ECE4220 – Final Project

ECE4220 students are required to work on a class project. They will work in pairs for this assignment. If there is an odd number of students, one group of three can be formed.
The project addresses an important engineering concept: the implementation of a Supervisory Control And Data Acquisition System (SCADA).

Disclaimer

The following text on SCADA systems, with the exception of the non-italicized paragraphs, is NOT of our authorship, but it was taken from Wikipedia. Please, refer to this URL for further information on SCADA systems.

SCADA (supervisory control and data acquisition) *is a system that operates with coded signals over communication channels so as to provide control of remote equipment (using typically one communication channel per remote station). The control system may be combined with a data acquisition system by adding the use of coded signals over communication channels to acquire information about the status of the remote equipment for display or for recording functions.[1] It is a type of industrial control system (ICS). Industrial control systems are computer-based systems that monitor and control industrial processes that exist in the physical world. SCADA systems historically distinguish themselves from other ICS systems by being large-scale processes that can include multiple sites, and large distances.[2] These processes include industrial, infrastructure, and facility-based processes, as described below:*

- *Industrial processes include those of manufacturing, production, power generation, fabrication, and refining, and may run in continuous, batch, repetitive, or discrete modes.*
- *Infrastructure processes may be public or private, and include water treatment and distribution, wastewater collection and treatment, oil and gas pipelines, electrical power transmission and distribution, wind farms, civil defense siren systems, and large communication systems.*
- *Facility processes occur both in public facilities and private ones, including buildings, airports, ships, and space stations. They monitor and control heating, ventilation, and air conditioning systems (HVAC), access, and energy consumption.*

Common system components

A SCADA system usually consists of the following subsystems:

- *Remote terminal units (RTUs) connect to sensors in the process and convert sensor signals to digital data. They have telemetry hardware capable of sending digital data to the supervisory system, as well as receiving digital commands from the supervisory system. RTUs often have embedded control capabilities such as ladder logic in order to accomplish boolean logic operations.*
- *Programmable logic controller (PLCs) connect to sensors in the process and convert sensor signals to digital data. PLCs have more sophisticated embedded control capabilities (typically one or more IEC 61131-3 programming languages) than RTUs. PLCs do not have telemetry hardware, although this functionality is typically installed alongside them. PLCs are sometimes used in place of RTUs as field devices because they are more economical, versatile, flexible, and configurable.*
- *A telemetry system is typically used to connect PLCs and RTUs with control centers, data warehouses, and the enterprise. Examples of wired telemetry media used in SCADA systems include leased telephone lines and WAN circuits. Examples of wireless telemetry media used in SCADA systems include satellite (VSAT), licensed and unlicensed radio, cellular and microwave.*

- *A data acquisition server is a software service which uses industrial protocols to connect software services, via telemetry, with field devices such as RTUs and PLCs. It allows clients to access data from these field devices using standard protocols.*
- *A human–machine interface or HMI is the apparatus or device which presents processed data to a human operator, and through this, the human operator monitors and interacts with the process. The HMI is a client that requests data from a data acquisition server.*
- *A Historian is a software service which accumulates time-stamped data, boolean events, and boolean alarms in a database which can be queried or used to populate graphic trends in the HMI. The historian is a client that requests data from a data acquisition server.*
- *A supervisory (computer) system, gathering (acquiring) data on the process and sending commands (control) to the SCADA system.*
- *Communication infrastructure connecting the supervisory system to the remote terminal units.*
- *Various processes and analytical instrumentation.*

Systems concepts

The term SCADA (Supervisory Control and Data Acquisition) usually refers to centralized systems which monitor and control entire sites, or complexes of systems spread out over large areas (anything from an industrial plant to a nation). Most control actions are performed automatically by RTUs or by PLCs. Host control functions are usually restricted to basic overriding or supervisory level intervention. For example, a PLC may control the flow of cooling water through part of an industrial process, but the SCADA system may allow operators to change the set points for the flow, and enable alarm conditions, such as loss of flow and high temperature, to be displayed and recorded. The feedback control loop passes through the RTU or PLC, while the SCADA system monitors the overall performance of the loop.

