Jesse Belanger

Dimitrios Tsitsirigkos

EC450

Final Project Report

“I am Having Trouble with My Basil”

Plant Monitoring System

The aim of our project was to create a plant monitor that could help a human take care of a plant with minimal effort. Our system utilized a light sensor and a moisture sensor to keep track of the most pertinent variables in the plant’s environment: soil moisture and light level. Depending on the combination of these levels, the monitor could raise various warnings to let the plant owner know if the plant entered a suboptimal state. Some of these warnings included LED light indicators, a notification alarm, and an email from the plant asking to be checked on.

At a software level, our project was largely a complex state machine implemented inside a handler for the MSP430’s Watchdog timer, with some additional functions for utilizing the ADC peripheral and creating a tone. We also use a number of functions which allow the main program to initialize the ADC, TimerA, USCI, and various buttons and LEDs at boot.

The state machine is driven by the plant state at the time of interrupt. To determine the plant state, we take readings from a light sensor (an amplified photoresistor) and a moisture sensor (hygrometer), which are converted to useful data using the MSP430’s ADC peripheral. The light sensor conversion is done by the main board, but the moisture sensor reading is done by the auxiliary board, and is then communicated back to the main board using SPI. Each time the WDT handler is entered, a function named *state\_monitor* is called, which checks the detected values against some thresholds we calibrated to the needs of a typical plant. It first determines whether the plant is experiencing too much, too little, or adequate moisture, and then checks if it has enough light. The resulting six combinations of light and moisture each correspond to a state of our state machine.

After determining the present state of the plant, action can be taken on its behalf. However, the machine first checks if the plant is in “Day” mode. We did some research on plant health and we found that plants should “rest” for at least 8 hours per day. When plants “rest” all lights have to be off and they shouldn’t be watered because it attracts various fungus. Therefore, we decided that the user should be able to change the state of the plant according to whether it is daytime or nighttime. In Day mode, the state of the plant triggers various events to happen, but in “Night” mode, the monitor doesn’t raise any alerts about the state of the plant, or try to water it. The current mode is demonstrated by the color of the on-board LED - green for Day, red for Night.

If the device is in Day mode on a particular cycle of the WDT handler, a series of if/elseif statements determine which actions to take based on the current state. A different set of actions takes place in each state, as defined in the table below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Good\_Good** | **Good\_Dark** | **Dry\_Good** | **Dry\_Dark** | **Wet\_Good** | **Wet\_Dark** |
| LEDs | All off | "Dark" LED on | "Dry" LED on | "Dry" and "Dark" LEDs on | "Wet" LED on | "Wet" and "Dark" LEDs on |
| Alarm Tone | No | Yes | Yes | Yes | Yes | Yes |
| Email Notification | No | Yes | Yes | Yes | No | Yes |
| Water Plant | No | No | Yes | Yes | No | No |

The simplest action to take is toggling the various LED indicators. For each state, we simply set the pins corresponding to the LEDs we want on, and we clear the pins associated with those we want off. The “Wet” and “Dry” LEDs cannot be on simultaneously, but the “Dark” LED can be on with either, or by itself. As described in the table, no LEDs on means the plant is in a comfortable state.

The alarm tone is generated through use of Timer A, as we have done in the homeworks. Whenever we enter a suboptimal state where the caretaker should be alerted that something is amiss, we turn on the sound by toggling OUTMOD in the TACCTL0 and allowing a signal to be sent to the speaker from the associated pin. The tricky part to manage in executing an alarm tone of a certain length was figuring out a mechanism for turning the sound back off after that time. To solve this problem, whenever the alarm begins to sound, a flag is set indicating that the alarm is on. Whenever the WDT handler is entered, it checks the flag to see if the alarm is on, and if so decrements the counter indicating how long the tone should run for. When this counter gets to 0, the sound is turned back off, the flag is cleared, and the tone counter is reset.

Another tricky part of implementing the alarm tone was that we had to figure out how to keep the alarm from going off every single time it detected that it was in a suboptimal state. For instance, if the plant is in a steady, dry state, the handler would detect this dry state repeatedly many times in a second. It did not make sense from a product design standpoint to have the alarm sound continuously while the plant needed care of some kind - that would grow very annoying. Instead, we designed it so that the alarm would only sound at transitions between states. We implemented this by tracking the plant’s previous state as well as its current state. The procedure to turn the alarm on is only executed if the last state is different from the current state. At the end of each run through the WDT handler, the current state is set to be the last state for the next run through.

