Freescale Semiconductor
Application Note

Document Number: AN4153
Rev. 0, 08/2010

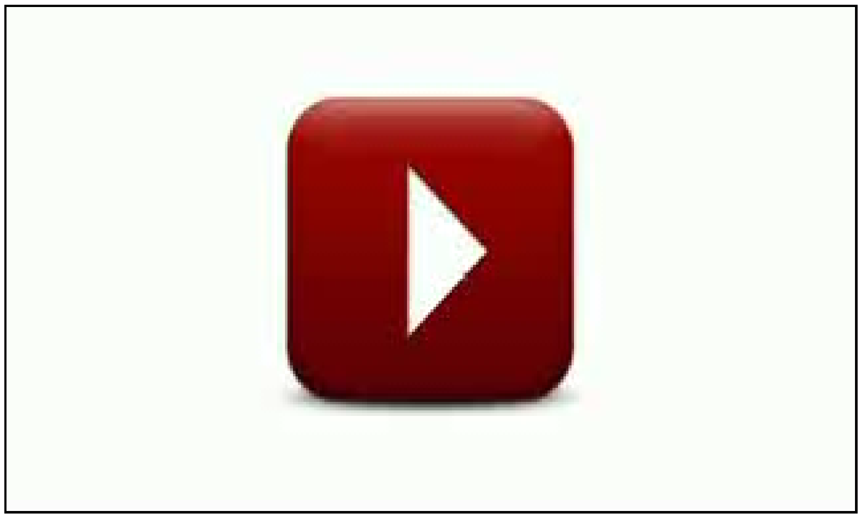
Using Freescale eGUI with TWR-LCD on MCF51MM Family

by: Yingyi Ju
32-bit Application Team, Microcontroller Solutions Group
Shanghai
China

1 Introduction

This application note describes how to connect a TFT-LCD that has a controller module to a Mini-Flexbus port or serial port (SPI) on a Freescale ColdFire® series MCU/MPU. It also describes how to design a low-level TFT-LCD driver for mini-FlexBus or SPI connections. Some of the material in this document was adapted from material created by Solomon Systech, manufacturer of the SSD1289.

Refer to the latest silicon and board documentation for updates to the information in this document. This document was written using the most recent documentation available at freescale.com and at www.solomon-systech.com.



Contents

1 Introduction..........1
2 Interface of TFT-LCD Controller..........2
2.1 16-bit data mode for Mini-FlexBus..........7
2.2 8-bit data mode for Mini-FlexBus..........8
2.3 SPI data mode..........9
3 TWR-LCDSTY board preparation..........9
4 Microcontroller configuration..........10
4.1 Mini-FlexBus interface..........10
4.2 SPI interface..........11
5 Low-level driver design..........11
5.1 16-bit Mini-FlexBus data mode driver..........12
5.2 SPI data mode driver..........13
5.3 8-bit data bus mode interface..........14
5.4 Integrate low-level drivers with eGUI..........15
6 Summary..........16

2 Interface of TFT-LCD Controller

The TFT-LCD module on the TWR-LCDSTY board is integrated with the Solomon Systech TFT-LCD controller driver SSD1289. This TFT-LCD controller driver integrates the RAM, power circuits, gate driver, and source driver into a single chip. Graphic display data RAM is interfaced with the common MCU/MPU through an 8-bit/9-bit/16-bit/18-bit 6800-series / 8080-series compatible parallel interface or serial peripheral interface.

For the correct interface selection, use the table below to configure the TFT-LCD driver module:

Table 1. Interface selection

PS3	PS2	PS1	PS0	Interface Mode
1	1	1	1	3-wire SPI
1	1	1	0	4-wire SPI
1	0	1	1	16-bit 6800 parallel interface
1	0	1	0	8-bit 6800 parallel interface
1	0	0	1	16-bit 8080 parallel interface
1	0	0	0	8-bit 8080 parallel interface
0	1	1	1	18-bit 6800 parallel interface
0	1	1	0	9-bit 6800 parallel interface
0	1	0	1	18-bit 8080 parallel interface
0	1	0	0	9-bit 8080 parallel interface
0	0	1	1	Reserved
0	0	1	0	Reserved
0	0	0	1	18-bit RGB interface + 4-wire SPI

Parallel Mode:

- In 6800/8080 8-bit parallel mode: D1 ~ D8, E(/RD), R/W(/WR), and D/C are used as I/O control.
- In 6800/8080 9-bit parallel mode: D0 ~ D8, E(/RD), R/W(/WR), and D/C are used as I/O control.
- In 6800/8080 16-bit parallel mode: D1 ~ D8 and D10 ~ D17, E(/RD), R/W(/WR), and D/C are used as I/O control.
- In 6800/8080 18-bit parallel mode: D0 ~ D17, E(/RD), R/W(/WR), and D/C are used as I/O control.
- RGB VSYNC, HSYNC, DEN and DOTCLK should be connected to VDDIO or VSS.

Serial Mode:

- D0 ~ D17, E(/RD) and R/W(/WR) should be connected to VDDIO or VSS. Please refer to device data sheet for different driver connections. RGB VSYNC, HSYNC, DEN and DOTCLK should be connected to VDDIO or VSS.
- In 3-wire serial mode: SDI, SCK and /CS are used as I/O control.
- In 4-wire serial mode: SDI, SCK, /CS and D/C are used as I/O control.

RGB Mode:

- D[5:0] are used as B[0:5]; D[11:6] are used as G[5:0]; and D[17:12] are used as B[5:0].
- VSYNC, HSYNC, DEN, and DOTCLK are used for signal control.
- In 18-bit RGB mode: D0 ~D17 with signal control pins are used.
- If using 16-bit color mode, it is better to instead use 18-bit RGB mode and connect RR0 and BB0 to RR5 and BB5.

NOTE

All unused pin(s) should be connected to VDDIO or VSS.

Table 2. Mapping for command

		Hardware pins																	
Interface	Cycle	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
18-bit	N/A	IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	x	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0	x
16-bit	N/A	IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	x	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0	x
9-bit	1st	▨	▨	▨	▨	▨	▨	▨	▨	▨	IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	x
	2nd	▨	▨	▨	▨	▨	▨	▨	▨	▨	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0	x
8-bit	1st	▨	▨	▨	▨	▨	▨	▨	▨	▨	IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	▨
	2nd	▨	▨	▨	▨	▨	▨	▨	▨	▨	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0	▨

NOTE

x = don't care bit; shaded table cell = pin not connected.

