 芯片上的实现。
- CSI-Kernel：定义了实时操作系统标准接口规范，以及 Rhino、FreeRTOSv10.3.1、uCos-III 等实时操作系统的对接示例代码。
- Tests 代码: 提供基于 DTEST 测试框架的测试程序，可帮助客户进行 CSI 接口的兼容性与功能验证。
- Example 实例：提供了相关驱动、组件的使用示例代码。
- Benchmark 实例：提供了 CPU 性能测试程序，包括 Dhrystone、Coremark、Helix、Linpack、Whetstone 等基准测试程序。

1.2　缩写 / 术语

表 1.1: 缩写/术语

缩写 / 术语	定义
CSI	Chip Software Interface。平头哥定义的一套嵌入式软件接口标准，该接口包含了 CPU 核、芯片驱动程序、操作系统内核、DSP 指令等部分的接口抽象。
CSI-Core	针对 CPU 硬件的 CSI 接口
CSI-Kernel	针对嵌入式实时操作系统的 CSI 接口
CSI-Driver	针对外围驱动的 CSI 接口
CDS	平头哥开发的基于 Eclipse 的 IDE
CDK	平头哥开发的适用于物联网和 MCU 领域的 IDE
Rhino	AliOS 内核

第二章　硬件简介

2.1　Smartl 平台概述

SmartL 平台是用于 E801/E802/E803S/E804/E805/E902/E906/E907 集成、调试仿真，以及 FPGA 应用评估的综合演示平台。

2.2　Smartl SoC 架构

2.2.1　AHB 总线架构

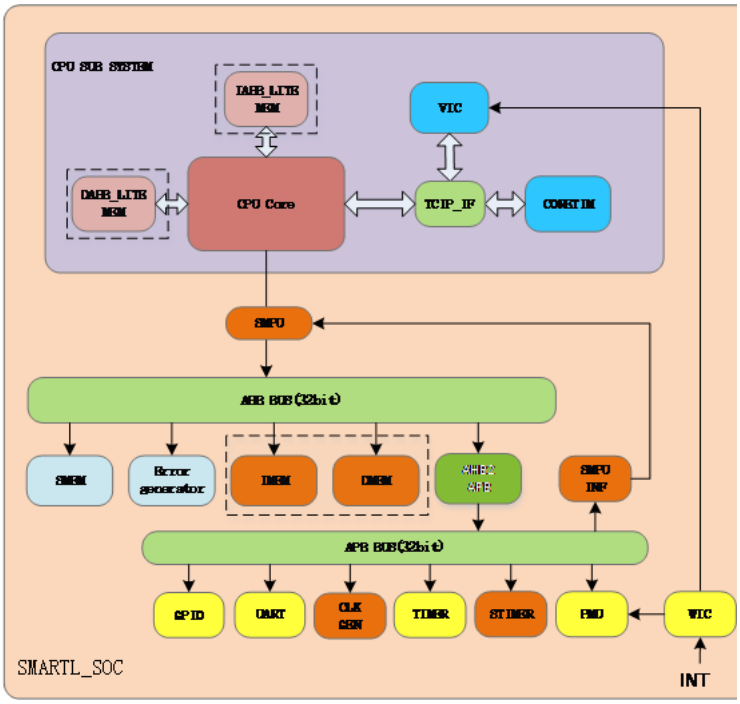

图 2.1: SmartL AHB 总线平台架构

虚线框内的模块由 CPU 配置决定是否存在，当 CPU 没有配置 DLITE 时，DMEM 会被放到 AHB 总线上，ILITE 也一样，对应 IMEM。

SMPU、SMPUINF、STIMER 几个模块是安全演示相关的，如果客户申请的 CPU 不包含安全配置选项 TEE，可以无视这些模块，这些模块也不会起作用。

2.2.2　AXI 总线架构

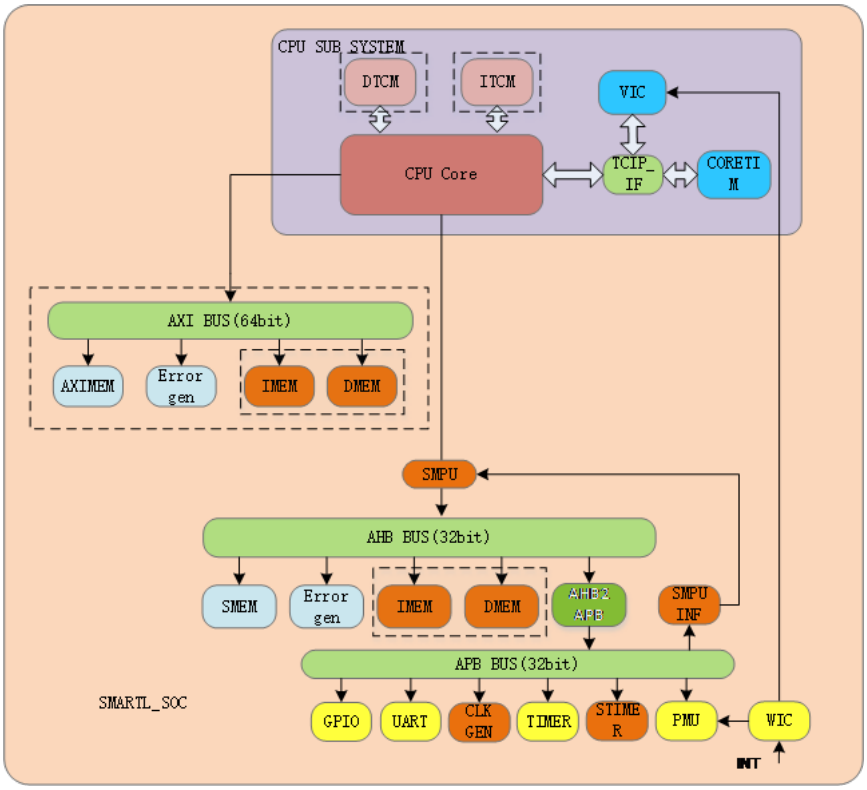

图 2.2: SmartL AXI 总线平台架构

AXI 总线架构是 E805 集成所用，虚线框内的模块由 CPU 配置决定是否存在。当 CPU 没有配置 DTCM 时，DTCM 会根据是否配置 AXI 总线被放到 AHB 或 AXI 总线，如果配置存在 AXI 总线时，DTCM 被放置在 AXI 总线上，否则在 AHB 总线上，对应 DMEM；ILITE 也一样，对应 IMEM。

SMPU、SMPUINF、STIMER 几个模块是安全演示相关的，如果客户申请的 CPU 不包含安全配置选项 TEE，可以无视这些模块，这些模块也不会起作用。

2.2.3　地址空间

地址空间分配如 表 2.1 所示:

表 2.1: 系统地址分配

Address	Mapping IP
0x00000000-0x1FFFFFFF	本地指令存储器可用地址空间
0x20000000-0x3FFFFFFF	本地数据存储器可用地址空间
0x40000000-0x4FFFFFFF	APB 总线地址空间
0x50000000-0x507FFFFF	片外存储可用地址空间
0x60000000-0x7FFFFFFF	系统内存可用地址空间
0x80000000-0x8FFFFFFF	系统内存可用地址空间 (AXI 总线架构存在)
0xE0000000-0xEFFFFFFF	紧耦合 IP 空间（TCIP）
其它	保留，读写返回错误

玄铁 800 系列各 IP 的地址映射如 表 2.2 所示:

表 2.2: 玄铁 800 系列 IP 地址映射

Base Address	IP
0x40011000	TIMER (APB Slave)
0x40015000	UART (APB Slave)
0x40016000	PMU (APB Slave)
0x40017000	CLK_GEN(APB Slave)
0x40018000	STIMER(APB Slave)
0x40019000	GPIO(APB Slave)
0x4001A000	SMPU(APB Slave)
0xE000E010	CORETIM (TCIP)
0xE000E100	VIC /CLIC(TCIP)

玄铁 900 系列各 IP 的地址映射如 表 2.3 所示:

表 2.3: 玄铁 900 系列 IP 地址映射

Base Address	IP
0x40011000	TIMER (APB Slave)
0x40015000	UART (APB Slave)
0x40016000	PMU (APB Slave)
0x40017000	CLK_GEN(APB Slave)
0x40018000	STIMER(APB Slave)
0x40019000	GPIO(APB Slave)
0x4001A000	SMPU(APB Slave)
0xE0000000	MTIME(CLINT)
0xE0800000	CLIC

2.2.4 中断号分配

玄铁 800 系列各 IP 的中断向量号如 表 2.4 所示:

表 2.4: 中断向量号

中断号	属性	中断源
0x20	非安全	UART (APB Slave)
0x21	非安全	CORETIM (TCIP)
0x22-0x25	非安全	TIMER (APB Slave)
0x26	保留	保留
0x27-0x2e	非安全	GPIO(APB Slave)
0x30-0x33	安全	STIMER (APB Slave)
0x34	非安全	Pad 直接输入，演示脉冲中断

玄铁 900 系列各 IP 的中断向量号如 表 2.5 所示：

表 2.5: 中断向量号

中断号	属性	中断源
0x10	非安全	UART (APB Slave)
0x11	非安全	CORETIM (TCIP)
0x12-0x15	非安全	TIMER (APB Slave)
0x16	保留	保留
0x17-0x1e	非安全	GPIO(APB Slave)
0x20-0x23	安全	STIMER (APB Slave)
0x24	非安全	Pad 直接输入，演示脉冲中断

