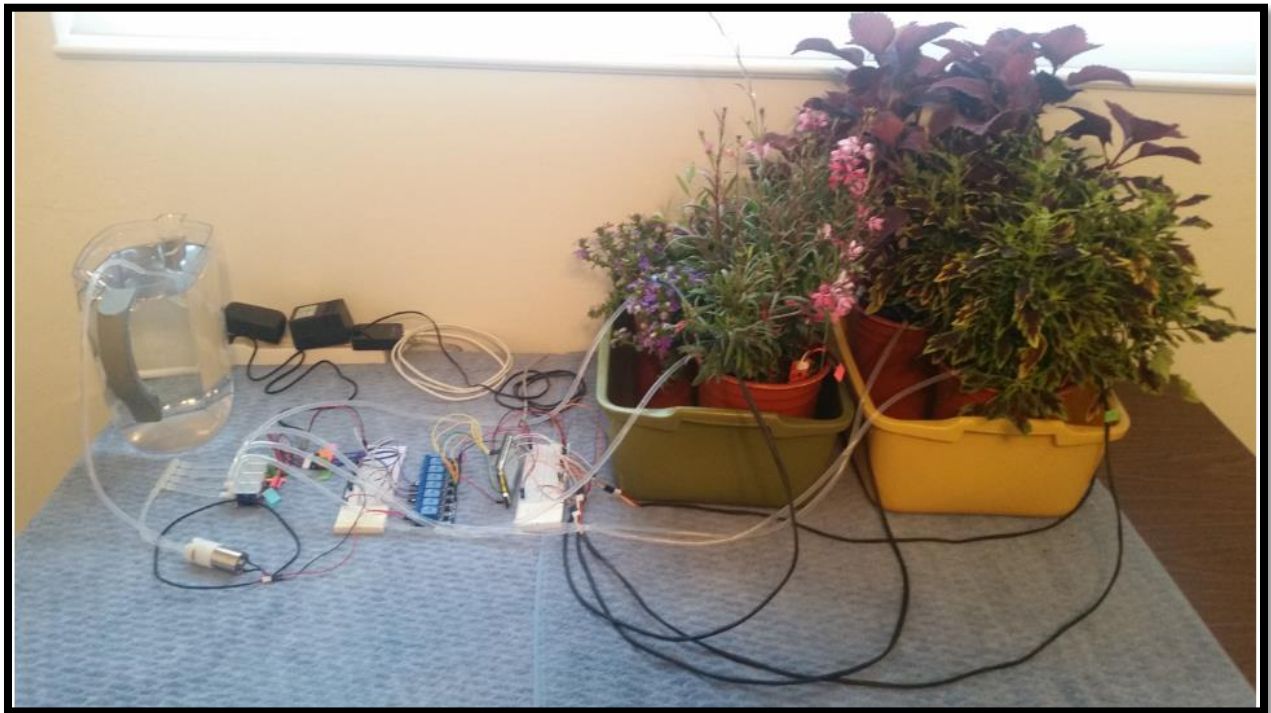


Automatic House Plant Watering System

Doyle B. Johnson
Final Project Report



EECE 6040C- Advanced Microsystems Design
August 9, 2019

Automatic House Plant Watering System

I. Problem Statement

The goal of this project is to design and develop a system to automatically water up to four house plants. Individual soil moisture sensors monitor the moisture associated with each plant. The microcontroller controls a water pump and valve to water each plant when a soil moisture sensor detects that the soil is too dry. Each plant will have its own sensor and separate water line to allow precise control of the water amount. A user can specify the watering duration for each plant, and the system will only water a plant once in a 24-hour period in order to allow time for the water to be absorbed into the soil.

The motivation for this project came from the fact that I have several plants in my office, but I often travel for work. When I am gone for extended periods of time, it would be convenient to have them automatically watered. While researching projects, I found that some of the project sites sell different types of “soil moisture sensors” and I thought these could be used to develop an automatic house plant watering system.

II. Design Overview

A. Design Requirements

The system was designed to meet several operational criteria. Specifically, the design must meet the following requirements:

1. Individually water up to four house plants
2. Allow the user to set the watering duration times
3. After watering, do not water the plant again for 24 hours
4. Display the watering date and time on the display

These requirements are met by a system with the following implementation details. First, the touchscreen of the Mikromedia PIC24EP board is used to display water duration times for each plant station to allow the user to select an appropriate watering amount. The watering date and time will be shown on the display. The voltage outputs from the sensors will be input to the on-board ADC. The ADC converts the analog voltage levels into digital values. The digital values are compared to baseline values, so that the plants are watered only when the soil conditions are determined to be too dry. The microcontroller will then output control signals to a set of relays to control a water pump and water valves to water the plants. Finally, the system will delay watering the same plant for 24 hours, in order to give the soil time to absorb the initial water.

B. Hardware Overview

The system uses the following hardware components:

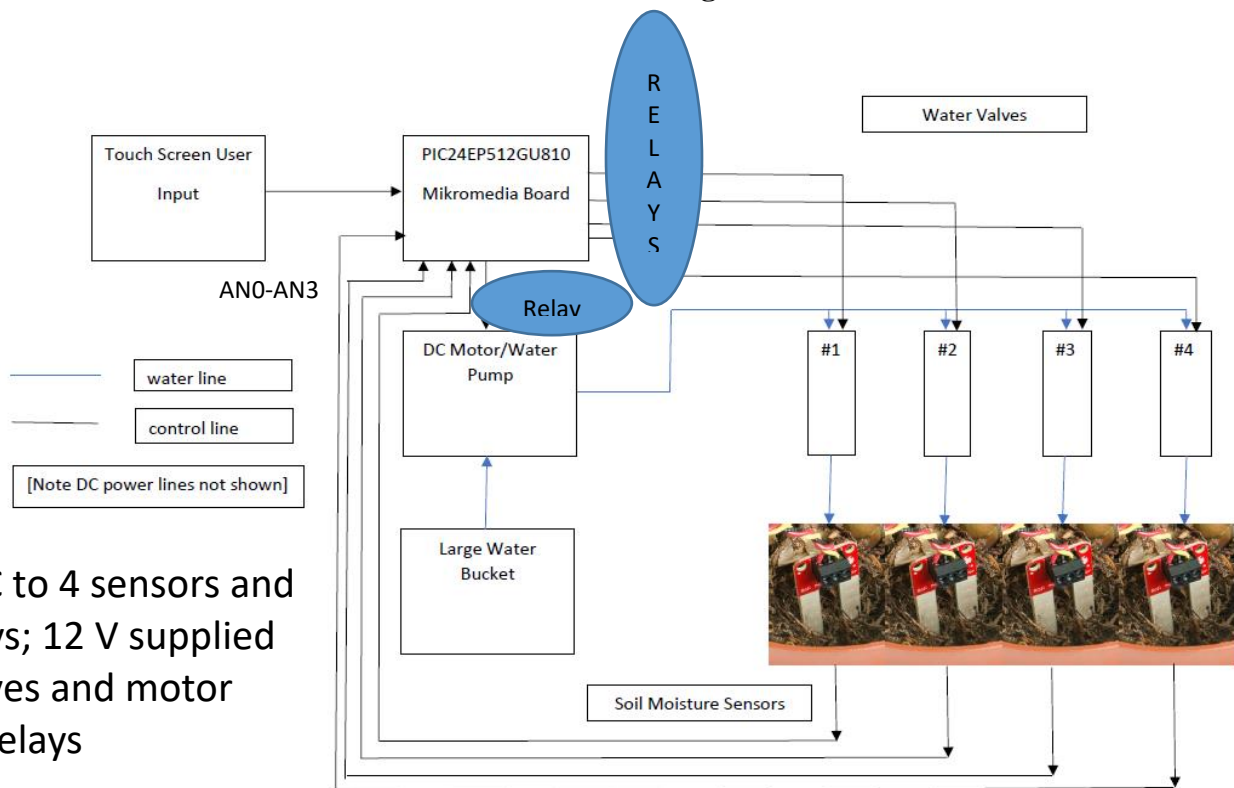
1. Mikromedia PIC24EP board (with touch screen interface)
2. Four (4) capacitive soil moisture sensors
3. One (1) small DC motor/water pump combination
4. Four (4) DC-controlled water valves

