Freescale Semiconductor
Application Note

Document Number: AN4393
Rev. 0, 05/2012

Using FlexBus Interface for Kinetis Microcontrollers

by: Carlos Musich and Alejandro Lozano
Technical Information Center

Contents

1 Introduction

In many cases it is necessary to connect external devices to a microcontroller. For example, when an application demands a larger amount of memory than the memory available, it is necessary to use external memories. Another example is when a system requires additional interfaces that are not available on the microcontroller, external peripherals are then needed. Thinking of these needs a multi-function external bus interface called the FlexBus interface controller is provided in Kinetis devices with basic a function of interfacing to slave-only devices. It can be directly connected to external ROMs, flash memories, programmable logic devices, or other simple target (slave) devices with little or no additional circuitry.

This application note describes how to set up and use the FlexBus interface on the K60 and shows some implementation examples. In these examples the TWR-K60 is connected to TWR-MEM and TWR-LCD via FlexBus. The TWR_MEM is used to demonstrate how to read, write, and execute from an external memory. The TWR-LCD is used to show connectivity with an LCD displaying an image. Although the focus of the application note is the K60, the information contained in this document will also apply to Kinetis devices with the FlexBus interface.

2 Functional description

The external interface is a configurable multiplexed bus, set to one of the following modes:

- Multiplexed 32-bit address and 32-bit data
- Multiplexed 32-bit address and 16-bit data (non-multiplexed 16-bit address and 16-bit data)
- Multiplexed 32-bit address and 8-bit data (non-multiplexed 24-bit address and 8-bit data)
- Non-multiplexed 32-bit address and 32-bit data busses

The fastest FlexBus transfers take four cycles of the FlexBus clock. This section describes the signals used by FlexBus in read and write cycles.

NOTE

Kinetis FlexBus supports burst. If you are connected to any device that supports burst (for example, FPGA), it can transfer data faster than 4 FlexBus cycles because it does not require the address phase between transferring the data each time.

2.1 Read cycle

1. The address drives in the first cycle. In non-multiplexed mode, the FB_A[31:0] carries the address. In multiplexed mode, the FB_AD[31:0] bus carries the address and data. The full 32-bit address is driven on the first clock of the bus cycle (address phase). The time between the first clock edge and the moment where the address is driven is called Output Valid time (FB2).
2. In non-multiplexed mode the Transfer Start (FB_TS) signal is asserted indicating that the device has started a bus transaction. In multiplexed mode, an inverted FB_TS (FB_ALE) is available as an address latch enable, which indicates when the address is driven on the FB_AD bus. These signals are negated in the next clock edge, but can be configured to remain asserted until the first positive clock edge after FB_CS asserts.
3. In the second edge, the corresponding Chip Select (FB_CS) and Output Enable (FB_OE) signals are asserted.
4. The external device must supply the data on the bus before the start of the third cycle. This time is called Input Setup time (FB4 = 13.5 ns for K60). If the external device is not fast enough, Wait States (extra clock cycles) can be added.

   If the auto-acknowledge feature is disabled (CSCRn[AA] = 0), then FB_TA must be negated 13.5 ns (FB4) before the third cycle.

5. If no Wait States were added, the data is read in the third clock edge. In non-multiplexed mode, the FB_D[31:0] buses carry data. In Multiplexed mode, after the address is latched, the data is driven on the FB_AD[31:0] bus (data phase). During the data phase, the address continues driving on the pins not used for data. For example, in 16-bit mode the lower address continues driving on FB_AD[15:0], and in 8-bit mode the lower address continues driving on FB_AD[23:0].
6. Data must hold in the bus at least for 0.5 nS. This time is called Input Hold (FB5).
7. Finally, FB_TA indicates the external data transfer is complete. When the processor recognizes FB_TA during a read cycle, it latches the data and then terminates the bus cycle.

   If auto-acknowledge is disabled (CSCRn[AA] = 0), the external device drives FB_TA to terminate the bus transfer; if auto-acknowledge is enabled (CSCRn[AA] = 1), FB_TA is generated internally after a specified number of wait states, or the external device may assert external FB_TA before the wait-state countdown, terminating the cycle early. The device negates FB_CSn one cycle after the last FB_TA asserts. During read cycles, the peripheral must continue to drive data until FB_TA is recognized. See Figure 1.

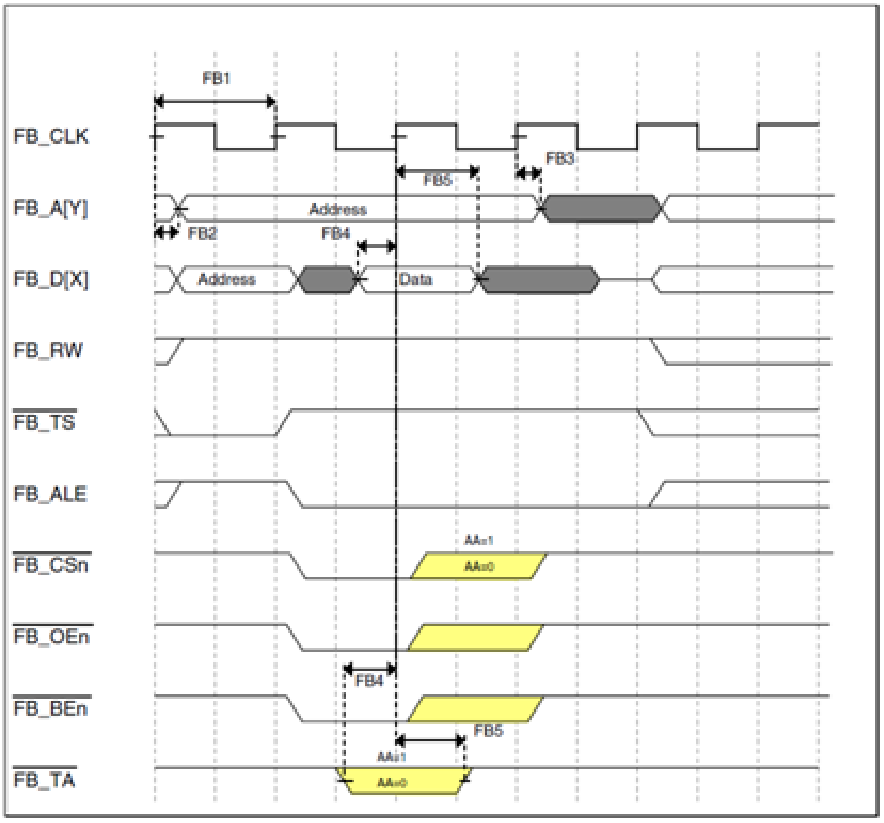


Figure 1. FlexBus read cycle

2.2 Write cycle

The FlexBus read and write cycles have almost the same timing. The main differences are:

1. FB_OE is never asserted
2. When the processor recognizes FB_TA during a write cycle, the bus cycle is terminated
3. For write cycles, the processor continues driving data one clock after FB_CS*n* is negated

See Figure 2.