如果客户申请的 CPU 不包含安全配置选项 TEE，请无视安全中断。

2.3 SmartH 概述

SMARTH 平台是用于 C610/C807/C810/C860 集成、调试仿真，以及 FPGA 应用评估的综合演示平台。

2.4 SmartH SOC 架构

由于 THEAD CPU 有灵活的总线配置，SMARTH 也根据 CPU 不同的总线做了适配，分别对应单总线 AHB，单总线 AXI 和双总线 AXI+AHB。

2.4.1 单 AHB 总线架构

地址空间分配如 表 2.6 所示：

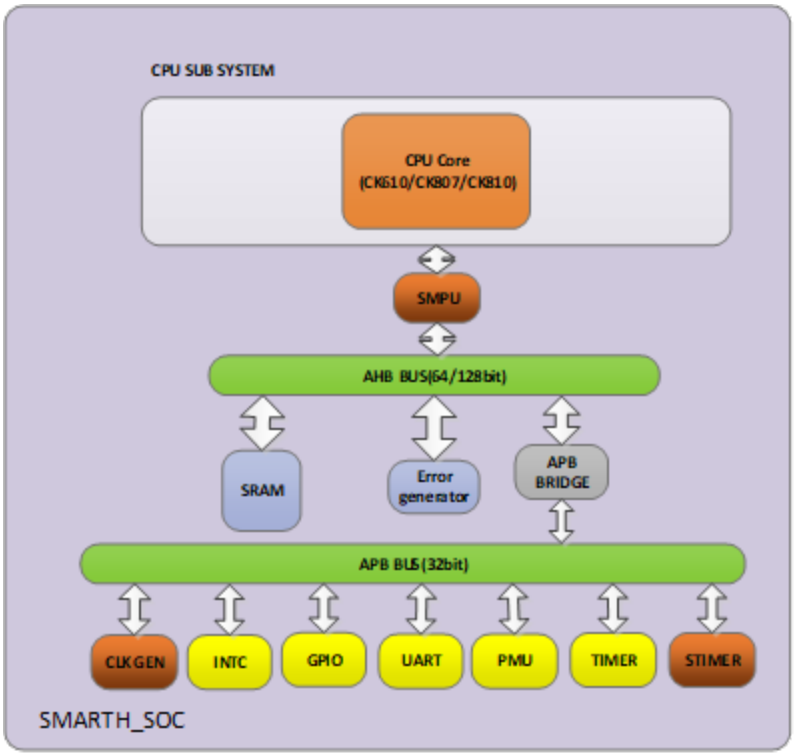

图 2.3: SMARTH AHB 协议平台架构

表 2.6: AHB 总线配置下地址分配

Address	Mapping IP
0x00000000-0x0001FFFF	指令存储器（SRAM，目前使用 128KB）
0x10000000-0x1FFFFFFF	APB 总线
其它	保留，读写返回错误

2.4.2　单 AXI 总线架构

其中地址空间分配如 表 2.7 所示：

表 2.7: AXI 总线配置下地址分配

Address	Mapping IP
0x00000000-0x0001FFFF	指令存储器（SRAM，目前使用 128KB）
0x10000000-0x1F0000000	APB 总线
0x1F000000-0x1F01FFFF	AHB 下的 mem 地址空间（SRAM，目前使用 128KB，可配置可信地址空间）
其它	保留，读写返回错误

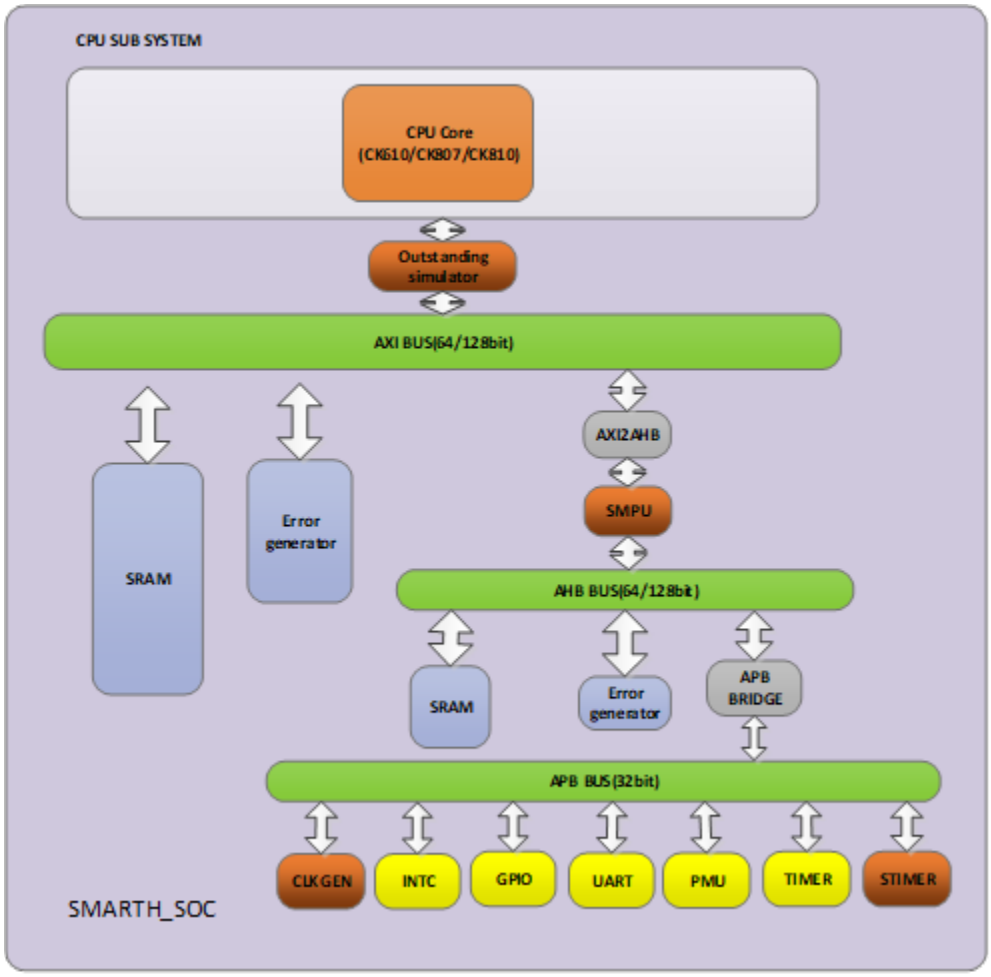

图 2.4: SMARTH AXI 协议平台架构

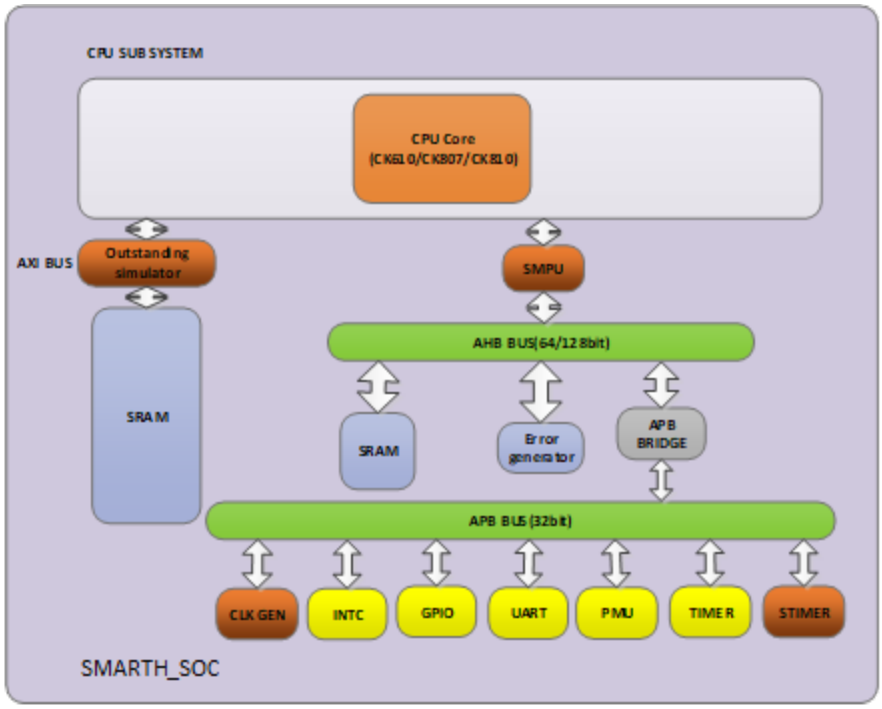

图 2.5: SMARTH 双总线协议架构

2.4.3　双总线架构

其中地址空间分配如 表 2.8 所示：

表 2.8: 双总线配置下地址分配

Address	Mapping IP
0x00000000-0x0001FFFF	指令存储器（SRAM，目前使用 128KB）
0x10000000-0x1F0000000	APB 总线
0x1F000000-0x1F01FFFF	AHB 下的 mem 地址空间（SRAM，目前使用 128KB，可配置可信地址空间）
0 x10000000-0x1FFFFFFF 中的其他空间	保留，读写返回错误
其他地址空间	未配置，行为不可知

2.4.4　IP 地址分配

为了写程序方便，不同 SOC 架构下，各 IP 的地址映射都是一样的，如 表 2.9 所示：

表 2.9: IP 地址映射

Base Address	IP
0x10010000	INTC(APB Slave)
0x10011000	TIMER (APB Slave)
0x10015000	UART (APB Slave)
0x10016000	PMU (APB Slave)
0x10017000	CLK GEN(APB Slave)
0x10018000	STIMER(APB Slave)
0x10019000	GPIO(APB Slave)
0x1001a000	SMPU(APB Slave)

2.4.5 中断号分配

为了写程序方便，不同 SOC 架构下，中断号分配也是一样的，如 表 2.10 所示：

表 2.10: 中断向量号

中断号	属性	中断源
0x20	非安全	UART (APB Slave)
0x21	无	无
0x22-0x25	非安全	TIMER (APB Slave)
0x26	非安全	PMU (APB Slave)，
0x27-0x2e	非安全	GPIO (APB Slave)
0x30-0x33	安全	STIMER (APB Slave)
0x34	非安全	Pad 直接输入，演示脉冲中断

2.5 CPU 子系统

在 SmartL 平台中，所有配置下的 THEAD CPU 都会构成一个独立的 AHB 或 AXI 子系统。在这个子系统中，CPU 的绝大部分外围信号将在子系统这个层次上被绑定成固定的值，AHB 和 AXI 接口信号会释放到系统上面和系统中的总线接口连接，调试引脚（JTAG）会连接到系统的 JTAG 引脚上面。这里特别要注意，对于 E802/E803S/E804/E805/E902/E906/E907，本地指令、数据存储器则为可配置选项。子系统集成了紧耦合 IP，包括计数器 CORETIM 和中断控制器 VIC。这些经过封装后的 THEAD CPU 从系统的角度看都是统一的，如下图所示：

在 CPU 配置了本地数据和指令存储器之后，SmartL RTL 仿真平台默认使用本地存储器存储程序指令或数据，否则使用 AHB 或 AXI 总线对应地址上的存储器。

2.5.1 AHB BUS

在 SmartL 平台中，设计有一个简易的 AHB 总线仲裁模块，该模块支持 32 位总线宽度。该模块仅用于 SmartL 平台中做简单的功能演示。AHB 按照地址空间对从 CPU 发送过来的请求进行仲裁，将其发送到对应的从设备（Slave）。

