|  |  |
| --- | --- |
| **学 号：** | **0121909360418** |

**SoC片上系统搭建**

**姓 名： 卢思远**

**班 级： 信息sy1901**

**专 业： 通信工程**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2022 | 年 | 2 | 月 | 25 | 日 |

**目录**

[第一章 简介 3](#_Toc96690308)

[**1.1 Cortex-M0 与 AMBA3 AHBLite** 3](#_Toc96690309)

[**1.2相关软硬件介绍** 4](#_Toc96690310)

[**1.3 地址映射** 4](#_Toc96690311)

[第二章 基础知识原理 5](#_Toc96690312)

[**2.1 ARM DesignStart** 5](#_Toc96690313)

[**2.2 AHB-Lite总线** 5](#_Toc96690314)

[2.2.1 总线简化设计 5](#_Toc96690315)

[2.2.2 AHB-Lite总线结构 5](#_Toc96690316)

[2.2.3 AHB-Lite接口信号 7](#_Toc96690317)

[**2.3 NONSEQ传输时序** 7](#_Toc96690318)

[2.3.1 零等待传输(no wait state transfer) 7](#_Toc96690319)

[2.3.2 等待传输(transfers with wait states) 8](#_Toc96690320)

[**2.4 向量表** 9](#_Toc96690321)

[第三章 实物效果 11](#_Toc96690322)

## 简介

本文将介绍如何搭建基于 ARM Cortex M0 CPU 的 SoC，参考 AMBA AHB 总线相关知识、cortex m0 用户手册、cortex m0 技术参考手册以及 ARMv6-M 架 构参考手册。

使用Keil、Modelsim、串口调试助手以及TD等工程软件以及逻辑分析仪和安路开发板EG4S20BG256硬件进行片上系统的构建。

## **1.1 Cortex-M0 与 AMBA3 AHBLite**

M0 CPU 总体结构如图 1-1 所示。提供了中断向量端口、AHB-Lite 端口以及 DAP 端口。

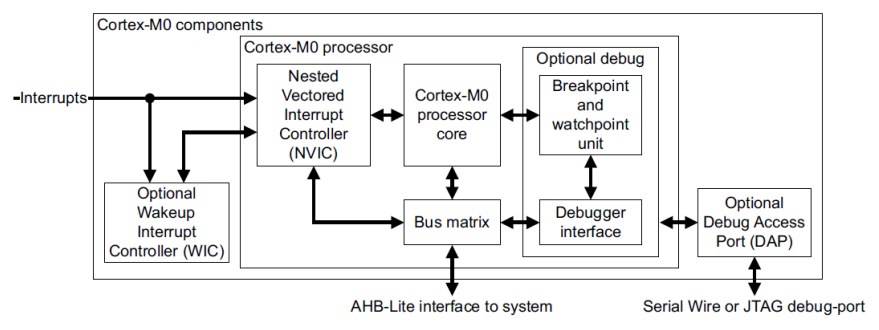


Figure 1-1. M0总线结构

根据题目要求和安路开发板EG4S20BG256的片上资源，搭建的SoC实现8个拨动按键分别控制8个LED，矩阵键盘输出到数码管，串口按1s的间隔发送字符并应逻辑分析仪持续显示字符。依据以上要求，设置如图1-2总线。

Figure 1-2. 自设总线架构

## **1.2相关软硬件介绍**

用到三个软件平台：

1）TD5.0.3.30786-64bit

2）Modelsim

3）Keil5

用到以下硬件平台：

1）FPGA (EG4S20BG256)

## **1.3 地址映射**

Cortex-M0处理器提供了4GB的存储器寻址空间，同时规定了这些区域的使用方法，如图1.2。根据要求的地址映射和题目要求进行对对应ram和rom的程序编写，根据题目要求设置相关外设地址，自定地址与M0要求地址映射对应关系如图1-3

