Freescale Semiconductor
Application Note

Document Number: AN3559
Rev. 0, 03/2008

Implementing an LCD Module to the MCF5223x

Extended Display for V2 Coldfire

by: Shen Li
Asia & Pacific Operation Microcontroller Division

1 Introduction

A liquid crystal display (LCD) is a useful module for a user interface feature of an embedded system. It provides a parallel interface and a serial interface of the SPI and IIC that connects to the processor.

The MCF5223x supports both SPI and IIC interfaces for communications.

This document introduces how to implement a serial interface (QSPI/IIC) LCD module for the MCF5223x. It also introduces the hardware connection and software driver development.

Contents

freescale semiconductor

2 MCF5223x Family

The MCF5223x family is a highly integrated implementation of the ColdFire family. It includes the MCF52231, MCF52233, MCF52234, and MCF52235. This family is designed for industrial and commercial control with an Ethernet feature requirement. This 32-bit device is based on the second version of the ColdFire 2 (V2) reduced instruction set computing (RISC) core. It includes the following on-chip modules:

- ColdFire V2 core with enhanced multiply-accumulate unit (EMAC)
- Cryptographic acceleration unit (CAU)
- 32 Kbytes of internal SRAM
- Up to 256 Kbytes of on-chip flash memory
- Fast Ethernet controller (FEC) with an on-chip transceiver (ePHY)
- Three universal asynchronous receivers/transmitters (UARTs)
- Controller area network 2.0B (FlexCAN) module
- Inter-integrated circuit (IIC) bus controller
- 12-bit analog-to-digital converter (ADC)
- Queued serial peripheral interface (QSPI) module
- Four-channel, 32-bit direct memory access (DMA) controller
- Four-channel, 32-bit input capture/output compare timers with optional DMA support
- Two 16-bit periodic interrupt timers (PITs)
- Programmable software watchdog timer
- Two interrupt controllers, each capable of handling up to 63 interrupt sources (126 total)

In this document a QSPI and IIC interface were adapted for the serial interface LCD.

3 LCD Module

The LCD module is one of the popular man-machine interfaces for an embedded system. Most LCD modules have a built-in LCD controller that supplies an easy-to-use serial bus, such as the QSPI and IIC interfaces. The LCD module allows a clear output of characters in a grid format. It also contains a library ROM for ASCII to grid an array conversion. It supports one, two, or four rows with each row containing sixteen, twenty, or forty characters. In this application note, a YM12864R P-1 16×4 LCD module for a QSPI interface application and a GLK12232-25-SM LCD module for a IIC interface application are chosen as examples.

3.1 QSPI LCD

Figure 1 shows a module using the Sitronix ST7920 as an LCD controller and driver. The Sitronix ST7920 supports ASCII characters and Chinese characters. Although it provides parallel and serial interface. Although the serial interface in this application note is used.

Figure 1. YM12864R P-1 16x4 LCD Module

The main features of this LCD are listed below:

- Working voltage: 2.7 V to 5.5 V
- 8-bit/4-bit parallel or serial MPU interface
- 64x16-bit character cell
- 2 Mbytes CGROM library to support 8192 Chinese characters (16x16 grid array)
- 16 Kbytes HCGROM library to support 126 ASCII characters (16x8 grid array)
- Auto powerup reset function and external reset function
- Low power design, normal mode (typical 450 µA in V_{dd}=5 V), standby mode (maxinum 30 µA in V_{dd}=5 V), and sleep mode (maxinum 3 µA in V_{dd}=5 V)
- Multi-display function
  - Display clear
  - Cursor return home
  - Display on/off
  - Cursor on/off
  - Display character blink
  - Cursor shift
  - Vertical line scroll
  - By_line reverse display
  - Sleep mode

This LCD serial interface has three function pins:

- Chip select (CS)
- SCLK (clock input)
- SID (serial data).

These pins are common in most serial LCD modules.

This application applies to general serial LCD modules and focuses on the timing of the serial interface, which is almost the same for all LCD modules.

When the interface is configured in a serial mode the CS, SCLK, and SID work as serial function pins. The SCLK can be absorbed by the LCD controller only if activated when the CS is in high active logic. When the CS is low logic, the internal serial counter resets and the transferring data inside the controller is cleared.

NOTE

There is no receive buffer inside the LCD controller. Therefore, the MPU transfers the following data after the instruction has been executed.

The timing diagram of a frame that contains 24-bit data is shown in Figure 2.

The first five bits of one form a synchronizing bit string. The synchronizing bit string is followed by an RW flag, an RS flag and a 0. The RW bit indicates a transfer in direction. 0 identifies the direction from the MPU to the LCD controller. And 1 identifies the direction from the LCD controller to the MPU. The RS bit indicates register selection, 0 identifies register access, and 1 identifies data buffer access. These eight bits compose the frame head byte.

When the controller recognizes the flag it resets the internal counter and starts to receive the following data. The data to be transferred is divided into two parts. The second byte of the frame is composed of the high four bits and the four zeros that follow. The low four bits and four zeros that follow form the third byte of the frame.