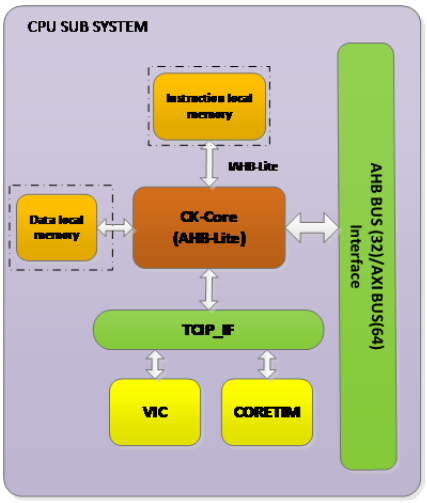

图 2.6: CPU 子系统结构图

在 SmartL 系统中，AHB 总线上挂了一个主设备（Master），为 CKCPU；默认情况下挂了 3 个从设备，分别是系统静态随机访问存储器（SRAM）、APB 桥和错误产生器（用于返回总线上的错误情况）。当 CPU 没有定义 IAHB/DAHB_LITE 时会在 AHB 总线上相对应的指令/数据空间上例化额外的系统存储器（SRAM）以实现指令、数据的正常访问，如没有定义本地指令以及数据存储器则总线上需要例化两块新的系统存储器，SLAVE 数增加到 5 个。

系统内存是直接物理实现的 SRAM，用于存放数据，栈相关内容以及相关的全局变量等内容。该设备的地址空间为 0x60000000-0x7FFFFFFF。该存储器允许读写访问，考虑到 FPGA 成本约束和使用需求，物理上实际实现低 128K。当没有配置 IAHB_LITE 时在地址空间 0x00000000 - 0x1FFFFFFF 上增加新的存储器，物理上实际实现低 128K；当没有配置 DAHB_LITE 时在地址空间 0x20000000 -0x3FFFFFFF 上增加新的存储器，物理上实际实现低 128K。

APB 桥下实现了 APB 低速子系统，目前设计有七个 APB 设备，分别是 UART、Timer、Stimer、CLK Generator、System MPU、Power Management Unit（PMU）、GPIO。

错误产生器是为了构建总线错误访问行为设计的模块，当 CPU 访问到该外设时，它将产生一个错误的请求。该模块的地址空间为所有保留区域。

2.5.2 AXI BUS

在 SmartL 平台中，设计有一个简易的 AXI 总线仲裁模块，该模块支持 64 位总线宽度。该模块仅用于 SmartL 平台中做简单的功能演示。AXI 按照地址空间对从 CPU 发送过来的请求进行仲裁，将其发送到对应的从设备（Slave）去。在 SMARTL 系统中，AXI 总线上挂了一个主设备（Master），为 THEAD CPU 挂了 4 个从设备，分别是系统指令存储器（IMEM）、系统数据存储器 (DMEM)、系统随机动态存储器 (SRAM) 和错误产生器（用于返回总线上的错误情况）。

系统内存是直接物理实现的 SRAM，用于存放指令，数据，栈相关内容以及相关的全局变量等内容。其中指令和数据存储器在 CPU 配置存在 AXI 总线且没有相应的 TCM 模块时存在。其中系统指令地址空间 0x00000000 - 0x1FFFFFFF，物理上实际实现低 128K；系统数据地址空间 0x20000000 -0x3FFFFFFF，物理上实际实现低 128K，系统随机动态存储器地址空间 0x80000000 -0x8FFFFFFF，物理上实际实现低 128K。

2.6　APB 外设

2.6.1　UART

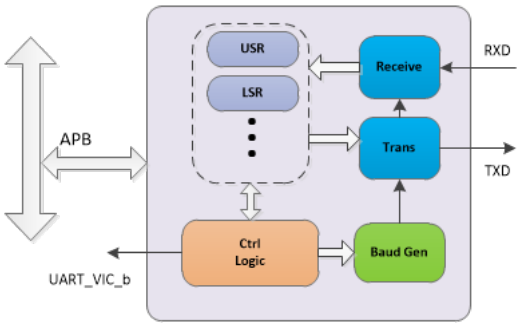

图 2.7: UART 结构图

UART 是一种通用串行数据总线，用于异步通信。该总线支持双向通信，可以实现全双工传输和接收。

UART 的功能：

- APB 总线接口；
- 全双工独立的发送和接收通道；
- 软件检测通道状态；
- 可配置奇偶校验中断、覆盖有效数据中断、帧错误中断产生；
- 可配置传输波特率；
- 最大传输速率为系统时钟的 1/16

UART 通过串行传输总线 TXD 传送数据，RXD 接收数据，数据流格式如下图所示：

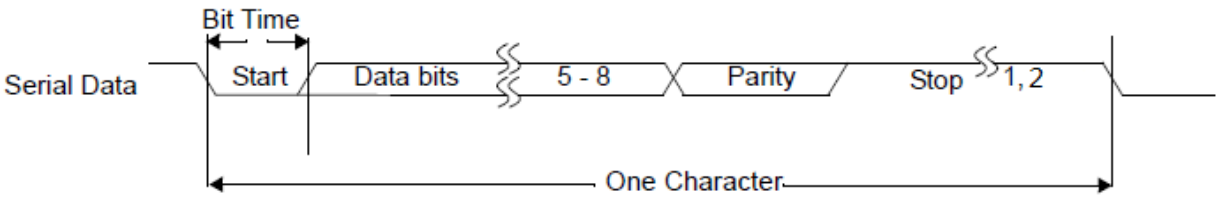

图 2.8: UART 数据流格式

每帧数据有起始位、5 到 8 位的数据位和可选这得奇偶校验位以及 1~2 位的停止位，其中起始位为低电平，停止位为高电平。

数据传输通过波特率同步，软件首先确定波特率，UART 自动产生波特率时钟，在接收总线监测到低电平起始位后，每 16 个时钟周期采样一次总线，从而接收数据。采样如下图所示：

UART 波特率产生

UART 通过软件设置 DLL 和 DLH（分频时钟寄存器）来产生相应的波特率时钟分频时钟寄存器的计算公式为：系统时钟/(波特率 x16)。DLL 寄存器存储分频时钟寄存器的地位值，DLH 存储分频时钟的高位值，设置 DLL、DLH 值之前需要将 LCR 传输控制器的 DLAB 位置 1，寄存器的具体描述见下文。

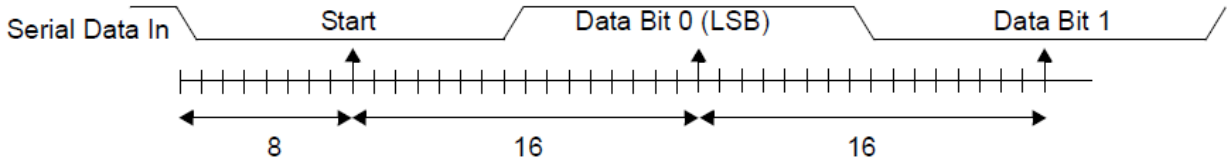

图 2.9: UART 采样时序

UART 中断时序

UART 支持覆盖有效数据、奇偶校验错误、帧错误的传输状态中断，也支持接受数据有效中断和传输保持寄存器空中断，以及支持 UART 忙中断。软件编程者可以通过写值到 IER 寄存器中使能这些中断。具体的中断描述和产生见下文的寄存器描述表格中。

下图位一次帧错误中断的产生，如图所示：数据位的长度为 5 位，停止位的长度为 2 位，在第二个停止位时检测到了低电平，所以产生帧错误中断，此中断在软件读 LSR 寄存器后清掉。

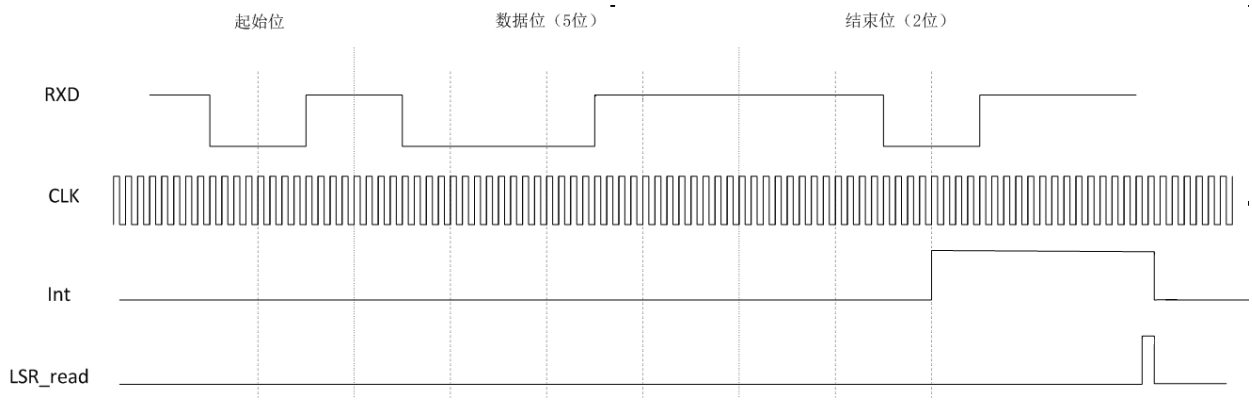

图 2.10: UART 中断时序

注意： 所有接收的中断，即接收状态中断和接收数据有效中断都在完整接收完一帧数据后产生，而传输保持寄存器空中断在传输保持寄存器空时就立即产生。

UART 的寄存器描述

表 2.11: UART 寄存器

地址	名称	读写	复位值	描述
0x00	RBR THR DLL	R W R/W	0x0 0x0 0x0	接收缓冲寄存器在 LCR[7] = 0 时有效 传输保持寄存器在 LCR[7] = 0 时有效 分频时钟寄存器低在 LCR[7] = 1 时有效
0x04	DLH IER	R/W R/W	0x0 0x0	分频时钟寄存器高在 LCR[7] = 1 时有效 中断使能寄存器在 LCR[7] = 0 时有效
0x08	IIR	R	0x1	中断标号寄存器
0x0c	LCR	R/W	0x0	传输控制寄存器
0x14	LSR	R	0x60	传输状态寄存器
0x7c	USR	R	0x0	UART 状态寄存器

RBR —— 接收缓冲寄存器

RBR 是接收缓冲寄存器，从 RXD 端口接收数据。寄存器低 8 位有效。当传输状态寄存器的 DR 位有效时，数据有效。数据必须在下一个数据接收到之前被读走，否则会出现覆盖有效数据中断。

THR —— 传输保持寄存器

THR 是传输保持寄存器，数据从 TXD 端口传输到接收端，数据的低 8 位有效。数据只能在传输状态寄存器的 THRE 为高时写到 THR 中，否则，有效数据会被覆盖

DLL —— 分频时钟寄存器低

DLL 是分频时钟寄存器的低 8 位。分频时钟寄存器控制传输波特率，寄存器只能在传输控制寄存器的 DLAB 位为高并且 UART 状态寄存器的 busy 位为 0 时写。当 DLH 和 DLL 都设为 0 时，传输不会使能。

DLH —— 分频时钟寄存器高

DLH 是分频时钟寄存器的低 8 位。分频时钟寄存器控制传输波特率，寄存器只能在传输控制寄存器的 DLAB 位为高并且 UART 状态寄存器的 busy 位为 0 时写。当 DLH 和 DLL 都设为 0 时，传输不会使能。

