ECE3375: Design Project

Objective

The design project for ECE3375 is intended to help you go through the design process for an embedded system. This project is not intended to be especially time-intensive, and we do not expect anyone to deliver a working prototype. Rather, the project is intended as practice designing and describing an embedded system and providing a plan for prototyping software. Hopefully this project will help you develop skills that will make your capstone design project easier to complete.

You are expected to implement at least part of your design as a software prototype using the DE10-Standard development board and/or the DE1-SoC simulator.

This work is to be completed in groups of at least two and at most four. Ideally these groups will be interdisciplinary, with some members with a stronger software background and other members with a stronger hardwarebackground, but this is left up to you.

Deliverables

The project should be completed in stages.

- A **Progress Report**, due last week of **March (see OWL course site)**. This document should be submitted on OWL. Only one member of the group needs to submit. This report will not be graded but will be reviewed to see if your design and methodology is appropriate.
- A **Demonstration** of your software prototype, to be scheduled with the TAs.
- A **Final Report**, due first week of **April (see OWL course site)**. This document should be submitted on OWL. Only one member of the group needs to submit.

It is important to recognize that this project is all about embedded system *design* and *implementation*, **not** making a mechatronic system. If you have the time, resources, and inclination you are welcome to try and implement your system in real hardware, but **no marks** will be awarded for this.

Student work in ECE3375 needs to assess whether students are aware of the *impact of engineering on society and the environment*, so some aspects of your reports should address this (as described in greater detail below). I know, it is a bit contrived, but it is an important part of the ECE program's accreditation so please take it somewhat seriously.

Design Approach

You should complete this project through the following stages:

1. Problem Definition.
2. Functional Description
3. Identify Input/Output Requirements.
4. Initial Software Design.
5. Prototyping Plan.

These first five stages should be described in your **Progress Report**.

6. Select a Microcontroller.
7. Revised Software Design.
8. Results from Prototyping.
9. Source Code.
10. Conclusions.

Your **Final Report** should include the first stages from your **Progress Report** and additionally describe these final five stages. All these stages are described in more detail below.

Problem Definition

The basic objective with this assignment is to give you a chance to design "something". It really does not matter what it is, as long as it is an **engineered system with embedded intelligence**. The nature of the world we live in is such that almost every engineered system can make use of embedded computing in some way, so you have a lot of freedom here.

To help you come up with some ideas, here are some projects that fourth-year students have worked on in recent years (under the supervision of Prof. McIsaac, who previously taught this course). This is definitely **not** an exhaustive list. You **may** choose a project from this list, but you are also welcome to come up with your own ideas.

- Wearable system to detect local "wind chill" factor while skiing.
- System to set the shower temperature and pressure automatically for a given user.
- RFID-based door lock/light switch.
- Automated system to turn on and off room lights based on occupancy.
- Motor controller for electric bicycle.
- Home thermostat with multiple, remotely located temperature sensors.
- System to lock out the stall and alert the custodial staff when the toilet paper runs out.

If you have any doubts about the suitability of your own idea, feel free to contact course instructors, or the head TA of the course.

As mentioned above, we would like you to include a brief discussion of how this device would change the lives of your users.

- What sorts of tasks would it make easier, more efficient or more pleasant?
- Can you imagine any drawbacks?
- Would other people around the user be affected?

Try to imagine how it would impact the lives of your users in their various contexts (home, work, commute, school, sports, whatever).

Deliverable*:* Describe the problem and provide the specifications to be met by the embedded system.

Deliverable*:* Discuss the impact on the user and society.

Functional Description

For many of you, this will come at the same time as problem definition, but it is important to make sure it gets done. Rather than describing what your system is supposed to do for a user, instead think more specifically about how your system interacts with the user and monitors the environment. The objective here is for you to answer questions like the following:

- What does the system do?
- How does a user interact with it?
- What other devices (if any) does it interact with? How does that happen?
- What quantities does it monitor? How often?
- What stimuli does it manage? How often?

Essentially, try to imagine your finished system as a "black box". Without getting into the details of the circuitry, mechanical parts, or code, how would you describe what is inside the box?

Deliverable*:* A functional description of your systems. This can be mostly diagrams if you prefer, but it can equally well be mostly written.