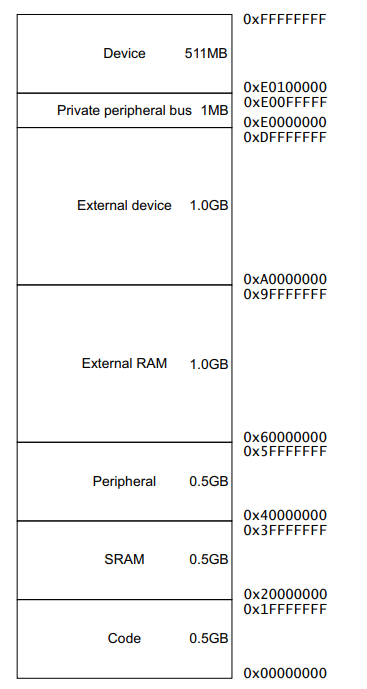


Figure 1-3.自设地址映射

## 基础知识原理

## **2.1 ARM DesignStart**

在2010年，ARM正式推出ARM DesignStart计划，开源了1000多个物理IP，为用户提供了快速获得ARM IP的方式。在2015年10月15日，ARM开源自己的Cortex-M0核，希望大家使用Cortex-M0核自主搭建SoC，实现定制化的物联网终端芯片的设计，让Cortex-M系列处理器在物联网领域得到更多的应用。这次开源引起了巨大的行业反响，掀起了一波超高能效定制化SoC的开发热潮。随后，ARM又在2017年6月20日开源了Cortex-M3核，并在2018年10月22日开源了Cortex-A5核。DesignStart计划的不断推进，使得ARM在物联网领域也逐渐建立起优势。

ARM公司开放了ARM Cortex-M0 DesignStart处理器的Verilog HDL源码级描述文件，通过该简化配置的处理器核，可实现对Cortex-M0的访问。此外，通过对此处理器的开发，我们可更好地理解和掌握Cortex-M0架构以及运行原理。

## **2.2 AHB-Lite总线**

### 2.2.1 总线简化设计

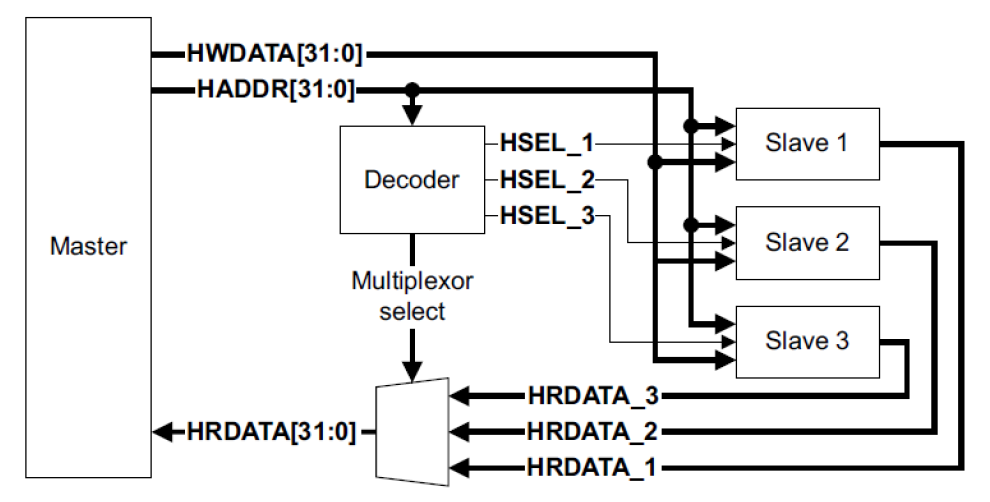
ARM公司提供的高级微控制器总线结构（Advanced Microcontroller Bus Architecture，AMBA）规范是实现ARM处理器和外部设备互连的基础，在SoC设计中用于片上总线。AMBA3中的AHB（Advanced High Performance Bus），被称为高性能总线，而AHB-Lite是其子集，简化了AHB总线的设计，典型地，只有一个主设备。

