

## main.c

```
108
109 #include <stdint.h>
110 #include <stdbool.h>
111 #include "main.h"
112 #include "drivers/pinout.h"
113 #include "utils/uartstdio.h"
114
115
116 // TivaWare includes
117 #include "driverlib/sysctl.h"
118 #include "driverlib/debug.h"
119 #include "driverlib/rom_map.h"
120 #include "driverlib/rom.h"
121 #include "driverlib/timer.h"
122 #include "driverlib/inc/hw_memmap.h"
123 #include "driverlib/inc/hw_ints.h"
124
125 // FreeRTOS includes
126 #include "FreeRTOSConfig.h"
127 #include "FreeRTOS.h"
128 #include <timers.h>
129 #include <semphr.h>
130 #include "task.h"
131 #include "queue.h"
132 #include "limits.h"
133
134
135 #define FIB_LIMIT_FOR_32_BIT 47
136 #define TIME_TO_RUN 240 //ms
137
138 SemaphoreHandle_t task1SyncSemaphore, task2SyncSemaphore;
139 TickType_t startTimeTick;
140 TaskHandle_t Task1_handle, Task2_handle;
141 uint32_t counter = 0;
142 double Hz = 100;
143 uint32_t ulPeriod;
144
145
146 void fiboncacci(int ms){
147     TickType_t xStartTick = xTaskGetTickCount();
148     TickType_t xCurrentTick = xTaskGetTickCount();
149     uint32_t fib = 1, fib_a = 1, fib_b = 1;
150     uint32_t i;
151     while((xCurrentTick - xStartTick) < (pdMS_TO_TICKS(ms) - 1)){
152         for (i = 0; i < FIB_LIMIT_FOR_32_BIT; i++){
153             if(((xCurrentTick - xStartTick) >= pdMS_TO_TICKS(ms) - 1)) break;
154             fib_a = fib_b;
155             fib_b = fib;
156             fib = fib_a + fib_b;
157         }
158         xCurrentTick = xTaskGetTickCount();
159     }
160 }
```

```
161
162
163
164
165 void Timer0Isr(void)
166 {
167     TickType_t xCurrentTick = xTaskGetTickCount();
168     BaseType_t xHigherPriorityTaskWoken = pdFALSE;
169     ROM_TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); // Clear the timer interrupt
170     counter ++;
171     if (counter % 3 == 0){
172         xTaskNotifyFromISR(Task1_handle, xCurrentTick, eSetValueWithOverwrite, &
xHigherPriorityTaskWoken);
173         portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
174     }
175     else if(counter % 8 == 0){
176         xTaskNotifyFromISR(Task2_handle, xCurrentTick, eSetValueWithOverwrite, &
xHigherPriorityTaskWoken);
177         portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
178         counter = 0;
179     }
180 }
181
182
183
184 // Process 1
185 void xTask1(void * pvParameters)
186 {
187     TickType_t xLastWakeTime;
188     xLastWakeTime = xTaskGetTickCount();
189     const TickType_t xMaxBlockTime = pdMS_TO_TICKS( 5000 );
190     BaseType_t xResult;
191     uint32_t ulNotifiedValue;
192
193     while((xLastWakeTime - startTimeTick) < TIME_TO_RUN){
194
195         xResult = xTaskNotifyWait( pdFALSE,
196                                     /* Don't clear bits on entry. */
197                                     ULONG_MAX,
198                                     /* Clear all bits on exit. */
199                                     &ulNotifiedValue, /* Stores the notified value. */
200                                     xMaxBlockTime );
201
202         if( xResult == pdPASS )
203         {
204             // xSemaphoreTake(task1SyncSemaphore, xMaxBlockTime);
205             TickType_t xCurrentTick = xTaskGetTickCount();
206             UARTprintf("Task 1 started at %d ms and Timer interrupt data: %d\n",
xCurrentTick, ulNotifiedValue);
207             fiboncacci(10);
208             TickType_t xFibTime = xTaskGetTickCount();
209             UARTprintf("Task 1 completed at %d ms (Execution: %d ms)\n",
xFibTime, (xFibTime - xCurrentTick));