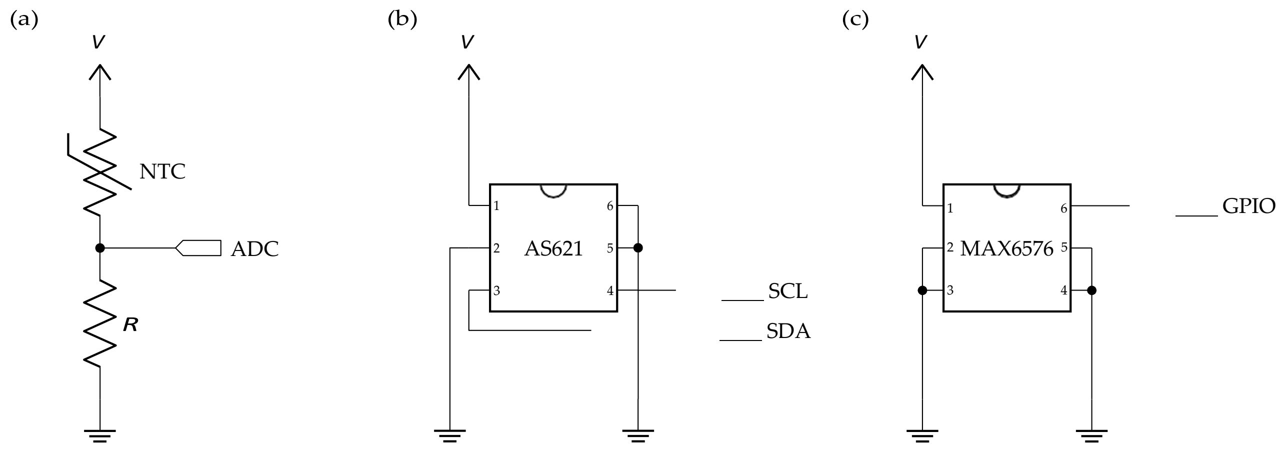
Identify Input/Output Requirements

Once you have identified a problem and you understand its functional requirements, you will need to determine what inputs and outputs you require. Inputs may take the form of analog values, digital values, user inputs, communication inputs and so on. Outputs may take the form of analog outputs, digital outputs, PWM outputs, communication outputs and so on. It is difficult to provide guidance in this area, because the range of possible applications is so broad. Basically, we proceed as follows:

- First, determine what you have to measure, and what you have to do. This can be left in the application domain (ie: "If the water is too hot, I need to cool it down").
- Second, identify what transducers (sensors and actuators) can be used to accomplish this objective.

In many cases, all you need is Google to find a whole lot of transducers. Digikey is also a great resource. You will find that in almost all cases, no matter what you want to measure or control, you have a choice between buying a simple transducer that requires you to do much of the wiring, signal conditioning and calibration **or** a digital system that already contains most of the wiring but requires you to understand some kind of communication protocol.

Example*:* Assume your system — whatever it is — needs to measure temperature (among other things). The DE10-Standard does not have a built-in temperature peripheral, however there are hundreds of electronic components that can perform the required task. Among these, three options are (a) a voltage divider using an NTC thermistor, (b) an AS621 digital temperature sensor, or (c) a MAX6576 digital temperature sensor.



Each of these options will measure temperature but require different connections to the microcontroller and different degrees of software processing to obtain the temperature.

Deliverable: Identify your inputs and outputs. Include in your discussion what tools or circuits you will need to interface to the transducers. Identify and select the appropriate circuit components, tools and resources for your design.

Initial Software Design

By now you should have a well-defined problem and some idea of the electronics needed. You don't have a microcontroller selected yet, but you will have one and you know what they are. So, all you need now is software. Provide a document outlining how the software will be designed. You may want to address the following questions:

- What initialization does your software need to perform?
- How are your various inputs sampled? How is the data communicated to the rest of the software?
- How are your various outputs computed?
- What does your software spend most of its time doing? Can it run forever?
- How do the various components of your software (input processing, output processing, computation) interact?

There are no format requirements: don't waste time drawing lots of UML diagrams unless you love them.

Example, Continued: To continue the above example regarding the three temperature sensors, the I/O requirements of each option are significantly different.

