PAN1070 DMA Sample Application Note

PAN-CLT-VER-A0, Rev 0.1

PanchipMicroelectronics

www.panchip.com

修订历史

版本	修订日期	描述
V0.1	2022-10-12	初始版本创建

目录

第1章 测试目的

1. PAN1070 DMA 基本功能测试。
2. 通过测试，形成 DMA 测试方案，DMA 功能应用库及使用方案。
3. 给出对 DMA 模块的使用说明文档。

第2章 测试内容

2.1 测试内容

1. 寄存器默认值
2. SPI 模块，使用 DMA 方式收发数据
   a) SPI 作为 Master 发送数据，Memory to Peripheral，配置一
   b) SPI 作为 Master 发送数据，Memory to Peripheral，配置二
   c) SPI 作为 Master 发送数据，Memory to Peripheral，配置三
   d) SPI 作为 Master 接收数据，Peripheral to Memory，配置一
   e) SPI 作为 Master 接收数据，Peripheral to Memory，配置一
   f) SPI 作为 Master 接收数据，Peripheral to Memory，配置三
3. SDIO 模块，使用 DMA 方式收发数据（PN108 无 SDIO 模块）
4. I2C 模块，使用 DMA 方式收发数据
   a) I2C 作为 Master 发送数据，Memory to Peripheral
   b) I2C 作为 Slave 接收数据，Peripheral to Memory
5. UART 模块，使用 DMA 方式收发数据
   a) UART 发送数据，Memory to Peripheral
   b) UART 接收数据，Peripheral to Memory
6. Memory to Peripheral Gather 功能（当前 DMAC 硬件不支持）
7. Peripheral to Memory Scatter 功能（当前 DMAC 硬件不支持）
8. DMA Peripheral to Peripheral 方式传输数据
   a) I2C to UART
   b) I2C to SPI
   c) SPI to UART
9. DMA Single Channel Multi-block 传输
10. DMA 两通道传输
11. DMA 三通道传输（PN107 DMAC 仅有 2 个通道）
12. DMA Bus Lock 功能（当前 DMAC 硬件不支持）
13. DMA Channel Lock 功能（当前 DMAC 硬件不支持）
14. DMA Channel Bus Lock 功能（当前 DMAC 硬件不支持）
15. DMA 优先级设定
16. DMA Multi-channel Block 传输
17. DMA Memory to Memory 方式传输数据
18. DMA 传输 1 个 Byte 数据（Memory to SPI）
19. DMA 传输 4092 个 Bytes 数据（Memory to SPI）
20. DMA 同一模块收发数据均使用 DMA（Memory 2 SPI and SPI 2 Memory）
21. DMA 硬件 multiblock 方式传输
22. DMA Flash to SPI 方式传输

2.2 环境配置

2.2.1 软件环境

2.2.1.1 待测代码

测试工程文件：

<PAN1070-DK>\03_MCU\mcu_samples\DMAC\keil\DMAC.uvprojx

测试源文件目录：

<PAN1070-DK>\03_MCU\mcu_samples\DMAC\src

2.2.2 硬件环境

1、PAN1070 COB Test Board 2 块

a) UART0（测试交互接口，TX：P16，RX：P17 ）

b) DMAC（待测模块，与辅助测试模块配合测试）

c) SPI0（辅助测试模块，MOSI: P11，MISO: P12，CLK: P04，CS: P03）

d) SPI1（辅助测试模块，MOSI: P21，MISO: P24，CLK: P22，CS: P23）

e) I2C0（辅助测试模块，SCL: P07，SDA: P10）

f) SWD（用来调试和烧录程序，SWDCLK: P00，SWDIO: P01）

2、USB 转串口小转板 x2，用来分别连接 PC 与 2 个 Test Board

3、逻辑分析仪（波形抓取工具）

4、JLink（SWD 调试与烧录工具）

2.2.2.1 软件工具

1、SecureCRT（用于显示 PC 与 Test Board 的交互过程，打印 log 等）

2、KingstVIS（逻辑分析仪 LA1010 配套软件）

第3章 测试流程

3.1 环境说明

3.1.1 测试程序编译烧录

打开测试工程，确保可以编译通过。

3.1.2 硬件接线

接线方面，需要：

1. 将 UART1 与 PC 串口调试助手通过 USB 串口小转板连接；

2. 在测试 Testcase 1 ~ 3 时，需要将 Test Board SPI0 的 MOSI (P11)、MISO (P12)、CLK (P04)、CS (P03)连接至逻辑分析仪；

3. 在测试 Testcase 4 ~ 6 时，需要将 Test Board SPI0 的 MOSI (P11)、MISO (P12)、CLK (P04)、CS (P03)与同一块 Test Board SPI1 的 MOSI (P21)、MISO (P24)、CLK (P22)、CS (P23)引脚对应连接起来；

4. 在测试 Testcase 8 ~ 9 时，需要将 Test Board 1 的 I2C_SCL (P07)、I2C_SDA (P10)与 Test Board 2 的相同引脚对应连接起来；

5. 在测试 Testcase A ~ B 时，需要将 Test Board UART1 的 Tx（P16）、Rx（P17）连接至逻辑分析仪；

6. 在测试 Testcase E 时，需要将 Test Board 1 的 I2C_SCL (P07)、I2C_SDA (P10)与 Test Board 2 的相同引脚对应连接起来并连至逻辑分析仪，同时将 Test Board 1 UART1 的 Tx（P16）引脚连接至逻辑分析仪；

7. 在测试 Testcase F 时，需要将 Test Board 1 的 I2C_SCL (P07)、I2C_SDA (P10)与 Test Board 2 的相同引脚对应连接起来并连至逻辑分析仪，同时将 Test Board 1 SPI0 的 MOSI (P11)、MISO (P12)、CLK (P04)、CS (P03)连接至逻辑分析仪；

8. 在测试 Testcase G 时，需要将 Test Board SPI0 的 MOSI (P11)、MISO (P12)、CLK (P04)、CS (P03)与同一块 Test Board SPI1 的 MOSI (P21)、MISO (P24)、CLK (P22)、CS (P23)引脚对应连接起来并连至逻辑分析仪，同时将 Test Board 的 UART1 Tx（P16）引脚连接至逻辑分析仪；

9. 在测试 Testcase H 时，需要将 Test Board SPI0 的 MOSI (P11)、MISO (P12)、CLK (P04)、CS (P03)连接至逻辑分析仪；

10. 在测试 Testcase I 时，需要将 Test Board SPI0 的 MOSI (P11)、MISO (P12)、CLK (P04)、CS (P03)与同一块 Test Board SPI1 的 MOSI (P21)、MISO (P24)、CLK (P22)、CS (P23)**分别**连接至逻辑分析仪；

11. 在测试 Testcase N ~ O 时，需要将 Test Board SPI0 的 MOSI (P11)、MISO (P12)、CLK (P04)、CS (P03)与 UART1 的 Tx (P16)分别连接至逻辑分析仪；

12. 在测试 Testcase Q ~ R 时，需要将 Test Board SPI0 的 MOSI (P11)、MISO (P12)、CLK (P04)、

CS (P03)连接至逻辑分析仪；

13. 在测试 Testcase S 时，需要将 Test Board SPI0 的 MOSI (P11)、MISO (P12)、CLK (P04)、CS (P03)与同一块 Test Board SPI1 的 MOSI (P21)、MISO (P24)、CLK (P22)、CS (P23)引脚对应连接起来。

2. 在测试 Testcaseu U 时，需要将 Test Board SPI0 的 MOSI (P11)、MISO (P12)、CLK (P04)、CS (P03)连接至逻辑分析仪；

3.2 DMA 工作流程

参考 User Manual 文档。

3.3 测试程序初始化

硬件连线完成并烧录测试程序后，Test Board 上电，观察 Debug Port 是否正常打印测试主菜单。

```
+-------------------------------------------------------------------+
|                    PN107 DMA Sample Code.                         |
+-------------------------------------------------------------------+
|   Press key to start specific testcase:                           |
|                                                                   |
|   Input '0'    Testcase 0:  Register Default Value Check.         |
|   Input '1'    Testcase 1:  SPI Dma Mem 2 Peripheral Tranfer  Case|
|   Input '2'    Testcase 2:  SPI Dma Mem 2 Peripheral Tranfer  Case|
|   Input '3'    Testcase 3:  SPI Dma Mem 2 Peripheral Tranfer  Case|
|   Input '4'    Testcase 4:  SPI Dma Peripheral 2 Mem Tranfer  Case|
|   Input '5'    Testcase 5:  SPI Dma Peripheral 2 Mem Tranfer  Case|
|   Input '6'    Testcase 6:  SPI Dma Peripheral 2 Mem Tranfer  Case|
|   Input '7'    Testcase 7:  SDIO Tranfer                      Case|
|   Input '8'    Testcase 8:  I2C Dma Mem 2 Peripheral Tranfer  Case|
|   Input '9'    Testcase 9:  I2C Dma Peripheral 2 Mem Tranfer  Case|
|   Input 'a'    Testcase 10: UART Dma Mem 2 Peripheral Tranfer Case|
|   Input 'b'    Testcase 11: UART Dma Peripheral 2 Mem Tranfer Case|
|   Input 'c'    Testcase 12: SPI Dma Mem 2 Peripheral Gather   Case|
|   Input 'd'    Testcase 13: SPI Dma Peripheral 2 Mem Scatter  Case|
|   Input 'e'    Testcase 14: DMAC Iic 2 Uart Tranfer           Case|
|   Input 'f'    Testcase 15: DMAC Iic 2 Spi Tranfer            Case|
|   Input 'g'    Testcase 16: DMAC Spi 2 Uart Tranfer           Case|
|   Input 'h'    Testcase 17: DMA MultiBlock Tranfer            Case|
|   Input 'i'    Testcase 18: DMA Two Channel Tranfer           Case|
|   Input 'j'    Testcase 19: DMA Three Channel Tranfer         Case|
|   Input 'k'    Testcase 20: DMA Bus Locked Tranfer            Case|
|   Input 'l'    Testcase 21: DMA Channel Locked Tranfer        Case|
|   Input 'm'    Testcase 22: DMA Channel Bus Locked Tranfer    Case|
|   Input 'n'    Testcase 23: DMA Priority Tranfer              Case|
|   Input 'o'    Testcase 24: DMA MultiChannel Block Tranfer    Case|
|   Input 'p'    Testcase 25: DMA Mem 2 Mem Tranfer             Case|
|   Input 'q'    Testcase 26: DMA One Byte Tranfer              Case|
|   Input 'r'    Testcase 27: DMA 4092 Byte Tranfer             Case|
|   Input 's'    Testcase 28: DMA SPI Dma Full Duplex Tranfer   Case|
|   Input 't'    Testcase 29: DMA hardware multiblock Tranfer   Case|
|   Input 'u'    Testcase 30: DMA flash to spi Tranfer   Case       |
+-------------------------------------------------------------------+
```

