飞思卡尔半导体
应用笔记

Document Number: AN4393
Rev. 0, 05/2012

将 FlexBus 接口用于 Kinetis 微控制器

Carlos Musich 和 Alejandro Lozano
技术信息中心

目录

1 简介

很多情况下，需要将外部器件连接到微控制器。例如，当某个应用程序所要求的内存量大于它所能提供的内存量时，就有必要使用外部内存。另一个例子是，当系统要求使用额外接口而微控制器无法提供时，需要使用外设。这些都需要 Kinetis 器件提供多功能外部总线接口（称为 FlexBus 接口控制器），并具有与只能作为从机的器件连接的功能。它可直接连接外部 ROM、Flash 存储器、可编程逻辑器件或其他简单的目标 （从机）器件，并且只需要很少甚至不需要额外的电路。

本应用笔记介绍如何在K60上设置并使用FlexBus接口，并给出一些实施示例。在这些示例中， TWR-K60 通过 FlexBus 连接到 TWR-MEM 和 TWR-LCD。TWR_MEM 用于演示如何从外部内存实现读、写和执行操作。TWR-LCD 用于体现与显示图像的 LCD 的连接性。尽管本应用笔记重点介绍 K60，但该文档中的信息也同样适用于具有 FlexBus 接口的 Kinetis 器件。

2 功能说明

外部接口是一个可配置多路复用总线，设为下列模式之一：

- 多路复用 32 位地址和 32 位数据
- 多路复用 32 位地址和 16 位数据 （非多路复用 16 位地址和 16 位数据）
- 多路复用 32 位地址和 8 位数据 （非多路复用 24 位地址和 8 位数据）
- 非多路复用 32 位地址和 32 位数据总线

最快的FlexBus传输要经过4个FlexBus时钟周期。本节介绍读周期和写周期中FlexBus使用的信号。

附注

Kinetis FlexBus 支持突发。如果连接支持突发的任意器件 （比如 FPGA），则传输数据的时间要快于 4 个 FlexBus 周期，因为它并非每次在数据传输之间都需要地址阶段。

2.1 读周期

1. 地址在第一个周期中驱动。非多路复用模式下， FB_A[31:0] 搭载地址。多路复用模式下，FB_AD[31:0] 总线搭载地址和数据。完整的 32 位地址在总线第一个时钟周期驱动 （地址阶段）。第一个时钟边沿和驱动地址的时刻之间的时间称为输出有效时间 (FB2)。
2. 非多路复用模式下，“传输开始 (FB_TS)”信号变为有效，表示器件开始执行一次总线传输。多路复用模式下，反相 FB_TS (FB_ALE) 用作地址锁存使能，表示 FB_AD 总线驱动地址的时间。这些信号在下一个时钟周期变为无效，但可配置为保留其有效状态，直到 FB_CS 变为有效后的首个上升沿。
3. 在第二个边沿处，对应的片选 (FB_CS) 和输出使能 (FB_OE) 信号变为有效。
4. 外部器件必须在第三个周期开始之前向总线提供数据。这段时间称为“输入建立时间”（对于 K60，FB4 = 13.5 ns）。如果外部器件不够快，则可添加“等待状态”（额外时钟周期）。

   如果自动应答功能禁用 (CSCRn[AA] = 0)，则在第三个周期开始之前， FB_TA 的无效状态必须持续 13.5 ns (FB4)。
5. 如果未添加 “等待状态”，则数据在第三个时钟沿读出。非多路复用模式下， FB_D[31:0] 总线搭载数据。多路复用模式下，地址锁存后，数据在 FB_AD[31:0] 总线上驱动（数据阶段）。数据阶段期间，没有作为数据的引脚，地址驱动会继续保持。例如，在 16 位模式下，低位地址继续在 FB_AD[15:0] 上驱动，而在 8 位模式下低位地址继续在 FB_AD[23:0] 上驱动。
6. 数据在总线上必须至少保持 0.5 ns。这一时间称为 “输入保持” (FB5)。
7. 最后， FB_TA 表示外部数据传输完成。在读周期中，当处理器识别出 FB_TA 时，它锁存数据，然后中断总线周期。

   如果自动应答禁用 (CSCRn[AA] = 0)，则外部器件驱动 FB_TA 来中断总线传输；如果自动应答使能 (CSCRn[AA] = 1)， FB_TA 将在指定数量的等待状态过后由内部生成，否则外部器件可能在等待状态向下计数完毕前使 FB_TA 变为有效，从而提前中断周期。设备将在 FB_TA 的最后一个有效状态的一个周期后使 FB_CSn 无效。在读周期中，外设必须继续驱动数据，直到识别 FB_TA。参见图 1。