For sending the email notification, the MSP430 does most of the work for figuring out that an email should be sent, but a Raspberry Pi handles the logistics of being connected to a network and being able to send an email. Whenever the monitor detects that it has entered a state where an email should be sent, it raises the voltage on the associated pin. The RPi listens on a GPIO pin which is connected to that pin for the voltage to go from low to high. Every five minutes, Cron is schedule to run a script which reads the value of that pin. If the value is a 1, and the previous value was a 0, the script knows that the state of the plant has gone from good to bad, and it executes a command which sends a standard email. Each time, before the script terminates, the “current” value is stored as the new “previous” value by being saved to a text file that the script knows how to read.

We designed it to only send an email on the “posedge” of the signal because we did not want the monitor to be annoying the plant owner by sending an email every five minutes that the plant was experiencing discomfort. We also purposefully designed it to not send an email if the plant was experiencing conditions that were too wet, but not too dark. The logic behind this was that the plant’s soil is not likely to spontaneously become too wet - such a circumstance likely means that the caretaker over-watered the plant, and should be aware of it from having seen the light come on as they did it. We felt it would thus be redundant to send an email in that case.

The implementation of the plant-watering mechanism was one of the biggest challenges in both software and hardware. From a coding standpoint, the challenge was similar to that of sounding the alarm for a period of time. However, there were even further complications. During testing we noticed that sometimes a transition from dry to good could fluctuate back into dry for a barely perceptible amount of time, and that this could cause the plant to be watered even if it didn’t need to. To solve this, we created a mechanism to delay the watering process. Instead of turning the motor on immediately after detecting entry into the dry state, each successive entry decrements a delay counter - indicating that the state has become steady (if the machine enters dry, then goes back to good, this will reset the counter and it will have to start over). A flag is also set or cleared indicating whether the motor should run or not. If the counter gets to 0 and the flag is set, the WDT will enter an if statement which turns the motor on, changes the motor’s state to running, and sets a counter to countdown the time the motor must run for. This part works exactly as the timer countdown worked.

For the hardware, we used two Launchpads, two breadboards, a number of LEDs, our two sensors, and a 5 volt DC motor. Some diagrams of our hardware layout are shown below:

Schematic 1: Board to peripherals connections

Schematic 2: Mock up of overall system layout (including plant)

Overall, we feel that our project was pretty successful. We were able to implement all of the features that we thought up and designed for our plant monitor, and even got to add special, extra features, such as the email notification. At first, we were not sure if we would be able to find a way to implement the email feature, but then we realized that we could devise a relatively simple communication protocol to interface the Launchpad and RPi via their pins.

However, there were some moments where not everything worked perfectly. For example, we noticed that the motor drew a lot of current, which was affecting the entire circuit. More specifically, whenever the motor was activated more than 3-4 times consecutive times, or encountered too much resistance to its rotation, the circuit would start behaving badly. In particular, this interference would cause the moisture sensor to stop being very precise and we figured out that happened because of an error in the communication between the two boards. More specifically, we would see the “wet” LED turn on even if the sensor was not as wet as it was supposed to be in order to cross that threshold.

These problems didn’t become apparent until very late in our project development (since we had been testing on an unburdened motor, and not our contraption). Though we didn’t have enough time to change our hardware design and pin mappings, we realized that we could minimize those issues by shifting our design around a bit. Specifically, if we were to do this again, we would interface both sensors to a single MSP430 and would have the motor alone driven by the auxiliary board. In this way, the motor would have a dedicated power supply all to itself, and would be isolated from the rest of our devices.

If we were to improve on the design, there are several more things we could do from a hardware perspective. For our first design, we didn’t focus too much on the watering system because obtaining a lot of supplies would have been very costly, and designing and building a perfect mechanical contraption would have used up valuable time we needed to write our code and figure out the core functionality for even a simple mechanical component. However, if we could improve we would definitely make a more stable, sophisticated watering apparatus. In particular, we could expand on our design to have the cup reset and refill itself, or come up with an entirely new apparatus that would be capable of repeatedly watering the plant at the necessary intervals.

Another thing we would improve would be to take the chips off of the Launchpads and put them directly into a breadboard. We forgot that doing so was possible until Professor Giles reminded us, and we realized that doing so would allow us to have a much neater, less cluttered device.

As for the teamwork on this project, we were always meeting to work on the project. We mostly always worked together in the lab, however, in the later stages, the various tasks to be done were kind of split up between us. At the start, we worked together to come up with the overall design that we wanted to implement. During this stage, we brainstormed the features we wanted to include, how we would like to implement them, and then came up with first drafts of the hardware layout and the state machine function.

After, we split up the work a little bit. Dimitri focused more on designing the circuit and figuring out how to manage the pins on the two boards. He also spent a lot of time on learning how to use the sensors, calibrating them, and designing the water mechanism. Jesse, on the other hand, focused more on implementing and writing the code for the state machine. He also wrote the script and figured out how to use the Raspberry Pi to send the emails. Together, we figured out how to interface the various pieces, and spent many hours debugging and improving our design to polish it and make it work reliably.