3.4 基本功能验证

3.4.1 DMA 所有寄存器默认状态

在主菜单下，输入‘0’命令 打印所有寄存器默认值：

测试目的：

检查所有 DMA 相关寄存器复位 Default 值状态。

测试预期：

寄存器默认值应和 Datasheet 上 DMA 模块默认值一致。

测试现象：

```
[20:22:48.781]发→◇0□
[20:22:48.786]收←◆
DMA->CH[0].SAR_L      :00000000
DMA->CH[0].SAR_H      :00000000
DMA->CH[0].DAR_L      :00000000
DMA->CH[0].DAR_H      :00000000
DMA->CH[0].CTL_L      :00304801
DMA->CH[0].CTL_H      :00000002
DMA->CH[0].CFG_L      :00000E00
DMA->CH[0].CFG_H      :00000004
DMA->CH[1].SAR_L      :00000000
DMA->CH[1].SAR_H      :00000000
DMA->CH[1].DAR_L      :00000000
DMA->CH[1].DAR_H      :00000000
DMA->CH[1].CTL_L      :00304801
DMA->CH[1].CTL_H      :00000002
DMA->CH[1].CFG_L      :00000E20
DMA->CH[1].CFG_H      :00000004
DMA->RAW_TFR_L        :00000000
DMA->RAW_TFR_H        :00000000
DMA->RAW_BLOCK_L      :00000000
DMA->RAW_BLOCK_H      :00000000
DMA->RAW_SRCTRAN_L    :00000000
DMA->RAW_SRCTRAN_H    :00000000
DMA->RAW_DSTTRAN_L    :00000000
DMA->RAW_DSTTRAN_H    :00000000
DMA->RAW_ERR_L        :00000000
DMA->RAW_ERR_H        :00000000
DMA->STATUS_TFR_L     :00000000
DMA->STATUS_TFR_H     :00000000
DMA->STATUS_BLOCK_L   :00000000
DMA->STATUS_BLOCK_H   :00000000
DMA->STATUS_SRCTRAN_L:00000000
DMA->STATUS_SRCTRAN_H:00000000
DMA->STATUS_DSTTRAN_L:00000000
DMA->STATUS_DSTTRAN_H:00000000
DMA->STATUS_ERR_L     :00000000
DMA->STATUS_ERR_H     :00000000
DMA->MSK_TFR_L        :00000000
DMA->MSK_TFR_H        :00000000
DMA->MSK_BLOCK_L      :00000000
DMA->MSK_BLOCK_H      :00000000
DMA->MSK_SRCTRAN_L    :00000000
DMA->MSK_SRCTRAN_H    :00000000
DMA->MSK_DSTTRAN_L    :00000000
DMA->MSK_DSTTRAN_H    :00000000
DMA->MSK_ERR_L        :00000000
DMA->MSK_ERR_H        :00000000
DMA->STATUS_INT_L     :00000000
DMA->STATUS_INT_H     :00000000
DMA->DMA_CFG_REG_L    :00000000
DMA->DMA_CFG_REG_H    :00000000
DMA->CH_EN_REG_L      :00000000
DMA->CH_EN_REG_H      :00000000
```

测试分析：

由 Log 可知，除 CTL 与 CFG 相关寄存器不为 0 外，其他寄存器均为 0，符合预期。

3.4.2 SPI 模块，使用 DMA 方式收发数据

3.4.2.1 SPI 作为 Master 发送数据，Memory to Peripheral，配置一

测试目的：

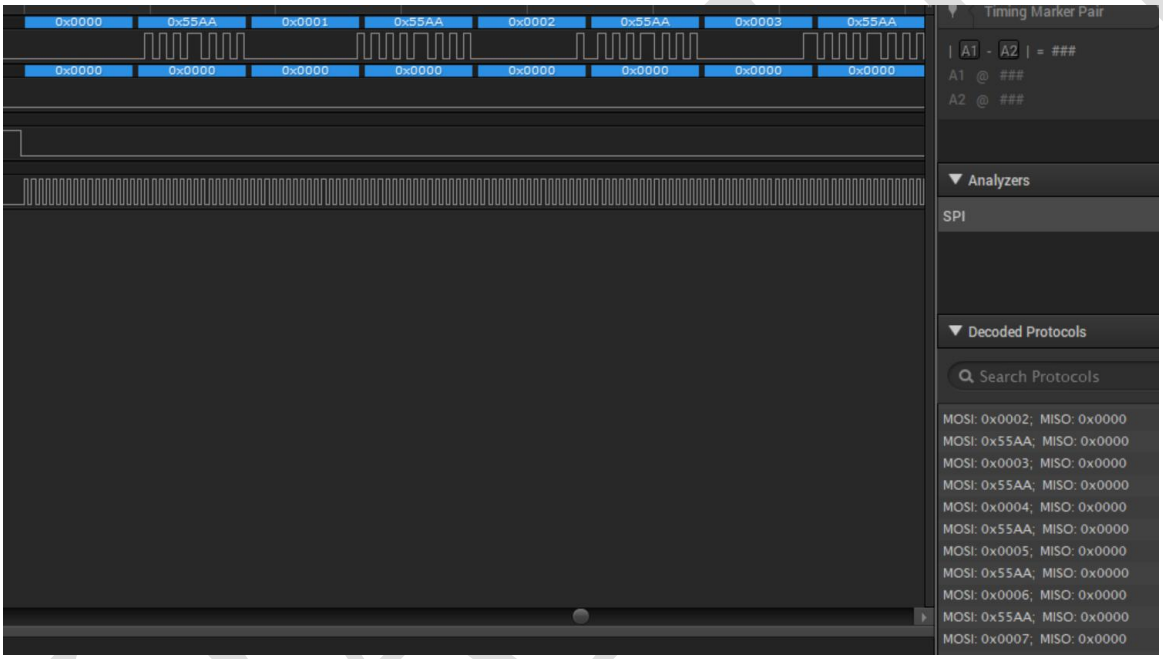
将 DMA 配置成 Memory Burst Length 为 1，验证传输 256 字（1024 字节）数据是否正常。

测试预期：

SPI 传输波形正常。

测试现象：

先打开逻辑分析仪软件，开启 SPI 波形抓取功能，然后在 Test Board 串口界面，输入‘1’命令，在 LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，SPI 成功发送了 5120 个半字（即 10240 个字节）出去。实际上，DMA 从 Memory 向 SPI 传输的数据范围是 0x55AA0000 ~ 0x55AA00FF，即重复传输 10 次，每次传输了 1024 个字节的数据，符合预期。

3.4.2.2 SPI 作为 Master 发送数据，Memory to Peripheral，配置二

测试目的：

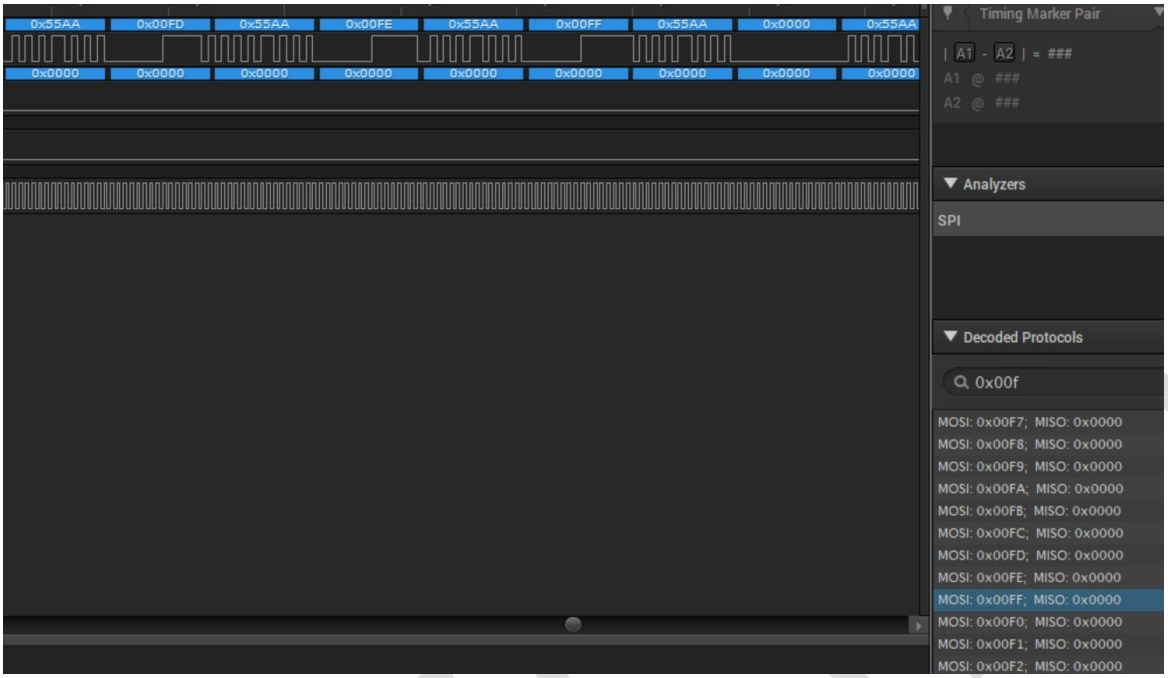
将 DMA 配置成 Memory Burst Length 为 4，验证传输 256 字（1024 字节）数据是否正常。

测试预期：

SPI 传输波形正常。

测试现象：

先打开逻辑分析仪软件，开启 SPI 波形抓取功能，然后在 Test Board 串口界面，输入‘2’命令，在 LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，SPI 成功发送了 5120 个半字（即 10240 个字节）出去。实际上，DMA 从 Memory 向 SPI 传输的数据范围是 0x55AA0000 ~ 0x55AA00FF，即重复传输 10 次，每次传输了 1024 个字节的数据，符合预期。

3.4.2.3 SPI 作为 Master 发送数据，Memory to Peripheral，配置三

测试目的：

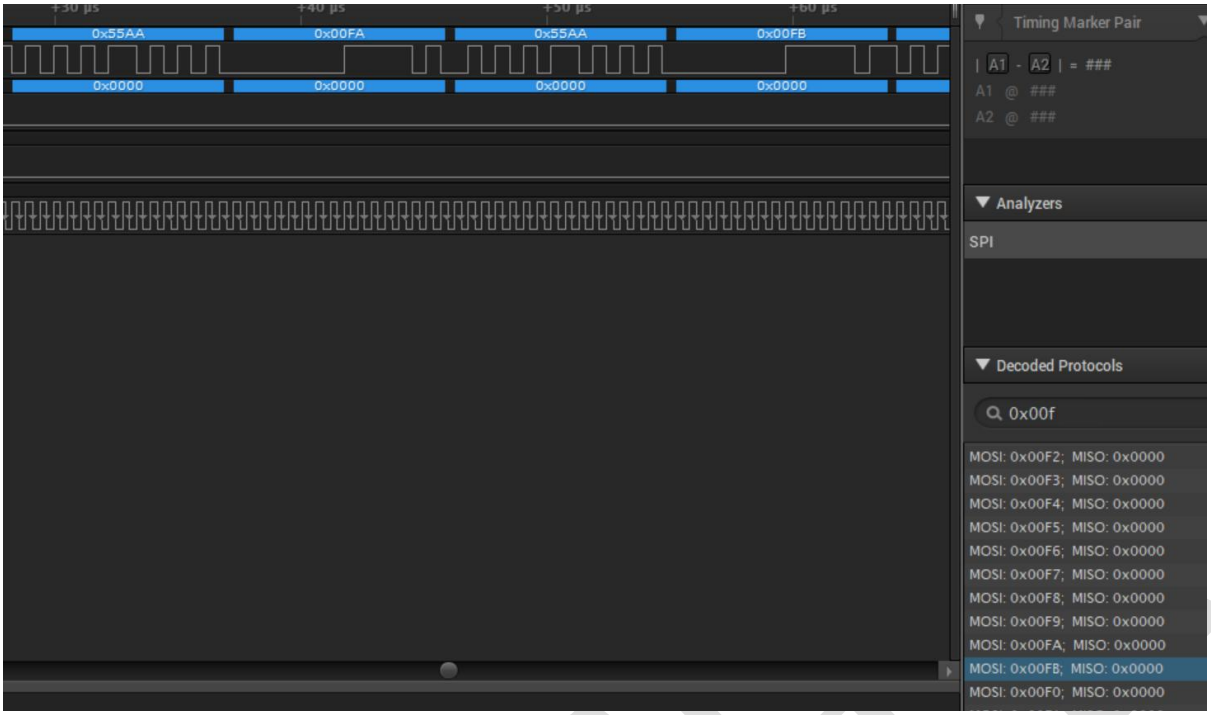
将 DMA 配置成 Memory Burst Length 为 8，验证传输 252 字（1008 字节）数据是否正常。