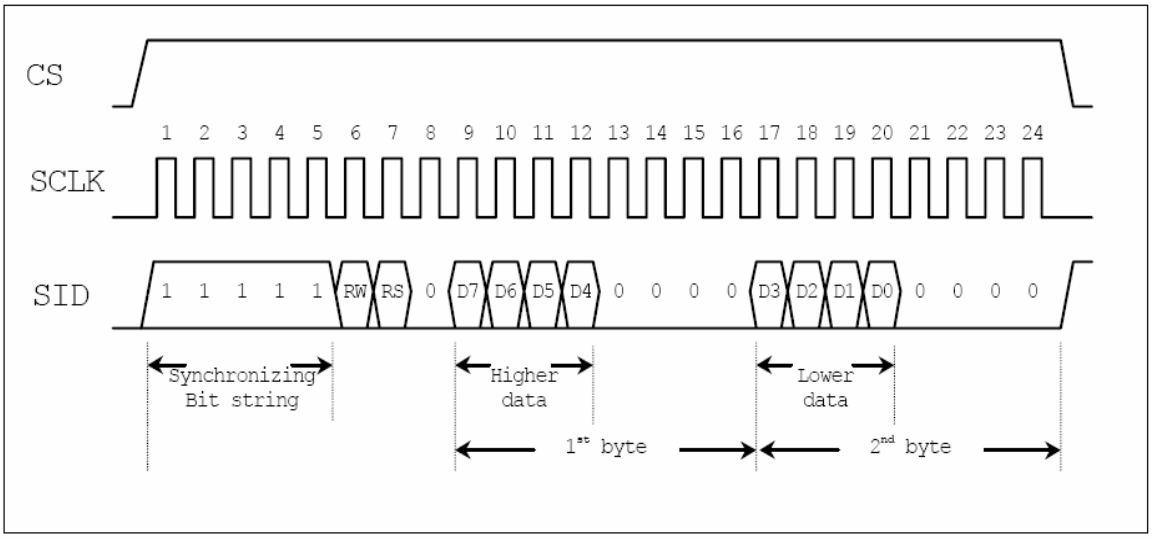

Figure 2. Timing Diagram of Serial Mode Data Transfer

3.2 IIC LCD

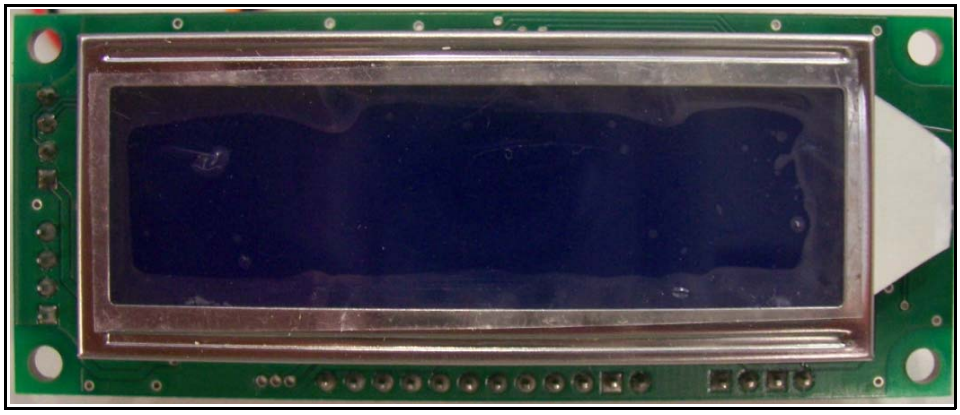

Figure 3. GLK12232-25-SM LCD Module

As shown in Figure 3, the GLK12232-25-SM is selected as the LCD module in this example. It has the following features:

- 122 × 32 pixel graphics display
- Text display using built-in or user supplied fonts
- Adjustable contrast with software control
- Adjustable backlighting with software control
- RS-232 or IIC communications
- Extended voltage module available (-V)

The IIC communications of this LCD module runs up to 100 kbps. The IIC data line works on 5 V CMOS levels. The IIC interface of the LCD module is completely compatible with the Phillips IIC specification.

The IIC is a two-wire bidirectional serial bus that provides a simple and efficient method that exchanges data. It reduces the interconnection between devices. Two wires that are serial data (SDA) and serial clock (SCL) connect to the bus and carry information between the devices. Each device recognized by a unique 7-bit address operates as a transmitter or receiver. For example, the LCD module is a receiver. In the IIC bus, all devices can be considered as masters or slaves when performing data transfer. The master initiates a data transfer on the bus and generates clock signals to permit that transfer. At that time, any other devices addressed are considered as slave. In this application, the MCF5223x is the master and the LCD module is the slave.

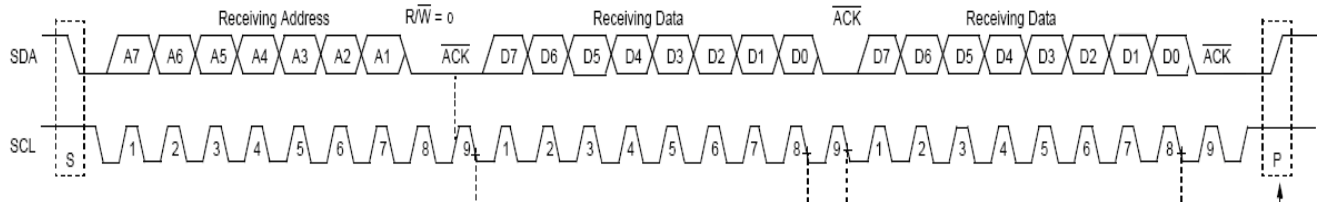

Figure 4. IIC Timing of the LCD Module

Figure 4 shows the timing of the LCD module IIC interface. The SDA signal triggers the transfer by a high to low edge and then transfers a 7-bit slave device address and R/W bit. The R/W flag indicates the direction of this transfer. In this application, the direction is always from master to slave. The R/W flag is always 0. After, that the LCD sends the $\overline{ACK}$ flag by pulling down the SDA at the next bit after acknowledging the address. The master then continues transferring the following byte.

