Embedded Systems Exam Bonus Project

Project name: Green Gardening

Contributers: Leander Lauenburg, Christoph Clement, Anselm Coogan

18.01.18

Contents

1 Hardware

1.1 CY8CKIT-042 PSoC 4 Pioneer Kit

Cypress' CY8CKIT-042 PSoC 4 Pioneer Kit is used as the basis for the project. We decided to use Cypress' product because of several reasons. First, the father of a team member works for Cypress and we got the Pioneer Kit for free. Second, Cypress' PSoC Creator Software to program the kit is a free IDE which features hardware and software co-design and a library with pre-characterized PSoC components. Third, Cypress offers the PSoC 101 Video Tutorial Series which demonstrates how to use the development kit and offers valuable guiding for the first steps of embedded programming. All contributors watched the videos and could learn a lot from them.

The kit contains the PSoC 4 Pioneer board, a quick start guide, a USB Standard-A to Mini-B cable and six jumper wires.

The Pioneer Kit consists of the following blocks:

- PSoC 4
  - 32-bit MCU subsystem, 48 MHz ARM Cortex-M0 CPU, 32 KB flash, 4 KB SRAM
  - Programmable analog: Two opamps, 12-bit SAR ADC
  - Low power 1.71 to 5.5 V operation
  - Serial communication via I2C, SPO and UART
  - Timing and pulse-width modulation
- PSoC 5LP
  - used to program and debug PSoC 4
  - 32-bit ARM Cortex-M3 CPU core
  - Precision, programmable clocking: 32.768 kHz watch crystal oscillator
- Power supply system
- PSoC 5LP GPIO header (J8)
- Programming interfaces (J6, J7, J10)
- CapSense slider
- Arduino compatible headers
- Pioneer board LEDs
- Digilent Pmod compatible header (J5)
- Push buttons (Reset and User buttons)

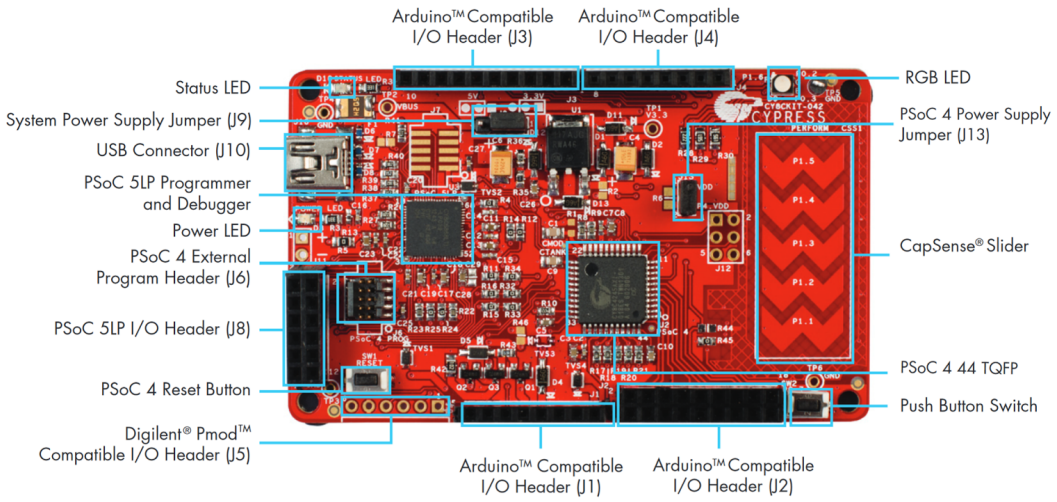

Figure 1: PSoC 4 Pioneer Board

1.2 Kuman moisture sensor

The moisture module is used to detect the moisture content of the soil. If the soil is too dry the electrical pump is switched on. It is possible to adjust the threshold of the soil humidity with the potentiometer. The digital output D0 is directly connected to the pioneer kit via a digital input pin. The operating voltage is 3.3 V to 5 V and is supplied from the pioneer kit.

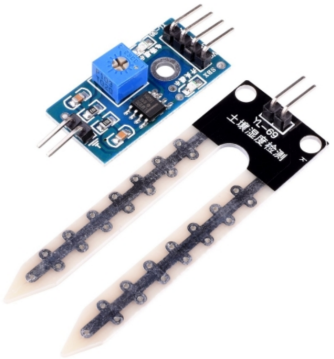

Figure 2: Kuman moisture sensor

1.3 Arduino light sensor LM393

The light sensor works pretty similar to the moisture sensor. It detects the brightness of the surroundings. If there is not enough sunlight the LEDs are turned on. The threshold can be adjusted with the potentiometer. The digital output pin D0 is connected to the pioneer kit via a digital input pin. The operating voltage is 3.3 V to 5 V and is supplied from the pioneer kit.

