REPÚBLICA BOLIVARIANA DE VENEZUELA
INSTITUTO UNIVERSITARIO POLITÉCNICO
"SANTIAGO MARIÑO"
EXTENSIÓN MATURÍN

DISPOSITIVO PORTÁTIL DE ADQUISICIÓN DE DATOS CON INTERFAZ VIRTUAL BASADO EN TECNOLOGÍAS LIBRES PARA LOS ESTUDIANTES DE ELECTRÓNICA DEL INSTITUTO UNIVERSITARIO POLITÉCNICO SANTIAGO MARIÑO EXTENSIÓN MATURÍN

Trabajo Especial de Grado para Optar al Título de Ingeniero Electrónico

Autor: TSU. Leopoldo Marcano
Tutor: Ing**.** Daniel Porras

Maturín, Enero 2016

APROBACIÓN DEL TUTOR

En mi carácter de Tutor del Trabajo Especial de Grado titulado: DISPOSITIVO PORTÁTIL DE ADQUISICIÓN DE DATOS CON INTERFAZ VIRTUAL BASADO EN TECNOLOGÍAS LIBRES PARA LOS ESTUDIANTES DE ELECTRÓNICA DEL INSTITUTO UNIVERSITARIO POLITÉCNICO SANTIAGO MARIÑO EXTENSIÓN MATURÍN, presentado por el(la) ciudadano(a) Leopoldo Alexander Marcano Matute , Cédula de Identidad N° 12.223.691, para optar al Título de Ingeniero en Electrónica, considero que éste reúne los requisitos y méritos suficientes para ser sometido a presentación pública y evaluación por parte del Jurado Examinador que se designe.

En la ciudad de Maturín, a los 11 días del mes de Enero del 2016.

_____

Ing. Daniel Porras

C.I .12.357.176

ÍNDICE GENERAL

LISTA DE FIGURAS vi
RESUMEN viii
INTRODUCCIÓN 9
CAPÍTULO I 11
EL PROBLEMA 11
Contextualización del Problema 11
Objetivos de la Investigación 12
Objetivo General 12
Objetivos Específicos 13
Justificación de la Investigación 13
CAPÍTULO II 16
MARCO REFERENCIAL 16
Antecedentes de la Investigación 16
Bases Teóricas 18
Sistemas de Adquisición de datos 18
Proceso de adquisición de datos 19
Tiempo de Conversión de una Señal Analógica 22
Teorema de muestreo de Nyquist-Shannon 22
Interfaz Virtual por Computador 25
Tecnologías Libres 26
Software Libre 27
KiCAD: 28
MyOpenLab: 29
Hardware Libre: 31
Arduino 31

Pingüino..... 32
CAPÍTULO III..... 35
MARCO METODOLÓGICO..... 35
Modalidad de la Investigación..... 35
Tipo de Investigación de acuerdo al Nivel..... 36
Técnicas de Procesamiento y Análisis de Datos..... 39
CAPÍTULO IV..... 40
RESULTADOS..... 40
Estudio las de tarjetas de adquisición de datos basadas en tecnologías libres..... 40
Características de los modelos más comunes de Arduino..... 42
Funcionamiento de Arduino UNO..... 43
Especificaciones técnicas..... 45
Funcionamiento de Arduino Leonardo..... 48
Funcionamiento de Arduino Mega..... 52
Funcionamiento de Arduino DUE..... 54
Análisis de las de tarjetas de adquisición de datos basadas en tecnologías libres..... 57
Definición del protocolo de comunicación del dispositivo de adquisición de datos..... 59
Diseño de una interfaz virtual amigable, para manipular la tarjeta de adquisición de datos a implementar..... 62
Desarrollo del interfaz virtual para de adquisición de datos:..... 66
Construccion del sistema integrado 71
Conexión de entradas analógicas en Arduino..... 74
Calculo del Circuito Divisor Tensión..... 76
Demostración con el dispositivo en el área de Sistemas Digitales:..... 82
Demostración con el dispositivo en el área de Instrumentación:..... 84
Conclusiones..... 87
Recomendaciones..... 89

REFERENCIAS BIBLOGRAFICAS..... 90
ANEXOS..... 92

LISTA DE FIGURAS

Figuras Pag

Figura 1 Sistema de adquisición de datos 19
Figura 2 Esquema del proceso de adquisición de datos 21
Figura 3 Función de interpolación g(t) 24
Figura 4 Vista de una interfaz virtual realizada con labview 26
Figura 5 Diseños 3D y pcb con kicad 28
Figura 6. Interfaz virtual con MyOpenLab 29
Figura 7 Diagrama de bloque lógico con MyOpenLab 30
Figura 8. Ejemplo de Arduino como Sensor de Temperatura 32
Figura 9. Ejemplo de Pingüino como Volt- Amperímetro 32
Figura 10 Tipos de placas Arduino comunes 42
Figura 11 Identificación de Arduino UNO 44
Figura 12 Bloque de Alimentación de Arduino 45
Figura 13 Bloque de Salidas Digitales y PWM 46
Figura 14 Bloque de Voltaje Referencial y Entradas Analógicas 47
Figura 15 Identificación de Arduino Leonardo 49
Figura 16 Identificación Arduino Mega 53
Figura 17 Identificación de Pines Arduino DUE 55
Figura 18 Esquema de comunicación del dispositivo de adquisición de datos 59
Figura 19 Panel Frontal de MyOpenLAb 64
Figura 20 Paneles Generales de MyOpenLab 64
Figura 21 Area de componentes y Proyectos **¡Error! Marcador no definido.**
Figura 22 Panel de Circuito MyOpeLab 65

Figura 23 Panel Frontal MyOpenLab 66
Figura 24 Interfaz para selección del Puerto de comunicación 67
Figura 25 Interfaz Virtual para Señales Digitales 68
Figura 26 Interfaz Virtual para Señales Analógicas 68
Figura 27 Bloque Lógico de la interfaz Virtual Diseñada 69
Figura 28 Interfaz Virtual del Dispositivo de Adquisición de datos 71
Figura 29 Bloque de ADC en Arduino 72
Figura 30 Esquema de pin Analogico 74
Figura 31 Esquema de Adaptación de Entrada a pin analógica 75
Figura 32 Diagrama de circuito adaptador a entrada de arduino 77
Figura 33 Esquema del integrado 74ls08con compuertas lógicas 82
Figura 34 Demostración en Sistemas Digitales 83
Figura 35 Esquema del circuito con Sensor de luz 84
Figura 36 Conexión para demostración con sensor de luz con el dispositivo de adquisición de datos 85

REPÚBLICA BOLIVARIANA DE VENEZUELA
INSTITUTO UNIVERSITARIO POLITÉCNICO
"SANTIAGO MARIÑO"
EXTENSIÓN MATURÍN- MONAGAS
INGENIERÍA EN ELECTRÓNICA

DISPOSITIVO PORTÁTIL DE ADQUISICIÓN DE DATOS CON INTERFAZ VIRTUAL BASADO EN TECNOLOGÍAS LIBRES PARA LOS ESTUDIANTES DE ELECTRÓNICA DEL INSTITUTO UNIVERSITARIO POLITÉCNICO SANTIAGO MARIÑO EXTENSIÓN MATURÍN

Línea De Investigación: Diseño Electrónico

Autor:TSU. Leopoldo Marcano
Tutor: Ing. Daniel Porras
Enero, 2016

RESUMEN

El estudiante de electrónica del Instituto Universitario Politécnico Santiago Mariño durante su formación académica presenta un número considerable de asignaturas que hacen uso de la experimentación en laboratorios acondicionados con instrumentos precisos, con frecuencia esta experimentación se hace insuficiente para el estudiante, tomando en cuenta que estos espacios son un factor primordial para que el estudiante logre afianzar los conocimientos teóricos, se propone el desarrollo de un dispositivo portátil de adquisición de datos con interfaz virtual basado en tecnologías libres como una herramienta alternativa y complementaria. El desarrollo estuvo enmarcado en la modalidad de proyecto factible, de tipo descriptiva con una revisión documental, observación directa y análisis de datos. Como resultado de disponer de ésta herramienta alternativa se pretende mejorar y potenciar los conocimientos teóricos de los estudiantes de electrónica del Instituto Universitario Politécnico Santiago Mariño logrando así la excelencia académica.

Descriptores: Adquisición de Datos, Tecnologías Libres, Dispositivo Portátil.

INTRODUCCIÓN

La adquisición de datos, proceso que involucra la recopilación de información de forma automatizada a partir de fuentes de medición análogas y digitales, como sensores y dispositivos bajo prueba, consiste básicamente en tomar un conjunto de variables mensurables en forma física y mediante una etapa de acondicionamiento de señal, adecuarlas a formato digital para procesarlas, almacenarlas y representarlas de forma gráfica para su análisis posterior, el dispositivo de adquisición de datos desarrollado puede ser empleado como una herramienta que proporcione al estudiante de electrónica valores, mediciones o curvas características de componentes electrónicos, brindando un mejor entendimiento del mismo; cada vez son más comunes las tareas que requieren adquisición y procesamiento de información, donde las tarjetas e interfaces comerciales no están al alcance del estudiante , se presenta un dispositivo de adquisición de datos que aprovecha las funcionalidades de las tecnologías libres y de código abierto: “Arduino” y “MyOpenLab”.

Se procederá al estudio y conocimiento de la adquisición de datos como instrumento de medida, sus características básicas, funcionamiento y sus propiedades. También es necesario entrar en la compresión y cada vez más creciente uso de la instrumentación virtual.

Esto es una gran ventaja en un espacio como es el de la enseñanza ya que un equipo convencional de medida requiere mayor presupuesto y un mantenimiento que con la instrumentación virtual no son necesarios ya que se cuenta con el procesador y monitor de un ordenador común (PC), además de la tarjeta de adquisición de datos necesaria.

Para conseguir desarrollar el diseño se cuentan con varias herramientas como son el software “MyOpenLab” y el hardware “Arduino”. Dichas herramientas son adquiridas fácilmente ya sea mediante descarga en la propia página web del producto o en el caso de la tarjeta por un precio reducido en cualquier tienda especializada. Se ha tomado la decisión del uso de estas herramientas porque se tratan de dispositivos libres en “código abierto”, otra ventaja a la hora de no tener que pagar licencias con su correspondiente coste y la posible mejora de los dispositivos por parte de especialistas. Todo ello permitirá el desarrollo virtual necesario para conseguir un instrumento esencial y útil dentro de las prácticas de laboratorio de asignaturas como Instrumentación y Sistemas Digitales, que nos permitirá capturar, analizar, procesar y visualizar diferentes tipos de sistemas digitales o instrumentos de medida con el propósito de mejorar el entendimiento y afianzamiento de los conocimientos teóricos. La experimentación práctica con diferentes demostraciones tanto en la asignatura de Sistemas Digitales (manipular compuertas lógicas) como en asignaturas de Instrumentación (Lectura de Luz y temperatura) generará unos resultados determinados que serán expuestos, previos a la conclusión final que determine este proyecto.