测试预期：

SPI 传输波形正常。

测试现象：

先打开逻辑分析仪软件，开启 SPI 波形抓取功能，然后在 Test Board 串口界面，输入‘3’命令，在 LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，SPI 成功发送了 5040 个半字（即 10080 个字节）出去。实际上，DMA 从 Memory 向 SPI 传输的数据范围是 0x55AA0000 ~ 0x55AA00FB，即重复传输 10 次，每次传输了 1008 个字节的数据，符合预期。

3.4.2.4 SPI 作为 Master 接收数据，Peripheral to Memory，配置一

测试目的：

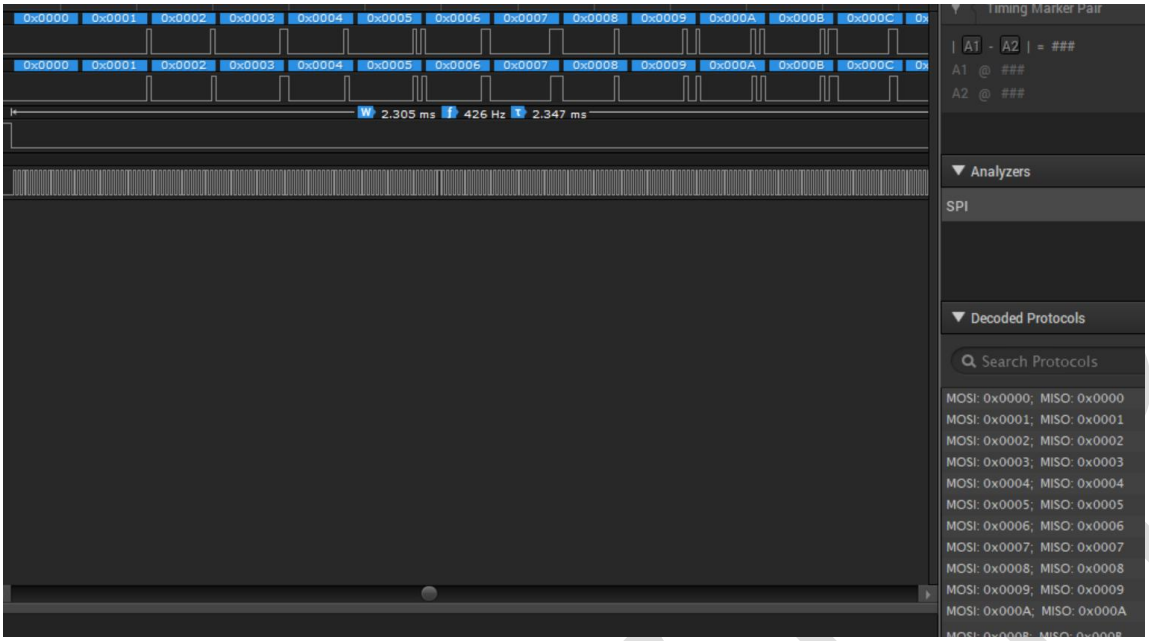
将 DMA 配置成 Memory Data Width 为 32，验证接收 256 个半字（512 字节）数据是否正常。

测试预期：

SPI 传输波形正常。

测试现象：

先打开逻辑分析仪软件，开启 SPI 波形抓取功能，然后在 Test Board 串口界面，输入‘4’命令，在 LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，SPI0 作为 Master，成功发送和接收了 2560 个半字（即 5120 个字节）。其中，接收数据的时候，DMA 从 SPI 向 Memory 传输的数据范围是 0x0000～0x00FF，即重复传输 10 次，每次传输了 512 个字节的数据，符合预期。

3.4.2.5 SPI 作为 Master 接收数据，Peripheral to Memory，配置二

测试目的：

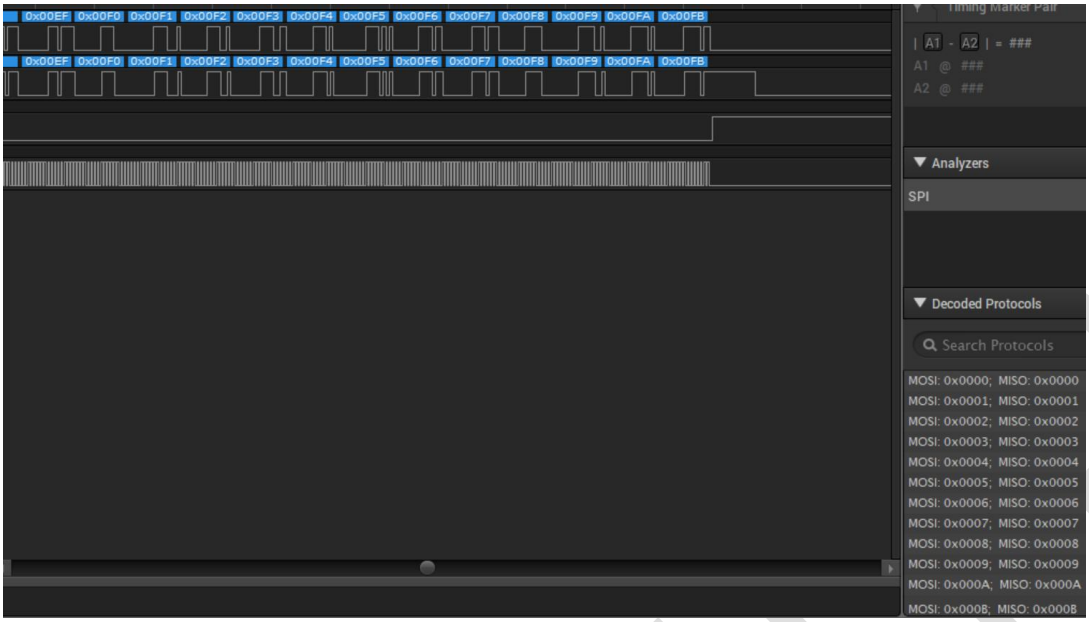
将 DMA 配置成 Memory Data Width 为 8，验证接收 252 个半字（504 字节）数据是否正常。

测试预期：

SPI 传输波形正常。

测试现象：

先打开逻辑分析仪软件，开启 SPI 波形抓取功能，然后在 Test Board 串口界面，输入‘5’命令，在 LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，SPI0 作为 Master，成功发送和接收了 2520 个半字（即 5040 个字节）。其中，接收数据的时候，DMA 从 SPI 向 Memory 传输的数据范围是 0x0000～0x00FB，即重复传输 10 次，每次传输了 504 个字节的数据，符合预期。

3.4.2.6 SPI 作为 Master 接收数据，Peripheral to Memory，配置三

测试目的：

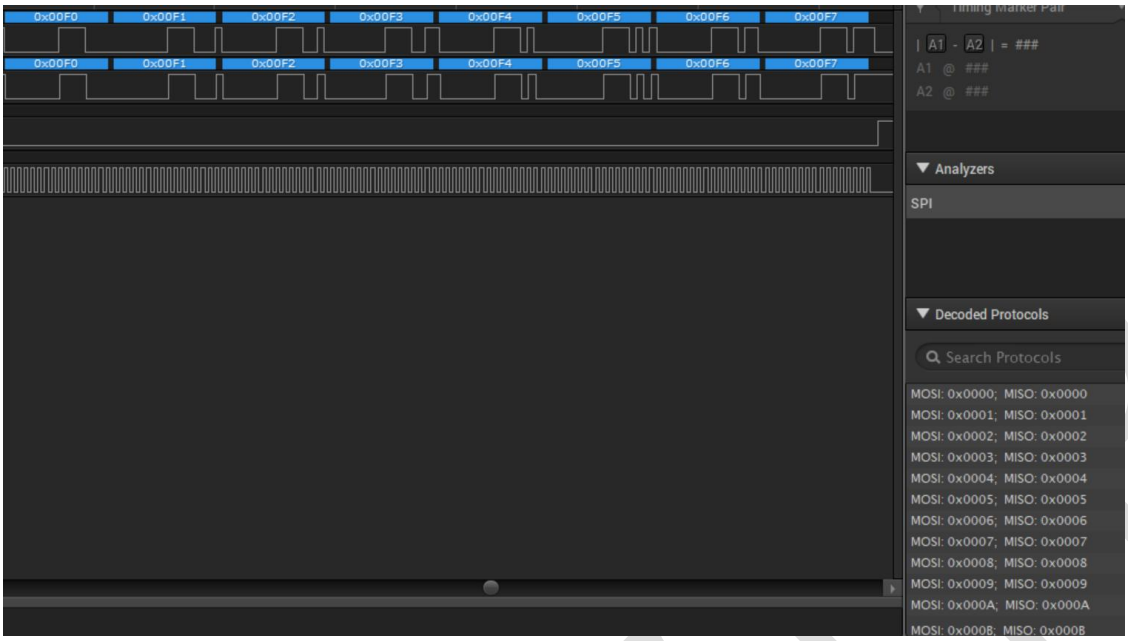
将 DMA 配置成 Memory Data Width 为 32，验证接收 248 个半字（496 字节）数据是否正常。

测试预期：

SPI 传输波形正常。

测试现象：

先打开逻辑分析仪软件，开启 SPI 波形抓取功能，然后在 Test Board 串口界面，输入‘6’命令，在 LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，SPI0 作为 Master，成功发送和接收了 2480 个半字（即 4960 个字节）。其中，接收数据的时候，DMA 从 SPI 向 Memory 传输的数据范围是 0x0000 ~ 0x00F7，即重复传输 10 次，每次传输了 496 个字节的数据，符合预期。

3.4.3 SDIO 模块，使用 DMA 方式收发数据

PAN1070 无 SDIO 模块！

3.4.4 I2C 模块，使用 DMA 方式收发数据

3.4.4.1 I2C 作为 Master 发送数据，Memory to Peripheral

测试目的：

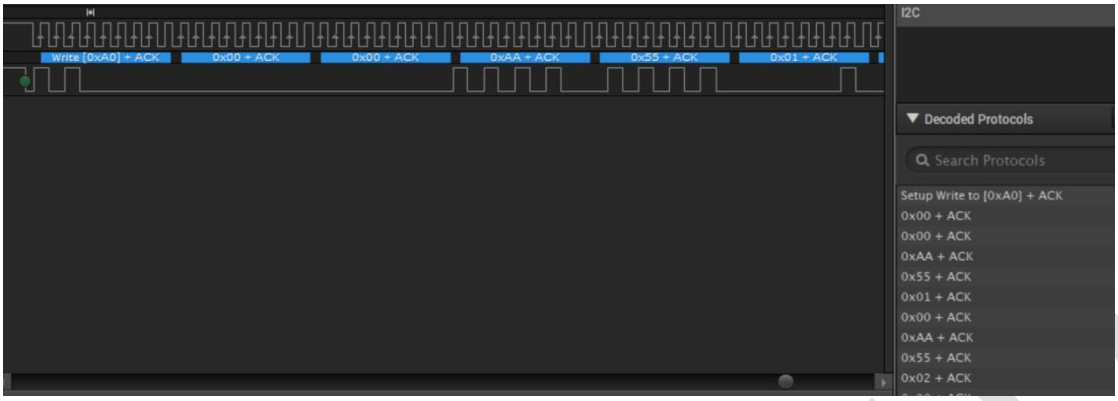
验证 I2C 使用 DMA 方式发送数据是否正常。

测试预期：

I2C 发送数据正常。

测试现象：

先打开逻辑分析仪软件，开启 I2C 波形抓取功能，然后在 Test Board 2 串口界面，输入‘9’命令，在 Test Board 1 串口界面，输入‘8’命令，在 LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，Test Board 1 的 I2C 作为 Master，以 96KHz 左右的时钟频率向总线上的 0xA0 地址发送了 64 个字（256 字节）的数据，范围为 0x55aa0000 ~ 0x55aa003f，符合预期。

