AN205546

F²MC-16FX Family, External Bus Interface

This application note describes the functionality of the External Bus Interface and gives some hardware and software examples.

Contents

1 Introduction

This application note describes the functionality of the External Bus Interface and gives some hardware and software examples.

The External Bus Interface is used to connect additional hardware to the MCU such like RAM, Flash-Memory or other controllers.

1.1 Key Features

- Data Bus Width 8 or 16 Bit
- Big or little Endian selectable for Data Bus
- Up to 24 Bit Address Bus (depending on Device)
- Multiplexed Bus (and non-multiplexed Bus at Devices with high Pin Count)
- Up to 6 Chip Select Signals with programmable Memory Area
- 0, 1, 2, 3, 4, 8, 16, and 32 Wait states can be inserted for a Read/Write Cycle
- Address, Read, Write, Upper-Byte and Lower-byte Strobe Signals
- Polarity of Address Strobe selectable
- Address Cycle extendable to 2 Clock Cycles
- External Clock Signal can be enabled, the polarity is selectable, the frequency can be divided up to 128 in 2^n-Steps
- Ready and Hold Function

2 The External Bus Interface

The Basic Functionality of the External Bus Interface

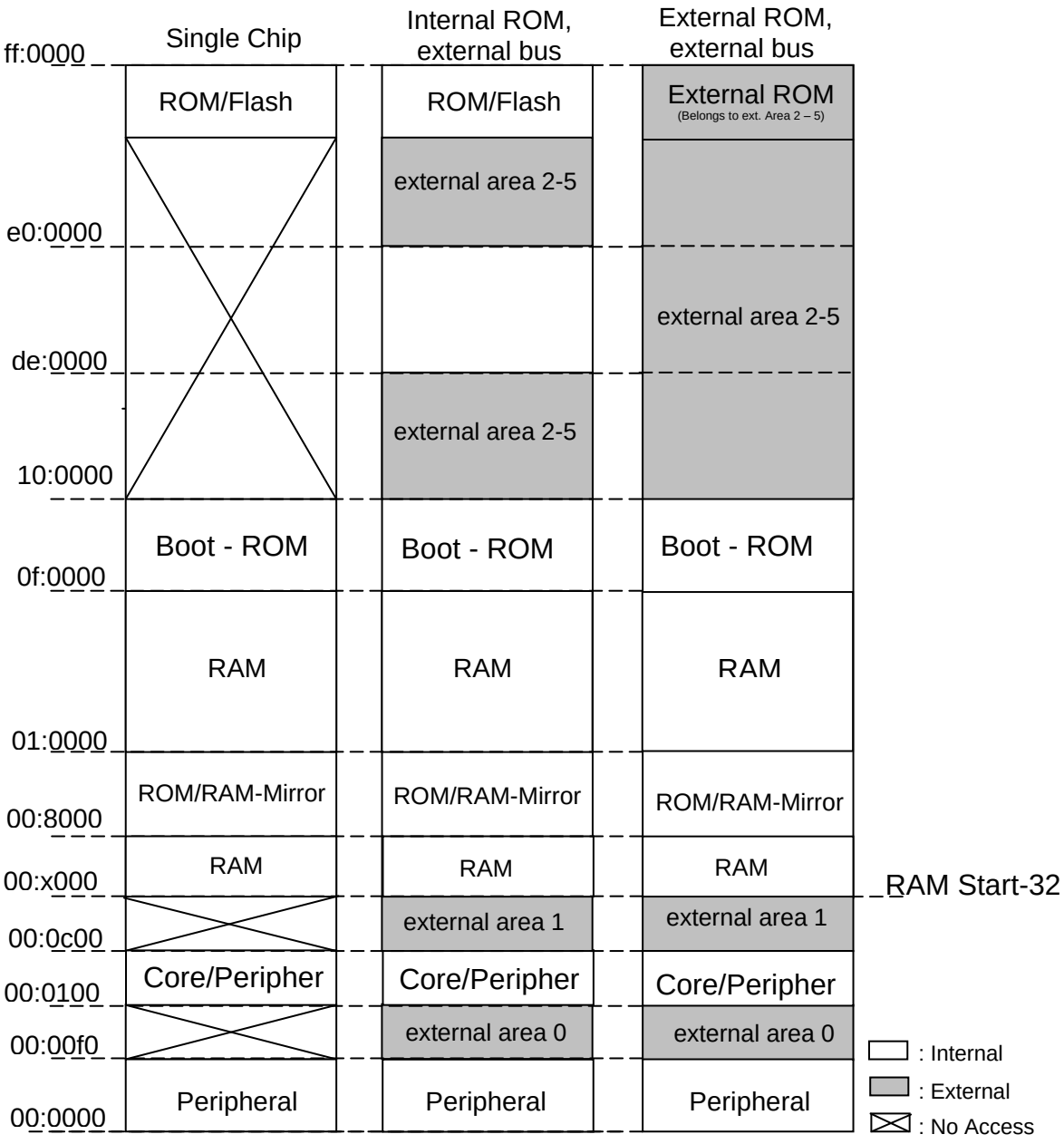
2.1 Outline

The External Bus Interface allows the user to connect external peripherals or memory to the MCU. The Bus consists of data, address and control signals. Various settings are possible for several bus timings.

2.1.1 Address Maps of Bus Modes

Figure 1 shows the different address maps for the three MCU bus modes.

Figure 1. Address Maps of Bus Modes



2.2 Registers

2.2.1 External Bus Mode Register (EBM)

Bit No.	Name	Explanation	Initial Value	Value	Operation
7	NMS	Non-multiplexed Bus select	0	0	Address/Data multiplexed
				1	Address/Data non-multiplexed
6	ERE	External ROM enable	0	0	Disable External ROM
				1	Enable External ROM
5	EAE5	External Area 5 Select	0	0	Disable 0xC00000-0xFFFFFF
				1	Enable 0xC00000-0xFFFFFF
4	EAE4	External Area 4 Select	0	0	Disable 0x800000-0xBFFFFF
				1	Enable 0x800000-0xBFFFFF
3	EAE3	External Area 3 Select	0	0	Disable 0x400000-0x7FFFFF
				1	Enable 0x400000-0x7FFFFF
2	EAE2	External Area 2 Select	0	0	Disable 0x100000-0x3FFFFF
				1	Enable 0x100000-0x3FFFFF
1	EAE1	External Area 1 Select	0	0	Disable 0x000C00-(RAM-Start – 0x21)
				1	Enable 0x000C00-(RAM-Start – 0x21)
0	EAE0	External Area 0 Select	0	0	Disable 0x0000F0-0x0000FF
				1	Enable 0x0000F0-0x0000FF

2.2.2 External Bus Clock and Function Register (EBCF)

Bit No.	Name	Explanation	Initial Value	Value	Operation
15	HDE	Hold function enable	0	0	Disable Hold Function
				1	Enable Hold Function
14	RYE	External Ready Function enable	0	0	Disable external Ready Function
				1	Enable external Ready Function
13	CKE	External Bus Clock Output Enable	0	0	Enable I/O Port Operation
				1	Enable external Bus Clock Output
12	CKI	External Bus Clock Inversion Control	0	0	Disable Clock Output Inversion
				1	Enable Clock Output Inversion
11	CSM	External Bus Clock Suspend Mode Control	0	0	External Bus Clock is always output
				1	External Bus Clock is only output during data transfer

Bit No.	Name	Explanation	Initial Value	Value	Operation
10 … 8	DIV2 … DIV0	External Clock Divider (ECLK)	0, 0, 0	0, 0, 0	CLKB bypassed to ECLK
				0, 0, 1	CLKB is divided by 2 for ECLK
				0, 1, 0	CLKB is divided by 4 for ECLK
				0, 1, 1	CLKB is divided by 8 for ECLK
				1, 0, 0	CLKB is divided by 16 for ECLK
				1, 0, 1	CLKB is divided by 32 for ECLK
				1, 1, 0	CLKB is divided by 64 for ECLK
				1, 1, 1	CLKB is divided by 128 for ECLK

2.2.3 External Bus Control Signal Register (EBCS)

Bit No.	Name	Explanation	Initial Value	Value	Operation
15	-	*Reserved*	X	-	Read: Undefined Write: Always write “0”
14	ASL	Address Strobe Level Select	1	0	Low-active: ASX
				1	High-active: ALE
13	ASE	Address Strobe Output Enable	0	0	Enable I/O Port
				1	Enable Address Strobe
12	RDE	Read Strobe Output Enable	0	0	Enable I/O Port
				1	Enable Read Strobe Output
11	WRHE	Write Strobe (WRHX) enable	0	0	Enable I/O Port
				1	Enable WRHX Output
10	WRLE	Write Strobe (WRLX/WRX) enable	0	0	Enable I/O Port
				1	Enable WRLX/WRX Output
9	UBE	Upper Byte (UBX) Output Enable	0	0	Enable I/O Port
				1	Enable UBX Output
8	LBE	Lower Byte (LBX) Output Enable	0	0	Enable I/O Port
				1	Enable LBX Output