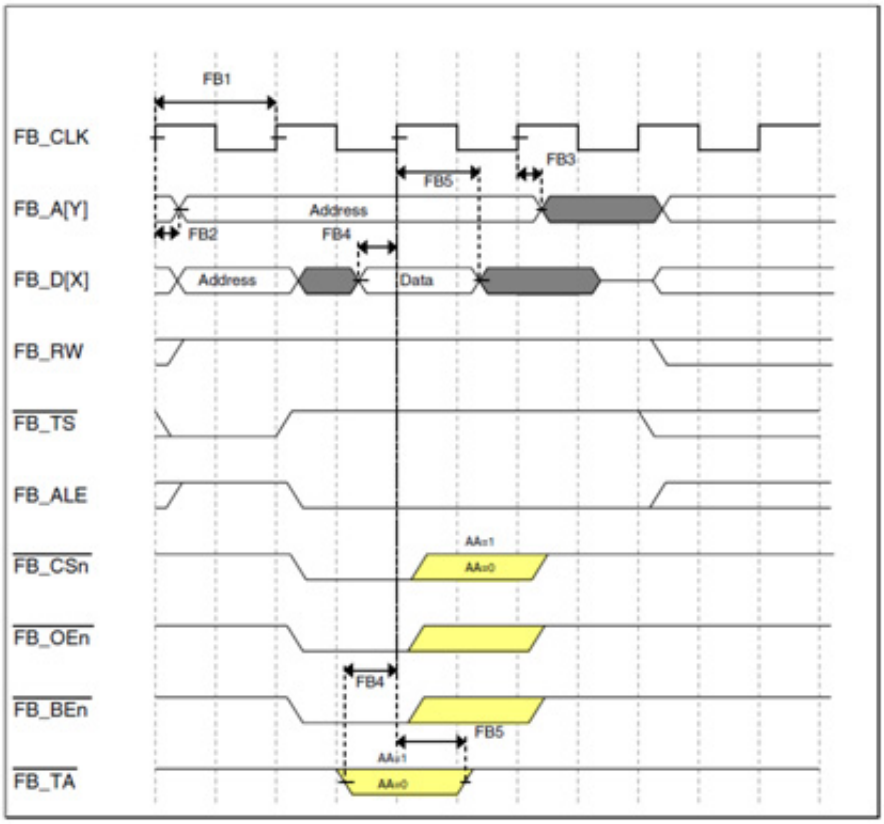


图 1. FlexBus 读周期

2.2 写周期

FlexBus 读周期和写周期时序基本一致。主要区别在于：

1. FB_OE 永远不会变为有效
2. 在读周期中，当处理器识别出 FB_TA 时，便会中断总线周期。
3. 对于写周期，在 FB_CSn 无效以后一个时钟周期内，处理器会继续驱动数据。

参见图 2。

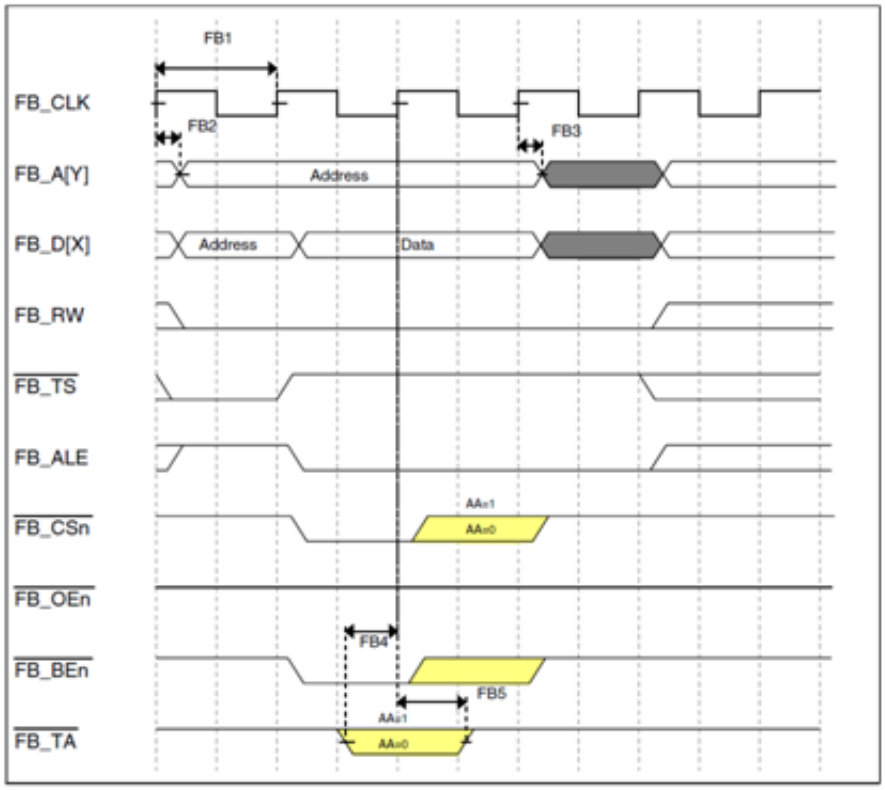


图 2. FlexBus 写周期

2.3 数据字节对齐和物理连接

器件在 FlexBus 字节通道中对齐数据传输，通道数取决于数据端口宽度。

图 3 显示当字节通道移位禁用或使能时，外部内存连接的字节通道，以及针对支持的端口大小进行 32 位顺序传输。

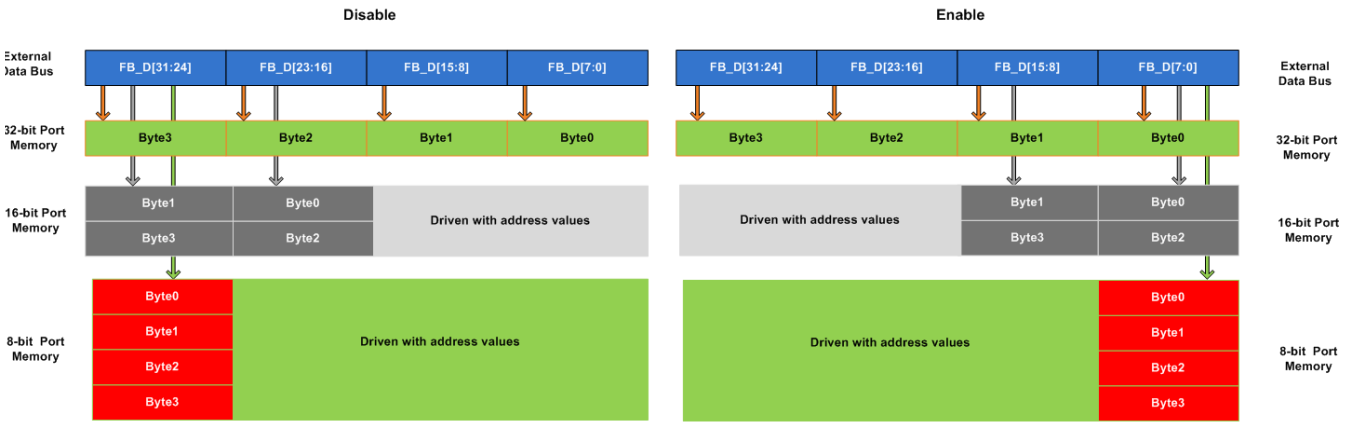


图 3. 顺序 32 位传输，字节通道移位不同

2.4 内存映射

如图4所示，典型内存映射为：0x6000_000 - 0xA000_0000是FlexBus空间，用于执行；0xA000_0000 - 0xE000_0000 只能用于数据。

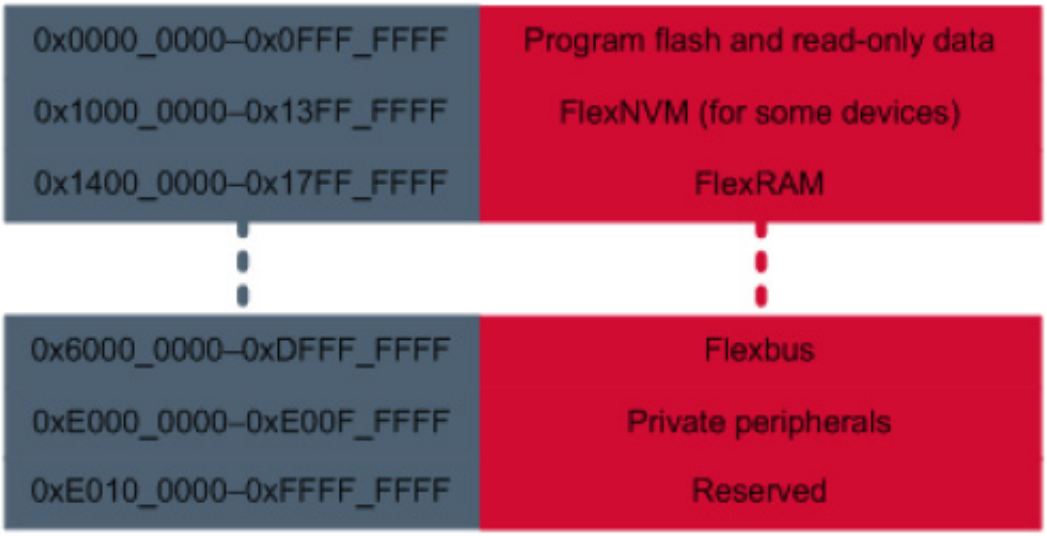


图 4. FlexBus 内存映射

3 示例

下文示例显示实现某些通用接口的方式。

附注

由于 TWR-MEM 模块的限制， MRAM 非多路复用和多路复用模式均采用 8 位数据。为了获得最佳性能，必须采用 16 位数据连接。

3.1 MRAM 非多路复用模式

在这些示例中， FlexBus 连接磁阻随机访问内存 (MRAM) 芯片，该芯片尺寸与业界标准 x16 异步快速 SRAM 相同。若需通过最小绝对值胶合逻辑连接 16 位存储器，则本例采用了非多路复用模式，具有 8 位数据和 20 位数据线路。

该解决方案采用 TWR-K60 评估板进行演示。

优势：

- 通过添加 512 KB MRAM 增大可用系统内存
- MRAM 是非易失性存储器，无需专用的擦除和编程例程
- 只需单个门控反相器

劣势：

- MRAM 的访问速度比 SRAM 慢，因此内存带宽也较低。
- 为了减少胶合逻辑，此方法将 MRAM 视为 8 位内存，从而提供的性能也不如 16 位。

3.1.1 原理图

尽管 MRAM 是 16 位的，但它是字节可读 / 写的。它具有高位字节使能和地位字节使能引脚，可在高字节和低字节之间进行选择。只需 FB_A0 上的单个门控反相器，便可从 FlexBus 信号构建出这些信号。 FB_A0 用作字节高电平使能，而反相器 FB_A0 的输出用作字节低电平使能。这样可以高效地将存储器作为 8 位器件使用。使用这种方法会牺牲性能。然而，这样可以最大程度减少胶合逻辑，节省元器件成本。

来自 FlexBus 的 8 根数据线同时连接 16 条 MRAM 数据通道的高位字节和低位字节。字节使能线路由 FB_A0 和非 FB_A0 来选择，选择哪一字节在数据线路上驱动。未选择时，MRAM 的其他字节线路将为三态，且不干扰目标字节。

存储器为字可访问（16 位），因此地址线路必须移位，使 FlexBus 字地址线路 FB_A[1] 连接存储器的 A[0]。因此， FB_A[18:1] 连接存储器的 A[17:0]。