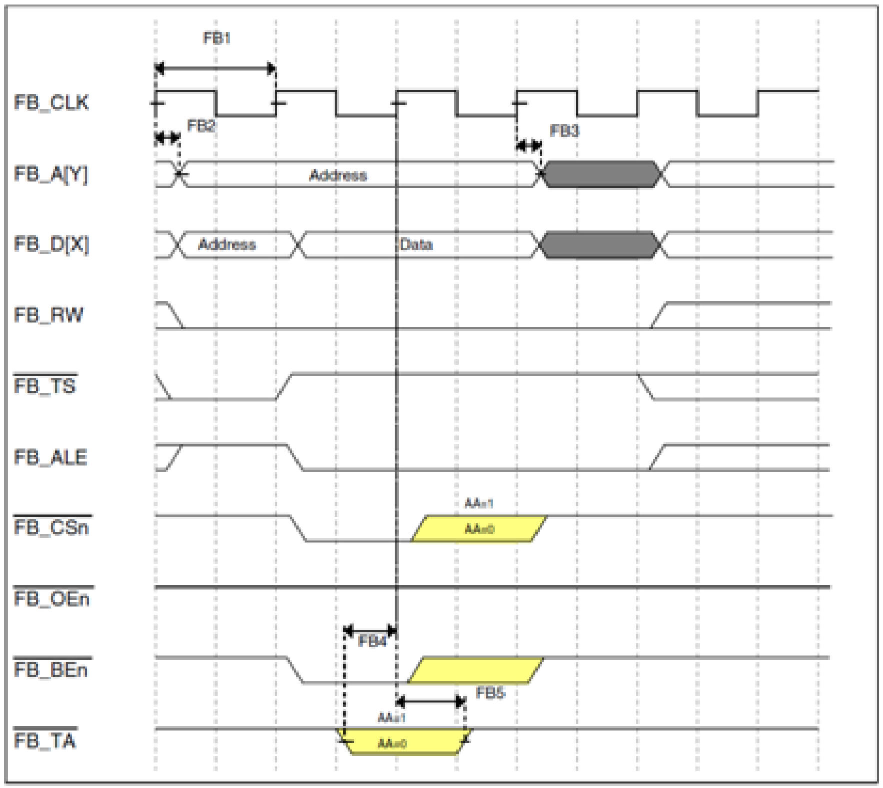


Figure 2. FlexBus write cycle

2.3 Data byte alignment and physical connections

The device aligns data transfers in FlexBus byte lanes with the number of lanes depending on the data port width.

Figure 3 shows the byte lanes the external memory connects to, and the sequential transfers of a 32-bit transfer for the supported port sizes when the byte lane shift is disabled or enabled.

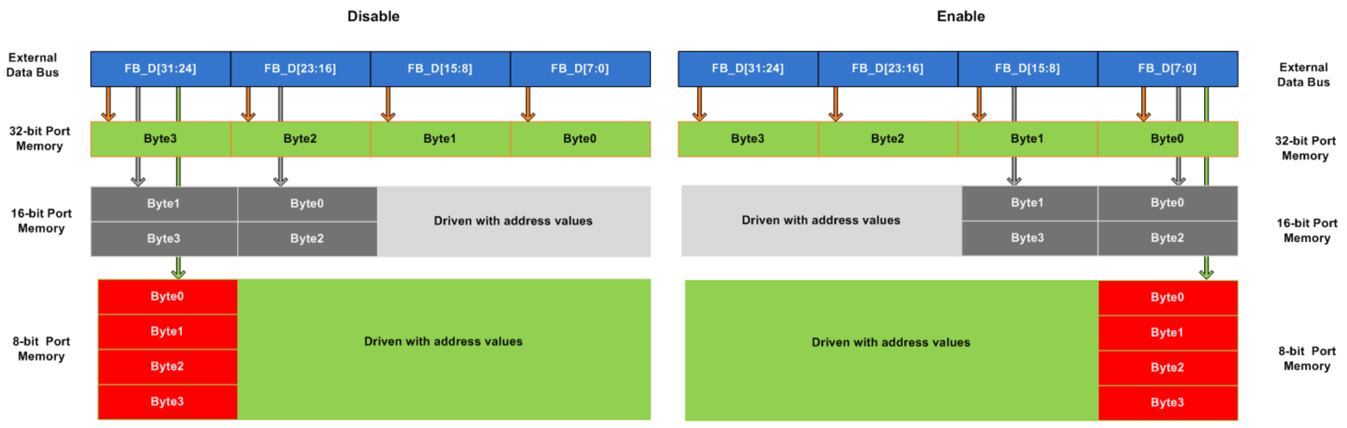


Figure 3. Sequential 32-bit transfers, byte lane shift differences

2.4 Memory map

Typical memory mapping as shown in Figure 4 0x6000_000 - 0xA000_0000 is the FlexBus space used for execution, 0xA000_0000 - 0xE000_0000 can only be used for data.

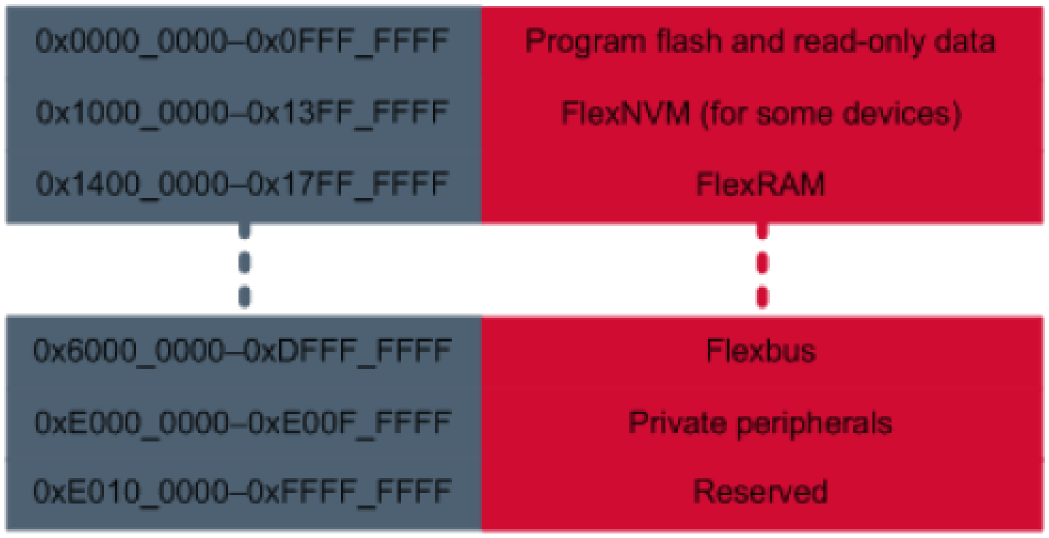


Figure 4. FlexBus Memory Map

3 Examples

The examples that follow show ways some common interfaces can be accomplished.