Data acquisition begins at the RTU or PLC level and includes meter readings and equipment status reports that are communicated to SCADA as required. Data is then compiled and formatted in such a way that a control room operator using the HMI can make supervisory decisions to adjust or override normal RTU (PLC) controls. Data may also be fed to a Historian, often built on a commodity Database Management System, to allow trending and other analytical auditing.

SCADA systems typically implement a distributed database, commonly referred to as a tag database, which contains data elements called tags or points. A point represents a single input or output value monitored or controlled by the system. Points can be either "hard" or "soft". A hard point represents an actual input or output within the system, while a soft point results from logic and math operations applied to other points. (Most implementations conceptually remove the distinction by making every property a "soft" point expression, which may, in the simplest case, equal a single hard point.) Points are normally stored as value-timestamp pairs: a value, and the timestamp when it was recorded or calculated. A series of value-timestamp pairs gives the history of that point. It is also common to store additional metadata with tags, such as the path to a field device or PLC register, design time comments, and alarm information.

SCADA systems are significantly important systems used in national infrastructures such as electric grids, water supplies and pipelines. However, SCADA systems may have security vulnerabilities, so the systems should be evaluated to identify risks and solutions implemented to mitigate those risks.[3]

(… removed information on Human-machine interfaces...)

An important part of most SCADA implementations is alarm handling. The system monitors whether certain alarm conditions are satisfied, to determine when an alarm event has occurred. Once an alarm event has been detected, one or more actions are taken (such as the activation of one or more alarm indicators, and perhaps the generation of email or text messages so that management or remote SCADA operators are informed). In many cases, a SCADA operator may have to acknowledge the alarm event; this may deactivate some alarm indicators, whereas other indicators remain active until the alarm conditions are cleared. Alarm conditions can be explicit—for example, an alarm point is a digital status point that has either the value NORMAL or ALARM that is calculated by a formula based on the values in other analogue and digital points—or implicit: the SCADA system might automatically monitor whether the value in an analogue point lies outside high and low- limit values associated with that point. Examples of alarm indicators include a siren, a pop-up box on a screen, or a coloured or flashing area on a screen (that might act in a similar way to the "fuel tank empty" light in a car); in each case, the role of the alarm indicator is to draw the operator's attention to the part of the system 'in alarm' so that appropriate action can be taken. In designing SCADA systems, care must be taken when a cascade of alarm events occurs in a short time, otherwise the underlying cause (which might not be the earliest event detected) may get lost in the noise. Unfortunately, when used as a noun, the word 'alarm' is used rather loosely in the industry; thus, depending on context it might mean an alarm point, an alarm indicator, or an alarm event.

Hardware solutions

SCADA solutions often have Distributed Control System (DCS) components. Use of "smart" RTUs or PLCs, which are capable of autonomously executing simple logic processes without involving the master computer, is increasing. (...removed information on languages...) In many electrical substation SCADA applications, "distributed RTUs" use information processors or station computers to communicate with digital protective relays, PACs, and other devices for I/O, and communicate with the SCADA master in lieu of a traditional RTU.

(… removed information on PLCs and HMI...) *The Remote Terminal Unit (RTU) connects to physical equipment. Typically, an RTU converts the electrical signals from the equipment to digital values such as the open/closed status from a switch or a valve, or measurements such as pressure, flow, voltage or current. By converting and sending these electrical signals out to equipment the RTU can control equipment, such as opening or closing a switch or a valve, or setting the speed of a pump.*

Supervisory station

The term supervisory station refers to the servers and software responsible for communicating with the field equipment (RTUs, PLCs, SENSORS etc.), and then to the HMI software running on workstations in the control room, or elsewhere. In smaller SCADA systems, the master station may be composed of a single PC. In larger SCADA systems, the master station may include multiple servers, distributed software applications, and disaster recovery sites. To increase the integrity of the system the multiple servers will often be configured in a dual-redundant or hot-standby formation providing continuous control and monitoring in the event of a server malfunction or breakdown.