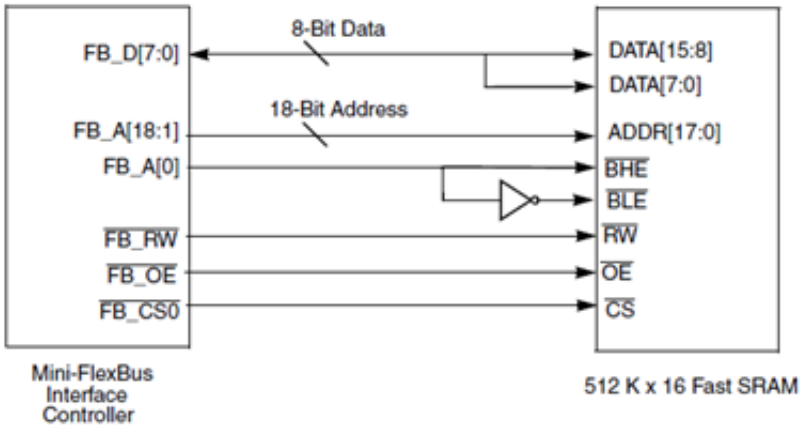


图 5. 非多路复用 x8 FlexBus 至 x16 存储器

3.1.2 软件设置

为确定 FlexBus 时序要求的调节，可参考参考手册中的 FlexBus 章节，标题为“*K60 子系列参考手册：采用 144 引脚封装的 100MHz 器件*”（文档编号：K60P144M100SF2RM）以及数据手册中的“时序规格”章节内容，标题为“K60 子系列数据手册”（文档编号：K60P144M100SF）。

- FlexBus 时钟 (FB_CLK) 运行速度与总线时钟相同。FB_CLK 频率可能与内部系统总线频率一致，也可能等于该频率整数分频后的值。其默认值为 50 MHz。

```
SIM_SCGC7 |= SIM_SCGC7_FLEXBUS_MASK; // Enable the clock to the FlexBus
SIM_CLKDIV1 |= SIM_CLKDIV1_OUTDIV3(0x0); //FlexBus Clock not divided
```

- FlexBus所需的IO引脚必须配置为FlexBus功能。一共有18个地址引脚，8个数据引脚，以及 4 个控制引脚。使用宏 PORT_PCR_MUX(5) 设置每一个引脚的 FlexBus 功能。必须设置 GPIO 端口时钟

```
// Set the GPIO ports clocks
SIM_SCGC5 = SIM_SCGC5_PORTA_MASK | SIM_SCGC5_PORTB_MASK |     SIM_SCGC5_PORTC_MASK |
SIM_SCGC5_PORTD_MASK | SIM_SCGC5_PORTE_MASK;

PORTB_PCR11 = PORT_PCR_MUX(5);            // fb_ad[18]
PORTB_PCR16 = PORT_PCR_MUX(5);            // fb_ad[17]
```

```
PORTB_PCR17 = PORT_PCR_MUX(5);          // fb_ad[16]
PORTB_PCR18 = PORT_PCR_MUX(5);          // fb_ad[15]
PORTC_PCR0 = PORT_PCR_MUX(5);           // fb_ad[14]
PORTC_PCR1 = PORT_PCR_MUX(5);           // fb_ad[13]
PORTC_PCR2 = PORT_PCR_MUX(5);           // fb_ad[12]
PORTC_PCR4 = PORT_PCR_MUX(5);           // fb_ad[11]
PORTC_PCR5 = PORT_PCR_MUX(5);           // fb_ad[10]
PORTC_PCR6 = PORT_PCR_MUX(5);           // fb_ad[9]
PORTC_PCR7 = PORT_PCR_MUX(5);           // fb_ad[8]
PORTC_PCR8 = PORT_PCR_MUX(5);           // fb_ad[7]
PORTC_PCR9 = PORT_PCR_MUX(5);           // fb_ad[6]
PORTC_PCR10 = PORT_PCR_MUX(5);          // fb_ad[5]
PORTD_PCR2 = PORT_PCR_MUX(5);           // fb_ad[4]
PORTD_PCR3 = PORT_PCR_MUX(5);           // fb_ad[3]
PORTD_PCR4 = PORT_PCR_MUX(5);           // fb_ad[2]
PORTD_PCR5 = PORT_PCR_MUX(5);           // fb_ad[1]
PORTD_PCR6 = PORT_PCR_MUX(5);           // fb_ad[0]

PORTB_PCR20 = PORT_PCR_MUX(5);          // fb_ad[31] used as d[7]
PORTB_PCR21 = PORT_PCR_MUX(5);          // fb_ad[30] used as d[6]
PORTB_PCR22 = PORT_PCR_MUX(5);          // fb_ad[29] used as d[5]
PORTB_PCR23 = PORT_PCR_MUX(5);          // fb_ad[28] used as d[4]
PORTC_PCR12 = PORT_PCR_MUX(5);          // fb_ad[27] used as d[3]
PORTC_PCR13 = PORT_PCR_MUX(5);          // fb_ad[26] used as d[2]
PORTC_PCR14 = PORT_PCR_MUX(5);          // fb_ad[25] used as d[1]
PORTC_PCR15 = PORT_PCR_MUX(5);          // fb_ad[24] used as d[0]

PORTB_PCR19 = PORT_PCR_MUX(5);          // fb_oe_b
PORTC_PCR11 = PORT_PCR_MUX(5);          // fb_rw_b
PORTD_PCR1 = PORT_PCR_MUX(5);           // fb_cs0_b
PORTD_PCR0 = PORT_PCR_MUX(5);           // fb_ale
```

- 基址在FlexBus片选地址寄存器FB_CSAR基址字段的对应位中设置。本例中，MRAM连接片选零，基址为0x60000000。该地址为64字节对齐，且处于可用内存空间中。

```
#define MRAM_START_ADDRESS(*(volatile unsigned char*)(0x60000000))
FB_CSAR0 = (unsigned int)&MRAM_START_ADDRESS; //Set Base address
```

- 下一步是配置FB_CSCRn。该寄存器控制自动应答、地址设置和保持时间、端口大小、突发能力以及等待状态数，以支持特定内存的时序。本例中用来配置FB_CSCRn寄存器的代码是：

```
FB_CSCR0  =   FB_CSCR_PS(1)       // 8-bit port
          | FB_CSCR_AA_MASK     // auto-acknowledge
              | FB_CSCR_WS(0x2)     // 2 wait states
          ;
```

在上述代码中， FlexBus 针对 8 位传输和自动应答而配置。假设内核以最大速度运行，那么 FlexBus 将以 50 MHz 运行，周期为 20 ns。这在步骤 A 中已经配置。如前所述，片选信号在第二个时钟沿变为有效，且外部器件必须在第三个边沿以前 13.5 ns（输入建立时间）提供数据。这意味着 MRAM 仅为 6.5 ns，但 MRAM 访问时间为 35 ns。

公式 1

```
20 ns（时钟周期） – 13.5 ns（输入建立时间） = 6.5 ns < 35 ns
```

片选信号变为有效后，MRAM 不提供速度比 6.5 ns 更快的数据传输，因此必须添加等待状态周期，延长时间。增加一个等待状态不足以让 MRAM 提供数据。因此，加入两个等待状态。

公式 2

```
20 ns（时钟周期）+ 40 ns（2 个等待状态）- 13.5 ns（输入建立时间）= 46.5 ns >
35 ns
```

- 下一步，写入 CSMR 寄存器，以指定片选信号的对应地址屏蔽和支持的访问类型。

```
FB_CSMR0 = FB_CSMR_BAM(0x7)     //Set base address mask for 512K address space
         | FB_CSMR_V_MASK       //Enable cs valid signal
         ;
```

将 CSMR[BAM] 设为 0x0007 可让基址屏蔽等于 0x0007FFFF。若基址为 0x60000000 且基址屏蔽为 0x0007FFFF，则片选在 0x60000000 – 0x6007FFFF 的地址范围内激活。

如果是二进制，则：

Base Address	Hex	6				0				0				0				0				0				0			
	Binary	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Base Address Mask	Hex	0				0				0				7				F				F				F			
	Binary	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Resulting Chip Select Decode	Binary	0	1	1	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

图 6. 片选基址和屏蔽

0 或 1 = 位用来确定是否处于片选地址范围内。 x = 位是 “无关”的。

选择下列基址屏蔽之一，以保持连续存储器空间，而不使片选地址范围产生间隙。

表 1. 建议的片选基址屏蔽值

内存大小	CSMR (BAM)
1 MB	0x000F
512 MB	0x0007
256 MB	0x0003
128 MB	0x0001
64 KB	0x0000