NOTE

Both MRAM non-multiplexed and multiplexed mode use 8-bit data because of the routing limitations of the TWR-MEM module. To get the best performance you must use a 16-bit data connection.

3.1 MRAM non-multiplexed mode

In these examples the FlexBus is connected to a Magnetoresistive Random Access Memory (MRAM) chip, which is in the same footprint as an industry standard x16 asynchronous Fast SRAM. To connect to a 16-bit memory using the absolute minimum glue logic, this examples uses the non-multiplexed mode with 8-bit data and 20-bit address lines.

This solution is demonstrated on the TWR-K60 evaluation board.

Advantages:

- Increases available system memory by adding a 512 KByte MRAM
- MRAM is a non-volatile memory that does not require special erasing and programming routines
- Only a single gate inverter is required

Disadvantages:

- MRAM has slower access time than SRAM, and therefore the memory bandwidth is lower.
- To reduce the amount of glue logic, this method treats the MRAM as an 8-bit memory, and thus provides less performance than 16-bit.

3.1.1 Schematics

Although the MRAM is 16-bit, it is byte-read/writable. It has byte high enable and byte low enable pins to select between the upper or lower byte. These signals can be constructed from the FlexBus signals using only a single gate inverter on FB_A0. FB_A0 is used as byte high enable and the output of the inverter FB_A0 is used as byte low enable. This effectively treats the memory as an 8-bit device. Performance is sacrificed using this method. However, it minimizes glue logic, which saves on component cost.

The eight data lines from the FlexBus are connected to both the upper byte and lower byte of the 16 data lines of the MRAM. The byte-enable lines that are generated by FB_A0 and FB_A0 select, which byte is driving on the data lines. When not selected, the other byte lines of the MRAM will tri-state and not interfere with the desired byte.

The memory is word-addressable (16-bit), so the address lines must be shifted such that the FlexBus word address line FB_A[1] is connected to A[0] of the memory. Thus, FB_A[18:1] are connected to A[17:0] of the memory.

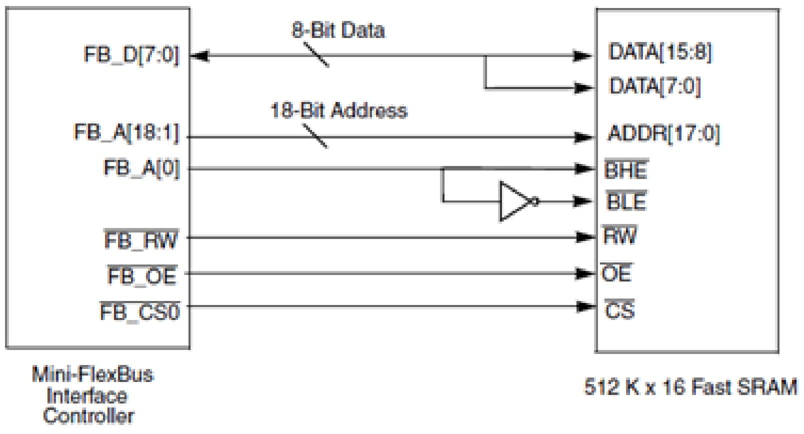


Figure 5. Non-Multiplexed x8 FlexBus to x16 memory

3.1.2 Software setup

To determine the required adjustments to the FlexBus timing, the FlexBus chapter of the reference manual titled *K60 Sub-Family Reference Manual for 100MHz devices in 144 pin packages* (document number, K60P144M100SF2RM) and the Timing Spec chapter in the datasheet titled *K60 Sub-Family Data Sheet* (document number K60P144M100SF) are used.

- FlexBus clock FB_CLK runs at the same speed as the Bus clock. The FB_CLK frequency may be the same as the internal system bus frequency or an integer divider of that frequency. Its default value is 50 MHz.

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

- The IO pins required by FlexBus must be configured for the FlexBus function. There are 18 pins for the address, 8 pins for data, and 4 control pins. Use the PORT_PCR_MUX(5) macro to set each pin for the FlexBus function. You must set the GPIOs ports clocks

```
// 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
```

- The base address is set in the base address field of the FlexBus Chip Select Address Register FB_CSAR for the corresponding chip select. In this example, the MRAM is connected to chip select zero and the base address is 0x60000000. This address is 64-byte aligned, and t is in an available memory space.

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

- The next step is to configure FB_CSCRn. This register controls the auto-acknowledge, address setup and hold times, port size, burst capability, and number of wait states to accommodate the timing of the specific memory.
The code used to configure FB_CSCRn register in this example is:

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

As in the code above, the FlexBus was configured for 8-bit transfers and auto acknowledge. Assuming at the core is running at maximum speed, the FlexBus will run at 50 MHz with 20 ns cycles. This was configured in step A. Described in the previous section, the chip select is asserted in the second clock edge and the external device must supply data 13.5 ns (Input Setup time) before the third edge. This means that MRAM has only 6.5 ns, but MRAM access time is 35 ns.

Eqn.1

```
    20 ns (clock cycle) - 13.5 ns (Input Setup time) = 6.5 ns < 35 ns
```

MRAM does not provide data faster than 6.5 ns after chip select asserts, then wait state cycles must be added to give it more time. Adding one wait state is not enough time for the MRAM to provide data. Therefore two wait states are added.

