

RTOS Assignment 4: 7-Segment Display

Dan Blanchette

January 30 2023

1 Program Tasks and Functions

1.1 Tasks

1.1.1 void task right disp: Priority 4

This task utilizes a binary semaphore to protect the 7-segment displays GPIO pins for drawing the one's digits. The xQueueDisp queue is read using xQueuePeek() to service the drawing of the digits to the right segment. A modulo ten is used to isolate the remainder from the count, then update the display. The semaphore is then given up by the task so that the next task (task left disp) can run its routine.

1.1.2 void task left disp: Priority 4

This task utilizes a binary semaphore to protect the 7-segment displays GPIO pins for drawing the one's digits. The xQueueDisp queue is read using xQueuePeek() to service the drawing of the digits to the left segment. A division by ten is used to reduce the integer from a power of 10 to a single digit, then update the display. The semaphore is then given up by the task so that the next task (task right disp) can run its routine. The task (task right display) trades off and helps sync the display, thanks to a binary semaphore. This allows the persistence of vision to be achieved and renders the display as both digits are turned on.

1.1.3 void task count: Priority 3

For this task, two variables are initialized (val to send and bin val to send). The routine relies on two for loops. The initial loop counts down from 42, while the other counts up once the iterator (i) has been decremented to 0. This causes the loop to see-saw when the scheduler services the task. In both loops, the xQueueSend function passes the value of the iterator to two tasks (task right disp and task left disp). Once the value is sent, there is a task delay of 500 ms based on the port tick value, and then xQueueReceive clears the queue. A binary value is then sent to task pico blink.

1.1.4 void task pico blink: Priority 2

In this task val is initialized to 0 as it will be used to reference the receiving binary value from the queue in task count. the logic is pretty simple for this function if the variable(queue val) is equal to one, then run the blink by turning the light on and off with a very small 100ms task delay before it is shut off. Any other input will result in the D13 LED being switched off. This allows for the xQueueSend in the count to utilize its 500ms per port tick rate delay to sync the two tasks.

1.2 Functions

1.2.1 void setup 7seg()

This function has the code to set up the GPIO pin initialization for the seven-segment display.

1.2.2 void numbers(const int)

In this function, I set up the GPIO pins to display the digit patterns on the seven-segment display. The implementation of this function uses a switch statement to allow for ease of access for each digit by passing an integer value as a function parameter.