4 ColdFire QSPI Module

In this chapter, the QSPI module was used to serve the LCD module connection. The queued serial peripheral interface (QSPI) module provides a serial peripheral interface with queued transfer capability. This allows users to queue up to 16 transfers at once. This eliminates CPU intervention between transfers.

4.1 Main Features of the QSPI

The main features of the QSPI are listed below:

- Programmable queue to support up to 16 transfers without user intervention
- Supports transfer size of 8 to 16 with 1-bit increments
- Four peripheral chip-select lines for control of up to 15 devices (All four chip selects may not be available on all devices.)
- Baud rates from 117.6 kbps to 15 Mbps at 60 MHz internal bus frequency
- Programmable delays before and after transfers
- Programmable QSPI clock phase and polarity
- Supports wraparound mode for continuous transfers
- Only supports master mode

4.2 Registers of the QSPI Module

The QSPI module includes the registers as shown in Table 1.

Table 1. QSPI Module Registers

IPSBAR Offset	Register	Width (Bits)	Access	Reset Value
0x00_0340	QSPI Mode Register (QMR)[1]	16	R/W	0x0104
0x00_0344	QSPI Delay Register (QDLYR)[2]	16	R/W	0x0404
0x00_0348	QSPI Wrap Register (QWR)[3]	16	R/W	0x0000
0x00_034C	QSPI Interrupt Register (QIR)[4]	16	R/W	0x0000
0x00_0350	QSPI Address Register (QAR)[5]	16	R/W	0x0000
0x00_0354	QSPI Data Register (QDR)[5]	16	R/W	0x0000

[1] QMR is used to configure the frame size, transfer baud rate, and clock phase.
[2] QDLYR is used to configure the delay time before and after transfer.
[3] QWR is used to control the queue point and wrap around mode.
[4] QIR is used to enable the interrupt and indicate interrupt status.

[5] QAR and QDR are used to access the Static RAM.

Refer to the *MCF5223x Reference Manual* for details of the QSPI registers.

4.3 QSPI Architecture and Function

Figure 5 shows the architecture of the QSPI module.

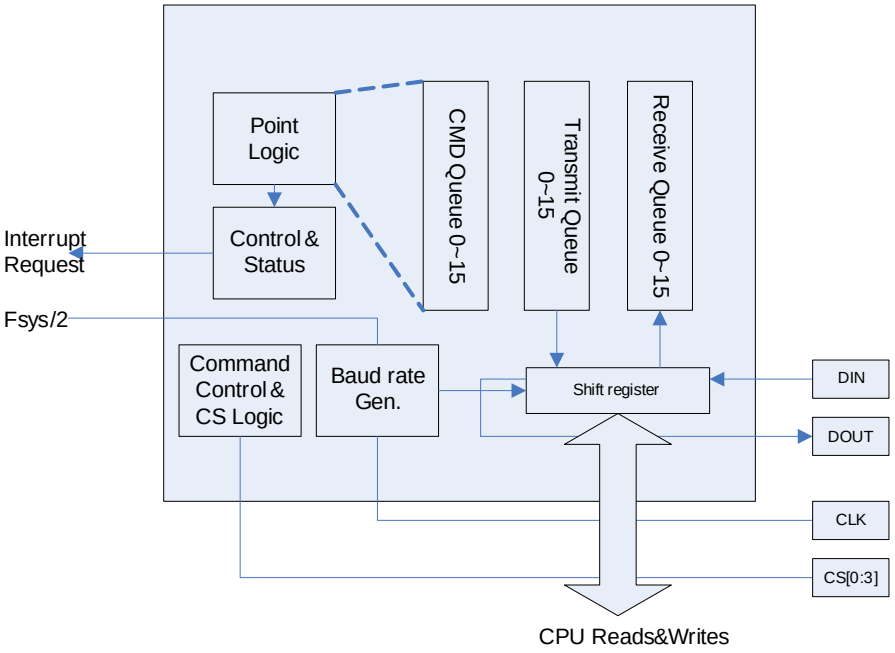

Figure 5. Architecture of the QSPI Module

The QSPI module defines three queue pointers to control the queue transfer:

- NEWQP points to the start of the queue to be transferred.
- ENDQP points to the end of the queue to be transferred.
- CPTQP points to the last element in the queue executed.

The QSPI uses a dedicated 80-byte static RAM block accessible to the module and the CPU to perform operations. The RAM is divided into three segments:

- 16 command control bytes (command RAM), takes the address of 0x20–0x2F.
- 32 transmitting data bytes (transfer RAM), takes the address of 0x00–0x0F.
- 32 receiving data bytes (transfer RAM), takes the address of 0x10–0x1F.

The 80-byte static block can be accessed indirectly by the CPU and accessed directly by the QSPI. The CPU can only access the three RAMs by the QSPI address register (QAR) and the QSPI data register (QDR). The RAM for transmitting, the RAM for receiving, and the command RAM are at the address of 0x20–0x2F.

The CPU initiates QSPI operation by writing the command RAM queue and any outbound data to the transmitting RAM queue. After the required number of commands and data frames are written. The CPU initializes the NEWQP and ENDQP with desired values, and then enables the QSPI by setting the SPE enabling bit in the QSPI delay register (QDLYR).