Automatic House Plant Watering System

5. One (1) 8-channel relay to activate motor/pump and valves based on control signals
6. One external 5V power supply
7. One external 12V power supply
8. Several feet of plastic tubing
9. Bucket for holding a sufficient supply of water

The following block-diagram illustrates the system hardware:

Hardware Block Diagram



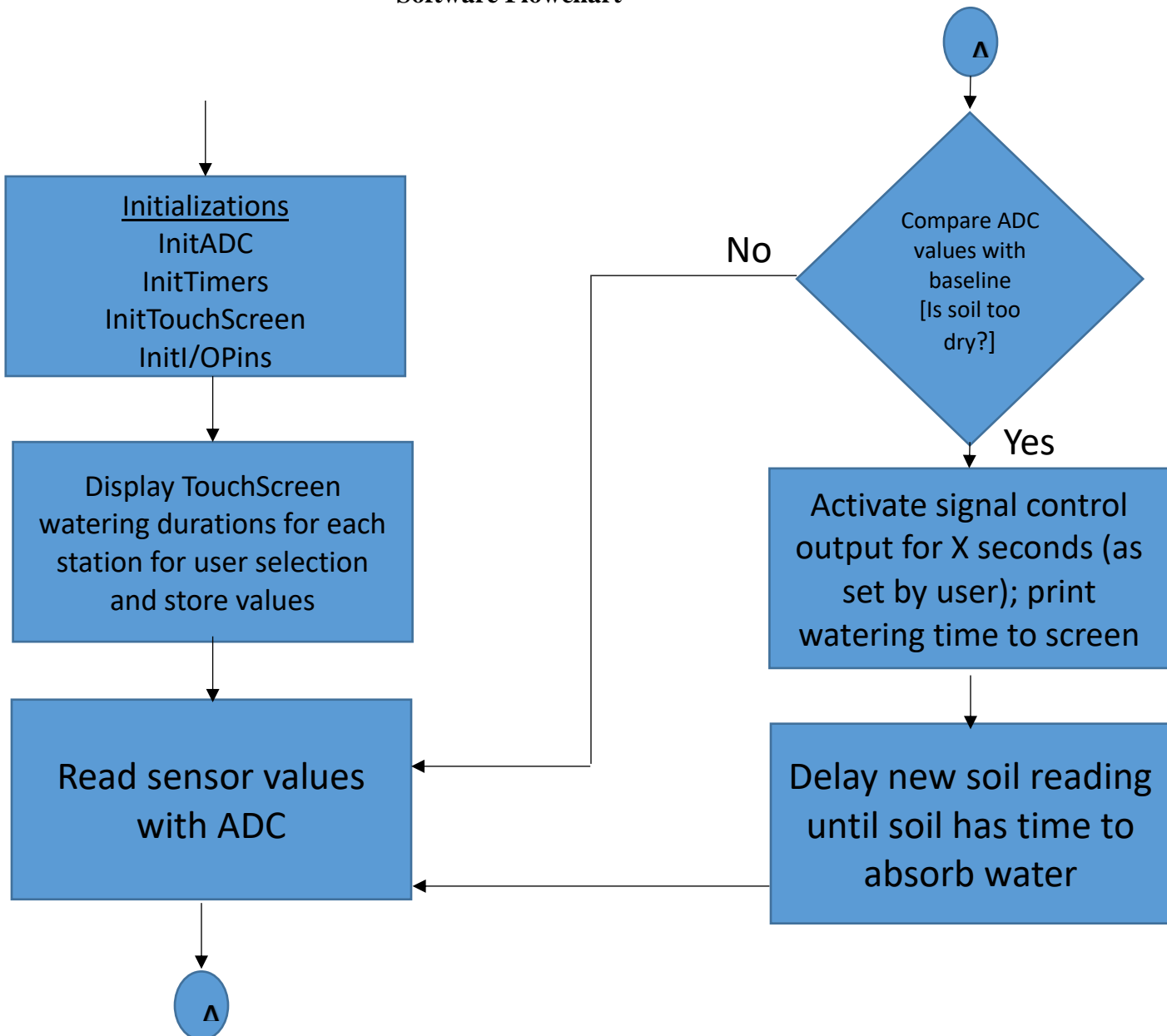
C. Software Overview

The control software initializes the microcontroller, the ADC, the touch screen display, the timers and interrupts, and the I/O pins. Specifically, the touch screen is initialized to display a series of screens to allow the user to set the watering duration for each plant station. The I/O pins are configured as input, output, analog and digital, as appropriate. The ADC is configured to read four analog input voltage values from the four capacitive sensors. The values are stored in the ADC1BUF and generate an interrupt. The input values are compared to baseline values to

Automatic House Plant Watering System

determine if a plant needs water. If so, output signals are generated to trigger the appropriate relays. An RTCC timer is used to limit each plant station to one watering in a 24-hour period. Specifically, the RTCC generates an interrupt every 10 seconds, and each station has a count value which is updated to ensure that the plant is not watered within 24 hours of a previous watering. Finally, the watering date and time are output to the touch screen display.