2 Block Diagram

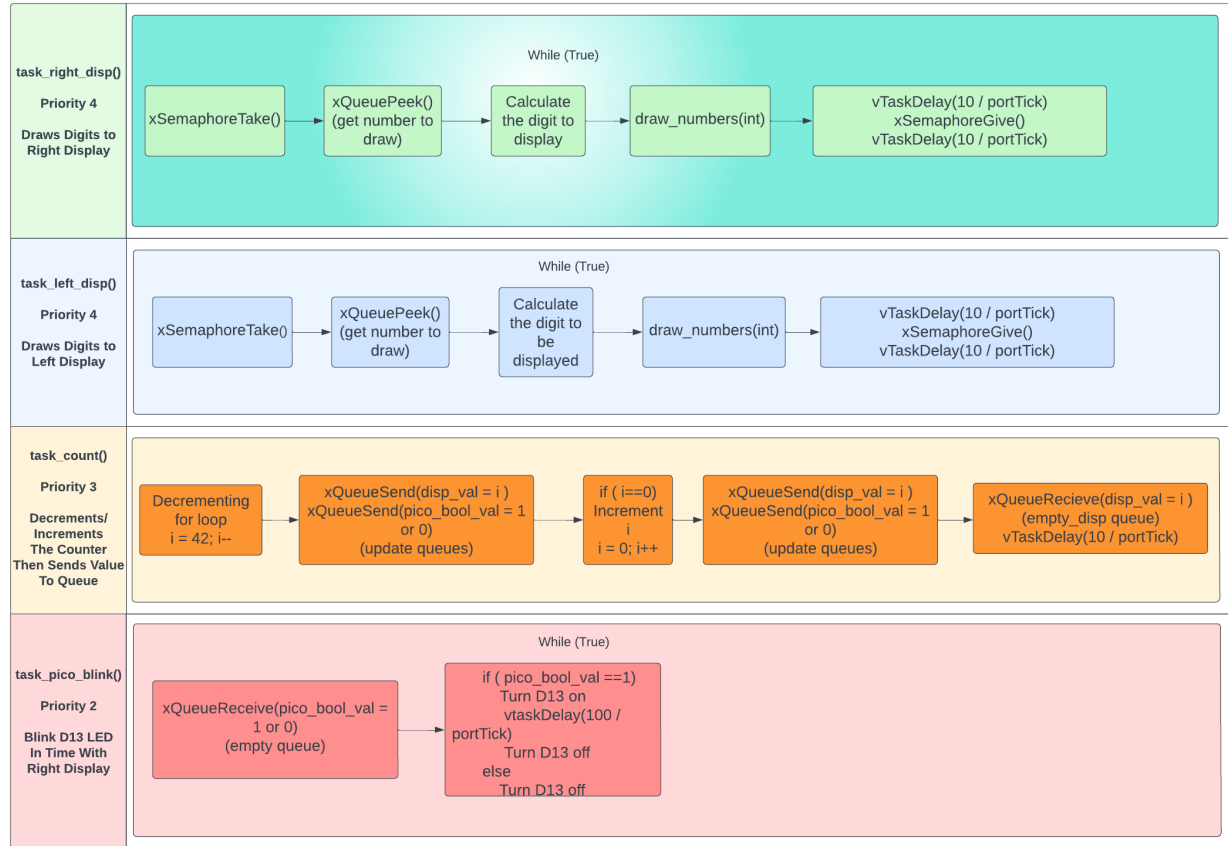


Figure 1: block diagram

3 Code

```
1  /**
2  *
3  * @file main.c
4  * @author Dan Blanchette
5  * @brief This program will count down from 42 to 00 and back up
6  *        from 00 to 42 using a 7-segment display.
7  *        As an additional feature, the D13 LED on the Pico Feather
8  *        will be synced with the second count.
9  * @version 0.1
10 * @date Started: 2023-02-3, Due: 2023-02-8
11 * Total Hours: 30 (coding and research)
12 * Credit: James Lasso (Pair Programming Partner)
13 * @copyright Copyright (c) 2023
14 */
15 #include "main.h"
16 #include "semphr.h"
17 #include <queue.h>
18
19 /*PICO PIN SETUP*/
20 // D13 Pin Assignment pico
21 const uint LED_PIN = PICO_DEFAULT_LED_PIN;
22
23 /*SEMAPHORES*/
24 // Semaphore initialization
25 SemaphoreHandle_t xSem;
26
27 /*****
28 *
29 * QUEUES
30 *
31 *****/
32 // digit queues for the 7-seg display
33 static QueueHandle_t xQueueDisp = NULL;
34 // pico D13 Blink
35 static QueueHandle_t xQueuePico = NULL;
36
37 /*SEVEN SEGMENT PIN ASSIGNMENT*/
38 /*****
39 *
40 * SEVEN SEGMENT PIN ASSIGNMENT *
41 *
42 *****/
43 // GPIO pin setup
44 #define SevenSegCC1 11
45 #define SevenSegCC2 10
46
47 #define SevenSegA 26
48 #define SevenSegB 27
49 #define SevenSegC 29
50 #define SevenSegD 18
51 #define SevenSegE 25
52 #define SevenSegF 7
53 #define SevenSegG 28
54 #define SevenSegDP 24
```

```

54
55
56
57 /*****
58 *
59 * FUNCTION PROTOTYPES *
60 *
61 *****/
62 // setup 7-seg I/O
63 void setup_7seg();
64 // function to draw the numbers on the 7-seg display
65 void draw_numbers(const int );
66
67
68 /*****
69 *
70 * TASKS
71 *
72 *****/
73 // Starts at draws the 0-9 count, 9-0 count on the right display
74 // PRIORITY: 5
75 void task_right_disp()
76 {
77
78     while (1)
79     {
80         // Take the semaphore
81         xSemaphoreTake(xSem, portMAX_DELAY);
82         //printf("In right display\n");
83         gpio_put(SevenSegCC1, 1);
84         gpio_put(SevenSegCC2, 0);
85         int rec_val = 0;
86         xQueuePeek(xQueueDisp, &rec_val, portMAX_DELAY);
87         int right_num = (rec_val % 10);
88         // draw the numbers to the display
89         draw_numbers(right_num);
90
91         vTaskDelay(10 / portTICK_PERIOD_MS);
92         xSemaphoreGive(xSem);
93         vTaskDelay(10 / portTICK_PERIOD_MS);
94
95     }
96 }
97
98 // counts 0-4 then 4-0
99 // PRIORITY: 5
100 void task_left_disp()
101 {
102     while (1)
103     {
104         xSemaphoreTake(xSem, portMAX_DELAY);
105         //printf("In left display\n");
106         gpio_put(SevenSegCC1, 0);
107         gpio_put(SevenSegCC2, 1);
108
109         int rec_val = 0;
110