When the QSPI is enabled, the NEWQP is copied into an internal pointer that increments to the next entry each time a command is executed. As the internal pointer reaches the value in the ENDQR and the instruction pointed by ENDQR is executed, the QSPI sets the finish flag (SPF) and signals an interrupt request to the CPU if enabled. The LCD data frame contains 24 bits split it into three bytes and writes the three bytes to the queue as one transfer. Refer to Chapter 8, “LCD Driver Development for NicheTask,” for detailed information.

In this application, wraparound mode was not used.

5 ColdFire IIC Module

The MCF5223x IIC module can operate at high baud rates, up to a maximum of the internal bus clock divided by 20 with reduced bus loading.

5.1 Main Feature of the IIC Module

The main feature of the IIC module are listed below:

- Compatible to the IIC bus standard version 2.1
- Support for 3.3 V tolerant devices
- Multi-master operation
- Software programmable for one of 50 different serial clock frequencies
- Software selectable acknowledge bit
- Interrupt-driven
- Byte data transfer
- Loss of arbitration interrupt with automatic mode switching from master to slave
- Address identification interrupt
- Start and stop signal generation/detection
- Repeated start signal generation
- Acknowledge bit generation/detection
- Bus busy detection

5.2 IIC Module Registers and Functions Implementation

Table 2. IIC Module Registers

IPSBAR Offset	Register	Access	Reset Value
0x00_0300	IIC Address Register (IIADR)[1]	R/W	0x00
0x00_0304	IIC Frequency Divider Register (IIFDR)[2]	R/W	0x00
0x00_0308	IIC Control Register (IICR)[3]	R/W	0x00
0x00_030C	IIC Status Register (IISR)[4]	R/W	0x81
0x00_0310	IIC Data I/O Register (IIDR)[5]	R/W	0x00

1 The IIADR holds the address responded by IIC when in slave mode. It is of no use in this application.

2 IIFDR provides a programmable prescaler to configure the IIC clock for bit-rate selection.

3 IICR is used to enable IIC module and IIC interrupt. It also contains bits that govern operation as a slave or a master.

4 IISR contains bits that indicate transaction direction and status.

5 IIDR is used to receive and transmit data on IIC bus. In master transmitting mode, when data is written to this register, a data transfer is initiated. The most significant bit is sent first.

In this application, after the system is reset the software configures the MCF5223x device as a bus master and expects the IIC bus to be idle. Because of a software request, the master initiates an address to select the LCD module. The normal data transfer protocol begins with a START bit, then followed by a 7-bit address, a read/write bit, and an acknowledge bit from the addressed slave. After, the CPU keeps on sending data by writing to the IIDR.

To end the transfer protocol, the master issues a STOP bit. The bus then goes idle.

6 Hardware Connection

Figure 6 shows the hardware connection between ColdFire and the LCD module.

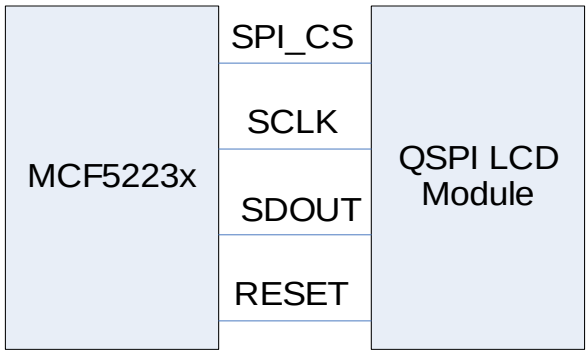

Figure 6. QSPI LCD Interface Connection with ColdFire

The MPU chip select, SCLK, SDOUT and $\overline{\text{RESET}}$ (active low) signal is connected to the LCD controller. The controller also has the V_{DD}, GND, mode select, and contrast adjust pin for configuration.

Figure 7 shows the IIC LCD interface hardware connections.

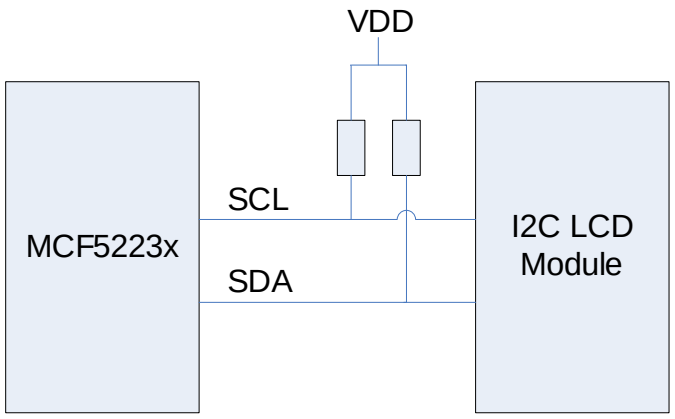

Figure 7. IIC LCD Interface

SCL and SDA are two signals in IIC connections. Both signals must be pulled up to V_{DD} by 2~10 kΩ resistor, because the SCL and SDA are open-drain or open-collector to perform the wired-AND function.

7 NicheTask Open Source RTOS Introduction

InterNiche technologies provides the NicheTask open source operating system for the MCF5223x. The NicheTask is royalty free. Users can get the source code from the Freescale website www.freescale.com. The LCD driver used in this application note is based on this operating system. Users can easily port the driver because the NicheTask has similar features to most operating systems.