El diseño del prototipo de dispositivo portátil de adquisición de datos con interfaz virtual se basa en los fundamentos de las Tecnología libres las cuales hacen referencia al conocimiento científico, técnico, literario que está a disposición del usuario, o de quien lo necesite y éste pueda ser tomado y usado sin ninguna restricción, permitiendo así la apropiación y asimilación de ese conocimiento; es decir hacer propio algo que se adquiere del medio externo. Lo similar es ya lo igual. Esto significa que cuando la tecnología externa se involucra como propia dentro de un determinado sistema, se está frente a lo que llamamos un proceso de asimilación.

CAPÍTULO I

EL PROBLEMA

Contextualización del Problema

El Instituto Universitario Politécnico Santiago Mariño es una institución de enseñanza con la misión de formar profesionales de elevada calidad que respondan a las necesidades del país y a los cambios que éste demande, fomentando así la investigación como vía para generar, aplicar y difundir nuevos conocimientos que favorezcan el avance científico, tecnológico, humanístico y social; llevando así a proyectarse en una institución de Educación Superior signada por la excelencia, que contribuya al desarrollo cultural, científico, humanístico y tecnológico del país y a la consolidación de los valores fundamentales de la sociedad.

Enmarcados en el contexto nacional, latinoamericano y mundial, no escapando de esta realidad la extensión de esta prestigiosa casa de estudio ubicada en Maturín Sede principal Anexo A3, del Estado Monagas en su carrera de pregrado en Ingeniería Electrónica; el estudiante durante su formación académica presenta prácticas de laboratorios que logran afianzar la teoría, con la realización de medición y observación de los valores propuestos por el docente mediante instrumentos de medida como osciloscopios, multímetros, vatímetro, entre otros.

Sin embargo estos laboratorios de electrónica del anexo 3 en la actualidad presentan un déficit de equipos e instrumentos y deterioro de los existentes debido al uso intensivo por parte del alumnado, esta situación en algún caso dificulta la realización de las practicas contempladas en la formación del estudiante, además que limita a los docentes la capacidad de desarrollar estrategias de enseñanza que profundicen el conocimiento mediante la experiencia al realizar las practicas.

Debido a esto se propone un dispositivo de adquisición de datos de bajo costo que aproveche las bondades del computador personal, cada vez mas presente en nuestros hogares, trabajo y universidad para que integrados mediante una interfaz virtual sea un dispositivo de medida e instrumentación que permita al estudiante medir, generar y analizar señales en las prácticas de laboratorio, ofreciendo una ventaja académica en la formación integral del estudiante del Instituto Universitario Politécnico Santiago Mariño Sede A3 Ext Maturín.

Objetivos de la Investigación

Objetivo General

Desarrollar un Dispositivo Portátil de adquisición de datos con interfaz virtual a través del puerto serial USB, basado en tecnologías libres para los estudiantes de electrónica del instituto Universitario Politécnico Santiago Mariño extensión Maturín.

Objetivos Específicos

1. Estudiar las tarjetas de adquisición de datos basadas en tecnologías libres con el propósito de analizar sus prestaciones.
2. Analizar las tarjetas de adquisición de datos, a fin de definir la mas acorde en el dispositivo portátil.
3. Definir el protocolo de comunicación del dispositivo de adquisición de datos para elaborar un diseño de una interfaz virtual.
4. Diseñar una interfaz virtual basada en tecnologías libres que permita visualizar el comportamiento del dispositivo adquisición de datos.
5. Construir un sistema integrado con los componentes definidos, con el fin de realizar una demostración en el área de Instrumentación (lectura de sensores) y Sistemas Digitales.(manipular compuertas lógicas).

Justificación de la Investigación

La formación de profesionales de elevada calidad, tiene como principal apalancamiento la vivencia o experimentación; con mayor énfasis en áreas practico-técnicas donde se implementa los principios teóricos adquiridos en la formación académica, es por ello que diseñando un dispositivo portátil de bajo costo en comparación con los instrumentos de laboratorio tradicionales, se dispone de una herramienta de medición, generación y análisis de señales para asignaturas como Instrumentación Electrónica y Sistemas Digitales por mencionar las que más se adaptan a sus parámetros de medición, no escapando poder utilizarse en otras asignaturas, permitiendo comprender, vivencial y experimentar la teoría, mediante la práctica e inferir

que el estudiante tendrá a su alcance todas las herramientas elementales, para lograr una formación completa y sustentada.

Disponiendo el estudiante del dispositivo de adquisición de datos en su formación se podría garantizar la experimentación de la mayoría de los circuitos digitales que se pretenden estudiar y además se pueden hacer lecturas de la mayoría de sensores implícitos en la instrumentación, evidenciando así cada uno de los aspectos teóricos elementales inmerso en las asignaturas de instrumentación y Sistemas Digitales, obteniendo de esta manera el Instituto Universitario Politécnico Santiago Mariño extensión Maturín sede principal anexo A3 (IUPSM-A3) , mantener y elevar su status como una institución signada por la excelencia e innovación académica; beneficiándose así el profesorado, estudiantado y la institución en conjunto.

Debido a que esta investigación, estudio e implementación será basada en los principios fundamentales del Software Libre y Hardware Libre (Las Tecnologías Libres) donde se reutilizan técnicas, códigos fuentes, esquemas de circuitos y todas aquellas herramientas inmersas en un licenciamiento GPL, este trabajo de investigación y toda su documentación teórica y técnica estará disponible para su implementación, distribución, mejora y estudio, en todos los ámbitos nacionales e internacionales contribuyendo así al ámbito académico y sociedad en general.

Es importante porque propone una nueva metodología para un desarrollo completo de conocimientos en los alumnos y mejora las herramientas de los docentes para impartir conocimientos teóricos. Los estudiantes adquirirán una mejor destreza en la manipulación de los diferentes experimentos.

Al disponer el estudiante de esta herramienta que no está presente en el Laboratorio o un dispositivo similar, se presenta una ventaja al estudiante, aunado a esto con un bajo coste al estar enmarcado su implementación en las tecnologías libres donde no se deben comprar licencias de software y el

hardware implícito no presenta restricciones duplicación masiva por estar protegido con licenciamiento GPL es decir hardware Libre. Hasta se podría visionar como una herramienta en un futuro no muy lejano, como apoyo en una capacitación E-Learning (educación a distancia) de los conocimientos que comúnmente se imparten en locaciones acondicionadas para ese fin, permitiendo al alumno vencer dificultades de tiempo, ubicación y disponibilidad.

CAPÍTULO II

MARCO REFERENCIAL

Antecedentes de la Investigación

Luego de haber indagado en bibliografías, estudios y referencias sobre el poder permitir que los estudiante tengan las herramientas adecuadas para poder obtener una valiosa y completa absorción de los conocimientos académicos elementales que durante su formación en la carrera de ingeniería Electrónica del Instituto Universitario Politécnico Santiago Mariño extensión Maturín sede principal anexo A3 ,de la ciudad de Maturín, estado Monagas, se ha encontrado algunos trabajo previos que brindan un aporte significativo al Trabajo Especial de Grado, que de guían en el desarrollo del modelo de la investigación, además de bases teóricas relacionados con el tema :

Bustamante, R (2011), “Diseño y Construcción de un prototipo de adquisición de datos para variaciones de voltaje, corriente y temperatura en función del tiempo, utilizando comunicación Ethernet, para el laboratorio de física de la facultad de ciencias de la escuela politécnica nacional”, Escuela Politécnica Nacional de Quito, Ecuador. El objetivo de este trabajo de grado fue dotar al laboratorio de física con un instrumento de medición de tamaño y peso reducido capaz de realizar un registro de voltaje, corriente y

temperatura para la óptima realización de prácticas y proyectos relacionados con estas magnitudes físicas, para el posterior análisis gracias al desarrollo de la interfaz gráfica que permite la comunicación con el dispositivo via Ethernet. El aporte de esta investigación será la metodología utilizada en la adquisición de las magnitudes a medir por el dispositivo y el enfoque de la interfaz gráfica para presentar los resultados.

Elermid Oviedo (2011).”Diseño de una Interface de adquisición de data de vibración de las miniplantas compresoras Jusepin 3,4 y 5 del distrito Furrial PDVSA Oriente en Jusepin Estado Monagas”. Trabajo de grado de I.U.P Santiago Mariño. Este Proyecto estuvo fundamentado esencialmente en el diseño de una interfaz de adquisición de data para el control y monitoreo de las vibraciones de la miniplantas el cual con su registro posterior se podría ejecutar con anticipación los mantenimientos y correcciones respectivos. En este trabajo de grado el autor concluye que usando un monitoreo de forma continua de las variables críticas, se podría optimizar las fallas por vibración de la minipalantas compresoras Jusepin 3,4, y 5. Este trabajo de grado deja un aporte al Trabajo Especial de Grado, en como muestra los pasos para lograr un eficiente sistema, a su vez su enfoque filosófico en denotar la importancia de los sistemas de adquisición de datos y registros de los mismo para su posterior análisis.

Montalvo, J (2011). “Diseño e implementación de un sistema SCADA para Control De Procesos De un Módulo Didáctico de Montaje Festo Utilizando PLC y una Pantalla HMI”, caso práctico: En el Laboratorio de la FIE. Escuela Superior Politécnica de Chimborazo. Riobamba, Ecuador. El estudio inicialmente visualiza la carencia de equipos de control de procesos dentro de las instalaciones y pretende establecer la importancia de contar con tecnología de medir y controlar procesos que pueden ser simulados y visualizados en una interfaz hombre-máquina complementando asi el

proceso de aprendizaje práctico de los estudiantes. Desarrollando un sistema SCADA a un módulo didáctico existente conformado por un PLC y un HMI; esta investigación brindó información técnica importante, ya que el autor tuvo que estar en contacto directo con los equipos neurales que coinciden con los del presente proyecto que es de adquirir señales analógicas o digitales y mostrarlas en una interfaz virtual hombre-máquina.

Bases Teóricas

Sistemas de Adquisición de datos

La adquisición de datos o adquisición de señales, consiste en la toma de muestras del mundo real (sistema analógico o digital) para generar datos que puedan ser manipulados por un ordenador u otras electrónicas (sistema digital). Consiste, en tomar un conjunto de señales físicas, convertirlas en tensiones eléctricas y digitalizarlas de manera que se puedan procesar en una computadora. Se requiere una etapa de acondicionamiento, que adecua la señal a niveles compatibles con el elemento que hace la transformación a señal digital. El elemento que hace dicha transformación es el módulo de digitalización o tarjeta de Adquisición de Datos (DAQ).

.