Table 3. Mapping for pixel data

			Hardware pins																	
Interface	Color mode	Cycle	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
18-bit	262K	N/A	R5	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B5	B4	B3	B2	B1	B0
16-bit	262K	1st	R5	R4	R3	R2	R1	R0	x	x	▨	G5	G4	G3	G2	G1	G0	x	x	▨
		2nd	B5	B4	B3	B2	B1	B0	x	x	▨	R5	R4	R3	R2	R1	R0	x	x	▨
		3rd	G5	G4	G3	G2	G1	G0	x	x	▨	B5	B4	B3	B2	B1	B0	x	x	▨
		1st	R5	R4	R3	R2	R1	R0	x	x	▨	G5	G4	G3	G2	G1	G0	x	x	▨
		2nd	x	x	x	x	x	x	x	x	▨	B5	B4	B3	B2	B1	B0	x	x	▨
		1st	R5	R4	R3	R2	R1	R0	x	x	▨	G5	G4	G3	G2	G1	G0	x	x	▨
		2nd	B5	B4	B3	B2	B1	B0	x	x	▨	x	x	x	x	x	x	x	x	▨
	65K	N/A	R4	R3	R2	R1	R0	G5	G4	G3	▨	G2	G1	G0	B4	B3	B2	B1	B0	▨
9-bit	262K	1st	▨	▨	▨	▨	▨	▨	▨	▨	▨	R5	R4	R3	R2	R1	R0	G5	G4	G3
		2nd	▨	▨	▨	▨	▨	▨	▨	▨	▨	G2	G1	G0	B5	B4	B3	B2	B1	B0

			Hardware pins																	
In-ter-face	Col-or mode	Cycle	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
8-bit	262K	1st	▨	▨	▨	▨	▨	▨	▨	▨	▨	R5	R4	R3	R2	R1	R0	x	x	▨
		2nd	▨	▨	▨	▨	▨	▨	▨	▨	▨	G5	G4	G3	G2	G1	G0	x	x	▨
		3rd	▨	▨	▨	▨	▨	▨	▨	▨	▨	B5	B4	B3	B2	B1	B0	x	x	▨
	65K	1st	▨	▨	▨	▨	▨	▨	▨	▨	▨	R4	R3	R2	R1	R0	G5	G4	G3	▨
		2nd	▨	▨	▨	▨	▨	▨	▨	▨	▨	G2	G1	G0	B4	B3	B2	B1	B0	▨

NOTE

x = don't care bit; shaded table cell = pin not connected.