Eqn. 2

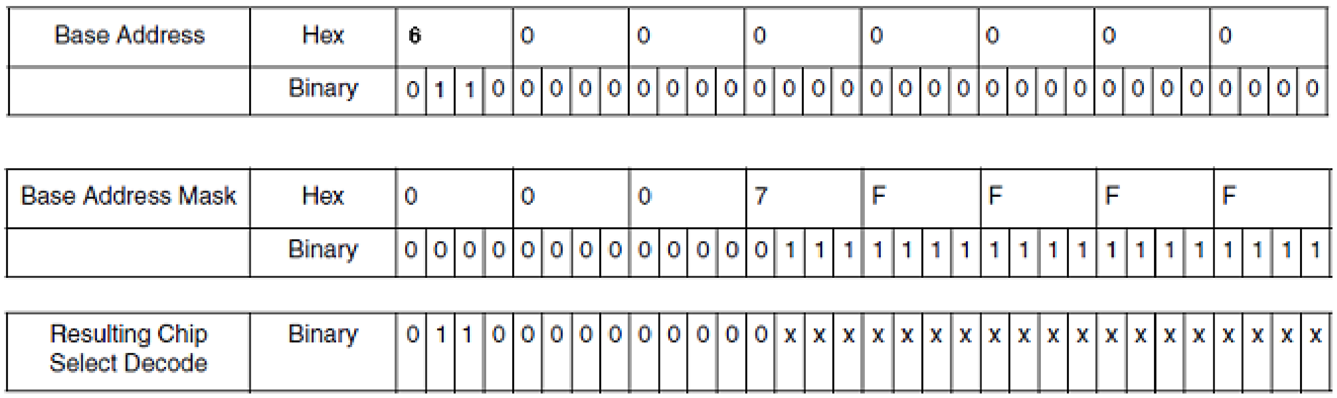
```
    20 ns (clock cycle) + 40 ns (2 wait states) - 13.5 ns (Input Setup time) = 46.5 ns >
    35 ns
```

- Next, write CSMR register to specify the address mask and allowable access types for the respective chip-selects.

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

Setting CSMR[BAM] to 0x0007 results in a base address mask of 0x0007FFFF. With base address 0x60000000 and base address mask 0x0007FFFF, the chip select is active for address range 0x60000000 – 0x6007FFFF.

In binary looks like the following:



Base Address	Hex	6				0				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	0	0	0	0

Base Address Mask	Hex	0				0				0				7				F				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	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	x	x	x	x

Figure 6. Chip select base address and mask

0 or 1 = bit is used to determine, if within chip-select address range. x = bit is a “don’t care.”

Select one of the following base address masks to maintain a contiguous memory space without gaps in the chip select address range.

Table 1. Suggested chip select base address mask values

Memory Size	CSMR (BAM)
1 Mbyte	0x000F
512 Mbyte	0x0007
256 Mbyte	0x0003
128 Mbyte	0x0001
64 Kbyte	0x0000

3.1.3 Transfer diagrams

With the above configuration chosen, the read and write cycles look as follows.

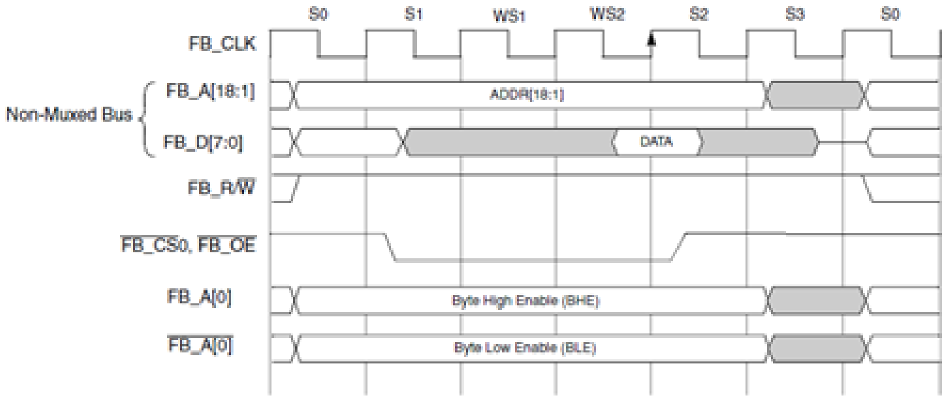


Figure 7. Read (non-muxed mode 8-bit with one wait state and A0 and inverted A0 used as byte enables)

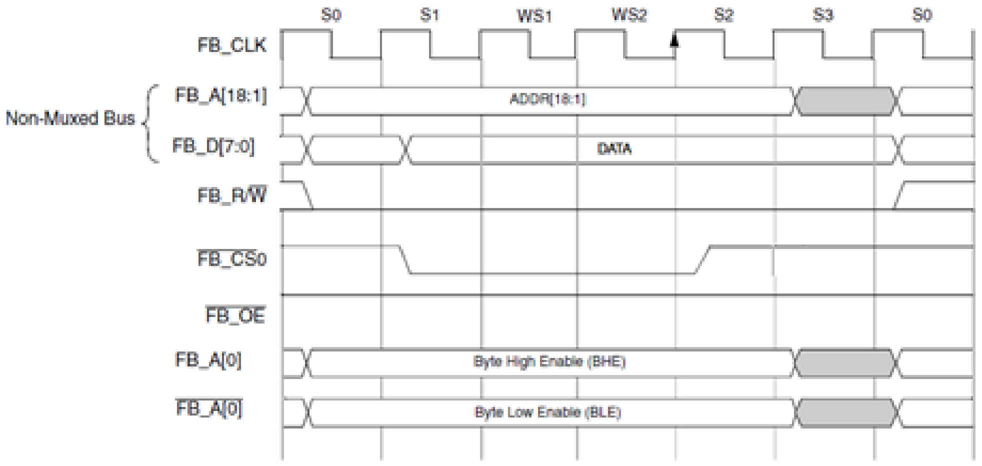


Figure 8. Write (non-muxed mode 8-bit with one wait state and A0 and inverted A0 used as byte enables)

3.1.4 Performance calculation

In this example, the theoretical maximal throughput is:

If you run the bus at 50 Mhz, with transfer cycles taking four cycles plus two wait state cycles, in 8-bit non-muxed mode you can transfer eight bits in six cycles, therefore eight bits in 120 ns. This translates to 66.6 Mbit / sec.

8-bits / 6 cycles × 20 ns = 66.6 Mbps

3.2 MRAM multiplexed mode

This example shows how to connect MRAM with 8-bit data and multiplexed mode. For this specific example the TWR-K40 is used.

3.2.1 Schematics

In multiplexed mode, the FB_A[19:0] pins are notated as FB_AD[19:0], because they now are both address and data signals. In this example, FB_AD[7:0] are connected to DATA[7:0] of the MRAM. FB_AD[0] is connected to the BHE and BLE through an inverter to access the 8-bit data needed.