3.4.4.2 I2C 作为 Slave 接收数据，Peripheral to Memory

测试目的：

验证 I2C 使用 DMA 方式接收数据是否正常。

测试预期：

I2C 接收数据正常。

测试现象：

先打开串口助手，然后在 Test Board 1 串口界面，输入‘9’命令，在 Test Board 2 串口界面，输入‘8’命令，发现 Test Board 1 打印如下 Log：

```
9
00 00 aa 55 01 00 aa 55 02 00 aa 55 03 00 aa 55 04 00 aa 55 05 00 aa 55 06 00 aa
 55 07 00 aa 55 08 00 aa 55 09 00 aa 55 0a 00 aa 55 0b 00 aa 55 0c 00 aa 55 0d 0
0 aa 55 0e 00 aa 55 0f 00 aa 55 10 00 aa 55 11 00 aa 55 12 00 aa 55 13 00 aa 55
14 00 aa 55 15 00 aa 55 16 00 aa 55 17 00 aa 55 18 00 aa 55 19 00 aa 55 1a 00 aa
 55 1b 00 aa 55 1c 00 aa 55 1d 00 aa 55 1e 00 aa 55 1f 00 aa 55 20 00 aa 55 21 0
0 aa 55 22 00 aa 55 23 00 aa 55 24 00 aa 55 25 00 aa 55 26 00 aa 55 27 00 aa 55
28 00 aa 55 29 00 aa 55 2a 00 aa 55 2b 00 aa 55 2c 00 aa 55 2d 00 aa 55 2e 00 aa
 55 2f 00 aa 55 30 00 aa 55 31 00 aa 55 32 00 aa 55 33 00 aa 55 34 00 aa 55 35 0
0 aa 55 36 00 aa 55 37 00 aa 55 38 00 aa 55 39 00 aa 55 3a 00 aa 55 3b 00 aa 55
3c 00 aa 55 3d 00 aa 55 3e 00 aa 55 3f 00 aa 55
```

测试分析：

测试例程 Testcase 9 中，将 I2C Rx 设置为 DMA 方式接收 256 个 Block 数据，每个 Block 为 8bit。由 Log 可知，I2C 成功从线上接收到 256 字节的数据，数据范围为 0x55aa0000 ~ 0x55aa003f，符合预期。

3.4.5 UART 模块，使用 DMA 方式收发数据

3.4.5.1 UART 发送数据，Memory to Peripheral

测试目的：

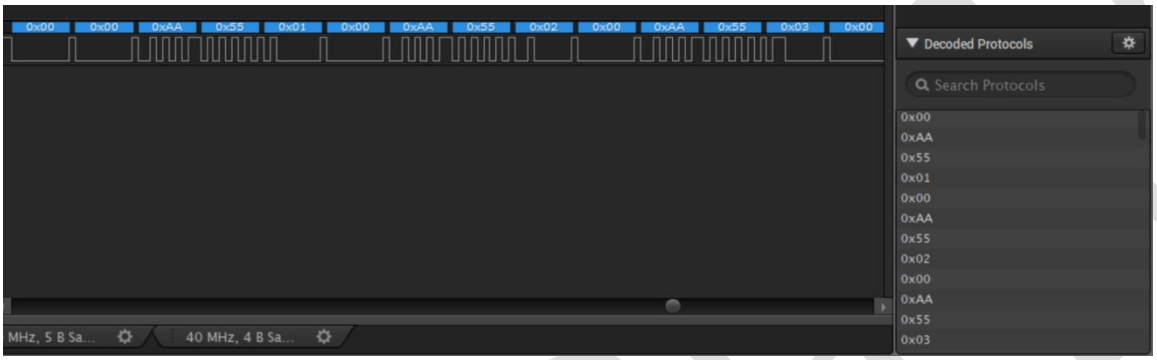
验证 UART 使用 DMA 方式发送数据是否正常。

测试预期：

UART1 发送数据正常。

测试现象：

先打开逻辑分析仪软件，开启 UART 波形抓取功能，然后在 Test Board 串口界面，输入‘A’命令，在 LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，Test Board 的 UART1 向 Tx 线上发送了 64 个字（256 字节）的数据，范围为 0x55aa0000 ~ 0x55aa003f，另外，由于 UART1 同时作为 Log 口使用，因此实际线上的数据会远远多于 256 个字节，这是符合预期的。