3.1.3 传输图

选定以上配置后，读周期和写周期便如下所示。

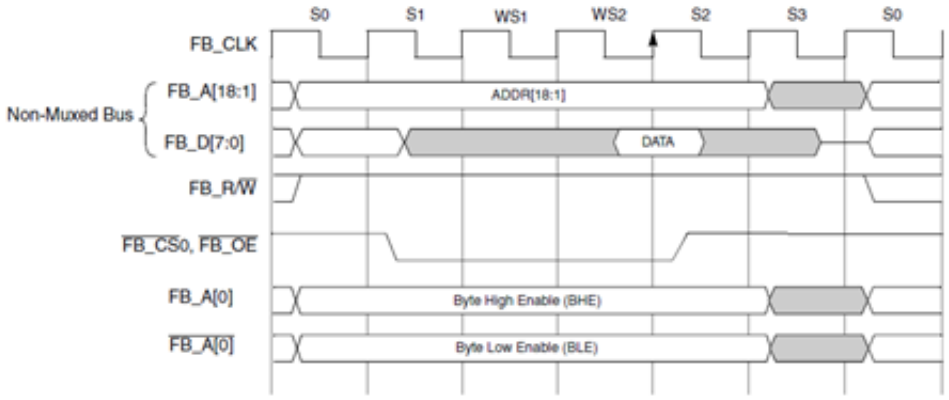


图 7. 读 （非多路复用模式 8 位， 1 个等待状态位， A0 和反相 A0 用作字节使能）

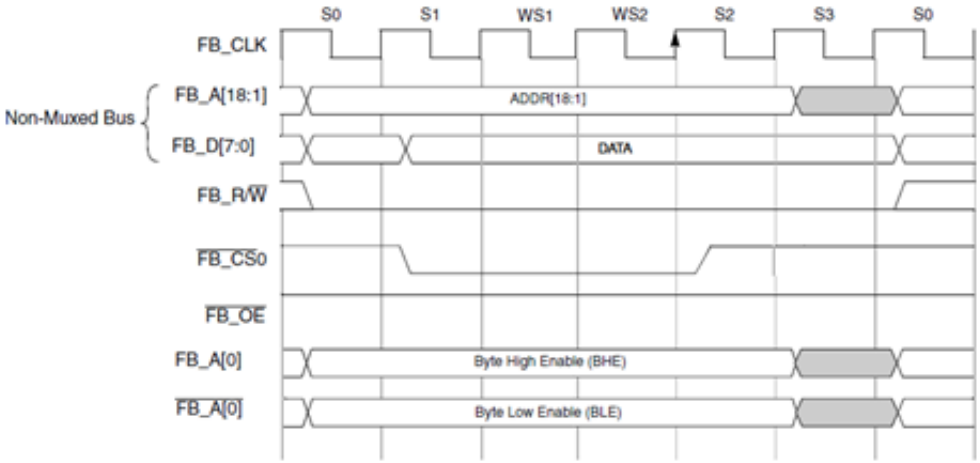


图 8. 写 （非多路复用模式 8 位， 1 个等待状态位， A0 和反相 A0 用作字节使能）

3.1.4 性能计算

本例中，最大吞吐速率的理论值为：

如果让总线工作在 50 Mhz，此时传输周期需 4 个周期加 2 个等待状态周期，则 8 位非多路复用模式下可在 6 个周期内传输 8 位数据，即 120 ns 内传输 8 位。这可以转换为 66.6 Mb/s。

8 位 / 6 个周期 × 20 ns = 66.6 Mbps

3.2 MRAM 多路复用模式

本例显示如何以 8 位数据并通过多路复用模式连接 MRAM。就本例而言，使用 TWR-K40。

3.2.1 原理图

多路复用模式下， FB_A[19:0] 引脚标记为 FB_AD[19:0]，因为它们现在同时为地址和数据信号。本例中，FB_AD[7:0] 连接 MRAM 的 DATA[7:0]。FB_AD[0] 通过反相器连接 BHE 和 BLE，以便按需访问 8 位数据。

来自 FlexBus 的 8 个数据通道 FB_AD[7:0] 同时连接 16 条 MRAM 数据通道的高位字节和低位字节。

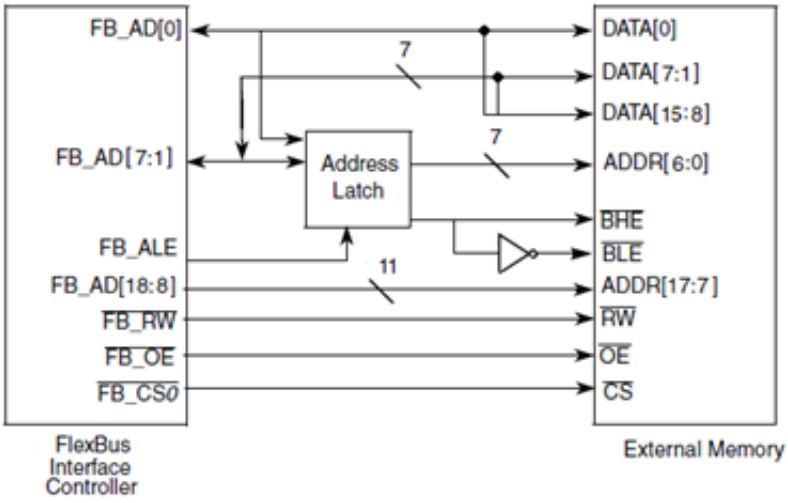


图 9. 多路复用 x8 FlexBus 至 x16 存储器

3.2.2 软件设置

多路复用模式代码与上述代码基本一致。对于多路复用模式下的 TWR-K40， FB_AD[7:0] 用作数据，因此字节通道必须右对齐。引脚数与 K60 不同。

FlexBus 配置如下所示：

```
  FB_CSCR0  =  FB_CSCR_BLS_MASK      // set byte lane shift for data
FB_AD[7:0.             right justified mode
               | FB_CSCR_PS(1)          // 8-bit port
               | FB_CSCR_AA_MASK        // auto-acknowledge
               | FB_CSCR_ASET(0x1)      // assert chip select on second clock edge after address
is asserted
               | FB_CSCR_WS(0x2) ;      // 2 wait state

  FB_CSMR0   =   FB_CSMR_BAM(0x7)       //Set base address mask for 512K address space
               | FB_CSMR_V_MASK         //Enable cs signal
               ;
```

另外，供使用的引脚为：

```
//fb clock divider 3
        SIM_CLKDIV1 |= SIM_CLKDIV1_OUTDIV3(0x3);

      /* Configure the pins needed to FlexBus Function (Alt 5) */
      /* this example uses low drive strength settings */
      //address/Data
      PORTA_PCR7=PORT_PCR_MUX(5);          //fb_ad[18]
      PORTA_PCR8=PORT_PCR_MUX(5);          //fb_ad[17]
```

```
PORTA_PCR9=PORT_PCR_MUX(5);          //fb_ad[16]
PORTA_PCR10=PORT_PCR_MUX(5);         //fb_ad[15]
PORTA_PCR24=PORT_PCR_MUX(5);         //fb_ad[14]
PORTA_PCR25=PORT_PCR_MUX(5);         //fb_ad[13]
PORTA_PCR26=PORT_PCR_MUX(5);         //fb_ad[12]
PORTA_PCR27=PORT_PCR_MUX(5);         //fb_ad[11]
PORTA_PCR28=PORT_PCR_MUX(5);         //fb_ad[10]
PORTD_PCR10=PORT_PCR_MUX(5);         //fb_ad[9]
PORTD_PCR11=PORT_PCR_MUX(5);         //fb_ad[8]
PORTD_PCR12=PORT_PCR_MUX(5);         //fb_ad[7]
PORTD_PCR13=PORT_PCR_MUX(5);         //fb_ad[6]
PORTD_PCR14=PORT_PCR_MUX(5);         //fb_ad[5]
PORTE_PCR8=PORT_PCR_MUX(5);          //fb_ad[4]
PORTE_PCR9=PORT_PCR_MUX(5);          //fb_ad[3]
PORTE_PCR10=PORT_PCR_MUX(5);         //fb_ad[2]
PORTE_PCR11=PORT_PCR_MUX(5);         //fb_ad[1]
PORTE_PCR12=PORT_PCR_MUX(5);         //fb_ad[0]
//control signals
PORTA_PCR11=PORT_PCR_MUX(5);         //fb_oe_b
PORTD_PCR15=PORT_PCR_MUX(5);         //fb_rw_b
PORTE_PCR7=PORT_PCR_MUX(5);          //fb_cs0_b
PORTE_PCR6=PORT_PCR_MUX(5);          //fb_ale
```

