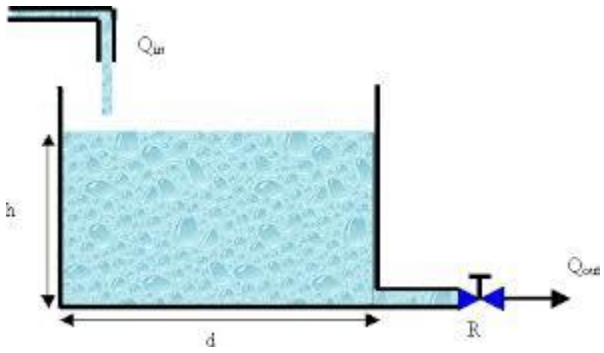
CEG3136 – Lab2
Simple State machine (C)

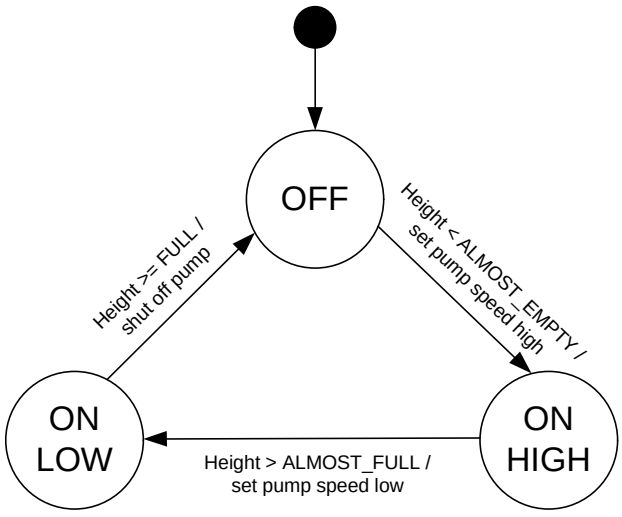
Objectives

To implement a state machine to control a water pump that is responsible for filling a water tank.



Introduction

A water pump is responsible for filling a water tank shown above. The tank is cylindrical with diameter d = 50”. The filling pipe diameter is 3”, and the draining pipe diameter is 2”. The Tank is equipped with water height sensor measuring the fill level in inches. The state machine that controls the pump control is as follows:

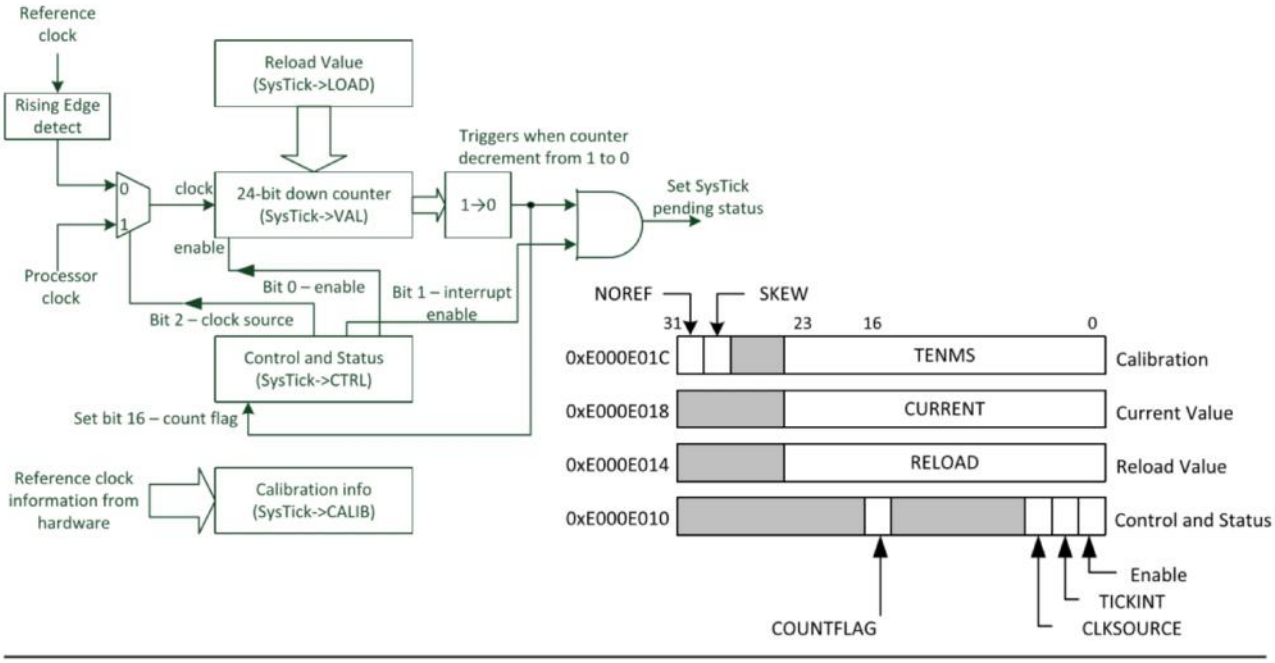


When the pump is at high speed, it pushes water at 1”/sec in the filling pipe, when it is set to low speed it pushed water at 0.5”/sec. To simulate the water consumption we are going to use a random number generator to generate numbers in the range [0, 1]”/sec.
The height sensor updates its measurements every 1 millisecond. And the pump state is updated accordingly.