2.2.4 External Area Configuration Register (Lower Byte) (EACL[5:0])

Bit No.	Name	Explanation	Initial Value	Value	Operation
7	BW	External Bus Data Width	0*[1]	0	16-Bit Bus Width
				1	8-Bit Bus Width
6	ES	Endian Select	0	0	Select little endian
				1	Select big endian
5	WSF	Write Strobe function of WRLX/WRX pin	0	0	Enable WRLX
				1	Enable WRX

Bit No.	Name	Explanation	Initial Value	Value	Operation
4	STS	Strobe time scheme select	0	0	Enable Address Strobe during first half of Address on Bus
				1	Enable Address Strobe during whole time of Address on Bus
3	ACE	Address Cycle Extension	0	0	Disable Cycle Extension
				1	Enable One Bus Cycle Extension
2 … 0	R2 … R0	Automatic Ready Function	0, 0, 0 0, 1, 1*2	0, 0, 0	Disable Automatic Ready Function
				0, 0, 1	Automatic Wait of 1 Cycle
				0, 1, 0	Automatic Wait of 2 Cycles
				0, 1, 1	Automatic Wait of 3 Cycles
				1, 0, 0	Automatic Wait of 4 Cycles
				1, 0, 1	Automatic Wait of 8 Cycles
				1, 1, 0	Automatic Wait of 16 Cycles
				1, 1, 1	Automatic Wait of 32 Cycles

*1 1 for EACL5 after external start vector fetch in 8-Bit Mode
*2 1 for EACL5 after external start vector fetch

Note: For each of the 6 external memory areas a corresponding EACLn exists.

2.2.5 External Area Configuration Register (Upper Byte) (EACH[5:0])

Bit No.	Name	Explanation	Initial Value	Value	Operation
15	-	*Reserved*	X	-	Read: Undefined Write: Always write “0”
14	-	*Reserved*	X	-	Read: Undefined Write: Always write “0”
13	ATL	Access Type Limitation	0	0	Enable Code Fetch and Data Read
				1	Enable only Data Read
12	CSL	Chip Select Level Select	0	0	Enable CSX
				1	Enable CS
11	CSE	Chip Select Output Enable	0	0	Enable I/O Port
				1	Enable Chip Select Output
10 … 8	EASZ2 … EASZ0	External Area Size	1, 1, 0 0, 1, 1*2	0, 0, 0	64 K Bytes, EASn_A[7:0] valid
				0, 0, 1	128 K Bytes, EASn_A[7:1] valid
				0, 1, 0	256 K Bytes, EASn_A[7:2] valid
				0, 1, 1	512 K Bytes, EASn_A[7:3] valid
				1, 0, 0	1 M Byte, EASn_A[7:4] valid
				1, 0, 1	2 M Bytes, EASn_A[7:5] valid
				1, 1, 0	4 M Bytes, EASn_A[7:6] valid
				1, 1, 1	8 M Bytes, EASn_A7 valid

*2 1 for EACL5 after external start vector fetch

Note: For each of the 6 external memory areas a corresponding EACHn exists.

The External Area Size can be selected only for areas 2 – 5. The size is selected by masking the address lines A16 – A22 (depending on configuration), which correspond of the lower bits of External Area Select Register (EAS). By masking address lines, the selected area is automatically aligned to its own size. Refer 0

Example Settings of EAS[5:2] for examples.

2.2.6 External Area Select Register (EAS[5:2])

For areas 2 to 5 an External Area Select Register exists. This Byte Register holds the address bits for the upper 8-Bit address comparison (address bank) for the chip select signals. It is also the start bank of the chip select. The end address is defined by these registers and the corresponding area size defined by EACHn_EASZ[2:0].

The following table shows the initial values.

External Area	Initial Values	Default Memory Area		Chip Select
		Start	End	
2	EAS2 = 0x10 EACH2_EASZ[2:0] = 110	0x100000	0x3FFFFF	CS2(X)
3	EAS3 = 0x40 EACH3_EASZ[2:0] = 110	0x400000	0x7FFFFF	CS3(X)
4	EAS4 = 0x80 EACH4_EASZ[2:0] = 110	0x800000	0xBFFFFF	CS4(X)
5	EAS5 = 0xC0 EACH5_EASZ[2:0] = 110	0xC00000	0xFFFFFF	CS5(X)

Note, that all areas except area 0 – 2 have a default size of 4 M Bytes. The default size of area 2 is 3 M Bytes, and the size of area 0 and 1 cannot be changed.

The External Area Select Register can point to any bank within the memory range that is selected in combination with EACH_EASZ bits.

Example Settings of EAS[5:2]

Address	EASZ: 1M EAS: A7, A6, A5, A4	EASZ: 2M EAS: A7, A6, A5	EASZ: 4M EAS: A7, A6	EASZ: 8M EAS: A7
$FFFFFF_H$ $F00000_H$	1, 1, 1, 1	1, 1, 1	1, 1	1
$EFFFFF_H$ $E00000_H$	1, 1, 1, 0			
$DFFFFF_H$ $D00000_H$	1, 1, 0, 1	1, 1, 0		
$CFFFFF_H$ $C00000_H$	1, 1, 0 ,0			
$BFFFFF_H$ $B00000_H$	1, 0, 1, 1	1, 0, 1	1, 0	
$AFFFFF_H$ $A00000_H$	1, 0, 1, 0			
$9FFFFF_H$ 900000_H	1, 0, 0, 1	1, 0, 0		
$8FFFFF_H$ 800000_H	1, 0, 0, 0			
$7FFFFF_H$ 700000_H	0, 1, 1, 1	0, 1, 1	0, 1	0
$6FFFFF_H$ 600000_H	0, 1, 1, 0			
$5FFFFF_H$ 500000_H	0, 1, 0, 1	0, 1, 0		
$4FFFFF_H$ 400000_H	0, 1, 0, 0			
$3FFFFF_H$ 300000_H	0, 0, 1, 1	0, 0, 1	0, 0	
$2FFFFF_H$ 200000_H	0, 0, 1, 0			
$1FFFFF_H$ 100000_H	0, 0, 0, 1	0, 0, 0		

It is possible to set individual areas, when using different area sizes for EACH_EASZ bits. The following table shows such an example configuration.

External Area	Initial Values	Default Memory Area		Chip Select
		Start	End	
2	EAS2 = 0x18 EACH2_EASZ[2:0] = 001	0x180000	0x19FFFF	CS2(X)
3	EAS3 = 0x60 EACH3_EASZ[2:0] = 010	0x600000	0x63FFFF	CS3(X)
4	EAS4 = 0x73 EACH4_EASZ[2:0] = 000	0x700000	0x73FFFF	CS4(X)
5	EAS5 = 0xFE EACH5_EASZ[2:0] = 001	0xFE0000	0xFFFFFF	CS5(X)

2.2.7 External Bus Address Output Enable Register (EBAE[2:0])

This 24-Bit Register (2 words) contains the output enable for the 24 address pins. Bit 0 is assigned to AD0, Bit 1 to AD1, and so on. The upper byte of the second word is ignored. The initial values are “0”, if internal reset vector fetch is used.

These enable bits are used to define the minimum of address pins needed and save the I/O port number.

Please note, that after external reset vector fetch all bits are “1”, i. e. all address lines are used.

2.2.8 Port Input Enable

Please also note, that the corresponding Port Input Enable Registers for the used data and control pins (if used) have to be enabled, otherwise data/control input does not work (always “0” is read).

Series	Data Bus Width		HRQ	RDY
	8 Bit	16 Bit		
MB96340/350	PIER00	PIER00, PIER01	PIER03_IE4	PIER03_IE6
MB96380	PIER01	PIER01, PIER02	PIER12_IE7	PIER00_IE2
MB96320	PIER00	PIER00, PIER01	PIER03_IE4	PIER03_IE6

3 Initialization in *Start.asm*

Initialization of the External Bus Inerface in Start.asm

3.1 *Start.asm*

In the startup file *Start.asm*, which is included in our template project, the External Bus Interface can be initialized before branching to the application. Therefore the application itself does not need to set up the External Bus Interface, but use it from the beginning on.

The user can adjust the setting in the lines with a “<<<” in the comments on the right side.

3.1.1 Enabling the External Bus Interface

```
;=====================================================================
; 4.7  External Bus Interface
;=====================================================================

#set      SINGLE_CHIP    0         ; all internal
#set      INTROM_EXTBUS  1         ; mask ROM or FLASH memory used
#set      EXTROM_EXTBUS  2         ; full external bus (INROM not used)

#set      BUSMODE INTROM_EXTBUS    ; <<< set bus mode (see mode pins)

#set      MULTIPLEXED     0        ;
#set      NON_MULTIPLEXED 1        ; only if supported by the device

#set      ADDRESSMODE MULTIPLEXED  ; <<< set address-mode

; Some devices support multiplexed and/or non-multiplexed Bus mode
; please refer to the related datasheet/hardware manual
```

BUSMODE INTROM_EXTBUS set the External Bus Interface to internal Flash memory and the external memory enabled. The setting corresponds to the EBM_ERE bit (2.2.1).

ADRESSMODE MULTIPLEXED sets the Interface to address/data bus in multiplexed mode (EBM_NMS)(2.2.1).