3.2.3 传输图

选定以上配置后，读周期和写周期便如下所示：

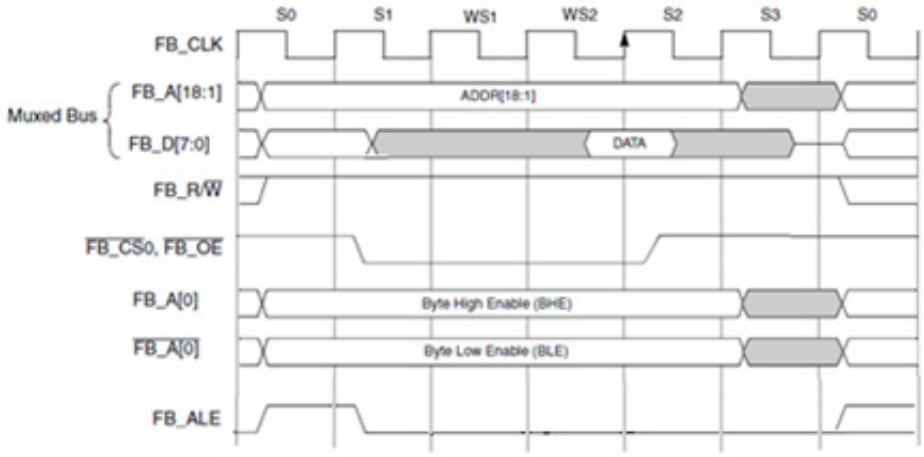


图 10. 读 （多路复用模式 8 位， 2 个等待状态位， A0 和反相 A0 用作字节使能）

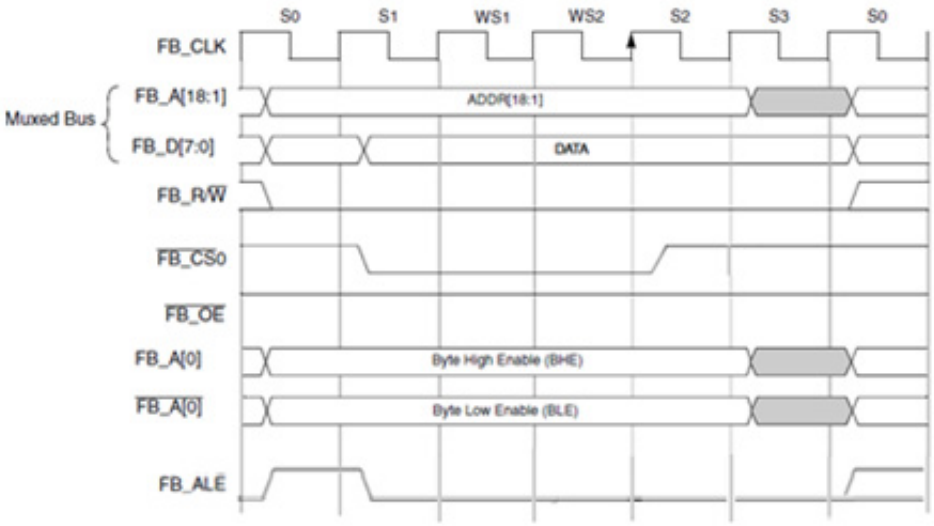


图 11. 写 （多路复用模式 8 位， 2 个等待状态位， A0 和反相 A0 用作字节使能）

3.2.4 性能计算

本例中，最大吞吐速率的理论值如下所示。

如果总线运行在 50 Mhz，且传输周期为 4 个周期加 2 个等待状态周期，则 8 位非多路复用模式下，可在 6 个周期内传输 8 位，即 120 ns 内 8 位。这可以转换为 66.6 Mb/s

3.3 读取和写入 MRAM

一旦 MRAM 正确连接且 FlexBus 完成初始化，则应用便可读取和写入外部 MRAM。这些操作很简单，可读取任意存储器位置，并直接写入存储器地址。

配置 MRAM，使可用地址范围为 0x60000000 – 0x6007FFFF。 MRAM_START_ADDRESS 指向第一个 MRAM 地址。下一个代码向外部存储器写入一个字节 (0x5A)，然后读取该字节。该操作在前 16 个地址中完成。

```
  uint8 wdata8 = 0x5A; //data to write to mram
uint8 rdata8;//variable to read mram

for(n=0x00000;n<0x000F;n++) //address offset
{
   *(vuint8*)(&MRAM_START_ADDRESS + n) = wdata8;
   rdata8=0x00; //clear data variable;
   rdata8=(*(vuint8*)(&MRAM_START_ADDRESS + n));
}
```

如此，便有了 16 位数据。读 / 写 16 位数据的唯一限制是访问偶地址。下一代码向 MRAM 写入 16 个字。每一个字在写入之前先行读取。写入 8 位数据后，第一个字写入第一个可用地址。

```
uint16 wdata16 = 0x1234;//data to write to mram
uint16 wdata16;   // variable to read mram
```

```
for(n=0x00010;n<0x001F;n+=2) //address offset
{
   *(vuint16*)(&MRAM_START_ADDRESS + n) = wdata16;
   rdata16=0x00; //clear data variable;
   rdata16=(*(vuint16*)(&MRAM_START_ADDRESS + n));
}
```

最后是相同的 32 位数据操作。若要读写双字，存储器地址必须为 4 的倍数。

```
uint32 wdata32 = 0x87654321;//data to write to mram
uint32 wdata32;   // variable to read mram

for(n=0x00020;n<0x002F;n+=4) //address offset
{
*(vuint32*)(&MRAM_START_ADDRESS + n) = wdata32;
rdata32=0x00; //clear data variable;
rdata32=(*(vuint32*)(&MRAM_START_ADDRESS + n));
}
```

3.3.1 从外部 MRAM 执行

还可以从外部 MRAM 执行代码。如需使外部 MRAM 具有可执行代码，必须执行三个主要步骤：

1. 初始化 FlexBus，以便与外部 MRAM 通信。
2. 通知 MCU 必须在外部存储器中寻找代码。这可通过编译器指令，并编辑链接器命令文件 (LCF) 实现。
3. 将代码从 Flash 复制到外部存储器。

本例显示如何重分配某个函数；该函数可让 K60 上的 LED 闪烁，并从外部 MRAM 执行。

- 使用 #pragma define_section 创建一个段，并且在 MRAM 中执行的函数要分配到这个段中。

```
#pragma define_section ExtMRAM ".myCodeInExtMRAM" abs32 RWX

__declspec (section "ExtMRAM") void toggle_LEDs(void){

int x = 0, i, n;
while(x<10){
GPIOA_PTOR=0x30000C00;
      for(i=0;i<500;i++){
      for(n=0;n<100;n++){
      asm("nop");
      }
      }
      x++;
    }
}
```

- 外部 MRAM 地址作为存储器段添加。这样做的原因，是为了让 CPU 知道去哪里寻找 toggle_LEDs(void) 函数。

```
MEMORY {
m_interrupts      (RX) : ORIGIN = 0x00000000, LENGTH = 0x000001E0
m_text            (RX) : ORIGIN = 0x00000800, LENGTH = 0x00080000-0x00000800
m_data            (RW) : ORIGIN = 0x1FFF0000, LENGTH = 0x00020000
m_cfmprotrom      (RX) : ORIGIN = 0x00000400, LENGTH = 0x00000010
extmram           (RX) : ORIGIN = 0x60000000, LENGTH = 0x00002000
}
```

这样便创建了链接器代码段，可将代码放置在所创建的存储器段内。另外还添加了某些标记，完成 Flash 到外部 MRAM 的复制。

```
___MRAMCodeStart = ___ROM_AT + SIZEOF(.app_data) + SIZEOF(.romp) ;

.my_mram : AT(___MRAMCodeStart)
{
. = ALIGN (0x4);
*(.myCodeInExtMRAM)
. = ALIGN (0x4);
} > extmram

___MRAMCodeSize = SIZEOF(.myCodeInExtMRAM);
```

最后，使用下列代码可将代码从 Flash 复制到外部 MRAM 中。

```
extern unsigned long ___MRAMCodeStart[];
#define MRAMCodeStart    (unsigned long)___MRAMCodeStart

extern unsigned long ___MRAMCodeSize[];
#define MRAMCodeSize     (unsigned long)___MRAMCodeSize

extern unsigned long ___MRAMStart[];
#define MRAMStartAddr    (unsigned long)___MRAMStart

void copyToExtMRAM(){

        unsigned char *MRAMSource;
      unsigned char *MRAMDestiny;
      unsigned int MRAMSize;

        // Initialize the pointers to start the copy from Flash to RAM
            MRAMSource = (unsigned char *)(MRAMCodeStart);
            MRAMDestiny = (unsigned char *)(MRAMStartAddr);
            MRAMSize = (unsigned long)(MRAMCodeSize);

            // Copying the code from Flash to External RAM
            while(MRAMSize--)
            {
               *MRAMDestiny = *MRAMSource;
               MRAMDestiny++;
               MRAMSource++;
            }
      }
```