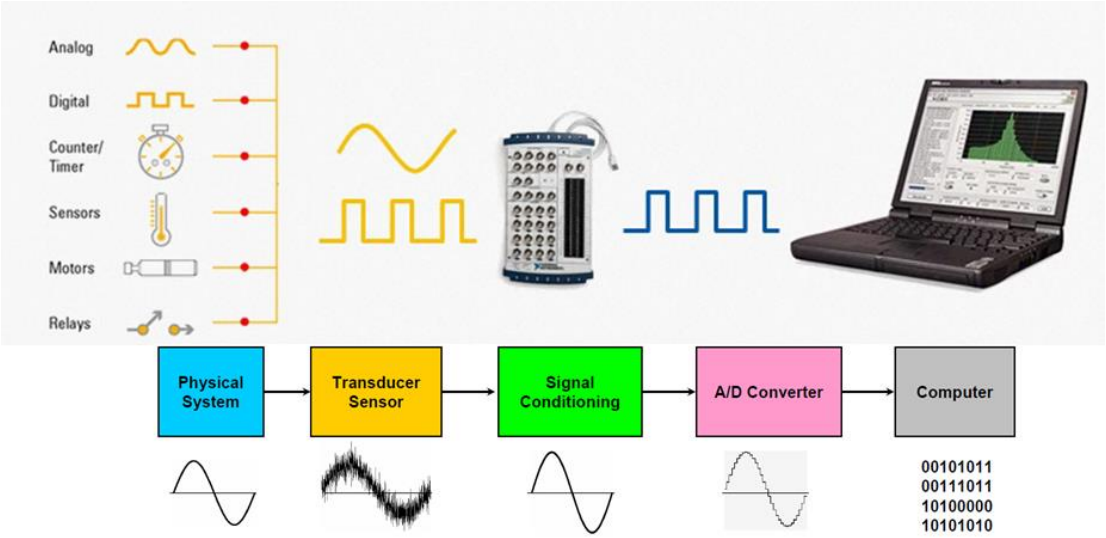


Figura 1 Sistema de adquisición de datos. ***Tomado de: http://wiki.genexus.com/commwiki***

Proceso de adquisición de datos

La adquisición de datos se inicia con el fenómeno físico o la propiedad física de un objeto (objeto de la investigación) que se desea medir. Esta propiedad física o fenómeno podría ser el cambio de temperatura o la temperatura de una habitación, la intensidad o intensidad del cambio de una fuente de luz, la presión dentro de una cámara, la fuerza aplicada a un objeto, o muchas otras cosas. Un eficaz sistema de adquisición de datos pueden medir todas estas diferentes propiedades o fenómenos.

Un sensor es un dispositivo que convierte una propiedad física o fenómeno en una señal eléctrica correspondiente medible, tal como tensión, corriente, el cambio en los valores de resistencia o condensador, etc. La capacidad de un sistema de adquisición de datos para medir los distintos

fenómenos depende de los transductores para convertir las señales de los fenómenos físicos mensurables en la adquisición de datos por hardware. Transductores son sinónimo de sensores en sistemas de DAQ. Hay transductores específicos para diferentes aplicaciones, como la medición de la temperatura, la presión, o flujo de fluidos. DAQ también despliega diversas técnicas de acondicionamiento de Señales para modificar adecuadamente diferentes señales eléctricas en tensión, que luego pueden ser digitalizados usando CED. Las señales pueden ser digitales (también llamada señales de la lógica) o analógicas en función del transductor utilizado.

El acondicionamiento de señales suele ser necesario si la señal desde el transductor no es adecuado para la DAQ hardware que se utiliza. La señal puede ser amplificada o des amplificada, o puede requerir de filtrado, o un cierre patronal, en el amplificador se incluye para realizar demodulación. Varios otros ejemplos de acondicionamiento de señales podría ser el puente de conclusión, la prestación actual de tensión o excitación al sensor, el aislamiento, linealización, etc. Este pre-tratamiento del señal normalmente lo realiza un pequeño módulo acoplado al transductor.

DAQ hardware son por lo general las interfaces entre la señal y un PC. Podría ser en forma de módulos que pueden ser conectados a la computadora de los puertos (paralelo, serie, USB, etc...) o ranuras de las tarjetas conectadas a (PCI, ISA) en la placa madre. Por lo general, el espacio en la parte posterior de una tarjeta PCI es demasiado pequeño para todas las conexiones necesarias, de modo que una ruptura de caja externa es obligatoria. Las tarjetas DAQ a menudo contienen múltiples componentes (multiplexores, ADC, DAC, TTL-IO, temporizadores de alta velocidad, memoria RAM). Estos son accesibles a través de un bus por un micro controlador, que puede ejecutar pequeños programas. El controlador es más

flexible que una unidad lógica dura cableada, pero más barato que una CPU de modo que es correcto para bloquear con simples bucles de preguntas.

Driver software normalmente viene con el hardware DAQ o de otros proveedores, y permite que el sistema operativo pueda reconocer el hardware DAQ y dar así a los programas acceso a las señales de lectura por el hardware DAQ. Un buen driver ofrece un alto y bajo nivel de acceso. Ejemplos de Sistemas de Adquisición y control: · DAQ para recoger datos (datalogger) medioambientales (energías renovables e ingeniería verde). · DAQ para audio y vibraciones (mantenimiento, test). · DAQ + control de movimiento(corte con laser). · DAQ + control de movimiento+ visión artificial (robots modernos).

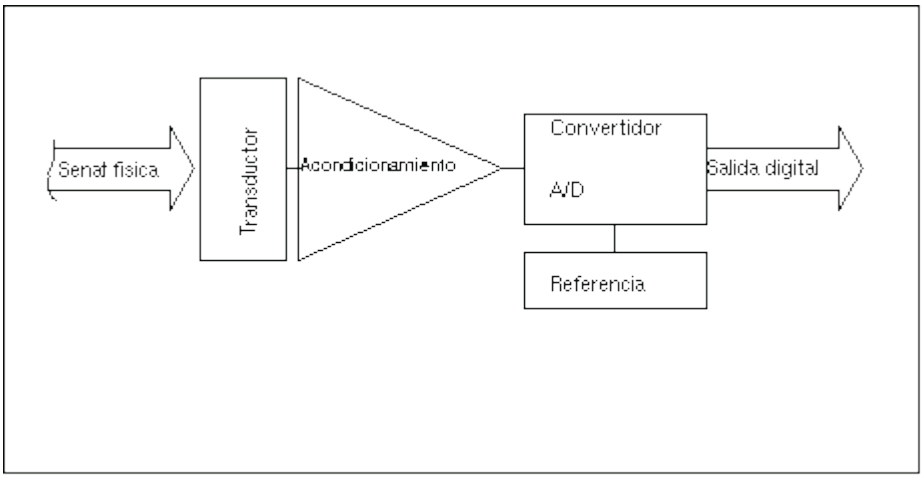


Figura 2 Esquema del proceso de adquisición de datos . *Elaborado para simplificar el proceso de conversión que ocurre en el proceso de adquisición de las señales físicas*.

Tiempo de Conversión de una Señal Analógica

Es el tiempo que tarda en realizar una medida el convertidor en concreto, y dependerá de la tecnología de medida empleada. Evidentemente nos da una cota máxima de la frecuencia de la señal a medir.
Este tiempo se mide como el transcurrido desde que el convertidor recibe una señal de inicio de "conversión" (normalmente llamada SOC, Start of Conversión) hasta que en la salida aparece un dato válido. Para que tengamos constancia de un dato válido tenemos dos caminos:

- Esperar el tiempo de conversión máximo que aparece en la hoja de características.
- Esperar a que el convertidor nos envíe una señal de fin de conversión.
- Si no respetamos el tiempo de conversión, en la salida tendremos un valor, que dependiendo de la constitución del convertidor será:
- Un valor aleatorio, como consecuencia de la conversión en curso
- El resultado de la última conversión

Teorema de muestreo de Nyquist-Shannon

El teorema de muestreo de Nyquist-Shannon, también conocido como teorema de muestreo de Whittaker-Nyquist-Kotelnikov-Shannon, teorema de

Nyquist, es un teorema fundamental de la teoría de la información, de especial interés en las telecomunicaciones.

Este teorema fue formulado en forma de conjetura por primera vez por Harry Nyquist en 1928 (Certain topics in telegraph transmission theory), y fue demostrado formalmente por Claude E. Shannon en 1949 (Communication in the presence of noise).

El teorema trata del muestreo, que no debe ser confundido o asociado con la cuantificación, proceso que sigue al de muestreo en la digitalización de una señal y que, al contrario del muestreo, no es reversible (se produce una pérdida de información en el proceso de cuantificación, incluso en el caso ideal teórico, que se traduce en una distorsión conocida como error o ruido de cuantificación y que establece un límite teórico superior a la relación señal-ruido). Dicho de otro modo, desde el punto de vista del teorema, las muestras discretas de una señal son valores exactos que aún no han sufrido redondeo o truncamiento alguno sobre una precisión determinada, es decir, aún no han sido cuantificadas.

Dicho de otro modo, la información completa de la señal analógica original que cumple el criterio anterior está descrita por la serie total de muestras que resultaron del proceso de muestreo. No hay nada, por tanto, de la evolución de la señal entre muestras que no esté perfectamente definido por la serie total de muestras. Ver fig 3.

Si la frecuencia más alta contenida en una señal analógica $x_a(t)$es $F_{max} = B$y la señal se muestrea a una tasa $F_s > 2F_{max} \equiv 2B$, entonces $x_a(t)$se puede recuperar totalmente a partir de sus muestras mediante la siguiente función de interpolación: $g(t) = \frac{\sin 2\pi Bt}{2\pi Bt}$

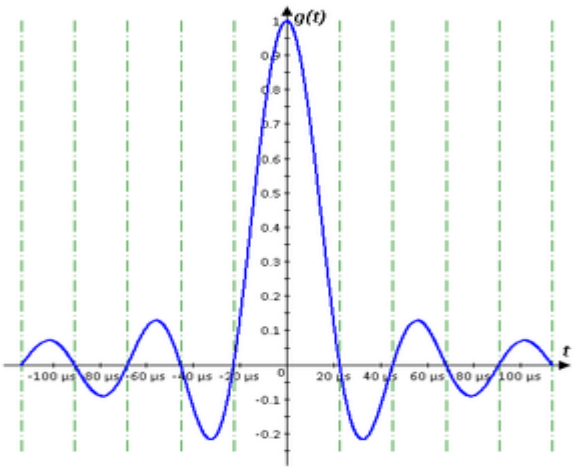


Figura 3 Función de interpolación g(t) para Fs=44100 mps. *Tomado de https://commons.wikimedia.org/wiki/User:Jmcalderon*

El teorema demuestra que la reconstrucción exacta de una señal periódica continua en banda base a partir de sus muestras, es matemáticamente posible si la señal está limitada en banda y la tasa de muestreo es superior al doble de su ancho de banda.

Así, $x_a(t)$ se puede expresar como:

$$x_a(t) = \sum_{n=-\infty}^{\infty} x_a\left(\frac{n}{F_s}\right) g\left(t - \frac{n}{F_s}\right)$$

donde $x_a\left(\frac{n}{F_s}\right) = x_a(nT) \equiv x(n)$ son las muestras de $x_a(t)$.

Hay que notar que el concepto de ancho de banda no necesariamente es sinónimo del valor de la frecuencia más alta en la señal de interés. A las señales para las cuales esto sí es cierto se les llama señales de banda base, y no todas las señales comparten tal característica (por ejemplo, las ondas de radio en frecuencia modulada). Si el criterio no es satisfecho, existirán frecuencias cuyo muestreo coincide con otras (el llamado aliasing) y están

ponderadas al valor de su correspondiente muestra (el máximo de cada función pasa por un punto azul que representa la muestra).

