IE3048 Electrónica Digital 3 – Proyecto Final

Para el proyecto final de Electrónica Digital 3, los estudiantes trabajarán en parejas. Explorarán un concepto ingenieril muy importante: los sistemas de Supervisión, Control y Adquisición de Datos (SCADA). Además, incorporarán un elemento de Internet de las Cosas.

Aclaración:

El siguiente texto sobre sistemas SCADA, exceptuando los párrafos que no están en cursivo, no es de la autoría del instructor. Fue tomado de Wikipedia. Consulte https://en.wikipedia.org/wiki/SCADA u otras fuentes para más información sobre estos sistemas.

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]

First generation: "Monolithic"

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.

Second generation: "Distributed"

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.

Third generation: "Networked"

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).

Como se mencionó antes, más información sobre sistemas SCADA, así como el texto original completo (y sus referencias), se puede encontrar en el documento de Wikipedia. Pueden revisar la versión en español, pero no es muy completa.

Requisitos del Proyecto

Este proyecto consiste en la implementación y verificación de una arquitectura de *hardware* y *software* para un sistema SCADA simplificado. A eso se le agregará un elemento de Internet de las Cosas (*Internet of Things* – IoT), que servirá como una introducción al tema.

Como se describe arriba, los sistemas SCADA permiten monitorear y controlar procesos industriales, plantas de generación de energía, etc. Típicamente, esos procesos son a gran escala, y consisten en muchas estaciones remotas, conocidas como Unidades Terminal Remota (UTR), o *Remote Terminal Units* (RTUs). Las UTRs se conectan a sensores y pueden enviar datos y recibir comandos de un sistema de supervisión.

Además de las UTRs y los sistemas de supervisión, los sistemas SCADA incluyen controladores de lógica programable (PLCs), sistemas de telemetría, interfaces humano-máquina (HMI), infraestructura de comunicación, un “Historiador” y un sistema de alarma y procesamiento de eventos. Este último es un componente muy importante, especialmente en plantas de energía. Las líneas de transmisión pueden sobrecargarse, causando que interruptores, relés y cortacircuitos se abran automáticamente, para prevenir daños. La caída de una línea puede, a su vez, provocar que líneas adyacentes se sobrecarguen, causando un efecto en cascada de los interruptores, relés y cortacircuitos. Para poder reiniciar el sistema o prevenir problemas similares en el futuro, los operadores necesitan analizar la secuencia de eventos en el orden exacto en el que ocurrieron. Para que dicho análisis sea posible (y correcto), el sistema SCADA debe ser capaz de reunir, mantener y almacenar cada alarma y evento detectado, así como el instante en el que éstos ocurrieron.

A continuación, se presenta la arquitectura de un sistema SCADA con múltiples UTRs, líneas de transmisión y mediciones (voltajes, estados de interruptores, etc.). Para este proyecto NO nos preocuparemos de las HMI ni de los elementos de control como PLCs. El énfasis estará en el sistema de alarma y procesamiento de eventos, los UTRs, actuadores simples, y todo lo necesario para interconectar esos subsistemas en una red local. Además, en la incorporación del elemento IoT, cuya aplicación será libre.

Deberán analizar el sistema propuesto, e implementar y verificar algunas partes (no todas).

Sistema Propuesto

La Figura 1 ilustra el sistema SCADA general que se propone. El proceso (subestación o planta de energía) consiste en *N* UTRs. Cada UTR tiene M_{Di} entradas digitales, M_{Do} salidas digitales, y M_A entradas analógicas para monitorear interruptores o botones, encender/apagar componentes, monitorear líneas de transmisión (voltajes, corrientes), obtener información de sensores, etc. Los UTRs se conectan al Historiador / Operador (*Historian / Console Operator*) a través de una red de área local (LAN).

Los UTRs proporcionarán actualizaciones periódicas del estatus de sus subsistemas correspondientes. Cualquier evento debe reportarse, con sus marcas de tiempo (*time-stamps*). Si la operación es normal, también debe reportarse. Los UTRs también podrán recibir comandos del Operador. El Historiador estará a cargo de llevar un registro del sistema completo. Todos los eventos deben registrarse en la secuencia en que ocurrieron.