Software Flowchart



Automatic House Plant Watering System

III. Project Implementation

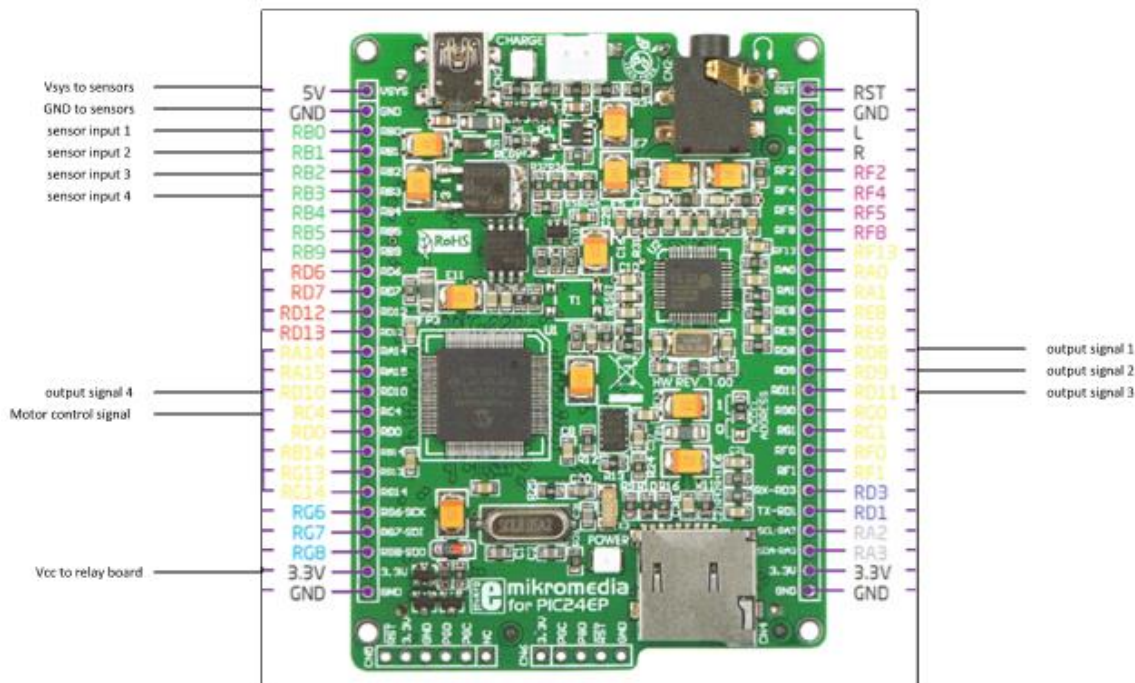
A. Hardware Implementation

As shown in this system photograph, the system includes several different components which are described in detail below.



1. Mikromedia PIC24EP board

The Mikromedia PIC24EP board is connected to the input sensors, and the output 8-channel relay board:



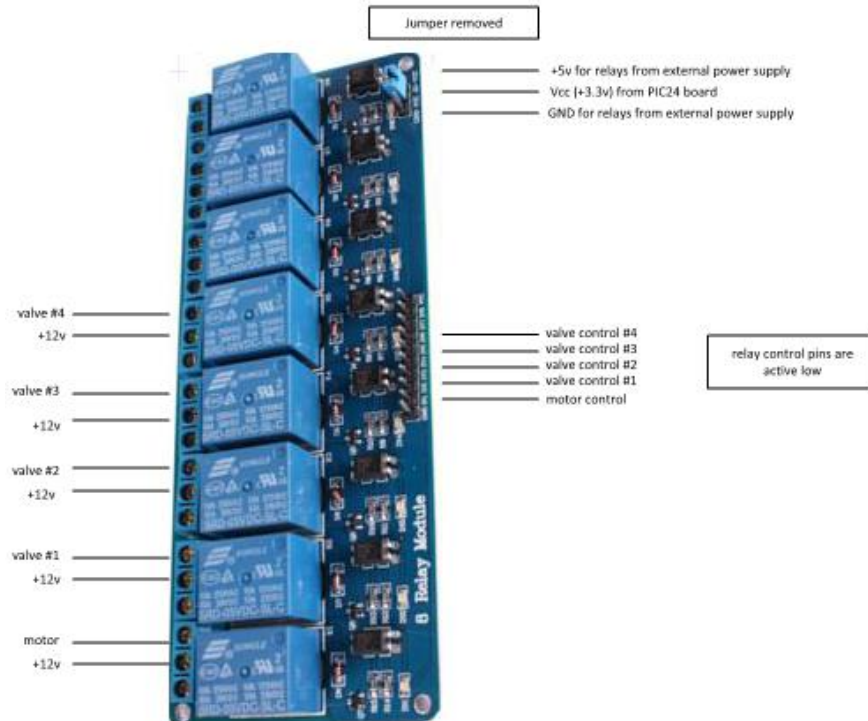
Specifically, pins RB0 – RB3 of the PIC24EP board are connected to the output of the four capacitive moisture sensors. These inputs correspond to AN0 – AN3, respectively, and these are the inputs to the ADC. Pins RD8 – RD11 are connected to separate relays on the relay board to control the watering valves. Pin RC4 is connected to a relay on the relay board to control the motor/pump. One of the 3.3 V pins is connected to the relay board to power the optoelectronic

Automatic House Plant Watering System

diodes, and Vsys and GND are connected to a bread board to power the capacitive sensors. Note the Vsys pin is +5V. The Mikromedia board itself is powered from the USB power connection.

2. 8-channel Relay Board

The 8-channel relay board is connected to the Mikromedia board, the control valves and the motor/pump.



<https://www.amazon.com/JBtek-Channel-Module-Arduino-Raspberry/dp/B00KTELP3I>

Input control pin IN1 is connected to pin RC4 of the Mikromedia board to control the motor/pump. Input pins IN2-IN5 are connected to pins RD8 – RD12, respectively, to control the valves. The Jd-Vcc pin and GND pins are connected to the external 5V supply in order to power the relay modules. The separate “Vcc” pin is connected to the 3.3V pin on the Mikromedia board. This wiring configuration provides opto-isolation between the Mikromedia board and the switched voltages on the opposite sides of the relays.

The input control pins are “active low”, which means that a low signal activates the relay to close. Therefore, the output pins from the Mikromedia board need to be set to a “0” (low) value in order to activate the relay. The output of the relays can be wired to normally open or normally closed switches, which would allow the outputs to be triggered by an active high signal, by just changing the output wiring. However, it is important not to trigger the outputs when the Mikromedia board is powered on or reset. In order to do this, it is best to wire the relays for active low control. The “trick” is to set the output pins high (“1”) during program initialization, even before the output pins are configured. This way, when the power is “off” the relays are not

Automatic House Plant Watering System

activated, and as soon as power is applied to the board, the pins are also “high”, ensuring that the relays will not be activated.

Each circuit on the output of the relays is wired similarly. Specifically, +12V from an external power supply is applied to the center connector, and the 12V input wire to the motor and valves is connected to the top connector as shown in the figure. When the relay is activated, the +12V from the supply is connected to the 12V wire of the device, and the device is powered. When the relay is deactivated, the power is disconnected, and the device stops operation.

3. External Power Supplies

Two external power supplies are used. A 5V supply is used to power the relays on the relay board, and an external 12V supply is used to power the motor/pump and valves. For purposes of wiring the project, two bread boards are used – one for supplying 5V (to the sensors from V_{sys} , and the relay from the external supply), and one for supplying the 12V from the external supply.

4. Capacitive Soil Moisture Sensors

The soil moisture sensors are “capacitive sensors” available from [elecrow](https://www.elecrow.com/crowtail-capacitive-soil-moisture-sensor.html) (<https://www.elecrow.com/crowtail-capacitive-soil-moisture-sensor.html>). After researching soil moisture sensors, I discovered that many “resistive” sensors on the market quickly corrode in use (most within 6 months of use). However, the capacitive sensors, which can be encapsulated in a protective sheath, do not corrode, and are therefore a better option.



However, there is really no documentation available for these sensors with respect to expected output voltages under various conditions. Therefore, I had to measure the output voltages under both wet and dry conditions to determine the full operating range. The dry sensors output approximately 2.6 volts, while wet sensors (placed in water) output approximately 1.5 volts. With the operative range, I determined that an adequate baseline value for activating the watering is about 2.3 volts. [These calculations are discussed in greater detail below with respect to the system software and ADC operation].