Communication infrastructure and methods

SCADA systems have traditionally used combinations of radio and direct wired connections, although SONET/SDH is also frequently used for large systems such as railways and power stations. The remote management or monitoring function of a SCADA system is often referred to as telemetry. Some users want SCADA data to travel over their pre-established corporate networks or to share the network with other applications. The legacy of the early low-bandwidth protocols remains, though.

SCADA protocols are designed to be very compact. Many are designed to send information only when the master station polls the RTU. Typical legacy SCADA protocols include Modbus RTU, RP-570, Profibus and Conitel. These communication protocols are all SCADA-vendor specific but are widely adopted and used. Standard protocols are IEC 60870-5-101 or 104, IEC 61850 and DNP3. These communication protocols are standardized and recognized by all major SCADA vendors. Many of these protocols now contain extensions to operate over TCP/IP. Although the use of conventional networking specifications, such as TCP/IP, blurs the line between traditional and industrial networking, they each fulfill fundamentally differing requirements.[4]

(… removed information on security and control...)

Alarm and Event Handling
One important component of SCADA systems that is not well described in the Wikipedia website is the Alarm and Event Processing System. In a power system, a power line may become overloaded, causing switches, relays and circuit breakers to start to automatically open, preventing damage to the power lines or even the power plants. The breaking of one power line may in turn cause the overload of adjacent lines, causing a cascade effect on their switches, relays and breakers. In order to restart the system or even to prevent similar problems from happening again, operators must analyze the sequence of events in the exact order that they occurred. This analysis is commonly referred to as post-dispatch.
For post-dispatch to be successful, the SCADA system must be able to gather, maintain and store every alarm and event detected, as well as the time in which they occurred. This is, of course, a very time-critical requirement and it can only be satisfied under the Real Time paradigm.

SCADA architectures
SCADA systems have evolved through four generations as follows:[6][7][8]

<u>First generation: "Monolithic"</u>
Early SCADA system computing was done by large minicomputers. Common network services did not exist at the time SCADA was developed. Thus SCADA systems were independent systems with no connectivity to other systems. The communication protocols used were strictly proprietary at that time. The first-generation SCADA system redundancy was achieved using a back-up mainframe system connected to all the Remote Terminal Unit sites and was used in the event of failure of the primary mainframe system. Some first generation SCADA systems were developed as "turn key" operations that ran on minicomputers such as the PDP-11 series made by the Digital Equipment Corporation.

<u>Second generation: "Distributed"</u>
SCADA information and command processing was distributed across multiple stations which were connected through a LAN. Information was shared in near real time. Each station was responsible for a particular task thus making the size and cost of each station less than the one used in First Generation. The network protocols used were still not standardized. Since the protocols were proprietary, very few people beyond the developers knew enough to determine how secure a SCADA installation was. Security of the SCADA installation was usually overlooked.

<u>Third generation: "Networked"</u>
Similar to a distributed architecture, any complex SCADA can be reduced to simplest components and connected through communication protocols. In the case of a networked design, the system may be spread across more than one LAN network called a process control

network (PCN) and separated geographically. Several distributed architecture SCADAs running in parallel, with a single supervisor and historian, could be considered a network architecture. This allows for a more cost effective solution in very large scale systems.

Fourth generation: "Internet of Things"
With the commercial availability of cloud computing, SCADA systems have increasingly adopted Internet of Things technology to significantly reduce infrastructure costs and increase ease of maintenance and integration. As a result, SCADA systems can now report state in near real-time and use the horizontal scale available in cloud environments to implement more complex control algorithms than are practically feasible to implement on traditional programmable logic controllers.[9] Further, the use of open network protocols such as TLS inherent in the Internet of Things technology, provides a more readily comprehensible and manageable security boundary than the heterogeneous mix of proprietary network protocols typical of many decentralized SCADA implementations. One such example of this technology is an innovative approach to rainwater harvesting through the implementation of real time controls (RTC).

Once again, more information on SCADA systems as well as the full original text, including its references, can be found in the Wikipedia document (https://en.wikipedia.org/wiki/SCADA).

