main.c

```c
 1  #include <stdio.h>
 2  #include <stdint.h>
 3  #include <stdbool.h>
 4  #include "main.h"
 5  #include "drivers/pinout.h"
 6  #include "utils/uartstdio.h"
 7
 8  // TivaWare includes
 9  #include "driverlib/sysctl.h"
10  #include "driverlib/debug.h"
11  #include "driverlib/rom_map.h"
12  #include "driverlib/rom.h"
13  #include "driverlib/timer.h"
14  #include "driverlib/inc/hw_memmap.h"
15  #include "driverlib/inc/hw_ints.h"
16
17  // FreeRTOS includes
18  #include "FreeRTOSConfig.h"
19  #include "FreeRTOS.h"
20  #include <timers.h>
21  #include <semphr.h>
22  #include "task.h"
23  #include "queue.h"
24  #include "limits.h"
25
26  #define FIB_LIMIT_FOR_32_BIT 47
27  #define ITERATION 1200
28  #define MULTIPLIER 10
29  #define Hz (30 * MULTIPLIER)          // Hz
30  #define SEQUENCER_COUNT 900
31  #define UART_BAUD_RATE 1000000
32
33  SemaphoreHandle_t task_1_SyncSemaphore, task_2_SyncSemaphore, task_3_SyncSemaphore,
    task_4_SyncSemaphore, task_5_SyncSemaphore, task_6_SyncSemaphore,
    task_7_SyncSemaphore;
34  TickType_t startTimeTick;
35  TaskHandle_t Task1_handle, Task2_handle, Task3_handle, Task4_handle, Task5_handle,
    Task6_handTe, Task7_handle;
36  uint32_t counter_isr = 0;
37  uint32_t ulPeriod;
38  volatile bool abort_test = false;
39  uint32_t wcet[7];
40  uint32_t execution_time[7];
41  uint32_t execution_cycle[7];
42
43  void init_Timer();
44  void init_Uart();
45  void init_Clock();
46
47  void fibonacci()
48  {
49      uint32_t i,j;
50      uint32_t fib = 1, fib_a = 1, fib_b = 1;
51      for ( i=0; i<ITERATION; i++)
```

```c
52      {
53          for(j=0; j<FIB_LIMIT_FOR_32_BIT; j++){
54              fib_a = fib_b;
55              fib_b = fib;
56              fib = fib_a + fib_b;
57          }
58
59      }
60  }
61
62
63  void print_data(){
64      uint32_t i = 0;
65      for (i = 0; i < 7; i++){
66          UARTprintf("***** Task %d wcet %d total_exectution_time %d execution unit %d
    *****\n\r", i+1, wcet[i], execution_time[i], execution_cycle[i]);
67      }
68  }
69
70  void Timer0Isr_Sequencer(void)
71  {
72      TickType_t xCurrentTick = xTaskGetTickCount();
73      ROM_TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); // Clear the timer interrupt
74      counter_isr++;
75
76      UARTprintf("Sequencer Thread ran at %d ms and Cycle of sequencer %d \n\r",
    xCurrentTick, counter_isr);
77
78      if ((counter_isr % 10) == 0)
79      {
80          // Service_1 = RT_MAX-1 @ 300 Hz
81          xSemaphoreGive(task_1_SyncSemaphore); // Frame Sampler thread
82      }
83
84      if ((counter_isr % 30) == 0)
85      {
86          // Service_2 = RT_MAX-2 @ 100 Hz
87          xSemaphoreGive(task_2_SyncSemaphore); // Time-stamp with Image Analysis
    thread
88          // Service_4 = RT_MAX-2 @ 100 Hz
89          xSemaphoreGive(task_4_SyncSemaphore); // Time-stamp Image Save to File thread
90          // Service_6 = RT_MAX-2 @ 100 Hz
91          xSemaphoreGive(task_6_SyncSemaphore); // Send Time-stamped Image to Remote
    thread
92
93      }
94
95      if ((counter_isr % 60) == 0)
96      {
97          // Service_3 = RT_MAX-3 @ 50 Hz
98          xSemaphoreGive(task_3_SyncSemaphore); // Difference Image Proc thread
99          // Service_5 = RT_MAX-3 @ 50 Hz
100         xSemaphoreGive(task_5_SyncSemaphore); // Processed Image Save to File thread
101     }
102
103
104     if ((counter_isr % 300) == 0)
```

```c
105      {
106          // Service_7 = RT_MIN    10 Hz
107          xSemaphoreGive(task_7_SyncSemaphore); // 10 sec Tick Debug thread
