REAL-TIME SAFETY AND FAULT INJECTION

LAB EXERCISE

Introduction:

You are an embedded systems engineer working for a water distribution company. One day, you step into the shower, only to be greeted by a weak trickle of water. Something is wrong with the water pressure. You quickly discover that the problem is not just with your shower, but throughout the city. Upon further investigation, you suspect that the water tower, which controls the water supply for the city, may be malfunctioning due to a potential fault in the reading from the water level sensor used by the controller.

Fortunately, you were able to fix the problem by manually controlling the inlet and outlet valves of the water tower. However, you realize that the company needs a long-term solution to prevent such issues from happening in the future. You also realize that the water tower has a float switch that turns on when the water level is low, but it was not used previously since the water level sensor was always accurate. You decide to implement a redundant system using the float switch. The redundant system should ensure that the water level remains at a safe and consistent level.

Approach:

One possible implementation is using Bounded Interference Approach.

> ACM Transactions on Embedded Computing SystemsVolume 16Issue 4 Article No.: 94pp 1–19 https://doi.org/10.1145/3063313
>
> Authors: Stefano Esposito, Massimo Violante, Marco Sozzi, Marco Terrone, Massimo Traversone

The steps include:

- Metric selection: quantity most likely affected by the soft error and that can be measured offline and at runtime.
- Metric characterization: measure and gather samples. Then analyze for example fitting to a probability density function.
- Determine 3 thresholds:
    - Alarm Threshold: interference has occurred.
    - Warning Threshold: interference could have occurred. Need further observation.
    - Max Consecutive Warning Occurrence Threshold: if warning threshold crossed for multiple consecutive times interference has occurred.
- Implement detection by measuring the metric at run time comparing against thresholds.
- Implement recovery when interference detected.
    - Graceful degradation: alternate scheduling to finish task within deadline.
    - Hard recovery: switch to a standby spare (alternate actions) to restore functionality.

Setup:

A base system has been provided as a Keil project for LandTiger V2.0 LPC17XX Development Board using uC/OS-III. You need to have Keil uVision installed. An arm license is needed for compiling the OS.

Following is the code workflow and the uC/OS objects created.

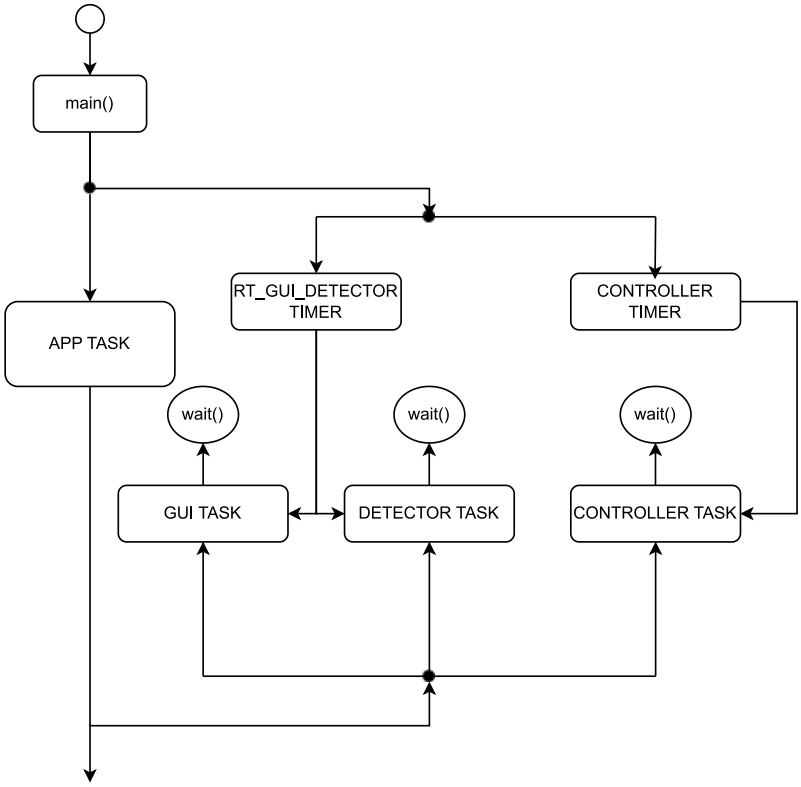


Figure 1. create/resume flow, main () creates timers and App Task. App task creates GUI, detector, and controller tasks. The timers resume the tasks periodically.