Interfaz Virtual por Computador

La interfaz Virtual o Instrumentación Virtual nace a partir del uso del computador personal (PC) como "instrumento" de medición de tales señales como temperatura, presión, caudal, etc. Es decir, el PC comienza a ser utilizado para realizar mediciones de fenómenos físicos representados en señales de corriente (Ej. 4-20 mA) y/o voltaje (Ej. 0-5Vdc). Sin embargo, el concepto de "instrumentación virtual" va más allá de la simple medición de corriente o voltaje, sino que también involucra el procesamiento, análisis, almacenamiento, distribución y despliegue de los datos e información relacionados con la medición de una o varias señales específicas. Es decir, el instrumento virtual no se conforma con la adquisición de la señal, sino que también involucra la interfaz hombre-máquina, las funciones de análisis y procesamiento de señales, las rutinas de almacenamiento de datos y la comunicación con otros equipos.

La interfaz Virtual funciona como una extensión de un instrumento real mostrado en un computador personal permitiendo ser diseñado a gusto del usuario y permitiendo abarcar gran cantidad de monitoreo de instrumentos en un solo entorno. Ver Figura 4 Vista de una interfaz virtual realizada con labview. Ver fig 4.

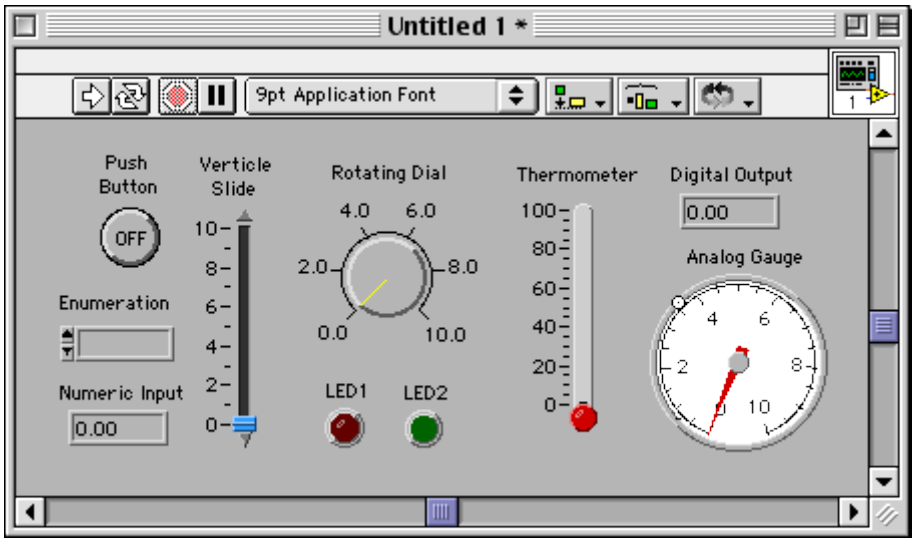


Figura 4 Vista de una interfaz virtual realizada con labview. Tomada de: http://sine.ni.com/cs/app/doc/p/id/cs-13030

Tecnologías Libres

La tecnología es el conjunto de conocimientos técnicos y científicos que permite desarrollar bienes y servicios que satisfagan las necesidades humanas y faciliten la adaptación al medio ambiente.

Las tecnologías libres son aquellas que no precisan de autorización o licencia para su uso. Más bien, pertenecen a la sabiduría y cultura popular, propias de la ciudadanía, que es quien las utiliza y explota en su propio beneficio. Las propuestas sobre tecnologías libres son, cada vezmás, perseguidas en los países ricos, condicionados por las políticas de patentes y copyright. Pero en los países pobres, el trabajo con estas soluciones es más frecuente, precisamente porque no dependen de ningún factor económico ni político que las prohíba, y en muchos casos se convierten en la única posibilidad de desarrollo.

La apuesta por alternativas de libre disposición es muy necesaria, ya que descentraliza el acceso a los recursos y reduce la dependencia que durante estos años se nos ha generado. La "tecnología libre" es aquella que respeta las libertades del conocimiento libre al protegerse con licencias de derechos de autor poco restrictivas como GNU, “creative commons” o dominio público. Resulta de principios científicos aplicados. Incluye Todas las ramas en que se usan tecnologías o técnicas centradas en el eco desarrollo y la sustentabilidad. Abordando cuestiones como en el Aprendizaje, el software libre, el hardware libre, el código abierto, y los estándares abiertos.

Estas tecnologías que permiten su libre reutilización, los productos y servicios generados con ellas no tienen necesariamente por qué ser gratuitos.

En Venezuela enmarcada en una era de cambios y en búsqueda de una independencia en todos los campos del saber y medios de producción, se crea en el 2006 la Fundación Centro Nacional de Desarrollo e Investigación en Tecnologías Libre(http://www.cenditel.gob.ve/), nacida de diversos proyectos de innovación de Fundacite Mérida, siendo esta institución en Venezuela la que apalanca el desarrollo de la filosofía de Hardware Libre en sus comienzo y posteriormente nacen otras iniciativas.

*Software Libre***:** ``Software Libre'' se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. De modo más preciso, se refiere a cuatro libertades de los usuarios del software: La libertad de usar el programa, con cualquier propósito (libertad 0). La libertad de estudiar cómo funciona el programa, y adaptarlo a tus necesidades (libertad 1). El acceso al código fuente es una condición previa para esto. La libertad de distribuir copias, con lo que puedes ayudar a tu vecino (libertad 2). La libertad de mejorar el programa y hacer públicas las mejoras a los demás, de modo que toda la comunidad se beneficie. (libertad 3). El acceso al código fuente es un requisito previo para esto.

Un programa es software libre si los usuarios tienen todas estas libertades. Así pues, deberías tener la libertad de distribuir copias, sea con o sin modificaciones, sea gratis o cobrando una cantidad por la distribución, a cualquiera y a cualquier lugar. El ser libre de hacer esto significa (entre otras cosas) que no tienes que pedir o pagar permisos. Ejemplode software libre en el ámbito de la electrónica son los siguientes:

KiCAD*:*

Es un programa de código libre (GPL) para la creación de esquemas electrónicos y circuitos impresos y simulación 3D de los prototipos, la suite Kicad es un conjunto de cuatro programas y un gestor de proyectos para realizar circuitos electrónicos:

- Schema :Creación de esquemas.
- PcbNew: Realización de circuitos impresos.
- Gerbview: Visualización de documentos generados en formato GERBER
- (Documentos de foto trazado).
- Cvpcb: Utilidad de selección de las huellas físicas de los componentes
- Electrónicos utilizados en el esquema.
- Kicad: Gestor de proyectos.

Figura 5 Diseños 3D y pcb con kicad

MyOpenLab:

Es un software libre escrito en Java, desarrollado para realizar simulaciones y modelar experimentos orientados al aprendizaje, utilizando y creando librerías de componentes, los cuales pueden ser fácilmente conectados.

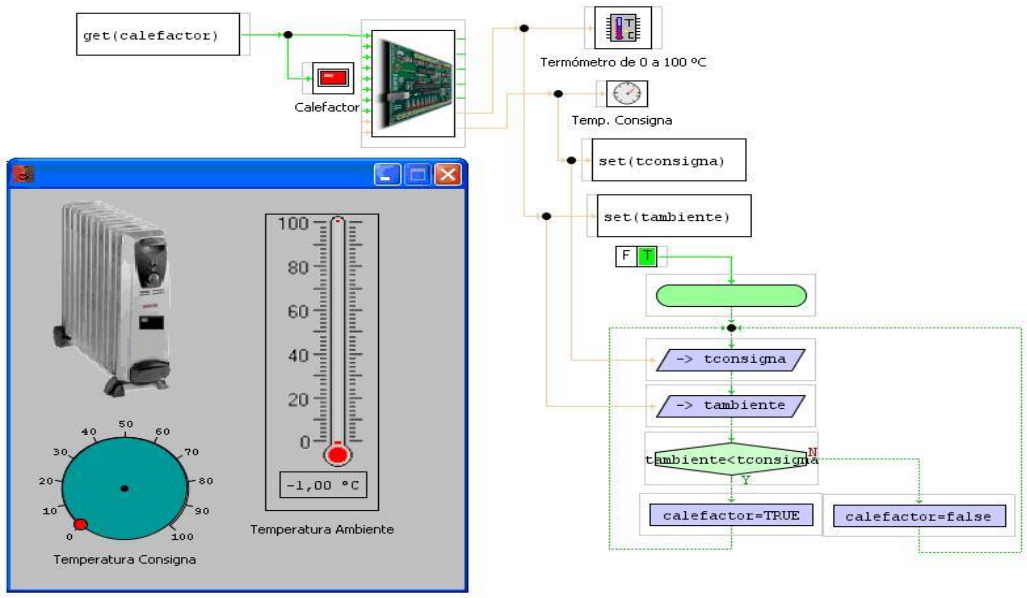


Figura 6.-Interfaz virtual con MyOpenLab

Sus características más importantes son:

- Amplias bibliotecas de funciones analógicas, digitales, visualización, etc.
- Tratamiento de diversos tipos de datos y operaciones con éstos.
- Programación mediante diagramas de flujo.
- Creación de pantallas de visualización que recojan el estado de las variables y eventos de las simulaciones.
- Posibilidad de ampliación de su librería de componentes.
- Posibilidad de creación de submodelos (componentes que encapsulan a otros componentes)
- Interconexión con el exterior (interfaces K8055, Arduino, etc.)

- Bloques de simulación para Robótica 2D y 3D.

Mediante MyOpenLab es posible diseñar instrumentos virtuales (VI) a través de los cuales se puede realizar una aproximación a los sistemas de medida y control de una manera mas realista.

Aplicaciones:

- ➢ Simulación de Circuitos digitales
- ➢ Simulación de Circuitos Analógicos
- ➢ Simulación de Instrumentos
- ➢ Simulación de Automatismos
- ➢ Modelado de Fenómenos Físicos
- ➢ Simulación de Automatismos
- ➢ Simulación de Robots
- ➢ Control de Elementos Físicos mediante Interfaces
- ➢ Tratamiento de Imágenes y Sonidos
- ➢ Operaciones con matrices y vectores 2D y 3D

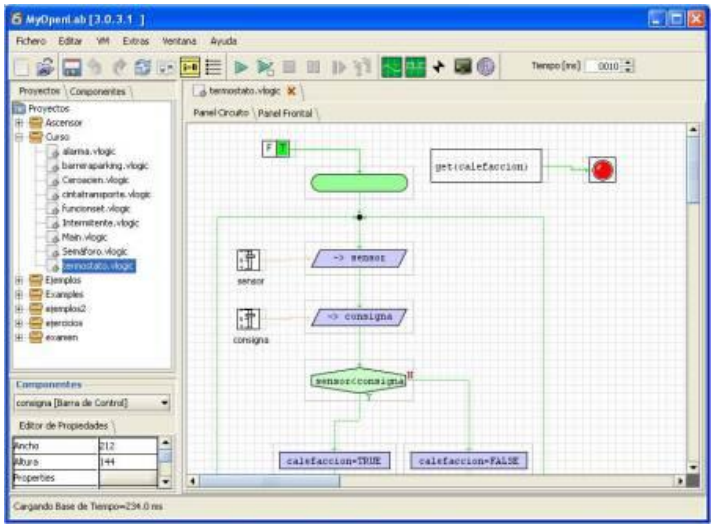


Figura 7 Diagrama de bloque lógico con MyOpenLab

.