108      }
109
110      if (counter_isr > SEQUENCER_COUNT)
111      {
112          xSemaphoreGive(task_1_SyncSemaphore); // Frame Sampler thread
113          xSemaphoreGive(task_2_SyncSemaphore); // Time-stamp with Image Analysis
     thread
114          xSemaphoreGive(task_3_SyncSemaphore); // Difference Image Proc thread
115          xSemaphoreGive(task_4_SyncSemaphore); // Time-stamp Image Save to File thread
116          xSemaphoreGive(task_5_SyncSemaphore); // Processed Image Save to File thread
117          xSemaphoreGive(task_6_SyncSemaphore); // Send Time-stamped Image to Remote
     thread
118          xSemaphoreGive(task_7_SyncSemaphore); // 10 sec Tick Debug thread
119          abort_test = true;
120          ROM_TimerDisable(TIMER0_BASE, TIMER_A);
121          print_data();
122      }
123  }
124
125  // Process 1
126  void xTask1(void *pvParameters)
127  {
128      BaseType_t xResult;
129
130      while (!abort_test)
131      {
132
133          xResult = xSemaphoreTake(task_1_SyncSemaphore, portMAX_DELAY);
134
135          if (xResult == pdPASS)
136          {
137              execution_cycle[0]++;
138              TickType_t xCurrentTick = xTaskGetTickCount();
139              UARTprintf("Task 1 (Frame Sampler thread) Start Time:%d ms, Release count
     %d \n\r", xCurrentTick, execution_cycle[0]);
140              fibonacci();
141              TickType_t xFibTime = xTaskGetTickCount();
142              TickType_t total_time = (xFibTime - xCurrentTick);
143              execution_time[0] += total_time;
144              if(wcet[0] < total_time) wcet[0] = total_time;
145
146              UARTprintf("Task 1 (Frame Sampler thread) Completion Time:%d ms,
     Execution Time:%d ms\n\r", xFibTime, total_time);
147          }
148      }
149      vTaskDelete( NULL );
150  }
151
152  void xTask2(void *pvParameters)
153  {
154      BaseType_t xResult;
155
156      while (!abort_test)
157      {
```

```c
158            xResult = xSemaphoreTake(task_2_SyncSemaphore, portMAX_DELAY);
159
160
161        if (xResult == pdPASS)
162        {
163            execution_cycle[1]++;
164            TickType_t xCurrentTick = xTaskGetTickCount();
165            UARTprintf("Task 2 (Time-stamp with Image Analysis thread) Start Time:%d
   ms, Release count %d \n\r", xCurrentTick, execution_cycle[1]);
166            fibonacci();
167            TickType_t xFibTime = xTaskGetTickCount();
168            TickType_t total_time = (xFibTime - xCurrentTick);
169            execution_time[1] += total_time;
170            if(wcet[1] < total_time) wcet[1] = total_time;
171            UARTprintf("Task 2 (Time-stamp with Image Analysis thread) Completion
   Time:%d ms, Execution Time:%d ms\n\r", xFibTime, (xFibTime - xCurrentTick));
172        }
173        }
174        vTaskSuspend( NULL );
175    }
176    void xTask3(void *pvParameters)
177    {
178        BaseType_t xResult;;
179
180        while (!abort_test)
181        {
182
183            xResult = xSemaphoreTake(task_3_SyncSemaphore, portMAX_DELAY);
184
185        if (xResult == pdPASS)
186        {
187            execution_cycle[2]++;
188            TickType_t xCurrentTick = xTaskGetTickCount();
189            UARTprintf("Task 3 (Difference Image Proc thread) Start Time:%d ms,
   Release count %d \n\r", xCurrentTick, execution_cycle[2]);
190            fibonacci();
191            TickType_t xFibTime = xTaskGetTickCount();
192            TickType_t total_time = (xFibTime - xCurrentTick);