3.4.5.2 UART 接收数据，Peripheral to Memory

测试目的：

验证 UART 使用 DMA 方式接收数据是否正常。

测试预期：

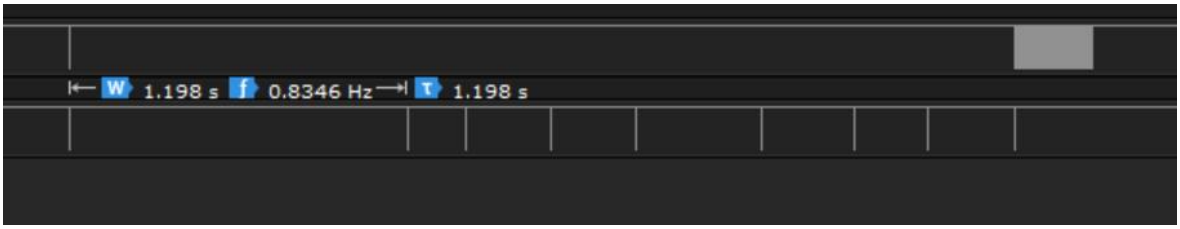
UART1 接收数据正常。

测试现象：

先打开逻辑分析仪软件，开启 UART 波形抓取功能，然后在 Test Board 串口界面，输入‘B’命令，在串口终端上提示输入要发送给目标板的字符串，这时候输入连续输入‘12345678’，在串口终端上看到打印如下 Log：

```
b
Please input data sending to UART...
12345678
0x31 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x32 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x33 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x34 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x35 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x36 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x37 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x38 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

同时，LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，Test Board 从 UART1 的 Rx 线上收到了 8 字节数据，范围为 0x31 ~ 0x38（即为数字 1~8 对应的 ASCII 码）；另外，由 Log 看到，从 DMA 接收了 64 字节的数据，其中每个 8 的倍数位置对应的值与 UART Rx 线上的一致，其余部分为 0，这是符合预期的——出现这种情况的原因是 UART1 Rx Trigger Level 没有配置，因此是默认值 0，表示 Rx FIFO 中只要出现 1 个字节的数据，即可触发 DMA 一次 Burst 传输，另一方面，DMA Source 端（即 UART 端）Burst Length 配置为 8，于是每次触发 DMA Burst 后，DMA 就会试图从 UART Rx FIFO 中读取 8 个字节的数据，存入 Memory 中，于是从 FIFO 中读到的数据，只有最外端的 1 个字节是有效数据，其余 7 个字节均为 0。若要避免此情况，可以将 UART Rx Trigger Level 配置为 Half Full（即 8 字节），或者将 DMA Source Burst Length 配置为 1 即可。

3.4.6 Gather 模式

DMAC 暂不支持此模式！

3.4.7 Scatter 模式

DMAC 暂不支持此模式！

3.4.8 DMA Peripheral to Peripheral 方式传输数据

3.4.8.1 将数据从 I2C 搬移至 UART

测试目的：

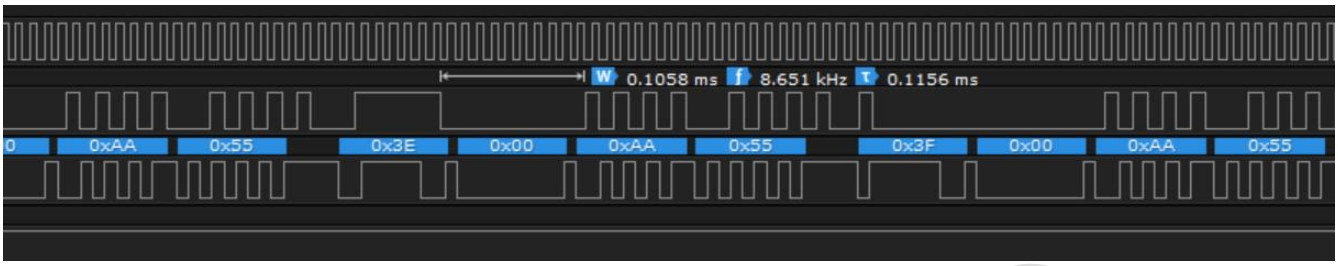
验证 DMA Peripheral to Peripheral 方式数据是否正常——I2C to UART 通路。

测试预期：

使用 DMA 方式将数据从 I2C 搬移至 UART 行为正常。

测试现象：

先打开逻辑分析仪软件，同时开启 UART 和 I2C 波形抓取功能，然后在 Test Board 1 串口界面，输入‘E’命令，在 Test Board 2 串口界面，输入‘8’命令，LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，Test Board 2 从 I2C 端口发送了 64 个字（256 字节）的数据，范围为 0x55aa0000 ~ 0x55aa003f；测试例程中，使用 DMA 将 I2C 接收到的数据，转发至 UART；从 UART 线上的波形可见，其成功发送了 256 字节的数据，内容也与 I2C 发送过来的一致，符合预期。

3.4.8.2 将数据从 I2C 搬移至 SPI

测试目的：

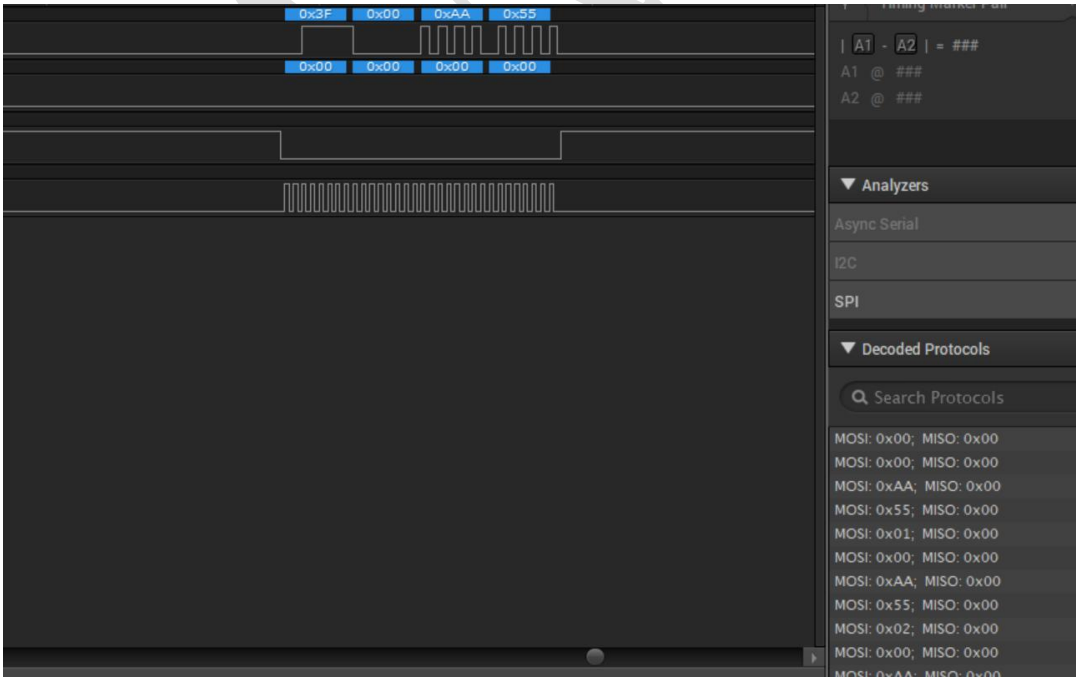
验证 DMA Peripheral to Peripheral 方式数据是否正常——I2C to SPI 通路。

测试预期：

使用 DMA 方式将数据从 I2C 搬移至 SPI 行为正常。

测试现象：

先打开逻辑分析仪软件，同时开启 I2C 和 SPI 波形抓取功能，然后在 Test Board 1 串口界面，输入‘F’命令，在 Test Board 2 串口界面，输入‘8’命令，LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，Test Board 2 从 I2C 端口发送了 64 个字（256 字节）的数据，范围为 0x55aa0000 ~ 0x55aa003f；测试例程中，使用 DMA 将 I2C 接收到的数据，转发至 SPI；从 SPI 线上的波形可见，其成功发送了 256 字节的数据，内容也与 I2C 发送过来的一致，并且由于 DMA Burst Length 配置为 4，数据宽度为 8bit，即 I2C 每接收到 4 字节数据才会触发 DMA 搬运，因此 SPI 线上的数据以 4 字节一组的方式向外发送，这也是符合预期的。

3.4.8.3 将数据从 SPI 搬移至 UART

测试目的：

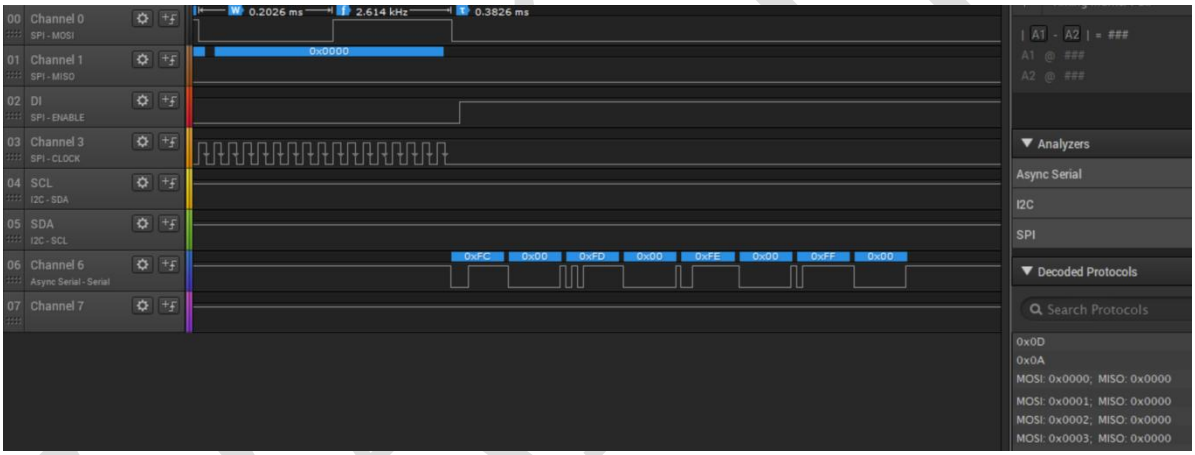
验证 DMA Peripheral to Peripheral 方式数据是否正常——SPI to UART 通路。

测试预期：

使用 DMA 方式将数据从 SPI 搬移至 UART 行为正常。

测试现象：