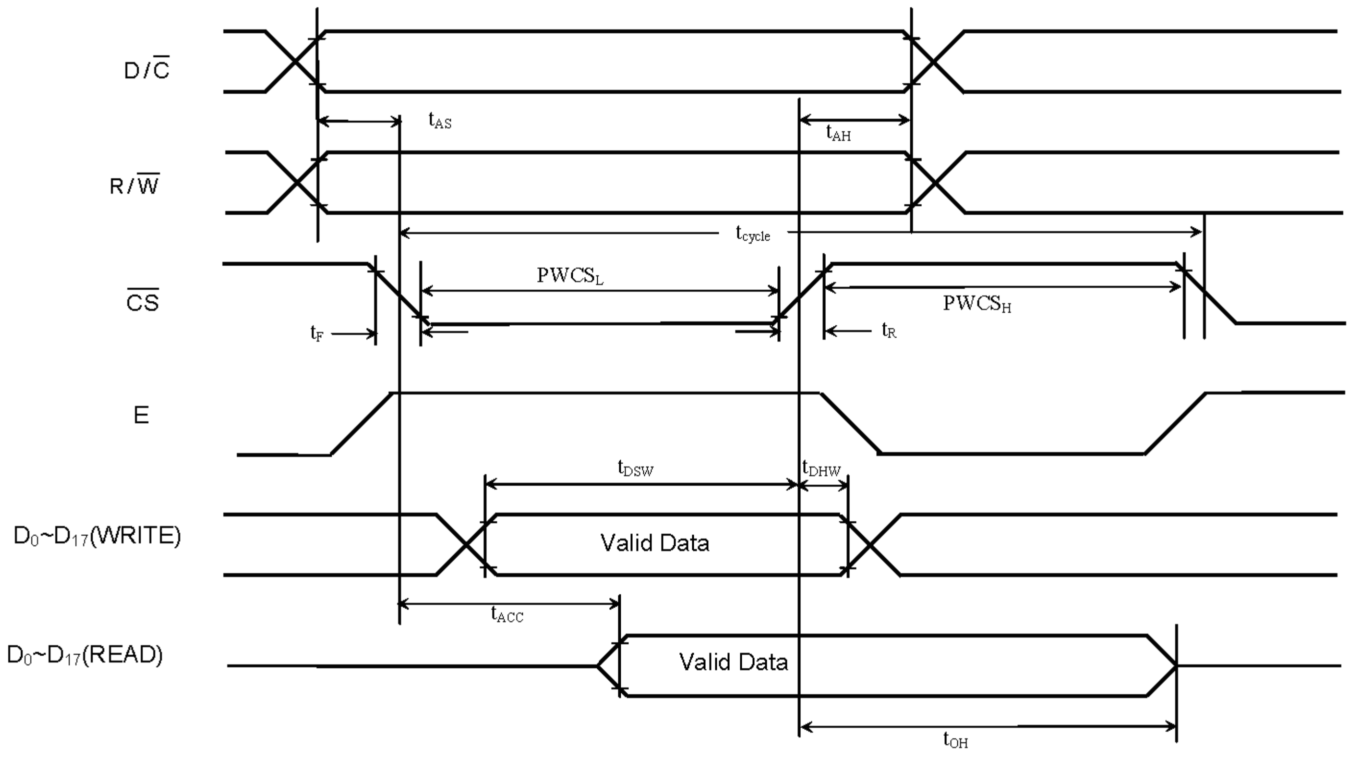


Figure 1. Parallel 6800-series interface read and write cycle timing

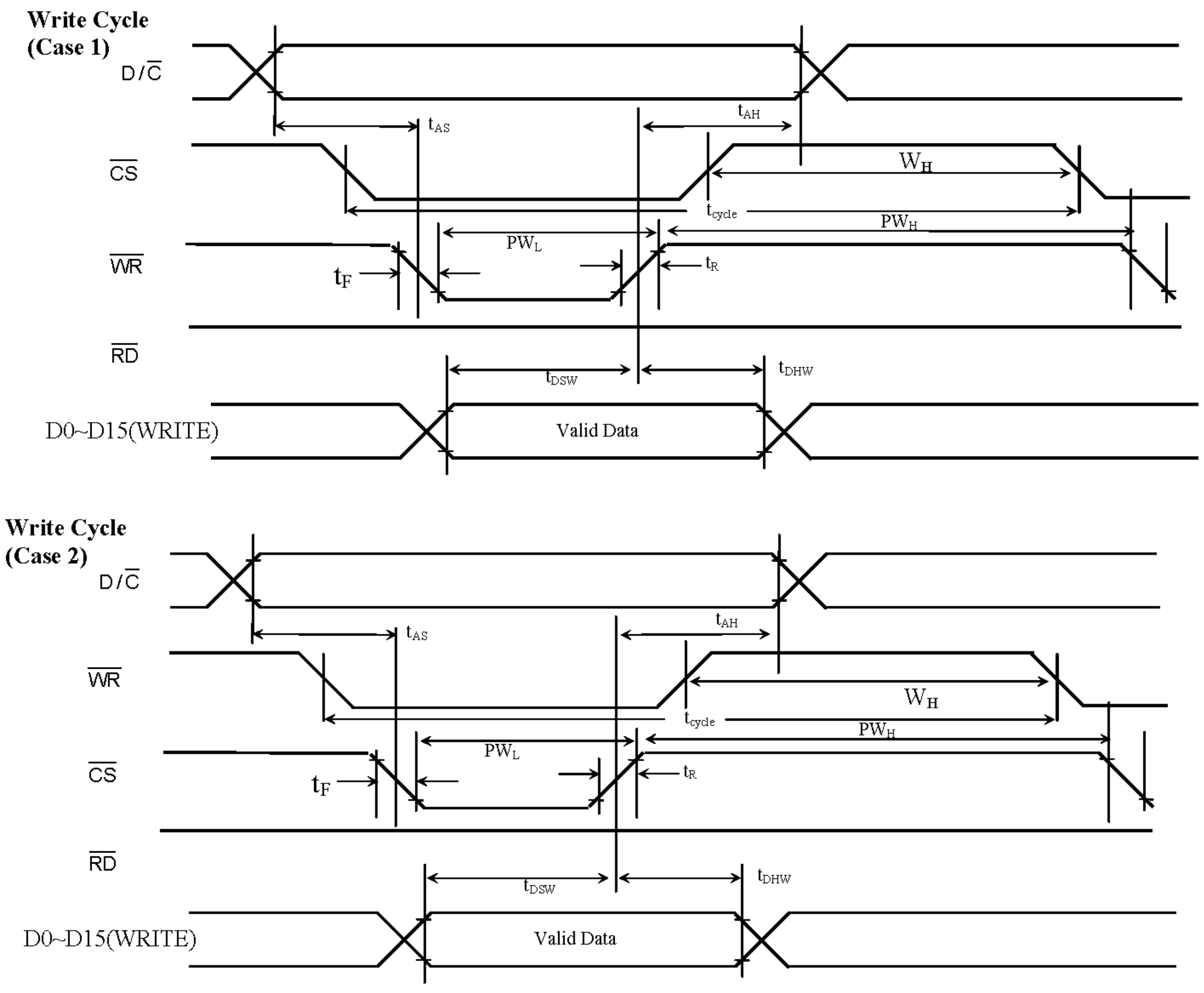


Figure 2. Parallel 8080-series interface read cycle timing

Figure 2 shows the parallel 8080-series interface read cycle timing. In case 1, the /CS signal is used to latch the data; in case 2, the /RD signal is used to latch the data.