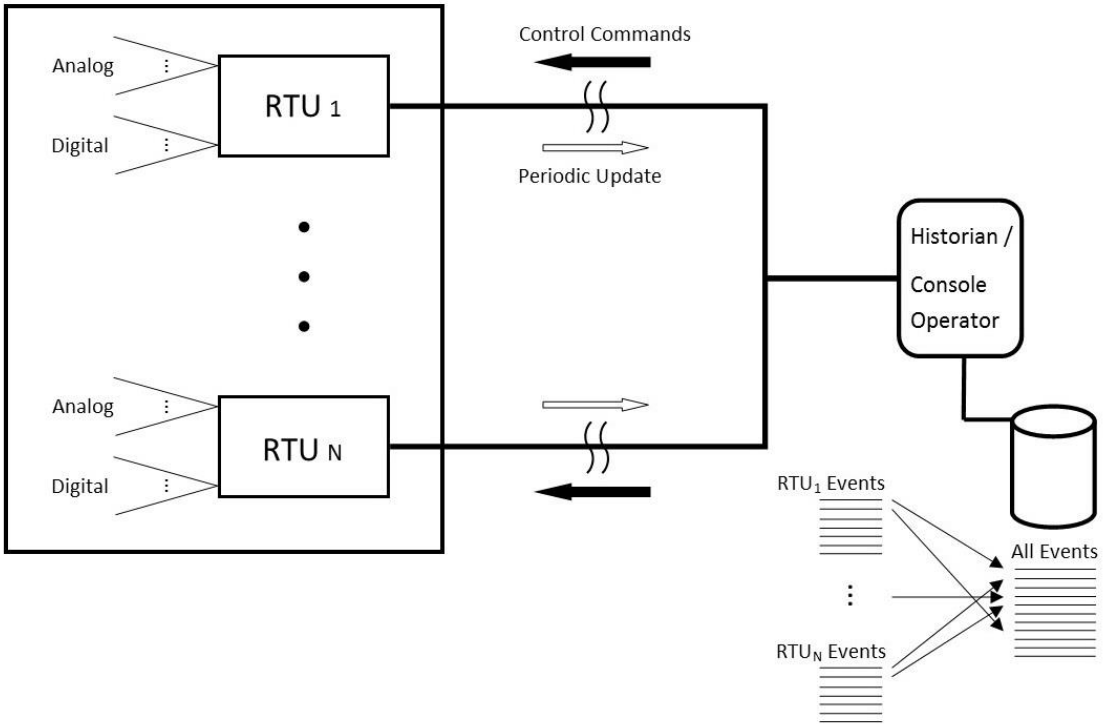

Figura 1. Esquema general del sistema.

Aspectos importantes para considerar:

1. Como se mencionó antes, las UTRs monitorearán sus correspondientes subsistemas. Deben leer las entradas analógicas y digitales en tiempo real. Los eventos que deben ser detectados y registrados son los siguientes: cambios en interruptores, botones presionados, “sobrecarga” (si la línea de transmisión sale de los límites de operación normales), comandos recibidos, actividad IoT registrada.
   Cada UTR enviará informes periódicos del estatus de su subsistema a través de la red local. Los informes incluirán todos los eventos detectados desde el informe anterior. Como se mencionó anteriormente, cada evento llevará su marca de tiempo.
   En el caso del evento “sobrecarga”, las UTR harán sonar una alarma.
2. Un aspecto crítico del sistema es la sincronización de los datos entre los distintos UTRs. Todos ellos estarán enviando informes al Historiador, que estará a cargo de mantener el registro de todos los eventos. Es importante que se puedan tener en la secuencia correcta.

La solución al problema de sincronización sería trivial si todos los componentes del sistema tuviesen el mismo reloj y latencia para procesar los eventos, y si todos los componentes arrancaran al mismo tiempo. Sin embargo, en la mayoría de los sistemas, ese no es el caso. Sin un tiempo de referencia común, las marcas de tiempo de los distintos UTRs serían insignificantes.
Una solución posible sería usar una configuración de Maestro/Esclavo, en la que el Maestro enviara señales a los Esclavos indicando el tiempo actual del sistema entero. Con esa referencia común, la sincronización se podría establecer, tomando en cuenta los relojes individuales de los UTRs.
Otra solución es usar un protocolo como el *Network Time Protocol* (NTP), que está diseñado para sincronización de relojes entre computadoras en una red. Mediante NTP, el tiempo de los UTRs conectados a la red es el mismo (dentro de las especificaciones del protocolo).