```

```

111 // extract the digit in the buffer
112 xQueuePeek(xQueueDisp, &rec_val, portMAX_DELAY);
113 // use the remainder of the division for the left side's count
114 int left_num = (rec_val / 10);
115 // draw the number to display
116 draw_numbers(left_num);
117
118 vTaskDelay(10 / portTICK_PERIOD_MS);
119 xSemaphoreGive(xSem);
120 vTaskDelay(10 / portTICK_PERIOD_MS);
121 }
122 }
123 // increment the count and pass the values to a queue
124 // PRIORITY: 4
125 void task_count()
126 {
127     int val_to_send;
128     int bin_val_to_send = 1;
129     // decrement counter
130     for (int i = 42; i >= 0; i--)
131     {
132
133         //printf("decrementing %d\n", i);
134
135         val_to_send = i;
136         bin_val_to_send;
137         // send the incremented value to the queue buffer
138         xQueueSend(xQueueDisp, &val_to_send, 0U);
139         xQueueSend(xQueuePico, &bin_val_to_send, 0U);
140         vTaskDelay(500 / portTICK_PERIOD_MS);
141
142         xQueueReceive(xQueueDisp, &val_to_send, 0U);
143         // the counter is 00
144         if (i == 0)
145         {
146             // increment the count
147             for (i; i <= 42; i++)
148             {
149                 // printf("incrementing %d\n", i);
150                 val_to_send = i;
151                 bin_val_to_send;
152                 // send the incremented value to the queue buffer
153                 xQueueSend(xQueueDisp, &val_to_send, 0U);
154                 xQueueSend(xQueuePico, &bin_val_to_send, 0U);
155                 vTaskDelay(500 / portTICK_PERIOD_MS);
156
157                 // clear the queue when done
158                 xQueueReceive(xQueueDisp, &val_to_send, 0U);
159             }
160         }
161     }
162     vTaskDelay(10 / portTICK_PERIOD_MS);
163 }
164
165 // blink pico
166 void task_pico_blink()
167 {