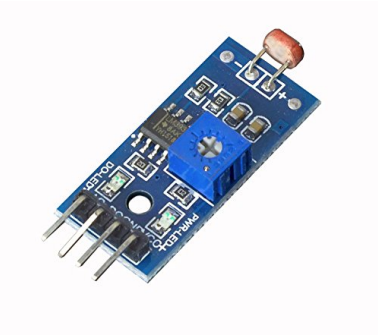

Figure 3: Arduino light sensor LM393

1.4 LE Trafo 12V DC

The 12V Trafo is used as a power supply for the LED stripes and brushless mini water pump. Its input voltage is 100-240 V AC, 50 - 60 Hz and output voltage 12 V DC, 2A.

1.5 SainSmart 4 channel relay module

The SainSmart relay module allows the PSoC to control larger loads and devices like the 12 V LED stripes and the brushless water pump with a digital output pin. The operating voltage is 5 V and each relay needs 15-20 mA driver current.

1.6 TSSS brushless mini water pump

The brushless mini water pump is switched on for 10 seconds via a digital output pin from the PSoC that controls the relay if the soil is too dry. It pumps water from a basket and moistures the soil. The operating voltage is 12 V and it needs 400 mA driver current. This is supplied by the 12 V Trafo.

1.7 Neuftech LED strips blue and red

The LED strips in blue and red work similar to the water pump. They are switched on via a digital output pin from the PSoC that controls the relay if there is not enough sunlight. The operating voltage is 12 V. This is supplied by the 12 V Trafo.

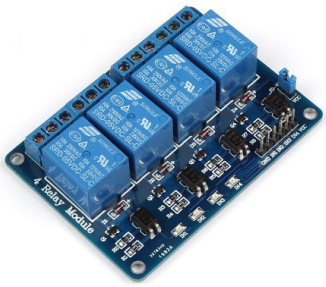

Figure 4: SainSmart 4 channel relay module

1.8 Plant

The most important part for our project: a nice plant that will hopefully survive for a long time with the help of all the other parts.

1.9 Schematic

TODO

2 Software

As stated in the previous chapter we used the free IDE *PsoC Creator* to program the micro controller. This software offers a three layer approach.

In the first layer, the top design, you can graphically configure the technical layout and wiring of the components you are using from the micro controller. In the second layer you assign the actual pins of the board to the different components configured in the top design. Lastly, the third layer is writing the code that is executed on the micro controller.

The entire source code discussed here (and successive versions) can be inspected in this git repository: https://github.com/AnselmC/GreenGardening

2.1 Top design

The top design enables you to easily make a circuit diagram for your project. You have access to all the actual components of the micro controller, as well as so-called off-board components that are there for documentation and clarification purposes only (colored blue). The components we used for this project are: a real-time clock, a universal asynchronous receiver transmitter, and digital input and output pins.

2.1.1 Real-time clock

The built-in real-time clock component can be configured extensively and has a resolution of one second. We set the clock to use the 24-hour time format and synched it to Central European Time.

2.1.2 UART

The universal asynchronous receiver transmitter component allows for setting up a bidirectional data connection on a given port. We are using this for debugging purposes.

2.1.3 Digital input pins

The input pins in our project are the light sensor and the moisture sensor. We configured these input pins to have dedicated interrupts. The interrupt for the light sensor is activated on both falling and rising edges, i.e. from light to dark and vice versa. This is because the light sensor is situated so that it is independent of the LED strips and only senses the environment light. Thus, if the LED's have been switched on because it was previously too dark and now the surrounding light suffices the LED's should be switched off. In contrast the interrupt for the moisture sensor is only activated on a falling edge, i.e. when it gets too dry.

2.1.4 Digital output pins

As output pins we have set up the water pump and the LED strips.

2.2 Pin connections

The second layer is actually called *Design Wide Resources*. However, we are only using the pin assignment part in our project. Here you get an overview of the micro controller and can then assign the components configured in the top design with the components on the board and the components that have been connected via the digital or analog GPIO pins.