NicheTask main features are listed as follows:

- Non-preemptive – requires that a task gives control to the next task.
- Each task has its own stack.
- Contains network stack.
- A task goes to sleep based on time or an event.
- If a task is not sleeping, it is ready to run.
- Developers can add their own task via the tk_new() function, but must sleep to give control to the next task in the task list
- There are no priorities. When task 1 gives up control, the RTOS tries to run task 2 and so on.
- Task 1 uses the system task, and must be the network task

7.1 Main Task Function

The main task functions are listed below:

- task * tk_init() - initializes the RTOS
- task * tk_new() - adds a new task to the task list
- void tk_block() - switches to the next runnable task
- void tk_exit() - ends and deletes current task
- void tk_kill() - marks a task for termination
- void tk_wake() - marks a task to run
- void tk_sleep() - sleeps for a number of CPU ticks
- void tk_ev_block() - blocks until event occurs
- void tk_ev_wake() - wakes tasks that are waiting for an event

Refer to application note *AN3470 – ColdFire TCP/UDP/IP Stack and RTOS* for details of these functions at www.freescale.com.

8 LCD Driver Development for NicheTask

8.1 Display Data Flow Chart

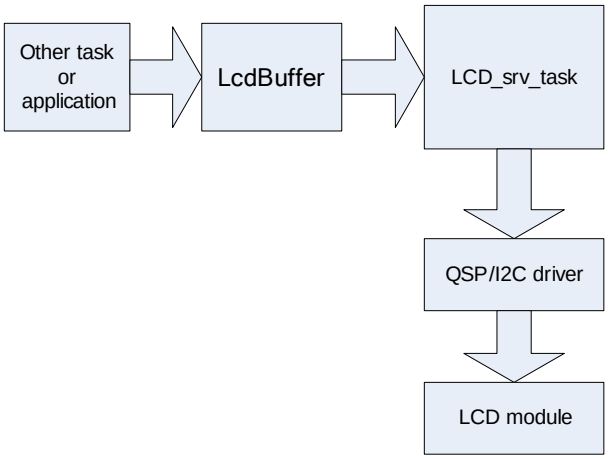

Figure 8. Data Flow of Display

Figure 8 shows the display data flow chart. There is a display buffer (LcdBuffer) in the system located in system RAM and can be accessed by any task. When any task or application function writes its display content to the LcdBuffer and updates refresh flags (detail in 8.3.2.1), the LCD serve task loads the LcdBuffer data to the LCD module through the QSP/IIC driver. The following sections introduce the implementation of the LCD serve task and QSP/IIC driver.

8.2 Create an Application Task

The following steps show how to create an application task:

1. Declare task information structure (struct inet_taskinfo app_task) to define the task name, entry, stack size, and so on.
2. Declare the entry function of the task and add the application code in the entry body. The function must be a none return loop.
3. Use TK_NEWTASK to create a task after the netmain task. Netmain must be the first task of the RTOS

In NicheTask RTOS, the application creates a task to:

- Serve the LCD module
- Respond to the LCD display content update
- Provide a display buffer for other tasks or application functions that update the LCD display content.

8.3 LCD Serve Task

8.3.1 Create the LCD Serve Task

Follow the steps below to create the task:

1. Define the task structure:

```
struct inet_taskinfo LCD_srv_task = {
    &to_LCDsrv,/*the pointer to the static task object */
    "LCD  server",/*task name */
    tk_LCDsrv,/*the entry function of the task */
    0,       /*0 is the priority, it is of no use in this example */
    0x800    /*the stack size of the task */
                                };
```

2. Add the TK_NEWTASK function to the initial process of create_apptasks( ):

```
…
   e = TK_NEWTASK(&LCD_srv_task); //create the task
   if (e != 0) //check if success
   {
          dprintf("LCD create error\n");
          panic("create_apptasks");
          return -1;  /* compiler warnings */
   }
…
```

If this process creates the LCD_srv_task successfully, the task is added to the RTOS task list.

8.3.2 Add LCD Serve Function

The main entry function of the LCD_srv_task is an endless loop that serves the LCD module access routine. The task provides a display buffer for other tasks or application functions to change the content of LCD display. Depending on the defined macro of the IIC_INTERFACE, the LCD serve functions call the IIC module driver or the QSPI module driver, if the IIC_INTERFACE is defined. The serve functions call the IIC module driver and executes the IIC LCD module routine. If the IIC_INTERFACE is not defined, the serve functions call the QSPI module driver and execute the QSPI LCD module routine.

8.3.2.1 Global Variable and Array

There are some useful global variables and arrays:

- Unsigned char LcdBuffer[ROW][COL]

  This is the display buffer resided in the SRAM with the size of ROW*COL in bytes. The user can decide LcdBuffer size based on the LCD type.
- Unsigned char LCD_update=0

  This flag is to mark whether the LCD has been updated. If the LCD_update is 1, the LCD_srv_task loads the LcdBuffer content to the LCD module. If the LCD_update is 0, the LCD_srv_task sleeps and gives the CPU control of the next task.
- Unsigned char LcdRowWriteEnable[ROW]

  This array indicates whether each corresponding row is allowed to refresh. If one row WriteEnable is 1, the LCD_srv_task loads the LcdBuffer content of the corresponding row into the LCD module.

The LcdRowWriteEnable and LCD_update are two different flags to save the CPU resource when no data needs to be refreshed. The LCD_update is for the whole frame. The LcdRowWriteEnable is for the corresponding row of the frame.