B. Hardware Issues Encountered during Design and Implementation

During the system design and construction, I encountered two main hardware issues. The first issue was the powering of the soil moisture sensors. The sensors require a 5V operating voltage. I originally supplied the sensors with power from an external power supply. However, I was not able to obtain consistent ADC readings. After debugging the ADC settings, and directly measuring the output voltages from the sensors with a meter, I realized that the ADC input values were “floating” because there was no common ground between the sensors and the ADC.

Automatic House Plant Watering System

For the temperature sensor experiment, the Embedded Shield board is directly connected to the Mikromedia board, so this was not an issue.

I solved this problem by connecting the Vsys pin (+5V) and GND pin on the Mikromedia board to the sensors (via bread board connections). The ADC values immediately stabilized and provided reproducible results. [You can also set the ADC to use an external Vref signal, but this is a more complicated solution.]

The second hardware problem was how to activate and wire the relays so that the relays would not trigger during initialization or reset. After experimenting with activating/deactivating the relays with low or high signals and using either the normally closed or normally open outputs, I determined that the only way to prevent the relays from triggering is to activate them with a “low” signal and set the output pins to “high” immediately upon initialization. This way, when the power to the Mikromedia board is off, the relays do not trigger. When power is applied, the pins are immediately set “high” which also keeps the relays from triggering. Only by explicitly setting the pins to low (“0”) will the relays activate, so the relays do not momentarily trigger during power cycles.

C. Software Implementation

The source code for this project is listed in Appendix A. I discuss each of the main modules of the software below.

1. Initializations

The device configuration and initializations are very important for this project. Specifically, with a number of different modules being used, each module must be properly configured before the main program body is executed.

I/O Pins

The analog input pins RB0 – RB3 must be set to analog and input. The digital output pins RD8 – RD11 and RC4 must be set to digital and output. Most importantly, the digital output pins must be set to “high” right away in order to prevent the relays from triggering on device reset.

```
void InitPins(void)
{
    _RD8=1;
    _RD9=1;
    _RD10=1;
    _RD11=1;

    _RC4=1;

    ANSELB = 0xFFFF;    // set PORTB to analog
    _TRISB0 = 0x1;      // set PORTB pins to input
    _TRISB1 = 0x1;
    _TRISB2 = 0x1;
    _TRISB3 = 0x1;
```


Automatic House Plant Watering System

```
ANSELD = 0x0000; // set PORTD to digital
_TRISD8 = 0x0;   // set PORTD pins to output
_TRISD9 = 0x0;
_TRISD10 = 0x0;
_TRISD11 = 0x0;

ANSELC = 0x0000; // set PORTC to digital
_TRISC4 = 0x0;   // set PORTC pin 4 to output

}
```

Touch Screen

The touch screen display is first initialized.

```
void InitDisplay(void)
{
    // init the graphics
    LCDInit();
    DisplayBacklightOn();

    // init the touch timer
    TickInit( 1);

    // init touch module (do not use NVM to store calibration data)
    TouchInit( NULL, NULL, NULL, NULL);

    // splash screen
    LCDClear();

    PutImage(0, 0, (void*) &intro, IMAGE_NORMAL);
}
```

Next, the screen displays a series of images for the user to input the watering time duration for each station. Each image includes areas for the user to touch to set the time, and to move to the next screen. A user's selection is determined based on the X-Y coordinates of the detected touch:

```
while(!finished)
{
    if (( TouchGetX() > 230) && ( TouchGetX() < 310) && ( TouchGetY() >
210)
        && ( TouchGetY() < 240))
    {
        PutImage(0, 0, (void*) &screen0, IMAGE_NORMAL);
        finished = 1;
    }
}
```

Automatic House Plant Watering System

```
}  
finished = 0;
```

Please select the watering duration
times for each plant station on the
following screens.

Click "Next" to continue....

Next -->

Plant Station One
select watering duration

15 seconds

45 seconds

30 seconds

60 seconds

Next -->

Plant Station Two
select watering duration

15 seconds

45 seconds

30 seconds

60 seconds

Next -->

Automatic House Plant Watering System

Plant Station Three
select watering duration

15 seconds	45 seconds
30 seconds	60 seconds

Next -->

Plant Station Four
select watering duration

15 seconds	45 seconds
30 seconds	60 seconds

Next -->

Click "Begin Watering" button to start automatic watering program.

Begin Watering

Automatic House Plant Watering System



I captured the plant watering duration values in debug mode (“watches”) to show that the set values can be entered and stored:

User selected input watering durations are correctly entered and saved (15, 30, 45 and 60 seconds; shown in hex)

<input checked="" type="checkbox"/>	plant0WateringTime	int	0x179A	0x000F
<input checked="" type="checkbox"/>	plant1WateringTime	int	0x179C	0x001E
<input checked="" type="checkbox"/>	plant2WateringTime	int	0x179E	0x002D
<input checked="" type="checkbox"/>	plant3WateringTime	int	0x17A0	0x003C

ADC

For this project, four separate analog inputs need to be read by the ADC and converted to digital values. To accomplish this, I chose to use a single channel (CH0), and to scan the inputs and place the results in the ADC1BUF (BUF0-BUF3). After the four values have been read, an interrupt is generated. The ADC1 configuration is as follows:

```
void InitADC(void)
{
    AD1CON1 = 0x04E4; // 12-bit mode; auto-sample; auto-convert
    AD1CON2 = 0x040C; // AVss and AVdd; scan; interrupt after 4th
sample
    AD1CON3 = 0x0002; // ADCS = 2 (76ns TAD)
    AD1CON4 = 0x0000; // ADDMAEN off; values stored in ADC1BUF

    AD1CSSL = 0x000F; // scan AN0 - AN3

    _AD1IF = 0; // Clear the A/D interrupt flag bit
    _AD1IE = 1; // Enable A/D interrupt

    _ADON = 1; // turn on ADC
}
```

Automatic House Plant Watering System

```
} // initADC
```

Inputs AN0 – AN3 are scanned, which correspond to input pins RB0 – RB3, respectively.

RTCC

The RTCC timer is initialized and the alarm is set to trigger every 10 seconds:

```
// set the alarm mask
_AMASK = 0b0010;    // once every 10 seconds
```

The interrupt service routine is called once every 10 seconds to count down each plant station's counter. This prevents a plant from being watered more than once in a 24-hour period. For example, if the pot is large, and the ground hard, the soil sensor might not detect the soil conditions until the water has been absorbed into the soil

```
void _ISR_RTCCInterrupt( void)
{
    // will call this routine every 10 seconds; plant only watered
    once in 24 hour period

    if (plant0Delay > 0)
    {
        plant0Delay--;
    }
    if (plant1Delay > 0)
    {
        plant1Delay--;
    }
    if (plant2Delay > 0)
    {
        plant2Delay--;
    }
    if (plant3Delay > 0)
    {
        plant3Delay--;
    }

    _RTCIF = 0;    // clear flag
} // RTCC interrupt service routine
```

2. Program Operation

After the initializations, the program executes the main loop.

```
while(1)
{
    DelaySec(10);
    _AD1IF = 1;    // set interrupt flag to process output
```