3.1.2 Enabling Chip Select

```
; Select the used Chip Select areas
#set      CHIP_SELECT0    OFF      ; <<< enable chip select area 0
#set      CHIP_SELECT1    OFF      ; <<< enable chip select area 1
#set      CHIP_SELECT2    OFF      ; <<< enable chip select area 2
#set      CHIP_SELECT3    ON       ; <<< enable chip select area 3
#set      CHIP_SELECT4    OFF      ; <<< enable chip select area 4
#set      CHIP_SELECT5    OFF      ; <<< enable chip select area 5
```

ON sets the corresponding Chip Select signal. Here it is for example CSX3 for the default area 0x400000 – 0x7FFFFF. This setting adjusts then EMB_EAE[5:0] bits (2.2.1).

3.1.3 Enabling Clock Timing Signals

```
#set      HOLD_REQ           OFF   ; <<< select Hold function
#set      EXT_READY          OFF   ; <<< select external Ready function
#set      EXT_CLOCK_ENABLE   ON   ; <<< select external bus clock output
#set      EXT_CLOCK_INVERT   OFF   ; <<< select clock inversion
#set      EXT_CLOCK_SUSPEND  OFF   ; <<< select if external clock is suspended
                                   ;     when no transfer in progress
```

These settings control several bits of the EBCF Register (2.2.2). In this example none of the options is set.

3.1.4 Setting Clock Divider

```
; The external bus clock is derived from core clock CLKB. Select the divider
;   for the external bus clock.

#set      EXT_CLOCK_DIV1     0
#set      EXT_CLOCK_DIV2     1
#set      EXT_CLOCK_DIV4     2
#set      EXT_CLOCK_DIV8     3
#set      EXT_CLOCK_DIV16    4
#set      EXT_CLOCK_DIV32    5
#set      EXT_CLOCK_DIV64    6
#set      EXT_CLOCK_DIV128   7

#set      EXT_CLOCK_DIVISION  EXT_CLOCK_DIV1 ; <<< select clock divider
```

In this example the clock for the External Bus Interface is not divided by the option EXT_CLOCK_DIV1. This setting controls EBCF_DIV[2:0] (2.2.2).

3.1.5 Selecting used Address Pins

With the following setting, the used address pins, corresponding to the size of the external memory space, are set to address output.

```
#set      ADDR_PINS_23_16    B'11111111     ; <<< select used address lines
                                            ;     A23..A16 to be output.
#set      ADDR_PINS_15_8     B'11111111     ; <<< select used address lines
                                            ;     A15..A8 to be output.
#set      ADDR_PINS_7_0      B'11111111     ; <<< select used address lines
                                            ;     A7..A0 to be output.
```

In this example the address lines A0 – A21 are used, which represents a memory space of 4 M Bytes. These bits are those of the 2 EBAE registers (2.2.7).

3.1.6 Setting Bus Control Lines

The settings for the EBCS register (2.2.3) can be found in the following settings. They mostly depend on the connected type of memory.

In this case all lines except HWRX are selected.

```
#set      LOW_BYTE_SIGNAL     ON    ; select low byte signal LBX
#set      HIGH_BYTE_SIGNAL    ON    ; select high byte signal UBX
#set      LOW_WRITE_STROBE    ON    ; select write strobe signal WRLX/WRX
#set      HIGH_WRITE_STROBE   OFF   ; select write strobe signal WRHX
#set      READ_STROBE         ON    ; select read strobe signal RDX
#set      ADDRESS_STROBE      ON    ; select address strobe signal ALE/ASX
#set      ADDRESS_STROBE_LVL  ON    ; select address strobe function:
                                    ;   OFF - active low; ON - active high
```

3.1.7 Setting External Area Configuration

In the following bit definitions the settings of the EACL/H register (2.2.4/2.2.5) is done. Please note, that for each external memory area, an own setting block exists (0 – 5). In the following code the settings for area 4 are done.

```
#set      CS3_CONFIG  B'0000111000100000    ; <<< select Chip Select Area 0 configuration
;                       |||||||||||||+++-- Automatic wait cycles (0: 0, 1: 1, 2: 2, 3: 3,
;                       |||||||||||||      4: 4, 5: 8, 6: 16, 7: 32)
;                       ||||||||||||+----- Address Cycle Extension (0: not extended,
;                       ||||||||||||       1: extension by 1 cycle)
;                       |||||||||||+------ Strobe timing (0: scheme 0, 1: scheme 1)
;                       ||||||||||+------- Write strobe function (0: WRLX strobe,
;                       ||||||||||         1: WRX strobe)
;                       |||||||||+-------- Endianess (0: little endian, 1: big endian)
;                       ||||||||+--------- Bus width (0: 16bit, 1: 8bit)
;                       |||||+++---------- External area size (0: 64kB, 1: 128kB,
;                       |||||              2: 256kB, 3: 512kB, 4: 1MB, 5: 2MB, 6: 4MB,
;                       |||||              7: 8MB)
;                       ||||+------------- Chip Select output enable (0: CS disabled,
;                       ||||               1: CS enabled)
;                       |||+-------------- Chip Select level (0: low active,
;                       |||                1: high active)
;                       ||+--------------- Access type limitation (0: code and data,
;                       ||                 1: data only)
;                      ++---------------- ignored
```

For correct bit position please follow the hinted lines down/right and left/up.

3.1.8 External Memory Program Bank

```
#set      CS2_START   0x00        ; chip select area
#set      CS3_START   0x40        ; chip select area
#set      CS4_START   0x80        ; chip select area
#set      CS5_START   0xC0        ; chip select area
```

These settings control the EAS[5:0] registers (2.2.6). Here are the default settings shown.

3.1.9 Automatic Port Enable

The start.asm code of our template project sets automatically the corresponding port input enable registers for the data and control lines. Its code depends on the used device and data bus width.

The device should be defined by SERIES.

```
;=====================================================================
; 4.1  Controller Series
;=====================================================================
#set      MB96320   0
#set      MB96340   1
#set      MB96350   2
#set      MB96360   3
#set      MB96380   4

#set      SERIES    MB96380        ; <<< select Series
```

```
#  if SERIES == MB96340 || SERIES == MB96350
            MOV  PIER00,#0xFF
#    if (CS0_CONFIG & 0x0080) == 0 || (CS1_CONFIG & 0x0080) == 0 ||
        (CS2_CONFIG & 0x0080) == 0 || (CS3_CONFIG & 0x0080) == 0 ||
        (CS4_CONFIG & 0x0080) == 0 || (CS5_CONFIG & 0x0080) == 0
            MOV  PIER01,#0xFF
#    endif
#    if HOLD_REQ == ON
            SETB PIER03:4
#    endif
#    if EXT_READY == ON
            SETB PIER03:6
#    endif
#  else if SERIES == MB96380
            MOV  PIER01,#0xFF
#    if (CS0_CONFIG & 0x0080) == 0 || (CS1_CONFIG & 0x0080) == 0 ||
        (CS2_CONFIG & 0x0080) == 0 || (CS3_CONFIG & 0x0080) == 0 ||
        (CS4_CONFIG & 0x0080) == 0 || (CS5_CONFIG & 0x0080) == 0
            MOV  PIER02,#0xFF
#    endif
#    if HOLD_REQ == ON
            SETB PIER12:7
#    endif
#    if EXT_READY == ON
            SETB PIER00:2
#    endif
#  endif
```

Note, that the MB96310 series does not have the External Bus Interface feature.

Important Note:

Please also see the caution remarks given in chapter 0.

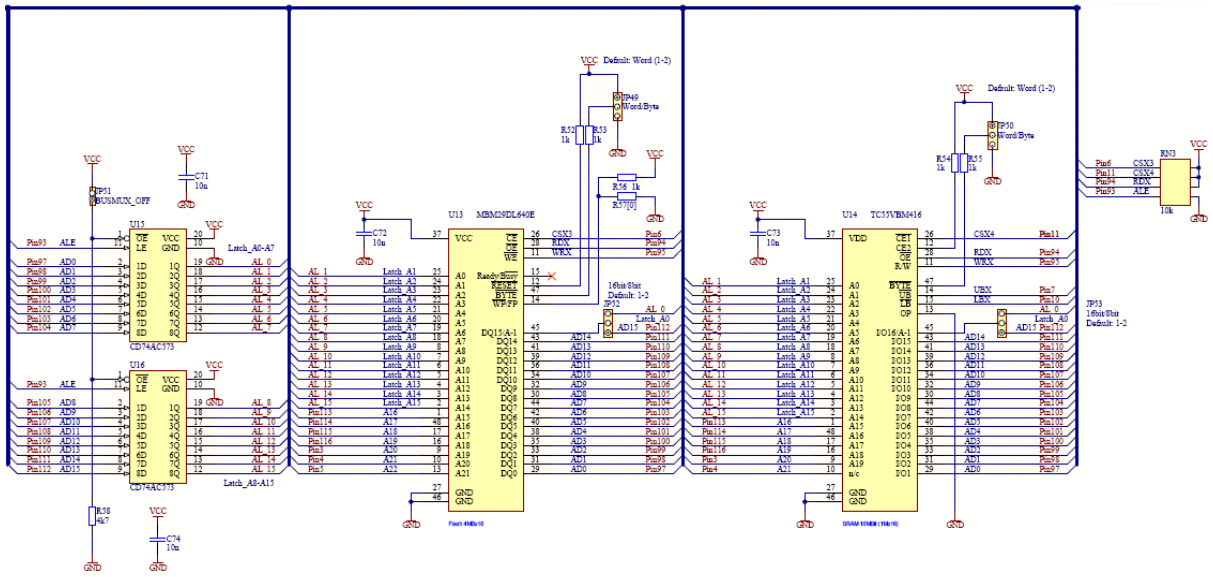
4 External Bus Interface Examples

Examples for the External Bus Interface

4.1 Hardware Example for Multiplex Mode

The following example schematic shows a possible address decoding of the multiplexed bus for external memory.