Assignment

This project consists of implementing and testing a hardware and software architecture for a SCADA system. As described above, a SCADA system allows the monitoring and control of industrial processes, power generation plants, etc. Typically, these processes are large scale, consisting of many remote stations, also known as Remote Terminal Units (RTUs), which connect to sensors and can send data to and receive commands from supervisory systems.

In addition to RTUs and supervisory systems, general SCADA systems include programmable logic controllers (PLCs), telemetry systems, human-machine interfaces (HMI), communication infrastructure, an Historian and an Alarm and Event Processing system. The latter is a very important component, especially in power systems. Power lines may become overloaded, causing switches, relays and circuit breakers to start to automatically open, preventing damage to the power lines or even the power plants. The breaking of one power line may in turn cause the overload of adjacent lines, causing a cascade effect on their switches, relays and breakers. In order to restart the system or even to prevent similar problems from happening again, operators must analyze the sequence of events in the exact order that they occurred. This analysis is commonly referred to as post-dispatch. For post-dispatch to be successful, the SCADA system must be able to gather, maintain and store every alarm and event detected, as well as the time in which they occurred. This is, of course, a very time-critical requirement.

Next you are presented with a SCADA system architecture with multiple RTUs, power lines, and measurements (i.e. voltages, currents, state of switches, etc.). We will not concern ourselves with the HMI and the control elements such as PLCs, but only with the Alarm and Event Processing System, the RTUs, simple actuators, and anything required to 'connect' those sub-systems.

You are required to analyze the proposed system, implement and test it, and write a report following the guidelines found at the end of this document.

Proposed System

Figure 1 illustrates the general SCADA system proposed here. The process (substation or power plant) consists of *N* RTUs. Each one of the RTUs has M_{Di} digital inputs, M_{Do} digital outputs, and M_A analog inputs for monitoring switches, turning on/off components, and monitor power lines (voltages, currents), respectively. The RTUs are connected to the Console Operator / Historian through a Local Area Network (LAN).

The RTUs will provide periodic updates of their corresponding sub-system's status. Normal operation plus any event will be reported, with time-stamps. The RTUs can also take in commands coming from the Console Operator. The Historian is in charge of keeping the log of the entire system. All events should be logged in the correct sequence in which they occurred.