Automatic House Plant Watering System

```
if ((soilMoisture[0] > plant0Threshold) && (!plant0Delay))
{
    _RD8 = 0;    // valve
    _RC4 = 0;    // motor/pump
    DelaySec(plant0WateringTime);
    _RD8 = 1;
    _RC4 = 1;

    RTCCProcessEvents();
    sprintf(s, "PlantStation1 was watered for %d",
plant0WateringTime);
    LCDPutString(s);
    sprintf(s, " seconds at: ");
    LCDPutString(s);
    LCDPutString(_date_str);
    LCDPutString(_time_str);
    sprintf(s, "\n\r");
    LCDPutString(s);

    plant0Delay = 8640;
}

if ((soilMoisture[1] > plant1Threshold) && (!plant1Delay))
{
    _RD9 = 0;    // valve
    _RC4 = 0;    // motor/pump
    DelaySec(plant1WateringTime);
    _RD9 = 1;
    _RC4 = 1;

    RTCCProcessEvents();
    sprintf(s, "PlantStation2 was watered for %d",
plant1WateringTime);
    LCDPutString(s);
    sprintf(s, " seconds at: ");
    LCDPutString(s);
    LCDPutString(_date_str);
    LCDPutString(_time_str);
    sprintf(s, "\n\r");
    LCDPutString(s);

    plant1Delay = 8640;
}

if ((soilMoisture[2] > plant2Threshold) && (!plant2Delay))
{
    _RD10 = 0;   // valve
    _RC4 = 0;    // motor/pump
    DelaySec(plant2WateringTime);
    _RD10 = 1;
```


Automatic House Plant Watering System

```
_RC4 = 1;

RTCCProcessEvents();
sprintf(s, "PlantStation3 was watered for %d",
plant2WateringTime);
LCDPutString(s);
sprintf(s, " seconds at: ");
LCDPutString(s);
LCDPutString(_date_str);
LCDPutString(_time_str);
sprintf(s, "\n\r");
LCDPutString(s);

plant2Delay = 8640;
}

if ((soilMoisture[3] > plant3Threshold) && (!plant3Delay))
{

    _RD11 = 0; // valve
    _RC4 = 0; // motor/pump
    DelaySec(plant3WateringTime);
    _RD11 = 1;
    _RC4 = 1;

    RTCCProcessEvents();
    sprintf(s, "PlantStation4 was watered for %d",
plant3WateringTime);
    LCDPutString(s);
    sprintf(s, " seconds at: ");
    LCDPutString(s);
    LCDPutString(_date_str);
    LCDPutString(_time_str);
    sprintf(s, "\n\r");
    LCDPutString(s);

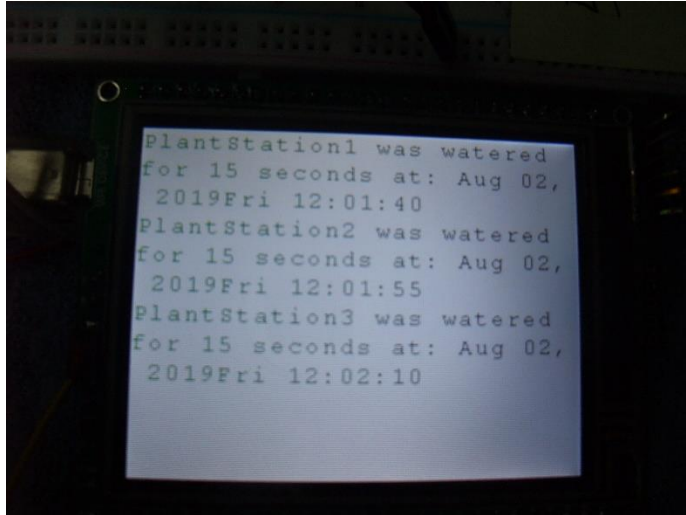
    plant3Delay = 8640;
}

    _AD1IF = 0; // clear interrupt flag
}
```

The process loops through four “if statements” checking to see if the detected soil moisture value is above a threshold, and whether the plant was watered in the last 24 hours. If the conditions are satisfied, then the appropriate output pins for the motor and associated valve are set to low to activate the corresponding relays. The relays are activated for the period of time specified by the user, and then the output pins are set high to turn off the relays. The plant delay count value is set to “8640” to prevent the same plant from being watered for at least 24 hours (10 second alarm interrupt).

Automatic House Plant Watering System

After each watering, the date and time will be output to the touch screen display. These values are taken from the RTCC via the “RTCCProcessEvents()” function in the rtcc library. This function saves the current date information to string “_date_str” and the current time information to “_time_str”.



The ADC interrupt will set an interrupt every time four values are scanned. The ADC ISR copies the ADC1BUF values to the soilMoisture[] values:

```
void __attribute__((interrupt, auto_psv)) _AD1Interrupt (void)
{
    soilMoisture[0] = ADC1BUF0;
    soilMoisture[1] = ADC1BUF1;
    soilMoisture[2] = ADC1BUF2;
    soilMoisture[3] = ADC1BUF3;

    _AD1IF = 0;    // clear interrupt flag
}
```

Also, the RTCC interrupt is triggered every 10 seconds, which will decrement any plant delay counters that are not already at zero (plants that are available to be watered with have a count value of “0”).




The process loop continues indefinitely monitoring the soil moisture of the four plant stations, and activating the relays as required.

3. Sensor and Baseline Values

As discussed above, the capacitive soil moisture sensors output voltage values between the range of approximately 2.6 volts (wet) and 1.5 volts (dry). More particularly, the ADC detects values in the range of approximately 1800 – 2100 (wet) and approximately 3100 – 3300 (dry).

Automatic House Plant Watering System

Capacitive sensor dry output baseline ~3200

 soilMoisture	int[4]	0x1788	
 soilMoisture[0]	int	0x1788	3232
 soilMoisture[1]	int	0x178A	3243
 soilMoisture[2]	int	0x178C	3197
 soilMoisture[3]	int	0x178E	3201

Capacitive sensor wet output baseline ~1900

 soilMoisture[0]	int	0x1788	1885
 soilMoisture[1]	int	0x178A	1908
 soilMoisture[2]	int	0x178C	1876
 soilMoisture[3]	int	0x178E	1919

Converting these values to voltages:

$$(3.3/4095)*3200 = 2.6 \text{ volts (12-bit mode)}$$

$$(3.3/4095)*1900 = 1.5 \text{ volts (12-bit mode)}$$

Therefore, there is only approximately a one volt swing between completely dry (sensor in air) and completely wet (sensor in water).

Based on these values, and repeated readings of the ADC values in debug mode (“watches”), I determined that a baseline value of “2900” will cause the system to water the plants when dry but will not trigger too often if the soil is only “moist.” In other words, the trigger value needs to be closer to the dry output maximum than the wet output minimum.

D. Software Issues Encountered during Design and Implementation

Initially, I had difficulty getting the touch screen images to sequence properly. I had originally placed all the screen images in the same while loop, with nested “if statements” to select the values and next images. However, I realized that this approach would not work, and that each screen needed its own “while” loop in order for the touch interrupt to be captured properly.

In addition, as I added additional modules and the image files, the code would not compile. The compiler returned numerous errors and stated that the “large model” should be used. After researching this issue, the problem is that by “default” the compiler only uses “near” addresses, and there is not enough memory space. By setting both the data and memory models to “large” in the compiler configuration, the program code would compile.