Figure 2. Memory Interface Example Schematic



Please note, that in this example a 16 bit wide data bus is used. To de multiplex address and data line 2- 8bit latches (CD74AC573) are used. Address latch enable (ALE) is used to latch the address and connected to Address bus of Flash and RAM chip.

Flash chip MBM29DL640E is configured for 16-bit data bus with the help of jumper JP49 and JP52. Similarly RAM chip TC55VBM416 is configured for 16-bit data bus with the help of jumper JP50 and JP53. BYTEX pin selects byte (8-bit) mode or word (16-bit) mode for the device. When this pin is driven high, the device operates in word (16-bit) mode. When this pin is driven low, the device operates in byte (8-bit) mode.

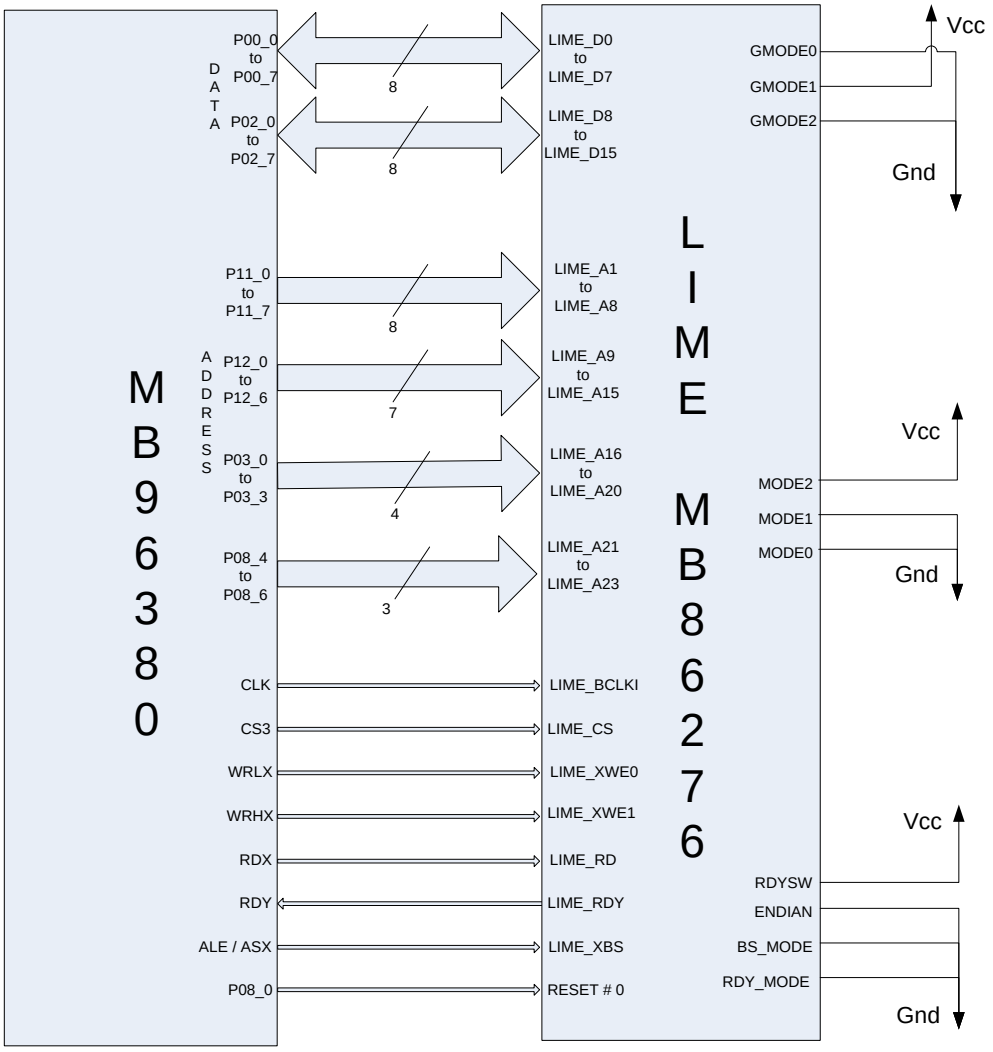
As can be seen from above diagram, CSX3 is used for MBM29DL640E and CSX4 is used for TC55VBM416.

Some external memories may also need the `UBX` and `LBX` signal.

The basic configuration for this example is `EACL_STS = 0` and `EACL_ACE = 0`.

4.2 Hardware Example for Non Multiplexed Mode

Figure 3. Lime Graphic Design Controller Interfaced with Non-Multiplex Address/Data Bus



Above block diagram shows connection between MCU and Lime GDC MB86276. Here Lime GDC is interfaced on Non-multiplexed address and Data bus.

In the above example, higher order address line, LIME_A21 to LIME_A23, of Lime GDC is driven by PORT08. Lime GDC is used in 16bit mode (MODE2:0 = B'100) and hence MCU address line A0 to A19 is connected to LIME_A1 to LIME_A20.

MCU data line AD0 to AD15 is connected to LIME_D0 to LIME_D15.

CSX3 of MCU is connected to LIME_CS. WRLX, WRHX and RDX of MCU is connected to LIME_XWE0, LIME_XWE1 and LIME_RD respectively.

LIME_RDY signal is connected to MCU RDY, hence MCU wait for ready signal to go active before writing or reading data from Lime GDC. LIME_RDYSW and LIME_RDY_MODE, select active level of LIME_RDY output signal.

4.3 Hardware Example for Chip Select using external address decoder

Assume that a 16K word E^{2}PROM with a starting address of `0x600000` and a 16K word SRAM with starting address of `0x700000` need to be connected to MCU using external bus interface

Accordingly E^{2}PROM will have memory space starting from `0x600000` to `0x607FFF`

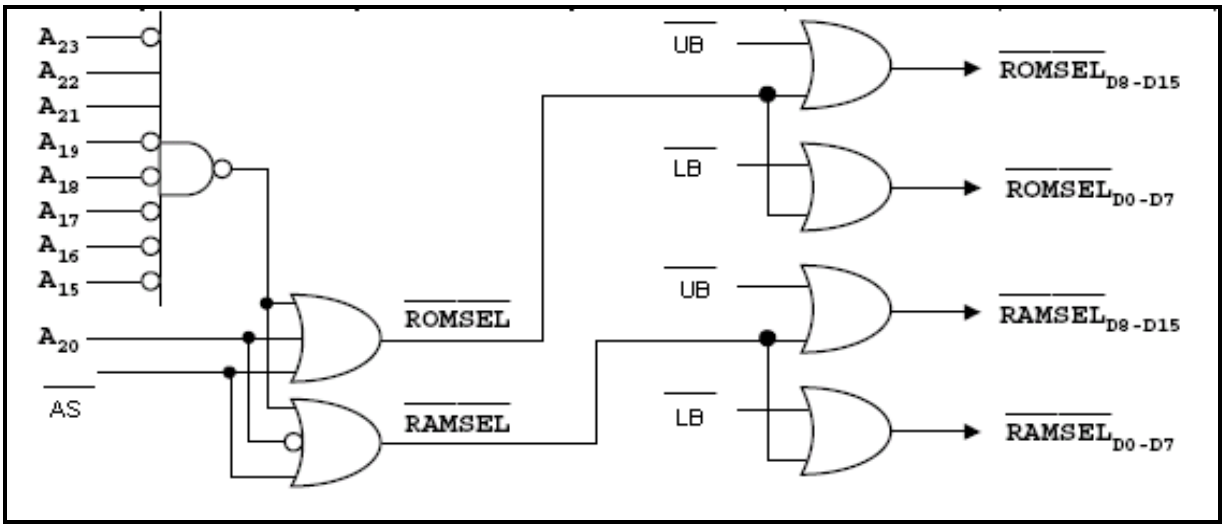
6				0				0 TO 7				0 TO F				0 TO F				0 TO F			
A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	1	1	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

And RAM will have memory space starting from `0x700000` to `0x707FFF`

7				0				0 TO 7				0 TO F				0 TO F				0 TO F			
A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	1	1	1	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

For the same, Decoder logic can be implemented as shown below

Figure 4. Decoder logic



Here, ROMSELX $_{Dx\ to\ DY}$ and RAMSELX $_{Dx\ to\ DY}$ should be connected to Chip select of respective E^{2}PROM and RAM chips.