先打开逻辑分析仪软件，同时开启 UART 和 SPI 波形抓取功能，然后在 Test Board 串口界面，输入‘G’命令，LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，Test Board SPI1 从同一块 Board 的 SPI0 接口收到 256 个半字（512 字节）的数据，范围为 0x0000 ~ 0x00ff；测试例程中，使用 DMA 将 SPI1 接收到的数据，转发至 UART1；而从 UART 线上的波形可见，其成功发送了 512 字节的数据，内容也与 SPI 发送过来的一致；另外由于 DMA Source 端（SPI 端）Burst Length 配置为 4，数据宽度为 16bit，即 SPI 每接收到 4 个半字数据才会触发 DMA 搬运，因此 UART 线上的数据以 8 字节一组的方式向外发送，这也是符合预期的。

3.4.9 DMA 配置后连续多次开启通道传输数据

测试目的：

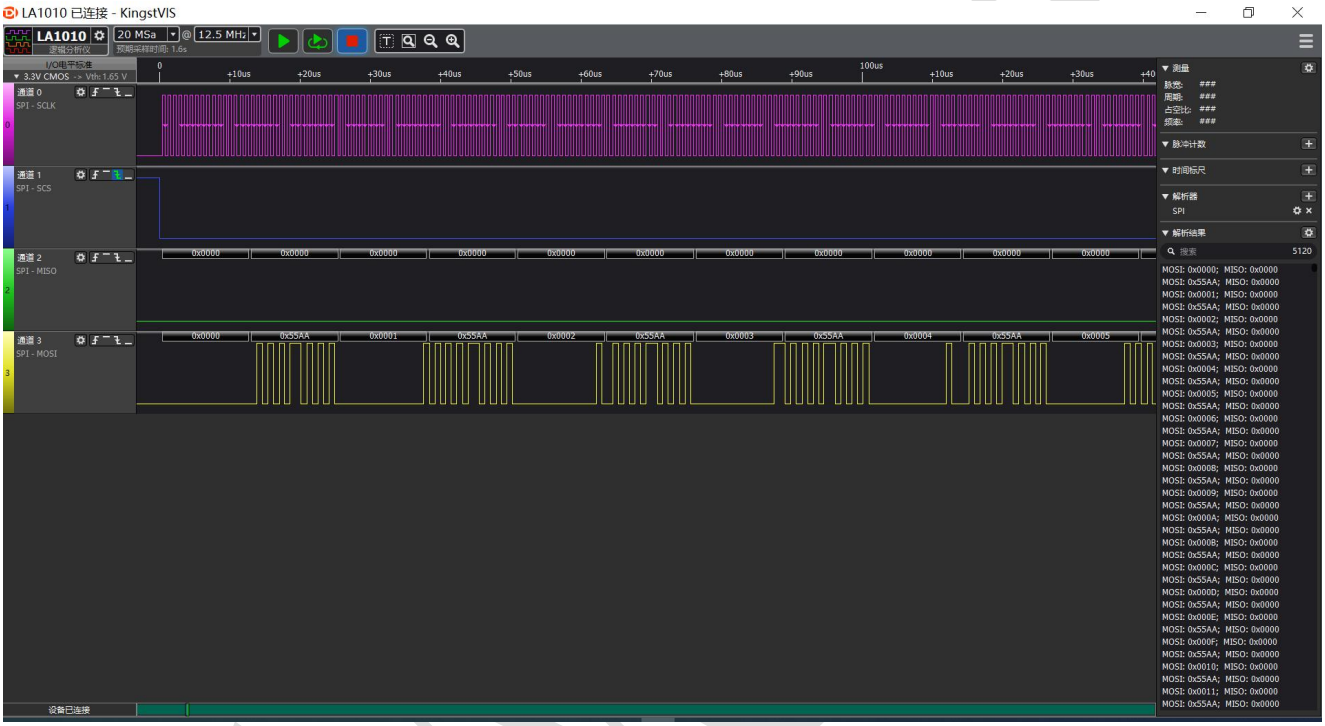
验证 DMA 在传输完成但 Channel 未释放的情况下，立刻再次开启通道传输数据是否正常。

测试预期：

SPI 作为 Master 发送数据波形正常。

测试现象：

先打开逻辑分析仪软件，开启 SPI 波形抓取功能，然后在 Test Board 串口界面，输入‘h’命令，在 LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，SPI 成功发送了 5120 个半字（即 10240 个字节）出去。实际上，DMA 从 Memory 向 SPI 传输的数据范围是 0x55AA0000 ~ 0x55AA00FF，即重复传输 10 次，每次传输了 1024 个字节的数据，符合预期。

3.4.10 DMA 多通道传输数据

3.4.10.1 同时使用 2 个通道

测试目的：

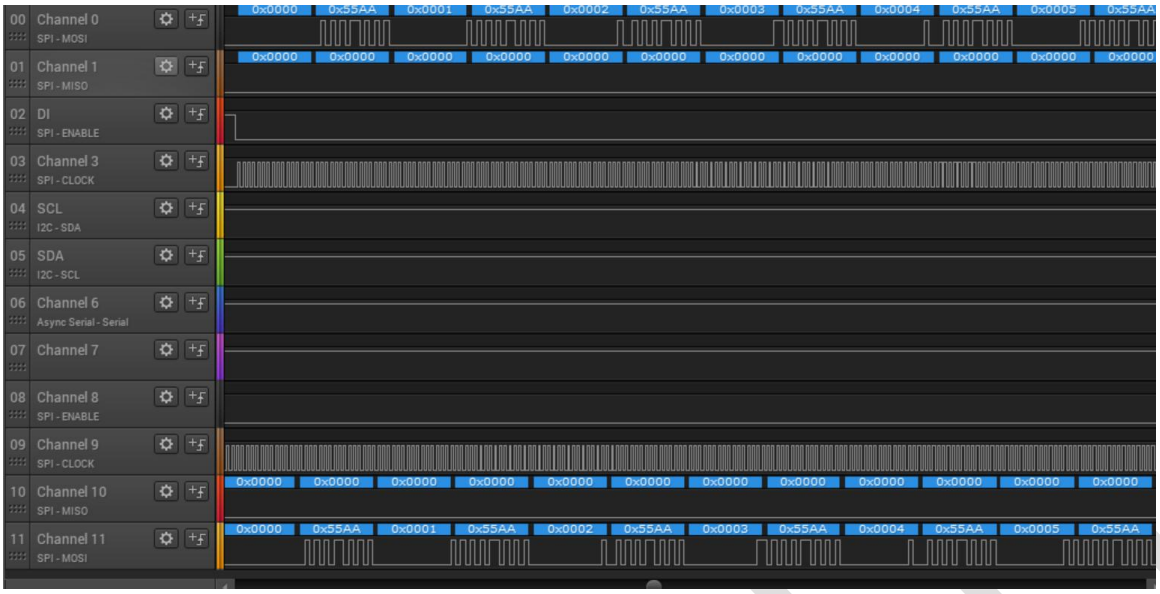
验证 2 个 DMA 通道分别同时被不同 Module 使用时，工作是否正常。

测试预期：

SPI0 与 SPI1 均作为 Master，分别使用 DMA 不同的 Channel 同时发送数据，波形正常。

测试现象：

先打开逻辑分析仪软件，开启 SPI 波形抓取功能，然后在 Test Board 串口界面，输入‘i’命令，在 LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，

1、SPI0 作为 Master，成功发送了 5120 个半字，在发送数据的时候，DMA 从 Memory 向 SPI 传输的数据范围是 0x55aa0000 ~ 0x55aa00FF，即重复传输 10 次，每次传输了 512 个半字的数据；

2、SPI1 作为 Master，成功发送了 1280 个半字，在发送数据的时候，DMA 从 Memory 向 SPI 传输的数据范围是 0x55aa0000 ~ 0x55aa003F，即重复传输 10 次，每次传输 128 个半字的数据；

另外，SPI1 比 SPI0 的 Clock 稍早一些发出，原因是测试例程中，SPI_SetDmaRxTxEnabled() 的调用时机上，SPI1 稍早一些。综上，从观察到的现象来看，DMA 行为均是符合预期的。

3.4.10.2 同时使用 3 个通道

DMAC 暂不支持 3 个通道！

3.4.11 DMA Lock 功能

3.4.11.1 Bus Lock

DMAC 暂不支持此功能！

3.4.11.2 Channel Lock

DMAC 暂不支持此功能！

3.4.11.3 Bus and Channel Lock

DMAC 暂不支持此功能！

3.4.12 DMA 配置通道优先级

测试目的：

验证 3 个 DMA 通道分别同时被不同 Module 使用时，配置成不同优先级的情况下，行为是否正常。

测试预期：

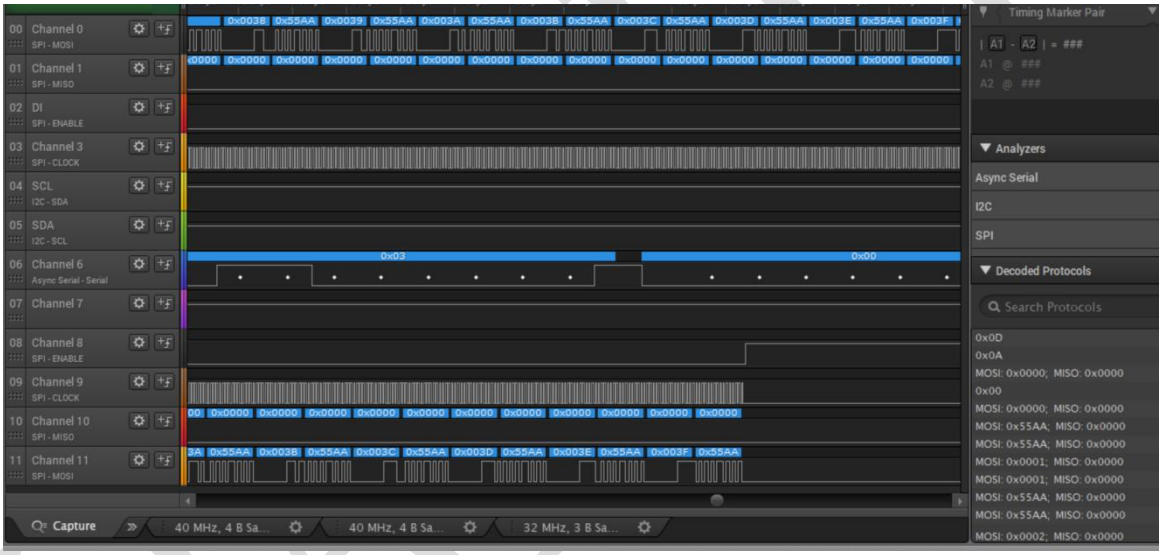
SPI0 作为 Master 发送数据，将使用的 DMA 通道配置成较高的优先级；

SPI1 作为 Master 发送数据，将使用的 DMA 通道配置成次高的优先级；

同时 UART1 发送数据，将使用的 DMA 通道配置成较低的优先级，输出波形正常。

测试现象：

先打开逻辑分析仪软件，开启 SPI 和 UART 波形抓取功能，然后在 Test Board 串口界面，输入‘n’命令，在 LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，SPI0 、SPI1 与 UART1 成功同时向外发送了数据，循环了 10 次。虽然理论上每次应该是高优先级的通道（SPI 占用）发送完成，DMA 才会空出时间给低优先级的通道（UART）使用，但由于 SPI 与 UART 均有 FIFO，因此即使 DMA 通道占用是分时的，但由于 DMA 将数据从 Memory 搬到外设 FIFO 所需时间极短，而外设将数据从 FIFO 发送至 IO 端口所需的时间较长，因此使用逻分从线上抓到的数据都是连续未间断的，这是符合预期的。实际上，本 Testcase 从 IO 端口抓波形是无法观察到通道优先级影响的，芯片内部仿真才可以观察到。

3.4.13 DMA 配置后连续多次开启多个通道传输数据

测试目的：

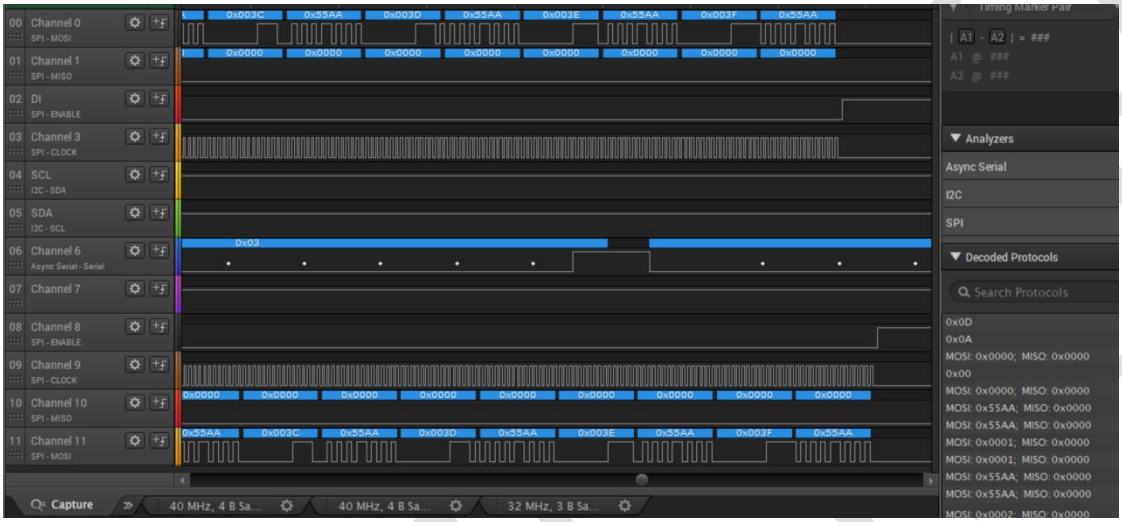
验证 2 个 DMA 通道分别同时被不同 Module 使用时，在不 Release Channel 的情况下连续多次开启通道传输，行为是否正常。

测试预期：

SPI0 作为 Master 发送数据，同时 UART1 发送数据，在 2 个通道传输完一轮数据后不 Release 的情况下，再次开启通道传输数据，输出波形正常。

测试现象：

先打开逻辑分析仪软件，开启 SPI 和 UART 波形抓取功能，然后在 Test Board 串口界面，输入‘o’命令，在 LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，SPI0、SPI1 与 UART1 成功同时向外发送了数据，循环了 10 次，每次传输的数据均正确，符合预期。

3.4.14 DMA Memory to Memory 方式传输数据

测试目的：

验证 DMA 将数据从一个 Memory 地址搬移至另一个 Memory 地址时，工作是否正常。

测试预期：

使用 DMA 将数据从一块 RAM 地址搬移至另一块 RAM 地址，工作正常。

测试现象：

先打开串口助手，然后在 Test Board 串口界面，输入‘p’命令，看到串口打印如下信息：