The eight data lines FB_AD[7:0] from the FlexBus are connected to both the upper byte and lower byte of the 16 data lines of the MRAM.

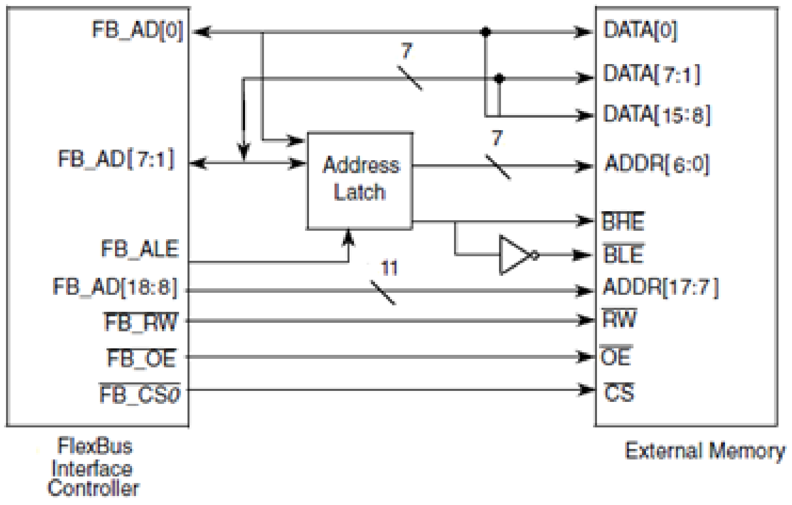


Figure 9. Multiplexed x8 FlexBus to x16 memory

3.2.2 Software setup

The code for multiplexed mode is almost the same as above. For the TWR-K40 in multiplexed mode, the FB_AD[7:0] are used for data, therefore the byte lane must be right justified. The number of pins are different than the K60.

The FlexBus configuration looks like:

```
   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
                ;
```

Also, pins to be used are:

```
//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 Transfer diagrams

With the above configuration chosen, the read and write cycles look like :

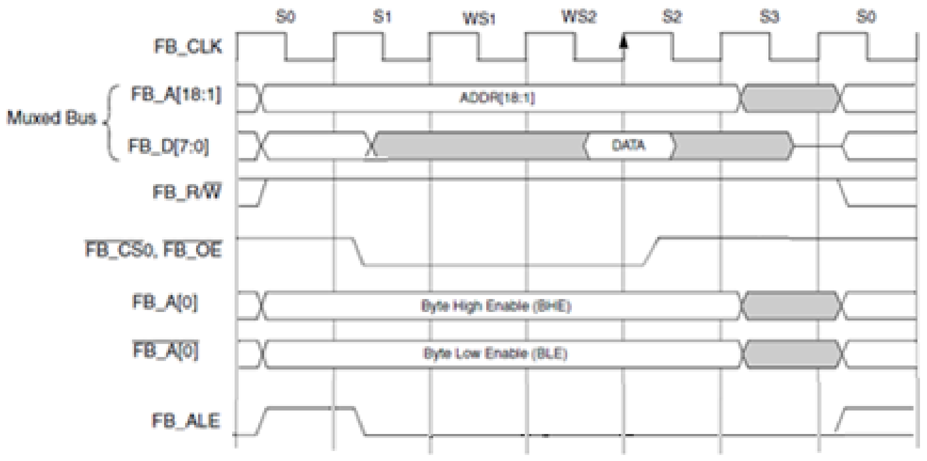


Figure 10. Read (muxed mode 8-bit with two wait states and A0 and inverted A0 used as byte enables)

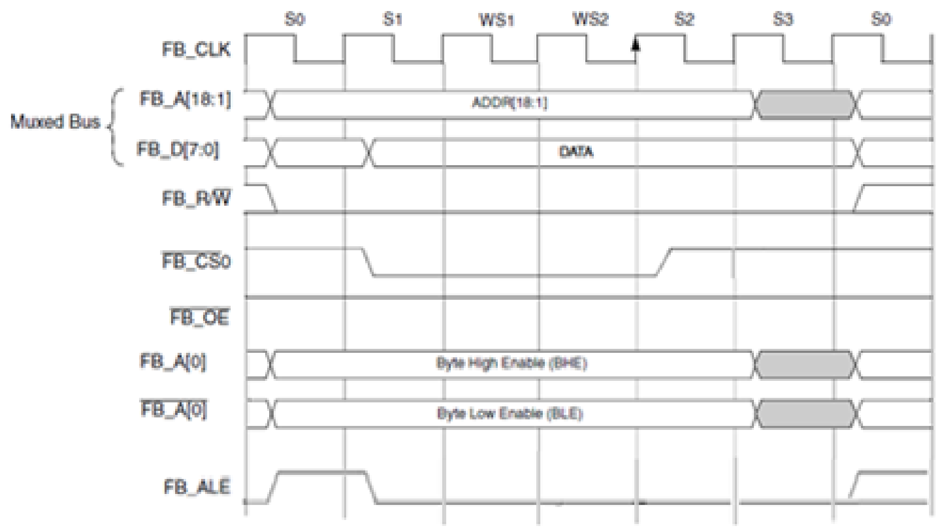


Figure 11. Write (muxed mode 8-bit with two wait states and A0 and inverted A0 used as byte enables)

3.2.4 Performance calculation

In this example, the theoretical maximal throughput is the following.

If you run the bus at 50 Mhz, with transfer cycles taking four cycles plus two wait state cycles. In 8-bit non-muxed mode you can transfer eight bits in 6 cycles, therefore eight bits in 120 ns. This translates to 66.6 Mbit / sec.

3.3 Reading and writing MRAM

Once the MRAM is connected appropriately and FlexBus is initialized, the application is ready to read and write the external MRAM. These operations are simple, you can read any memory location and write directly to a memory address.

MRAM is configured in such a way that the available address range is 0x60000000 – 0x6007FFFF. MRAM_START_ADDRESS is pointing to the first MRAM address. The next code writes a byte (0x5A) in the external memory and then it is read. This is done in the first 16 addresses.

```
 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));
}
```

Now you have 16-bit data. The only restriction to read and write 16-bit data is to access to an even address. The next code writes 16 words in MRAM. Each word is read before it is written. The first word is written in the first address available after writing the 8-bit data.

```
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));
}
```

Finally the same operations with 32-bit data. To read or write a double word the memory address must be a multiple of 4.