4.4 Software Example an External Data Section Declaration

Assume the External Bus Interface is set like described in 3 in the *Start.asm* file, the following further definitions have to be done. Assume further, that an external FLASH is connected.

4.4.1 Linker Settings

For an external Flash area of 4 M Bytes beginning at address `0x400000`, the following linker settings have to be added:

```
-ra _EXT_FLASH=0x400000/0x7FFFFF
-sc EXT_FLASH/Data/WORD=_EXT_FLASH
```

This setting can also be added by the Softune menu Project → Setup Project → Linker → Disposition/Connection → Set and … → Set Section → Specify in Address.

The `-ra` option is to define the memory space and the `-sc` option is to define the section name. Please note, that `WORD` alignment should be used if the external Flash has a 16 bit wide bus.

4.4.2 External Memory Variable Declaration

Add the following code to your project to define variables, which should be linked to the external RAM memory.

```
   . . .
#pragma segment FAR_DATA=EXT_FLASH, attr=DATA

__far unsigned int ext_variable;

#pragma segment FAR_DATA

   . . .
```

`ext_variable` is now linked to `0x800000`, if no other previous modules define external FLASH variables. Outside the two `#pragma segment` directives all other variables are linked to the default data section `DATA`, if not other defined in the linker settings.

After project built the external area can be found in the **.mp1* file of the project:

```
   . . .

00400000-00400001  00000002  DATA  P RW-- 02 REL  EXT_FLASH

   . . .
```

4.5 Software Example to Write and Read data from flash

Assume the External Bus Interface and the section declaration are set like described in Section 3 and 4.3 respectively. Further assume that target board SK-96380-120PMT with external flash connected to MCU in multiplexed 16-bit mode is used.

On target board since Flash pin A0 is connected to latch output pin A1, all address output from MCU are shifted 1 bit right before sending on a Multiplexed address bus. In section 3.1.8 change default setting of `CS3_START` to `0x00`

Main.c

In main.c file there are two functions `Chiperase()` and `flashwrite()`. As the name suggest Chip erase function erases the chip and flashwrite function writes particular data at particular address location in external flash.

In main function, program first erases the external Flash chip. It writes random data in some memory location in external Flash and reads it back. If read data is different than written data then an error is indicated by outputting fixed bit pattern at PORT09

```
#define DQ7 0x0080
#define DQ5 0x0020

#define HEX    0
#define DEC    1

#define seq555 (__far unsigned int*)0x100AAA          // shifted x555
#define seq2AA (__far unsigned int*)0x100554          // shifted x2AA
#define start_adr (__far unsigned int*)0x100000


#include "mb96348rs.h"

#pragma segment FAR_DATA=EXT_FLASH, attr=DATA
__far char dummy;
#pragma segment FAR_DATA

unsigned char *byte_ptr1, *byte_ptr2;

// ==========================================
//  ATTENTION! CHIP ERASE TAKES ABOUT 1 MINUTE!
//   ==========================================

void chiperase(void)
{
    unsigned char flag = 0;

    *seq555 = 0x00AA;
    *seq2AA = 0x0055;
    *seq555 = 0x0080;
    *seq555 = 0x00AA;
    *seq2AA = 0x0055;
    *seq555 = 0x0010;

    while(flag == 0)
    {
        if((*start_adr & DQ7) == DQ7)     /* Toggle bit */
        {
            flag = 1;         /* successful erased */
        }

        else if((*start_adr & DQ5) == DQ5)       /* time out */
        {
            if((*start_adr & DQ7) == DQ7)
            {
                flag = 1;     /* successful erased */
            }
            else
            {
                flag = 2;     /* timeout error */
            }
        }
    }
}
```

```
void flashwrite(__far unsigned int *adr, unsigned int wdata)
{
        unsigned char flag = 0;
        *seq555 = 0x00AA;
        *seq2AA = 0x0055;
        *seq555 = 0x00A0;
        *adr = wdata;
        while(flag == 0)
        {
                if((*(__far unsigned int*)adr & DQ7) == DQ7) /* Toggle bit */
                {
                        flag = 1;               /* successful erased */
                }
                        else if((*(__far unsigned int*)adr & DQ5) == DQ5)    /* time out */
                {
                        if((*(__far unsigned int*)adr & DQ7) == DQ7)
                        {
                                flag = 1;       /* successful erased */
                        }
                        else
                        {
                                flag = 2;       /* timeout error */
                        }
                }
        }
}

void wait(unsigned int a)
{
        unsigned int i;
        for (i = 0; i < a; i++)
        {
                __wait_nop();
        }
}
```

```
void main(void)
{
        unsigned int rnd, err;
        unsigned long i;

        InitIrqLevels();
        __set_il(7);            /* allow all levels          */
        __EI();                 /* global enable interrupts */

        DDR09 = 0xFF;
        PDR09 = 0x00;
        chiperase();            // Takes about 1 Minute!

        rnd = 0x56B1;           // Random seed
        for (i = 0x100000; i < 0x800000; i += 2)
        {
                flashwrite((__far unsigned int*)i, rnd);
                rnd = ((rnd * 13) >> 2) ^ 0x8A27 + (0xFFFF & i);     // pseudo pseudo random
        }

        err = 0;
        rnd = 0x56B1;           // Random seed
        for (i = 0x100000; i < 0x800000; i += 2)
        {
                if ((*(__far unsigned int*)i) != rnd)
                err = 1;
                rnd = ((rnd * 13) >> 2) ^ 0x8A27 + (0xFFFF & i);     // pseudo pseudo random
        }

        if (err == 0)
        {
                PDR09 = 0xAA;
        }
        else
        {
                PDR09 = 0x01;
        }

        while(1);
}
```

5 External Vector Modes

Recommendations and Cautions When Using External Vector Modes

5.1 External Boot Vector

The External Bus Interface allows to use external ROM memory for reset vector fetch and code execution after power-on.

These external boot modes are announced to the MCU via the mode pins. There are 3 different external vector fetch modes:

MD2	MD1	MD0	External Vector Mode
0	0	0	Mode 0: 8-Bit mulitplexed
0	0	1	Mode 1: 16-Bit multiplexed
1	1	0	Mode 2: 8-Bit non-multiplexed

5.2 Software Recommendations and Cautions

The external bus modes can be selected in the *Start.asm* code.

```
;===================================================================
; 4.10  External Bus Interface
;===================================================================

#set      SINGLE_CHIP     0        ; all internal
#set      INTROM_EXTBUS   1        ; mask ROM or FLASH memory used
#set      EXTROM_EXTBUS   2        ; full external bus (INROM not used)

#set      BUSMODE EXTROM_EXTBUS    ; <<< set bus mode (see mode pins)

#set      MULTIPLEXED     0        ;
#set      NON_MULTIPLEXED 1        ; only if supported by the device

#set      ADDRESSMODE MULTIPLEXED  ; <<< set address-mode

; Some devices support multiplexed and/or non-multiplexed Bus mode
; please refer to the related datasheet/hardware manual
```

5.2.1 Mode Byte

To denote to the Boot ROM, which chip select should be used from power-on cycle on, the so-called Mode Byte is used, which is located just behind the reset vector in the external memory.

The Boot ROM code first fetches the reset vector from the external memory and then the Mode Byte. This Mode Byte has the identical value as the External Bus Mode Register (EBM) described in chapter 2.2.1. It sets the corresponding chip select pins, for the case that e. g. external RAM is used by the start code.

Caution!
Always enable Chip Select 5*, external bus mode, and multiplexed/non-multiplexed bus in the standard *Start.asm* code! Please also use *Start.asm* revision ≥ 1.59!

* for MB96380 series use chip select 4 when 8-Bit non-multiplexed mode is chosen.

The Mode Byte is defined in the *Start.asm* code as follows:

```
          .SECTION        RESVECT, CONST, LOCATE=H'FFFFDC
          .DATA.E _start
          .DATA.B ((ADDRESSMODE << 7) | ((BUSMODE >> 1) << 6) |
                  (CHIP_SELECT5 << 5) | (CHIP_SELECT4 << 4) |
                  (CHIP_SELECT3 << 3) | (CHIP_SELECT2 << 2) |
                  (CHIP_SELECT1 << 1) |  CHIP_SELECT0)
```

5.2.2 Chip Select Configuration Order in Start Code

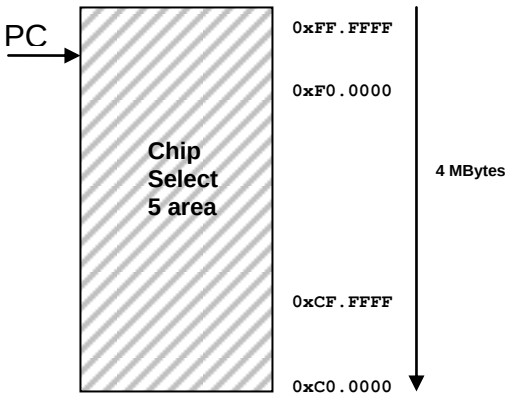
The *order* of setting the chip select start bank and the chip select size is very important for the chip select which is used for external code fetch.