193            execution_time[2] += total_time;
194            if(wcet[2] < total_time) wcet[2] = total_time;
195            UARTprintf("Task 3 (Difference Image Proc thread) Completion Time:%d ms,
   Execution Time:%d ms\n\r", xFibTime, (xFibTime - xCurrentTick));
196        }
197        }
198        vTaskDelete( NULL );
199    }
200    void xTask4(void *pvParameters)
201    {
202        BaseType_t xResult;
203
204        while (!abort_test)
205        {
206
207            xResult = xSemaphoreTake(task_4_SyncSemaphore, portMAX_DELAY);
208
209        if (xResult == pdPASS)
210        {
```

```c
211                 execution_cycle[3]++;
212                 TickType_t xCurrentTick = xTaskGetTickCount();
213                 UARTprintf("Task 4 (Time-stamp Image Save to File thread) Start Time:%d
    ms, Release count %d \n\r", xCurrentTick, execution_cycle[3]);
214                 fibonacci();
215                 TickType_t xFibTime = xTaskGetTickCount();
216                 TickType_t total_time = (xFibTime - xCurrentTick);
217                 execution_time[3] += total_time;
218                 if(wcet[3] < total_time) wcet[3] = total_time;
219                 UARTprintf("Task 4 (Time-stamp Image Save to File thread) Completion
    Time:%d ms, Execution Timet:%d ms\n\r", xFibTime, (xFibTime - xCurrentTick));
220             }
221         }
222         vTaskDelete( NULL );
223 }
224 void xTask5(void *pvParameters)
225 {
226     BaseType_t xResult;
227
228     while (!abort_test)
229     {
230
231         xResult = xSemaphoreTake(task_5_SyncSemaphore, portMAX_DELAY);
232
233         if (xResult == pdPASS)
234         {
235             execution_cycle[4]++;
236             TickType_t xCurrentTick = xTaskGetTickCount();
237             UARTprintf("Task 5 (Processed Image Save to File thread) Start Time:%d
    ms, Release count %d \n\r", xCurrentTick, execution_cycle[4]);
238             fibonacci();
239             TickType_t xFibTime = xTaskGetTickCount();
240             TickType_t total_time = (xFibTime - xCurrentTick);
241             execution_time[4] += total_time;
242             if(wcet[4] < total_time) wcet[4] = total_time;
243             UARTprintf("Task 5 (Processed Image Save to File thread) Completion
    Time:%d ms, Execution Time:%d ms\n\r", xFibTime, (xFibTime - xCurrentTick));
244         }
245     }
246     vTaskDelete( NULL );
247 }
248 void xTask6(void *pvParameters)
249 {
250     BaseType_t xResult;
251
252     while (!abort_test)
253     {
254
255         xResult = xSemaphoreTake(task_6_SyncSemaphore, portMAX_DELAY);
256
257         if (xResult == pdPASS)
258         {
259             execution_cycle[5]++;
260             TickType_t xCurrentTick = xTaskGetTickCount();
261             UARTprintf("Task 6 (Send Time-stamped Image to Remote thread) Start
    Time:%d ms, Release count %d \n\r", xCurrentTick, execution_cycle[5]);
262             fibonacci();
```

```
263             TickType_t xFibTime = xTaskGetTickCount();
264             TickType_t total_time = (xFibTime - xCurrentTick);
265             execution_time[5] += total_time;
266             if(wcet[5] < total_time) wcet[5] = total_time;
267             UARTprintf("Task 6 (Send Time-stamped Image to Remote thread) Completion
    Time:%d ms, Execution Time:%d ms\n\r", xFibTime, (xFibTime - xCurrentTick));
268         }
269     }
270     vTaskDelete( NULL );
271 }
272 void xTask7(void *pvParameters)
273 {
274     BaseType_t xResult;
275
276     while (!abort_test)
277     {
278