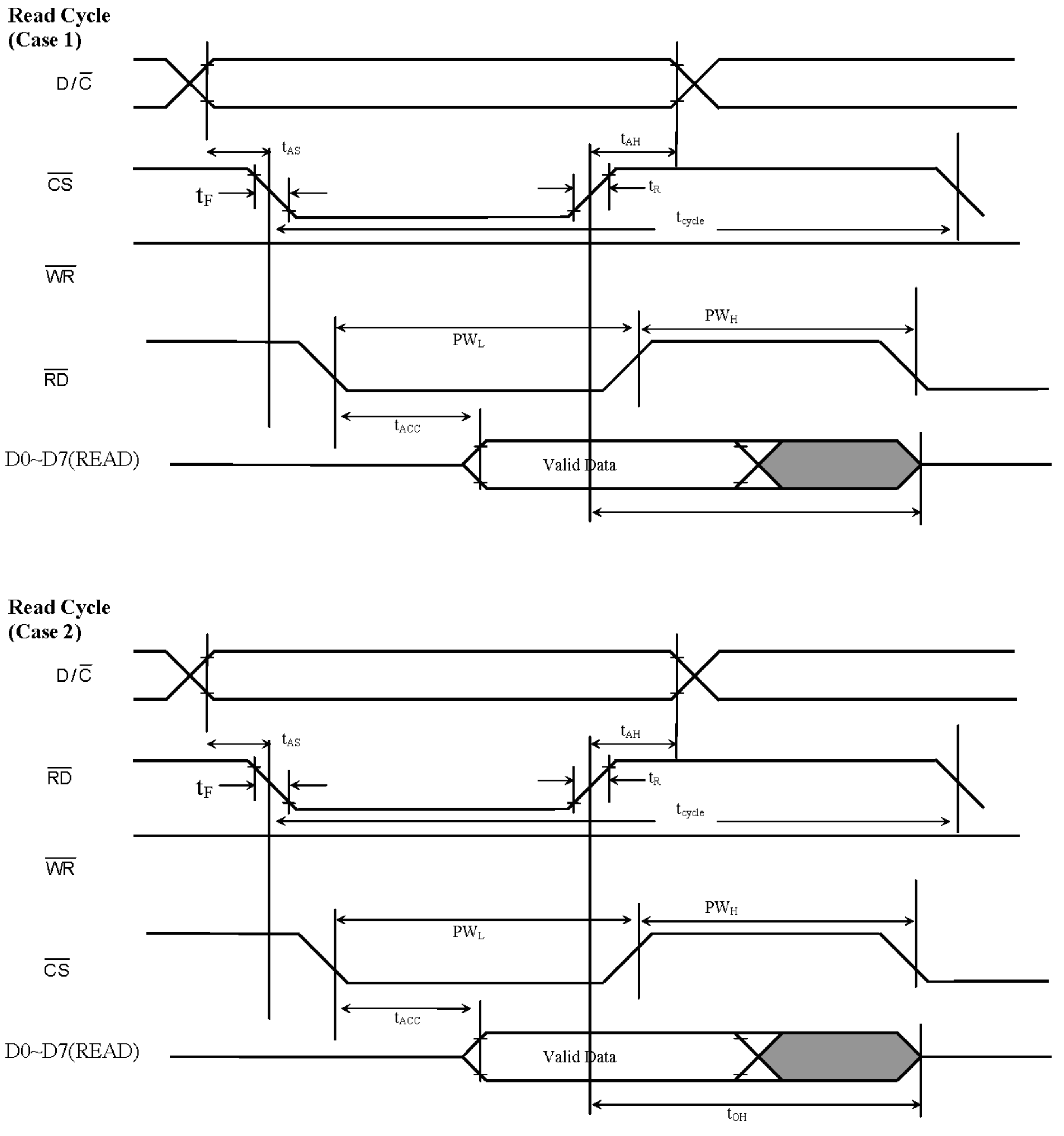


Figure 3. Parallel 8080-series interface write cycle timing

Figure 3 shows the parallel 8080-series interface write cycle timing. In case 1, /CS signal is used to latch the data while in case 2 /WR signal is used to latch the data.

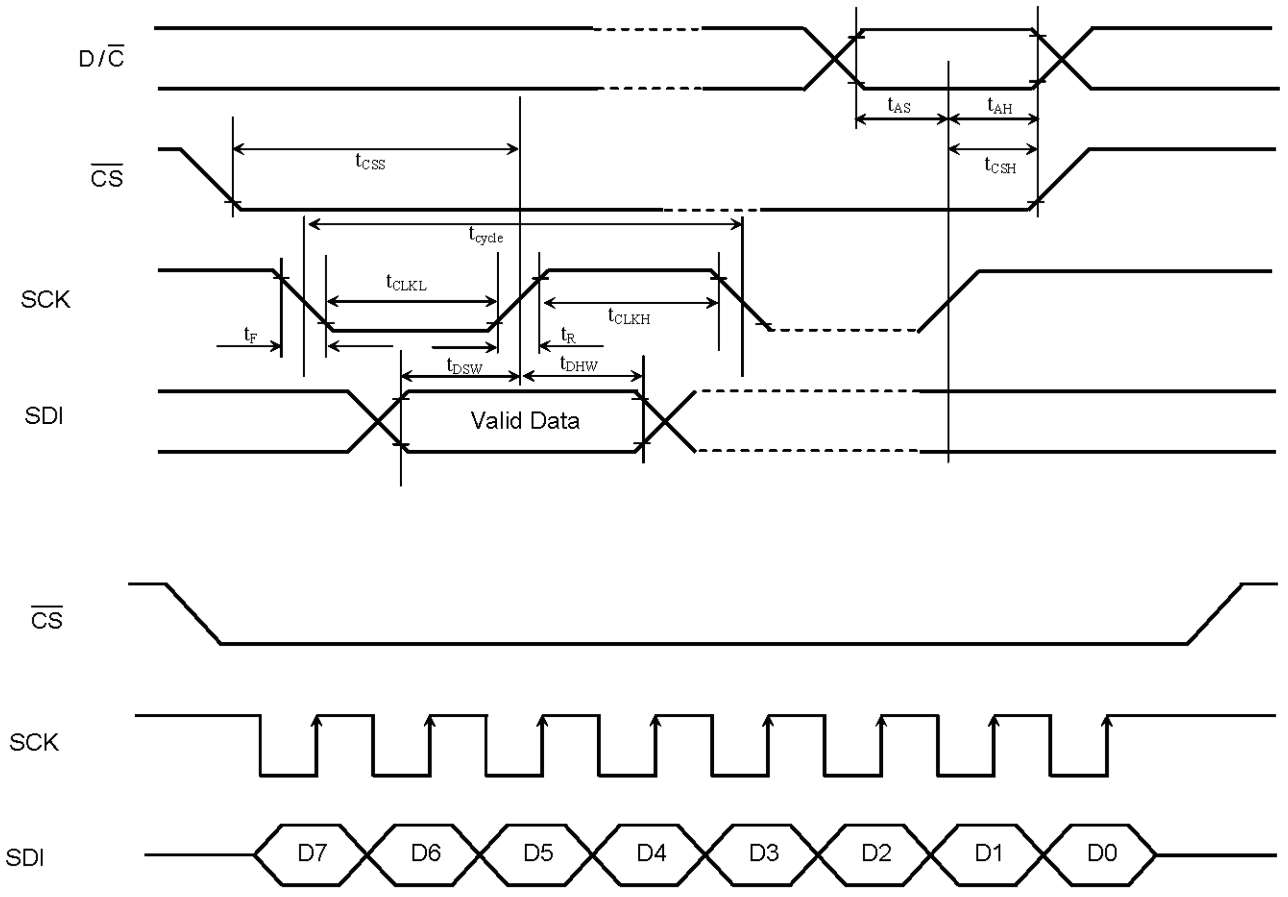


Figure 4. 4-wire serial timing

2.1 16-bit data mode for Mini-FlexBus

Configure PS[3:0] on SSD1289 to 0b1011 for the 16-bit 6800 parallel interface or to 0b1001 for the 16-bit 8080 parallel interface. On TWR-LCDSTY, 16-bit bus interface to Coldfire MCU/MPU:

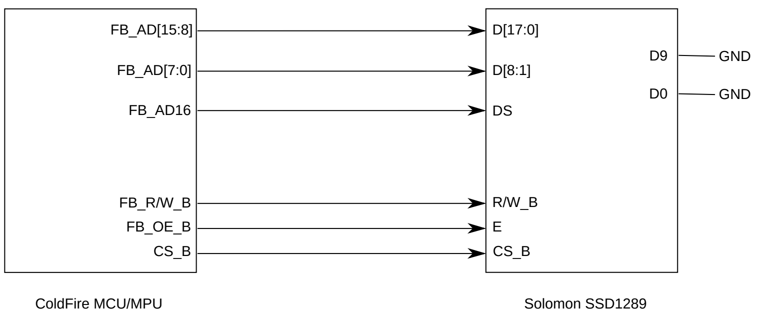


Figure 5. 16-bit bus interface to ColdFire MCU/MPU

If you do not want to read the pixel data, the E(RD_B) pin of the LCD controller can be pulled up directly.