Hardware Libre*:*

Son dispositivos de hardware cuyas especificaciones y diagramas esquemáticos son de acceso público, ya sea bajo algún tipo de pago o de forma gratuita. La filosofía del software libre (las ideas sobre la libertad del conocimiento) es aplicable a la del hardware libre. Se debe recordar en todo momento que libre no es sinónimo de gratis. El hardware libre forma parte de la cultura libre, algo que tiene en común el hardware con el software es que ambos corresponden a las partes tangibles de un sistema informático sus componentes son; eléctricos, electromecánicos y mecánicos son cables gabinetes o cajas.

Existen muchas comunidades que trabajan en el diseño, desarrollo y pruebas de hardware libre, y que además brindan soporte. Algunas de ellas son Open Collector, 5 OpenCores6 y el Proyecto gEDA.7.En Venezuela existe una comunidad recientemente de hardware libre llamada pingüino VE cuyo proyecto principal es una tarjeta micro controlador que compite directamente con arduino. http://www.pinguino.org.ve/. Algunos ejemplos de proyectos de hardware libre son los siguientes:

Arduino: Es una tarjeta electrónica de computación física basada en una sencilla placa con un micro controlador y un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring (basado en C++); se puede utilizar para desarrollar objetos interactivos autónomos, las placas se pueden montar a mano o adquirirse. El entorno de desarrollo integrado libre se puede descargar gratuitamente, Al ser open-hardware, tanto su diseño como su distribución son libres. Es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia. Ver fig 8.

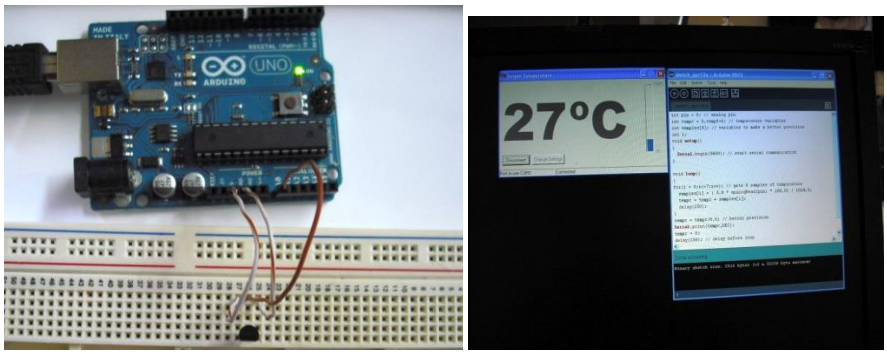


Figura 8.- Ejemplo de Arduino como Sensor de Temperatura. Tomado de: https://learn.adafruit.com/adafruit-arduino-lesson-1-blink/overview

Pingüino: Es una tarjeta como el Arduino pero basada en micro controlador PIC. La meta de este proyecto es de construir un IDE de fácil utilización en LINUX, WINDOWS y MAC OS X. En Venezuela hay un proyecto autóctono basado en este proyecto Francés llamado Pingüino VE, Apoyado ampliamente por entes del gobierno como: CENDITEL, CNTI, MCTI y Proyecto Canaima.

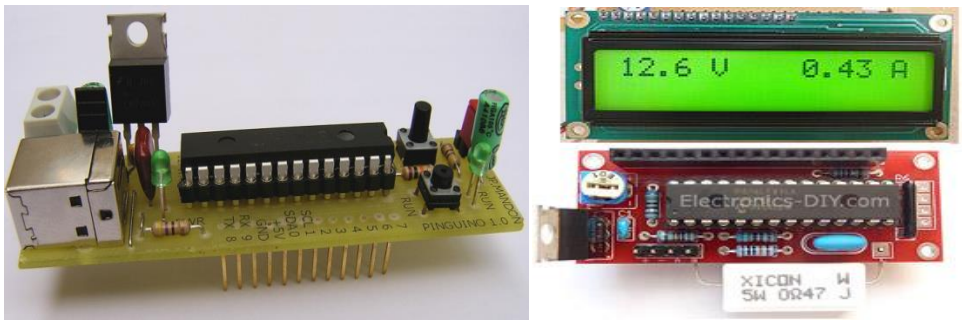


Figura 9.- Ejemplo de Pingüino como Volt- Amperímetro. Tomado de : http://www.electronics-diy.com/70v_pic_voltmeter_amperemeter.php

Bases Legales

Ley Orgánica de Ciencia, Tecnología e Innovación (LOCTI)

Artículo 5.: Las actividades de ciencia, tecnología, innovación y sus aplicaciones, así como, la utilización de los resultados, deben estar encaminadas a contribuir con el bienestar de la humanidad, la reducción de la pobreza, el respeto a la dignidad, a los derechos humanos y la preservación del ambiente.

Artículo 23.: El Ministerio de Ciencia y Tecnología fomentará y desarrollará políticas y programas, tendientes a orientar la cooperación internacional a objeto del fortalecimiento del Sistema Nacional de Ciencia, Tecnología e Innovación.

Artículo 24.:

El Ejecutivo Nacional, a través del Ministerio de Ciencia y Tecnología, podrá crear los centros de investigación que considere necesarios para promover la investigación científica y tecnológica en las áreas prioritarias de desarrollo económico y social del país.

Decreto en Gaceta Oficial

39.109 - Normas Técnicas de Formato Abierto de Documentos ODF PDF y portales web de la APNEl uso de los formatos de archivos ODF y PDF, así como las características mínimas de los portales web de la Administración

Pública (AP) cuentan con el soporte jurídico que garantizará la gobernabilidad de las TI en la AP.

Gaceta Oficial 39.633: En cuya ordenanza 025, artículo 5 dice “Los Órganos y Entes de la Administración Pública Nacional deben incluir en los términos de referencia de aquellos contratos que tengan por objeto la adquisición de estaciones de trabajo, el requerimiento de justificar su funcionamiento bajo la distribución Canaima GNU/Linux sin la necesidad de la instalación adicional de componentes o partes privativas o cerradas para su operatividad; debiendo además ser éste el único sistema instalado en los equipos desestimando las ofertas que no cumplan esta condición.

Constitución de la República Bolivariana de Venezuela

Articulo 110: El Estado reconocerá el interés público de la ciencia, la tecnología, el conocimiento, la innovación y sus aplicaciones y los servicios de información necesarios por ser instrumentos fundamentales para el desarrollo económico, social y político del país, así como para la seguridad y soberanía nacional. Para el fomento y desarrollo de esas actividades, el Estado destinará recursos suficientes y creará el sistema nacional de ciencia y tecnología de acuerdo con la ley. El sector privado deberá aportar recursos para las mismas. El Estado garantizará el cumplimiento de los principios éticos y legales que deben regir las actividades de investigación científica, humanística y tecnológica. La ley determinará los modos y medios para dar cumplimiento a esta garantía.

CAPÍTULO III

MARCO METODOLÓGICO

Modalidad de la Investigación

La investigación propuesta, está basada en la modalidad de Proyecto Factible, ya que persigue como objetivo solucionar la disponibilidad de una herramienta fundamental para la enseñanza-aprendizaje en el área de Sistemas Digitales e Instrumentación Electrónica, como lo es un dispositivo de adquisición de datos que permita comprender y manipular Sistemas Digitales e interpretar señales analógicas de instrumentos de medición. Según el Manual de la UPEL, (2006) define al "Proyecto Factible" como:

> La investigación, elaboración y desarrollo de una propuesta de un modelo operativo viable para solucionar problemas, requerimientos o necesidades de organizaciones o grupos sociales, puede referirse a la formación de políticas, programas, tecnologías, métodos o procesos. El proyecto debe tener apoyo en una investigación de tipo documental, de campo o un diseño que incluya amabas modalidades (p21).

Por lo tanto, este equipo permitirá a los estudiantes de la carrera de Ingeniería Electrónica contar con una herramienta para la enseñanza practica y así lograr absorber con mayor facilidad la teoría presente en el área de Sistemas Digitales e permitiendo el ahorro de tiempo y dinero debido al alto costo de equipos comerciales actuales en el mercado venezolano.

Tipo de Investigación de Acuerdo al Nivel

Para establecer el tipo de investigación que se realiza, el investigador debe profundizar en el grado de objeto o fenómeno que se desea estudiar. Cuando se aborda un tema de investigación es necesario saber cuál es el nivel de dificultad, para poder atacarlo de manera sistemática con las posibles soluciones a este problema.

Cabe destacar que los tipos de investigación difícilmente se presentan de una sola forma; generalmente se combinan entre si y obedecen a la aplicación de la investigación como tal. Según Sampieri Roberto (1997, Pág. 60)

> "Las investigaciones descriptivas buscan especificar las propiedades importantes de personas, grupos, comunidades o cualquier otro fenómeno que sea sometido a análisis. Miden o evalúan diversos aspectos, dimensiones o componentes del fenómeno o fenómenos a investigar".

Así mismo Arias G. Fidias (1999, Pág. 20) expresa que:

> "Los estudios descriptivos miden de forma independiente las variables, y aun cuando no se formulen hipótesis, las primeras aparecerán enunciadas en los objetivos de investigación".

De acuerdo a lo antes mencionado, esta investigación se clasifica del tipo descriptiva ya que busca estudiar las características importantes de la herramienta de un laboratorio de electrónica y con eso poder desarrollar un Dispositivo de adquisición de datos con interfaz Virtual Basado en tecnología libres y realizar una demostración práctica para las asignaturas de Instrumentación y Sistemas Digitales.

Unidad de Estudio

En vista de que se desea realizar un dispositivo que sea de carácter didáctico que permita a los estudiantes de la carrera de ingeniería Electrónica servir como una herramienta útil para la enseñanza-aprendizaje en el área de Sistemas Digitales e Instrumentación del Instituto Universitario Politécnico Santiago Mariño, se tomaron como unidad de estudio el análisis de la lógica booleana de las compuertas digitales y la interacción de sensores electrónicos utilizando una fotocelda. Haciendo referencia a lo antes mencionado, Hurtado (2008) señala que:

> "La Unidad de Estudio se refiere al contexto, al ser o entidad poseedores de la característica, evento, cualidad o variable, que se desea estudiar; una unidad de estudio puede ser una persona, un objeto, un grupo, una extensión geográfica o una Institución."(Pág. 141).

Es por ello que la elaboración de este proyecto está orientada a los Sistemas Digitales E Instrumentación Electrónica, presentes en la carrera de Ingeniería Electrónica del Instituto Universitario Politécnico "Santiago Mariño" Extensión Maturín. Debido a que no existen módulos didácticos que permitan afianzar la teoría mediante la experimentación. Por tal motivo en el presente proyecto se desarrollara un Dispositivo de adquisición de datos con interfaz Virtual Basado en tecnología libre, que sea accesible en el mercado venezolano y de bajo costo para la comunidad estudiantil.

Técnicas e Instrumento de Recolección de Datos

Toda investigación basa sus conclusiones en los resultados obtenidos luego del procesamiento y análisis de un conjunto de datos obtenidos directamente de la población estudiada, por ello es indispensable aplicar diversas técnicas que permitan la extracción de esa información vital, sin la cual es imposible desarrollar un proyecto de forma exitosa. Hurtado (2008), hablando al respecto, dice que:

> "Las técnicas tienen que ver con los procedimientos utilizados para la recolección de los datos, es decir, el cómo, estas pueden ser la revisión documental, observación directa, y las entrevistas no estructuradas" (p.153).

