The Bayert Real Time Operating System for the MSP430

Released: April 13th, 2018

1. System Features

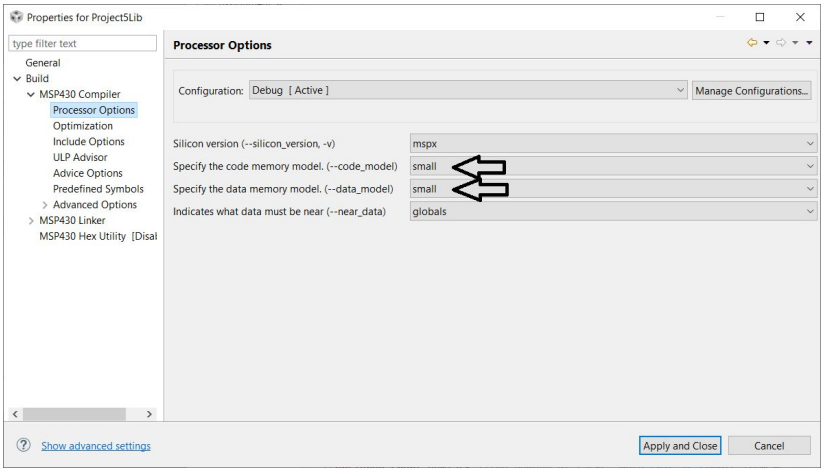
a. Preemptive operating system
b. Ability to create tasks on the fly.
c. All register data is saved and restored between tasks.
d. Seperate stacks for each task.
e. Memory management

I have neither given or received nor have I tolerated others use of unauthorized aid. *Jon Bayert*

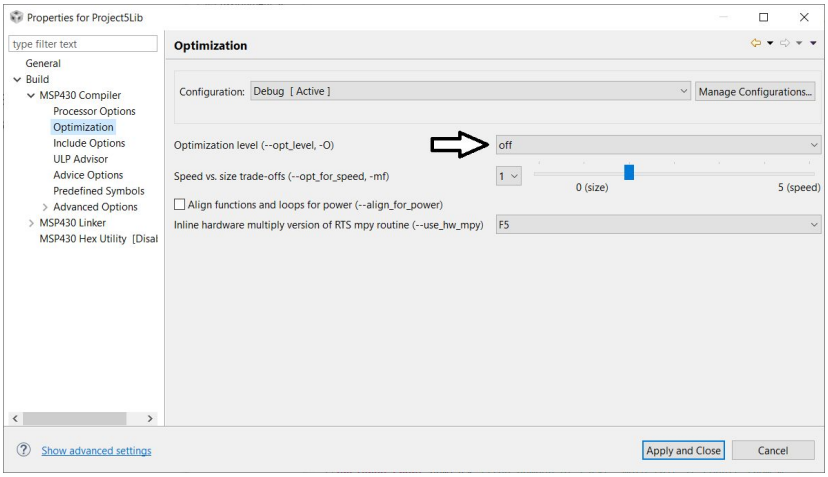
2. General use

a. System requirements

i. The system code memory must be set to small and data memory must be set to small (so that the PC only stores 16-bits



ii. Also disable optimization

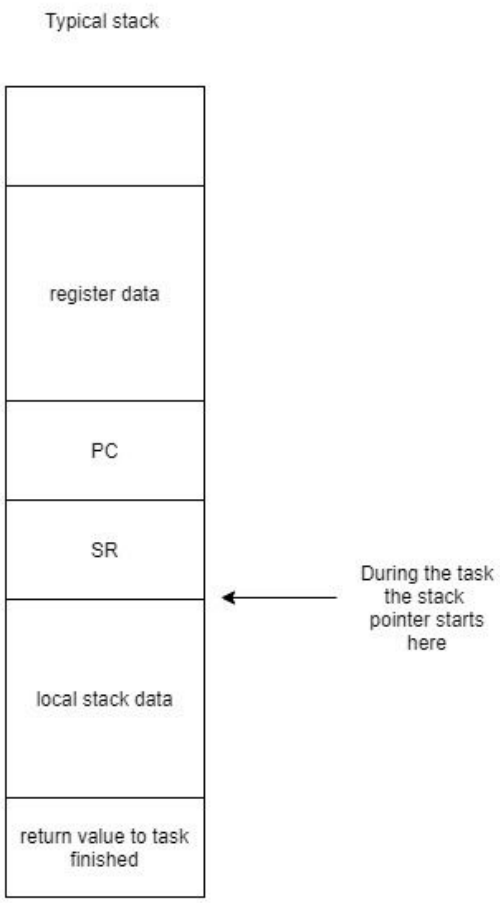


iii. Also the tasks can not disable interrupts or use TimerA0.