Caution!
Always set the chip select start bank **before** the chip select size is changed to avoid malfunction of exteral code fetch!

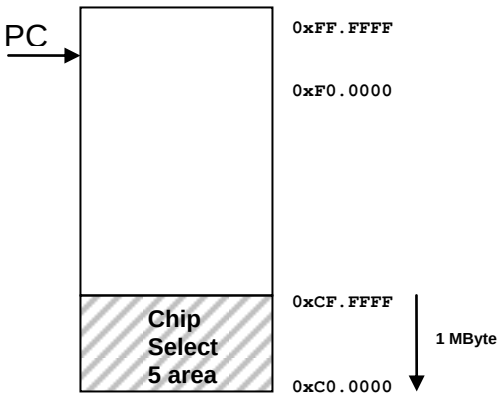
The Boot Code uses the initial values of the chip select 5 (4 Mbytes) and corresponding start bank (0xC0).

If the start code modifies the size for the chip select (because a smaller external ROM device is used) before setting the new start bank, it may happen, that the fetched code area is immediately outside of the chip select bank, and code execution fails.

Example for Size is changed before Start Bank (Bad Case):

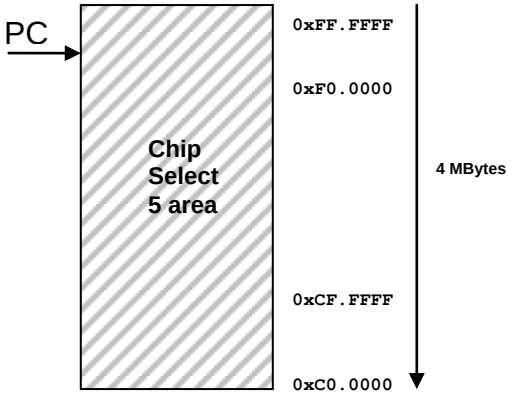


After Boot Code: Size is 4 MBytes default and Start Bank is 0xC0 default.

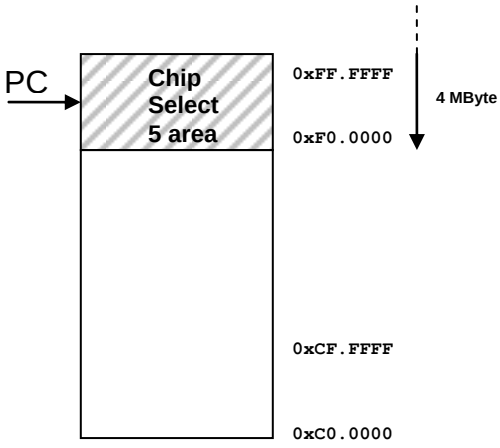


After changing size to 1 MByte: Start Bank is still 0xC0, but the current Program Counter is outside the Chip Select Area .

Example for Start Bank is changed before Size (Good Case):



After Boot Code: Size is 4 MBytes default and Start Bank is 0xC0 default.



After changing Start Bank to 0xF0, the current Program Counter is still inside the new Chip Select Area. Afterwards the size can be set down to 1 MByte.

Please check the configuration order in your used *Start.asm* code. It should be like:

```
. . .

            MOV  EBCF, #((HOLD_REQ << 7) | (EXT_READY << 6) |
                   (EXT_CLOCK_ENABLE << 5) | (EXT_CLOCK_INVERT << 4) |
                   (EXT_CLOCK_SUSPEND << 3) | EXT_CLOCK_DIVISION)
            MOV  EBAE0,#ADDR_PINS_7_0
            MOV  EBAE1,#ADDR_PINS_15_8
            MOV  EBAE2,#ADDR_PINS_23_16
            MOV  EBCS, #((ADDRESS_STROBE_LVL << 6) | (ADDRESS_STROBE
                   << 5) | (READ_STROBE << 4) | (HIGH_WRITE_STROBE << 3)
                   | (LOW_WRITE_STROBE << 2) | (HIGH_BYTE_SIGNAL << 1) |
                   LOW_BYTE_SIGNAL)
            MOV  EAS2, #CS2_START
            MOV  EAS3, #CS3_START
            MOV  EAS4, #CS4_START
            MOV  EAS5, #CS5_START
            MOVW EACL0,#CS0_CONFIG
            MOVW EACL1,#CS1_CONFIG
            MOVW EACL2,#CS2_CONFIG
            MOVW EACL3,#CS3_CONFIG
            MOVW EACL4,#CS4_CONFIG
            MOVW EACL5,#CS5_CONFIG
            MOV  EBM,  #((ADDRESSMODE << 7) | ((BUSMODE-1) << 6) |
                  (CHIP_SELECT5 << 5) | (CHIP_SELECT4 << 4) |
                  (CHIP_SELECT3 << 3) | (CHIP_SELECT2 << 2) |
                  (CHIP_SELECT1 << 1) | CHIP_SELECT0)

. . .
```

CSn_START must be set before CSn_CONFIG!

Start.asm revisions ≥ 1.59 are taking care of the correct order.

6 Timing Analysis

6.1 Flash Read AC characteristics

Figure 5 below shows timing diagram of Flash read cycle.

Figure 5. Flash Read Timing Diagram

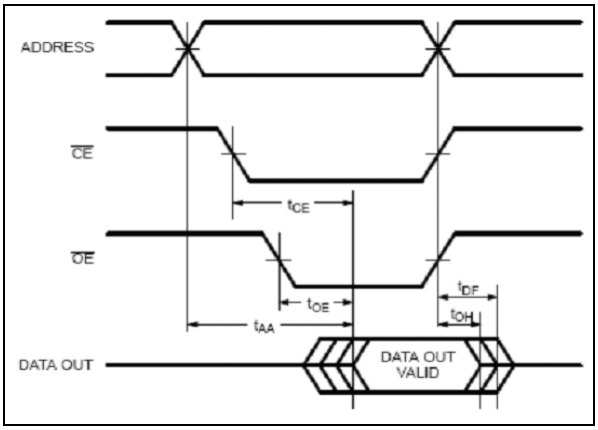


Table 1. Flash Read Characteristics

Parameter	Symbol	Value						Unit
		80		90		12		
		Min	Max	Min	Max	Min	Max	
Read Cycle Time	t_{RC}	80	-	90		120		ns
Address to Output Delay	t_{AA}	-	80	-	90	-	120	ns
Chip Enable to Output Delay	t_{CE}	-	80	-	90	-	120	ns
Output Enable to Output Delay	t_{OE}	-	30	-	35	-	50	ns
Chip Enable to Output High-Z	t_{DF}	-	25	-	30	-	30	ns
Output Enable to Output High-Z	t_{DF}	-	25	-	30	-	30	ns
Output Hold Time From Addresses, CEX or OEX, whichever Occurs First	t_{OH}	0	-	0	-	0	-	ns

Here, t_{AA}, defines the maximum time after the address stabilizes that the Flash will return valid data. This parameter is commonly referred to as the ‘access time’ of the device.

Similarly, t_{CE} defines the maximum time after the CEX input is asserted that the Flash will return data. This spec is typically, though not always, the same as t_{AA}. t_{OE} defines the maximum time from OEX assertion to valid data output much like t_{CE}. t_{OH} indicates the minimum time the data is guaranteed to remain valid after CEX/OEX de assertion. t_{DF}, defines the maximum time after which the output is guaranteed to completely float

6.2 RAM Read and Write AC characteristics

Figure 6. RAM Read Timing Diagram

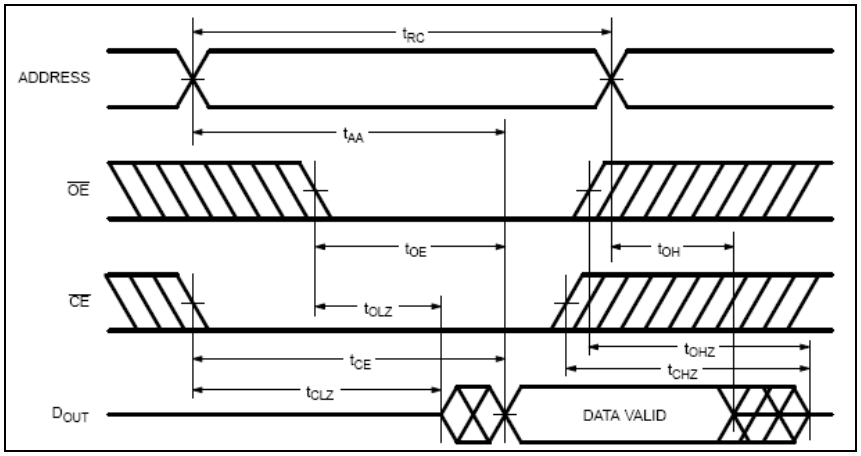


Table 2. RAM Read Characteristics

Parameter	Symbol	Value				Unit
		70		55		
		Min	Max	Min	Max	
Read Cycle Time	t_{RC}	70	-	55	-	ns
Address Access Time	t_{ACC}	-	70	-	55	ns
Chip Enable Access Time	t_{CE}	-	70	-	55	ns
Output Enable Access Time	t_{OE}	-	35	-	30	ns
Chip Enable Low to Output Active	t_{CLZ}	5	-	5	-	ns
Output Enable Low to Output Active	t_{OLZ}	0	-	0	-	ns
Chip Enable High to Output High-Z	t_{CHZ}	-	30	-	25	ns
Output Enable High to Output High-Z	t_{OHZ}	-	30	-	25	ns
Output Data Hold Time	t_{OH}	10	-	10	-	ns

Figure 7 shows the read cycle timing for a RAM which is quite similar to that of a Flash.