3. Otro aspecto importante es cómo manejar acceso concurrente al registro de eventos. Todos los UTRs estarán enviando información al Historiador en cualquier momento. Además, un operador podría solicitar visualización del registro actual del sistema. El problema puede abordarse por medio de una base de datos. Las bases de datos están diseñadas para organizar datos ingresados por múltiples usuarios/aplicaciones, para buscar y desplegar los datos, de acuerdo con criterios específicos (por ejemplo, marcas de tiempo).
   Otro enfoque (sin depender de una base de datos) es usar un paradigma de Cliente/Servidor. Los distintos UTRs establecerían conexiones Cliente/Servidor con el Operador. El control de acceso concurrente se haría "manualmente", por medio de semáforos, operadores *mutex*, etc. El uso de *buffers* (ej. *buffers* circulares) podría ser necesario para garantizar que los datos no se pierdan o sobrescriban. Además, la organización de los datos también se haría "a mano", al momento de guardar los datos o al recuperarlos.

4. Finalmente, el sistema necesita garantizar que los datos no se pierdan en caso de fallas de alimentación de energía o la desconexión de las UTRs. Una o más UTRs pueden estar fuera de línea temporalmente, por falta de energía, o por problemas de conexión a la red (desde desconexión/rotura de cables, hasta fallas completas de la red). Aún en esos casos, se desearía registrar los eventos que podrían ocurrir entre la última actualización exitosa y el momento en que se cayó la energía, o durante el tiempo de baja de la red. Este es un problema de <u>persistencia</u> de los datos.

<u>Implementación</u>

Deberán probar y validar <u>algunas partes</u> del sistema general descrito anteriormente. Para ello, se propone una implementación con las especificaciones <u>mínimas</u> siguientes:

A. N = 2 UTRs y una estación adicional para el Historiador/Operador. Las UTRs se implementarán con las Raspberry Pi 3B+ (RPi), integrados para conversiones analógico-digital y/o microcontroladores (μC). El Historiador/Operador correrá en una computadora personal, o en otra RPi, si cuentan con una propia.

B. Cada UTR tendrá por lo menos M_{Di} = 4 entradas digitales (0 – 3.3V): 2 para interruptores y 2 para botones (push buttons); M_{Do} = 3 salidas digitales (0 – 3.3V): 2 para LEDs y 1 para una bocina; y M_A = 1 entrada analógica. La entrada analógica podría representar una línea de transmisión (sólo voltaje en este caso). **La condición normal de operación es: señal entre 0.5 y 2.5 V.**
   Se podrán habilitar más entradas y/o salidas para conexiones con el Arduino IoT.

C. Como se mencionó antes, los siguientes eventos deben ser detectados:
   i. **Cambios en los interruptores:** Cualquier transición de 0 a 1 (lógico) o vice-versa en cualquiera de los puertos digitales conectados a los interruptores representa un cambio, y debe ser registrado.
   ii. **Eventos de botones:** Dos puertos digitales estarán conectados a *push buttons*. Cuando se presione alguno de ellos, ese es un evento que debe registrarse.
   iii. **Sobrecarga de la línea:** Se deben detectar y registrar los instantes en que la línea salga de los límites normales de operación ($V < 0.5$ ó $V > 2.5$). Cuando esto ocurra, el UTR deberá hacer sonar una alarma, usando una bocina.
   iv. **Comando recibido:** Las UTR pueden recibir comandos para encender o apagar cualquiera de los LEDs conectados a los puertos digitales de salida. Cada comando recibido es un evento.
   v. **Evento IoT:** Cada grupo deberá implementar algún ejemplo sencillo de aplicación IoT con los Arduinos que se les prestaron (ej. encender un LED desde un celular). Cuando se dé alguno de estos eventos, el Arduino podrá enviar alguna señal a la RPi, para que esta pueda hacer la notificación correspondiente.
   vi. **Reporte periódico:** aunque ninguno de los eventos anteriores haya ocurrido, el estatus actual de la UTR debe ser reportado.

D. La información que debe ser registrada consiste en lo siguiente:
   i. Marca de tiempo (*Time-stamp*)[1].
   ii. Número de identificación de la UTR.
   iii. El estatus de todos los interruptores (Abierto/Cerrado), botones (Encendido/Apagado), LEDs (Encendido/Apagado), y, según sea la aplicación IoT, el estatus del elemento.
   iv. Voltaje en la línea al momento del evento (en la práctica, se registraría la potencia).
   v. Tipo de evento. En el caso de un cambio en un interruptor, botón o LED, se debe especificar cuál fue.

E. Las actualizaciones periódicas se enviarán al Historiador/Operador cada ~2 segundos. Estas actualizaciones consistirán en todos los eventos qua hayan ocurrido desde la actualización anterior. Se deberá enviar el estatus de la UTR al momento de enviar la actualización, sin importar si no "pasó nada" en la última ventana de dos segundos.

F. La señal de voltaje puede obtenerse con un potenciómetro como divisor de voltaje, o con un generador de funciones (teniendo cuidado de que la señal esté entre 0 y 3.3 V).