```
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 Executing from external MRAM

Executing code form external MRAM is also possible. There are three main steps that must be performed to have executable code in external MRAM:

1. Initialize FlexBus to communicate with external MRAM.
2. Let the MCU know that it must look for the code in external memory. This is done with a compiler pragma and editing the Linker Command File.
3. Copy the code from Flash to external Memory.

This example shows how to relocate a function that makes the LEDs on the K60 blink and execute it from the external MRAM.

- A section is created using `#pragma define_section` and the function wanted to execute from MRAM is located within the section.

```
#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++;
   }
}
```

- The external MRAM address is added as a memory segment. The reason to do this is to let the CPU know where to find `toggle_LEDs(void)` function.

```
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
}
```

A linker section is now created to place the code in the memory segment created. Some labels are added to make the copy from Flash to the external MRAM.

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

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

___MRAMCodeSize = SIZEOF(.myCodeInExtMRAM);
```

Finally, the following code is used to copy the code from flash to the external MRAM.

```
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++;
            }
    }
```

Now the function is relocated in the external MRAM and it is possible to execute it from here.

NOTE

Please notice that execution from MRAM in considerably slower, therefore you must consider these timing specs in your application development.

3.4 LCD

This example describes how to connect a TFT-LCD to the TWR-K60 by means of FlexBus. In this case the TWR-LCD board which has the Solomon Systech TFT-LCD controller driver SSD1289 is used. This controller driver integrates the RAM, power circuits, and so on. It can be interfaced with the common MCU/MPU as 8-bit, 16-bit, and 18-bit 6800-series/8080-series compatible parallel, or a serial peripheral interface. For more information you can see the application note titled *Using Freescale eGUI with the TWR-LCD on MCF51MM family* (document number AN4153)

This particular case uses the 16-bit 6800-series mode. The parallel Interface consists of 16 bi-directional data pins D[17:0], R /W, D/C , E and CS . R /W input high indicates a read operation from the Graphical Display Data RAM (GDDRAM) or the status register. R /W input low indicates a write operation to Display Data RAM or Internal Command Registers depending on the status of the D/C input.

For the correct interface selection, use the below configuration in the LCD.

- PS3 =1, PS2 =0, PS1 =1, PS0 =1. This configuration must be set externally.
- D1 ~ D8 and D10 ~ D17, R/W(/WR), and D/C are used as I/O control. These lines are the ones that are connected to the FlexBus lines.

3.4.1 Schematics

To interface the LCD the FlexBus in 16-bit mode and multiplexed mode are used. The use of FB_ALE is not necessary, therefore the first FlexBus cycle when the address in set-up is ignored. Also the byte-lane shift is right justified, so the data is set-up in the 16 lower significant bits. In other words, FB_AD[0:15] lines are used for data. The FB_AD[16] is used for the D/C or DS input of the SSD1289. The WR and CS signals are connected to FB_RW and FB_CS0 of the K60:

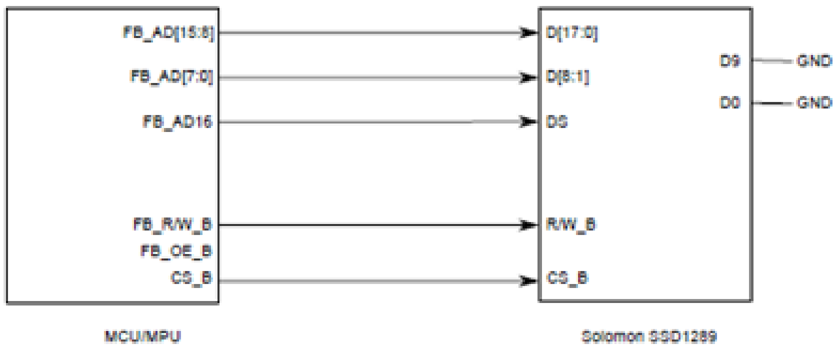


Figure 12. Multiplexed 16-bit FlexBus to TFT-LCD

The E(RD) pin of the LCD controller can be pulled up directly if you do not want to read the pixel data. Also, notice that D0 and D9 of the SSD1289 are not connected. Those pins are not needed in 16-bit mode. You can find more information in the schematics of the TWR-LCD and the TWR-K60.

3.4.2 Software setup

To determine the required adjustments to the FlexBus timing, the FlexBus chapter of the reference manual titled *K60 Sub-Family Reference Manual for 100MHz devices in 144 pin packages* (document number, K60P144M100SF2RM) and the Timing Spec chapter in the datasheet titled *K60 Sub-Family Data Sheet* (document number K60P144M100SF) are used.

- The IO pins required by FlexBus must be configured for the FlexBus function. Because of working in a 16-bit multiplexed mode, only 16 lines for data are needed, 1 line for the D/C signal, RW, and CS signals. Therefoere the K60 pins must be configured in the ALT 5. The LCD and FlexBus initialization are the same as the D4D, you can find more information in the reference manual title *Freescale Embedded GUI (D4D)* (document number 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
```

- The base address is set in the base address field of the FlexBus Chip Select Address Register FB_CSAR for the corresponding chip select. In this example, the TWR-LCD is connected to chip select zero and the base address is 0x60000000. This address is 64-byte aligned, and is in an available memory space.

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

- Chip-Select Registers — CSMR register specifies the address mark and allowable access types for the respective chip-select. Setting the base address mask to 1 (CSMR0[BAM]) = 1) means that the block size for the CS is 2^n (n = number of bits in CSMR[BAM] + 16) must be 128 KB. This means address 0x60000000 to 0x6001FFFF can be accessed by the CS. Therefore address 0x6000000 (FB_AD16 is low) can be used to access the index register of the SSD1289, and address 0x60010000 (FB_AD16 is high) can be used to access the data buffer of the SSD1289.

```
#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
```

- When connecting the CS0 to the CS_B of the LCD controller the CSMR0[V] must be set to enable the CSn. At reset, no chip-select other than FB_CS0 can be used until the CSMR0[V] is set. According to the datasheet of the SSD1289m the minimum write cycle time is 100 ns. And there are at least four FB cycles during read and write cycles. The FB_CLK should not be higher than 40 MHz or wait states must be added.

Table 2. SSD1289 Spec

Symbol	Parameter	Min	Type	Max	Unit
t_{cycle}	Clock cycle time (write cycle)	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
```

- For sending command and data the below functions were used. Notice that the only difference is the address where the data is sent. 0x6000000 (FB_AD16 is low) is used to access the index register and address 0x60010000 (FB_AD16 is high) to access the data buffer.