Here separate CEX and OEX data float specs (t_{CHZ}, t_{OHZ}) are defined instead of a single t_{DF}. Since the SRAM (unlike the EPROM/Flash) can be written. It also specifies the other side of the data float (i.e., enable to output driven) with t_{CLZ} and t_{OLZ}.

Figure 7. RAM Write Timing Diagram

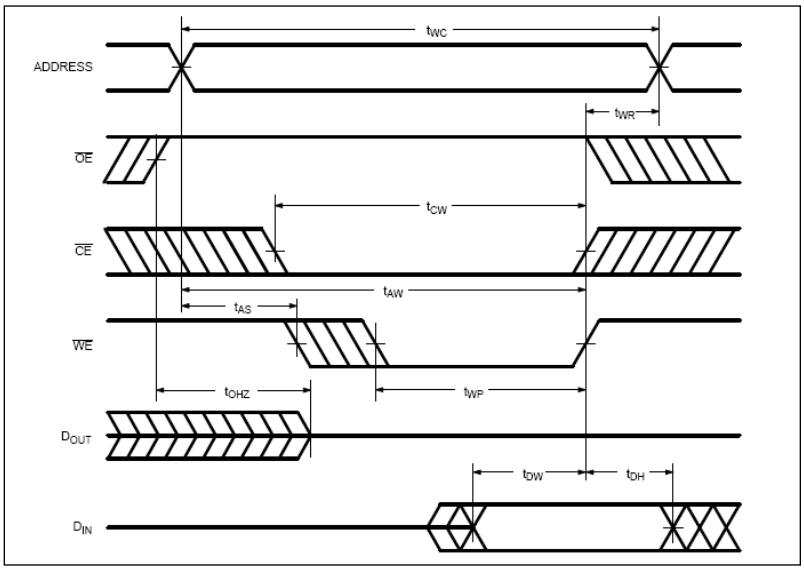


Table 3. RAM Write Characteristics

Parameter	Symbol	Value				Unit
		70		55		
		Min	Max	Min	Max	
Write Cycle Time	t_{WC}	70	-	55	-	ns
Write Pulse Width	t_{WP}	50	-	40	-	ns
Chip Enable to End of Write	t_{CW}	55	-	45	-	ns
Address Setup Time	t_{AS}	0	-	0	-	ns
Write Recovery Time	t_{WR}	0	-	0	-	ns
Output Enable High to Output High-Z	t_{OHZ}	-	30	-	25	ns
Data Setup Time	t_{DS}	30	-	25	-	ns
Data Hold Time	t_{DH}	0	-	0	-	ns

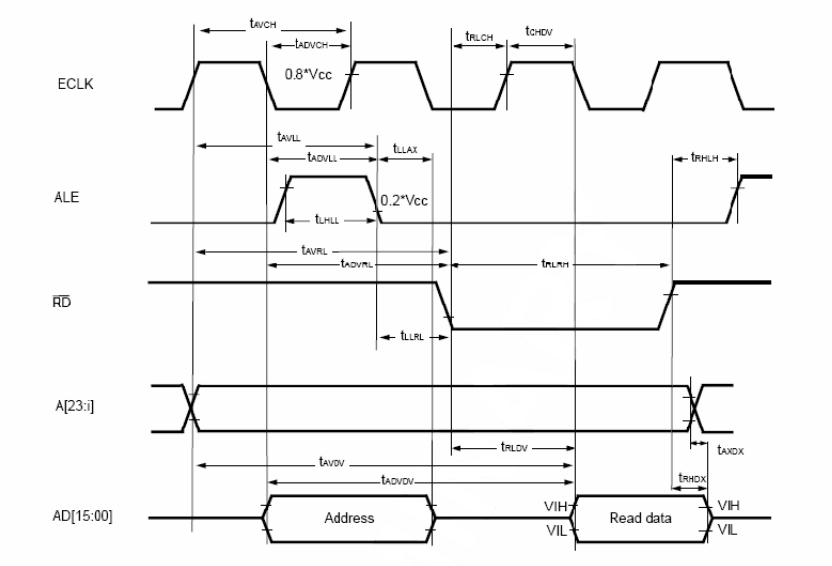
Figure 8 shows the write cycle timing.

Write time (t_{WP}) is defined as the time during which both CEX and WE are asserted.

t_{WC} simply defines the write cycle time which, along with t_{RC}, is the same as the 'access time'. t_{CW} and t_{AW} specify the minimum time from valid CEX and address inputs to the end of the write cycle. t_{AS} defines an address setup to the beginning of the write cycle. t_{WP} simply specifies the minimum write pulse (the overlap of CEX and WE) width. t_{WR} specifies a minimum write 'recovery' time, essentially an address hold time after the end of write. t_{DW} and t_{DH} specify the input data setup and hold times relative to the end of write. t_{OHZ} defines time duration that must be elapsed before a new write cycle begin to allow previous read data to disappear and to avoid bus contention.

6.3 MCU Read AC characteristics

Figure 8. External Memory Read Cycle



6.4 MCU Write AC characteristics

Figure 9. External Memory Write Cycle

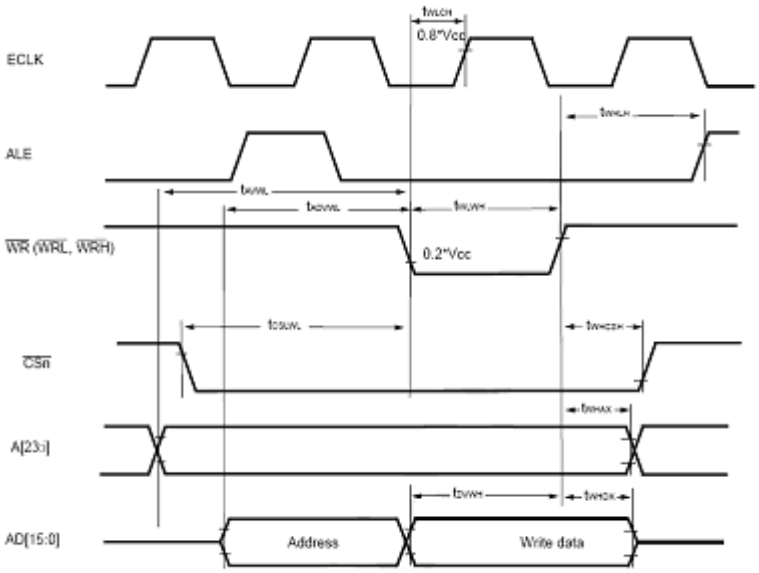


Table 4. MCU Characteristics

Parameter	Symbol	Condition	Value		Unit
			Min	Max	
Valid address to valid data input	t_{ADVDV}	EBM:NMS=0 & EACL:ACE=0	-	$3t_{CYC}$ -55	ns
		EBM:NMS=0 & EACL:ACE=1	-	$4t_{CYC}$ -55	
		EBM:NMS= 1	-	$2t_{CYC}$ -55	
		EACL:ACE=0	-	$5t_{CYC}$ /2-55	
		EACL:ACE=1		$7t_{CYC}$ /2-55	
Valid address to ALE low	t_{ADVLL}	EACL:STS=0 & EACL:ACE=0	t_{CYC}/2 - 15	-	ns
		EACL:STS=1 & EACL:ACE=0	t_{CYC} - 15	-	
		EACL:STS=0 & EACL:ACE=1	$3t_{CYC}$/2 -15	-	
		EACL:STS=1 & EACL:ACE=1	$2t_{CYC}$ -15	-	
ALE low to address valid	t_{LLAX}	EACL:STS=0	t_{CYC}/2 -15		
		EACL:STS=1			
RDX low to valid data input	t_{RLDV}		-	$3t_{CYC}$ /2-50	ns
RDX high to Data hold time	t_{RHDX}		0		ns
CS low to valid Data input	t_{CLDV}				ns
RDX high to input data float	t_{RHDX}		0	-	ns
RDX low to address invalid	t_{RLAX}				ns
Address valid to write low	t_{AVWL}	EBM:NMS=0 & EACL:ACE=0	$3t_{CYC}$ /2-15	-	ns
		EBM:NMS=0 &EACL:ACE=1	$5t_{CYC}$ /2-15	-	ns
		EBM:NMS=1 & EACL:STS=0	t_{CYC} /2-15	-	ns
		EBM:NMS=1 &EACL: STS=1	t_{CYC}-15	-	ns
		EACL:ACE=0	t_{CYC}-15	-	ns
		EACL:ACE=1	$2t_{CYC}$ -15	-	ns
Write low to write high	t_{WLWH}		t_{CYC}-5		ns

Parameter	Symbol	Condition	Value		Unit
			Min	Max	
WR high to ALE high	t_{WHLH}	EBM:ACE=1 & EACL:STS=1	$2t_{CYC}$-10		ns
		other EBM:ACE & EACL:STS setting	t_{CYC}-10		
Data valid to WR transition	t_{DVWH}		t_{CYC}-20		ns
Write High to Data Invalid	t_{WHDX}		t_{CYC}/2-15		ns

Detail spec can be found in hardware manual

6.5 Timing Analysis

Let's start with the 'classic' circuit shown in Figure 2 that uses a transparent latch to demultiplex the address/data bus AD0–15. The latch, in Figure 2, is controlled with two pins – LE (Enable) and OEX (Output Enable).. This is exactly the behaviour called for to demultiplex the MCU AD0–15 bus with the ALE output from the MCU. Usually, OEX is simply connected to ground enabling the output at all times.