G. Las RPis no cuentan con módulo de conversión Analógico – Digital (AD). Deberán usar integrados o μC externos para hacer la conversión, y deberán interconectarlos con las RPis. El puerto del integrado/μC será la entrada analógica de la UTR.
   Nota: En el kit que se les dio se incluyeron integrados MCP3002, los cuales cuentan con dos canales, y hacen conversiones de 10 bits. Se comunican vía protocolo SPI. También pueden usar PICs, Arduinos u otros μC que hayan usado en otros cursos. Pueden usar código desarrollado en dichos cursos. Las RPis cuentan con módulos SPI e I^2C, que pueden usarse para la comunicación con los integrados/μC. La librería *wiringPi* cuenta con funciones para configurar y usar dichos módulos de comunicación.

H. La detección de eventos de los interruptores y botones se hará por medio de interrupciones asociadas a los puertos GPIO de las RPis. La librería *wiringPi* también cuenta con funciones para configuración y manejo de estas interrupciones.

I. Sincronización de los relojes: el protocolo NTP está disponible en las RPis. Éste funciona siempre y cuando las RPis tengan acceso a Internet. Verifiquen que las RPis que usen para

[1] La función `gettimeofday()` puede usarse para obtener marcas de tiempo. Investiguen cómo usar la función.

las UTRs tengan la misma hora cuando trabajen en el proyecto. No es necesario que la computadora donde se corra el Historiador/Operador tenga la misma hora, aunque tampoco estaría mal.

J. La comunicación entre UTRs y el Historiador/Operador se hará por medio del modelo Servidor/Cliente. Se deberá tener cuidado con posibles accesos concurrentes al registro de eventos en el Historiador/Operador. No se requiere implementar algoritmos para ordenar los datos según las marcas de tiempo.

K. El problema de persistencia no se abordará en esta implementación.

Varios temas del curso IE3048 y cursos previos se han mencionado arriba. Esos y otros temas adicionales que pueden ser requeridos para la implementación propuesta son los siguientes: multi-procesos/hilos, eventos periódicos, comunicación entre tareas, cooperación y sincronización, control de concurrencia y región crítica, comunicación de red, modelo cliente/servidor, interfaz con dispositivos periféricos, interrupciones, conversión analógico-digital, Internet de las Cosas.

Experimentos

Se deberán realizar los siguientes experimentos para demostrar la funcionalidad adecuada del sistema:

- Pruebas de conversión AD. Señales de voltaje en condiciones normales de operación deberán ser enviadas a la entrada analógica del UTR. Los resultados de la conversión deberán ser graficados, para ilustrar qué tan fielmente pueden ser recuperadas las señales. Grafiquen uno o dos segundos de la señal. Una frecuencia de muestreo de 1 kHz es más que suficiente.
  Nota: estas pruebas son para verificar que la conversión AD funcione adecuadamente. Pueden crear un programa independiente que haga las conversiones y guarde los datos en un archivo de texto. Luego, pueden importar los datos del archivo a Excel o Matlab para generar las gráficas. **En el sistema final, no se necesita guardar ni graficar las señales muestreadas**.
- Eventos “sobrecarga” pueden ser probados incrementando o disminuyendo el valor del voltaje de entrada con el potenciómetro o la amplitud de la señal del generador de funciones. **Deben tener mucho cuidado de no sobrepasar los niveles máximos permitidos por los controladores**. Recuerden generar la alarma usando la bocina.
- Comandos de control. El Operador deberá enviar comandos para encender y apagar los distintos LEDs conectados a las UTRs. Especifiquen y envíen distintas secuencias de comandos. El registro de eventos deberá reflejar esas secuencias.
- Cambios en las entradas digitales pueden verificarse usando los *dip switches* y *push buttons*. Especifiquen y ejecuten distintas secuencias (presionando y soltando los botones y moviendo los *switches* de una posición a otra), usando ambas UTRs. El registro de eventos deberá reflejar esas secuencias. Todas las entradas digitales, de ambas UTRs, deberán ser usadas.
- Eventos IoT. Antes de conectar el Arduino IoT a la RPi, deberán investigar cómo se configura y programa. Hay varios ejemplos que pueden probar y usar para el proyecto (https://www.arduino.cc/en/Guide/NANO33IoT). Luego de probar la aplicación IoT por separado, conecten el Arduino a la RPi. Pueden usar un puerto GPIO e interrupciones, por ejemplo.