- A NTC thermistor is like a resistor in which the resistance is a sensitive function of temperature T, given by the equation:

  $$R_{ntc}(T) = R_0 \operatorname{Exp}\left(\frac{\beta}{T} - \frac{\beta}{T_0} \right),$$

  where R_0, T_0, and β are device-specific constants. A NTC thermistor can be combined with a known resistance R to make a voltage divider, the voltage of which can be measured with an analog-to-digital converter (ADC). From this the NTC resistance can be calculated in software, and the above equation can be inverted to obtain the temperature T. (Although — can a microcontroller calculate a natural logarithm without an extensive math library? Is a look-up table better?)
- An AS621 digital temperature sensor is a 6-pin chip that measures the temperature in $^\circ\mathrm{C}$ as a signed two's-complement 16-bit number. This data is transmitted to the microcontroller using the I^{2}C communications protocol (microcontroller inputs SCL and SDA). Most microcontrollers have the necessary I^{2}C communications hardware.
- A MAX6576 digital temperature sensor is also a 6-pin chip that measures temperature in K. This data is transmitted to the microcontroller as a square-wave pulse train with a period of:

  $$\tau = (10\,\mu\mathrm{s/K})\,T,$$

  where T is the temperature and τ is the period. This can use a GPIO port and a timer (or an input capture, if available!) to keep track of the period of the input pulse train.

Each of these three options can provide an output that is a reasonably accurate representation of temperature, however reading that data into the microcontroller and converting it (if necessary) into a temperature demands significantly approaches depending on the device used.

Deliverable: A software design document that addresses the questions above, and any other aspect you think pertinent.

Prototyping Plan

Embedded systems developers in the real world have a common problem: **The hardware is always late**. Too often, this means waiting until the hardware is finished before you begin any kind of software testing. That leads to angry customers and failed projects.

One possible solution is software prototyping. Basically, this means "faking out" the presence of the hardware, to see if the software behaves properly. You are not expected to develop hardware in this project, but you should attempt to implement some pieces of your design in software that you can test using the tools we have available in the lab. Although the DE10-Standard board (and/or the DE1-SoC in the simulator) probably don't have the exact peripherals needed in your design, they likely have something that can be used to approximate those peripherals. We have the ability to present both analog and digital inputs to our ARM platforms, and we have the ability to monitor digital and timed outputs. The simulator also supports some communication capabilities andsome more advanced concepts, including a VGA display and an audio input, should they be needed.

Example, Continued*:* To continue the above example regarding the three temperature sensors,

- Like most microcontrollers, the DE10-Standard has an ADC peripheral that will convert an analog voltage to a binary number. From a software prototyping perspective, it doesn't really matter whether in input analog signal comes from an NTC thermistor voltage divider or any other arbitrary source — you can write the software to control the ADC and convert the ADC result to a temperature. The simulator also implements the same ADC peripheral. Because it is a simulator, the ADC doesn't actually connect to a real analog input — but again, that doesn't really matter for software prototyping.
- Again, like most microcontrollers, the DE10-Standard has an I^{2}C communications controller. Again, it doesn't really matter what is on the other end of the I^{2}C communications channel — you can write the software to control the I^{2}C device and request 16-bit data from whatever hardware is available.
- The DE10-Standard does not have an input capture device that would streamline reading in a square wave input, but it does have several timers and available GPIO pins. You can write the software to implement an input capture device. You can even test this software by using *another* timer and some software to generate square waves at various frequencies and piping that output as the input for your input capture (first using software to pipe the data, maybe secondly by shorting an input and output GPIO pin).

Hopefully the above example demonstrates that even though temperature-sensing hardware isn't readily available in our lab, we do have plenty of hardware that provides output which can be processed as *if it were* from a temperature sensor. Consequently, we can write and test software with the appropriate functionality.

Note*:* A lot of hardware peripherals use the I^{2}C communications protocol to send data to the microcontroller. The DE10-Standard *does* have an I^{2}C controller, as mentioned above, but it isn't described in the "DE10-Standard Computer System with ARM Cortex-A9" document for some reason.

Deliverable*:* In your prototyping plan, pick the pieces of your software design that you would like to get working in your prototype. What tests would you like to run? How will you verify success? Define your plan of investigation to ensure your prototype is working correctly.

Select a Microcontroller

In our class, we have focused on the ARM. However, the ARM is not the only microcontroller in the world. In fact, as the ARM Cortex-A9 is a fairly power-hungry processor, it probably isn't the best for embedded systems (does an embedded wind-chill sensor actually need a two core 2 GHz CPU?), so take that into consideration.
For this section, please compare and contrast least two microcontrollers, one of which (or both) is *not* an ARM system. Outline how you might decide which one to use for your application.

- You can use the A9 as one comparison system, but if you want to look at an ARM microcontroller that is more suitable for embedded systems, maybe look at the M-series (such as the M3 or M4).

A non-exhaustive list of some good options for low-power and inexpensive embedded systems that do not require significant processing power is:

- The Texas Instruments MSP430 family of microcontrollers.