IER —— 中断使能寄存器

IER 是中断使能寄存器。使能传输和接收所产生的中断。它的定义如 表 2.12 所示：

表 2.12: UART 中断使能寄存器

位	名称	读写	描述
31:8			保留
7:3			保留
2	ELSI	R/W	使能接收状态中断
1	ETBEI	R/W	使能传输保持寄存器空中断
0	ERBFI	R/W	使能接收数据有效中断

IIR ——中断标号寄存器

中断标号寄存器指示出当前最高优先级中断，寄存器定义和中断优先级如 表 2.13 所示：

表 2.13: UART 中断使能寄存器

位	名称	读写	描述
31:8			保留
7:4			保留
3:0	IID	R	中断标号 0001 = 无中断 0010 = THR 空中断 0100 = 接收数据有效中断 0110 = 接收状态中断 0111 = uart 忙

表 2.14: UART 中断优先级

中断优先级分为 4 级，详细信息如下表所示：中断标号	优先级	中断类型	中断源	中断复位控制
0001		无中断		
0110	最高	接收状态中断	覆盖有效数据 奇偶校验错误 帧错误	读 LSR 寄存器
0100	第二	接收数据有效中断	接受数据有效	读 RSR 寄存器
0010	第三	THR 空中断	THR 空	读 IIR 寄存器或写数据到 THR 中
0111	第四	uart 忙	当 uart 忙时，写数据到 LCR 中	读 USR 寄存器

LCR —— 传输控制寄存器

传输控制寄存器控制 UART 的传输和接收，主要控制传输的数据位长度和是否奇偶校验等，详细定义如 表 2.15 所示：

表 2.15: UART 传输控制寄存器

位	名称	读写	描述
31:8			保留
7	DLAB	R/W	分频时钟寄存器访问控制位。分频时钟寄存器一直可读但只有在 uart 不忙时写。此位应当在初始化了分频时钟寄存器后复位，这样可以访问其他寄存器
6：5			保留
4	EPS	R/W	奇偶校验选择。当 uart 不忙时写。 1—— 偶校验 0—— 奇校验
3	PEN	R/W	奇偶检验使能位
2	STOP	R/W	停止位个数 0 —- 1 个停止位 1 —- 2 个停止位
1:0	DLS	R/W	数据长度选择 00 = 5 位 01 = 6 位 10 = 7 位 11 = 8 位

LSR —— 传输状态寄存器

传输状态寄存器，标识传送和接收的状态。详细的定义如 表 2.16 所示：

表 2.16: UART 传输状态寄存器

位	名称	读写	描述
31:8			保留
7			保留
6	TEMT	R	传送空标识。当传输保持寄存器空并且传输移位寄存器也为空时置高
5	THRE	R	传送保持寄存器空标识。当传送保持寄存器为空时置高
4			保留
3	FE	R	帧错误位 0 —- 没有帧错误 1 —- 帧错误 读 LSR 寄存器将复位 FE 位
2	PE	R	奇偶校验错误位。当 LC R 的 PEN 位置高，此位将在发生奇偶校验错误时置高 0 —- 没有奇偶校验错误 1 —- 奇偶校验错误 读 LSR 寄存器将复位 PE 位
1	OE	R	覆盖有效数据错误 0 —- 无错误 1 —- 覆盖有效数据错误 读 LSR 寄存器将复位 OE 位
0	DR	R	接收数据有效位 0 —- 没有接收到有效数据 1 —- 接收数据有效 读 RBR 此位复位

UART 状态寄存器标识 UART 是否在接收或是发送数据。

表 2.17: UART 状态寄存器

位	名称	读写	描述
31:8			保留
7:1			保留
0	BUSY	R	UART 忙 0 —- UART 在空状态 1 —- UART 忙（接收或传送数据）

2.6.2 TIMER

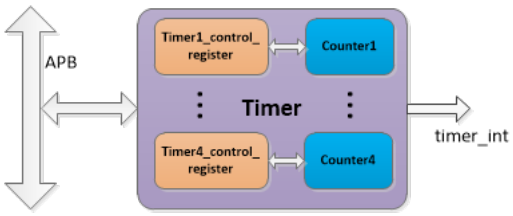

图 2.11: TIMER 结构图

TIMER 是一个用户可编程的计时设备，具有 APB 总线接口。它内部包含 4 个独立可编程的 32 位计数器，计数器可以从一个软件写入的值开始向下计数，计数到 0 时产生独立的中断。所以 TIMER 具有 4 个中断源。每次计数到 0，计数器都会从相应的回填值寄存器读取计时初始值。

TIMER 的功能特点如下：

- 4 个独立可编程计数器
- 32 位计时位宽
- 计时时间可编程
- 支持两种运行模式：free-running 模式和 user-defined count 模式

TIMER 计时器寄存器地址：（TIMER 基址为 0x40011000）

表 2.18: TIMER 计时器寄存器地址

Address Range (offset)	Function
0x00 to 0x10	Timer1 Registers
0x14 to 0x24	Timer2 Registers
0x28 to 0x38	Timer3 Registers
0x3c to 0x4c	Timer4 Registers
0xa0 to 0xa8	Timer System Registers

下面以 TIMER 中的 Timer1 为例进行寄存器描述，Timer2~4 与其相同。

表 2.19: Timer1 寄存器描述

名称	地址偏移量	位宽	R/W	复位值	描述
Timer1LoadCount	0x00	32	R/W	32’b0	Timer1 回填值寄存器
Timer1CurrentValue	0x04	32	R	32’b0	Timer1 当前值寄存器
Timer1Control Reg	0x08	32	R/W	3’b0	Timer1 控制寄存器
Timer1EOI	0x0C	32	R	1’b0	Timer1 中断清除寄存器
Timer1IntStatus	0x10	32	R	1’b0	Timer1 中断状态寄存器
TimersInt Status	0xa0	32	R	32b’0	TIMER 中断状态寄存器
TimersEOI	0xa4	32	R	32’b0	TIMER 中断清除寄存器
TimersRaw IntStatus	0xa8	32	R	32’b0	TIMER 原始中断状态寄存器

Timer1LoadCount

表 2.20: Timer1 回填值寄存器

位	名称	R/W	描述
31:0	Timer1 回填值寄存器	R/W	存储 Timer1 的回填值，在每一个计数循环开始时该寄存器的值会赋给 Timer1 的计数器。

Timer1CurrentValue

表 2.21: Timer1 当前值寄存器

位	名称	R/W	描述
31:0	Timer1 Current Value Register	R	存储 Timer1 的当前计数值。

Timer1ControlReg

表 2.22: Timer1 控制寄存器

位	名称	R/W	描述
31:3	保留，读为 0		
2	计时中断屏蔽	R/W	0：计时中断没有被屏蔽 1：计时中断被屏蔽
1	计时模式选择	R/W	0 ： free-running 模式: 回填值为 32’ bFFFFFFFF 1 ： user-defined running 模式: 回填值从回填值寄存器读取
0	计时使能	R/W	0 ： 不使能 1 ： 使能

Timer1EOI

表 2.23: Timer1 中断清除寄存器

位	名称	R/W	描述
31:1	保留，读为 0		
0	Timer1 中断清除寄存器	R	读操作将清除 Timer1 中断

Timer1IntStatus

表 2.24: Timer1 中断状态寄存器

位	名称	R/W	描述
31:1	保留，读为 0		
0	Timer1 中断状态寄存器	R	指示 Timer1 中断状态

TimersIntStatus

表 2.25: TIMER 中断状态寄存器

位	名称	R/W	描述
31:4	保留，读为 0		
3:0	TIMER 中断状态寄存器	R	相应比特指示各计时器的中断状态，读操作不会清除有效中断。 0：中断无效 1：中断有效

TimersEOI

表 2.26: TIMER 中断状态寄存器

位	名称	R/W	描述
31:4	保留，读为 0		
3:0	TIMER 中断清除寄存器	R	读操作返回 0 并清除所有有效中断。

TimersRawIntStatus

2.6.3 STIMER

STIMER 的基地址 0x400018000, 其内部结构、功能特点与控制方法和 TIMER 完全相同。

所生成的中断源需在中断控制器的对应位置根据需要配置为安全中断，以实现安全中断的产生。如需要实现安全中断的功能，应将 STIMER 所在地址写入 SPMU 的 entry 中，实现对该安全模块配置信息进行保护。同时把该中断在 VIC 内设置成安全中断，这个 IP 就变成了安全 IP。

2.6.4 GPIO

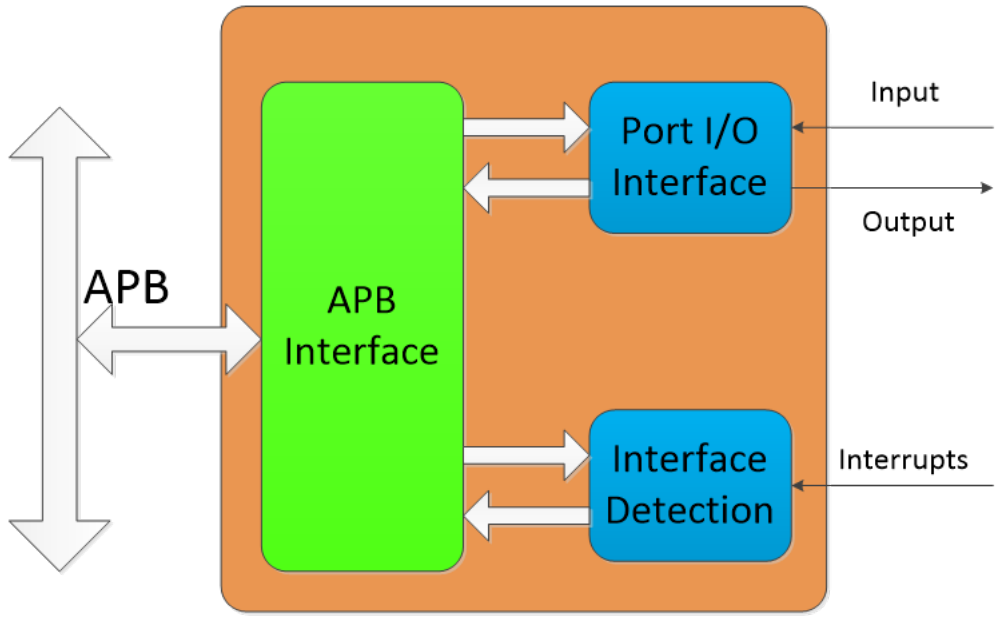

图 2.12: GPIO 结构图

General Purpose Input Output(通用输入/输出)，简称 GPIO，也称为总线扩展器。

GPIO 的功能：

- APB 总线接口
- 8 个独立可配置引脚的 I/O 端口；
- 每个 I/O 端口有独立的数据寄存器和数据方向寄存器；

- A 端口有其可配置的中断模式；

GPIO 寄存器地址：（GPIO 基址为 0x40019000）

表 2.27: GPIO 寄存器地址

Address Range (offset)	Function
0x00 to 0x08,0x50	PortA I/O Registers
0x30 to 0x44,0x4c,0x60	PortA Interrupts Registers