A particularly difficult problem was trying to initialize and set the RTCC/Alarm to trigger once an hour (to count to 24 hours). However, setting the Alarm Mask to 1 minute or 1 hour does not trigger an interrupt as expected. I tested the RTCC routine from the “blinking LED” lab and determined that for mask setting of 0.5 sec, 1 sec, and 10 sec the interrupts are triggered. However, setting the mask for 1 minute or 1 hour does not work (the LEDs do not blink). As a

Automatic House Plant Watering System

work around, I set the mask to 10 seconds, and then set the count to “8640” to count a 24-hour period (60sec x 60min x 24hr = 86,400; with a 10 second mask, the count is 8640).

Finally, with so many “include” files and dependencies, it became difficult to keep track of what each library module was initializing. For example, the “uMedia.c” file sets all inputs to “digital”, so I had to comment this out to prevent it from conflicting with my settings. Including the PICconfig.h conflicted with duplicate oscillator settings in the main program. Also, including rtcc.h (to have access to the time and date strings for display of each watering), caused conflicts between Generic.h and GenericTypeDef.h files (duplicate entries).

IV. Device Testing and Results

I performed a series of tests to confirm the proper operation of the system. First, using the “debug” feature in MPLAB, I was able to capture the variables input by the user. Specifically, the user input on the touchscreen is properly captured and saved in the appropriate variables.

User selected input watering durations are correctly entered and saved (15, 30, 45 and 60 seconds; shown in hex)

<input checked="" type="checkbox"/>	plant0WateringTime	int	0x179A	0x000F
<input checked="" type="checkbox"/>	plant1WateringTime	int	0x179C	0x001E
<input checked="" type="checkbox"/>	plant2WateringTime	int	0x179E	0x002D
<input checked="" type="checkbox"/>	plant3WateringTime	int	0x17A0	0x003C

Next, I captured values of the capacitive soil moisture sensors in both dry (air) and wet (water) conditions. I used these values to determine an appropriate baseline value for watering.

Capacitive sensor dry output baseline ~3200

<input checked="" type="checkbox"/>	soilMoisture	int[4]	0x1788	
<input checked="" type="checkbox"/>	soilMoisture[0]	int	0x1788	3232
<input checked="" type="checkbox"/>	soilMoisture[1]	int	0x178A	3243
<input checked="" type="checkbox"/>	soilMoisture[2]	int	0x178C	3197
<input checked="" type="checkbox"/>	soilMoisture[3]	int	0x178E	3201

Capacitive sensor wet output baseline ~1900

<input checked="" type="checkbox"/>	soilMoisture[0]	int	0x1788	1885
<input checked="" type="checkbox"/>	soilMoisture[1]	int	0x178A	1908
<input checked="" type="checkbox"/>	soilMoisture[2]	int	0x178C	1876
<input checked="" type="checkbox"/>	soilMoisture[3]	int	0x178E	1919

With both the user input, and sensor input/ADC values confirmed and verified, I performed a series of “dry” experiments to confirm the proper operation of the output control signals, relays, and motor/valves. In these experiments, everything was connected except the water supply tubes. I could confirm the operation by seeing the LED lights on the relay board, and hearing the

Automatic House Plant Watering System

motor turn on and off. I used cups filled with water to confirm the proper operation of the control logic with respect to wet and dry sensor conditions.

In the first part of the attached video demonstration, I show these “dry” experiments, and show how the relays turn on and off as expected. Specifically, if a sensor is in the “air” it should trigger the relay to water the corresponding station. If, however, the sensor is in water, the relay should not trigger. The video demonstration confirms this operation. In addition, if a sensor is originally in water, but is then removed, this should trigger an immediate watering cycle (which it does). Finally, if a station has been watered, then the sensor status should not have any immediate effect, since there is a 24-hour delay before the next watering cycle. Again, the video demonstration confirms this correct operation.

I first tested the 24-hour delay by setting the delay values for initially only 5 minutes, and then 1 hour. Once I confirmed that the logic for the watering delay would work (i.e. no watering during delay period), I then began to debug why the RTCC alarm would not trigger with a mask for one hour. Eventually, I found that the alarm mask works for 0.5 sec, 1 sec, and 10 sec, but not one minute or 1 hour. I was unable to determine why the mask for times greater than 10 seconds would not work, so I used a 10 second mask, and set the counter to 8640 (60sec x 60min x 24 hours = 86,400 seconds; using a 10 second alarm, results in a counter of 8640).

Next, I performed a series of “wet” experiments, as shown in the video. After connecting the water supply tubes to a water supply, I placed the watering tubes in tall glasses, with the sensors in air, to show that in fact the system will supply water to each station. In the test, I showed that each station could be set to a different watering duration time. Again, I also showed that if the sensors are in the water, then no watering would occur.

Finally, I showed an actual plant watering scenario with four live house plants. The main purpose of these tests was to show that the 24-hour delay functionality works in a real environment. After resetting the system, the system watered three of the four houseplants (stations 1, 2 and 3). During the 24-hour delay period, the system would not re-water the plants, even if the sensors were removed from the pots. However, after waiting 24 hours, the system could again water the plants, depending on the soil conditions (or by removing the sensor from the pot). The touch screen display shows the last watering dates and times for the system.

As a result of the testing, I confirmed that the sensor outputs, ADC processing, user input, and control logic all function as designed and expected. Further, the 24-hour delay mechanism prevented another watering within a 24-hour period, but after the 24 hours, the plants can be watered again.

V. Conclusion

While the initial idea for an automatic house plant watering system did not seem particularly complex, getting all the components, especially the software modules, to work together was challenging.

Automatic House Plant Watering System

The first lesson I learned from the hardware design was to make sure the ADC is operating on input signals with a common ground (otherwise, the values will float). The other main lesson is that activating outputs with an “active low” signal can be difficult, and it is important to set any output pins to a desired state even before initializing the pins.

With respect to the software, I learned that each touch screen should have its own “while” loop in order to properly catch a valid screen touch.

In addition, I spent a considerable amount of time debugging the 24-hour delay function. I was not able to get the system to acknowledge the 1-hour alarm setting and process the interrupt. After significant experimentation, I determined that the RTCC alarm mask works properly for 0.5 sec, 1 sec, and 10 sec. However, for 1 minute and 1 hour, the interrupt is never processed. I confirmed this behavior using the “blinking LED” lab example. In the end, the work around was just to use the 10 second alarm and set the counts to generate a 24 hour watering delay.

The final system has been fully debugged and tested and meets all of the initial design goals. In addition, I added a feature to display the date and time on the display each time a plant station is watered. The full project code is attached hereto as APPENDIX A.