When using a ColdFire V1 MCU with Mini-FlexBus, it should be configured to multiplexed mode to support a 16-bit data bus. To avoid conflict, any pin of FB_AD[19:16] can be used as a DS signal.

In Figure 5, the DS pin of SSD1289 is connected to FB_AD16 of the Mini-FlexBus port. This means that when there is a read/write to or from an address with bit 16 driven to low, the index register of SSD1289 is being accessed; but if address bit 16 is driven high, then another register or display buffer is being accessed.

For example, assume that CS0_B of the MCU is connected to CS_B of the SSD1289, and that CSAR0 on the MCU side has been set to 0x400000. In this case reading/writing address 0x400000 is used to access the index register of the SSD1289, while reading/writing address 0x410000 is used to access another register or data buffer of the SSD1289.

2.2 8-bit data mode for Mini-FlexBus

Configure PS[3:0] to 0b1010 for 16-bit 6800 parallel interface or to 0b1000 for 8-bit 8080 parallel interface.

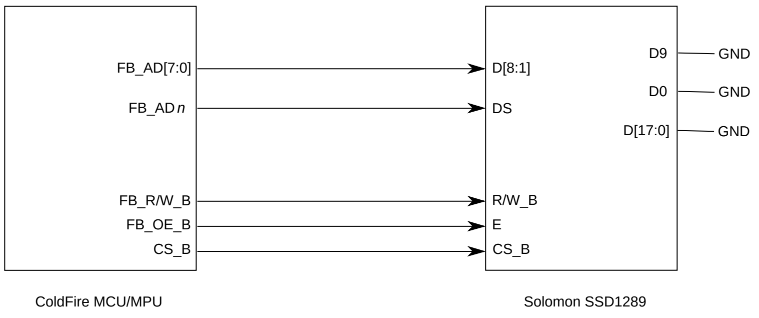


Figure 6. 8-bit bus interface to ColdFire MCU/MPU (non-mux'd bus)

Any pin of FB_AD[19:0] can be used as a DS signal.

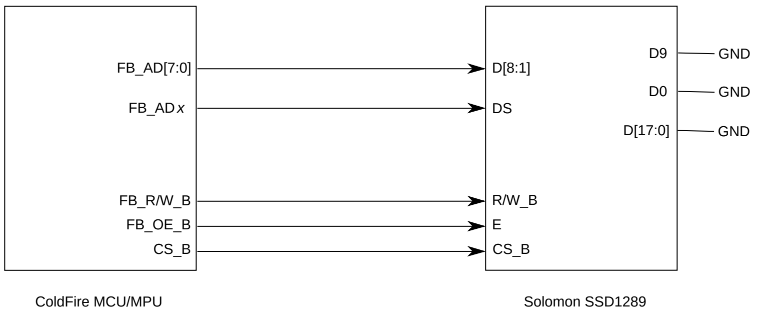


Figure 7. 8-bit bus interface to ColdFire MCU/MPU (mux'd bus)

Any pin of FB_AD[19:8] can be used as a DS signal.

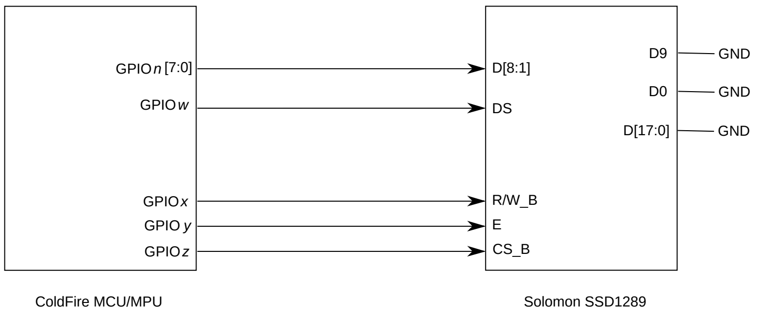


Figure 8. 8-bit bus interface to ColdFire MCU/MPU (simulated by GPIO pins)

Any GPIO pins can be used.

2.3 SPI data mode

Configure PS[3:0] to 0b1110 for 4-wire interface.

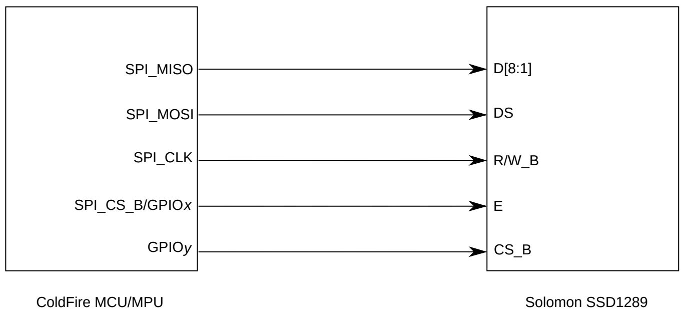


Figure 9. Serial interface to ColdFire MCU/MPU

NOTE

If using serial transfer but with 16 clock cycles per CS low pulse, it is recommended that the data be transferred twice. For example, if we want to send data 0x9290 to register 0x03, the data format is 0303 (command), 9292 (first data transmission), 9090 (second data transmission). For more details, please refer to the SSD1289 Application Note available at www.solomon-systech.com.

3 TWR-LCDSTY board preparation

Plug the TWR-LCDSTY board onto the TWR-ELV functional board. (See Figure 4. The TWR-ELV function board is on the left.)