下表 表 2.28 以 PortA 的 I/O 寄存器和 Interrupts 寄存器进行描述。

表 2.28: Port A I/O 寄存器和 Interrupts 寄存器描述

名称	地址偏移量	位宽	R/W	复位值	描述
Port A Data Register	0x00	32	R/W	32'b0	Port A 数据寄存器
Port A Data Direction Register	0x04	32	R/W	32'b0	Port A 数据方向寄存器
Port A Data Source	0x08	32	R	32'b0	Port A 源数据寄存器
External Port A	0x50	32	R	32'b0	Port A 外部数据寄存器
Interrupt enable	0x30	32	R/W	32'b0	中断使能寄存器
Interrupt mask	0x34	32	R/W	32'b0	中断屏蔽寄存器
Interrupt level	0x38	32	R/W	32'b0	中断类型寄存器
Interrupt polarity	0x3c	32	R/W	32'b0	中断极性寄存器
Interrupt status	0x40	32	R	32'b0	中断状态寄存器
Raw Interrupt status	0x44	32	R	32'b0	未屏蔽中断状态寄存器
Clear interrupt	0x4c	32	W	32'b0	中断清除寄存器
Synchronization level	0x60	32	R/W	32'b0	中断同步寄存器

Port A Data Register

表 2.29: Port A 数据寄存器

位	名称	R/W	描述
31:8	保留，读为 0		
7:0	Port A 数据寄存器	R/W	在端口 A 为输出状态时，写该寄存器即为端口 A 的输出值。 读该寄存器所得到的值即为最后一个更新这个寄存器的值。

Port A Data Direction Register

表 2.30: Port A 数据方向寄存器

位	名称	R/W	描述
31:8	保留，读为 0		
7:0	Port A 数据方向寄存器	R/W	该寄存器低 16 位中的每一位都控制端口 A 的每个引脚为输入或者输出。 0：输入 1：输出

Port A Data Source

表 2.31: Port A 源数据寄存器

位	名称	R/W	描述
31:8	保留，读为 0		
7:0	Port A 源数据寄存器	R	该寄存器为 0 值，即端口 A 的数据均来自软件模式。

External Port A

表 2.32: Port A 外部数据寄存器

位	名称	R/W	描述
31:8	保留，读为 0		
7:0	Port A 外部数据寄存器	R	在端口 A 为输入状态时，读这个地址得到的值即为外部输入的值；在端口 A 为输出状态时，读这个地址得到的值为 Port A 数据寄存器的值。

Interrupt enable

表 2.33: 中断使能寄存器

位	名称	R/W	描述
31:8	保留，读为0		
7:0	中断使能寄存器	R/W	该寄存器中低 8 位的每一位分别控制 Port A 的每个引脚的中断使能。 0：禁能中断 1：使能中断

Interrupt mask

表 2.34: 中断屏蔽寄存器

位	名称	R/W	描述
31:8	保留，读为0		
7:0	中断屏蔽寄存器	R/W	该寄存器中低 8 位的每一位分别对 Port A 的每个引脚中断进行屏蔽。 0：不屏蔽中断 1：屏蔽中断

Interrupt level

表 2.35: 中断类型寄存器

位	名称	R/W	描述
31:8	保留，读为0		
7:0	中断类型寄存器	R/W	该寄存器中低 8 位的每一位分别对 Port A 的每个引脚中断的类型进行设置 0：电平触发中断 1：边沿触发中断

Interrupt polarity

表 2.36: 中断极性寄存器

位	名称	R/W	描述
31:8	保留，读为0		
7:0	中断极性寄存器	R/W	该寄存器中低 8 位的每一位分别对 Port A 的每个引脚中断的极性进行设置 0 ： 低电平或下降沿触发中断 1 ： 高电平或上升沿触发中断

Interrupt status

表 2.37: 中断状态寄存器

位	名称	R/W	描述
31:8	保留，读为0		
7:0	中断状态寄存器	R	该寄存器中低 8 位的每一位分别描述了 Port A 每个引脚的已屏蔽中断状态。 0：无屏蔽中断 1：有屏蔽中断

Raw interrupt status

表 2.38: 未屏蔽中断状态寄存器

位	名称	R/W	描述
31:8	保留，读为0		
7:0	未屏蔽中断状态寄存器	R	该寄存器中低 8 位的每一位分别描述了 Port A 每个引脚的未屏蔽中断状态。 0：无未屏蔽中断 1：有未屏蔽中断

Clear interrupt

表 2.39: 中断清除寄存器

位	名称	R/W	描述
31:8	保留，读为0		
7:0	中断清除寄存器	W	该寄存器中低 8 位的每一位分别清除 Port A 每个引脚的边沿触发中断。 0：不清除中断 1：清除中断

Synchronization level

表 2.40: 中断同步寄存器

位	名称	R/W	描述
31:8	保留，读为0		
7:0	中断同步寄存器	R/W	该寄存器中低 8 位的每一位分别控制 Port A 每个引脚的电平触发中断是否同步于 pclk_intr。 0：不同步 1：同步

2.6.5 Clock Gen

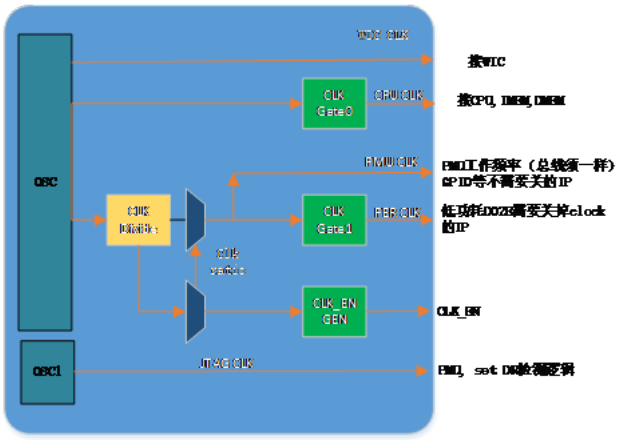

图 2.13: clock gen 结构图

Clock Gen 是一个生成并控制时钟信号的模块。主要由 Clock divider 和 Clock aligner 两部分组成。其中 Clock Divider 输入为 forever cpuclk，生成 clk ratio 0~7 的所有时钟及相应的 clk_en 信号。Clock Aligner 输入为 Divider 生成的所有时钟信号，根据配置寄存器信息进行选择，并实现动态变频的控制。同时 PMU 单元产生的时钟控制信号也会输入该单元实现对特定时钟信号的控制。

Clock Gen 的功能特点如下：

- 1 个配置寄存器，实现 clock ratio 0 -7 的动态变频
- 变频后产生对应的 clk en 信号
- 根据低功耗场景控制响应的时钟信号

产生的时钟包括：

表 2.41: 生成时钟列表

Clock 名称	应用模块	分频情况	Clock gating
pmu_clk	GPIO TIMER STIMER PMU	分频	不关闭
per_clk	系统总线及系统总线上的其他 IP	分频	Doze Stop 模式下关闭
cpu_clk	CPU 及 MEM	不分频	所有低功耗模式下关闭
wic_clk	WIC	不分频	不关闭

Clock Gen 寄存器地址：

表 2.42: clock gen 寄存器地址

Address	Function
0x40017000	时钟频率控制

寄存器设置位描述：

表 2.43: clock gen 寄存器描述

位	名称	R/W	描述
31:3	保留，读为 0		
2:0	时钟频率	R/W	支持 clk ratio 0-7 的配置

2.6.6 System MPU (SMPU)

SMPU 是一个用于保护特定安全区域的模块。具有三个可配置的表项，可分别对应保护三段不同的地址空间。如配置有 TEE，非可信世界下通过系统总线访问被保护的区间将会返回访问错误。

SMPU 的功能特点如下：

- 实现系统总线上安全地址区间的保护；
- 三个保护区控制寄存器可配置；

SMPU 保护区控制寄存器地址：（SMPU 基址为 0x4001A000）

表 2.44: SMPU 寄存器地址

Address Range (offset)	Function
0x0	Entry1 Registers
0x4	Entry2 Registers
0x8	Entry3 Registers

在 SmartL 使用时三个 entry 可分别用于保护系统存储器上的自定义空间，SMPU 本身的地址空间以及 STIMER 的地址空间。保护空间大小 256B 到 1KB。

下面以 Entry1 为例进行寄存器描述，Entry2,3 的结构与功能与 1 完全相同。

表 2.45: SMPU 寄存器描述

位	名称	R/W	描述
31:9	Base address	R/W	保护区基地址
8:5	保留，读为 0		
4:1	Size	R/W	保护区大小
0	E	R/W	使能位

Base Address-保护区地址的基地址：

该寄存器指出了保护区地址的基地址，但写入的基地址必须与设置的页面大小对齐

例如设置页面大小为 1K，SPACR[10:9] 必须为 0，各页面的具体要求见图表 2-45 保护区大小及其基地址要求。

Size-保护区大小：

保护区大小从 256B 到 1KB，它可以通过公式：保护区大小= 2^（Size+1）计 得到。Size 可设置的范围为 0111 到 1001，其它一些值都会造成不可预测的结果。

表 2.46: 保护区大小及其基地址要求

Size	保护区大小	对基地址的要求
0000—0110	保留	-
0111	256B	没有要求
1000	512B	Bit[9]=0
1001	1KB	Bit[10:9]=0

E-保护区有效设置：

当 E 为 0 时，保护区无效；

当 E 为 1 时，保护区有效。

2.6.7 PMU

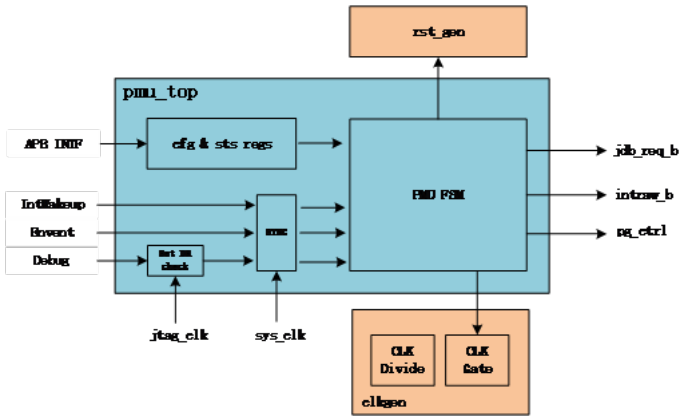

图 2.14: PMU 结构图

PMU 为功耗管理单元，负责系统在各个低功耗场景下的时钟、电源及唤醒机制的管理。在任何情况下该模块均不断电，可在任何低功耗场景下采样唤醒源进行低功耗唤醒。有一个可配置的控制寄存器，对唤醒源是否有效进行控制。Set DR 模块内部实现了两线的 TAP 状态机，用于检测 SoC 顶层输入的 tms 信号时序。当检测到 JTAG 的 tms 信号时序为写操作时，即产生相应的唤醒源。