同时，又由于Cortex-M0处理器核的特性，可在此基础上精简设计，简化协议。在处理器核的代码中，端口的HTRANS最低位恒为0，说明M0仅支持NONSEQ传输，且HBURST也恒为0，说明M0不支持BURST传输。因此，在后面的外设总线接口的设计中，只需要满足NONSEQ类型的传输即可，为简化设计提供了条件。

### 2.2.2 AHB-Lite总线结构

AMBA AHB-Lite是面向高性能的可综合设计，提供了一个总线接口来支持Master并提供高操作带宽。下图是一个具有一个Master的AHB-Lite的系统，包括一个Master和三个Slave。利用内部逻辑生成了一个地址解码器和一个Slave-to-Master多路转换器。

Figure 2-1.AHB-Lite总线结构



在SoC具有多个外设，但处理器核只有一个总线接口的情况下，就需要用到总线扩展模块，使处理器核能够访问多个外设，实现复用。对于Decoder，只要地址总线位于某外设的地址编码区域中，则判定为处理器核对该外设提起的一次总线操作，对应的HSEL将置位；对于Slave Mux，则是根据每个外设的选择信号，对所有外设返回的读取数据（HRDATA）、响应信号（HRESP）以及反馈信号（HREADYOUT）进行选择，保证返回的端口数据来自于当前总线操作的目标外设，总线扩展模块如图2-2所示。