b. Stack Setup

i. Each task gets its own stack to use. Once a task time is up, the system will interrupt the task. The system will save the stack pointer and program counter, along with all the other registers. These values are saved onto the task's stacks when other tasks are running.

ii. A typical stack for a task will look like this:



iii. When a task finishes it will return from the bottom of the stack. This return value is saved in the last value of the stack.

iv. The stack will be deallocated and can be reused after a task has finished.

c. Process ID

i. Each task receives a unique Id. This id can be used to get the current status of the task or to kill the task.

ii. The process ids are not reused unless there are more than MAX_PROCESSES have been created

d. Maximum run time

i. Every function will be guaranteed to be called every CYCLE_LENGTH microseconds. The run time of each task is dynamic, so that each task will run CYCLE_LENGTH divided by the (# of tasks).

e. TimerA0

i. The timerA0 interrupt is used to schedule the change of each task. This timer is set up to use the TA0CCTL0 interrupt.

3. General Use Functions

a. These can all be found in the RTOS.h file.
b. RTOSsetup
    i. C Usage: void RTOSsetup(void)
    ii. Set up the RTOS system. This must be done only once before the system starts
c. RTOSinitTask
    i. C Usage: int RTOSinitTask(void (*pFun)(void))
    ii. RTOSinitTask sets up a task that will be run. This function adds a task to RTOS. This can be done while the RTOS is running, so a task can create another task.
    iii. Returns -1 is it could not add a task because there was an error or the system already reaches the mask stacks. Otherwise returns a process id.
d. RTOSrun
    i. C Usage: int RTOSrun(void);
    ii. Runs all tasks. Once all tasks have finished the function returns 0 when all tasks finished without errors.
e. killProcess
    i. C Usage: void killProcess(int proc_id);
    ii. Stops a process from running
f. get_proc_state
    i. C Usage: char get_proc_state(int process)
    ii. Returns the state of a process.
    iii. Possible return values:
        1. PROCESS_NOT_ALLOCATED
            a. process not defined yet
        2. PROCESS_NOT_STARTED
            a. process has not started yet
        3. PROCESS_RUNNING
            a. the process is running
        4. PROCESS_FINISHED_WITHOUT_ERRORS
            a. process finished without any known errors

4. General Use Constants

These can all be found in the RTOS.h file. These can be changed by the user.

a. MAX_TASKS
    i. the maximum number of tasks that will be called.
    ii. Default: 6 tasks
b. STACK_SIZE
    i. The maximum size (in words) of the stack of each function.

ii. Note that the stack size must also contain enough space for the 16 registers to be stored between stacks.
iii. Default: 40 words

c. CYCLE_LENGTH
   i. How often each task will be called (in microseconds).
   ii. Each task will run for CYCLE_LENGTH divided by (number of tasks)
   iii. Default: 40000 microseconds
d. MAX_PROCESSES
   i. How many process ids might be created in a single run
   ii. Default: 100