For calculation let us assume MCU is running at 16 MHz. For 16MHz t_{CYC} is 62.5ns.Further EACL:STS is set to 0 (Strobe scheme 0) , EBM:NMS is set to 0 (Multiplexed AD) and EACL:ACE is set to 0 (Address cycle is not extended)

For the latch CD74AC573, following table indicates its AC characteristics.

Table 5. Latch Characteristics

Characteristics	Symbol	Value		Unit
		Min	Max	
LE pulse width	t_W	4.9	-	ns
Setup time data to LEX	t_{SU}	2	-	ns
Hold time data to LEX	t_H	3.7	-	ns
Propagation Delay	t_{PROP}	3.1	10.8	ns

The first step is to confirm the MCU meets the setup and hold times for the chosen latch.

$t_S < t_{ADVLL}$ (address valid to ALE low) = tCYC/2 - 15 = 16.25ns
$t_H < t_{LLAX}$ (address hold after ALE low) = tCYC/2 -15 = 16.25ns

6.5.1 Flash Timing Analysis

The procedure is simply to step through each Flash spec one by one to identify a speed grade that meets all the relevant MCU timing requirements.

Starting with t_{AA}, it is apparent that address access time for the Flash must be less than the MCU t_{ADVDV} (address to valid data in)

t_{AA} (Flash) < t_{ADVDV} (MCU) – t_{PROP} (TTL)
t_{AA} (Flash) < (3tCYC -55) – 10.8 = 121.7ns

According to the Flash spec chart (refer back to Table 1), this can be met by either '–80' (80ns) or '–90' Flash.

$t_{OE,}$ should be less than t_{RLDV} (RDX low to valid data in)

t_{OE}(Flash) < t_{RLDV} (MCU)
t_{OE}(Flash) < (3tCYC /2 – 50) = 43.75ns

According to the Flash spec chart (refer back to Table 1), this can be met by either '–80' (t_{OE} is 30ns Max) or '–90' (t_{OE} is 35ns Max) Flash.

The Flash t_{OH} spec is 0ns. On the MCU side, the corresponding spec is t_{RHDX} (RDX high to Data hold time) which is also 0ns. This spec is also met as per the requirement because MCU will see RDX going high before the Flash and also in fact Flash will take some time to clear its output.

Flash spec t_{DF} specifies how long data will be available on bus after OEX signal is de-asserted. Flash should stop driving the data output before MCU put address on multiplexed address and data line to avoid bus contention.

For that, t_{DF} should be less than t_{RHLH}

t_{DF}(Flash) < t_{RHLH}
t_{DF}(Flash) < (tCYC /2 – 10) = 21.25ns

Unfortunately, '-70' flash with t_{DF} max 25ns can not meet this spec. We need to use faster flash. However we can use '-70' flash ignoring this problem because this situation will arise quite rarely. In real life application this problem will be taken care by the processing time required by application in between two transfers.

6.5.2 RAM Timing Analysis

The process of evaluating SRAM interface is similar to that for the Flash

For a data read, the SRAM t_{AA} is compared with the MCU t_{ADVDV} (address to valid data in).

t_{AA}(RAM) < t_{ADVDV}(MCU)–t_{PROP}(TTL)
t_{AA} (RAM) < (3tCYC -55) – 10.8 = 121.7ns

This meets specs of either '–70' or '–55' RAM.

$t_{OE,}$ should be less than t_{RLDV} (RDX low to valid data in)

t_{OE} (RAM) < t_{RLDV} (MCU)
t_{OE}(RAM) < (3tCYC /2 – 50) = 43.75ns.

This meets spec of either '–70' (t_{OE} is 35ns Max) or '–55' (t_{OE} is 30ns Max) RAM.

Flash spec t_{DF} specifies how long data will be available on bus after OEX signal is de-asserted. t_{DF} should be sufficient enough so that RAM should stop driving the data output before MCU put address on multiplexed address and data line to avoid bus contention.

t_{OHZ}(RAM) < t_{RHLH}
t_{OHZ} (RAM) < (tCYC /2 – 10) = 21.25ns

Unfortunately, '-55' RAM with t_{OHZ} max 25ns can not meet this spec. We need to use faster RAM. However we can use '-55' RAM ignoring this problem because this situation will arise quite rarely. In real life application this problem will be taken care by the processing time required by application in between two transfers.

On the other side, it is possible to for RAM to drive data on the multiplex bus before the address is removed which may cause bus contention. The SRAM will drive the bus with in 0ns (t_{OLZ}) of OEX assertion. It is taken care by MCU spec (t_{LLRL} - t_{LLAX}) which guarantees that address is off the bus when RDX is asserted.

For a data write, the SRAM t_{AW} (address valid to end of write) is compared against the sum of the MCU t_{AVWL} (address valid to write low) and t_{WLWH} (write low to write high) specs.

Again considering the latch propagation delay..

$t_{LLRL} - t_{LLAX} = (tCYC/2 - 15) - (tCYC/2 - 20) = 5ns$

This is met by RAM spec.

$t_{AW}(RAM) < t_{AVWL}(MCU) + t_{WLWH}(MCU) – t_{PROP}(TTL)$
$t_{AW}(RAM) < 3tCYC /2-15 + tCYC-5 – 10.8 = 187.95ns$

The SRAM t_{AS} spec defines the time addresses must be setup prior to the assertion of WE which is connected to the MCU WR line so...

$t_{AS}(RAM) < t_{AVWL}(MCU) – t_{PROP}(TTL)$
$t_{AS}(RAM) < 3tCYC /2-15 – 10.8 = 68.75ns$

This meets RAM spec of t_{AS} = 0ns.

The SRAM write pulse is defined as the overlap of CEX and WE, t_{WP} (write pulse width) is simply defined by t_{WLWH}

$t_{WP}(RAM) < t_{WLWH}(MCU)$
$t_{WP}(RAM) < tCYC-5 = 57.5ns$

This can be met by either '–70' (t_{WP} is 50ns Max) or '–55' (t_{WP} is 40ns Max) RAM.

The SRAM t_{WR} (write recovery) spec defines how long the addresses must be held after the end of write, which is the end of MCU WR in this design. Since addresses are guaranteed to remain stable while ALE is low, this becomes t_{WHLH} (RDX or WR high to ALE high)...

$t_{WR}(RAM) < t_{WHLH}(MCU)$
$t_{WR}(RAM) < tCYC-10 = 52.5ns$

This meets RAM spec of t_{WR} = 0ns.

As for t_{DS} (data setup to end of write), the corresponding MCU timing is derived by summing t_{DVWH} (data valid to WR transition) and t_{WLWH} (WR pulse width) so...

$t_{DS}(RAM) < t_{DVWH}(MCU) + t_{WLWH}(MCU)$
$t_{DS}(RAM) < tCYC-20 + tCYC-5 = 100ns$

This can be met by either '–70' (t_{DS} is 30ns Max) or '–55' (t_{DS} is 25ns Max) RAM.

The SRAM hold spec t_{DH} is simply compared with t_{WHDX}

$t_{DH}(RAM) < t_{WHDX}(MCU)$
$t_{DH}(RAM) < tCYC /2-15 = 16.25ns$

This meets RAM spec of t_{DH} = 0ns.

7 Additional Information

Information about Cypress can be found on the following Internet page:

www.cypress.com/cypress-microcontrollers

The software examples related to this application note is:

96380_ext_bus

It can be found on the following Internet page:

www.cypress.com/cypress-mcu-product-softwareexamples

Document History

Document Title: AN205546 - F²MC-16FX Family, External Bus Interface

Document Number: 002-05546

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	07/13/2006	Initial Release
			06/22/2007	Updated software example and timing analysis
			08/10/2007	Updated Timing analysis chapter
			09/28/2007	Updated example in section 4
			10/04/2007	Updated example in section 4
			07/08/2008	Correct start address of Chip Select area 1 to 0x000C00; add more information on chip select area size and alignment
			11/12/2009	Typos corrected and Chapter External Vector Mode added
*A	5071406	MKEA	04/19/2016	Migrated Spansion Application Note from MCU-AN-300208-E-V16 to Cypress format
*B	5865450	AESATP12	08/30/2017	Updated logo and copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709