8.3.2.2 LCD_srv_task Main Entry

Here is the main entry of the task:

```
TK_ENTRY(tk_LCDsrv)
{
    LCD_init();
    for(;;)
    {
        LCD_check();
        TK_SLEEP(4);//4RTOS click = 20ms
        if (net_system_exit)
        break;
    }
    TK_RETURN_OK();
}
```

The LCD_init() implements the LCD module initial function. Details are mentioned in the next section. The LCD_check() checks the LCD_update flag and decides whether to update or return. After return, the task sleeps with TK_SLEEP(). This is because the LCD display is low priority in an embedded system. It does not take much CPU time or the schedule time in a loop task list. Here the LCD sleeps during 4 RTOS clicks which equal 20 ms. This depends on the TPS defined in the RTOS system configuration (ipport.h). 20 ms is enough for the display service because the refresh frequency is 50 kHz. The user must decide the refresh frequency, depending on the real project request.

8.3.2.3 LCD_init() Function

```
void LCD_init()
{
      #ifdef IIC_INTERFACE
      init_IIC(1); //initilize IIC module
      #else
      init_spi(8,0); //initilize spi interface as 8bit , no interrupt
      LcdSendCommand(0x30);//8bit control interface
      LcdSendCommand(0x02);//set AC to 0, and move cursor to 0
      LcdSendCommand(0x04);//set direction of the cursor moving
      LcdSendCommand(0x0c);//set the display on and no cursor display
      LcdSendCommand(0x01);//clear display
      LcdSendCommand(0x80);//set DDRAM address to AC
      #endif
      flush_buf(); //init LCD frame buffer

}
```

When the IIC_INTERFACE is defined, it initializes the IIC module with the init_IIC() function. If the IIC_INTERFACE is not defined it initializes the spi module with the init_spi() function, Afterwards, the function initializes the display buffer LcdBuffer and updates the flag. Then, the function sends commands to initialize the LCD module. Depending on the real LCD module requirement the user must rewrite the LCD initialization routine.

8.3.2.4 LCD_check( ) Function

```
void LCD_check()
{
      if(LCD_update == 1) //need to update LCD
      {
          LCD_DisplayBuf();
          LCD_update = 0;
      }
}
```

This function is to check the update flag of the display buffer.

8.3.2.5 LCD_DisplayBuf( ) Function

```
void LCD_DisplayBuf()
{
      unsigned char i, j;
      for (i = 0; i < ROW; i ++)
      {
               if (LcdRowWriteEnable[i])  //allow this line to refresh
               {
               #ifdef IIC_INTERFACE
               IIC_display_line(IIC_LCD_ADDRESS, LcdBuffer[i], i);
               #else
               LcdSendCommand(0x80+ ((i&0x1)<<4) + ((i>>1)<<3));//move the cursor
               for (j = 0; j < COL; j ++)
               {//16 character every line
               LcdSendData(LcdBuffer[i][j]);//refresh new data into the controller
               }
               #endif
               LcdRowWriteEnable[i] = 0;//forbid this ROW to refresh after this action
          }
      }
}
```

This is the update content function for the LCD module. It checks the write enable flag of every line and sends the content of the LcdBuffer to the LCD module by character. For the IIC interface routine, function IIC_display_line( ) is described in 8.3.2.7.

NOTE

The command and write content routine may be different depending on the LCD controllers. The user must check the update action and command of the LCD controller.

8.3.2.6 LcdSendCommand( ) and LcdSendData( )

These two functions are only for the QSPI module LCD:

```
void LcdSendCommand(char cCommand)
{
      write_to_qspi_ram(QSPI_TX_RAM,0xf8);
      write_to_qspi_ram(QSPI_TX_RAM+1,cCommand & 0xf0);
      write_to_qspi_ram(QSPI_TX_RAM+2,cCommand << 4);
      write_to_qspi_ram(QSPI_COMMAND_RAM,SPI_COM_CONT|0x0F00);
      write_to_qspi_ram(QSPI_COMMAND_RAM+1,SPI_COM_CONT|0x0F00);
      write_to_qspi_ram(QSPI_COMMAND_RAM+2,SPI_COM_CONT|0x0F00);
      start_spi_xfer(3,0); //transfer 3 element of the queue, no csiv
      delay(27); //the LCD module need time to execute the command
}
```

The LcdSendData( ) function is the same as the LcdSendCommand, except for the first byte of the transfer frame. For this application, the LCD module LcdSendData's first byte is 0xFA not 0xF8.

These functions are the low-level interface between the LCD serving routine and the QSPI driver. For the LCD module timing, one complete frame contains 24-bit data (refer to Figure 2). In this application, the SPI module configures the queue width as 8-bit and uses three elements of the queue for one frame. As in the LcdSendCommand( ), the program writes the content to be transferred to three data Ram elements and

command Ram elements. The function uses start_spi_trans( ) to trigger the QSPI transfer and waits for the completion of the transfer. After the transfer, the function gives the LCD controller time to execute the transfer command or update data. The time depends on the specification of the LCD controller. As an alternative, the user can make the task sleep to a specific time. This makes the system more efficient.