5. Operating functions

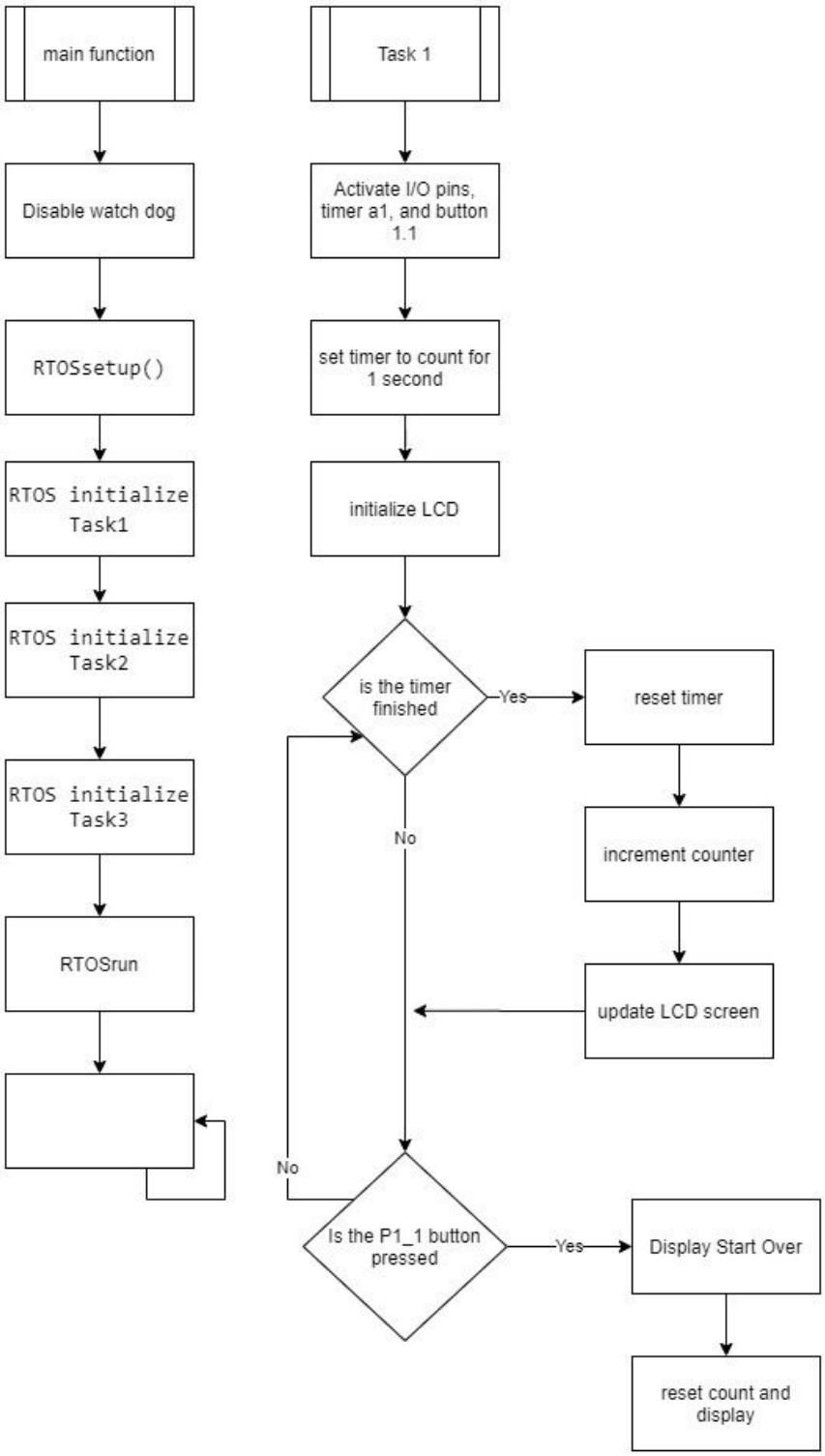
These are operating functions that are used in the background. These should not be used in the main code. Use caution when editing them.

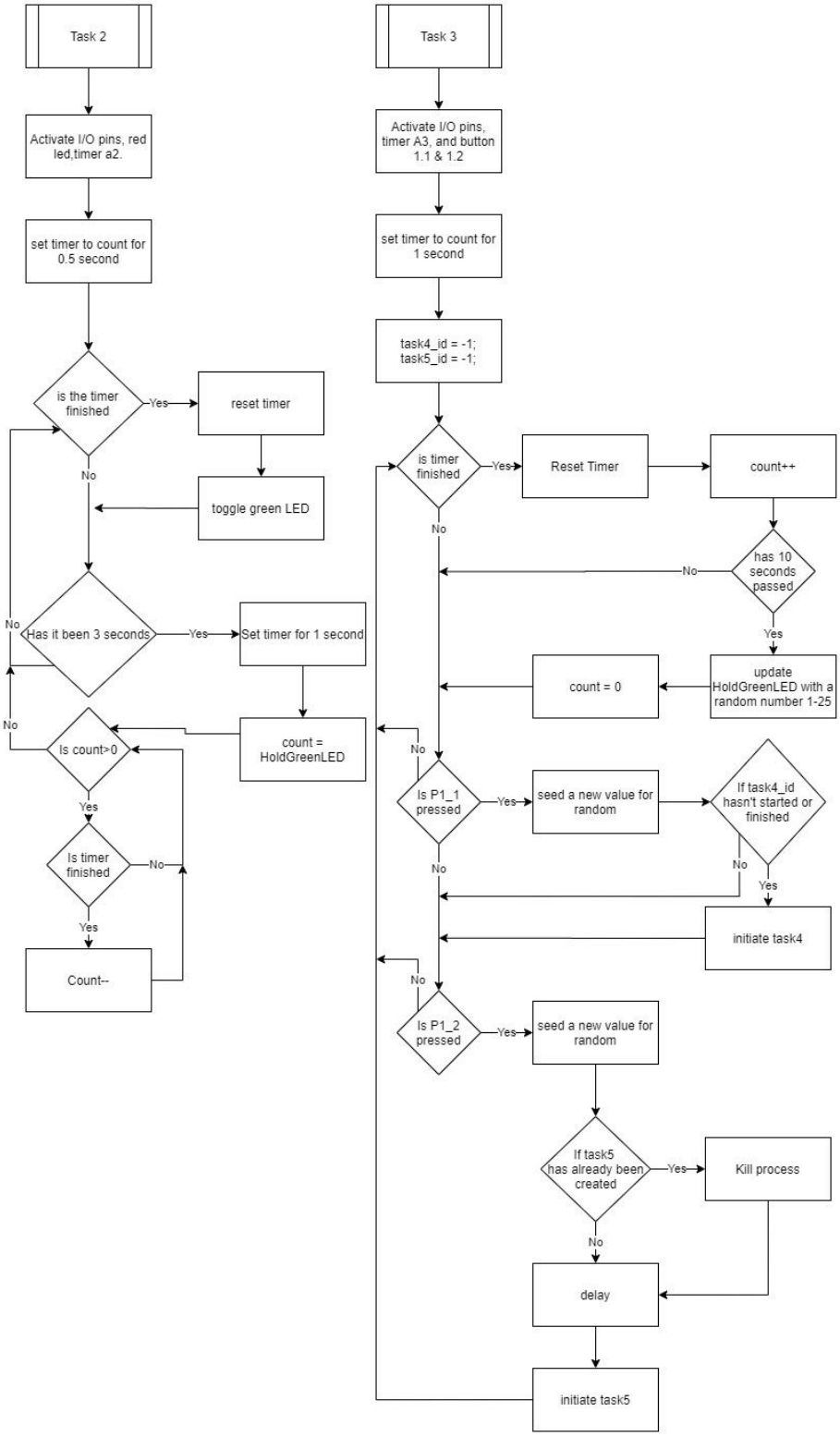
These functions were written in a mix of assembly and c.

