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 calling 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.

1. 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 ascii to the LCD display.

2. 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).

- A semaphore handle that shows that the sending of the buffer is not complete.

- Pointer to the ascii database.

- 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.

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,

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

xLcdTaskCreate

This creates the task and queue for “struct LCDTASK\_LINEBUF”.

- Second, each display needs to be “instantiated”.

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 is required for each display and a linked list is created of all the displays.

This is done when LcdTask starts, e.g. the following intantiates a 4x20 display on the I2C1 bus--

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

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

- Third: Obtaining buffers to be used with the display.

XlcdTaskintgetbuf

This creates a block and a buffer. The block has a pointer to the block with the unit created the first step, plus row, column, max size, buffer semaphore, and pointer to the ascii buffer. The call returns a pointer to the block and this pointer is used in subsequent calls.

There might be a number of these ‘getbuf. E.g, the second of two calls in quick succession might find the first buffer not sent. By using a second buffer, that buffer will be queued and the calling task would not stall.

The calls to setup these buffers is usually done before the “for ( ;; )” in the FreeRTOS task that wants to output to the display. The calls cannot be made until ‘createunit call has been made. A way to handle this is, e.g. the following is executed when GevcuStates.c task starts--

uint8\_t loopctr = 0;

while ((punitd4x20 == NULL) && (loopctr++ < 10)) osDelay(10);

if (punitd4x20 == NULL) morse\_trap(2326);

if (pbuflcdi2s1 == NULL)

pbuflcdi2s1 = xLcdTaskintgetbuf(punitd4x20, 32);

if (pbuflcdi2s1 == NULL) morse\_trap(82);

The ‘while’ makes sure the display unit instantiation is complete. The ‘getbuf obtains a buffer (if for some reason “someone else” didn’t get that buffer.

- Fourth: Formatting and sending data to a display

The following two routines in included LcdTask--

int lcdi2cprintf(struct LCDTASK\_LINEBUF\*\* pplb, int row, int col, const char \*fmt, ...);

int lcdi2cputs(struct LCDTASK\_LINEBUF\*\* pplb, int row, int col, char\* pchr);

The first argument is a pointer to the buffer pointer to be used. If the buffer has its semaphore taken the task will stall until the buffer have been sent and is released. 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 place the LCDTASK\_LINEBUF pointer on the queue for LcdTask.

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. 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, e.g. 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, union LCDSETVAR u could have been used to pass a value.

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. 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.