PMU 的功能特点如下：

- 在低功耗场景下可采样调试，中断及事件唤醒源，并发出唤醒请求；
- 有可配置寄存器，可配置不同种类唤醒源的使能；

- 在低功耗场景下控制对应时钟的 clock gating 控制信号;
- 在 STOP 模式下控制进入低功耗及唤醒机制，生成相应的控制及 reset 信号。

PMU 寄存器地址:

表 2.47: PMU 寄存器地址

Address	Function
0x40016000	唤醒源使能
0x40016004	计数器计数值

寄存器设置位描述:

表 2.48: PMU 唤醒源使能寄存器

位	名称	R/W	描述
31:4	写无效读为 0		保留
3	计数器使能	R/W	计数器开始计数
2	调试唤醒使能	R/W	调试可唤醒
1	事件唤醒使能	R/W	事件可唤醒
0	中断唤醒使能	R/W	中断可唤醒

2.6.7.1 E802 与 E801/E803S/E804/E805 相关接口比较

E802 的相关信号

表 2.49: E802 接口

信号名	I/O	Reset	定义
中断相关信号:			
pad_vic_int_vld[31:0]	I	0	中断有效信号：高电平有效。
vic_wic_int_exit[31:0]	O	0	退出中断信息
vic_wic_pending_clr[31:0]	O	0	清除中断 pending
vic_wic_pending_set[31:0]	O	0	Set 中断 pending
唤醒控制信号:			
wic_vic_int_awake_en[31:0]	I	0	中断唤醒的使能位
intraw_b	I	1	唤醒请求
had_pad_wakeup_req_b	O	1	Jtag 唤醒源
vic_wic_awake_data[31:0]	O	0	Awake enable 信息
vic_wic_awake_disable	O	0	写 IWDR 操作
vic_wic_awake_enable	O	0	写 IWER 操作
vic_wic_intraw_ack	O	0	唤醒握手信号

E801/E803S /E804/E805 的相关信号:

表 2.50: E801/803S/E804/E805 信号

信号名	I/O	Reset	定义
中断源信号:			
pad_vic_int_cfg[31:0]	I	•	中断源类型配置信号：0：电平中断源 1：脉冲中断源
pad_vic_int_vld[31:0]	I	0	中断有效信号：高电平有效。

由上表可见 E801/E803S/E804/E805 仅能通过 PMU 输入给 CPU 中断信号及中断配置信息。对于 E902/E906/E907 来说 vic 需替换为 clic。E802 与 PMU 交互的信号在中断处理方面额外包括了 set pending 触发中断的信息，以及退出中断的信息；在唤醒方面交互了 jtag 唤醒源的部分信息，awake en 的维护信号以及唤醒的握手信号。所以在实现相关功能时, E802 的设计与 E801/E803S/E804/E805 有所不同，具体不同点将在各个功能点进行描述。

2.6.7.2 唤醒控制

PMU 模块实现了下列三个唤醒源

- 事件唤醒

在低功耗模式下任何事件，都会造成事件唤醒源处于 pending 状态，直到低功耗模式被唤醒。只有该唤醒源 pending 且对应的控制位处于使能状态时该唤醒源能恢复时钟，产生有效的唤醒请求。

- 调试唤醒

在低功耗模式下通过观测 jtag2 状态机，任何通过 jtag 的写操作都会造成调试唤醒源处于 pending 状态，直到低功耗模式被唤醒。只有该唤醒源 pending 且对应的控制位处于使能状态时该唤醒源能恢复时钟，产生有效的唤醒请求。

- 中断唤醒

如 WIC 模块中有中断处于 pending 状态响应的中断，对应的 WIC 中存储的 awake en 处于使能状态且控制寄存器使能中断源唤醒时，该唤醒源能恢复时钟，产生有效的唤醒请求。

除 802 外其他核均不能直接向 cpu 发送唤醒请求。所以 PMU 只能通过向中断控制器发出中断请求，由中断控制器产生相应的唤醒请求。同时不支持 set pending 所引起的中断，因此只实现了部分中断唤醒的功能。

2.6.7.3 低功耗状态管理及相关控制

低功耗状态的管理主要通过主状态机和 power gating 状态机实现。

主状态机主要实现各个低功耗场景下的时钟控制

在 normal 状态下所有时钟均不进行控制，任何 lpmd 模式的改变都会进入对应的 lpmd 状态。

1) 在 wait 状态下仅关闭 CPU clk , 在 wakeup 后返回 normal
2) 在 doze 状态下关闭 CPU clk 和 per clk , 在 wakeup 后返回 normal
3) 在 stop 状态下关闭 CPU clk 和 per clk 并进入启动 power gating 状态机，只有等待 power gating 状态机完成所有处理后返回 IDLE 模式

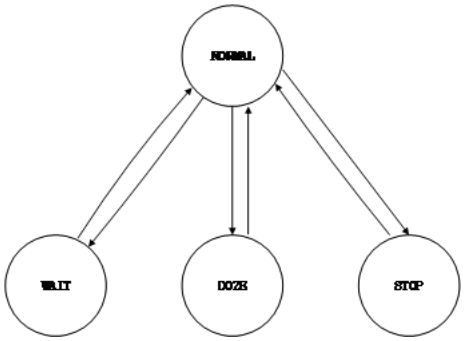

图 2.15: PMU 状态机

表 2.51: 时钟分布

时钟	**应用模块**	**Clock gating**
pmu_clk	GPIO TIMER STIMER PMU	不关闭
wic_clk	WIC	不关闭
per_clk	系统总线及系统总线上的其他 IP	Doze Stop 模式下关闭
cpu_clk	CPU 及 I/DMEM	所有低功耗模式下关闭

Power gating 状态机主要负责 power gating 过程以及 reset 相关信号的管理。

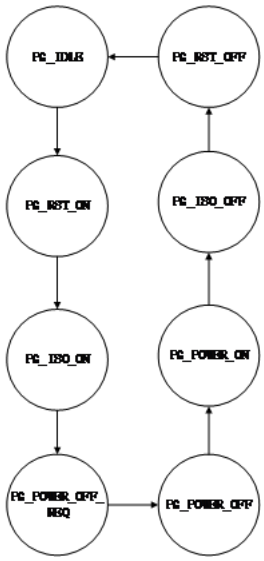

图 2.16: power gating 状态机

PG_IDLE 主状态机进入 STOP 状态后, 状态机启动

PG_RST_ON 下个周期无条件进入 PG_ISO_ON, 开始 reset 操作

PG_ISO_ON 下个周期无条件进入 PG_POWER_OFF_REQ，开始隔离操作

PG_POWER_OFF_REQ 等待 CPU 完成断电操作后进入 PG_POWER_OFF，否则保持状态等待

PG_POWER_OFF 等待唤醒，任何有效唤醒，进入 PG_POWER_ON，否则保持状态等待

PG_POWER_ON 下个周期无条件进入 PG_ISO_OFF，开始恢复供电操作

PG_ISO_OFF 下个周期无条件进入 PG_RST_OFF, 结束隔离操作

PG_RST_OFF 下个周期无条件返回 IDLE，结束 reset 操作

该状态机负责管理如下信号:

PG_RESET 该信号仅在 IDLE 和 RESET_OFF 两个状态下为高。和 pad_cpu_rst_b 相与后生成 pg_reset 用于在 power gating 时给除了 mem 外所有关闭时钟的模块进行 reset

PG_SLEEP_IN 表示处于 power gating 状态，仅在状态机处于 PG_POWER_OFF_REQ 和 PG_POWER_OFF 下有效

PG_ISOLATION 表示系统进行隔离处理，当状态机处于 PG_ISO_ON, PG_POWER_OFF_REQ, PG_POWER_OFF, PG_POWER_ON, PG_ISO_OFF 时有效

2.6.8 WIC

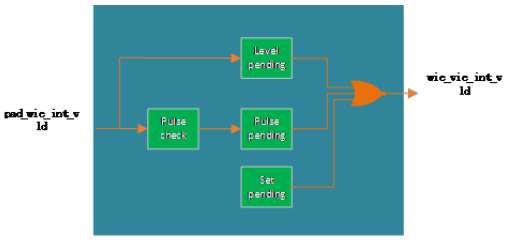

图 2.17: WIC 结构图

WIC 模块负责在任何场景下进行中断的采样与处理。在任何场景下均不断电，clock 均不停止。最大支持 32 个中断源。对电平、脉冲及 SET pending 操作等行为均会使中断请求处于 pending 状态，并将中断请求发送至 PMU 进行唤醒控制，同时发送至中断控制器，直到中断被响应后清除。WIC 也负责 awake en 的维护，任何对 IWER 和 ISDR 的操作都将作用于 WIC 内的相关寄存器，并将该使能信息传递给 PMU。

除 E802 外，SmartL 平台上的 WIC 均不支持 set pending 所引起的中断。不负责 awake en 的维护，寄存器组写无效读为 0，使能信息保存在中断控制器当中。对电平及脉冲中断源进行采样后，处于 pending 状态，拉起 int_vld 信号，并将 cfg 全部置 0，以电平中断的形式向中断控制器发出请求。当 cpu 退出中断模式后清除 pending 状态。

en 的维护，寄存器组写无效读为 0，使能信息保存在中断控制器当中。对电平及脉冲中断源进行采样后，处于 pending 状态，拉起 int_vld 信号，并将 cfg 全部置 0，以电平中断的形式向中断控制器发出请求。当 cpu 退出中断模式后清除 pending 状态。

2.7 开发板简介

SmartL 和 SmartH 开发板基于 Xilinx Artix-7 FPGA 通用平台。平头哥基于该通用平台可以评估各种 CPU 和 SoC。开发板实物图如下:

对 SmartL 和 SmartH，可用的资源由图中红色标出。

注意开发板 (类型 2): 内嵌了 JTAG, 不需要外置仿真器盒子, USB 线接 J18(CK-LINK), 串口线接 J9, 跳线帽接 PS CFG

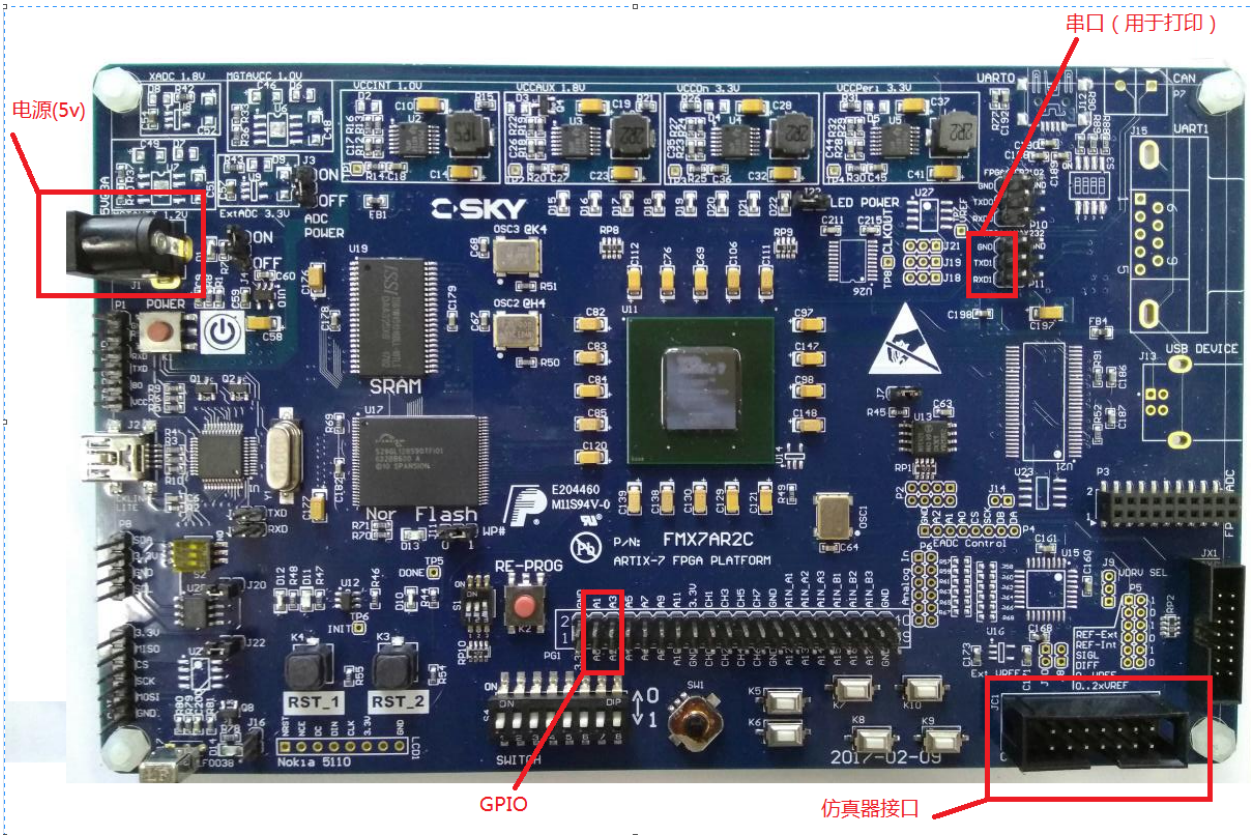

图 2.18: FPGA 开发板实物图

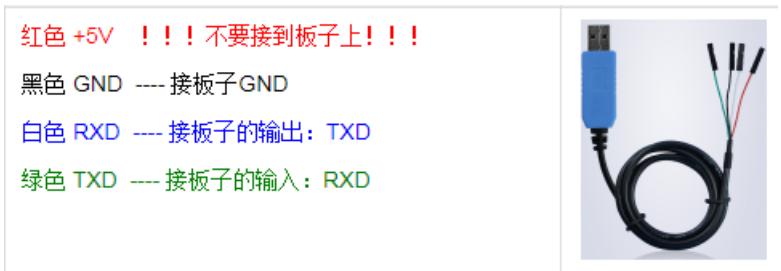

图 2.19: 串口线实物图

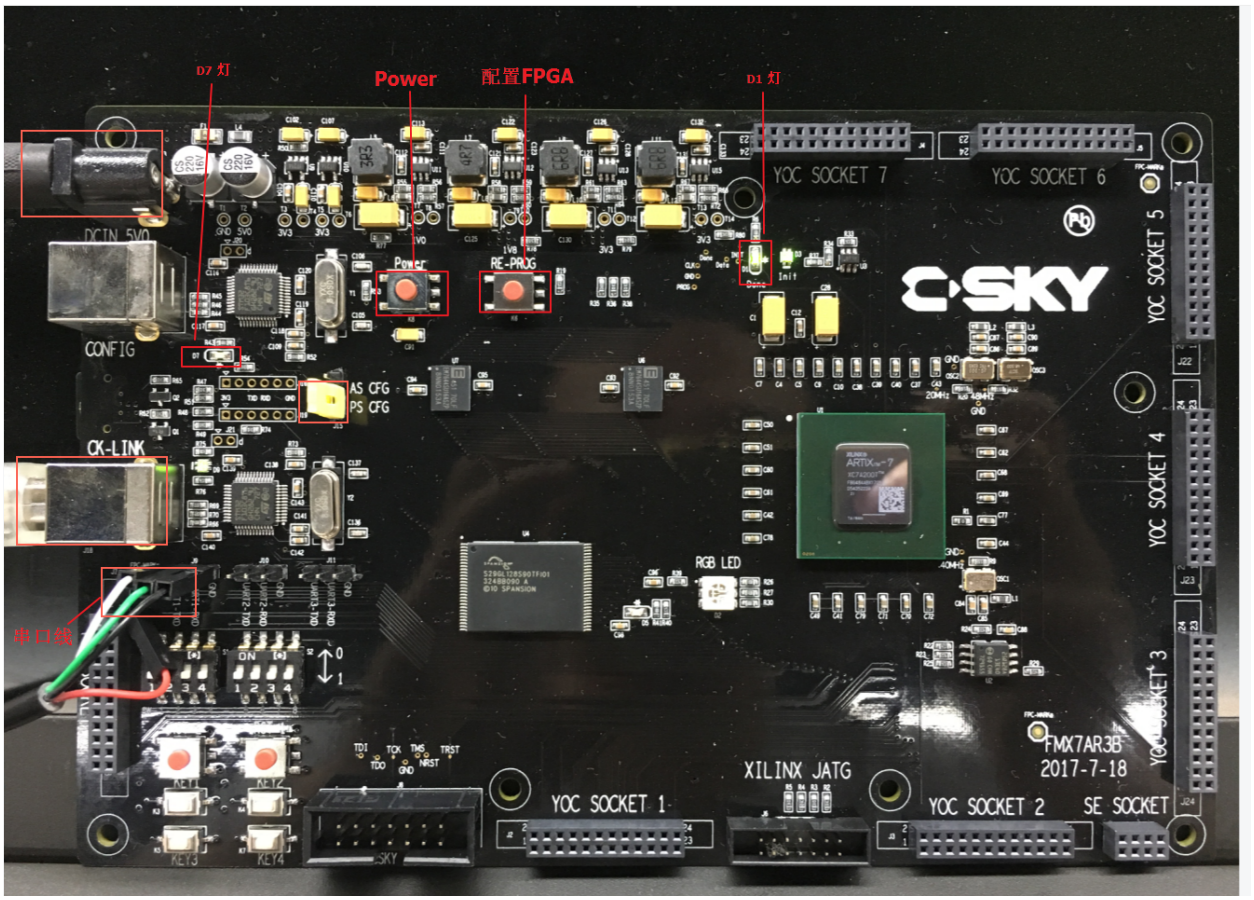

图 2.20: FPGA 开发板 (类型 2) 实物图

使用时先按 Power 键上电, 再按 RE-PROG 键配置 FPGA(FPGA bit 文件在 SD 卡上, 名为 cfg.bit), 待 D7 灯不闪, D1 灯正常后 (呼吸) 后配置完成.

第三章　代码结构

各目录内容简述如下：

- csi_core

CSI-Core 相关的接口定义，以及接口基于各 CPU 上的实现。

- csi_kernel

CSI-Kernel 相关的接口定义，以及 Rhino、FreeRTOSv10.3.1、uCos-III 等实时操作系统的对接示例代码。

- csi_driver

CSI-Driver 相关的接口定义，以及 SmartL 和 SmartH 上外围驱动的实现。

- board

存放板级相关代码。

- libs

存放通用的库实现。

- projects

存放各种参考示例，包括 benchmark 测试程序、驱动示例程序、rtos 示例程序等。同时包括了相关的工程项目文件。

- utilities

目前是空目录。

第四章　示例程序清单

4.1　CPU benchmark 程序

CPU benchmarck 程序用来测试 CPU 性能，包括 dhrystone、coremark、linpack（部分 CPU SDK）、helix（部分 CPU SDK）、whetstone（部分 CPU SDK）等标准测试程序。性能测试代码及工程在 projects/benchmark 目录下。

4.2　驱动示例程序

驱动示例程序保存在 projects/examples/driver 目录下。驱动示例程序演示了 SmartL 和 SmartH 的各种外围驱动的基本功能，包括：

- gpio 示例：演示了 gpio 中断功能
- timer 示例：演示了定时器中断以及计数模式功能
- uart 示例：演示了 uart 在各种模式（波特率、数据位、奇偶校验位、停止位）下的收发功能
- helloworld 示例：演示“Hello World!”

4.3　RTOS 示例程序

实时操作系统（RTOS）测试程序保存在 projects/examples/kernel 目录下。RTOS 示例程序演示了 RTOS 的基本功能，包括：

- 任务的创建与删除
- 事件/event 的使用
- 消息队列的使用
- 信号量的使用
- 互斥量的使用
- 软件定时器的使用
- 内存池的使用
- 时间管理的使用

第五章　工程配置

5.1　csi_config.h 配置

SDK 中提供的组件和示例程序可根据用户需要来配置，配置文件为对应工程下的 csi_config.h 文件。以下是对 SDK 中配置项的功能描述，用户一般仅需修改中断栈和堆的大小：

表 5.1: csi_config.h 配置

配置项	**意义**
CONFIG_SYSTEM_SECURE	配置为安全运行环境
CONFIG_CHIP_XXX	芯片平台为 XXX
CONFIG_KERNEL_XXX	配置 XXX Kernel
CONFIG_HAVE_VIC	使用 VIC 中断嵌套功能
CONFIG_ARCH_INTERRUPTSTACK	配置中断栈大小