279         xResult = xSemaphoreTake(task_7_SyncSemaphore, portMAX_DELAY);
280
281         if (xResult == pdPASS)
282         {
283             execution_cycle[6]++;
284             TickType_t xCurrentTick = xTaskGetTickCount();
285             UARTprintf("Task 7 (10 sec Tick Debug thread) Start Time:%d ms , Release
    count %d \n\r", xCurrentTick, execution_cycle[6]);
286             fibonacci();
287             TickType_t xFibTime = xTaskGetTickCount();
288             TickType_t total_time = (xFibTime - xCurrentTick);
289             execution_time[6] += total_time;
290             if(wcet[6] < total_time) wcet[6] = total_time;
291             UARTprintf("Task 7 (10 sec Tick Debug thread) Completion Time:%d ms,
    Execution Time:%d ms\n\r", xFibTime, (xFibTime - xCurrentTick));
292         }
293     }
294     vTaskDelete( NULL );
295 }
296
297 // Main function
298 int main(void)
299 {
300     init_Clock();
301     init_Uart();
302     init_Timer();
303
304     task_1_SyncSemaphore = xSemaphoreCreateBinary();
305     task_2_SyncSemaphore = xSemaphoreCreateBinary();
306     task_3_SyncSemaphore = xSemaphoreCreateBinary();
307     task_4_SyncSemaphore = xSemaphoreCreateBinary();
308     task_5_SyncSemaphore = xSemaphoreCreateBinary();
309     task_6_SyncSemaphore = xSemaphoreCreateBinary();
310     task_7_SyncSemaphore = xSemaphoreCreateBinary();
311
312     UARTprintf("Cyclic executer : %d Hz\n\r", Hz);
313     xTaskCreate(xTask1, "Task1", configMINIMAL_STACK_SIZE, NULL, 4, &Task1_handle);
314     xTaskCreate(xTask2, "Task2", configMINIMAL_STACK_SIZE, NULL, 3, &Task2_handle);
315     xTaskCreate(xTask3, "Task3", configMINIMAL_STACK_SIZE, NULL, 2, &Task3_handle);
316     xTaskCreate(xTask4, "Task4", configMINIMAL_STACK_SIZE, NULL, 3, &Task4_handle);
```

```c
        xTaskCreate(xTask5, "Task5", configMINIMAL_STACK_SIZE, NULL, 2, &Task5_handle);
        xTaskCreate(xTask6, "Task6", configMINIMAL_STACK_SIZE, NULL, 3, &Task6_handle);
        xTaskCreate(xTask7, "Task7", configMINIMAL_STACK_SIZE, NULL, 1, &Task7_handle);

        startTimeTick = xTaskGetTickCount();

        vTaskStartScheduler();
        UARTprintf("\nTEST COMPLETE\n");
        return (0);
}

void init_Timer()
{
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
        ROM_TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);          // 32 bits Timer
        TimerIntRegister(TIMER0_BASE, TIMER_A, Timer0Isr_Sequencer); // Registering  isr

        ulPeriod = (SYSTEM_CLOCK / Hz);
        ROM_TimerLoadSet(TIMER0_BASE, TIMER_A, ulPeriod - 1);

        ROM_TimerEnable(TIMER0_BASE, TIMER_A);
        ROM_IntEnable(INT_TIMER0A);
        ROM_TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
}

void init_Clock()
{
        // Initialize system clock to 120 MHz
        uint32_t output_clock_rate_hz;
        output_clock_rate_hz = ROM_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
SYSCTL_OSC_MAIN | SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), SYSTEM_CLOCK);
        ASSERT(output_clock_rate_hz == SYSTEM_CLOCK);
}

void init_Uart()
{
        // Initialize the GPIO pins for the Launchpad
        PinoutSet(false, false);
        UARTStdioConfig(0, UART_BAUD_RATE, SYSTEM_CLOCK);
}

/*  ASSERT() Error function
 *
 *  failed ASSERTS() from driverlib/debug.h are executed in this function
 */
void __error__(char *pcFilename, uint32_t ui32Line)
{
        // Place a breakpoint here to capture errors until logging routine is finished
        while (1)
        {
        }
}
```