Automatic House Plant Watering System

APPENDIX A

```
/*
 * File:    FinalProject.c
 * Author:  Doyle Johnson
 *
 * Created on July 15, 2019, 12:52 PM
 */

#include <xc.h>
#include <p24EP512GU810.h>

#include "PICconfig.h"
#include "LCDterminal.h"
#include "TouchScreen.h"

#include "uMedia.h"
#include "Graphics/Graphics.h"
#include "resources.h"
#include "string.h"

int soilMoisture[4];
int plant0Delay = 0;
int plant1Delay = 0;
int plant2Delay = 0;
int plant3Delay = 0;
int plant0WateringTime = 15; // 15 seconds watering duration default
int plant1WateringTime = 15;
int plant2WateringTime = 15;
int plant3WateringTime = 15;
int plant0Threshold = 2900; //wet baseline ~1900; dry baseline ~3200
int plant1Threshold = 2900;
int plant2Threshold = 2900;
int plant3Threshold = 2900;
int finished = 0;
char s[128];
char _time_str[16];
char _date_str[16];

#define TICK_PERIOD( ms)  (GetPeripheralClock() * (ms)) / 8000

void Init(void)
{
    RCONbits.SWDTEN=0;                // Disable Watch Dog Timer

    // Configure Oscillator to operate the device at 40Mhz
    // Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
    // Fosc= 8M*40/(2*2)=80Mhz for 8M input clock
    PLLFBD=38;                        // M=40
```

Automatic House Plant Watering System

```
CLKDIVbits.PLLPOST=0;          // N1=2
CLKDIVbits.PLLPRE=0;          // N2=2
OSCTUN=0;                     // Tune FRC oscillator, if FRC is used

// Clock switching to incorporate PLL
__builtin_write_OSCCONH(0x03); // Initiate Clock Switch to
Primary                        // Oscillator with PLL (NOSC=0b011)
                               // Start clock switching
__builtin_write_OSCCONL(0x01); while (OSCCONbits.COSC != 0b011); // Wait for Clock switch to
occur                          // Wait for PLL to lock
while(OSCCONbits.LOCK!=1) {}; } // Init

void InitRTCC( void)
{
    // unlock and enable SOSC
    __builtin_write_OSCCONL( 2);

    // unlock RCFGAL, set RTCWREN
    __builtin_write_RTCWEN();

    _RTCEN = 0;          // disable the clock

    // set 08/02/2019 FRI 12:00:00
    _RTCPTR = 3;         // start the sequence
    RTCVAL = 0x2019;     // YEAR
    RTCVAL = 0x0802;     // MONTH/DAY
    RTCVAL = 0x0512;     // WEEKDAY/HOURS
    RTCVAL = 0x0000;     // MINUTES/SECONDS

    // optional calibration
    //_CAL = 0x00;

    // lock and enable
    _RTCEN = 1;          // start the clock
    _RTCWREN = 0;        // lock
} // InitRTCC

void SetALARM( void)
{
    // disable alarm
    _ALRMEN = 0;

    // set the repeat counter
    _ARPT = 0;           // once
    _CHIME = 1;          // no reload
```

Automatic House Plant Watering System

```
// set the alarm mask
_AMASK = 0b0010;    // alarm every 10 seconds

_ALRMEN = 1;        // enable alarm

_RTCIF = 0;         // clear interrupt flag

_RTCIE = 1;         // enable interrupt
} // Set ALARM

void _ISR_RTCCInterrupt( void)
{
    // will call this routine every 10 seconds; plant only watered
    once in 24 hour period

    if (plant0Delay > 0)
    {
        plant0Delay--;
    }
    if (plant1Delay > 0)
    {
        plant1Delay--;
    }
    if (plant2Delay > 0)
    {
        plant2Delay--;
    }
    if (plant3Delay > 0)
    {
        plant3Delay--;
    }

    _RTCIF = 0;      // clear flag
} // RTCC interrupt service routine

// initialize pins
void InitPins(void)
{
    _RD8=1;
    _RD9=1;
    _RD10=1;
    _RD11=1;

    _RC4=1;

    ANSELB = 0xFFFF; // set PORTB to analog
    _TRISB0 = 0x1;    // set PORTB pins to input
}
```

Automatic House Plant Watering System

```
_TRISB1 = 0x1;
_TRISB2 = 0x1;
_TRISB3 = 0x1;
ANSELD = 0x0000; // set PORTD to digital
_TRISD8 = 0x0;   // set PORTD pins to output
_TRISD9 = 0x0;
_TRISD10 = 0x0;
_TRISD11 = 0x0;

ANSELC = 0x0000; // set PORTC to digital
_TRISC4 = 0x0;   // set PORTC pin to output

}

// initialize the ADC
void InitADC(void)
{
    AD1CON1 = 0x04E4; // 12-bit mode; auto-sample; auto-convert
    AD1CON2 = 0x040C; // AVss and AVdd; scan; interrupt after 4th
sample
    AD1CON3 = 0x0002; // ADCS = 2 (76ns TAD)
    AD1CON4 = 0x0000; // ADDMAEN off; values stored in ADC1BUF

    AD1CSSL = 0x000F; // scan AN0 - AN3

    _AD1IF = 0; // Clear the A/D interrupt flag bit
    _AD1IE = 1; // Enable A/D interrupt

    _ADON = 1; // turn on ADC
} // initADC

#define __ISR __attribute__((interrupt, shadow, no_auto_psv))

void __ISR _T3Interrupt( void)
{
    _T3IF = 0;
    TouchDetectPosition();
}

void TickInit( unsigned period_ms)
{
    // Initialize Timer3
    TMR3 = 0;
    PR3 = TICK_PERIOD( period_ms);
    T3CONbits.TCKPS = 1; // Set prescale to 1:8
    IFS0bits.T3IF = 0; // Clear flag
}
```

Automatic House Plant Watering System

```
    IEC0bits.T3IE = 1;          // Enable interrupt
    T3CONbits.TON = 1;          // Run timer
}

void InitDisplay(void)
{
    // init the graphics
    LCDInit();
    DisplayBacklightOn();

    // init the touch timer
    TickInit( 1);

    // init touch module (do not use NVM to store calibration data)
    TouchInit( NULL, NULL, NULL, NULL);

    // splash screen
    LCDClear();

    PutImage(0, 0, (void*) &intro, IMAGE_NORMAL);

    while(!finished)
    {
        if (( TouchGetX() > 230) && ( TouchGetX() < 310) && ( TouchGetY() >
210)
                && ( TouchGetY() < 240))
        {
            PutImage(0, 0, (void*) &screen0, IMAGE_NORMAL);
            finished = 1;
        }
    }
    finished = 0;

    while(!finished)
    {
        if (( TouchGetX() > 15) && ( TouchGetX() < 140) && ( TouchGetY() > 50)
                && ( TouchGetY() < 115))
        {
            plant0WateringTime = 15;
        }

        if (( TouchGetX() > 15) && ( TouchGetX() < 140) && ( TouchGetY() >
130)
                && ( TouchGetY() < 195))
        {
            plant0WateringTime= 30;
        }
    }
}
```

Automatic House Plant Watering System

```
if (( TouchGetX() > 165) && ( TouchGetX() < 290) && ( TouchGetY() >
50)
    && ( TouchGetY() < 115))
{
    plant0WateringTime= 45;
}

if (( TouchGetX() > 165) && ( TouchGetX() < 290) && ( TouchGetY() >
130)
    && ( TouchGetY() < 195))
{
    plant0WateringTime= 60;
}

if (( TouchGetX() > 230) && ( TouchGetX() < 310) && ( TouchGetY() >
210)
    && ( TouchGetY() < 240))
{
    PutImage(0, 0, (void*) &screen1, IMAGE_NORMAL);
    finished = 1;
}
}
finished = 0;

while(!finished)
{

if (( TouchGetX() > 15) && ( TouchGetX() < 140) && ( TouchGetY() > 50)
    && ( TouchGetY() < 115))
{
    plant1WateringTime = 15;
}

if (( TouchGetX() > 15) && ( TouchGetX() < 140) && ( TouchGetY() >
130)
    && ( TouchGetY() < 195))
{
    plant1WateringTime= 30;
}

if (( TouchGetX() > 165) && ( TouchGetX() < 290) && ( TouchGetY() >
50)
    && ( TouchGetY() < 115))
{
    plant1WateringTime= 45;
}

if (( TouchGetX() > 165) && ( TouchGetX() < 290) && ( TouchGetY() >
130)
    && ( TouchGetY() < 195))
{
    plant1WateringTime= 60;
}
```