The SysTick timer

Refer to “The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors”, chapter 9.5: The SysTick timer.

The Cortex-M processors have a small integrated timer called the **SysTick (System Tick) timer**. It is integrated as part of the nested vector interrupt controller (NVIC). It can generate the SysTick exception (#15). The SysTick timer is a simple decrement 24-bit counter and can run on either processor clock or a reference clock. The reason for having the timer inside the processor is to help software portability between Cortex-M processor systems. The SysTick timer can be used as a simple timer peripheral for periodic interrupt generation, delay generation, or timing measurement.



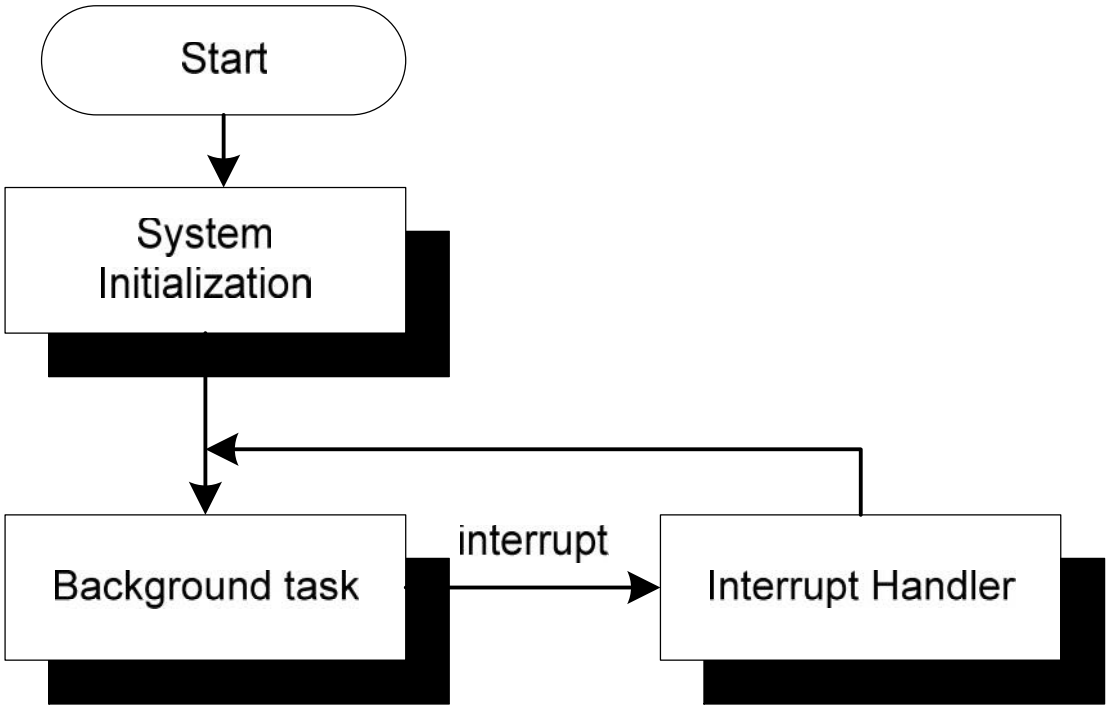
A simplified block diagram of SysTick timer

Using the SysTick timer

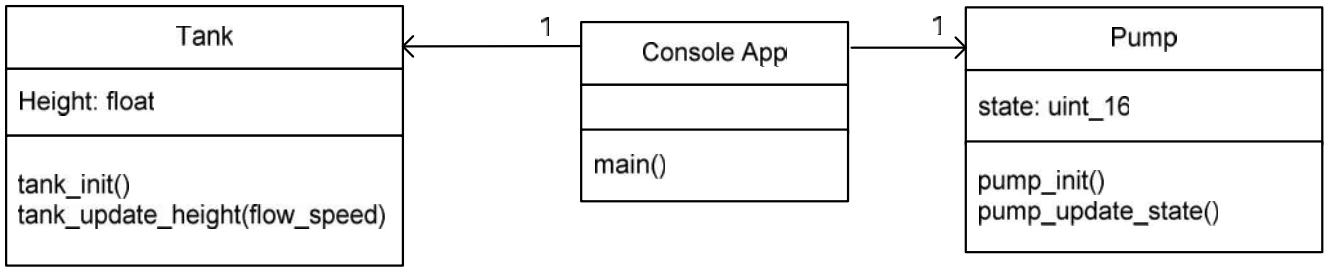
If you only want to generate s periodic SysTick interrupt, the easiest way is to use a CMSIS-Core function: **uint32_t SysTick_Config (uint32_t ticks).**
For example for a 1 millisecond interval, you can use: **SysTick_Config (systemCoreClock / 1000).** That means when we divide the core clock frequency in Hz by 1000, we get the number of clocks per millisecond. The timer interrupt handler: **void SysTick_Handler(void)**, will be invoked every 1 millisecod.