De lo anterior se deduce que los Instrumentos representan la herramienta importante en la cual va a cumplir con los objetivos y metas planteadas en el proyecto., por lo tanto, las técnicas de recolección de datos que serán utilizadas en la presente investigación son:

Revisión Documental

Hurtado (2008) la define la revisión documental como :

> "el proceso mediante el cual un investigador recopila, revisa, analiza, selecciona extrae diversas fuentes acerca de un tema particular" (p.119).

para elaborar el proyecto será necesario consultar diversas fuentes escritas en relación a los procesos que se llevan a cabo en el las prácticas de Laboratorio para cumplir con asignaturas de Instrumentación y Sistemas

Digitales, con el fin de identificar las mejoras que se pueden efectuar con el Dispositivo de adquisición de datos con interfaz Virtual Basado en tecnología libres propuesto.

Observación Directa

Hurtado (2008) señala que:

"…la observación directa es la técnica de investigación que consiste en ver u oír hechos o fenómenos que se deseen estudiar, para este fin adoptar modalidades y utilizar una serie de medidas e instrumentos que son propios…". (p.461).

Dado que la investigación se llevará a cabo en los laboratorios de electrónica de Instituto Universitario Politécnico Santiago Mariño extensión Maturín sede principal anexo A3 (IUPSM-A3) será posible tener contacto directo con el personal que allí labora, de esta forma se podrá presenciar y observar cómo se realizan procedimientos y las actividades dentro del mismo permitiendo recoger los datos.

Técnicas de Procesamiento y Análisis de Datos

Los datos adquiridos mediante las técnicas de recolección anteriormente expuestas, deberán ser analizados con el fin de obtener la información necesaria para la elaboración del modelado, y de esta forma dar solución al problema planteado y cumplir con los objetivos de la investigación, para ello se utilizará el Análisis de contenido, Hurtado (2008) expresa que:

> La técnica de análisis constituye un proceso que involucra la clasificación las codificaciones, el procedimiento y la interpretación de la información obtenida durante la recolección de datos, la finalidad del análisis es llegar a conclusiones específicas en relación al evento de estudio y dar respuesta a la pregunta de investigación (p83).

CAPÍTULO IV

RESULTADOS

Para la elaboración del dispositivo portátil de adquisición de datos con interfaz virtual basado en tecnologías libres para los estudiantes de electrónica del Instituto Universitario politécnico Santiago Mariño extensión Maturín, es necesario realizar una serie de actividades para cumplir con la finalidad de los objetivos específicos planteados.

Estudio las de tarjetas de adquisición de datos basadas en tecnologías libres.

Para el estudio de las tarjetas de adquisición de datos basadas en tecnologías libre tenemos un cuadro comparativo de las placas Arduino ya que son ampliamente conocidas como una plataforma de electrónica abierta (open Hardware)

Arduino nació como un proyecto educativo allá por el año 2005 sin pensar que algunos años más tarde se convertiría en líder del mundo DIY (Do It Yourself). Su nombre viene del nombre del bar Bar di Re Arduino donde Massimo Banzi pasaba algunas horas, el cual a su vez viene del nombre de un antiguo rey europeo allá por el año 1002. Banzi dice que nunca surgió como una idea de negocio, es más nació por una necesidad de subsistir ante

el eminente cierre del Instituto de diseño Interactivo IVREA en Italia. Es decir, al crear un producto open hardware (de uso público) no podría ser embargado. Es más hoy en día Arduino tiene la difícil tarea de subsistir comercialmente y continuar en continuo crecimiento.

Para su creación participaron alumnos que desarrollaban sus tesis como Hernando Barragan (Colombia) quien desarrollo la plataforma de programación Wiring con la cual se programa el microcontrolador.

Hoy en día con Arduino se pueden fabricar infinidad de prototipos y cada ves su uso se viene expandiendo más. Desde cubos de leds, sistemas de automatización en casa (domotica), integración con el Internet, displays Twitter, kit analizadores de ADN.

Google ha apostado por el proyecto y ha colaborado en el Android ADK (Accesory Development Kit), una placa Arduino capaz de comunicarse directamente con spmartphones Android para obtener las funcionalidades del teléfono (GPS, acelerómetros, GSM, abases de datos) y viceversa para que el teléfono controle luces, motores y sensores conectados de Arduino.

El primer prototipo fue desarrollado en el instituto IVRAE pero aún no se llamaba Arduino.

El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basasdo en Wiring) y el entorno de desarrollo Arduino (basado en Processing).

Las placas pueden ser hechas a mano o compradas las cuales fueron hechas en una fábrica; el software puede ser descargado de forma gratuita. Los ficheros de diseño de referencia (CAD) están disponibles bajo una licencia abierta, la cual puede ser adaptada a necesidades específicas. Ver Fig 10.

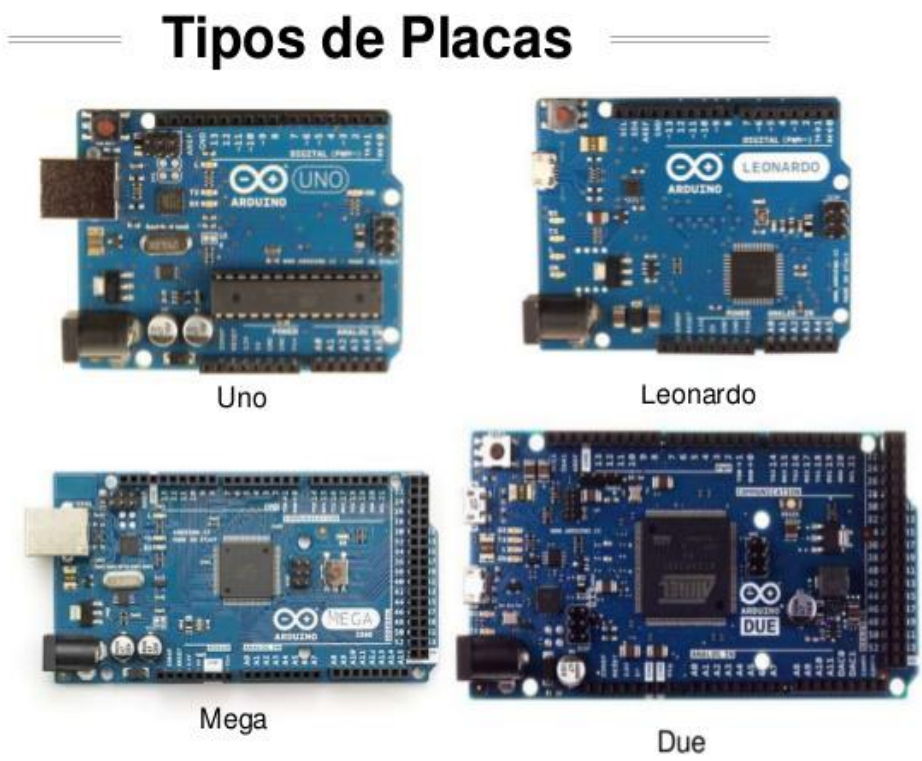


Figura 10 Tipos de placas Arduino comunes. Tomado de : https://www.arduino.cc/

Características de los modelos más comunes de Arduino

A través del cuadro Nº 1 se muestran las características para poder hacer una comparación entre las placas más comunes de Arduino y de esta forma poder tener claro cuáles son la diferencias técnicas, bondades cantidad de entradas/salidas digitales, analógicas, frecuencia de trabajo, voltajes de alimentación y demás parámetro de funcionamiento

.

Cuadro 1 –Comparativa de Tarjetas de Adquisición de datos. Tomado de : https://www.arduino.cc/

Modelo	Arduino UNO	Arduino Leonardo	Arduino Mega 2560	Arduino DUE
Microcontroller	ATmega 328	ATmega 32u4	ATmega2560	AT91SAM3X8E
Operating Voltage	5V	5V	5V	3.3V
Input Voltage	7-12V	7-12V	7-12V	7-12V
Input Voltage (limits)	6-20V	6-20V	6-20V	6-20V
Digital I/O Pins	14	20	54	54
Digital I/O Pins PWM output	6	7	15	12
Analog Input Pins	6	12	16	12
Analog Outputs Pins				2 (DAC)
Total DC Output Current on all I/O lines	40 mA	40 mA	40 mA	130 mA
DC Current for 3.3V Pin	50 mA	50 mA	50 mA	800 mA
DC Current for 5V Pin				800 mA
Flash Memory	32 KB 0.5 KB used by bootloader	32 KB KB used by bootloader	256 KB8 KB used by bootloader	512 KB available
SRAM	2 KB (ATmega328)	2.5 KB	8 KB	96 KB two banks: 64KB y 32KB
EEPROM	1 KB (ATmega328)	1 KB	4 KB	
Clock Speed	16 MHz	16 MHz	16 MHz	84 MHz
Tipo de USB	Estandar	Mini	Estándar	Mini

Funcionamiento de Arduino UNO

El funcionamiento del hardware de la placa de ARDUINO UNO permite determinar dar una visión de lo que se requiere para su implementación y está compuesto de la siguiente manera: (fig 11).

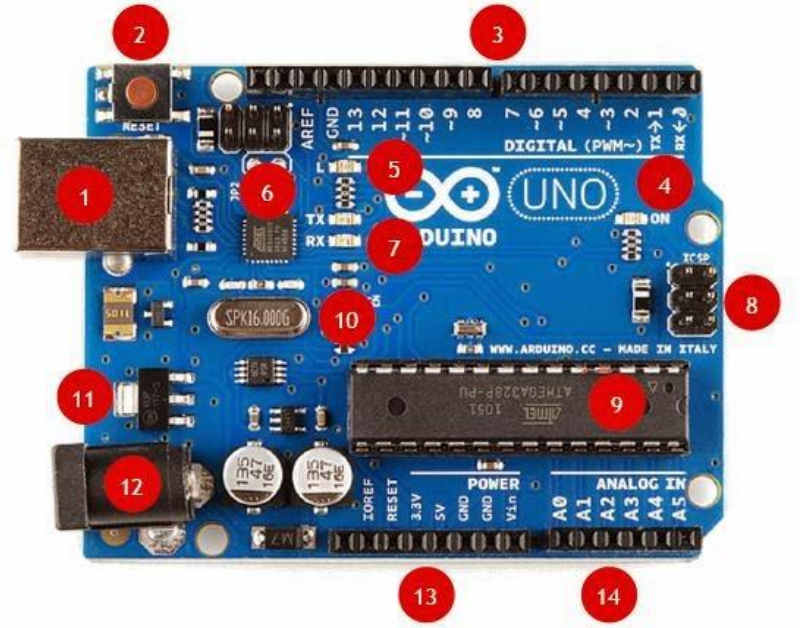


Figura 11 Identificación de Arduino UNO. Tomado de : https://www.arduino.cc/uno

1. Cable USB para conectar a PC.
2. Botón de RESET.
3. Pines de Entrada/Salida digital y PWM.
4. Mini LED verde de ON (placa alimentada o encendida).
5. Mini LED naranja conectado al PIN 13.
6. ATmega 16U2, responsable de las comunicaciones con el PC.
7. LED TX (transmisor) y LED RX (receptor) de la comunicación serial.
8. Puerto ICSP para programación serial.
9. Microcontrolador ATmega 328 (el cerebro de ARDUINO)
10. Cristal de cuarzo de 16 MHz.
11. Regulador de voltaje.
12. Conector para alimentación.
13. Pines de voltaje y tierra.
14. Entradas analógicas.

