OBC R7018R HT2017

Assignment 2: Tasks in FreeRTOS

Background and goal

The goals are :

- To familiarize with the task concept
- To familiarize with the FreeRTOS
- To familiarize with the STK500 development board
- To prepare for the final RTOS system lab

Report

Work in groups of two students.
Report deadline **2017-09-19**

Theory

Study the task.h file available at the FreeRTOS library on your computer or the FreeRTOS website. The **task.h** file gives the prototypes for task handling system calls**.** The file explains all parameters, informs on prerequisites and gives an example on how to use the function in a program. Below some important system calls are listed together with a brief description. The calls are further explained in the **task.h** file:

xTaskCreate Create a new task and add it to the list of tasks that are ready to run

vTaskDelayUntil Delay a task until a specified time. This function can be used by cyclical tasks to ensure a constant execution frequency.

vTaskStartScheduler Starts the real time kernel tick processing. After calling the kernel has control over which tasks are executed and when.

The **semphr.h** file gives the prototypes for semaphore system calls**.** The file explains all parameters, informs on prerequisites and gives an example on how to use the function in a program. If you are using interrupts in your code, note the examples in the file on how to use semaphores when a task is waiting for an interrupt to occur. Below some important system calls are listed together with a brief description. The calls are further explained in the **semphr.h** file:

vSemaphoreCreateBinary Creates a binary semaphore mechanism.

xSemaphoreTake

xSemaphoreGive

xSemaphoreGiveFromISR Releases a semaphore from an ISR (Interrupt service routine).

The FreeRTOS uses MUTEXes and QUEUEs, which has a higher level of abstraction than semaphores, but you do not need to use them, semaphores are OK in this lab.

portmacro.h includes macros for types used in FreeRTOS example:

```
/* Type definitions. */
#define portCHAR            char
#define portFLOAT           float
#define portDOUBLE          double
#define portLONG            long
#define portSHORT           int
#define portSTACK_TYPE      unsigned portCHAR
#define portBASE_TYPE       char
```

Microcontroller specific code for RTOS tick, context switching etc is found in **port.c** (not necessary to study: low level code).

Instructions

You are going to work on the **STK500** development board including an AVR324p processor. If you are not familiar with the processor you can study the data sheet for more information (on Canvas and lab computer). The board will be set to a certain frequency. You must therefore set a configuration parameter to fit the frequency parameter used in the software library to the hardware. Details regarding this will be available in the lab room.

1) Use the test file (on Canvas) to implement the light rope from Assignment 0 (or microcomputer course) by using a periodic task instead of a delay. Port B is connected to the diodes on the board. Rates are set by the buttons on the board. Port D is connected to the buttons. Try with different rates. Run the program.

2) Create two cyclic tasks that toggles one diode each (on the same port) on and off, but with different frequencies.

   The tasks shall be implemented as two independent objects with the only operations initTask1 and initTask2. The header files for the two objects shall include a marcro defining the task period.

   Use a protected object ***diode*** to ensure *mutual exclusion*. The operations shall be initDiodes, and toggleDiode with diode number as a parameter. Implement the protected object using a FreeRTOS mutex.

   You need one main file and two files each (.h and .c) for tasks and for protected objects.

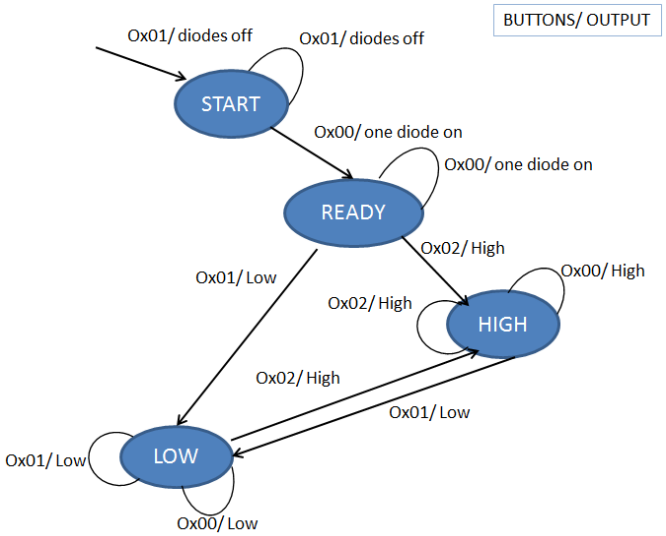
   The main program shall include the header-files for the tasks and the diode and the main code will be:

```
int main(void)

{       initDiode();
        initTask1();
        initTask2();
        vTaskStartScheduler();
        return 0;
}
```

3)

a) Now you shall have only one “lightrope”-task but also include a second *cyclic task* that reads the buttons at a suitable rate (preset by you) and interpret the command from the buttons according to the following state chart, and then “inform” the lightrope task on when to start, when to light the ready diode and when to start the lightrope. The rate may thereafter be set to “High” or “Low”. Note that one task is communicating with the buttons, and the other one (lightrope task) with the diodes. The “lightrope”-task starts with all diodes lit and stops waiting for the first command from the “command” task.



The two tasks shall communicate using *shared memory* that are protected i.e a “protected object” with read and write operations. The communication shall be performed through a mutex. This means that you need two more files for the *communication* between these two tasks

Note that the diode no longer need to be protected for mutual exclusion since it is used by one task only.

The main program shall include the header-files for the two tasks, the diode and the main code will be:

```
int main(void)
{       initDiode( );
        initCom( );
         initLigthrope ( );
         initTC( );
         vTaskStartScheduler();
        return 0;
}
```

Hint: Use a switch-statement for the states of the tasks and a second level of switch (or if-statements) for the commands.

b) You shall now change the second task to a *sporadic task* that only reads the command buttons on an external interrupt event (INT0 or INT1). The hardware for the event-button will be set up on a small breadboard in the lab-room. The initiation programming of the interrupt hardware interface is described in the file STK500 Interrupts.pdf on CANVAS.

Hint: try the FreeRTOS QUEUE-primitive for the communication. It is on a slightly higher abstraction level than the pure binary semaphore implementation and is suitable for communication. Det kan se ut som:

In interrupt:

xQueueSendToBackFromISR(xQueueButtons, (void*) &buttons),
buttons variable must be a global static variable

In sporadic task:

xQueueReceive(xQueueButtons,&command, portMAX_DELAY);
command is a local variable (inside the task function)

The QUEUE -primitive moves the information from *buttons* to *command*.

c) Redraw the state chart above to include all transitions (reaction to all possible inputs in all states).

.