**A. Overview**

Some routines to work with the LCD displays using the PCF8574T port extender on the I2C bus are briefly reviewed. These routines work with the STM32CubeMX w FreeRTOS.

Some goals--

- routines that allowed reasonably easy formatting of messages based on ‘printf (including floating point)

- buffering and semaphore arrangement that allows composing and sending a msg to the display without stalling the originating task.

- multiple displays on the same I2C bus. The scheme naturally allows multiple I2C buses.

**B. Routines**

1. STM32CubeMX setup

For an example see GEVCUr.ioc, in GliderWinchItems/GEVCUr.

Note: The SCL and SDA lines require pullup resistors. The value is not critical, so something in the 1.5K – 10K range should be work. It depends on the capacitance of the cabling to the display. Small values add more to the processor pin current load. I used 3.3K.

2. lcd\_hd44780\_i2c.[ch]

This is the level above the STM32CubeMX I2C routine.

It deals with the initialization of the LCD display via the I2C bus.

Sends commands via HAL i2c driver to the PCF8574T which outputs bits to manipulate the hd44780 LCD controller on the LCD display. The HAL i2c driver is setup to use DMA, (though the number of bytes sent with each call is only 8 bytes, so there I doubt there is much gained, but that is how the original version before hacking was setup).

3. LcdTask.[ch]

This is a FreeRTOS task that waits for pointers on a queue. The pointers point to a buffer control block--

- Points to location for control block that has the I2C bus handle (i.e. there can be multiple I2C buses)

- Number of bytes to be sent to the display

- Line number on the display where the data loading begins

- Column number on the line where the data loading begins.

- Twenty char buffer (max for these types of LCD displays

LcdTask takes the item from the queue and calls the lower level routine to, first, set the row and column, then a call that sends the data. When these calls complete it takes, or waits, for the next pointer to be queued.

The setup for this routine involves,

- Creation of the task. In ‘main’ before the FreeRTOS kernel is started, call--

xLcdTaskCreate(task\_priority, number\_buffers);

This creates the task and queue of “number\_buffers” for struct LCDTASK\_LINEBUF\* pointers.

It also creates a buffer of “number\_buffers” buffer blocks. The lcdi2dprintf and lcdi2cputs (put string) routines, included in LCDTask, add lines to be displayed to this list of buffers. The ‘printf and ‘puts routines are called during the execution from a different task—LcdmsgsetTask.

- When the FreeRTOS kernel starts, StartLcdTask begins and before the infinite loop is started “instantiates” each LCD display. This allocates a block that holds the I2C bus handle, and I2C bus address of the display, as well as number of rows, columns, and number of characters in a row. A call to each LCD unit is required. A linked list is created of all the displays, however the linked list is only used during the instantiation to check for duplicates.

E.g. the following instantiates a 4x20 display on the I2C1 bus--

punitd4x20 = xLcdTaskcreateunit(&hi2c1,0x27,4,20);

if (punitd4x20 == NULL) morse\_trap(227);

As noted above two routines closely coupled to LCDTask, but called during the execution of LcdmsgsetTask, are--

int lcdi2cprintf(struct LCDI2C\_UNIT\*\* punit, int row, int col, const char \*fmt, ...);

int lcdi2cputs(struct LCDI2C\_UNIT\*\* punit, int row, int col, char\* pchr);

The first argument is a pointer to the LCD unit block. The pointer was returned when the LCD unit was instantiated. The row and column arguments specify where the formatted data loading begins. For ‘printf the usual format specification follows. For ‘puts, a pointer to the characters is given. Both of these routines truncate a ‘printf or ‘puts that would exceed the line buffer (i.e. 20 chars).

Both of these routines place the LCDTASK\_LINEBUF pointer on the queue for LcdTask. LcdTask takes the pointer; sets up the row and column; then, sends the buffer to the LCD unit.

3. LcdmsgsetTask.[ch]

This approach uses a separate task to handle calls to lcdi2cprintf (the routine contained in LcdTask). The reason for this is that the call to vsnprintf within lcdi2cprintf is also protected with a semaphore. A task that uses vsnprintf, e.g. defaultTask using yprintf (monitor uart output), may stall if vsnprintf is busy due to a call from another task, i.e. vsnprintf is not reentrant. Therefore, vsnprintf is protected with a semaphore. So, to avoid a call to lcdi2cprintf from stalling the calling task, e.g. in GevcuStates.c, a pointer to a lcdi2cprintf function is placed on a queue to LcdmsgetTask. This task takes the pointer from the queue and executes a lcdi2cprint which does the vsnprintf call and puts the buffer pointer on the queue for LcdTask. Any stalling due to conflicts with vsnprintf are with LcdmsgsetTask and not the originating task, in this example, GevcuStates.c. An example in GevcuStates.c where message #8 is to be sent. The GevcuStates code sets the pointer to a function, i.e. lcdi2cmsg8, and places it on the queue for LcdmsgsetTask to execute.--

static struct LCDMSGSET lcdi2cfunc;