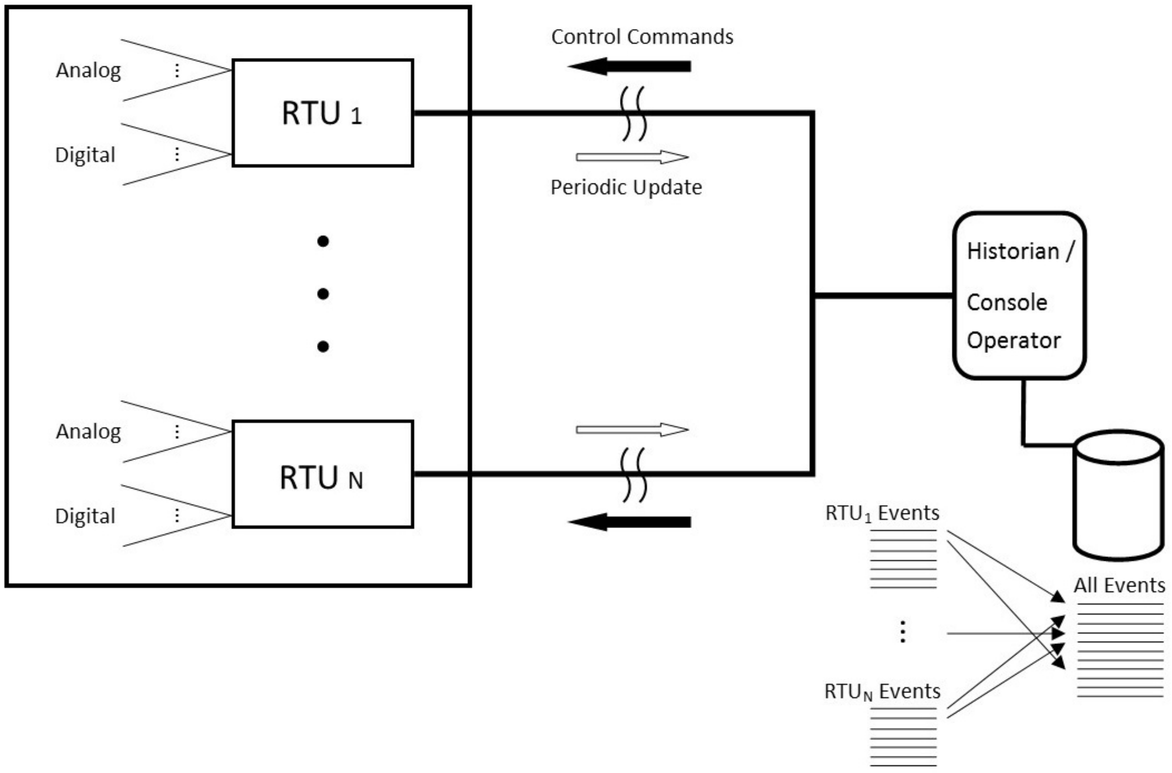

Figure 1. General system scheme.

Important aspects to be considered:

1. As mentioned before, the RTUs will monitor their corresponding subsystems. They need to sample the analog inputs and check the digital inputs in real time. Events that need to be detected and logged are the following: i) changes in switches; ii) power line overload (if power line goes beyond normal operation limits); iii) no power; iv) commands received. One important question is how close from each other can the events be detected. Each RTU will send periodic updates on the status of its subsystem through the network, including all events detected since the last update. Each piece of information should be time-stamped.
2. A critical aspect of the system is to synchronize the data log between the *N* RTUs. All those RTUs will be sending the updates to the Historian, which is in charge of keeping a log of all events. It is imperative to have them in the correct sequence. The solution to this synchronization issue would be trivial if all the system components had the same clock and latency to process the events, and if they all started at the same time. However, in most systems that is not the case. Without a common time reference, the time stamps from the different RTUs would be meaningless. One possible solution would be to use a Master/Slave

setup, where the Master would send signals to the Slaves indicating the current time of the entire system. With that common time reference, synchronization could be achieved, taking into account the individual clocks of the RTUs.
Another solution is to use a protocol such as Network Time Protocol (NTP), which is designed for clock synchronization between computers over networks. Through NTP, the time of the RTUs connected to the networks is the same (within the NTP specifications).

3. Another important aspect is how to handle concurrence on access of the event list. All RTUs will be sending information to the historian at any given time. Furthermore, an operator may request visualization of the current system log. Concurrence may be addressed with a database system. Such a system is designed for organizing data entered by multiple users/applications, and also for querying and displaying the data, all according to specific criteria (e.g. time of occurrence).
   Another approach (not relying on a database) would use a Client/Server paradigm. The different RTUs would establish Client/Server connections with the Console Operator. Concurrence control would need to be done “manually”, by means of semaphores, mutex operators, etc. The use of buffers (e.g. circular buffers) may also be required to ensure that data are not lost/overwritten. Moreover, the sorting of the data would also need to be done “manually”, either at storage or retrieval.

4. Finally, the system needs to guarantee the persistence of the events in case of power failure of an RTU or connection issues. An RTU may go off-line temporarily, either because of power loss or because of problems with the network (from broken network cables to complete network failure). Still, we want to eventually log events that may have happened between the last successful update and the time that the power went down, or during the network downtime.

Implementation

In order to test and validate the general system described above, we propose an implementation with the following minimum specifications:

A. $N = 2$ RTUs and a station for the Console Operator / Historian. The RTUs are to be implemented using Raspberry Pi 3 (RPi) boards. The Operator/Historian will run on a workstation, not on a RPi.

B. Each RTU will have $M_{Di} = 3$ Digital inputs (0 – 3.3V) for switches, $M_{Do} = 3$ Digital outputs (0 – 3.3V) for LEDs, and $M_A = 1$ Analog input. The analog input represents a power line (only voltage in this case). Normal operation is considered to be: AC signals between 1V and 2V, at 10 Hz [1].