第六章　Build、调试和上电固化

SmartL 和 SmartH SDK 支持三种编译调试环境，分别是 CDK、CDS 和 Makefile 命令行，推荐使用 CDK 进行开发。下面分章节介绍：

6.1　CDK

6.1.1　示例使用

以 usart example 为例。

第一步：双击打开工程文件

projects/examples/driver/usart/CDK/sxxx_evb.cdkproj

第二步：右键工程，选择 build，如下图

第三步：连接开发板电源，CKLink，串口线，打开串口软件（波特率 115200），参见开发板简介。

第四步：点击“Start/Stop Debugger”按钮进入调试。按钮如图所示：

更多 CDK 操作见 CDK 用户手册。

6.2　CDS

6.2.1　示例使用

以 usart example 为例。

第一步：import 工程

在 CDS 的“Project Explorer”窗口右键，点击“Import…”按钮

跳出“Import”窗口，选择“Existing CSKY Projects into Workspace”，点击 next

选择 projects/examples/driver/usart/CDS 目录，如下图：

去除“Copy projects into workspace”选项

点击“Finish”按钮完成 import

第二步：右键工程，点击“Build Project”按钮

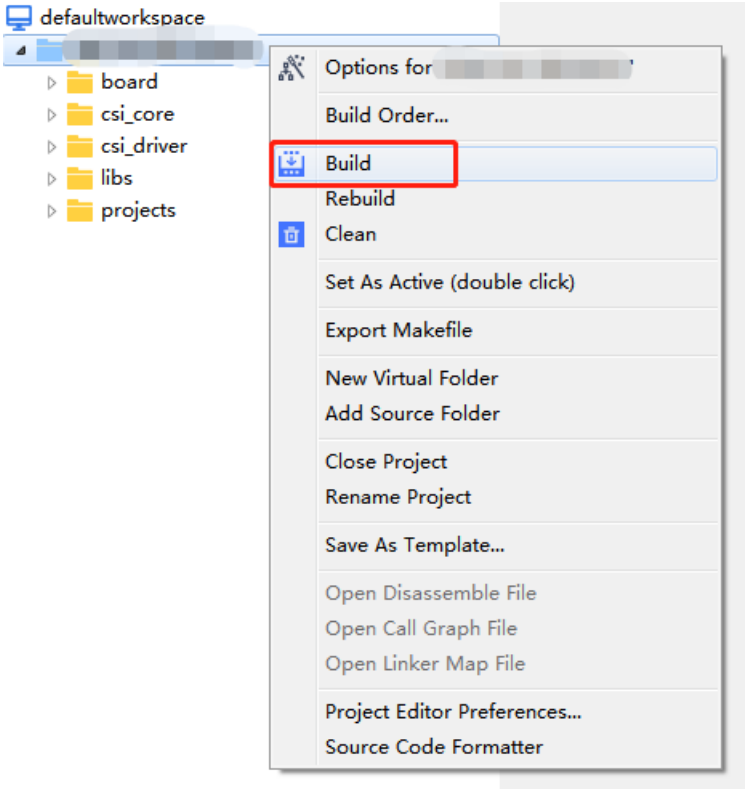

图 6.1: 选择 build

图 6.2: “Start/Stop Debugger” 按钮

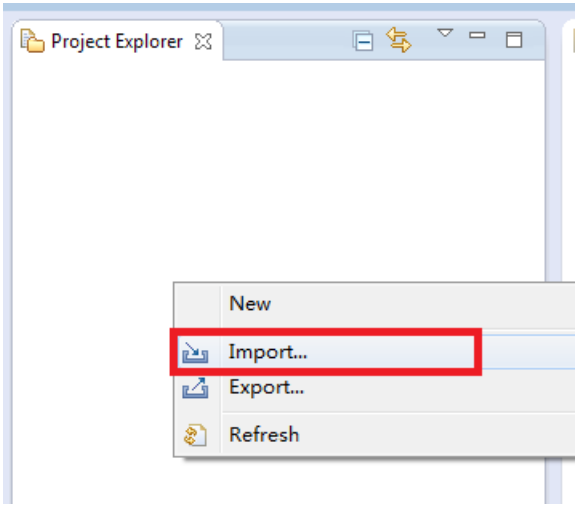

图 6.3: 示例使用图一

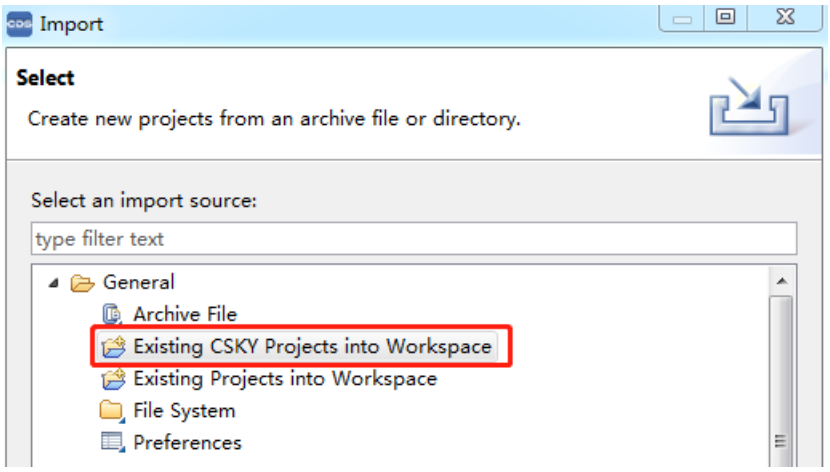

图 6.4: 示例使用图二

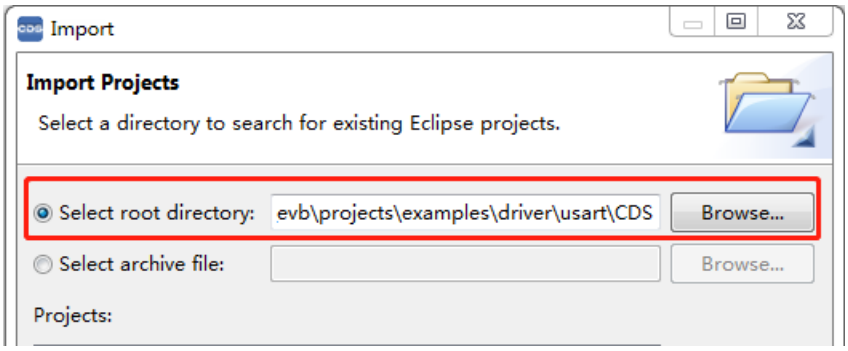

图 6.5: 示例使用图三

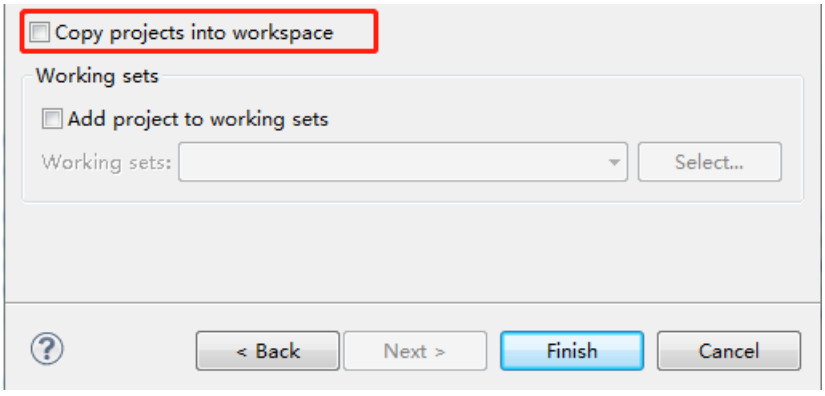

图 6.6: 示例使用图四

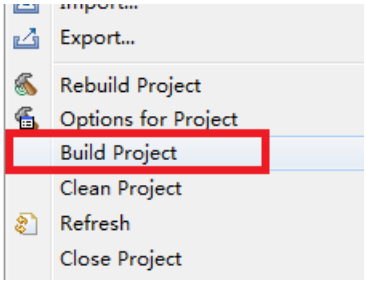

图 6.7: 示例使用图五

第三步：连接开发板电源，CKLink，串口线，打开串口软件（波特率 115200），参见开发板简介。

第四步：Debug

在工程上右键，点击 “Debug As” -> “CSky Application”

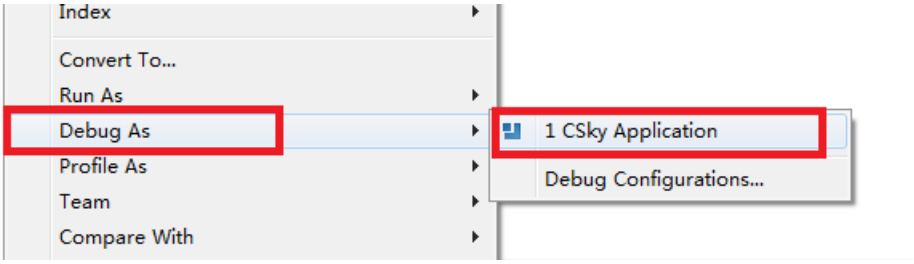

图 6.8: 示例使用图六

6.3 Makefile 命令行

6.3.1 示例使用

以 usart example 为例。

第一步：进入工程目录

```
$ cd projects/examples/driver/usart/
```

第二步：Build

```
$ make
```

第三步：连接 CskyDebugServer，参考《C-Sky Debugger Server User Guide》

第四步：生成.gdbinit 文件

在根目录生成一个.gdbinit，IP 和 PORT 可从 CskyDebugServer 打印中获取：

```
target remote <IP>:<PORT>
load
```

第五步：连接开发板电源，CKLink，串口线，打开串口软件（波特率 115200），参见开发板简介。

第六步：运行

玄铁 800 系列的命令如下所示：

```
$ csky-abiv2-elf-gdb out/xxx_evb.elf
...
(gdb) c
```

玄铁 900 系列的命令如下所示：

```
$ riscv64-unknown-elf-gdb out/xxx_evb.elf
...
(gdb) c
```

第七章　Tests 测试用例

为帮助客户进行 CSI 接口的兼容性与功能验证，THEAD 针对 CSI 接口设计了一套标准测试用例集，这些测试用例集覆盖了 CSI-Core、CSI-Driver、CSI-Kernel 基本功能的测试，测试用例采用 dtest 测试框架。dtest 测试框架使用：

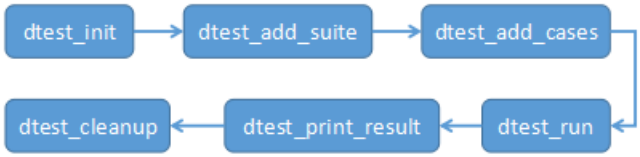

图 7.1: dtest 测试框架使用

表 7.1: dtest 测试框架使用

接口	说明
dtest_init	初始化 DTEST 框架
dtest_add_suite	增加一个测试模块
dtest_add_cases	增加测试 case 集
dtest_run	运行测试 case
dtest_print_result	打印测试结果
dtest_cleanup	回收资源

第八章　参考文档

[1] <CPU> 用户手册

[2] SmartL 用户手册、SmartH 用户手册

[3] XC7A-FPGA 开发板用户手册

[4] CDS 用户手册

[5] CDK 用户手册

[6] CSI-RTOS API 文档