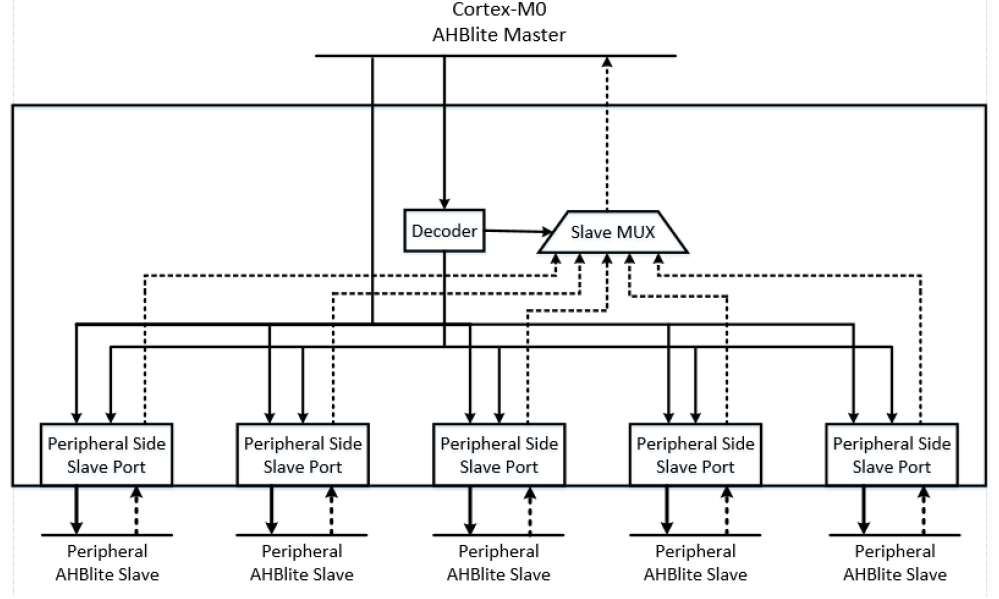


Figure 2-2.总线扩展模块结构图

### 2.2.3 AHB-Lite接口信号

AHB-Lite的接口信号包括：全局信号、主设备接口、从设备接口。它们分别是实现Cortex-M0处理器、各外设接口的关键。

Figure 2-3.主要接口信号

## **2.3 NONSEQ传输时序**

### 2.3.1 零等待传输(no wait state transfer)

**第一个周期的上升沿，**master驱动地址和控制信号；

**第二个周期的上升沿，**slave采样地址和控制信号，并将HREADY拉高；

如果是写操作，master会在第二个周期的上升沿传输要写入的数据；

如果是读操作，slave会在HREADY信号拉高后将读取的数据写入总线；

**第三个周期的上升沿，**

如果是写操作，master获取HREADY高信号，表明slave已成功接收数据，操作成功；

如果是读操作，master获取HREADY高信号，表明此时的读数据有效并且接收下来，操作成功。需要注意，HREADY信号在数据有效期间必须为高，并且延续到第三个周期的上升沿之后，确保master的正确采样。

当 HCLK 正沿触发后， M 送出地址与控制信号，HCLK 由 CLOCK 驱动送至 AHB 接口。

S 在 HCLK下一个正沿触发时， S 收到地址与控制信号。

在 HCLK 的第三个正沿触发时， M 收到S 的响应信号。AHB支持流水线的动作，在收上一笔数据的同时，可将下一笔数据的地址送出。

Figure 2-4.基本读操作

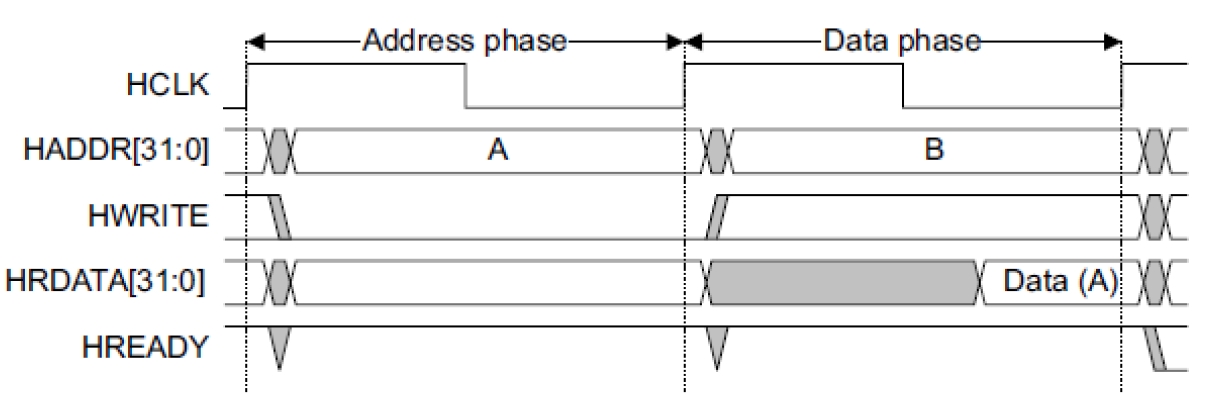
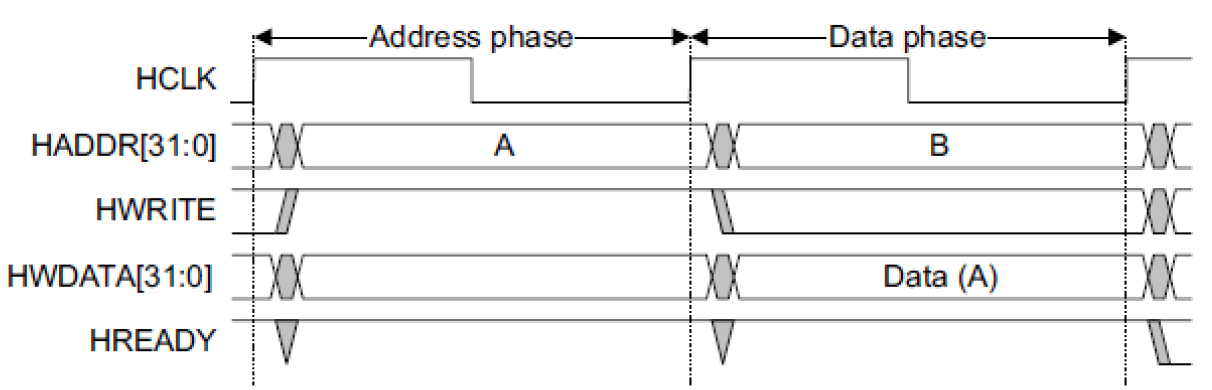