...

static void lcdi2cmsg8(union LCDSETVAR u){lcdi2cputs(&pbuflcdi2s1, GEVCUTSK,0,"GEVCU\_ARM ");}

…

lcdi2cfunc.ptr = lcdi2cmsg8;

// Place ptr to struct w ptr

if (LcdmsgsetTaskQHandle != NULL)

xQueueSendToBack(LcdmsgsetTaskQHandle, &lcdi2cfunc, 0);

In the foregoing example if lcdi2cprintf was the call, (instead of lcdi2cputs), union LCDSETVAR u could have been used to pass eight bytes of value. The union allows the originating routine to pass two four byte values which could be int, float, etc., or four two byte values, etc. to the function that calls lcdi2cprintf. The union is copied onto the queue so there is no problem with the originating routine changing values before, or during, the displaying.

4. LcdmsgsTask.[ch]

This is a different approach for separating the sending and calling task to avoid stalling a key task waiting for the lcd display. It is based on preset msgs. It isn’t as straight-forward, but it is “out there” though not being used in GEVCUr.

**C. Recapitulation**

1. Initialization

Main, before FreeRTOS kernel starts: Create LCDtask, Create LCDsetmsgTask

LCDTask startup: Instantiate each LCD display, which returns a pointer to its control block.

2. Task originating a msg

In a struct, sets a pointer to function, and loads values into the unit, if needed.

Places the struct on a queue for LcdmsgsetTask.

3. LcdmsgsetTask waits on the queue and executes a function call via the pointer, supplying the union as an argument. Typically, that function is a single statement function that is a lcdi2cprintf, or lcdi2cputs line that has the format statement an maybe values passed in by the union.

The lcdi2cprintf and lcdi2cputs calls place the formatted line on the list of buffers, places the buffer pointer pointer on a queue to LCDTask and advances the pointer in the list. Since lcdi2cprintf might stall waiting for some other task to complete vsnprintf use, LcdmsgsetTask is the task that deals with stalling due to others using vsnprintf.

4. LCDTask waits on the queue and takes the buffer pointer, sets the row and column, and sends the line to the lcd\_hd44780\_i2c which does the manipulation via the HAL i2c driver and LCD PCF8574T bus extender on the LCD unit. Upon return from these calls LCDTask returns to check the queue. The sending is not interrupt driven so it is LCDTask that stalls during the sending.

**D. Misc**

1. vsnprintf and semaphores

Originally, vsnprintf was only called from one routine—yprintf. yprintf was designed to handle ‘printf calls for multiple uart outputs. The uart output routine buffered lines to a uart, and there could be multiple uarts. Later, the lcd uart was added and that required adding a line & column specification. The I2C adds the I2c bus address along with multiple displays on one bus. Net, there are multiple places and tasks where vsnprintf is called. vsnprintf is protected by a semphore. Originally, yprintf created this semaphore, then the lcd uart routines piggybacked on that. To avoid the problem of who creates the semaphore, and when, the following is used--

if (vsnprintfSemaphoreHandle == NULL)

vsnprintfSemaphoreHandle = xSemaphoreCreateMutex(); // Semaphore for this buffer

if (vsnprintfSemaphoreHandle == NULL) morse\_trap(38);

2. Polling—poor man’s task

Rather than setup a separate task, such as done with LcdmsgsetTask for lcd i2c displays, the lcd uart display ‘printf statement executions was relegated to defaultTask where there is a periodic call to lcd\_poll. lcd\_poll merely checks a queue, and if not empty, takes the queue item which is a pointer and (if not NULL) does a call on that pointer. The pointer then points to a call for lcdprintf, i.e. the uart lcd display. Obviously, if the queue is empty, the quickest a call to send something to the lcd uart depends on the loop cycle of the defaultTask (which is about 50 ms).

Since lcd\_poll merely does a function call to the pointer that is passed on the queue, other tasks could setup lcd i2c output as well as lcd uart output, but LcdmsgsetTask does this and doesn’t depend on the defaultTask loop timing. Furthermore, it provides a means of passing variable to the ‘printf.