2.3 C Code

PsoC Creator automatically generates an API from the top design previously configured. This allows you to control all components with very little self-written C code. In our main.c we have defined the following global variables used throughout our code:

- WATERING_TIME_IN_SEC: the time the watering pump should be on for one watering session
- STARTING_HOUR and ENDING_HOUR: the timeframe in which the plant should be exposed to light - in 24 h format (e.g. from STARTING_HOUR: 10 to ENDING_HOUR 20)

- NUMBER_OF_MEASUREMENTS: The number of measurements a sensor should make in one setting
- TIME_BETWEEN_MEASUREMENTS_MILSEC: The time (in milliseconds) to wait between the single measurements
- BUFFER_SIZE: The array size of the buffer used for writing debug information to the console
- ON and OFF: Macros used for more clarity in code

In order to use less CPU time and power, we are working with interrupts. There are two interrupts: one for the moisture sensor and one for the light sensor. The interrupts are triggered on both falling and rising edges of the sensors. This means that whenever the light changes from too dark to light or vice versa, or the moisture sensor changes from not moist enough to moist and vice versa the interrupts are called. In PsoC Creator these interrupts are configured in main.c as follows (for an arbitrary pin named Pin_A):

```
CY_ISR(<<name_of_interrupt_handler>>)
{
    /* Code to be executed on interrupt */

    Pin_A_ClearInterrupt();
}
```

The name of the interrupt handler can be whatever you choose. However, best practice is to name it *Pin_A_Handler*. It is essential to clear the interrupt after the code has been executed, otherwise the interrupt can?t be called again. This is done with the generated *Pin_A_ClearInterrupt()* function. In order to enable interrupt handling, in your *main()* method you will need to call *CyGlobalIntEnable*. Then, you need to initialize every interrupt like this: *Pin_A_int_StartEx(Pin_A_Handler)* The StartEx function of your interrupt-enabled pin is also automatically generated by PsoC Creator. This means that in our project the *main()* function solely consists of initializing the interrupts and an endless for-loop with no-code. Thus, the CPU can sleep until one of the sensors interrupts it.

```
int main(void)
{
    /* Enable global interrupts. */
    CyGlobalIntEnable;

    /* Place initialization/startup code here */
    UART_Start();
    RTC_Start();
    Pin_Light_Sensor_int_StartEx(Pin_Light_Sensor_Handler);
    Pin_Moisture_Sensor_int_StartEx(Pin_Moisture_Sensor_Handler);
```

```
    for(;;)
    {

    }
}
```

Our two interrupt functions each call one function, namely *checkLight()* and *checkMoisture()*. *checkLight()* first calls the real-time clock component to return the current time and then extracts the current hour. It then checks whether the returned hour is in the designated timeframe. If it isn't, the light is turned off and the function terminates. If it is, then we take a pre-defined amount of measurements (as configured in the global variables) with a pre-defined delay between each measurement. If the light sensor only measures enough light for less than half of the measurements, the LED's are turned on and the function terminates. Else, the lights are turned off and the function terminates.

```
void checkLight()
{
    /* Get current hour */
    int hour = RTC_GetHours(RTC_GetTime());

    /* Check whether hour is outside of designated time frame
    if so, turn it off */
    if(hour < STARTING_HOUR || hour > ENDING_HOUR){
        Pin_LED_blue_Write(OFF);
    }
    else {
        int positive_measurements = NUMBER_OF_MEASUREMENTS;
        /* Make several measurements of the light sensor */
        for(int i = 0; i < NUMBER_OF_MEASUREMENTS; i++){
            positive_measurements -= Pin_Light_Sensor_Read();
            CyDelay(TIME_BETWEEN_MEASUREMENTS_IN_MILSEC);
        }
        /* If the light sensor catches not enough light,
        turn on the LED's */
        if (positive_measurements < NUMBER_OF_MEASUREMENTS/2)
        {
            Pin_LED_blue_Write(ON);
        }
        /* Else turn them off */
        else
        {
            Pin_LED_blue_Write(OFF);
        }
```

```
    }
}
```

The *checkMoisture()* function operates similarly. Again, we take a pre-defined amount of measurement to rule out anomalies. Then, if less than half of these measurements indicate that the soil is too dry, the water pump is turned on for a specified amount of time and then turned off again after which the function terminates. If the soil is moist enough, nothing happens and the function terminates.

```
void checkMoisture()
{
    int positive_measurements = NUMBER_OF_MEASUREMENTS;
    /* Make several measurements of the moisture sensor */
    for(int i = 0; i < NUMBER_OF_MEASUREMENTS; i++){
        positive_measurements -= Pin_Moisture_Sensor_Read();
        CyDelay(TIME_BETWEEN_MEASUREMENTS_IN_MILSEC);
    }

    /* If the ground isn't wet enough,
    water the plant for a specified amount of time */
    if (positive_measurements < NUMBER_OF_MEASUREMENTS/2)
    {
        /* Turn pump on */
        Pin_pump_Write(ON);
        /* Wait for specified amount of watering time */
        CyDelay(WATERING_TIME_IN_SEC * 1000);
        /* Turn pump off */
        Pin_pump_Write(OFF);
    }
}
```

Throughout the code we are outputting debug information that was critical for development. For this we are using the UART component. To use it it must be be initialized in the *main()* method as can be seen above. Then, when you want to write to the connected pod (e.g. a terminal on your development machine) you can do so like this:

```
sprintf(buffer, "Let's go\r\n");
UART_UartPutString(buffer);
```

3 Learnings from the lecture

The lecture related to is a union of the two lectures Embedded Systems and Real Time Systems, as they are closely related. How strong these two subject interleave became as much clearer while releasing this Project.