```

```

168 while (1)
169 {
170     int val = 0;
171
172     xQueueReceive(xQueuePico, &val, portMAX_DELAY);
173     // printf("what is %d\n", val);
174     int que_val = val;
175     // printf("the value %d\n", que_val);
176
177     if (que_val == 1)
178     {
179         gpio_put(LED_PIN, 1);
180         vTaskDelay(100 / portTICK_PERIOD_MS);
181         gpio_put(LED_PIN, 0);
182     }
183     else
184     {
185         gpio_put(LED_PIN, 0);
186     }
187 }
188 }
189
190 int main()
191 {
192     // setup 7-seg GPIO OUT
193     setup_7seg();
194
195     // Init Pico
196     gpio_init(LED_PIN);
197     gpio_set_dir(LED_PIN, GPIO_OUT);
198     // Use for debugging
199     stdio_init_all();
200
201     // creates Queue NOTE: like malloc specify data type
202     xQueueDisp = xQueueCreate(1, sizeof(int));
203     // create Pico Queue (has two values 0 or 1)
204     xQueuePico = xQueueCreate(1, sizeof(int));
205     // NOTE: create binary semaphore before tasks but after init!
206     xSem = xSemaphoreCreateBinary();
207     // take the flag from semaphore
208     xSemaphoreGive(xSem);
209
210     // This first task function's format is meant as a reference
211     // guide for the parameters
212     xTaskCreate(
213         task_right_disp, //function to be called
214         "Task_Right_Dispatch", // Name of Task
215         256, // Stack Size
216         NULL, // Parameter to pass to a function
217         4, // Task Priority (0 to configMAX_PRIORITIES - 1)
218         NULL // Task handle (check on status, watch memory
219         // usage, or end the task)
220     );
221     xTaskCreate(task_left_disp, "Task_Left_Dispatch", 256, NULL, 4, NULL);
222     xTaskCreate(task_count, "Task_Count", 256, NULL, 3, NULL);

```

```

221 xTaskCreate(task_pico_blink, "Task_Pico_Blink", 256, NULL, 2,
      NULL);
222 // tell the scheduler to start running
223 vTaskStartScheduler();
224
225 while (1){}
226 }
227
228
229 void setup_7seg()
230 {
231     // initialize digital pin LED_BUILTIN as an output.
232     gpio_init(SevenSegA);
233     gpio_init(SevenSegB);
234     gpio_init(SevenSegC);
235     gpio_init(SevenSegD);
236     gpio_init(SevenSegE);
237     gpio_init(SevenSegF);
238     gpio_init(SevenSegG);
239     // This GPIO pin activates the decimal point on the 7 segment
      display
240     gpio_init(SevenSegDP);
241
242     gpio_init(SevenSegCC1);
243     gpio_init(SevenSegCC2);
244
245     gpio_set_dir(SevenSegA, GPIO_OUT);
246     gpio_set_dir(SevenSegB, GPIO_OUT);
247     gpio_set_dir(SevenSegC, GPIO_OUT);
248     gpio_set_dir(SevenSegD, GPIO_OUT);
249     gpio_set_dir(SevenSegE, GPIO_OUT);
250     gpio_set_dir(SevenSegF, GPIO_OUT);
251     gpio_set_dir(SevenSegG, GPIO_OUT);
252     gpio_set_dir(SevenSegDP, GPIO_OUT);
253
254     gpio_set_dir(SevenSegCC1, GPIO_OUT);
255     gpio_set_dir(SevenSegCC2, GPIO_OUT);
256 }
257
258 void draw_numbers(const int segNum)
259 {
260     switch (segNum)
261     {
262     case 0:
263         /*
264          A
265
266          F | G | B   | _ _
267          E | _ _ | C   | _ _
268             D
269          */
270         gpio_put(SevenSegA, 1);
271         gpio_put(SevenSegB, 1);
272         gpio_put(SevenSegC, 1);
273         gpio_put(SevenSegD, 1);
274         gpio_put(SevenSegE, 1);
275         gpio_put(SevenSegF, 1);