8.3.2.7 IIC_display_line( ) and IIC_display_C_COLROW

This function is only for the IIC LCD module.

```
void IIC_display_line(UINT8 address,UINT8* disp_buf,UINT8 line)
{
      UINT8 i,buf[6];
      buf[0]=0;
      buf[1]=line+1; //because the GLK12232-25-SM LCD module starting ROW is 1
      IIC_send_command(SET_INSPO_COLROW, address, 2, buf);
      i = strlen((char*)disp_buf);
      IIC_send_data(address,i,disp_buf);
      return;
}
```

The IIC_display_line( ) function is used to display a line string in a designated line of the LCD, and calls the IIC driver to implement the function.

```
void IIC_display_C_COLROW(UINT8 address, char c, UINT8 COL, UINT8 ROW)
{
      UINT8 i,buf[6];
      buf[0]=COL;
      buf[1]=ROW;
      IIC_send_command(SET_INSPO_COLROW, address, 2, buf);
      IIC_send_data(address,1,&c);
}
```

The IIC_display_C_COLROW( ) function is similar to the IIC_display_line( ) and is used to display a character at a designated position.

8.3.2.8 Application Display Example

Here is an example of the display application:

```
void display_test() //display test function
{
      unsigned char i,j;
      unsigned char lab1[ROW][COL]={
      "Hello world!",
      "Freescale Coldfi",
      "re K2E LCD demo",
      {'B','y',' ','S','h','e','n','L','i',0x01}
      };
      LCD_update = 1;
      for(i=0;i<ROW;i++)
      {
           LcdRowWriteEnable[i]=1; //enable refresh this line
           for(j=0;j<COL;j++)
           {
                if(lab1[i][j]==0) break;
                LcdBuffer[i][j]= lab1[i][j]; //just for test
           }
      }
}
```

8.4 QSPI Driver

The main interface between the LCD task and the QSPI driver is the LcdSendCommand and LcdSendData. The QSPI driver provides the following functions for the interface:

8.4.1 init_spi( )

```
void init_spi(unsigned char bitcnt, unsigned int Interrupt)
{
      MCF_QSPI_QDLYR = 0; // No delay, disable transfer
      // QSPI interrupt
      MCF_QSPI_QIR   = Interrupt;
      if(Interrupt != 0)
      {
      //QSPI interrupt is ICR18, so the vector table index is 18+64=0x52
           MCF_INTC0_ICR18 = MCF_INTC_ICR_IL(4);
           MCF_INTC0_IMRH  &= ~MCF_INTC_IMRL_MASK18;
           MCF_QSPI_QIR |= 0xf;
      }
      // QMR[BAUD] = fsys/ (2 × [desired QSPI_CLK baud rate])
      // Using 15 yields a baud rate of 2MHz
      MCF_QSPI_QMR = (0|
                  MCF_QSPI_QMR_MSTR|
                  MCF_QSPI_QMR_CPHA|
                  MCF_QSPI_QMR_BITS(bitcnt)|
                  MCF_QSPI_QMR_BAUD(15)
                  );
          MCF_GPIO_DDRQS = 0;
          MCF_GPIO_PQSPAR =  MCF_GPIO_PQSPAR_PQSPAR3(0x1) |
          MCF_GPIO_PQSPAR_PQSPAR2(0x1) |
          MCF_GPIO_PQSPAR_PQSPAR1(0x1) |
```

```
            MCF_GPIO_PQSPAR_PQSPAR0(0x1);
}
```

This function initializes the QSPI module of the MCF5223x. The input variable of the bitcnt is the bit width of the queue and parameter of the interrupt. This is for the MCF_QSPI_QIR register configuration. In this application, the LCD requires the SPI to be configured at 2 MHz speed. The function then configures the dedicated QSPI pin as the primary function.

8.4.2 write_to_qspi_ram( ) and read_from_qspi_ram( )

```
void write_to_qspi_ram( uint8 address, uint16 data )
{
      MCF_QSPI_QAR = address;
      MCF_QSPI_QDR = data;
}
uint16 read_from_qspi_ram( uint8 address )
{
      MCF_QSPI_QAR = address;
      return( MCF_QSPI_QDR );
}
```

The QSPI module ram can be indirectly accessed through the QAR and QDR register. These functions are used to write and read from the QSPI module RAM, including transfer RAM, receive RAM, and command RAM.

8.4.3 start_spi_xfer ( )

```
void start_spi_trans(uint8 bytes, uint8 csiv)
{
      MCF_QSPI_QIR = MCF_QSPI_QIR_SPIF;                       //clear the complete flag
      if( csiv == 1 )
          MCF_QSPI_QWR = MCF_QSPI_QWR_ENDQP(bytes-1)|MCF_QSPI_QWR_CSIV;
      else
          MCF_QSPI_QWR = MCF_QSPI_QWR_ENDQP(bytes-1);
      MCF_QSPI_QDLYR = MCF_QSPI_QDLYR_SPE;// Start Xfer
      while( !(MCF_QSPI_QIR & MCF_QSPI_QIR_SPIF ))
      {
                                                  // Spin here waiting for completion
      };
}
```

This function configures the QWR register, starts the transfer of the QSPI, and polls the SPIF flag for the end status.

8.5 IIC Driver

The main IIC driver function are IIC_send_data( ) and IIC_send_command( ). They provide the transfer function for data and command.