Especificaciones técnicas

Cuadro 2 Especificaciones Técnicas de Arduino. Tomado de : https://www.arduino.cc/Uno

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current for I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328)
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Vamos a explicar con un mayor detalle los tres bloques de pines más importantes de la placa. fig 12:

Figura 12 Bloque de Alimentación de Arduino.Tomado de: https://www.arduino.cc/Uno

Bien se energiza al Arduino mediante la conexión USB o mediante una fuente externa (recomendada de 7-12V), para obtener unas salidas de

tensión continua debido a unos reguladores de tensión y condensadores de estabilización.

Estos pines son:

- VIN: se trata de la fuente tensión de entrada que contendrá la tensión a la que estamos alimentando al Arduino mediante la fuente externa.
- 5V: fuente de tensión regulada de 5V, esta tensión puede venir ya sea de pin VIN a través de un regulador interno, o se suministra a través de USB o de otra fuente de 5V regulada.
- 3.3V: fuente de 3.3 voltios generados por el regulador interno con un consumo máximo de corriente de 50mA.
- GND: pines de tierra.
- 3. Entradas/salidas digitales

Figura 13 Bloque de Salidas Digitales y PWM Tomado de : https://www.arduino.cc/uno-bloque

Cada uno de los 14 pines digitales (Fig 13) se puede utilizar como una entrada o salida. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia de pull-up (desconectado por defecto) de 20 a 50 kOhm. Además, algunos pines tienen funciones especializadas como:

- Pin 0 (RX) y 1 (TX). Se utiliza para recibir (RX) y la transmisión (TX) de datos serie TTL.
- Pin 2 y 3. Interrupciones externas. Se trata de pines encargados de interrumpir el programa secuencial establecido por el usuario.
- Pin 3, 5, 6, 9, 10 y 11. PWM (modulación por ancho de pulso). Constituyen 8 bits de salida PWM con la función analogWrite ().
- Pin 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Estos pines son de apoyo a la comunicación SPI.
- Pin 13. LED. Hay un LED conectado al pin digital 13. Cuando el pin es de alto valor, el LED está encendido, cuando el valor está bajo, es apagado.
- 14. Entradas analógicas.

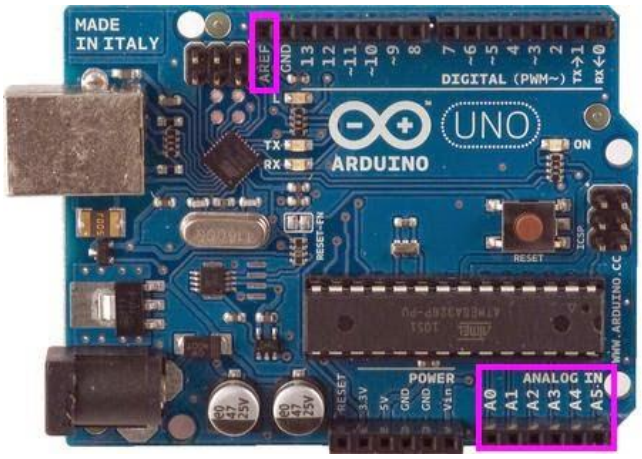


Figura 14 Bloque de Voltaje Referencial y Entradas Analógicas Tomado de : https://www.arduino.cc/uno-analog

El Arduino UNO posee 6 entradas analógicas, etiquetadas desde la A0 a A5, cada una de las cuales ofrecen 10 bits de resolución (es decir, 1024 estados). Por defecto, tenemos una tensión de 5V, pero podemos cambiar este rango utilizando el pin de AREF y utilizando la función

analogReference(), donde le introducimos una señal externa de continua que la utilizara como referencia.

Funcionamiento de Arduino Leonardo

Es una placa electrónica basada en un microcontrolador. En este caso, hablamos del ATmega32u4, una de las características de la placa es que cuenta con 20 pines de entrada/salida digitales, de los cuales, podemos usar 7 como salidas PWM (Pulse-Width Modulation) y 12 como entradas analógicas. Además cuenta con un oscilador de cristal que funciona a 16 MHz, una conexión micro USB, un conector de alimentación, una cabecera de ICSP (In-Circuit Serial Programming) y, un botón de reinicio.

Una de las novedades que incorpora el Leonardo, es que el ATmega32u4 incorpora comunicación USB, eliminando la necesidad de un procesador secundario.

La placa Arduino Leonardo (fig. 15) también posee, aparte del objeto “Serial”, del objeto “Serial1”. En este caso es para separar las dos posibles vías de comunicación serie que puede manejar el chip TTL-UART incorporado dentro del ATmega32U4: el objeto “Serial1” se reserva para la transmisión de información a través de los pines 0 (RX) y 1 (TX), y el objeto “Serial” se reserva para esa misma transmisión serie pero realizada a través de la comunicación USB-ACM (que a su vez es diferente de la comunicación USB usada para las simulaciones de teclado y ratón).

Hay que tomar en cuenta también, que, a diferencia de lo que ocurre con Arduino UNO, la ejecución de un sketch en la placa Leonardo no se reinicia cuando se abre el “Serial monitor”, por lo que no se podrán ver los datos que ya han sido enviados por la placa previamente al computador (como por ejemplo, los envios dentro de la función “setup()” con Serial.print() o similares). Para sortear este inconveniente, se escribe la siguiente línea

justo después de Serial.begin(): while(!Serial){;} Esto hará que mientras no se abra la comunicación serie (mientras no se abra el “Monitor serial”), el sketch no haga nada y se mantenga en espera.

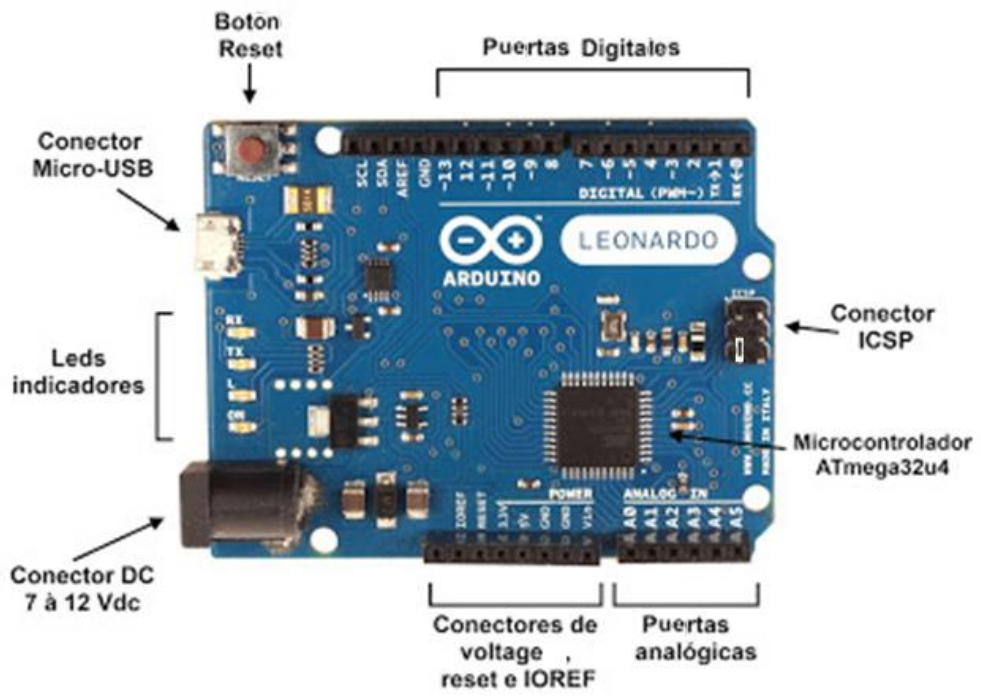


Figura 15 Identificación de Arduino Leonardo. Tomado de : https://www.arduino.cc/leonardo-vista.html

.

Alimentación: La placa Arduino Leonardo, puede ser alimentada a través de la conexión micro USB o con una fuente de alimentación externa. La fuente de alimentación se selecciona automáticamente.

Para usar alimentación externa (no mediante USB), ésta puede venir de un adaptador de CA a CC o de una batería. El adaptador (transformador), se conecta mediante un jack de 2.1mm con centro positivo a la toma de alimentación de la placa. Sin embargo, los cables de los que disponga la batería, pueden insertar en los pines GND y Vin de la placa.

La placa puede operar mediante una alimentación externa de 6 a 20 Voltios. En caso de alimentar la placa con menos de 7Voltios, el pin de 5Voltios puede no llegar a este valor y la placa podría volverse inestable. Sin embargo, si se utiliza más de 12Voltios, el regulador de tensión puede sobrecalentarse y llegar a dañar la placa. Por lo tanto, el rango recomendado de alimentación es de 7 a 12 Voltios.

Los pines de alimentación son los siguientes:

VIN : El voltaje de entrada a la placa Arduino cuando se utiliza una fuente de alimentación externa (a diferencia de 5 Voltios de la conexión USB o de otra fuente de alimentación regulada). Se puede suministrar tensión a través de este pin, o, si el suministro de tensión se realiza a través del transformador, podremos acceder a través de este pin.

- 5V. Fuente de alimentación regulada usada para alimentar el microcontrolador y otros componentes en la placa.
- 3V3. Un suministro de 3.3 Voltios, generados por el integrado. Consumo de corriente máxima: 50 mA.
- GND. Pines de masa (0 Voltios).
- IOREF. Tensión a la cual los pines de entrada y salida están funcionando.

Memoria : El ATmega32u4 que incorpora el Arduino Leonardo, posee 32 KB (4 KB utilizados para el bootloader). También ofrece 2,5 KB de SRAM y 1 KB de EEPROM (los cuales pueden ser leídos y escritos con la librería EEPROM).

Entradas y salidas: Cada uno de los 20 pines digitales de entrada/salida que posee la placa, se pueden utilizar mediante las funciones de siempre: pinMode(), digitalWrite() o digitalRead().

Cada uno de los 20 pines de E/S digitales se pueden utilizar como entrada o salida, usando las funciones pinMode(), digitalWrite() y digitalRead(). Cada pin proporcionará un máximo de 40 mA y, tampoco podrá recibir más de este valor.