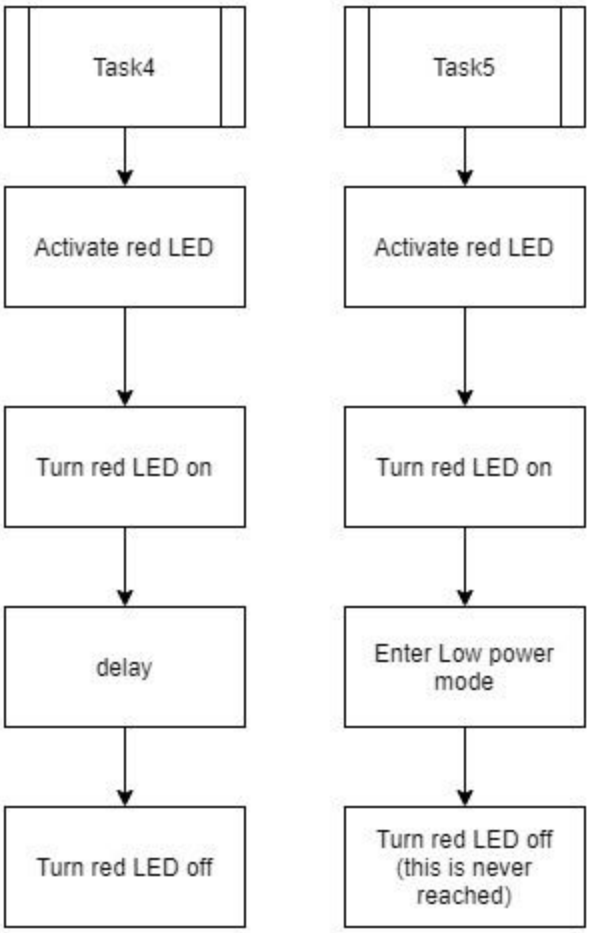
a. RTOSrunReturn
   i. When all tasks are finished this returns out of the RTOSrun function.
b. RTOSrestart
   i. restarts the RTOS when the timer has stopped (e.i a task has finished or a task was added).
c. TaskFinished
   i. When tasks are finished they return here. This removes the task and deallocates the stack.
d. Alloc_stack
   i. Find and allocate a stack. returns -1 if all the stacks are used otherwise returns the stack number.
e. Alloc_proc
   i. Find and allocate a stack otherwise returns the stack number.
f. Set_proc_state
   i. sets the process state.
g. Get_proc_task_number
   i. Get the task number based on a process id.
h. Get_proc_stack
   i. Set the stack number based on a process id.
i. Get_proc_state
   i. Set the process state.
j. Set_proc_state_by_stack
   i. Set the state of a process. stack is the stack location of the process.