Figure 2-5.基本写操作



### 2.3.2 等待传输(transfers with wait states)

传送完成时间因等待而延迟，这是由于寻址到较慢的 S 单元所造成。当 HREADY 被 S 拉低时，会发生等待状态。需要延迟传送的周期时，就会在数据期间将 HREADY 拉低，此时地址与数据都必须延伸。有等待状态的 AHB 传递，下图中可看到在数据模式时，时序必须多花二个周期的延迟等待时间(HREADY 为 0)，S 才能正常的接收到 M 写的数据或准备好 M 欲读取的数据。

slave可以及时处理master请求，但也可能存在slave太慢不能立即处理的情况。这时需要让master稍微等一等，需要slave插入一些等待的状态。如下图所示，HREADY信号在第二和第三周期拉低，意在告诉master，slave不能立即处理，需要master等待2个周期。在这里需要注意：

**如果是写操作，master需要在等待期间保持写数据不变，直到本次传输完成；**

**如果是读操作，slave不需要一开始就给出数据，仅当HREADY拉高后才给出有效数据。**

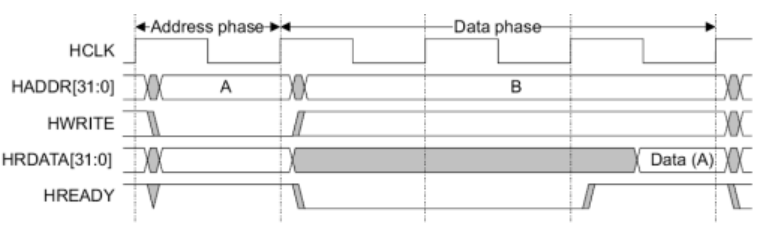