Following is the workflow of each task created. See the input and output variables each task can access. Exception is in controller where it needs to access current level using sensor reading. (copy to sensorReading first)

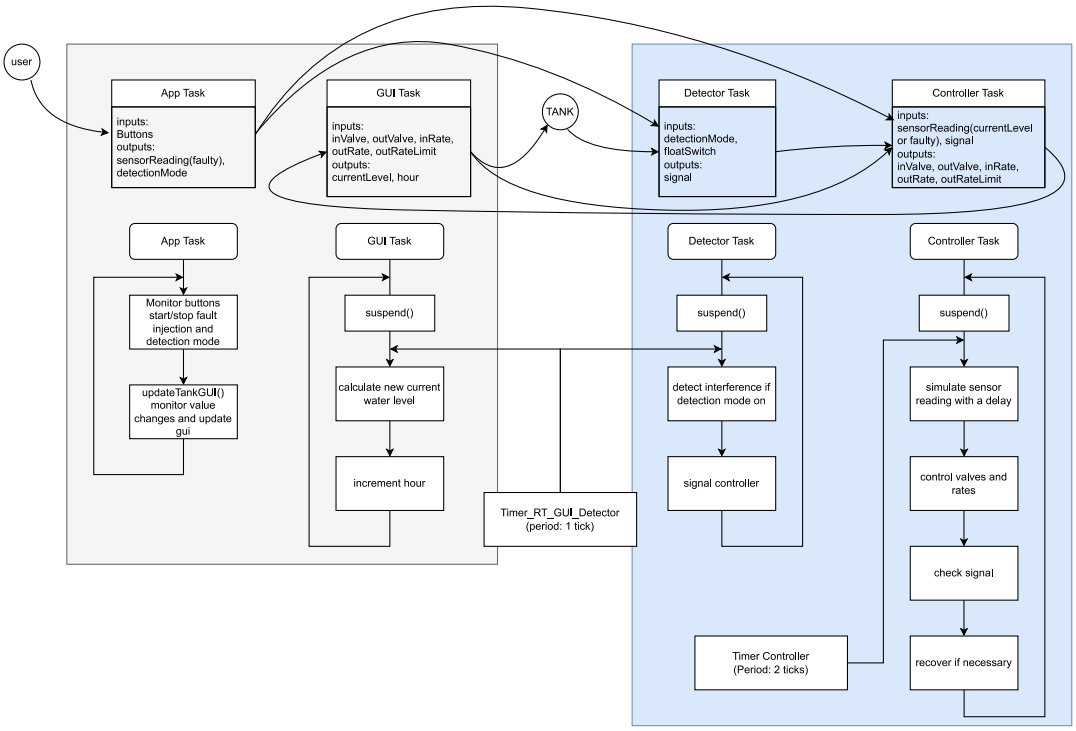
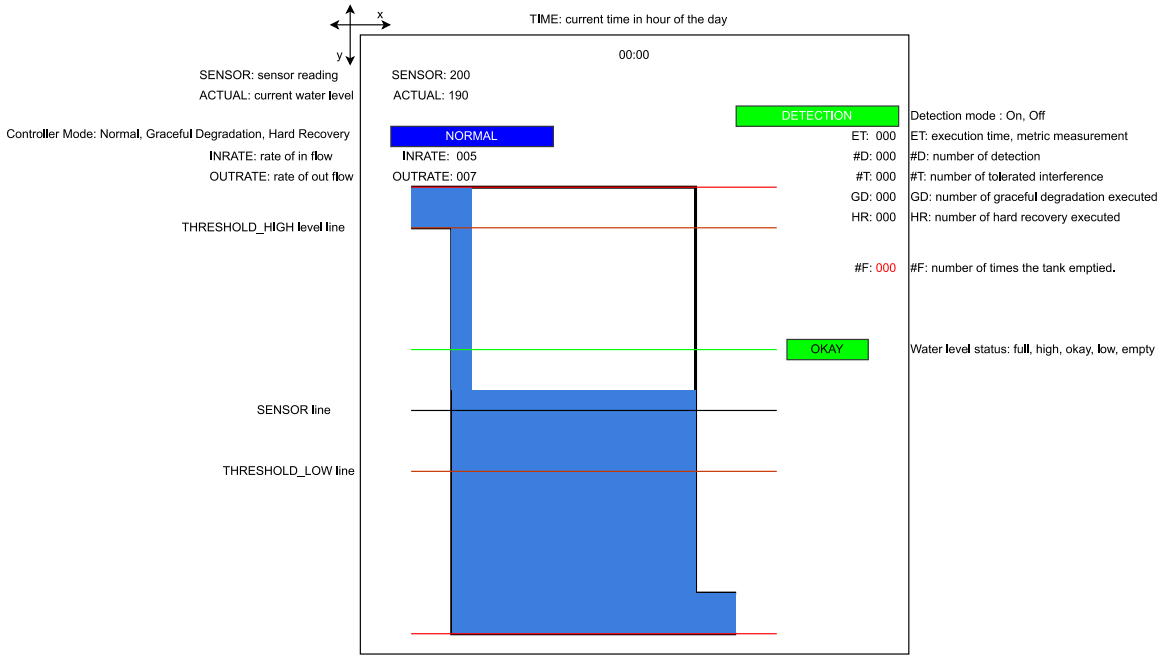


Figure 2. tasks flow and variables they can access, input output flow, blue background tasks need to be implemented

The water tower system is simulated and displayed on the lcd screen. Following is the GUI available.