6. Flowcharts

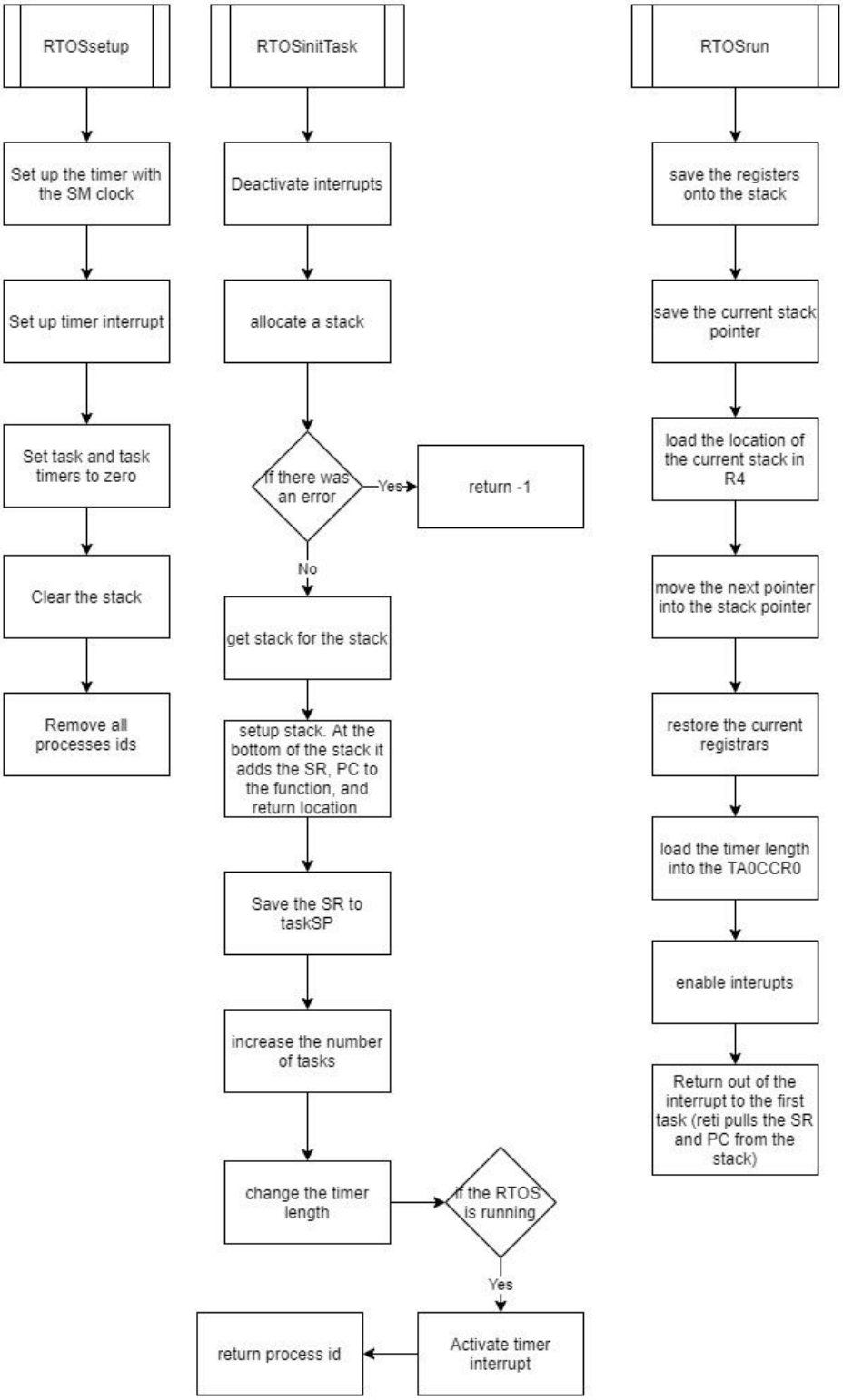
a. Demo

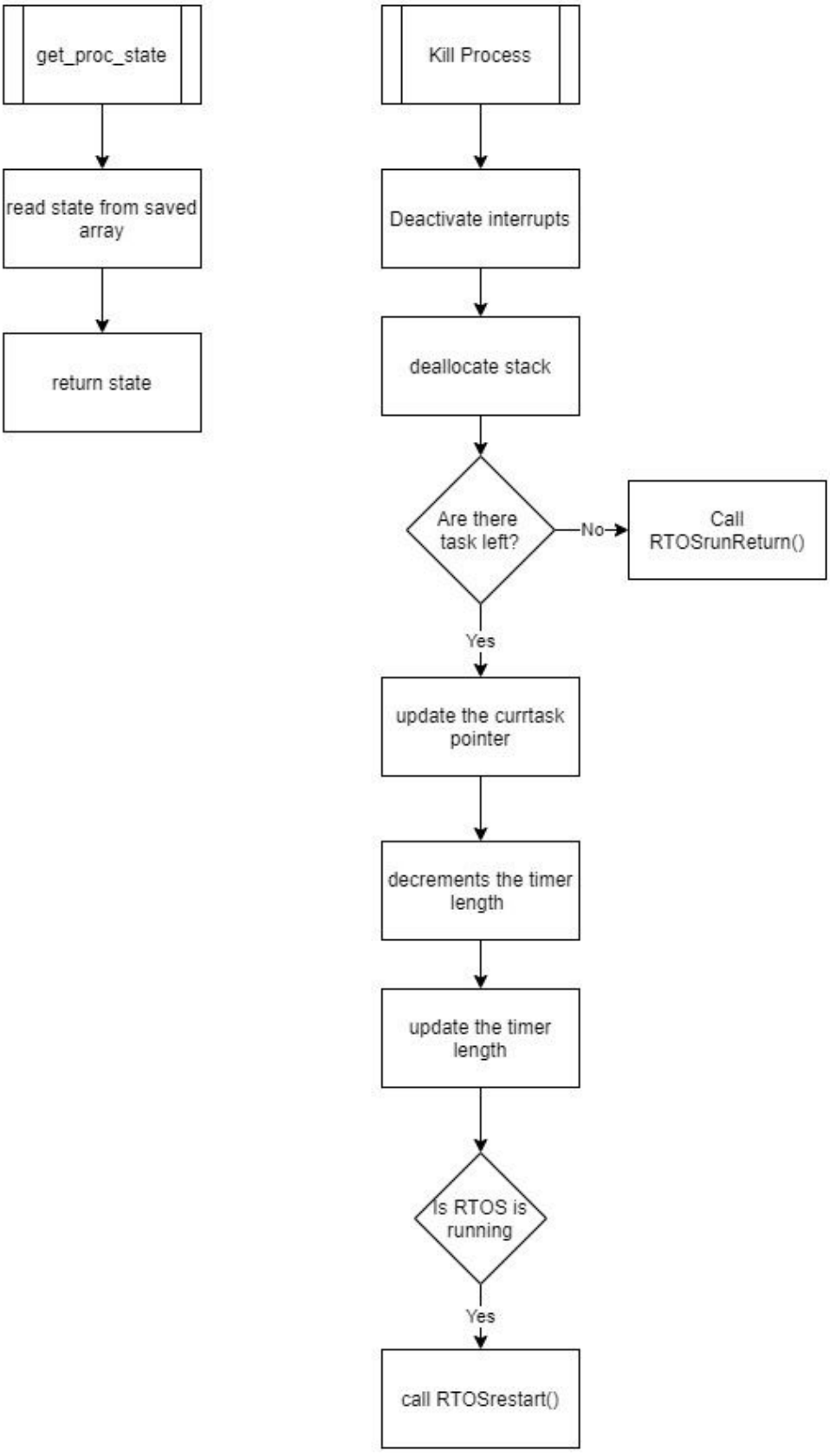



Task 2
Activate I/O pins, red led,timer a2.
set timer to count for 0.5 second
is the timer finished
Yes
reset timer
No
toggle green LED
Has it been 3 seconds
Yes
Set timer for 1 second
No
count = HoldGreenLED
Is count>0
No
Yes
Is timer finished
No
Yes
Count--
Task 3
Activate I/O pins, timer A3, and button 1.1 & 1.2
set timer to count for 1 second
task4_id = -1;
task5_id = -1;
is timer finished
Yes
Reset Timer
count++
No
has 10 seconds passed
No
Yes
update HoldGreenLED with a random number 1-25
count = 0
No
Is P1_1 pressed
Yes
seed a new value for random
If task4_id hasn't started or finished
No
Yes
No
initiate task4
No
Is P1_2 pressed
Yes
seed a new value for random
If task5 has already been created
Yes
Kill process
No
delay
initiate task5