现在，函数已完成外部 MRAM 的重分配，可从此处开始执行。

附注

请注意，从 MRAM 执行时速度极慢，因此应用开发中必须考虑到这些时序规格。

3.4 LCD

本例说明如何通过 FlexBus 将 TFT-LCD 连接至 TWR-K60。本例使用了集成 Solomon Systech TFT-LCD 控制器 SSD1289 的 TWR-LCD 板。该控制器驱动器集成 RAM、电源电路等。它能用作 8/16/18 位 6800/8080 系列兼容型并行或串行外设接口，与通用 MCU/MPU 实现接口。更多信息，请参考应用笔记，标题为*在 MCF51MM 系列器件上使用带 TWR-LCD 的 Freescale eGUI*（文档编号：AN4153）

本例使用 16 位 6800 系列模式。并行接口由 16 个双向数据引脚 (D[17:0])、R /W、D/C、E 和 CS 组成。R/W 输入高电平表示针对图形显示数据 RAM (GDDRAM) 或状态寄存器的读操作。R/W 输入低电平表示针对显示数据 RAM 或内部命令寄存器的写操作，具体取决于 D/C 输入状态。

为了正确选择接口，请在 LCD 中使用下列配置。

- PS3 =1， PS2 =0， PS1 =1， PS0 =1。该配置必须通过外部设置。
- D1～D8 和 D10～D17，R/W(/WR)，以及 D/C 用作 I/O 控制。这些线路与 FlexBus 线路相连。

3.4.1 原理图

为了实现与 LCD 的接口，FlexBus 应当采用 16 位模式和多路复用模式。无需使用 FB_ALE，因此忽略地址建立期间的第一个 FlexBus 周期。另外，字节通道移位为右对齐，从而数据在 16 个最低有效位中建立。换言之，FB_AD[0:15] 线路供数据使用。FB_AD[16] 用作 SSD1289 的 D/C 或 DS 输入。 WR 和 CS 信号连接至 K60 的 FB_RW 和 FB_CS0：

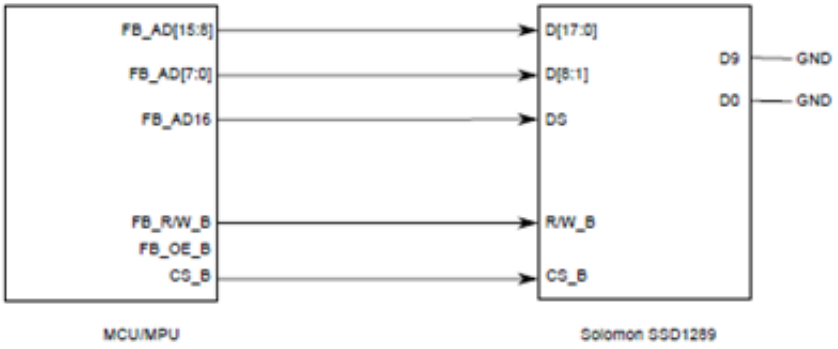


图 12. 多路复用 16 位 FlexBus 到 TFT-LCD

如果不希望读取像素数据，可直接拉高 LCD 控制器的 E(RD) 引脚。此外，注意 SSD1289 的 D0 和 D9 未连接。这些引脚在 16 位模式下不需要。更多信息，请参考 TWR-LCD 与 TWR-K60 的原理图。

3.4.2 软件设置

为确定 FlexBus 时序要求的调节，可参考参考手册中的 FlexBus 章节，标题为 *K60 子系列参考手册：采用 144 引脚封装的 100MHz 器件*（文档编号：K60P144M100SF2RM）以及数据手册中的“时序规范”章节内容，标题为 *K60 子系列数据手册*（文档编号：K60P144M100SF）。

- FlexBus所需的IO引脚必须配置为FlexBus功能。由于16位多路复用模式下只需将16条线路用于数据，1 条线路用于 D/C 信号、RW 和 CS 信号，因此，K60 的引脚必须配置成 ALT5。LCD 和 FlexBus 的初始化过程与 D4D 相同，更多信息请参考参考手册，标题为 *Freescale 嵌入式 GUI (D4D)*（文档编号：DRM116）。

```
#define ALT5 (PORT_PCR_MUX(5)|PORT_PCR_DSE_MASK)        // Alternative function 5 =
FB enable
#define FLEX_CLK_INIT (SIM_CLKDIV1 |= SIM_CLKDIV1_OUTDIV3(1))// FlexBus = Sysclk/2

SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK | SIM_SCGC5_PORTB_MASK | SIM_SCGC5_PORTC_MASK |
SIM_SCGC5_PORTD_MASK | SIM_SCGC5_PORTE_MASK;
```

```
PORTC_PCR0=ALT5;
PORTC_PCR1=ALT5;
PORTC_PCR2=ALT5;
PORTC_PCR3=ALT5;
PORTC_PCR4=ALT5;
PORTC_PCR5=ALT5;
PORTC_PCR6=ALT5;
PORTC_PCR7=ALT5;
PORTC_PCR8=ALT5;
PORTC_PCR9=ALT5;
PORTC_PCR10=ALT5;
PORTC_PCR11=ALT5;
PORTD_PCR1=ALT5;
PORTD_PCR2=ALT5;
PORTD_PCR3=ALT5;
PORTD_PCR4=ALT5;
PORTD_PCR5=ALT5;
PORTD_PCR6=ALT5;
PORTB_PCR17=ALT5;
PORTB_PCR18=ALT5;

SIM_SOPT2 |= SIM_SOPT2_FBSL(3);
SIM_SCGC7 |= SIM_SCGC7_FLEXBUS_MASK;

/*PTC0 PTC1 PTC2 PTC3 PTC4 PTC5 PTC6 PTC7 PTC8 PTC9 PTC10 PTC11
  14    13   12  CLK  11   10   9    8     7    6    5     RW
PTD1 PTD2 PTD3 PTD4 PTD5 PTD6 PTB17 PTB18
CS   4     3    2    1    0    16D/C    15
```

- 基址在FlexBus片选地址寄存器FB_CSAR基址字段的对应位中设置。本例中，TWR-LCD连接片选零，基址为 0x60000000。该地址为 64 位对齐，且处于可用存储器空间中。

```
#define FLEX_DC_ADDRESS    0x60000000
FLEX_CSAR = FLEX_DC_ADDRESS; // CS0 base address
```

- 片选寄存器 CSMR 寄存器指定相应片选信号的地址屏蔽和允许的访问类型。将基址屏蔽设为 1 (CSMR0[BAM]) = 1) 表示 CS 块大小为 2^n（n = CSMR[BAM] 中的位数 + 16），即 128 KB。这意味着地址 0x60000000 到 0x6001FFFF 可通过 CS 访问。因此，地址 0x6000000（FB_AD16 为低电平）可用来访问 SSD1289 的索引寄存器，而地址 0x60010000（FB_AD16 为高电平）可用来访问 SSD1289 的数据缓冲器。

```
#define FLEX_ADRESS_MASK 0x00010000

FLEX_CSMR = FLEX_ADRESS_MASK | FLEX_CSMR_V_MASK; // The address range is set to 128K
because the DC signal is connected on address wire
```

- 当CS0与LCD控制器的CS_B连接时，CSMR0[V]必须置位为使能CSn。复位时，除FB_CS0之外，没有可用的片选信号，直到置位 CSMR0[V]。根据 SSD1289m 数据手册，最小写周期为 100 ns。读周期和写周期之间至少有 4 个 FB 周期。FB_CLK 不可超过 40 MHz，否则必须加入等待状态。