- Pruebas de visualización del registro de eventos: Los eventos registrados deberán irse guardando en un arreglo de cadenas suficientemente grande. Cuando el usuario (Operador) lo solicite (vía la consola), la información completa deberá desplegarse en la terminal de forma clara y ordenada (inciso D de la sección de implementación). Deberán probar distintos casos: cuando no ocurran eventos (sólo los reportes de estatus periódicos deberán aparecer); cuando sólo algunos pocos eventos hayan ocurrido; cuando muchos eventos hayan ocurrido (mismos o distintos tipos de eventos), etc.
  Los experimentos serán exitosos si lo desplegado refleja adecuadamente todos los eventos. Es decir, ningún evento deberá hacer falta.
  Nota: En la práctica, los eventos se guardarían en un archivo ("log") o en una base de datos. Además, los eventos se deberían poder desplegar en orden cronológico, incluyendo los eventos de los distintos UTRs. Omitiremos el almacenamiento y el ordenamiento en este proyecto.

Todos los experimentos deberán realizarse varias veces, para asegurarse que el funcionamiento es el adecuado. Noten que muchos experimentos se pueden realizar con un solo UTR (conversiones, interrupciones, etc.). Luego de hacer las pruebas individuales, deberán hacer pruebas con todos los dispositivos conectados.

Experimentos Adicionales (Opcional, para Crédito Extra – máximo 10%)

Adicional a los requisitos y experimentos descritos anteriormente, tendrán la oportunidad de implementar características adicionales para puntos extra. Las siguientes son algunas posibilidades:

- Archivo "log". Agregar una opción para el usuario (Operador) para guardar en un archivo de texto los eventos registrados.
- Implementación de un algoritmo para ordenar los eventos por marca de tiempo. Es posible que las actualizaciones de las distintas UTRs lleguen al Historiador en un orden distinto al que ocurrieron los eventos. En ese caso, los eventos se guardarían desordenados. Para que los eventos se muestren en el orden correcto, se puede usar uno de varios algoritmos para ordenar datos.
- Inclusión de sensores (ej. temperatura). Muchos sensores usan SPI, UART o I^2C, los cuales están disponibles en las RPis. Pueden usar algún sensor que hayan usado antes, y definir y registrar eventos asociados.
- Uso de *displays* de 7 segmentos u otro tipo de *displays* en las UTRs, para indicar los eventos que vayan ocurriendo.

Si tienen otras ideas, las pueden discutir con el profesor.

Nota: Antes de implementar las características adicionales, deben asegurarse de completar los requisitos y realizar los experimentos principales.

Cronograma, Fechas de Entrega y Recomendaciones

La demostración del proyecto se hará durante la semana de exámenes finales. La fecha exacta se determinará más adelante. Escribirán un reporte final, el cual deberá ser entregado a más tardar el **domingo 28 de noviembre**. Más adelante se publicará una guía para la elaboración del reporte y la forma de entrega.

Se recomienda seguir el siguiente cronograma:

- Semana 1 (3 – 7 de noviembre): Definir los componentes de hardware y software a utilizar, así como los escenarios de prueba. Investigar e implementar las interrupciones del GPIO y los módulos de comunicación entre las RPis y los integrados/microcontroladores. Reciclar código utilizado en prácticas y cursos anteriores. Investigar y definir aplicación IoT a realizar.
- Semana 2 (8 – 14 de noviembre): Hacer pruebas de conversión AD. Finalizar módulos de las UTRs. Trabajar en el Historiador. Implementar la comunicación Cliente/Servidor. Empezar a correr los experimentos. Detectar y corregir errores que se puedan presentar.
- Semana 3 (15 – 21 de noviembre): Terminar la implementación del proyecto y la realización de los experimentos. Recolectar datos, generar imágenes, etc. Si se completaron los requisitos principales, decidir si se harán experimentos adicionales. Empezar a preparar el reporte y la demostración.
- Semana 4 (22 – 28 de noviembre): Hacer la demostración. Terminar de preparar el reporte. Subir el reporte y cualquier material adicional requerido, a Canvas.

Deberán trabajar fuera de las sesiones de laboratorio. Hay algunas cosas nuevas, pero hay mucho que pueden reciclar de los laboratorios y los ejemplos de clase. Si se organizan, se dividen equitativamente el trabajo, y aprovechen bien el tiempo, no deberían necesitar demasiado tiempo fuera de los períodos de clase. El proyecto es demandante, pero no es imposible. Al final, debería ser una gran experiencia de aprendizaje.

En las sesiones de clase discutiremos aspectos del proyecto, así que pueden traer preguntas. También se recomienda que se conecten a las sesiones de atención para aclarar dudas que vayan surgiendo.