Figure 2-6.有等待读操作

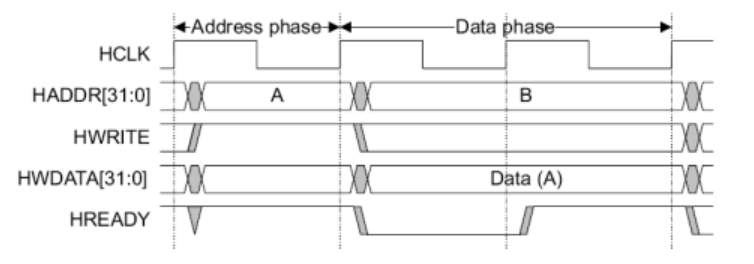


Figure 2-7.有等待写操作

## **2.4 向量表**

Cortex-M0的程序代码保存在片上的代码存储空间，程序代码以镜像文件的形式存在，如图2.6所示。程序镜像文件包含：

（1）向量表。包含异常（向量）起始地址，以及主堆栈指针（Main Stack Pointer，MSP）的值。

（2）C语言编写的启动代码。

（3）程序代码，包含应用程序和数据。

（4）C库代码，用于C库函数和程序代码。

根据ARMv6-M架构参考手册，Cortex-M0的启动过程如下：

（1）在复位使能时，CPU处于Reset异常状态；

（2）复位释放后，从地址0x00000000处加载栈顶地址，即汇编代码中\_\_Vector的第一行。

（3）从地址0x00000004处加载复位处理函数的地址。

（4）PC改变为0x00000004中的值，开始执行复位处理，同时CPU的工作状态从异常模式切换为线程模式，开始正常工作。

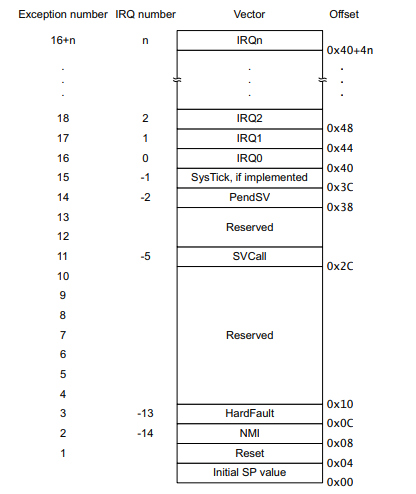


Figure 2-8.M0向量表