表 2. SSD1289 规格

符号	参数	最小值	类型	最大值	单位
t_{cycle}	时钟周期时间 （写周期）	100	–	–	ns

```
// MUX mode + Wait States
#define FLEX_CSCR_MUX_MASK (FB_CSCR_BLS_MASK)
#define FLEX_CSMR_V_MASK FB_CSMR_V_MASK
#define FLEX_CSCR_AA_MASK FB_CSCR_AA_MASK
#define FLEX_CSCR_PS1_MASK (FB_CSCR_PS(2))

FLEX_CSCR = FLEX_CSCR_MUX_MASK | FLEX_CSCR_AA_MASK | FLEX_CSCR_PS1_MASK; // FlexBus
setup as fast as possible in multiplexed mode
```

- 若要发送命令和数据，应使用下列函数。注意，唯一不同之处是发送时的数据地址。通过 0x6000000 （FB_AD16 为低电平）访问索引寄存器，通过 0x60010000 （FB_AD16 为高电平）访问数据缓冲器。

```
void vfnSendDataWord(unsigned short value)
   {
     *((unsigned short*)FLEX_BASE_ADDRESS) = value;
   }

void vfnSendCmdWord(unsigned short cmd)
{
*((unsigned short*)FLEX_DC_ADDRESS) = cmd;
```

3.4.3 传输图

选定以上配置后，读周期和写周期便如下所示：

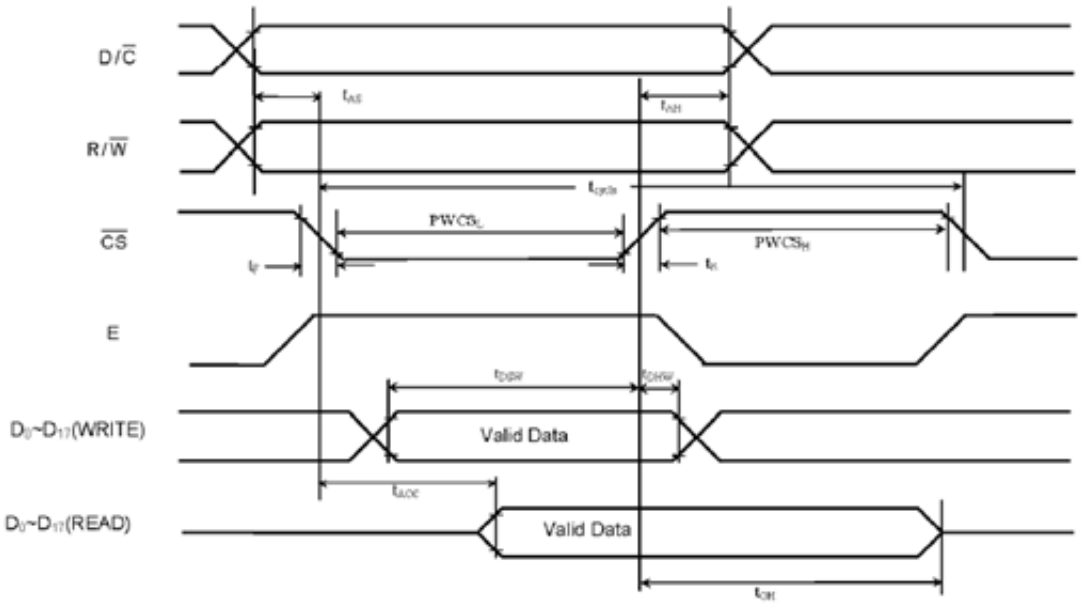


图 13. SSD1289 传输图

图 10 中的规格可在 SSD1289 数据手册中找到。这些规格可用来正确配置 Flexbus 模块。

3.5 在 MQX 应用中使用 FlexBus

在 MQX 应用中使用 FlexBus 的情况与裸板应用很相似。可以使用相同的寄存器定义来配置 FlexBus。使用 TWR-K40 与 TWR-K60 与外部 MRAM 通信时， MQX 配置 FlexBus 模块。可对该模块重新配置，以满足应用要求。

3.5.1 应用说明

当应用耗尽内部存储器时，就不能再创建新的任务。本例显示如何配置 FlexBus，使其与外部 MRAM 通信，扩展 MQX 存储器池。

3.5.2 软件设置

如下列代码所示，所有任务所需的存储器总空间超出了 K60 的内部 RAM 尺寸。

```
TASK_TEMPLATE_STRUCT MQX_template_list[] =
{
/*  Task number, Entry point, Stack, Pri, String, Auto? */
   {MAIN_TASK, Main_task, 40000, 9, "main", MQX_AUTO_START_TASK},
   {HELLO_TASK, Hello_task, 40000, 10, "hello", 0},
   {TOGGLE_TASK, ToggleLEDs_task, 40000, 10, "toggle", 0},
   {COUNTER_TASK, Counter_task, 6000, 10, "counter", 0},
   {0,          0,          0,    0,  0,    0,                 }
};
```

Main_task 初始化 FlexBus。该初始化过程与 3.4.2 节 “软件设置”相同。在此之后，必须让 MQX 内核知道，有可用的外部存储器。使用 _mem_extend() 函数便可实现。该函数收到的参数为指向开始地址的指针，以及外部存储器模块的大小。

```
/*TASK*-----------------------------------------------------
*
* Task Name : Main_task
* Comments :
*   This task creates an instance of Hello_task and ToggleLEDs_task
*
*END*-----------------------------------------------------*/
void Main_task(uint_32 initial_data)
{
      printf("Mem Extend Test\n");
      TWRK60_flexbus_init();
      result = _mem_extend((pointer)0x70000000,0x80000);
      while (1){
        t1 = _task_create(0, HELLO_TASK, 0);
        t2 = _task_create(0, COUNTER_TASK, 0);
        t3 = _task_create(0, TOGGLE_TASK, 0);

        _task_block();
        _task_destroy(t3);
      }
}
```

为了使能 _mem_extend() 函数，有必要对 user_config.h 文件进行适当修改，并重建 BSP 和 PSP 库。所需的修改如下所示：

```
#define MQX_USE_MEM                1 //Enable to use _mem_extend
#define MQX_USE_LWMEM              0
#define MQX_USE_LWMEM_ALLOCATOR    0
```

如需重建 BSP 和 PSP 库，请参考 MQX 文档目录下的 FSL_MQX_getting_started.pdf。

4 PCB 设计建议

由于驱动外部存储器时对时序要求极高，在 PCB 布局布线时需考虑到很多因素。

每组信号走线都必须具有相等的负载和类似的分布，以保持时序和信号完整性。

控制和时钟信号以点对点方式分布。元件可以，并且应当尽可能靠近 MCU 放置。为了避免产生串扰，应将地址和命令信号 （即不同的路由层）与数据和数据选通信号隔离。

5 结语

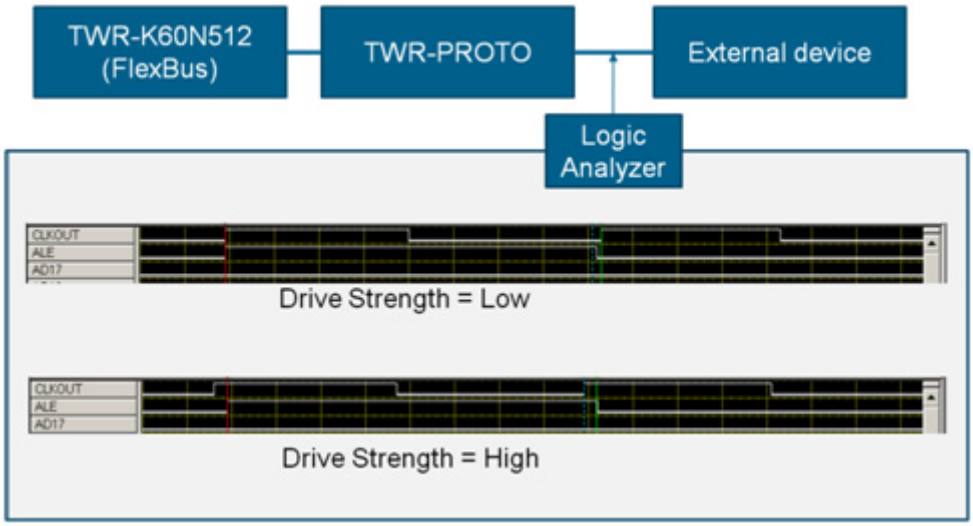
FlexBus 是一种接口，有助于以简单的方式实现外部器件的通信。完成 Flexbus 的配置后，便可轻松使用，因为其工作对于用户而言是透明的。例如，访问外部存储器的过程与访问内部存储器相同，因为可直接写入存储器地址。