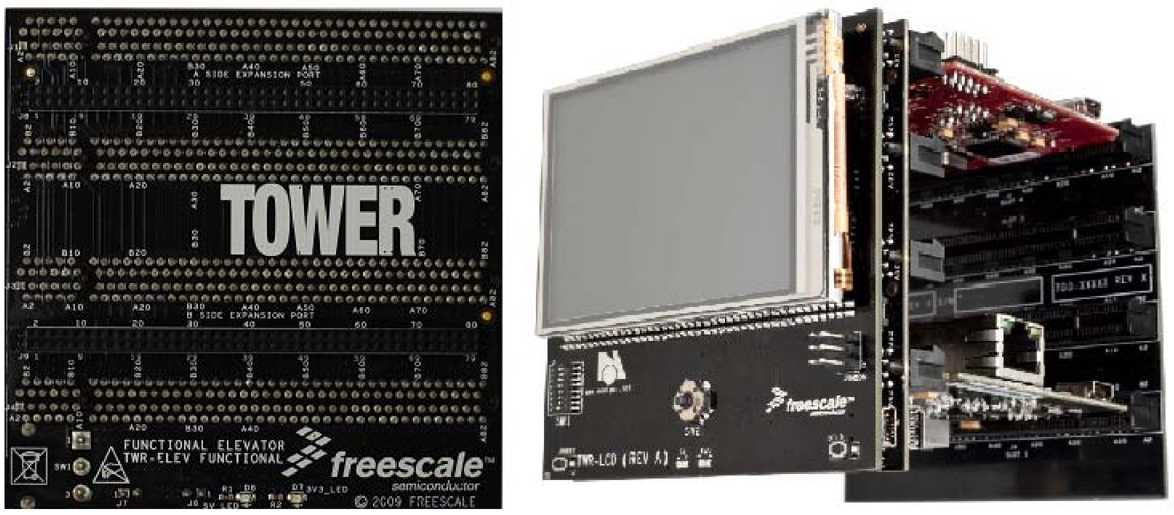


Figure 10. TWR-ELV function board and TWR-LCDSTY board

The TWR-LCDSTY board supports a 16-bit FlexBus/Mini-FlexBus interface and a SPI interface. The content below shows you the DIP switch settings on board:

For 16-bit FlexBus/Mini-FlexBus interface:
- SW[1]: on, PS2 = 0
- SW[2]: off, PS0 = 1

- SW[3]: on, disable the control of MCF51JM128
- SW[4]: for Micro SD card interface, doesn’t matter
- SW[5]: select SPI channel, doesn’t matter
- SW[6]: select TP, doesn’t matter
- SW[7]: on, backlight on
- SW[8]: to control the buzzer on board, doesn’t matter

For 8-bit FlexBus/Mini-FlexBus interface:

- SW[1]: on, PS2 = 0
- SW[2]: off, PS0 = 0
- SW[3]: on, disable the control of MCF51JM128
- SW[4]: for Micro SD card interface, doesn’t matter
- SW[5]: select SPI channel, doesn’t matter
- SW[6]: select TP, doesn’t matter
- SW[7]: on, backlight on
- SW[8]: to control the buzzer on board, doesn’t matter

For SPI interface:

- SW[1]: on, PS2 = 1
- SW[2]: off, PS0 = 0
- SW[3]: on, disable the control of MCF51JM128
- SW[4]: for Micro SD card interface, doesn’t matter
- SW[5]: select SPI channel; on is for SPI channel 0, off is for SPI channel 1
- SW[6]: select TP, doesn’t matter
- SW[7]: on, backlight on
- SW[8]: to control the buzzer on board, doesn’t matter

4 Microcontroller configuration

4.1 Mini-FlexBus interface

For MCF51MM256, to make the Mini-FlexBus work, you must:

- Configure register Mini-FlexBus pin control 1-4 (MFBPC1-MFBPC4) to enable Mini-FlexBus pins. For more details about these registers, please refer to Freescale document MCF51MM256RM, *MCF51MM256 Reference Manual*, section 5.7.19, "MiniFlex Bus Pin Control 1 (MFBPC1)," through section 5.7.22, "MiniFlex Bus Pin Control 4 (MFBPC4)."

For MCF51CN128, to make the Mini-FlexBus work, you must:

- Configure the GPIO pins to a Mini-FlexBus pin by setting the Pin Mux Control registers. For more details about these control registers, please refer to Freescale document MCF51CN128RM, *MCF51CN128 Reference Manual*, section 2.3, "Pin Mux Controls."

Mini-FlexBus settings: (please refer to the chapter “Mini Flex Bus (MFB)” of each device’s reference manual)

- Chip-Select Address Registers: CSARn register specifies the chip-select base address. This address for the TWR-LCDSTY board is 0x400000.
- Chip-Select Mask Registers: CSMRn register specifies the address mark and allowable access types for the respective chip-select. Setting for the TWR-LCDSTY board is 0x00010001; this means addresses from 0x400000 to 0x41FFFF can be accessed by the CS. Therefore address 0x400000 (FB_AD16 is low) can be used to access the index register of the SSD1289, and address 0x410000 (FB_AD16 is high) can be used to access another register or data buffer of the SSD1289.

NOTE

- When connecting CS1 to the CS_B of the LCD controller, bit CSMR0[V] must be set to enable the Mini-FlexBus module. You also need to enable the clock gate of the Mini-FlexBus module. (For MCF51CN128, bit SCGC4[MB] should be set. For MCF51MM256, bit SCGC2[MFB] should be set.)

- According to the SSD1289 data sheet, the minimum write clock cycle time is 100 ns. And there are at least 4 FB_CLK cycles during one read/write flex bus cycle. So FB_CLK should not be higher than 40MHz, or wait/dummy cycles must be inserted.
- The read clock cycle is much longer than the write clock cycle. The minimum clock cycle is 1000 ns. So if you want to read data through the Mini-FlexBus port, the proper number of wait/dummy cycles should be inserted. For detailed information, please refer to the SSD1289 data sheet and the reference manual of the MCU that you are using.
- Register settings of the Mini-FlexBus module for the TWR-LCDSTY board:

```
CSAR0 = 0x400000; /* CS0 base address */
CSCR0 = 0x00000380; /* multiplexed 16-bit mode, 0 wait */
CSMR0 = 0x00010001; /* FB_AD16 is available*/
```

4.2 SPI interface

For MCF51CN128, to make the SPI port work, you must:

- Configure the GPIO pins to the SPI port by setting the Pin Mux Control registers. For more details about these control registers, please refer to Freescale document MCF51CN128RM, *MCF51CN128 Reference Manual*, section 9.7, "Pin Mux Controls."

For MCF51MM256, to make the SPI port work:

- Enabling the SPI module will configure the pins with multiple functions to work as an SPI port.

Other settings:

- The DS signal should be controlled by a GPIO pin.
- The CS signal can be controlled by the SPI_CS or GPIO pin.

For SPI port configuration, you can refer to the SPI chapter of the device's reference manual.

NOTE

- Make sure that the clock gate of SPI module is enabled.
- For TWR-LCDSTY, the LCDC (SSD1289) has a limitation for SPI clock speed. The maximum frequency is 13 MHz. For more details, please refer to SSD1289 data sheet and SSD1289 application note available at www.solomon-systech.com.