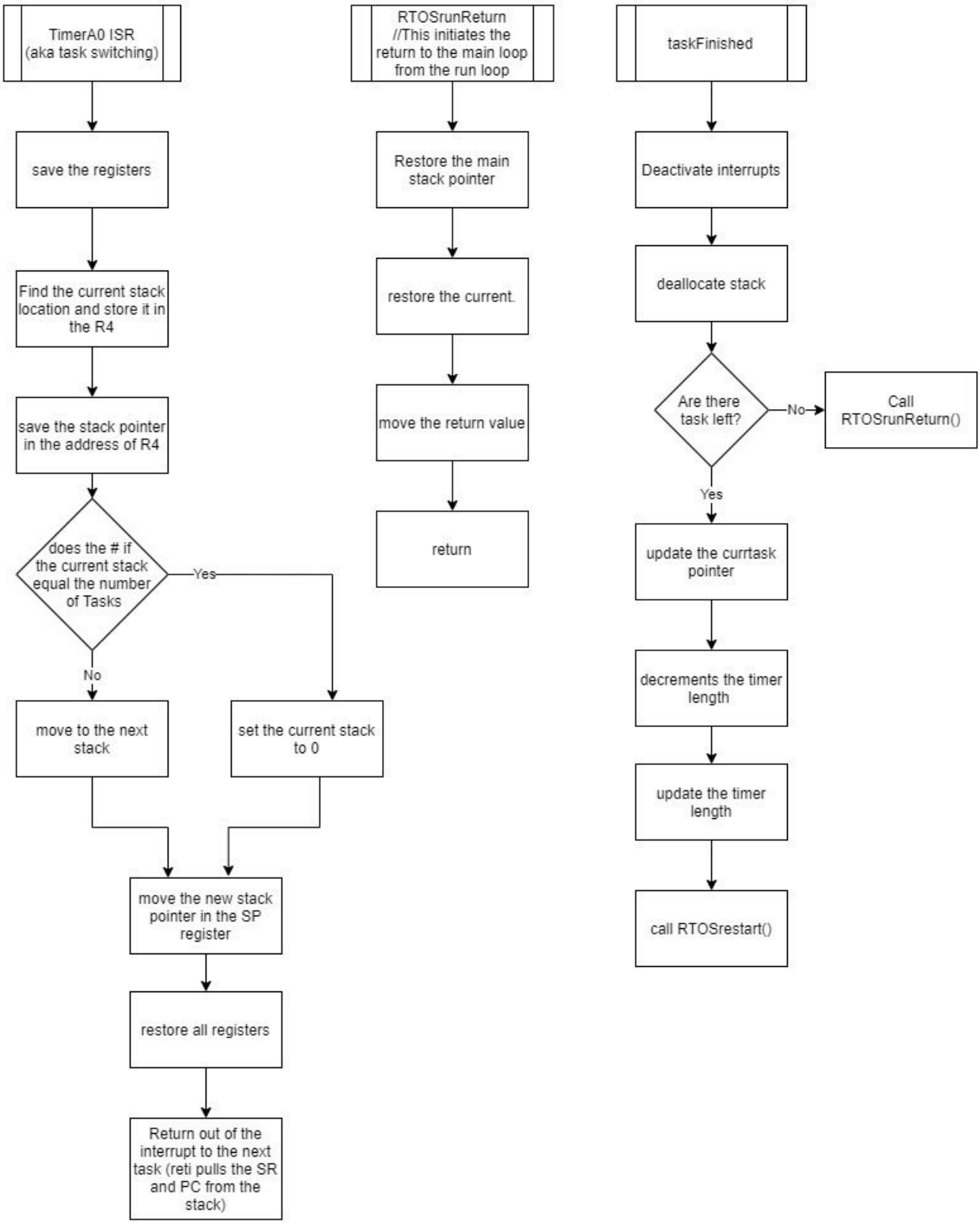

Task4
Activate red LED
Turn red LED on
delay
Turn red LED off
Task5
Activate red LED
Turn red LED on
Enter Low power mode
Turn red LED off (this is never reached)