```



```

276         gpio_put(SevenSegG, 0);
277         break;
278
279     case 1:
280
281         // display #1
282         /*
283          | B
284          | C
285         */
286         gpio_put(SevenSegA, 0);
287         gpio_put(SevenSegB, 1);
288         gpio_put(SevenSegC, 1);
289         gpio_put(SevenSegD, 0);
290         gpio_put(SevenSegE, 0);
291         gpio_put(SevenSegF, 0);
292         gpio_put(SevenSegG, 0);
293         break;
294
295     case 2:
296         // display #2 on the right segment
297         /*
298          --
299          --|
300          |--
301         */
302         gpio_put(SevenSegA, 1);
303         gpio_put(SevenSegB, 1);
304         gpio_put(SevenSegC, 0);
305         gpio_put(SevenSegD, 1);
306         gpio_put(SevenSegE, 1);
307         gpio_put(SevenSegF, 0);
308         gpio_put(SevenSegG, 1);
309         break;
310
311     case 3:
312         // display #3 on the right segment
313         /*
314          --
315          --|
316          --|
317         */
318         gpio_put(SevenSegA, 1);
319         gpio_put(SevenSegB, 1);
320         gpio_put(SevenSegC, 1);
321         gpio_put(SevenSegD, 1);
322         gpio_put(SevenSegE, 0);
323         gpio_put(SevenSegF, 0);
324         gpio_put(SevenSegG, 1);
325         break;
326
327     case 4:
328         // display #4 on the right segment
329         /*
330          |__|
331          |
332         */
333         gpio_put(SevenSegA, 0);
334         gpio_put(SevenSegB, 1);
335         gpio_put(SevenSegC, 1);
336         gpio_put(SevenSegD, 0);

```

```

333     gpio_put(SevenSegE, 0);
334     gpio_put(SevenSegF, 1);
335     gpio_put(SevenSegG, 1);
336     break;
337 case 5:
338     // display #5 on the right segment
339     /*  --
340         |__
341     __|*/
342     gpio_put(SevenSegA, 1);
343     gpio_put(SevenSegB, 0);
344     gpio_put(SevenSegC, 1);
345     gpio_put(SevenSegD, 1);
346     gpio_put(SevenSegE, 0);
347     gpio_put(SevenSegF, 1);
348     gpio_put(SevenSegG, 1);
349     break;
350 case 6:
351     // display #6 on the right segment
352     /*  --
353         |__
354     |__|*/
355     gpio_put(SevenSegA, 1);
356     gpio_put(SevenSegB, 0);
357     gpio_put(SevenSegC, 1);
358     gpio_put(SevenSegD, 1);
359     gpio_put(SevenSegE, 1);
360     gpio_put(SevenSegF, 1);
361     gpio_put(SevenSegG, 1);
362     break;
363 case 7:
364     // display #7 on the right segment
365     /*
366         --
367         |
368         |
369     */
370     gpio_put(SevenSegA, 1);
371     gpio_put(SevenSegB, 1);
372     gpio_put(SevenSegC, 1);
373     gpio_put(SevenSegD, 0);
374     gpio_put(SevenSegE, 0);
375     gpio_put(SevenSegF, 0);
376     gpio_put(SevenSegG, 0);
377     break;
378 case 8:
379     // display #8 on the right segment
380     /*  --
381         |__|
382     |__|*/
383     gpio_put(SevenSegA, 1);
384     gpio_put(SevenSegB, 1);
385     gpio_put(SevenSegC, 1);
386     gpio_put(SevenSegD, 1);
387     gpio_put(SevenSegE, 1);
388     gpio_put(SevenSegF, 1);
389     gpio_put(SevenSegG, 1);

```

```

390     break;
391 case 9:
392     // display #9 on the right segment
393     /*
394     |__|
395         |
396     */
397     gpio_put(SevenSegA, 1);
398     gpio_put(SevenSegB, 1);
399     gpio_put(SevenSegC, 1);
400     gpio_put(SevenSegD, 0);
401     gpio_put(SevenSegE, 0);
402     gpio_put(SevenSegF, 1);
403     gpio_put(SevenSegG, 1);
404     break;
405
406 default:
407     // this is for debug purposes
408     printf("Please enter a value between 0-9");
409 }
410 }

```