```
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 Transfer diagrams

With the above configuration chosen, the read and write cycles look like:

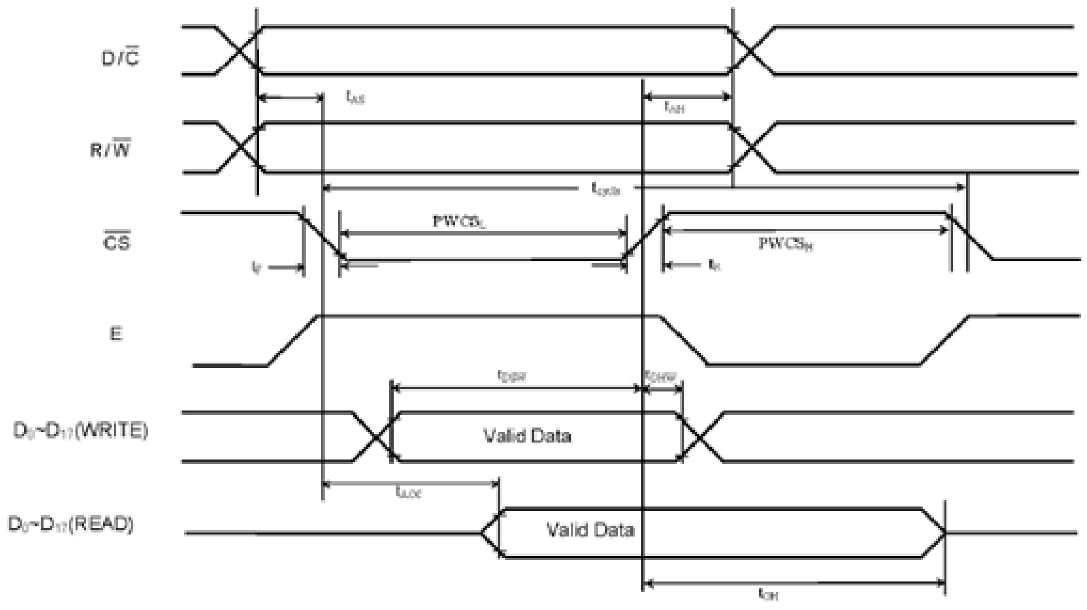


Figure 13. SSD1289 Transfer Diagram

The specs shown in Figure 10 can be found in the SSD1289 datasheet. These specs to correctly configure the Flexbus module.

3.5 Using FlexBus in a MQX application

Using FlexBus in an MQX application is very similar to a bareboard application. You can use the same register definitions to configure the FlexBus settings. MQX configures the FlexBus module when using the TWR-K40 and TWR-K60 to communicate with the external MRAM.. It is possible to reconfigure this module to meet your application requirements.

3.5.1 Application description

When an application runs out of internal memory, it is not possible to create new tasks. This example shows how to configure FlexBus to communicate with an external MRAM and expand the memory pool of MQX.

3.5.2 Software setup

As seen the in the code below, the total amount of memory space required by all the tasks exceeds the K60 internal RAM size.

```
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,                 }
};
```

The Main_task initizalizes FlexBus. This initialization is the same as the one shown in section 3.4.2 Software Setup. After this, you must let the MQX kernel know that there is external memory available. This is done by using the _mem_extend() function. The parameters this function receives are the pointer to the start address and the size of the external memory block.

```
/*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);
      }
}
```

To enable the _mem_extend() function it is necessary to make a few changes in the user_config.h file and rebuild the BSP and PSP libraries. See the necessary changes below:

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

To rebuild the BSP and PSP libraries refer to FSL_MQX_getting_started.pdf in the MQX documentation folder.

4 PCB design recommendations

Due to the critical timing required while driving external memories, there are a number of considerations that must be taken into account during PCB layout.

Each group of signal traces must have identical loading and similar routing to maintain timing and signal integrity.

Control and clock signals are routed point-to-point. Components could and should be placed as close as possible to the MCU. To avoid crosstalk, keep address and command signals separate (that is, a different routing layer) from the data and data strobes.

5 Conclusion

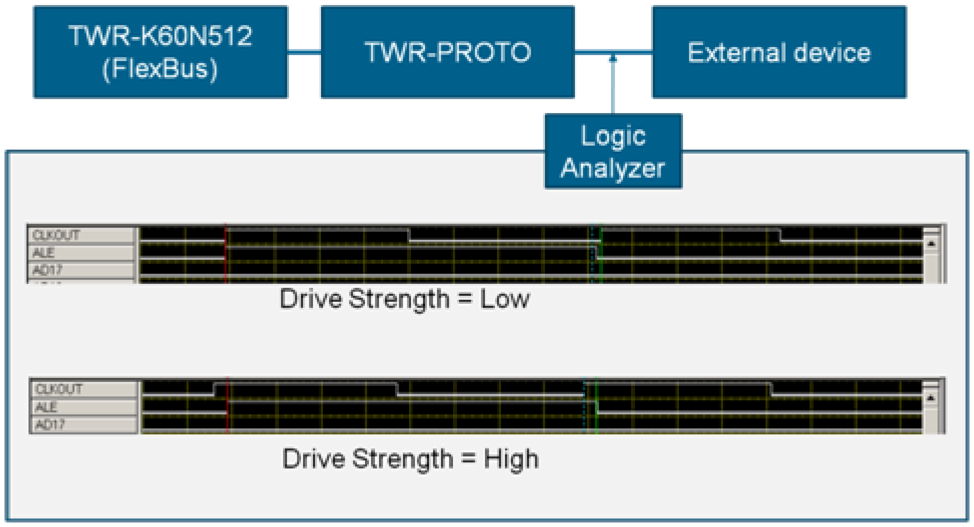
FlexBus is an interface that can be helpful to communicate with external devices in a simple way. After configuring Flexbus, it is easy to use as operations are transparent for users. For example, accessing to an external memory is the same process as accessing to internal memory because you can write directly to the memory address.

Due to its multiple connection configurations, FlexBus provides the flexibility to determine the optimum balance between speed and footprint size.

All these features make FlexBus one of the best options to communicate with almost every parallel communication device.

Appendix A Getting the correct FlexBus signal by using the TWR-PROTO

The TWR-PROTO is used to get the FlexBus signal in a lot case when the user's logic is connected with the Kinetis Tower Board. However, these signal output timings depend on the load capacitors, board layout, room temperature and so on. Sometimes the correct wave of the FlexBus from the TWR-PROTO cannot be obtained. To help customers with this issue, this document describes how to get the correct FlexBus signal by using the TWR-PROTO and TWR-K60N512. You can get the correct wave on the TWR-PROTO if the drive strength setting is high. Shown is the sample code of setting the high drive strength and the monitored wave by using the TWR-PROTO.