The water tank application is designed as interrupt based application with two modes – see figure below.

- Background task that handle user interface – intially is endless loop
- Forground task that is serving the system tick timer that is used for controling the tank operation.



The application was designed using object oriented programming concepts, the class diagram is shown below:



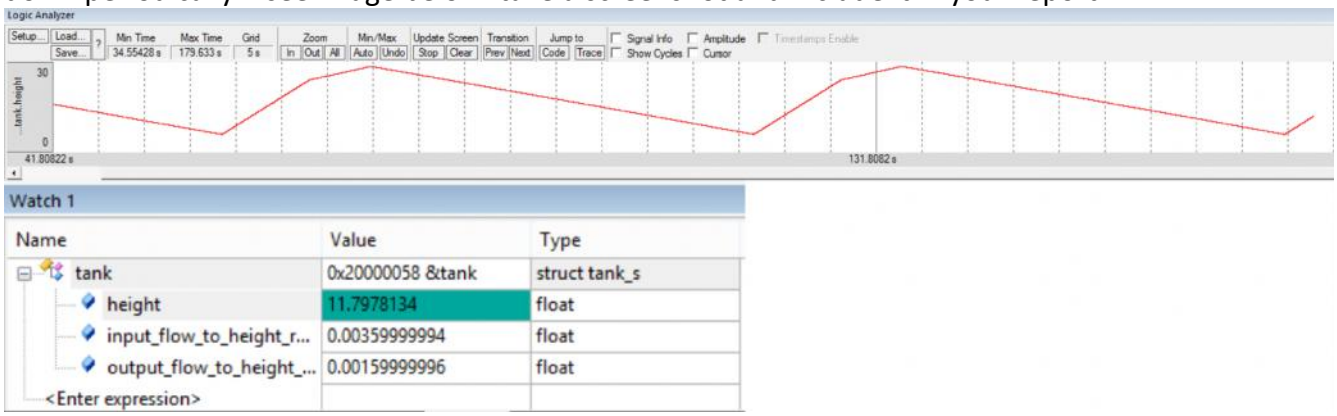
During the initialization phase the console application instantiates the pump object and the tank object. The tank object is estimating the water height inside the tank by monitoring the input & output flows in the pipes. The pump class controls the pump and set the input flow rate by implementing the state machine shown earlier.

Provided code

The provided code include the console application and the two classes for pump and tank.
The main initialize the tank & pump objects, then enter an endless loop. The tick handler calls the pump_update_state and tank_update_height respectively.
The pump_update_state in the pump class is supposed to implement a Mealy state machine as shown earlier. It's a Mealy state machine because the flow setting take place on the state transition only.

Task1

1) Download the attached µVision project from Bright Space.
2) Compile and run stubbed project code
3) Implement the pump state machine by completing the pump_update_state method
4) Compile and run the simulation – debug your simulation to make it work properly
5) Add the tank object to Watch1 and monitor the water height raising and dropping as time progresses – to add to watch highlight the tank object during debugging and right-click, then select add to Watch1.
6) Add a logic analyzer to monitor the height – expand the tank object, right click on height field and select Logic Analyzer. Click the setup button on the Logic Analyzer pan and set the max value to 30 and min value to 0 to see the full range.
7) When the system is working properly you should see the water level in the tank going up and down periodically – see image below. take a screenshot and include it in your report.



Task2

Modify the main (background loop) to include a prompt to ask the operator if she want to drain the tank (in case we want to do some maintenance activity). When the operator accept, the pump state machine will enter a new state called DRAIN, where the input rate is set to 0. Also the tank drainage should be set to twice the maximum rate. The main loop will also immediately prompt the operator if she want to exit the drainage mode. When the system exit the drain mode, it will resume normal operation as described before.

8) Modify the above state machine to include the DRAIN state. Include the state diagram in your report.
9) Modify the application to adequately support that modification – you need to figure out what to change to support such a change, e.g. add DRAIN to state enum, add a global variable to indicate drain request, etc. explain what you've done in your lab report.

10) Run the simulation for a bit, then force the tank drain. Monitor how the tank is drained, then resume normal operation to fill the tank once more. Take a screenshot and include it in your report.
11) Stop the debugger. Clean all targets – remove all intermediate files from your project. Project>Clean Targets
12) Close μVision
13) Zip the project directory and submit to Bright Space including the lab report.