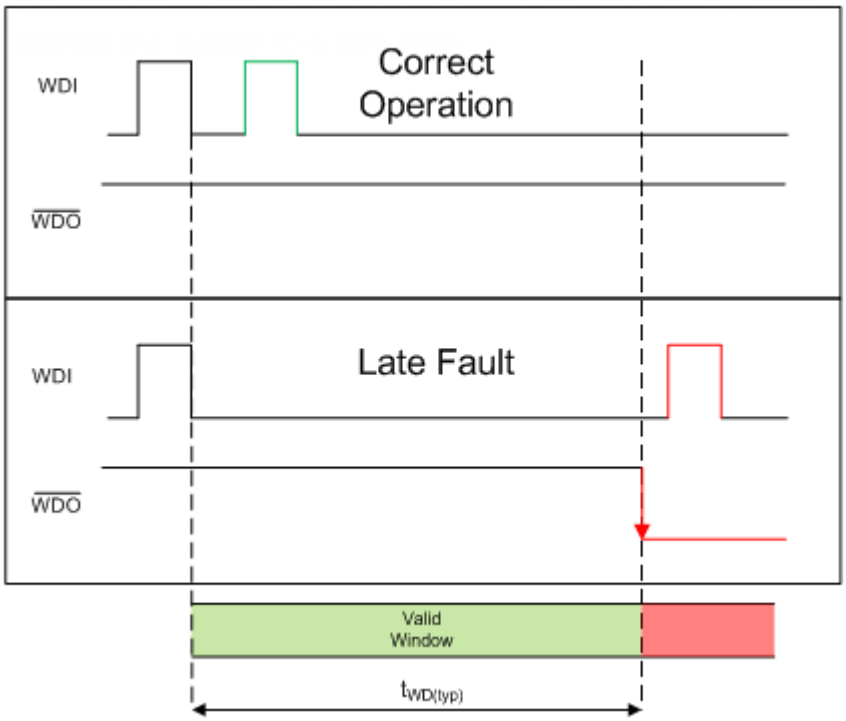
Lesson Watch Dogs

- Watchdogs
- Task Resuming & Suspending
- EventGroups
- Lab Assignment: Watchdogs

Watchdogs

Watchdog is a timer which can continuously check if there is any malfunction in the system operation and perform certain actions to restore normal operation. Watchdogs are commonly found in embedded system devic cal failure in the system th



Operation of a standard watchdog timer

There are hardware and software watchdogs. Hardware watchdog awaits a signal/pulse from a resource in the system during its timer period. If it receives the signal, it will reset itself and restart the timer. This continues till the system operations are normal and stable. If the resource fails or there is any fault in the system, the watchdog receives no indication during its timer period. Once the timer period elapses, it can take certain actions to bring the system back to stable state. The action could be resetting the system or any other action which restores normal operation.

Software watchdog is a task which monitors other tasks. Each task will report to the watchdog about its

normal operation. If any of the tasks misbehave, then the watchdog can alert the user/system or take corrective action to get the task back to normal state.

> **Ref: https://e2e.ti.com/cfs-file/__key/communityserver-blogs-components-weblogfiles/00-00-00-03-59/1538.fig2.PNG**

?

Task Resuming & Suspending

A freeRTOS task that is currently running can be suspended by another task or by its own task. A suspended task will not get any processing time from the micro-controller. Once suspended, it can only be resumed by another task.

API which can suspend a single task is:

```
void vTaskSuspend( TaskHandle_t xTaskToSuspend );
```

> Refer this link to explore more details on the API. **https://www.freertos.org/a00130.html**

API to suspend the scheduler is:

```
void vTaskSuspendAll( void );
```

> Refer this link to explore more details on the API. **https://www.freertos.org/a00134.html**

API to resume a single task:

```
void vTaskResume( TaskHandle_t xTaskToResume );
```

> Refer this link to explore more details. **https://www.freertos.org/a00131.html**

API to resume the scheduler:

```
BaseType_t xTaskResumeAll( void );
```

> Refer this link to explore more details. **https://www.freertos.org/a00135.html**

EventGroups

Event group APIs can be used to monitor a set of tasks. A software watchdog in an embedded system can make use of event groups for a group of tasks and notify/alert the user if any of the task misbehaves.

Each task uses an event bit. After every successful iteration of the task, the bit can be set by the task to mark completion. The event bits are then checked in the watchdog task to see if all the tasks are running successfully. If any of the bits are not set, then watchdog task can alert about the task to the user.

Below are the APIs that can be used. Refer to each of the API to understand how to use them in your application.

- xEventGroupCreate
- xEventGroupCreateStatic
- xEventGroupWaitBits
- xEventGroupSetBits
- xEventGroupSetBitsFromISR
- xEventGroupClearBits
- xEventGroupClearBitsFromISR
- xEventGroupGetBits
- xEventGroupGetBitsFromISR
- xEventGroupSync
- vEventGroupDelete

Lab Assignment: Watchdogs

Please follow the steps precisely in order to complete the objectives of the assignment. If you use the C++ FreeRTOS framework, it should make the assignment significantly easy.

1. Create a `producer task` that takes 1 light sensor value every 1ms.
   - After collecting 100 samples (after 100ms), compute the average.
   - Write average value every 100ms (avg. of 100 samples) to the `sensor queue`.
   - Use medium priority for this task

2. Create a `consumer task` that pulls the data off the `sensor queue`
   - Use infinite timeout value during queue receive API
   - Open a file (sensor.txt), and append the data to an output file on the SD card.
   - Save the data in this format: `printf("%i, %i\n", time, light)"`
   - Note that if you write and close a file every 100ms, it may be very inefficient, so try to come up with a better method such that the file is only written once a second or so...
   - Use medium priority for this task

3. At the end of the loop of each task, set a bit using FreeRTOS event group API.
   - At the end of each loop of the tasks, set a bit using the `xEventGroupSetBits()`
   - Task 1 should set bit1, Task 2 should set bit2 etc.

4. Create a `watchdog task` that monitors the operation of the two tasks.
   - Use high priority for this task.
   - Use a timeout of 1 second, and wait for all the bits to set. If there are two tasks, wait for bit1, and bit2 etc.
   - If you fail to detect the bits are set, that means that the other tasks did not reach the end of the loop.
   - In the event of failed to detect the bits, append a file (stuck.txt) with the information about which task may be "stuck"
   - Open the file, append the data, and close the (stuck.txt) file to flush out the data immediately.
   - Extra Credit: Every sixty seconds, save the CPU usage info to a file named "cpu.txt". See terminal command "infoHandler" for reference. Open the file, write the file, and close it

immediately so the data is immediately flushed.

5. Create a terminal command to "suspend" and "resume" a task by name.
   - "task suspend task1" should suspend a task named "task1"
   - "task resume task2" should suspend a task named "task2"
6. Run the system, and under normal operation, you will see a file being saved with sensor data values.
   - Plot the file data in Excel to demonstrate.
7. Suspend the producer task. The watchdog task should display a message and save relevant info to the SD card.
8. Let the system run for a while, and note down the CPU usage in your text file.

What you created is a "software watchdog". This means that in an event when a loop is stuck, or a task is frozen, you can save relevant information such that you can debug at a later time.

> You may use any built in libraries for this lab assignment.

> For File I/O refer to the example here - http://socialledge.com/sjsu/index.php/SJ_One_Board#File_I.2FO

?

?