C. As mentioned above, the following events need to be detected:
   i. Changes in switches. Any transition from logic 0 to 1 or vice-versa in any of the digital input ports represents a change, and should be logged.
   ii. Line overload. An event would be any time the power line goes beyond normal operation limits ($V < 1$ or $V > 2$).
   iii. “No power”. If the voltage remains constant for a few measurements, it will be assumed that the power line is down.
   iv. Command received: The RTUs may receive commands to turn On or Off any of the LEDs connected to the digital output ports.

[1] The voltage levels and frequency considered “Normal” may be redefined, depending on the source used. The actual values used should be clearly indicated in the report.

D. Information to be logged consists of the following:
   i. Time stamp
   ii. RTU identification number.
   iii. The status of all the switches (Open/Closed) and LEDs (On/Off).
   iv. Voltage at the line at the time of the event (in practice, the power would be recorded).
   v. Type of event / current status. If there was a change in a switch or LED, it should be specified which one.

E. The periodic updates will be done every <u>1 second</u>. These updates consist of all events that may have happened since the previous update, and the RTU status at the time of the update. Even if there was no event since the previous update, the current status should be reported and logged.

F. The 10 Hz voltage signal can be obtained with a function generator, using an Arduino or a similar μC, or by building a simple circuit (many such circuits can be found on the web).

G. Analog to Digital Conversion (ADC) should be done using the **MCP3008 chip**, available in the auxiliary boards. This chip allows up to eight, 10-bit conversion channels, with a maximum sampling rate of 200,000 samples per second.

H. Synchronization of the clocks: Network Time Protocol (NTP) is available in the RPis. However, internet access may be required. Tests should be carried out to determine if the time on the RPis is the same (within the NTP specifications). If not, a Master/Slave setup may be necessary.

I. Concurrence on access of the event list can be implemented through <u>either</u>
   i. Server/Client paradigm; <u>or</u>
   ii. A database system.

   A database system takes care of the concurrence control and sorting of the events. However, learning to use such a system takes some time. On the other hand, if the database system is not used, the sorting and concurrence control needs to be done "manually". Nevertheless, no time needs to be spent getting familiar with the database system and learning how to incorporate it into the project.

J. The problem of persistence will <u>not</u> be addressed in this implementation.

K. Many topics/aspects of the ECE4220/7220 class have been highlighted above. Those and some additional ones required for the proposed implementation are listed next: Multi-tasking/threading, Kernel Modules, Periodic and Real Time tasks, task communication, cooperation and synchronization, concurrence control, network communication, client/server model, interfacing to peripheral devices, interrupts, Analog to Digital conversion.

<u>Testing</u>

The following tests will be performed to demonstrate proper functionality of the system:

- Clock synchronization tests. Verify if multiple RPis have the same time. If not, a master slave setup will be implemented to synchronize the boards. This would be discussed in class.
- Proper ADC conversion test. Voltage signals with the normal operation specifications will be input to the analog channel. The conversion results should be plotted, to illustrate how accurately can the signals be recovered (plot a few seconds worth of data). A sampling rate of 1 kHz should be more than enough.

- “Line overload” events can be tested by increasing the amplitude of the input signals. **Care should be taken not to go beyond the ADC limits.**
- “No power” events can be tested by switching from a normal signal to ground. If a function generator is used, the signal can be temporarily set to a constant voltage value.
- Control commands test. The console operator should send commands to turn on and off the different LEDs connected to the RTUs. Different sequences should be specified and performed. The log file should reflect those sequences.
- Sporadic switch changes can be tested using the push buttons from the auxiliary boards used in the lab. The use of actual switches would be better. A simple circuit can be built for that purpose. That is not mandatory, though.
  Different sequences (pressing, releasing the buttons or moving the switches from one position to the other) using both RTUs should be specified and performed. The log file should reflect the specified sequences. All the digital inputs from both RTUs should be used.
- Manually pressing/releasing the push buttons (or moving the switches) will hardly allow to test how far apart the events can be. Instead, some of the unused GPIO ports of the RPis boards can be used as outputs, to be connected to other ports configured as inputs. Very precise and specific output sequences of logic 1s and 0s can be programmed for testing (a small shell script using the GPIO command line utility could be used).
  Consider the example shown in Figure 2. Two output channels (representing switches) are connected to two input channels. Let $\Delta t_{\min}$ be the minimum time difference between two events so that they can be properly detected and logged. Any two events happening within a time difference less than $\Delta t_{\min}$ may or may not be logged in the right sequence. For this example, let $\Delta t_{\min} = 5\ \mathrm{ms}$. At time $10\ \mathrm{ms}$, both channels change (approximately) at the same time. Two events should be logged, but either one could be logged first. It can even be the case that only one of the events will be detected, given that they are so close together.
  On the other hand, at time $30\ \mathrm{ms}$, only channel 1 changes, and at time $40\ \mathrm{ms}$, only channel 2 changes. Once again, both events should be logged, but this time, the log should reflect the correct sequence. The same should be true whenever the time difference between two events is greater than $\Delta t_{\min}$.