System:

- App Task updates display with new values.
- GUI Task calculates actual water level and increments hour.
- The outrate varies with time.
- Controller stores the value of currentLevel in the sensorReading variable and suspends.
    - This is to simulate delay in the sensor Reading.
    - Think of it as controller starting the sensor and executing other tasks (during the time it suspends), after the execution (when resumed by timer) controlling the inlet and outlet valves using the sensorReading.
- Basic controller works in Normal Mode: if the sensor reading is below low threshold, open inlet valve. If the sensor reading above high threshold close inlet valve.

Buttons:

Fault injection and detection mode toggle has been already implemented using buttons.

- Joystick up: start injecting fault (bit flip) in sensor reading.
- Joystick down: stop injecting fault in sensor reading.
- Key1: turn Detection mode on.
- Key2: turn Detection mode off.
-

Relevant Files:

- Source: main.c (to be implemented), watertank.c (water tank simulation code)
- Headers: watertank.h definitions and declarations for watertank.c
- Configuration: Relevant values (constant definitions) for implementation are available in the watertank_cfg.h file. Modify as needed but with care.

Exercises

Exploration

1) Download base system, compile, and flash the board.
2) Test the fault injection using joystick and observe how the tank behaves in normal mode with and without fault injection.
3) Test if the detection mode on and off buttons work properly.
4) See the source code in GUI task to understand how the water level and hourly outrate is calculated and the variables involved.
5) See the code in Controller Task to understand how the controller works in normal mode and how sensor reading is delayed.
6) Optionally: download the proposed_solution, compile, flash, and test for reference.

Detection Implementation: Detector Task

7) Choose a metric that can be affected by bit flip on the sensor reading and that can be analyzed offline and measured at run-time.

   To get you going, you can choose the metric to be the time water stays below low water threshold. The water level continuously fluctuates due to varying water demand(outrate) and controller action (inlet valve on/off). The water level falls, the controller when detects low water level starts filling the tank again raising the water level above low threshold. The situation that must be avoided is the water tank getting emptied. The interference in the sensor reading can cause controller to not detect low water. Sensor reading can no longer be trusted but we have a float switch to detect when the water level falls below low threshold.

8) Analyze/estimate the different times taken to rise back after it falls below threshold when controller is in normal mode and no-fault injection occurs.

   The water level and hour (therefore outrate) are updated at each tick by GUI task, controller controls inlet valve at each 2 ticks using sensor reading from the last time controller activated.

   You can write a small program to simulate n days and generate a histogram (count the number of occurrences) of each different time values.

9) From the analysis/estimation set the three thresholds (can be set in configuration file or use your own global variable):
   a. Alarm Threshold: interference has occurred.
   b. Warning Threshold: interference could have occurred. Need further observation.
   c. Max Consecutive Warning Occurrence Threshold: if warning threshold crossed for multiple consecutive times interference has occurred.
10) Now implement metric measurement. Measure time needed for the water to rise back when it falls below threshold. Implement the consecutive occurrence counting mechanism.
11) Implement the comparison to the different threshold and set signal variable appropriately (use values in the configuration file or your own values).

Recovery Implementation: Controller Task

12) implement normal mode and recovery modes: graceful degradation, hard recovery.
13) Use signal variable to change the mode.
14) See configuration file for the values you need to use to display current controller mode on the display.
15) For graceful degradation, you need to simulate change in controller scheduling. (Remember how we simulated the delay.
16) For Hard recovery, avoid turning off the outlet valve.

Considerations: the inlet rate can be increased by 20%. Out rate can be limited to a certain max value. Set the inlet rate and outlet limit appropriately considering the real-world situation.

Test, test, test

17) Test your code with detection and fault injection on/off. Adjust thresholds as needed. Test the detection and recovery when detection is on. The goal is to avoid an empty tank situation.
18) You can use following additional variables to show on the display:
    a. #T: numTolerated
    b. #D:numDetected
    c. ET: executionTime
    d. HR: numHR
    e. GD: numGD