210             xLastWakeTime = xCurrentTick;
211             // xSemaphoreGive(task1SyncSemaphore);
212         }
213     }
214 }
```

```
215 }
216
217
218
219 // Process 2
220 void xTask2(void *pvParameters)
221 {
222     TickType_t xLastWakeTime;
223     xLastWakeTime = xTaskGetTickCount();
224     const TickType_t xMaxBlockTime = pdMS_TO_TICKS( 5000 );
225     BaseType_t xResult;
226     uint32_t ulNotifiedValue;
227
228     while ((xLastWakeTime - startTimeTick) < TIME_TO_RUN)
229     {
230
231         xResult = xTaskNotifyWait( pdFALSE,
232                                     /* Don't clear bits on entry. */
233                                     ULONG_MAX,
234                                     /* Clear all bits on exit. */
235                                     &ulNotifiedValue, /* Stores the notified value. */
236                                     xMaxBlockTime );
237
238         if( xResult == pdPASS )
239         {
240             // xSemaphoreTake(task1SyncSemaphore, xMaxBlockTime);
241             TickType_t xCurrentTick = xTaskGetTickCount();
242             UARTprintf("Task 2 started at %d ms (Preempted Task 1) and Timer
interrupt data %d\n", xCurrentTick, ulNotifiedValue);
243             fiboncacci(40);
244             TickType_t xFibTime = xTaskGetTickCount();
245             UARTprintf("Task 2 completed at %d ms (Execution: %d ms)\n",
246                         xFibTime, (xFibTime - xCurrentTick));
247             xLastWakeTime = xCurrentTick;
248             // xSemaphoreGive(task1SyncSemaphore);
249         }
250     }
251 }
252
253
254
255 // Main function
256 int main(void)
257 {
258     // Initialize system clock to 120 MHz
259     uint32_t output_clock_rate_hz;
260     output_clock_rate_hz = ROM_SysCtlClockFreqSet(
261         (SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
262          SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480),
263         SYSTEM_CLOCK);
264     ASSERT(output_clock_rate_hz == SYSTEM_CLOCK);
265
266
267     // Initialize the GPIO pins for the Launchpad
268     PinoutSet(false, false);
269     UARTStdioConfig(0, 230400, SYSTEM_CLOCK);
```

```
270
271     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
272     ROM_TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);    // 32 bits Timer
273     TimerIntRegister(TIMER0_BASE, TIMER_A, Timer0Isr);    // Registering isr
274
275
276     ulPeriod = (SYSTEM_CLOCK / Hz);
277     ROM_TimerLoadSet(TIMER0_BASE, TIMER_A, ulPeriod -1);
278
279     ROM_TimerEnable(TIMER0_BASE, TIMER_A);
280     ROM_IntEnable(INT_TIMER0A);
281     ROM_TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
282
283     task1SyncSemaphore = xSemaphoreCreateBinary();
284     task2SyncSemaphore = xSemaphoreCreateBinary();
285
286
287     xTaskCreate(xTask1, "Task1", configMINIMAL_STACK_SIZE, NULL, 2, &Task1_handle);
288     xTaskCreate(xTask2, "Task2", configMINIMAL_STACK_SIZE, NULL, 1, &Task2_handle);
289     startTimeTick = xTaskGetTickCount();
290     xTaskNotifyFromISR(Task1_handle, startTimeTick, eSetValueWithOverwrite, NULL);
291     xTaskNotifyFromISR(Task2_handle, startTimeTick, eSetValueWithOverwrite, NULL);
292
293     vTaskStartScheduler();
294
295     return (0);
296 }
297
298
299 /* ASSERT() Error function
300 *
301 * failed ASSERTS() from driverlib/debug.h are executed in this function
302 */
303 void __error__(char *pcFilename, uint32_t ui32Line)
304 {
305     // Place a breakpoint here to capture errors until logging routine is finished
306     while (1)
307     {
308     }
309 }
```