8.5.1 init_IIC( )

```
void init_IIC(UINT8 addr)
{
      UINT8 temp;
      MCF_GPIO_DDRAS = 0;
      /* Enable the IIC signals */
      MCF_GPIO_PASPAR |= ( MCF_GPIO_PASPAR_SDA_SDA
               | MCF_GPIO_PASPAR_SCL_SCL);
      IIC_set_bps(100000ul);// 100k
      /* start the module */
      MCF_IIC_IICR = MCF_IIC_IICR_IEN | 0;
      /* set slave address */
      MCF_IIC_IIAR = addr;
      /* if bit busy is set, the function sends a stop condition to slave module */
      if( MCF_IIC_IISR & MCF_IIC_IISR_IBB)
      {
           MCF_IIC_IICR = 0;                                  /* clear control register */
           MCF_IIC_IICR = MCF_IIC_IICR_IEN |/* enable module */
                MCF_IIC_IICR_MSTA;/* send a START flag */
           temp = MCF_IIC_IIDR;                                        /* dummy read */
           MCF_IIC_IISR = 0;                                  /* clear status register */
           MCF_IIC_IICR = 0;                                  /* clear control register */
           MCF_IIC_IICR = MCF_IIC_IICR_IEN | 0;/* enable the module again */
      }
      IIC_send_command(CLEAR_DISPLAY,IIC_LCD_ADDRESS, 0, NULL); //clear IIC LCD
      return;
}
```

This function initializes the IIC module of the MCF5223x. It sets the SDA and SCL pin for the IIC function. It also sets the baud rate for the transfer. Then, it enables the IIC module and checks the busy flag. If the flag is busy it makes a dummy transfer to free the module. Finally, the software clears the LCD module by sending the clear command.

8.5.2 IIC_set_bps( )

```
UINT8 IIC_set_bps(unsigned int bps)
{
      UINT8 x;
      UINT8 best_ndx=(UINT8)-1u;
      unsigned short e=(unsigned short)-1u;
      unsigned int d=(IIC_CLK*1000ul)/bps;
      for(x=0; x<sizeof(IIC_prescaler_val)/sizeof(IIC_prescaler_val[0]); x++)
      {
           unsigned short e1;
           if (d>IIC_prescaler_val[x])
           {
                continue;
           }
           e1=(unsigned short)(IIC_prescaler_val[x]-d);
           if (e1<e)
           {
                e=e1;
                best_ndx=x;
           }
      }
```

```
    if (best_ndx == (UINT8)-1u)
    {
        return(1);
    }
    MCF_IIC_IIFDR = best_ndx;
    return(0);
}
```

The IIC_set_bps( ) sets the IIFDR register to acquire transfer speed for the IIC module. It goes through the programmable prescaler and calculates to find the baud rate equal to or less than the required one.

8.5.3 IIC_send_command( ) and IIC_send_data( )

```
void IIC_send_command(UINT8 command,UINT8 address, UINT8 number, UINT8* buf)
{
    UINT8 i;
    while(MCF_IIC_IISR & MCF_IIC_IISR_IBB);             /* wait till IIC is busy. */
    MCF_IIC_IICR |= MCF_IIC_IICR_MTX;/* setting in Tx mode */
    /* generates start condition */
    MCF_IIC_IICR |= MCF_IIC_IICR_MSTA;
    MCF_IIC_IIDR = (UINT8)address;                      /* set device ID to write */
    while( !(MCF_IIC_IISR & MCF_IIC_IISR_IIF ))
    ;                                  /* wait until one byte transfer completion */
    MCF_IIC_IISR &= ~MCF_IIC_IISR_IIF;      /* clear the completion transfer flag */

    MCF_IIC_IIDR = (UINT8)254;                                          /* command  */
    iic_delay();/* wait until one byte transfer completion */
    while( !(MCF_IIC_IISR & MCF_IIC_IISR_IIF ));
    MCF_IIC_IISR &= ~MCF_IIC_IISR_IIF;      /* clear the completion transfer flag */
    MCF_IIC_IIDR = (UINT8)command;                                       /* command */
    iic_delay();/* wait until one byte transfer completion */
    while( !(MCF_IIC_IISR & MCF_IIC_IISR_IIF ));
    MCF_IIC_IISR &= ~MCF_IIC_IISR_IIF;      /* clear the completion transfer flag */
    for(i=0;i<number;i++)
    {
        if(buf == NULL)break;
        MCF_IIC_IIDR = buf[i];
        iic_delay();
        while( !(MCF_IIC_IISR & MCF_IIC_IISR_IIF ))
        ;
        MCF_IIC_IISR &= ~MCF_IIC_IISR_IIF;/* clear the completion transfer flag */
    }
    /* generates stop condition */
    MCF_IIC_IICR &= ~MCF_IIC_IICR_MSTA;
    iic_delay();
    return;
}
```

The IIC_send_command( ) is used to send the control command to the IIC LCD module. The input parameters are: the command, the IIC LCD address, the buffer that contains command parameters and the buffer length. This function follows the routine of the LCD module, therefore it is module specified. It generates the START flag to start the IIC frame, then transfers the IIC address, command, and parameters. Finally, it generates a STOP flag.

The routine of the IIC_send_data( ) is almost the same as that of the IIC_send_command( ). The difference is, it sends the buffer content instead of the command.

9 Conclusion

For embedded control application, the LCD display can help enhance the UI usage. The ColdFire V2 product MCF5223x without a flex bus can conveniently adopt the LCD module through a serial interface. The driver introduced in this application note runs on the Interniche RTOS. It does not take much for the user to migrate to any other software platform or OS with the same task routine.

How to Reach Us:

Home Page:
www.freescale.com

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

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
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 Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3559
Rev. 0
03/2008

freescale™
semiconductor