## 实物效果

1. 成功使用业界主流指令集兼容的软核处理器构建 SoC，系统包含：

1) Cortex-M0的开源软核处理器；

2) 成功利用片上资源实现ROM 与 RAM；

3) 有支持外部在线开发调试的JTAG 调试接口并用stlink进行debug调试；

4) 存在与芯片外部引脚连接的 GPIO 外设控制灯的亮灭和数码管的显示。

2. 成功使用 Keil μVision5 工具编写并生成软件程序，实现对于 SoC 上运行 的软件程序的在线实时调试。

板上调试成功让led发生亮灭，矩阵按键成功控制数码管进行正确的数字显示，如图3-1。

3. 成功通过 SoC 硬件数字 GPIO 来驱动逻辑分析仪。

以串口通信协议成功在逻辑分析仪上滚动打出对应的 SoC 内核名称cortex\_M0以及当前数码管显示数模SEG\_X，如图3-2

根据上述调试结果，成功实现题目要求全部内容。

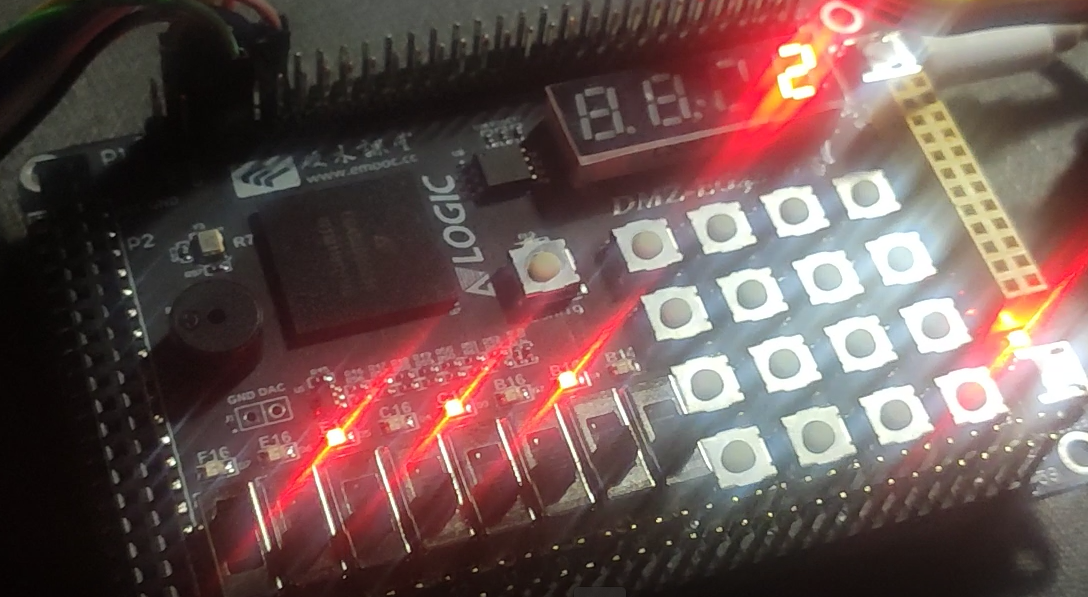


Figure 3-1.GPIO控制LED

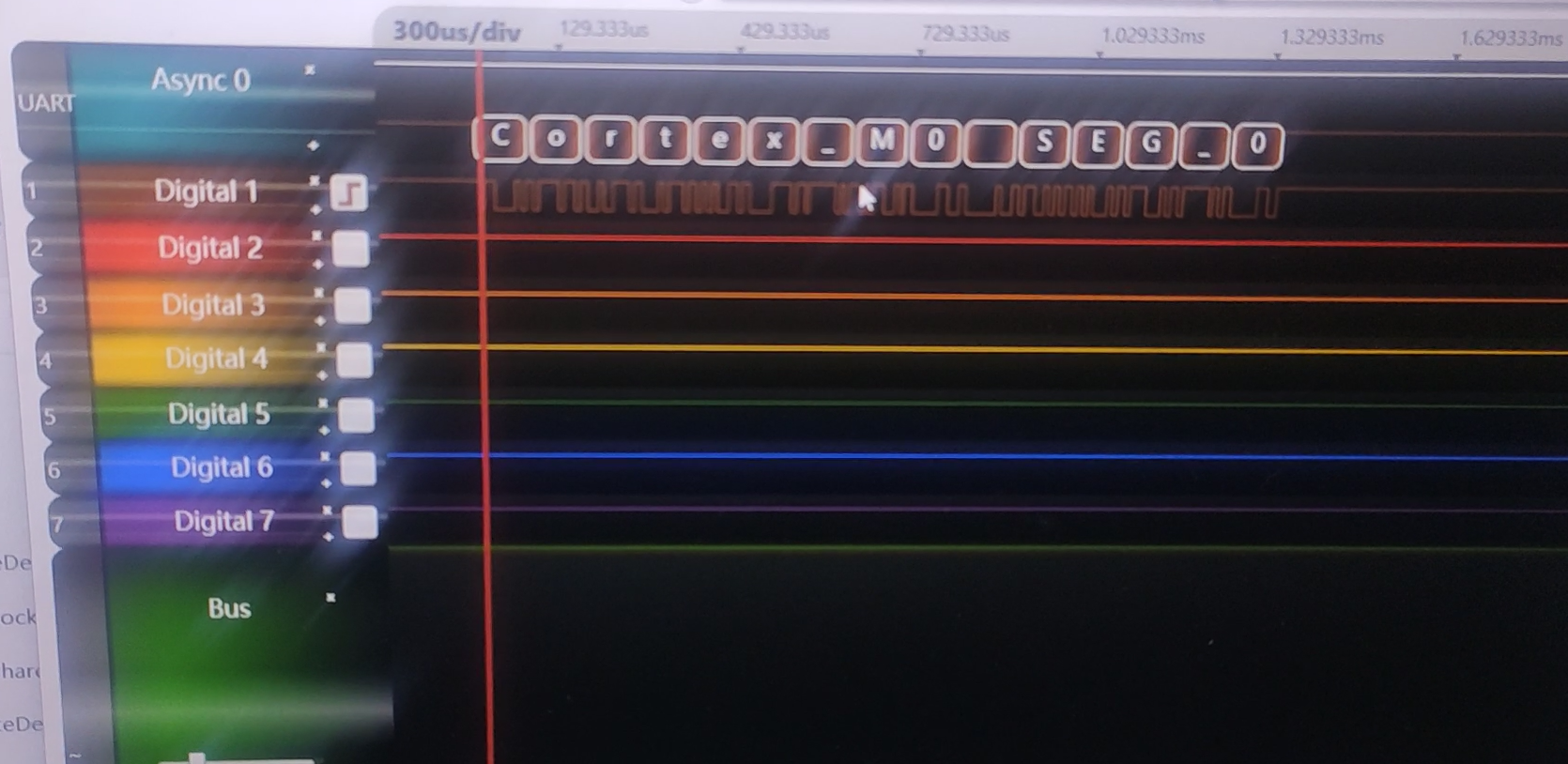


Figure 3-2.逻辑分析仪显示内核