- The Analog Devices ADuC8xx family of microcontrollers (such as the ADuC845).
- The Microchip PIC24 family of microcontrollers.
- The STMicroelectronics ST10F family of microcontrollers.

As usual, Digikey has a huge list of microcontrollers that can easily be sorted by feature. Obviously if your embedded system needs a particular peripheral, it is helpful to find an microcontroller that already has that peripheral built in.

Hopefully by this point in the course you recognize that as long as you write software in a common language (i.e., C), and write reasonably portable, functional code, switching from one microcontroller to another is largely a matter of changing memory addresses and the structures of hardware peripherals. A timer on a ST10F276 will still have some form of status, control, and data registers, and will still provide the same basic functionality (i.e., timing user-defined intervals) as a timer on the DE10-Standard.

Example, Continued*:* To continue the above example regarding the three temperature sensors,

- The Analog Devices ADuC845 has an on-board temperature sensor attached to one of the channels on the ADC. Assuming this temperature sensor is accurate enough for my system's design requirements, using this microcontroller could simplify the hardware design by reducing the amount of peripherals that need to be externally connected to the microcontroller.

The other requirements of the system should also be compared against the performance and peripherals of the microcontrollers — the above does not represent a complete discussion of which microcontroller is best for this project.

Note that for implementing your software prototypes, you **do not** have to work on the DE10-Standard hardware or DE1-SoC simulator if you have a preferred platform, but you **do** have to work "on the bare metal". This means **no** Arduino, Raspberry PI, or any other system that provides a runtime environment.

The DE10-Standard is an expensive piece of equipment, but there are a few microcontroller development boards that are quite affordable (around $20 or so), and can be coded in C using an IDE that is often free (Keil writes programming tools for a lot of different microcontroller families, and usually has free evaluation or "lite" versions of their software). You certainly **do not have to buy anything** for this course, but if you are really interested in embedded system design and want your own hardware, you can use it for this project.

Deliverable*:* Provide a discussion outlining how you made your decision.

Revised Software Design

No plan survives contact with the enemy. Reflect on what might be changed from your original software design. Questions you should address include:

- After completing the above steps, can you revise your software design?
- What has changed as you learned and worked with the ARM (or whatever microcontroller you used for software prototyping)?
- As interrupt-driven programming is the last section discussed in this course, you probably did not consider it during your original design. However, by this point you should know what interrupts are, so can you seea role for interrupts in your revised design?

There are no format requirements: don't waste time drawing new UML diagrams unless you love them.

Deliverable*:* A software design document that addresses the questions above and any other aspect you think pertinent.

Results from Prototyping

Describe how the software prototyping went. In particular, you may want to address the following questions:

- Were you able to get any parts of the code working in the lab?
- Were there any unexpected challenges that significantly delayed your progress? (Design/coding-related challenges, not personal life-related challenges.)
- What is the next step towards completing your system?

The prototyping results should also be described in the context of how accurately you consider the software prototyping approach to mimic the behaviour in actual hardware.

Describe all the various software iterations of your code and the improvements made at each iteration. For example, at the first iteration your code may have worked for one sensor and in the next iteration your code worked for multiple sensors. Or at the first iteration your code may have had one feature and in the next iteration your code implemented multiple features. Also describe which group member wrote which part of the code.

Deliverable*:* Report on the lab experience.

Deliverable*:* Provide a debug/test plan for when the hardware is ready.

Source Code

Provide all your source code, hopefully with some comments. This can be included with your report as a separate file or inserted into the report as an appendix.

Deliverable*:* The code. Describe all the various software iterations of your code and the improvements made at each iteration. Describe which group member wrote which part of the code.

Conclusions

Comment on the feasibility of your system design. Based on your experience thus far, do you expect any major challenges blocking the way to producing a real product?

Please reflect on this design process, and on the course. In particular, you might want to address the following:

- Have you learned anything?
- Did this project help pull together how the various pieces of this course fit together?
- Do you see how you can integrate microcontroller technology into future projects?

As mentioned above, we would also like you to include a brief discussion of the environmental impact of your system. Are there any sustainability issues in the development of your device, once the device is commercially available, or after the device is disposed of?

Deliverable*:* Discuss the predicted feasibility of finalizing your design in hardware.

Deliverable*:* Discuss the sustainability of your system design and development.

Deliverable*:* Discuss any environmental issues from the use and disposal of your system once it is brought to market.

Deliverable*:* Describe what you learned from this project that was not taught during class. This can include the specs of peripheral electronic components used by your design, the algorithms or calculations made by your code, etc.