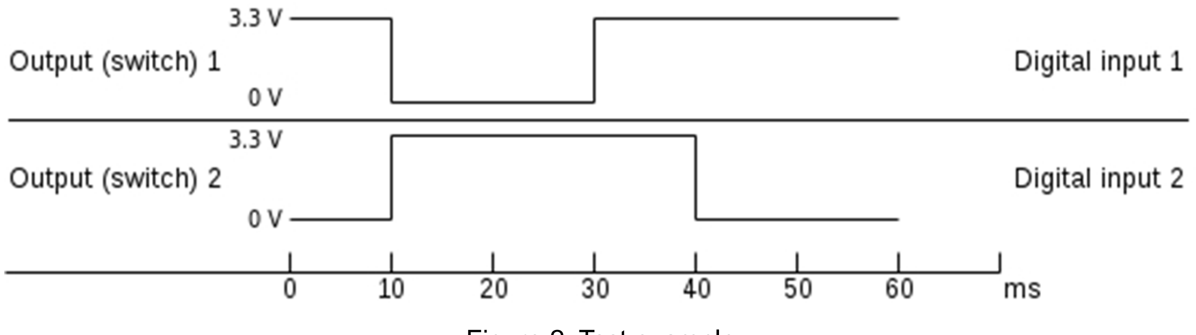

Figure 2. Test example

<u>Several tests should be run to determine what the $\Delta t_{\min}$ is for the system.</u>

- Visualization tests. When a user (operator) requests the current log file, it should be displayed in a clear, convenient way. All requested pieces of information should be there (implementation item D). Different scenarios should be presented: when no event has happened (only the periodic status reports should be in the log file); when a few events have happened sporadically; when many events have been happening (same or different types of events), etc. The tests described above would be successful if the log file reflects the events properly. That is, no event should be missing, and they should be properly sorted according to the time-stamps.

All tests will be performed multiple times. Statistics will be calculated when appropriate.

Additional (Optional) Experiments (Extra Credit – up to 20%)

In addition to the requirements and tests described above, students will have the opportunity to implement additional features for extra credit. The following are a few possibilities:

- Using Object Oriented programming in C++. You can define classes, use inheritance, exception handling, and other concepts from ECE3220 to implement your project.
- Using a Database to address the issue of concurrence, logging and sorting events.
- The RPi has SPI, UART and I^2C modules, which can be used to get data from sensors (e.g. temperature) or communicate with other devices. One or more of those modules could be explored and added to the system.
- The auxiliary board comes with a 7-segment display, which could be used to indicate the different events that may occur.

If you have other ideas, you may discuss them with the instructor.

Time-line and Milestones

Implementing, testing, debugging the system, as well as writing and submitting the final report and additional material should be completed by: **Friday, December 8th, 2017**. The following time-line is recommended:

- Weeks 1 – 3: Work on the software and hardware components, define the testing scenarios.
- Weeks 4 – 5: Finalize the individual components, interconnect and test the system, debug and fix any issues. Run the test scenarios to validate the system.
- Last week: collect final data, statistics; write and submit the final report and the additional required material (see guidelines in the next page).