3.1 The System

According to the lecture an embedded system is a system which has a particular function within a larger mechanical or electrical system, often with real-time computing constraints. It is therefore quite far-fatched to call the Modul in use embedded. The Modul is the only and therefore main logic in our system and most definitely isn't the system a ?larger? mechanical system. But since the system does has real-time computing constraints and the System is just in the making, expecting expansion through hardware and additional logic parts in the near future, it will from her on be referred to as an Embedded System. However at this moment the system is a synchronous digital circuit since ?all? logic is connected to the same clock. A real-time System is defined as a computer system which accuracy doesn?t only depend on the logical result but also on the physical point in time by which the result is produced.

Since certain tasks of our system depend on the time of the day it has to be referred to as ?real-time System? according to this definition, also the author probably had a bit smaller unit for the time in mind. At the moment we are working on extending the systems functionality by making use of the PSoC?s integrated Bluetooth interface. Thereby transforming the system to a smart device making it controllable via smartphone as well as graphically representing the accumulating data on the phone. By doing so the system will become a Cyber-physical systems in accordance with the definition of Cyber-physical systems from the lecture.

3.2 Operator Cluster

The human who makes use of the system, calibrating it to the specifics of the plant and turning the system on and off.

3.3 Computational Cluster

The _____ Modul from Cypress.

3.4 Controlled Cluster

Is made up of the sensors as well as the light and the pump.

3.5 The Logic

Since ?embedded System? refers to how a logic device is integrated in a larger system and not the device itself it can refer to just about every logic device. All these different devices were categorized in the lecture. The module in use (for more information on the model refer to the hardware chapter) is an ?Programmable System on a Chip? (PSoC). We choose the module inspired by a reference from the lecture referring to its great flexibility as well as great compatibility. Additionally we were really keen on trying out the IDE PSoC creator which was also referred to in the lecture. At the beginning of our project we sat down analysing the module with the information given to us by the

lecture. We recognised the real time capability of the ARM Cortex-M0 CPU through characteristic like the on board real time clock supported by an on board button cell and the by default missing cache. From the lecture we knew that also most of the time a cache increases the performance of the device it can in extreme cases lead to a greater worst case runtime making timing issues difficult to analyse.

3.6 Real time

As stated already in the beginning of this chapter the System can be categorized as a real time system. This is at moment manly do to the implementation of the lighting system. The implementation of the lighting system as it is is a combination of a ?Time triggered applications? and an ?Event-driven applications?. The light is supposed to switch on whenever the light sensor signals that it is to dark. Therefore it is clearly event driven. But since a plant naturally also needs a time during which it is dark the signal of the light sensor should only be incorporate during certain times of the day. In the extreme case of permanent total darkness the light will always be switch on over a constant time period during a certain time of the day. In this case you can without objection say that the entire time cycle was determined during compile time thereby making it a pure time triggered application. It would be possible to realise the system through purely time triggered applications. But this would lead to an reduction in flexibility and effectiveness of the system since parameters for temperature and humidity would have to be known already during compile time. These parameters would also have to be final since time triggered applications by definition don?t react to external signals and are therefore inflexible. The dependency on the time of the day is what makes the system a real time system. Thanks to an on board real time clock the modul can guarantee the accuracy of the system. A button cell powers the clock during times in which the modul isn?t connected to its main power supply, for example during transport. It would be preferable to use an mor accurate external quartz crystal oscillators to minimize the drift. But since most plants are quite durable the don't require such high precision for when the light shines and when not. This leads directly to the question of what kind of real time system we are dealing with. For hard real time systems the lecture states:

?A violation of the defined deadlines can immediately lead to severe consequences?

A plant pretty much doesn't care when it is watered and at what time of the day it gets light as long as it gets enough of both in the end of the day. Therefore if the system fails to react instantaneously to a signal of one of the sensors or the clock there won't be any severe consequences. For soft real time systems the lecture states among other things: ?...an overrun of those deadlines is not catastrophic...? and ?Overall it is possible that the breach of the time limit will produce a degradation in service...? The system is therefore clearly a soft real time system. Since a miss of a deadline can in the worst case only result in a degradation of growth of the plant.

List of Figures