A.1 Point at issue

In the case of connecting the TWR-K60 with ASIC by using FlexBus, generally the original circuit is created by using TWR-PROTO. Figure A-1 shows the FlexBus signal waves on the TWR-PROTO by using the logic analyzer.

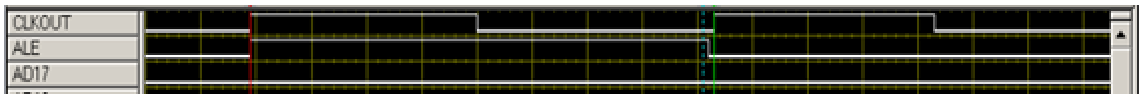


Figure A-1. FlexBus signals wave

Figure A-2 is the wave shown in reference manual in the case of a write transaction.

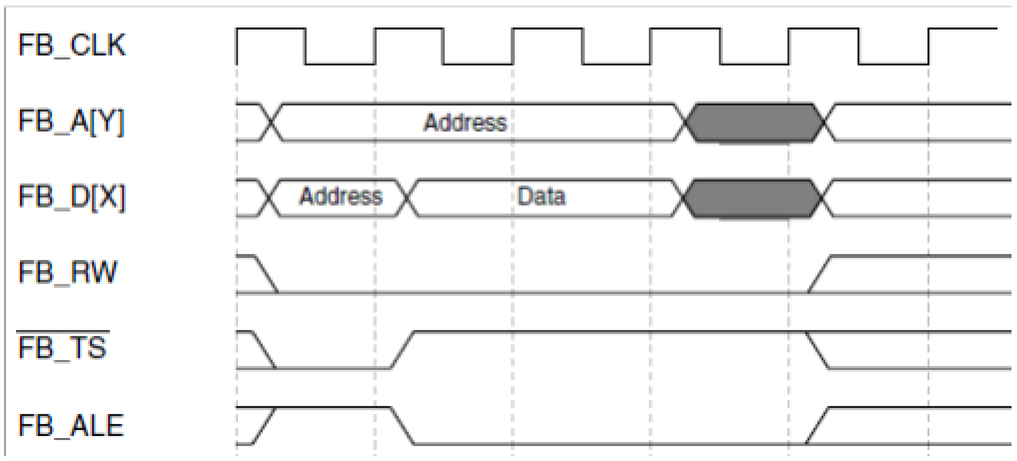


Figure A-2. Basic write byte cyle

The observed FB_CLK and FB_ALE in Figure A-1 rise almost at the same time and FB_ALE falls before the next FB_CLK rising edge. However the FB_ALE in Figure A-2 falls after the second FB_CLK rising edge. Therefore, it is impossible for the synchronous device to correctly sample the FlexBus signals.

The reason for this, is that FB_CLK (PTC3) is bifurcated to use PWM3, see Figure A-3. So, this pin is overloaded and delayed compared with other FlexBus signals.

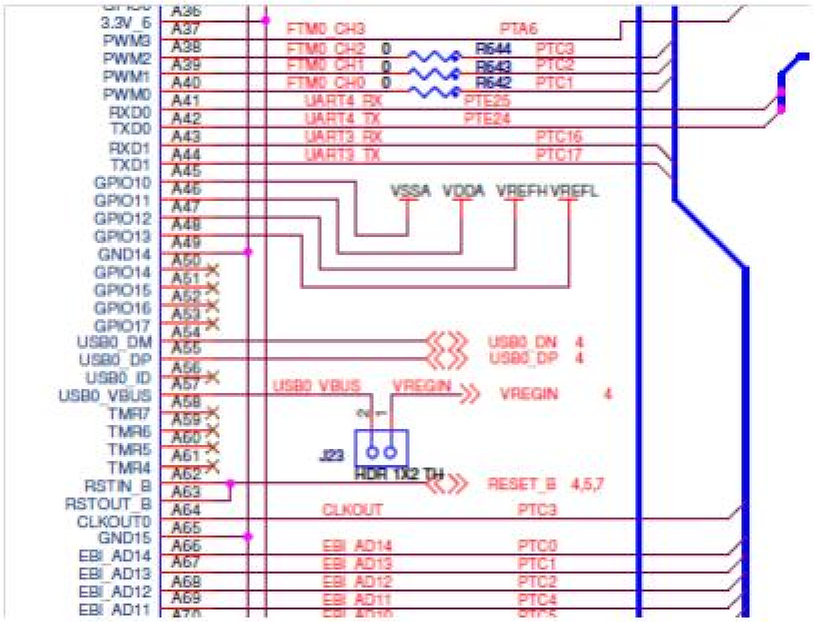


Figure A-3. Part of the TWR-K60 circuit

Appendix B Countermeasure

B.1 Countermeasure configuration

FB_CLK is delayed compared with other FlexBus signals observed on TWR-PROTO from the reason of the previous captor. The countermeasure is:

Set drive strength High on the output pin

Due to set the drive strength High, the output current from Kinetis is raised and Kinetis can output the signal where the tower board load capacity is not lost. To raise the drive strength, set PORTx_PCRn[DSE] = 1. FB_CLK and PTC3 on K60 are multiplexed. Therefore, you need to set PORTC_PCR3[DSE] = 1 to raise the drive strength of FB_CLK.

B.2 Sample code

Header code:

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

Program code:

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

Insert the above code in your code.

B.3 Result

Figure B-1 shows the wave in case of setting the drive strength High.

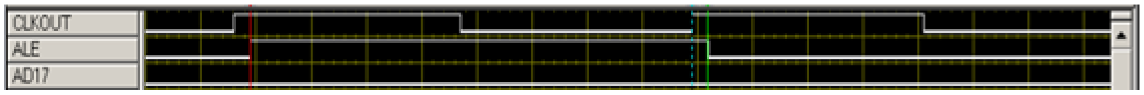


Figure B-1. Countermeasure result wave of FlexBus

As in Figure A-2 FB_ALE (ALE in Figure B-1) is asserted after the first FB_CLK (CLKOUT in Figure B-1) rising edge and FB_ALE is negated after the second FB_CLK rising edge. Then the synchronous device connected with Kinetis via FlexBus can sample the signals correctly.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: AN4393
Rev. 0, 05/2012