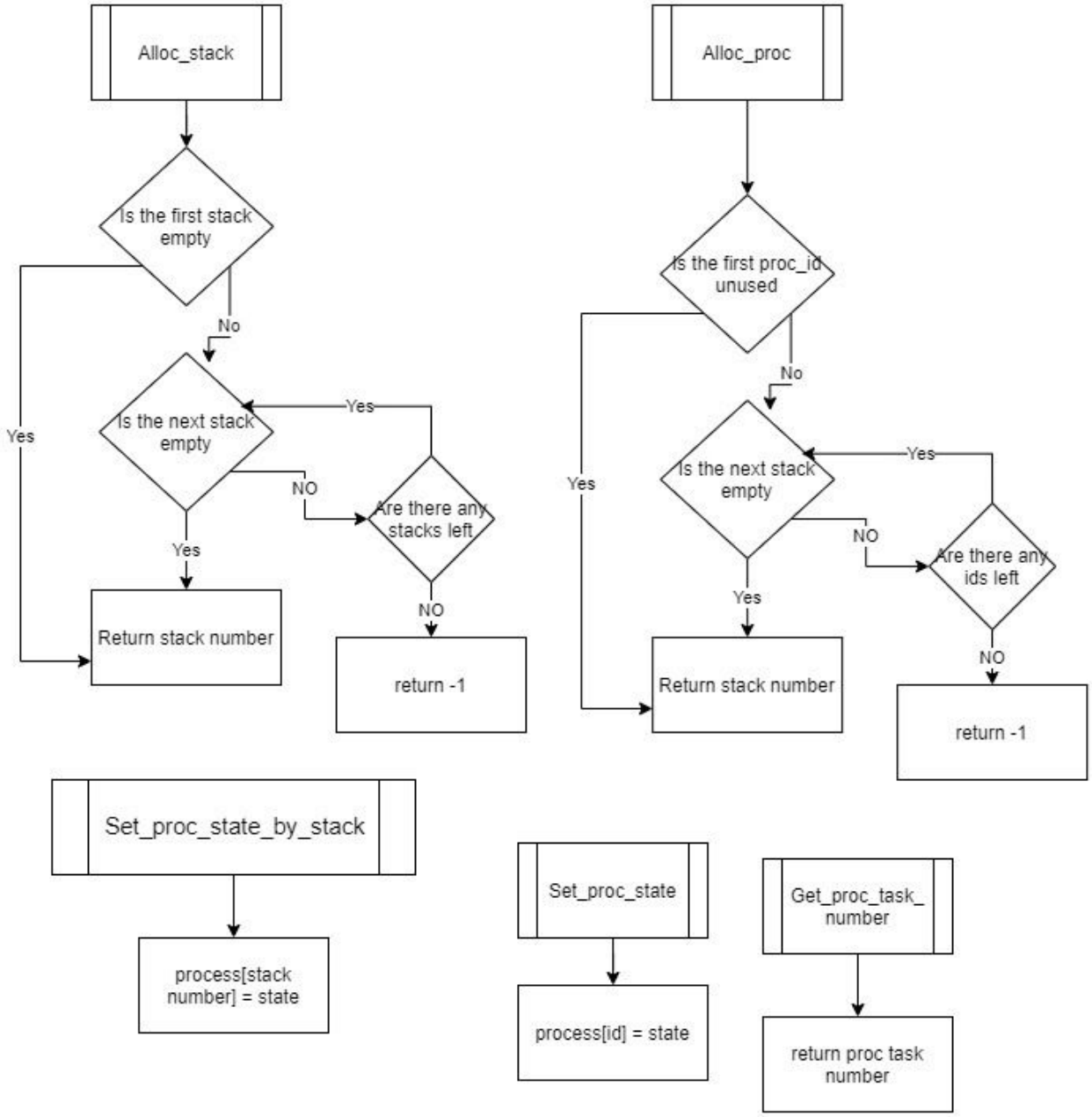
b. General Use Function

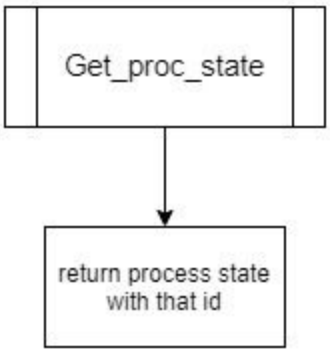


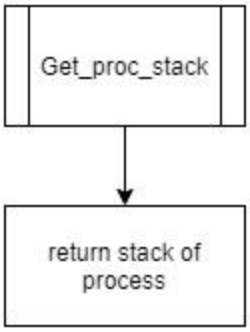

get_proc_state
read state from saved array
return state
Kill Process
Deactivate interrupts
deallocate stack
Are there task left?
No
Call RTOSrunReturn()
Yes
update the currtask pointer
decrements the timer length
update the timer length
Is RTOS is running
Yes
call RTOSrestart()

c. Operating Functions




Alloc_stack
Is the first stack empty
No
Yes
Is the next stack empty
Yes
NO
Are there any stacks left
Yes
NO
Return stack number
return -1
Alloc_proc
Is the first proc_id unused
No
Yes
Is the next stack empty
Yes
NO
Are there any ids left
Yes
NO
Return stack number
return -1
Set_proc_state_by_stack
process[stack number] = state
Set_proc_state
process[id] = state
Get_proc_task_ number
return proc task number


Get_proc_state
return process state with that id


Get_proc_stack
return stack of process