```
p
Data Sent:

55aa0000 55aa0001 55aa0002 55aa0003 55aa0004 55aa0005 55aa0006 55aa0007
55aa0008 55aa0009 55aa000a 55aa000b 55aa000c 55aa000d 55aa000e 55aa000f
55aa0010 55aa0011 55aa0012 55aa0013 55aa0014 55aa0015 55aa0016 55aa0017
55aa0018 55aa0019 55aa001a 55aa001b 55aa001c 55aa001d 55aa001e 55aa001f
55aa0020 55aa0021 55aa0022 55aa0023 55aa0024 55aa0025 55aa0026 55aa0027
55aa0028 55aa0029 55aa002a 55aa002b 55aa002c 55aa002d 55aa002e 55aa002f
55aa0030 55aa0031 55aa0032 55aa0033 55aa0034 55aa0035 55aa0036 55aa0037
55aa0038 55aa0039 55aa003a 55aa003b 55aa003c 55aa003d 55aa003e 55aa003f

Data Rcvd:

55aa003f 55aa003e 55aa003d 55aa003c 55aa003b 55aa003a 55aa0039 55aa0038
55aa0037 55aa0036 55aa0035 55aa0034 55aa0033 55aa0032 55aa0031 55aa0030
55aa002f 55aa002e 55aa002d 55aa002c 55aa002b 55aa002a 55aa0029 55aa0028
55aa0027 55aa0026 55aa0025 55aa0024 55aa0023 55aa0022 55aa0021 55aa0020
55aa001f 55aa001e 55aa001d 55aa001c 55aa001b 55aa001a 55aa0019 55aa0018
55aa0017 55aa0016 55aa0015 55aa0014 55aa0013 55aa0012 55aa0011 55aa0010
55aa000f 55aa000e 55aa000d 55aa000c 55aa000b 55aa000a 55aa0009 55aa0008
55aa0007 55aa0006 55aa0005 55aa0004 55aa0003 55aa0002 55aa0001 55aa0000
```

测试分析：

测试例程中，使用 DMA 将 RAM 中的数据以 Word 为单位搬移至另一块 RAM 地址中，其中源地址变化规则配置成递增，目的地址变化规则配置成递减。源地址中的数据为“0x55AA0000 ~ 0x55AA003F”，使用 DMA 搬移之后，相当于将数据以 Word 为单位进行反序，因此目的地址中的数据范围应该变为“0x55AA003F ~ 0x55AA0000”，从 Log 可以看出，搬移后的数据是符合预期的。

3.4.15 DMA 传输 1 Byte 数据

测试目的：

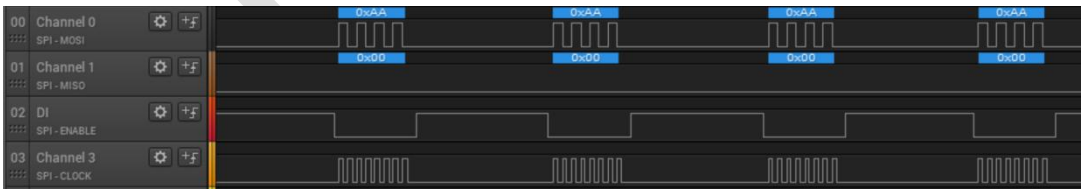
验证极端情况，使用 DMA 只传输 1 个字节的行为是否正常。

测试预期：

DMA 可以成功传输单个字节的数据，输出波形正常。

测试现象：

先打开逻辑分析仪软件，开启 SPI 波形抓取功能，然后在 Test Board 串口界面，输入‘q’命令，在 LA 软件上抓到如下波形：



测试分析：

由 LA 抓到的波形可见，SPI0 成功向外发送了 1 个字节的数据（0xAA），重复了 10 次，每次传输的数据均正确，符合预期。

3.4.16 DMA 传输 4092 Bytes 数据

测试目的：

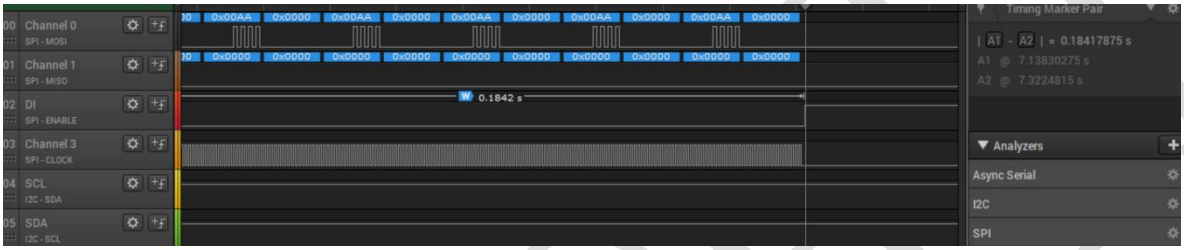
验证极端情况，使用 DMA 传输 4092 个 Bytes（即 1023 个 word）的行为是否正常。

测试预期：

DMA 可以成功传输 4092 个 Bytes 的数据，输出波形正常。

测试现象：

先打开逻辑分析仪软件，开启 SPI 波形抓取功能，然后在 Test Board 串口界面，输入‘r’命令，在 LA 软件上抓到如下波形：



测试分析：

测试例程中，SPI 数据位宽配置成 16 bit，Memory 数据位宽配置成 32 bit，传输的数据固定为 0xAA，即等价于将 0xAA 按照 Word 宽度传输 1023 遍，然后将此过程重复 10 次。

由 LA 抓到的波形可见，SPI0 成功向外发送了 4092 个字节的数据（0x000000AA），重复了 10 次，每次传输的数据均正确，符合预期。

3.4.17 DMA 全双工传输（SPI Tx Rx 同时使用 DMA）

测试目的：

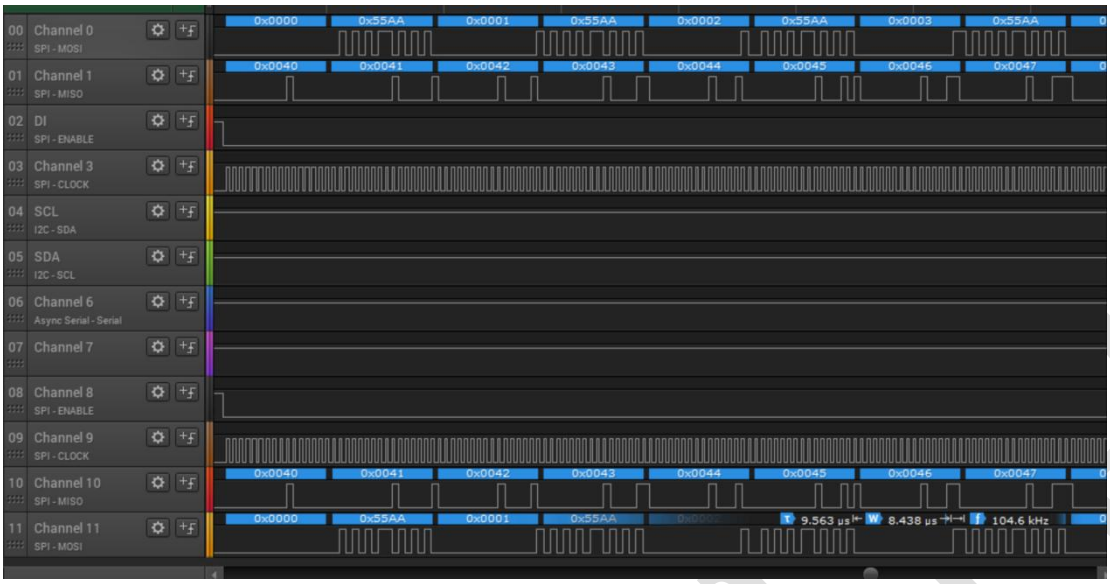
验证同一个 Module Tx Rx 同时使用 DMA 传输时，行为是否正常。

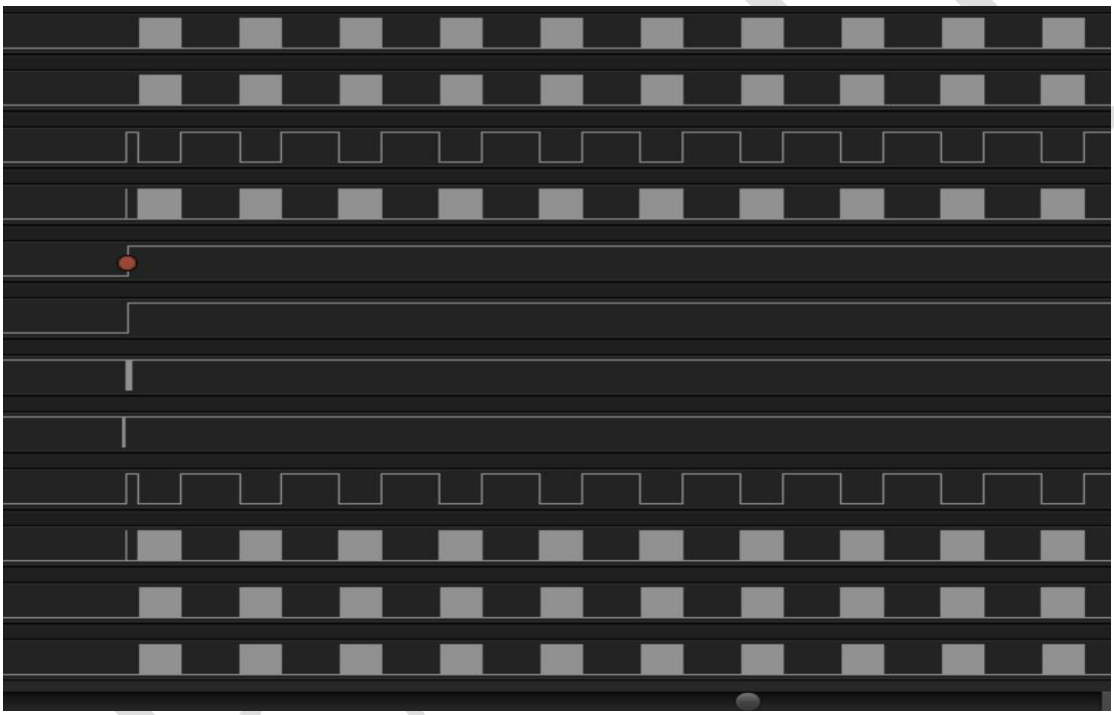
测试预期：

DMA 可以成功在特定 Module 上进行全双工传输。

测试现象：

先打开逻辑分析仪软件，开启 SPI 波形抓取功能，然后在 Test Board 串口界面，输入‘s’命令，在 LA 软件上抓到如下波形：





测试分析：

测试例程中，SPI 数据位宽配置成 16 bit，Memory 数据位宽配置成 32 bit，Master（SPI0）使用 DMA 方式向 Slave（SPI1）传输 64 个 Word，数据范围为“0x55AA0000 ~ 0x55AA003F”，Slave 使用 Polling 方式向 Master 传输 32 个半字，数据范围为“0x0040 ~ 0x005F”，然后 Master 使用 DMA 方式接收 Slave 端发送过来的数据。

由 LA 抓到的波形可见，SPI0 成功传输了预期的数据，重复了 10 次的情况下，每次传输的数据均正确，符合预期。

3.4.18 DMA 硬件 multiblock 方式传输

测试目的：

验证使用硬件 multiblock 方式传输，行为是否正常。

测试预期：

DMA 可以以 src 或 dst 或两端同时自动 load 方式传输，传输 16 次。

测试现象：

先打开串口助手，然后在 Test Board 串口界面，输入‘t’命令，看到串口打印如下信息：

SRC 自动 load，DST 手动：

```
|    Press key to choose auto reload config :
|
|    Input '0'    src auto reload,dst configuration
|    Input '1'    dst auto reload,src configuration
|    Input '2'    src auto reload,dst auto reload
|
0ch0_block_cnt:2
ch0_block_cnt:3
ch0_block_cnt:4
ch0_block_cnt:5
ch0_block_cnt:6
ch0_block_cnt:7
ch0_block_cnt:8
ch0_block_cnt:9
ch0_block_cnt:10
ch0_block_cnt:11
ch0_block_cnt:12
ch0_block_cnt:13
ch0_block_cnt:14
ch0_block_cnt:15
ch0_block_cnt:1
Data Sent:

55aa0000 55aa0001 55aa0002 55aa0003 55aa0004 55aa0005 55aa0006 55aa0007
55aa0008 55aa0009 55aa000a 55aa000b 55aa000c 55aa000d 55aa000e 55aa000f
55aa0010 55aa0011 55aa0012 55aa0013 55aa0014 55aa0015 55aa0016 55aa0017
55aa0018 55aa0019 55aa001a 55aa001b 55aa001c 55aa001d 55aa001e 55aa001f
55aa0020 55aa0021 55aa0022 55aa0023 55aa0024 55aa0025 55aa0026 55aa0027
55aa0028 55aa0029 55aa002a 55aa002b 55aa002c 55aa002d 55aa002e 55aa002f
55aa0030 55aa0031 55aa0032 55aa0033 55aa0034 55aa0035 55aa0036 55aa0037
55aa0038 55aa0039 55aa003a 55aa003b 55aa003c 55aa003d 55aa003e 55aa003f

Data Rcvd:

55aa003f 55aa003e 55aa003d 55aa003c 55aa003b 55aa003a 55aa0039 55aa0038
55aa0037 55aa0036 55aa0035 55aa0034 55aa0033 55aa0032 55aa0031 55aa0030
55aa002f 55aa002e 55aa002d 55aa002c 55aa002b 55aa002a 55aa0029 55aa0028
55aa0027 55aa0026 55aa0025 55aa0024 55aa0023 55aa0022 55aa0021 55aa0020
55aa001f 55aa001e 55aa001d 55aa001c 55aa001b 55aa001a 55aa0019 55aa0018
```

DST 自动 load，SRC 手动：

```
|    Press key to choose auto reload config :
|
|    Input '0'    src auto reload,dst configuration
|    Input '1'    dst auto reload,src configuration
|    Input '2'    src auto reload,dst auto reload
|
1ch0_block_cnt:2
ch0_block_cnt:3
ch0_block_cnt:4
ch0_block_cnt:5
ch0_block_cnt:6
ch0_block_cnt:7
ch0_block_cnt:8
ch0_block_cnt:9
ch0_block_cnt:10
ch0_block_cnt:11
ch0_block_cnt:12
ch0_block_cnt:13
ch0_block_cnt:14
ch0_block_cnt:15
ch0_block_cnt:1
Data Sent:

55aa0000 55aa0001 55aa0002 55aa0003 55aa0004 55aa0005 55aa0006 55aa0007
55aa0008 55aa0009 55aa000a 55aa000b 55aa000c 55aa000d 55aa000e 55aa000f
55aa0010 55aa0011 55aa0012 55aa0013 55aa0014 55aa0015 55aa0016 55aa0017
55aa0018 55aa0019 55aa001a 55aa001b 55aa001c 55aa001d 55aa001e 55aa001f
55aa0020 55aa0021 55aa0022 55aa0023 55aa0024 55aa0025 55aa0026 55aa0027
55aa0028 55aa0029 55aa002a 55aa002b 55aa002c 55aa002d 55aa002e 55aa002f
55aa0030 55aa0031 55aa0032 55aa0033 55aa0034 55aa0035 55aa0036 55aa0037
55aa0038 55aa0039 55aa003a 55aa003b 55aa003c 55aa003d 55aa003e 55aa003f

Data Rcvd:

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

SRC 自动 load，DST 自动 load：

```
|    Press key to choose auto reload config :
|
|    Input '0'    src auto reload,dst configuration
|    Input '1'    dst auto reload,src configuration
|    Input '2'    src auto reload,dst auto reload
|
2ch0_block_cnt:1
ch0_block_cnt:2
ch0_block_cnt:3
ch0_block_cnt:4
ch0_block_cnt:5
ch0_block_cnt:6
ch0_block_cnt:7
ch0_block_cnt:8
ch0_block_cnt:9
ch0_block_cnt:10
ch0_block_cnt:11
ch0_block_cnt:12
ch0_block_cnt:13
ch0_block_cnt:14
ch0_block_cnt:15
ch0_block_cnt:1
Data Sent:

55aa0000 55aa0001 55aa0002 55aa0003 55aa0004 55aa0005 55aa0006 55aa0007
55aa0008 55aa0009 55aa000a 55aa000b 55aa000c 55aa000d 55aa000e 55aa000f
55aa0010 55aa0011 55aa0012 55aa0013 55aa0014 55aa0015 55aa0016 55aa0017
55aa0018 55aa0019 55aa001a 55aa001b 55aa001c 55aa001d 55aa001e 55aa001f
55aa0020 55aa0021 55aa0022 55aa0023 55aa0024 55aa0025 55aa0026 55aa0027
55aa0028 55aa0029 55aa002a 55aa002b 55aa002c 55aa002d 55aa002e 55aa002f
55aa0030 55aa0031 55aa0032 55aa0033 55aa0034 55aa0035 55aa0036 55aa0037
55aa0038 55aa0039 55aa003a 55aa003b 55aa003c 55aa003d 55aa003e 55aa003f

Data Rcvd:

55aa003f 55aa003e 55aa003d 55aa003c 55aa003b 55aa003a 55aa0039 55aa0038
55aa0037 55aa0036 55aa0035 55aa0034 55aa0033 55aa0032 55aa0031 55aa0030
55aa002f 55aa002e 55aa002d 55aa002c 55aa002b 55aa002a 55aa0029 55aa0028
55aa0027 55aa0026 55aa0025 55aa0024 55aa0023 55aa0022 55aa0021 55aa0020
55aa001f 55aa001e 55aa001d 55aa001c 55aa001b 55aa001a 55aa0019 55aa0018
```

测试分析：

Src 自动 load，dst 不自动 load 模式下，源数据不需要重新 load，dst 地址递增或递减，一旦超过 buf 大小可能会破坏堆栈，造成 hardfault，打印信息显示传输 16 次 block，符合预期

Dst 自动 load，src 不自动 load 模式下，目的地数据不需要重新 load，src 地址递增或递减，一旦超过 buf 大小可能会破坏堆栈，造成 hardfault，或者源端没有重新赋值则为 memory 默认值，打印信息显示传输 16 次 block，符合预期

Src 自动 load，dst 不自动 load 模式下，源数据不需要重新 load，目的地数据不需要重新 load，打印信息显示传输 16 次 block，符合预期

3.4.19 DMA Flash to SPI 方式传输数据

测试目的：

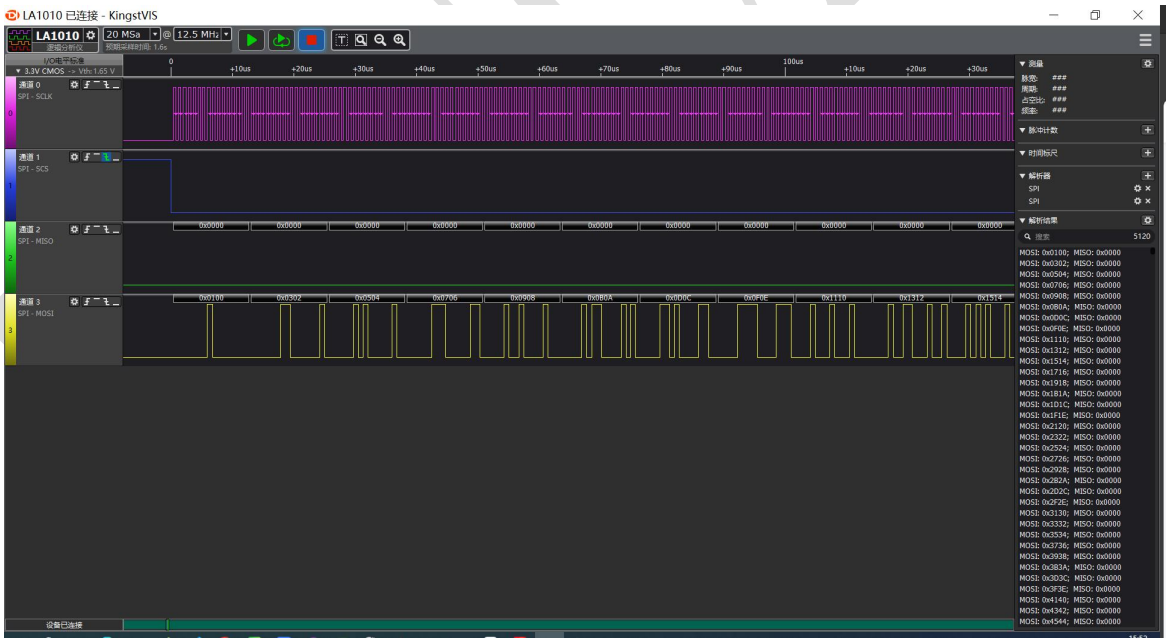
验证 DMA Memory to Peripheral 方式数据是否正常——Flash to SPI 通路。

测试预期：

SPI 传输波形正常。

测试现象：

先打开逻辑分析仪软件，开启 SPI 波形抓取功能，然后在 Test Board 串口界面，输入‘u’命令，在 LA 软件上抓到如下波形：



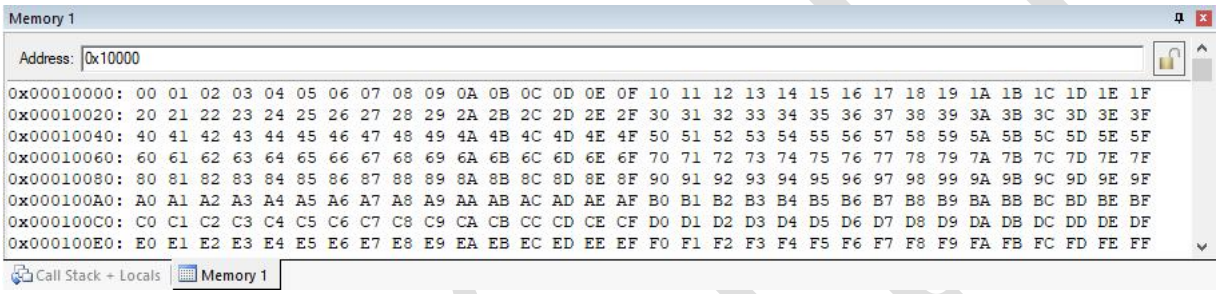
串口助手打印 LOG:

```
[15:38:48.866]发→◇V□
[15:38:48.873]收←◆

[15:38:48.927]收←◆10000 data is 3020100
10000 data is 3020100
10000 data is 3020100
10000 data is 3020100
10000 data is 3020100
10000 data is 3020100
10000 data is 3020100
10000 data is 3020100
10000 data is 3020100
10000 data is 3020100
```

测试分析：

Debug 工程查看目标 Memory:



由 LA 抓到的波形可见，SPI0 成功向外发送了目标 Memory 数据，循环了 10 次。

第4章 注意事项

4.1 SPI DMA 使用注意事项

1、SPI 初始化时 SPI 数据位宽 SPI_dataFrameSize（1）选择可为 8 或者 16，DMA 配置目标或者源为 SPI 时，DMA 通道位宽（2）最好与之相同，可能出现情况如下：

A）1 为 8，2 为 16：丢失一半的数据

B）1 为 16，2 为 8：不会丢失据，但是每次传输高 8bit 数据恒为 0

C）1 和 2 同为 8：正常传输，效率比较低

D）1 和 2 同为 16：正常传输，效率比较高，要求数据总量是 2 字节对齐，否则会出现无效的数据。

2、传输数据的总数据量（byte 数量）与 dma 的数据位宽必须为倍数关系，否则会出现数据丢失或者多出数据的情况，出现此种情况可降低 dma 的数据位宽来解决。

例如：需要传输 15 个 byte 的数据（memory 到外设），DMA 源位宽为 32bit，那么 block_ts 的值要么为 15/4=3（丢失 3 个数据），或者 15/4+1=4（多出一个 0 的数据），降低源位宽为 8bit，block_ts 设置为 15，可解决此问题，但会降低 DMA 的使用效率，对于数据量不大的 case，没什么影响，但对于数据量大且频繁请求 DMA 的 case，可能会限制 DMA 的使用效率。

3、SPI 发送给 UART 的 case 情况下，由于 UART 的输出速率比较慢，需要降低 SPI 的主机端的发送速率，否则 UART 会因为 FIFO 满的情况还在往 UART FIFO 发送数据，从而导致数据丢失。

4、全双工工作时，应尽量保证 TX 和 RX 同时开启 DMA，避免时钟产生时 RX 没有数据而导致数据丢失的情况。

4.2 I2C DMA 使用注意事项

1、I2C 设置的 DMA 发送接收 triglevel 等级需与 DMA 通道的 burstlenth 设置的一致，否则在接收数据的情况下可能出现丢失数据的情况，例如：

A、I2C_DMAReceiveDataLevel(I2Cx, I2C_RX_TL_3); I2C_RX_TL_3 表示 4 个数据触发 DMA 请求

B、TxConfigTmp1.BurstLenSrc = DMAC_BurstLen_1;1 次 DMA 请求传输一个宽度数据

以上 A、B 配置在传输 20 个字节的 case 下，I2C 端最后 3 个数据残留在 I2C FIFO 中不能触发 DMA 请求，导致丢失。

4.3 UART DMA 使用注意事项

与 I2C 类似，UART 设置的 DMA 发送接收 trigger level 同样需与 DMA 通道的 burst length 保持一致，否则可能得不到预期的结果。

第5章 测试结论

5.1 测试结论

Modular	Test Case	Test Result
DMAC	寄存器默认值	PASS
	SPI 作为 Master 发送数据，Memory to Peripheral，配置一	PASS
	SPI 作为 Master 发送数据，Memory to Peripheral，配置二	PASS
	SPI 作为 Master 发送数据，Memory to Peripheral，配置三	PASS
	SPI 作为 Master 接收数据，Peripheral to Memory，配置一	PASS
	SPI 作为 Master 接收数据，Peripheral to Memory，配置二	PASS
	SPI 作为 Master 接收数据，Peripheral to Memory，配置三	PASS
	I2C 作为 Master 发送数据，Memory to Peripheral	PASS
	I2C 作为 Slave 接收数据，Peripheral to Memory	PASS
	UART 发送数据，Memory to Peripheral	PASS
	UART 接收数据，Peripheral to Memory	PASS
	DMA Peripheral to Peripheral 方式传输数据，I2C to UART	PASS
	DMA Peripheral to Peripheral 方式传输数据，I2C to SPI	PASS
	DMA Peripheral to Peripheral 方式传输数据，SPI to UART	PASS
	DMA Single Channel Multi-block 传输	PASS
	DMA 两通道传输	PASS
	DMA 三通道传输	PASS
	DMA 优先级设定	PASS
	DMA Multi-channel Block 传输	PASS
	DMA Memory to Memory 方式传输数据	PASS
	DMA 传输 1 个 Byte 数据（Memory to SPI）	PASS
	DMA 传输 16380 个 Bytes 数据（Memory to SPI）	PASS
	DMA 同一模块收发数据均使用 DMA（Memory 2 SPI and SPI 2 Memory）	PASS
	DMA 硬件 multiblock 方式传输	PASS