Automatic House Plant Watering System

```
    }

if (( TouchGetX() > 230) && ( TouchGetX() < 310) && ( TouchGetY() >
210)
    && ( TouchGetY() < 240))
{
    PutImage(0, 0, (void*) &screen2, IMAGE_NORMAL);
    finished = 1;
}
finished = 0;

while(!finished)
{
    if (( TouchGetX() > 15) && ( TouchGetX() < 140) && ( TouchGetY() > 50)
        && ( TouchGetY() < 115))
    {
        plant2WateringTime = 15;
    }

    if (( TouchGetX() > 15) && ( TouchGetX() < 140) && ( TouchGetY() >
130)
        && ( TouchGetY() < 195))
    {
        plant2WateringTime= 30;
    }

    if (( TouchGetX() > 165) && ( TouchGetX() < 290) && ( TouchGetY() >
50)
        && ( TouchGetY() < 115))
    {
        plant2WateringTime= 45;
    }

    if (( TouchGetX() > 165) && ( TouchGetX() < 290) && ( TouchGetY() >
130)
        && ( TouchGetY() < 195))
    {
        plant2WateringTime= 60;
    }

    if (( TouchGetX() > 230) && ( TouchGetX() < 310) && ( TouchGetY() >
210)
        && ( TouchGetY() < 240))
    {
        PutImage(0, 0, (void*) &screen3, IMAGE_NORMAL);
        finished = 1;
    }
}
finished = 0;
```

Automatic House Plant Watering System

```
while(!finished)
{
    if (( TouchGetX() > 15) && ( TouchGetX() < 140) && ( TouchGetY() > 50)
        && ( TouchGetY() < 115))
    {
        plant3WateringTime = 15;
    }

    if (( TouchGetX() > 15) && ( TouchGetX() < 140) && ( TouchGetY() >
130)
        && ( TouchGetY() < 195))
    {
        plant3WateringTime= 30;
    }

    if (( TouchGetX() > 165) && ( TouchGetX() < 290) && ( TouchGetY() >
50)
        && ( TouchGetY() < 115))
    {
        plant3WateringTime= 45;
    }

    if (( TouchGetX() > 165) && ( TouchGetX() < 290) && ( TouchGetY() >
130)
        && ( TouchGetY() < 195))
    {
        plant3WateringTime= 60;
    }

    if (( TouchGetX() > 230) && ( TouchGetX() < 310) && ( TouchGetY() >
210)
        && ( TouchGetY() < 240))
    {
        PutImage(0, 0, (void*) &start, IMAGE_NORMAL);
        finished = 1;
    }
}
finished = 0;

while(!finished)
{
    if (( TouchGetX() > 60) && ( TouchGetX() < 260) && ( TouchGetY() > 85)
        && ( TouchGetY() < 150))
    {
        PutImage(0, 0, (void*) &watering, IMAGE_NORMAL);
        finished = 1;
    }
}
}
```

Automatic House Plant Watering System

```
void __attribute__((interrupt, auto_psv)) _AD1Interrupt (void)
{
    soilMoisture[0] = ADC1BUF0;
    soilMoisture[1] = ADC1BUF1;
    soilMoisture[2] = ADC1BUF2;
    soilMoisture[3] = ADC1BUF3;

    _AD1IF = 0;    // clear interrupt flag
}

void DelaySec( long int t)
{
    t=t*1000;
    T1CON = 0x8000;    // enable tmr1, Tcy, 1:1
    while (t>0)        // wait for t (msec)
    {
        TMR1 = 0;
        while ( TMR1 < (40000)); // wait 1ms
        t=t-1;
    }
} // DelaySec

int main( void )
{
    Init();
    InitPins();
    InitDisplay();
    InitADC();
    InitRTCC();
    SetALARM();

    while(1)
    {
        DelaySec(10);
        _AD1IF = 1;    // set interrupt flag to process output

        if ((soilMoisture[0] > plant0Threshold) && (!plant0Delay))
        {
            _RD8 = 0;    // valve
            _RC4 = 0;    // motor/pump
            DelaySec(plant0WateringTime);
        }
    }
}
```

Automatic House Plant Watering System

```
_RD8 = 1;
_RC4 = 1;

RTCCProcessEvents();
sprintf(s, "PlantStation1 was watered for %d",
plant0WateringTime);
LCDPutString(s);
sprintf(s, " seconds at: ");
LCDPutString(s);
LCDPutString(_date_str);
LCDPutString(_time_str);
sprintf(s, "\n\r");
LCDPutString(s);

plant0Delay = 8640;
}

if ((soilMoisture[1] > plant1Threshold) && (!plant1Delay))
{
    _RD9 = 0; // valve
    _RC4 = 0; // motor/pump
    DelaySec(plant1WateringTime);
    _RD9 = 1;
    _RC4 = 1;

    RTCCProcessEvents();
    sprintf(s, "PlantStation2 was watered for %d",
plant1WateringTime);
    LCDPutString(s);
    sprintf(s, " seconds at: ");
    LCDPutString(s);
    LCDPutString(_date_str);
    LCDPutString(_time_str);
    sprintf(s, "\n\r");
    LCDPutString(s);

    plant1Delay = 8640;
}

if ((soilMoisture[2] > plant2Threshold) && (!plant2Delay))
{
    _RD10 = 0; // valve
    _RC4 = 0; // motor/pump
    DelaySec(plant2WateringTime);
    _RD10 = 1;
    _RC4 = 1;

    RTCCProcessEvents();
    sprintf(s, "PlantStation3 was watered for %d",
plant2WateringTime);
    LCDPutString(s);
    sprintf(s, " seconds at: ");
```

Automatic House Plant Watering System

```
LCDPutString(s);
LCDPutString(_date_str);
LCDPutString(_time_str);
sprintf(s, "\n\r");
LCDPutString(s);

plant2Delay = 8640;
}

if ((soilMoisture[3] > plant3Threshold) && (!plant3Delay))
{
    _RD11 = 0; // valve
    _RC4 = 0; // motor/pump
    DelaySec(plant3WateringTime);
    _RD11 = 1;
    _RC4 = 1;

    RTCCProcessEvents();
    sprintf(s, "PlantStation4 was watered for %d",
plant3WateringTime);
    LCDPutString(s);
    sprintf(s, " seconds at: ");
    LCDPutString(s);
    LCDPutString(_date_str);
    LCDPutString(_time_str);
    sprintf(s, "\n\r");
    LCDPutString(s);

    plant3Delay = 8640;
}

    _AD1IF = 0; // clear interrupt flag
}

return 0;

}
```

APPENDIX B