A few more comments and recommendations:

- A good way to start is to work on the voltage signal generation and the ADC module. That is not part of any lab assignment.
- Most of the concepts from the labs will be useful for the project. Make sure you work hard on your lab assignments, and that your programs work properly. **There is a good chance that you will be able to re-use a lot of your lab code**.
- DO NOT leave the project for the last couple of weeks. There is a lot to be done. It is a challenging assignment, but very doable (and rewarding). Make sure to manage your time properly.
- We will discuss the project in class, and I will give more specific details and suggestions. You are welcome (and encouraged) to stop by my office to talk about the project.

- Pay close attention to the guidelines below. Run meaningful tests and report everything you do. Your report and video should highlight your experiments and results. Even if you are not able to finish everything, a good, well-written and detailed report can get you a good grade. On the other hand, even if you finish everything, if your report/submission is incomplete, not detailed, and is not well written, you will not get a good grade.

ECE4220 – Guidelines for Final Project Report and Submission

The final project reports should contain the main sections required for the lab reports, plus some additional requirements. The following organization is recommended:

<u>1) Abstract</u>
Brief overall summary of your project.

<u>2) Introduction</u>
Including the description of the project, the objectives/goals, the motivation behind the project, etc.

<u>3) Background</u>
Include any method that you are replicating (e.g. from a paper), common/relevant approaches used when addressing your particular problem or application, examples of where/why your application is used, etc.

<u>4) Proposed method / System description / Implementation</u>
Explain what you did and why you did it that way. This section should contain flowcharts/diagrams of your program(s) and of your hardware components. It is very important to explain how the multiple processes/threads interact, as well as how the devices/components interface.

<u>5) Experiments and Results</u>
Here you describe the tests that you did and present the results that you obtained. How did you make sure that your program was working properly? What cases did you try? How many times did you run your program(s)? How did you divide the work? In addition to describing your results, you should include tables, charts, snapshots, etc. Be sure to include statistics, when appropriate.

<u>6) Discussion and Conclusions</u>
Discussion/analysis of the results. Did your observations/results make sense? Were they what you were expecting? Why or why not? Discuss any problems that you may have encountered and how you fixed them. Talk about what you learned, interesting/unexpected things that you observed, problems/limitations that the current approach may have, alternative solutions/approaches, etc. Overall conclusions of the project.

<u>7) Appendices</u>
Include your well-commented code, and any other thing that you consider necessary.

In addition to the report itself -- described above -- you are also required to submit:

a) Your programs -- i.e. source code, Makefiles, and binary files -- everything needed to <u>create</u> your binary files.
b) A short ReadMe file explaining how to interconnect your system, and how to compile and run your program(s).
c) A video demonstration of your project. <u>Your video file should not be larger than 500 MB</u> (read note 1 below).
d) Your code should <u>also</u> be included in the appendix of the report, with comments, caption, and other details that will tell us what we are looking at.

Please, DO NOT submit "hundreds" of files!!! I will NOT accept such submissions. Everything must be in ONE SINGLE .zip/.tgz/.tar file!!

Name the file like this: ECE4220-LastName1_LastName2.zip (or .tgz or .tar). The last names should be in alphabetical order. You should upload the file to **<u>Canvas</u>**.

<u>Note 1</u>: If your video file is too large, Canvas may not accept your submission. If that is the case, create a .zip/.tgz/.tar file with everything except your video, and upload that to Canvas. Then, upload your video (named ECE4220-LastName1_LastName2-video) using the following link:

http://vigir.ee.missouri.edu/~gdesouza/UploadDir

<u>Note 2:</u> Presenting the code both in the Appendix and as an attachment (with Makefiles, README etc...) is threefold: 1) it allows us to judge the complexity of your project; 2) it allows us to measure your programming skills (the software designing skills are measured by the diagrams that were requested, and the explanations on how the multiple processes/threads interact); and 3) to make sure that your program compiles and runs (the latter, when possible). In the past, there have been cases where students submitted programs that wouldn't even compile due to syntax errors (C/C++ language). Once confronted, they replied that they didn't know that they were supposed to test their program/project. <u>Do not</u> be like that…