5 Low-level driver design

To add 16/8 bit MiniBus, SPI, or GPIO low-level LCD driver support, the APIs below are needed.

Hardware level Driver API introduction: defined in d4d_lldapi.h

```
typedef struct D4DLCDHW_FUNCTIONS_S
{
  unsigned char (*D4DLCDHW_Init)(void);
  unsigned char (*D4DLCDHW_SendDataWord)(unsigned short value);
  unsigned char (*D4DLCDHW_SendCmdWord)(unsigned short cmd);
  unsigned short (*D4DLCDHW_ReadDataWord)(void);
  unsigned short (*D4DLCDHW_ReadCmdWord)(void);
  unsigned char (*D4DLCDHW_PinCtl)(D4DLCDHW_PINS pinId, D4DHW_PIN_STATE setState);
  void (*D4DLCDHW_FlushBuffer)(void);
  unsigned char (*D4DLCDHW_DeInit)(void);
}D4DLCDHW_FUNCTIONS;
```

unsigned char (*D4DLCDHW_Init)(void)

- LCD initialization.

unsigned char (*D4DLCDHW_SendDataWord)(unsigned short value)

- Send word data to LCDC.

unsigned char (*D4DLCDHW_SendCmdWord)(unsigned short cmd)

- Send command word data to LCDC.

unsigned short (*D4DLCDHW_ReadDataWord)(void)

- Read 16-bit data from LCDC.

unsigned short (*D4DLCDHW_ReadCmdWord)(void)

- Read 16-bit command data from LCDC.

unsigned char (*D4DLCDHW_PinCtl)(D4DLCDHW_PINS pinId, D4DHW_PIN_STATE setState)

- Control GPIO pins for LCD control purposes.

void (*D4DLCD_FlushBuffer)(void)

- For buffered low level interfaces is used to inform driver the complete object is drawn and pending pixels should be flushed.

For more details about display driver APIs, please refer to Freescale document DRM116, *Freescale Embedded GUI (D4D)*, available at www.freescale.com/eGUI.

At least three low-level functions for driver API are needed: LCDHW init, Send Command, Write Pixel Data. If you want to read data through a mini-FlexBus port, SPI port, or GPIO, the Read Data function is also needed.

For the Freescale TWR-LCD board, there is also a touch screen deployed. For more details about how to make it work, please refer to DRM116, *Freescale Embedded GUI (D4D)*, available at www.freescale.com/eGUI.

5.1 16-bit Mini-FlexBus data mode driver

It is very easy to design the low-level driver for 16-bit Mini-FlexBus data mode.:

For TWR-LCDSTY board:

```
#define LCD D4DLCD_FLEX_DC_ADDRESS  0x00400000
#define D4DLCD_FLEX_BASE_ADDRESS    0x00410000
```

You can find these macro definitions in d4dlcdhw_flexbus_16b_cfg.h.

Then APIs (found in d4dlcdhw_flexbus_16b.c) could be:

```
  const D4DLCDHW_FUNCTIONS d4dlcdhw_flexbus_16b =
  {
    D4DLCDHW_Init_Flexbus_16b,
    D4DLCDHW_SendDataWord_Flexbus_16b,
    D4DLCDHW_SendCmdWord_Flexbus_16b,
    D4DLCDHW_ReadDataWord_Flexbus_16b,
    D4DLCDHW_ReadCmdWord_Flexbus_16b,
    D4DLCDHW_PinCtl_Flexbus_16b,
    D4DLCD_FlushBuffer_Flexbus_16b,
    D4DLCDHW_DeInit_Flexbus_16b
  };
  static unsigned char D4DLCDHW_Init_Flexbus_16b(void)
  {
    #ifdef D4DLCD_DISPLAY_MCU_USER_INIT
      D4DLCD_DISPLAY_MCU_USER_INIT
    #endif

  D4DLCD_FLEX_CSAR = D4DLCD_FLEX_DC_ADDRESS; // CS0 base address
  D4DLCD_FLEX_CSMR = D4DLCD_FLEX_ADRESS_MASK | D4DLCD_FLEX_CSMR_V_MASK; // The address range
is set to 128K because the DC signal is connected on address wire
  D4DLCD_FLEX_CSCR = D4DLCD_FLEX_CSCR_MUX_MASK | D4DLCD_FLEX_CSCR_AA_MASK |
D4DLCD_FLEX_CSCR_PS1_MASK; // FlexBus setup as fast as possible in multiplexed mode
```

```
    return 1;
  }
  static unsigned char D4DLCDHW_SendDataWord_Flexbus_16b(unsigned short value)
  {
    *((unsigned short*)D4DLCD_FLEX_BASE_ADDRESS) = value;

    return 1;
  }
  static unsigned char D4DLCDHW_SendCmdWord_Flexbus_16b(unsigned short cmd)
  {
    *((unsigned short*)D4DLCD_FLEX_DC_ADDRESS) = cmd;

    return 1;
  }
```

For other APIs, please refer to Freescale's official D4D demo at www.freescale.com/eGUI.

5.2 SPI data mode driver

Here we introduce only the 8-bit SPI for the low-level LCDC driver:

```
  const D4DLCDHW_FUNCTIONS d4dlcdhw_spi_8b =
  {
    D4DLCDHW_Init_Spi_8b,
    D4DLCDHW_SendDataWord_Spi_8b,
    D4DLCDHW_SendCmdWord_Spi_8b,
    D4DLCDHW_ReadDataWord_Spi_8b,
    D4DLCDHW_ReadCmdWord_Spi_8b,
    D4DLCDHW_PinCtl_Spi_8b,
    D4DLCD_FlushBuffer_Spi_8b,
    D4DLCDHW_DeInit_Spi_8b
  };
```