由于 FlexBus 采用多连接配置，它能提供灵活性，实现速度和大小的最优平衡。

FlexBus 的这些特性使其成为与几乎所有并行通信器件实现通信的最佳选择。

附录 A　　使用 TWR-PROTO 获得正确的 FlexBus 信号

当用户逻辑连接至 Kinetis 塔式系统板时， TWR-PROTO 可在多种情况下获取 FlexBus 信号。然而，这些信号输出时序取决于负载电容、电路板布局、室温等。有时无法从 TWR-PROTO 获取正确的 FlexBus 波形。为了帮助用户解决这个问题，本文描述如何通过 TWR-PROTO 和 TWR-K60N512 获取正确的 FlexBus 信号。在驱动强度设置为较高数值时，可获得 TWR-PROTO 的正确波形。本文中用来设置高驱动强度与使用 TWR-PROTO 监控波形的代码相同。



A.1　　指出问题

使用 FlexBus 并以 ASIC 连接 TWR-K60 时，通常使用 TWRPROTO 创建原始电路。图 A-1 显示使用逻辑分析仪后， TWR-PROTO 上的 FlexBus 信号波形。

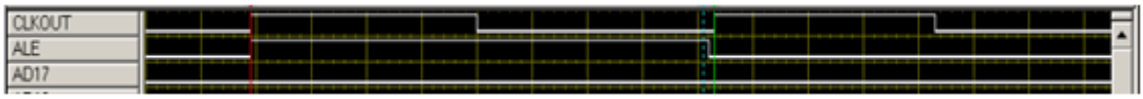


图 A-1. FlexBus 信号波形

图 A-2 是执行写操作事务时，参考手册中显示的波形。

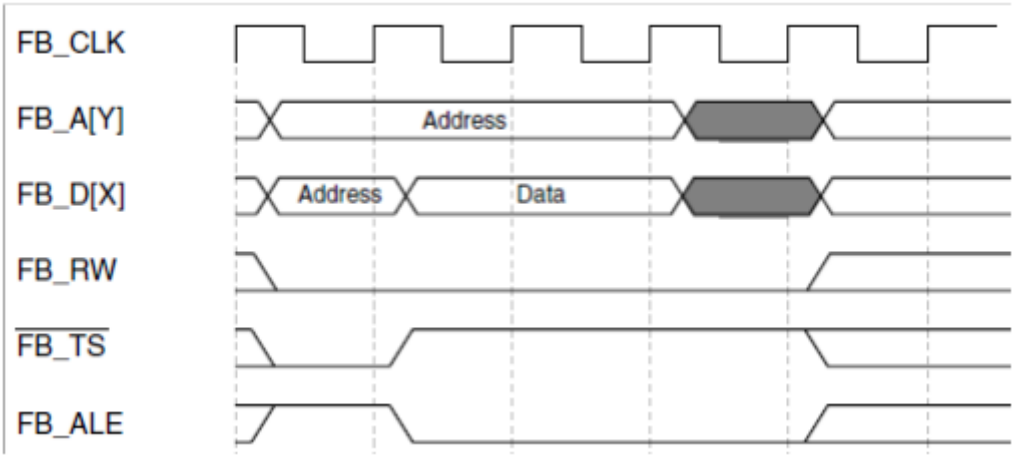


图 A-2. 基本写字节周期

图 A-1 中，观察到的 FB_CLK 和 FB_ALE 几乎同时上升，且 FB_ALE 在下一个 FB_CLK 上升沿之前下降。但是，图 A-2 中的 FB_ALE 在第二个 FB_CLK 上升沿之后下降。因此，同步器件不可能正确采样 FlexBus 信号。

其原因就在于， FB_CLK (PTC3) 分叉使用 PWM3，见图 A-3。因此，与其它 FlexBus 信号相比，该引脚发生过载和延迟。

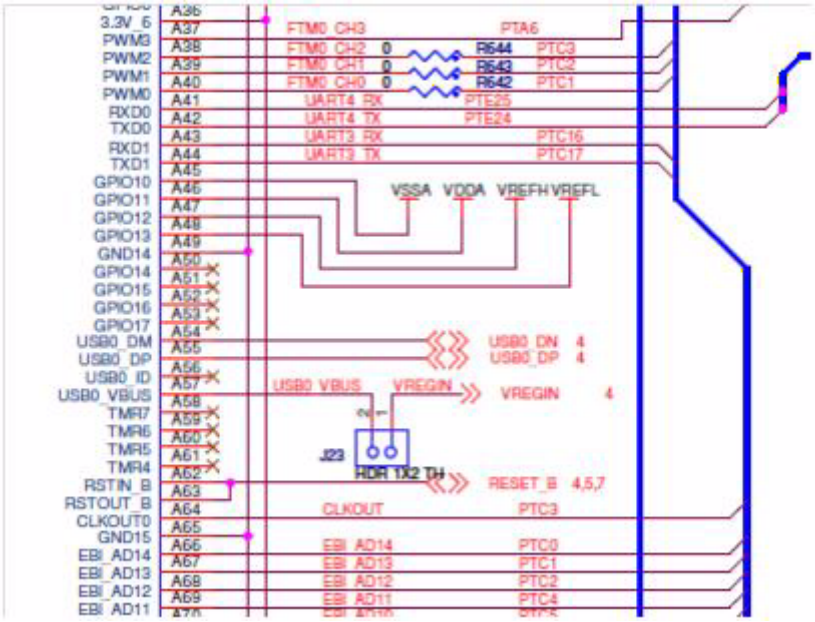


图 A-3. TWR-K60 部分电路

附录 B　　预防措施

B.1　　预防措施的配置

与 TWR-PROTO 上观察到的其它 FlexBus 信号相比，FB_CLK 有一定延迟，原因如前文所述。应对措施为：

在输出引脚上，设置高驱动强度

由于将驱动强度设得较高，Kinetis 输出电流上升，Kinetis 可以在塔式系统板上负载电容没有失去的地方输出信号。为了增加驱动强度，可将 PORTx_PCRn[DSE] 设为 1。 K60 上的 FB_CLK 和 PTC3 经多路复用处理。因此，需将 PORTC_PCR3[DSE] 设为 1，增加 FB_CLK 的驱动强度。

B.2　　示例代码

头文件代码：

```
#define PORT_PCR_DSE(x) (((uint32_t)((uint32_t)(x))<<PORT_PCR_DSE_SHIFT))&PORT_PCR_DSE_MASK)
```

程序代码：

```
PORTC_PCR3 = (PORTC_PCR3 & ~PORT_PCR_DSE_MASK) | PORT_PCR_DSE(1) ;
```

在您的代码中插入上述代码。

B.3　　结果

图 B-1 显示驱动强度设置为较高时的波形。

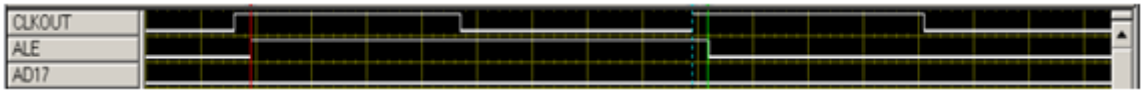


图 B-1. 采取应对措施后的 FlexBus 结果波形

图 A-2 中， FB_ALE （图 B-1 中为 ALE）在第一个 FB_CLK （图 B-1 中为 CLKOUT）上升沿之后变为有效，而 FB_ALE 在第二个 FB_CLK 上升沿之后忽略。然后，通过 FlexBus 连接 Kinetis 的同步器件便可正确对信号进行采样。

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

本文档中的信息仅供系统和软件实施方使用 Freescale 产品。本文并未明示或者暗示授予利用本文档信息进行设计或者加工集成电路的版权许可。Freescale 保留对此处任何产品进行更改的权利，恕不另行通知。

Freescale 对其产品在任何特定用途方面的适用性不做任何担保、表示或保证，也不承担因为应用程序或者使用产品或电路所产生的任何责任，明确拒绝承担包括但不局限于后果性的或附带性的损害在内的所有责任。Freescale 的数据表和 / 或规格中所提供的 “典型”参数在不同应用中可能并且确实不同，实际性能会随时间而有所变化。所有运行参数，包括“经典值”在内，必须经由客户的技术专家对每个客户的应用程序进行验证。Freescale 未转让与其专利权及其他权利相关的许可。Freescale 销售产品时遵循以下网址中包含的标准销售条款和条件：
freescale.com/SalesTermsandConditions.

Document Number: AN4393
Rev. 0, 05/2012