Compared with the 16-bit Mini-FlexBus low-level driver, two necessary APIs are required:

```
  static unsigned char D4DLCDHW_Init_Spi_8b(void)
  {
      #ifdef D4DLCD_DISPLAY_MCU_USER_INIT
        D4DLCD_DISPLAY_MCU_USER_INIT
      #endif

      D4DLCD_ASSERT_CS;
      D4DLCD_ASSERT_DC;

      D4DLCD_INIT_CS;
      D4DLCD_INIT_DC;


      /* Select the highest baud rate prescaler divisor and the highest baud rate divisor */
 //AR
      D4DLCD_SPIBR = 0x00;
      /* SPI Interrupt disable, system enable and master mode selected */ //AR
      D4DLCD_SPIC1 = 0x50;
      D4DLCD_SPIC2 = 0x00;

      return 1;
  }
  static unsigned char D4DLCDHW_SendDataWord_Spi_8b(unsigned short value)
  {
    D4DLCD_ASSERT_CS;

    while (!D4DLCD_SPIS_SPTEF) {};

    // Send data byte
    D4DLCD_SPID = (unsigned char) ((value >> 8) & 0xff);
    while (!D4DLCD_SPIS_SPTEF) {};

    // Send data byte
    D4DLCD_SPID = (unsigned char) (value & 0xff);

    while (!D4DLCD_SPIS_SPTEF) {};

    asm     // 2 nops are OK, 3 nops are OK for QE DEMO, we go for 6 RE-CHECK [PL]
    {
        nop
        nop
        nop
        nop
        nop
        nop
    };

    D4DLCD_DEASSERT_CS;

    return 1;
  }
```

And the other one could call the basic function on:

```
  static unsigned char D4DLCDHW_SendCmdWord_Spi_8b(unsigned short cmd)
  {
    unsigned char ret;
    D4DLCD_ASSERT_DC;                           // DataCmd := 0
    ret = D4DLCDHW_SendDataWord_Spi_8b(cmd);
    D4DLCD_DEASSERT_DC;                         // DataCmd := 1

    return ret;
  }
```

Before you can use these low-level drivers to send commands or data to SSD1289, you need to initialize the SPI port. This means you must modify the definition of "D4DLCD_DISPLAY_MCU_USER_INIT" which can be found in d4dlcdhw_flexbus_16b_cfg.h. For more details about the Serial Peripheral Interface (SPI), please refer to the appropriate reference manual for the device.

NOTE

- The CS signal can be controlled by SPI_CS directly by setting the control register of the SPI module: SPIxC1[SSOE] = 1, SPIxC2[MODFEN] = 1.
- When using the TWR-MCF51CN board, you must configure the MCU pins from the GPIO port to the SPI port by setting the Pin Mux Control registers.
- When configuring the SSD1289 to 4-wire SPI mode, the QSPI/DSPI port on the MCU side can also be used. For more details about QSPI/DSPI, please refer to the Coldfire (V2 or above) MCU reference manual.

5.3 8-bit data bus mode interface

It is possible to simulate both 6800 and 8080 series parallel timing by controlling the GPIO pins. Here is code for the 8-bit 6800 series GPIO low-level LCDC driver:

```
  const D4DLCDHW_FUNCTIONS d4dlcdhw_gpio6800_8b =
  {
    D4DLCDHW_Init_Gpio6800_8b,
    D4DLCDHW_SendDataWord_Gpio6800_8b,
    D4DLCDHW_SendCmdWord_Gpio6800_8b,
    D4DLCDHW_ReadDataWord_Gpio6800_8b,
    D4DLCDHW_ReadCmdWord_Gpio6800_8b,
    D4DLCDHW_PinCtl_Gpio6800_8b,
    D4DLCD_FlushBuffer_Gpio6800_8b,
    D4DLCDHW_DeInit_Gpio6800_8b
  };
```

```
static unsigned char D4DLCDHW_Init_Gpio6800_8b(void)
{
  #ifdef D4DLCD_DISPLAY_MCU_USER_INIT
    D4DLCD_DISPLAY_MCU_USER_INIT
  #endif

  D4DLCD_ASSERT_CS;
  D4DLCD_ASSERT_DC;

  D4DLCD_INIT_CS;
  D4DLCD_INIT_DC;


  D4DLCD_INIT_DATA;    // 8 bit data port all outputs

  D4DLCD_INIT_RW;
  D4DLCD_INIT_E;

  D4DLCD_DEASSERT_RW;
  D4DLCD_DEASSERT_E;

  return 1;
}

static unsigned char D4DLCDHW_SendDataWord_Gpio6800_8b(unsigned short value)
{
  D4DLCD_ASSERT_RW; // RW = 0

  D4DLCD_DEASSERT_E; // E = 1;

  D4DLCD_ASSERT_CS;   // CS = 0;

  D4DLCD_DATA_RWITE((unsigned char) ((value >> 8) & 0xff)); // msb!

  D4DLCD_DEASSERT_CS;
  D4DLCD_ASSERT_CS;

  D4DLCD_DATA_RWITE((unsigned char) (value & 0xff));           // lsb!

  D4DLCD_DEASSERT_CS;

  return 1;
}

static unsigned char D4DLCDHW_SendCmdWord_Gpio6800_8b(unsigned short cmd)
{
  D4DLCD_ASSERT_DC;                        // DataCmd := 0
  D4DLCDHW_SendDataWord_Gpio6800_8b(cmd);
  D4DLCD_DEASSERT_DC;                      // DataCmd := 1

  return 1;
```

5.4 Integrate low-level drivers with eGUI

When the low-level drivers are ready, it is easy to integrate the board-related low-level drivers with the eGUI. You can modify some definitions in d4d_user_cfg.h to make your drivers work on your own board. You can change the macros below:

```
#define D4D_LLD_LCD d4dlcd_ssd1289   // the name of low level driver descriptor structure
#define D4D_LLD_LCD_HW d4dlcdhw_flexbus_16b   // the name of LCD hw interface driver descriptor
#define D4D_LLD_TCH  d4dtch_resistive
#define D4D_LLD_TCH_HW d4dtchhw_s08_adc
```

6 Summary

Because the LCDC (SSD1289) controls the refresh rate of the LCD, the content shown below indicates the frame rate only when the MCU is loading display data into the LCDC display buffer.

Table 4. Frame rate for each interface

Type of interface	Performance (unit: frames/s)
16-bit parallel Mini-FlexBus	34.7
8-bit parallel bus (GPIO)	6.9
4-wire SPI	4.2

Test conditions:

- MCU runs at 48 MHz, FB_CLK = 24 MHz, SPI_CLK = 12 MHz.
- Fill screen with single color, set TFT-LCD to 16-bit (64k color), screen size = 320 by 240.
- Test C code loop show as below:

```
LOOP:
    for(i=76800;i>0;i--)
    {
        D4DLCDHW_SendDataWord (0xF800); // fill with RED
    }
    PTED_PTED5 = ~PTED_PTED5;        // trigger for scope
    goto LOOP;
```

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

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4153
Rev. 0, 08/2010