- Serial: 0 (RX) y 1 (TX). Se utiliza para recibir (RX) y transmitir (TX) datos serie TTL con la capacidad de hardware serie del ATmega32U4. Hay que tener en cuenta que, en Arduino Leonardo, la clase Serial se refiere a la comunicación USB (CDC) ; para serie TTL en los pines 0 y 1, se utiliza la clase Serial1.
- IST: 2 (SDA) y 3 (SCL). Soportan la comunicación TWI utilizando la librería Wire.
- Interrupciones externas: 2 y 3. Estos pines se pueden configurar para desencadenar una interrupción en un valor bajo, un flanco ascendente o descendente, o un cambio en el valor. Ver attachInterrupt () para obtener más detalles.
- PWM: 3, 5, 6, 9, 10, 11 y 13. Proporcionan 8-bit de salida PWM con la función analogWrite ().
- SPI: en la cabecera de ICSP. Estos pines soportan la comunicación SPI utilizando la librería SPI. Se Debe tener en cuenta que los pines de SPI no está conectados a cualquiera de los pines digitales de entrada/salida, como están en Arduino UNO. Sólo están disponibles en el conector ICSP.
- LED: 13. Hay un LED smd conectado al pin digital 13. Cuando el pin tiene valor alto, el LED está encendido, cuando se pasa a valor bajo, se apaga.
- Entradas Analógicas: A0-A5, A6 – A11 (en los pines digitales 4, 6, 8, 9, 10, y 12). La placa Arduino Leonardo tiene 12 entradas analógicas,

las cuales van desde A0 hasta A11. También pueden ser utilizadas como entradas/salidas digitales. Los pines A0-A5 están situados en el mismo lugar que en Arduino UNO; las entradas A6-A11 se corresponden con las entradas/salidas digitales de los pines 4, 6, 8, 9, 10 y 12, respectivamente. Cada entrada analógica puede proporcionar 10 bits de resolución, es decir 1024 valores diferentes (desde 0 hasta 1023). Por defecto, la medida de las entradas analógicas van desde 0 a 5 Voltios, aunque se puede cambiar el extremo superior de su rango con el pin AREF y la función analogReference().

- AREF. Tensión de referencia para las entradas analógicas. Se utiliza con analogReference().
- Reset: reinicia el microcontrolador.

Funcionamiento de Arduino Mega

La placa Arduino Mega dispone de cuatro chips TTL-UART. Esto significa que puede utilizar hasta cuatro objetos serie, llamados “Serial”, “Serial1”, “Serial2” y “Serial3”. El primero sigue estando asociado a los pines 0 y 1, el objeto “Serial1” está asociado a los pines 18 (TX) y 19 (RX), el “Serial2” a los pines 16 (TX) y 17 (RX) y el “Serial3” a los pines 14 (TX) y 15 (RX). Cada uno de estos objetos se puede abrir independientemente (escribiendo Serial.begin(9600); Serial1.begin(9600); Serial2.begin(9600); o Serial3.begin(9600); respectivamente), y pueden enviar y recibir datos también independientemente. Pero a pesar de tener 4 conexiones serial, el único objeto asociado también a la conexión USB es “Serial” (porque es el único conectado al chip conversor ATmega16U2). Arduino Mega 2560 es

una versión ampliada de la tarjeta original de Arduino y está basada en el microcontrolador Atmega2560.(fig 16).

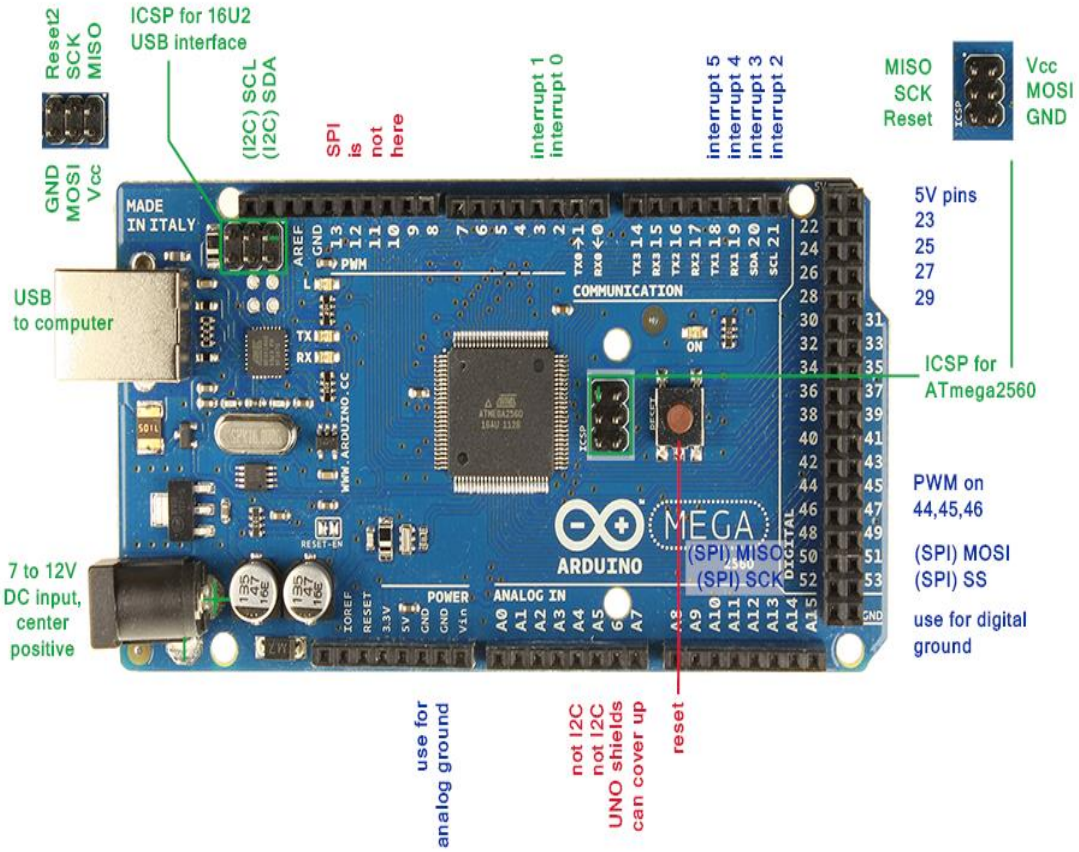


Figura 16 Identificación Arduino Mega. Tomado de : https://www.arduino.cc/mega-descripcion.html

Dispone de 54 entradas/salidas digitales, 14 de las cuales se pueden utilizar como salidas PWM (modulación de anchura de pulso). Además dispone de 16 entradas analógicas, 4 UARTs (puertas series), un oscilador de 16MHz, una conexión USB, un conector de alimentación, un conector ICSP y un pulsador para el reset. Para empezar a utilizar la placa sólo es necesario conectarla al ordenador a traves de un cable USB , o alimentarla con un adaptador de corriente AC/DC. También, para empezar, puede alimentarse

mediante una batería. Una de las diferencias principales de la tarjeta Arduino MEGA es que no utiliza el convertidor USB-serie de la firma FTDI. Por lo contrario, emplea un microcontrolador Atmega8U2 programado como convertidor USB a serie. La tarjeta Arduino MEGA es compatible con la mayoria de los shield o tarjetas de aplicación/ampliación disponibles para las tarjetas Arduino UNO original.

Las características principales son:

- Microprocesador ATmega2560
- Tensión de alimentación (recomendado) 7-12V
- Integra regulación y estabilización de +5Vcc
- 54 líneas de Entradas/Salidas Digitales (14 de ellas se pueden utiliza como salidas PWM)
- 16 Entradas Analogicas
- Maxima corriente continua para las entradas: 40 mA
- Salida de alimenatción a 3.3V con 50 mA
- Memoria de programa de 256Kb (el bootloader ocupa 8Kb)
- Memoria SRAM de 8Kb para datos y variables del programa
- Memoria EEPROM para datos y variables no volátiles
- Velocidad del reloj de trabajo de 16MHz
- Reducidas dimensiones de 100 x 50 mm

Funcionamiento de Arduino DUE

El cerebro del Arduino DUE es un procesador ARM Cortex-M3 denominado SAM3X8E.

Esta nueva evolución cuenta con dos puertos micro USB, 512 KB de memoria para las aplicaciones, así como salidas y entradas analógicas de 12 bits de precisión, lo que permite obtener unas 1.000 kilomuestras por segundo, comparadas con las 15 kilomuestras por segundo de placas anteriores como Mega 2560 o Leonardo. Además, Arduino DUE (fig 17) es el primer arduino que incluye un conversor digital-analógico por defecto, Arduino DUE es compatible con todas las shields arduino. Sin embargo, más abajo observaremos que esta nueva placa funciona a 3.3 V, mientras que otras placas como la mega, uno o leonardo lo hacían a 5V. En éstos casos, y siempre que la shield no cumpla con las especificaciones R3, es necesario hacer una adaptación de tensiones para evitar quemar placas.

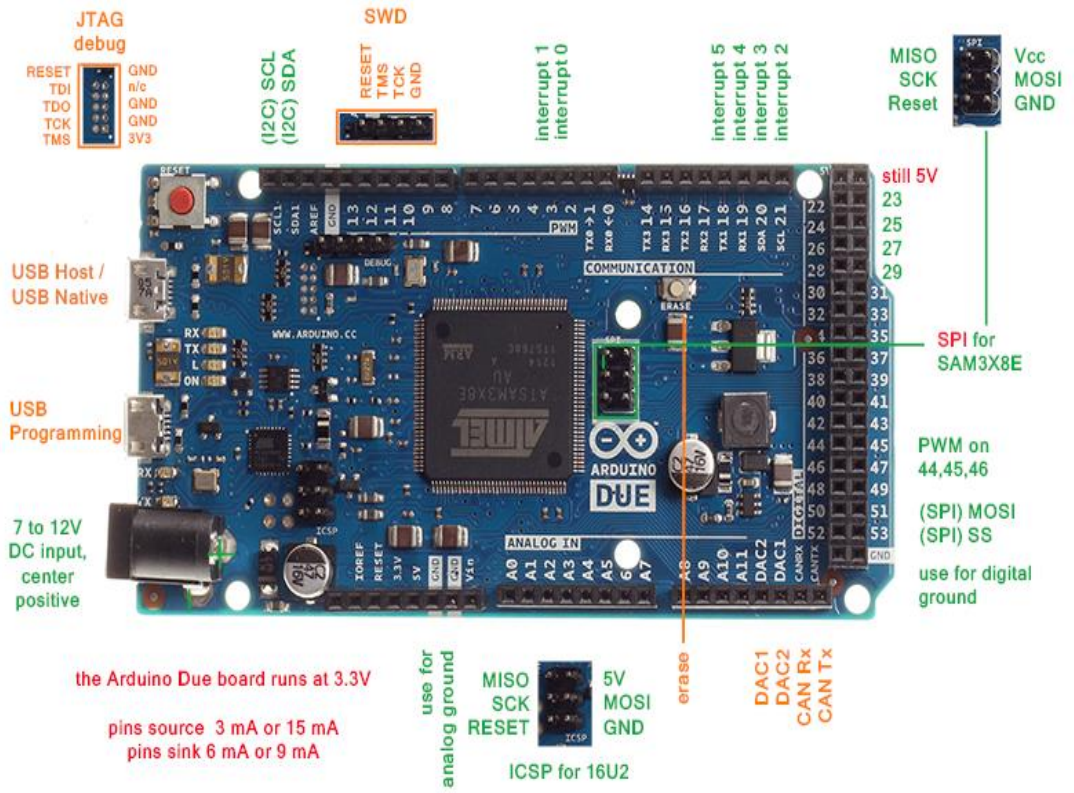


Figura 17 Identificación de Pines Arduino DUE.tomado de : https://www.arduino.cc/mega-descripcion.html

Las especificaciones son las siguiente:

Technical Specification	
Microcontroller:	AT91SAM3X8E
Operating Voltage:	3.3V
Input Voltage (recommended):	7-12V
Input Voltage (min/max):	6-20V
Digital I/O Pins:	54 (of which 6 provide PWM)
Analog Input Pins:	12
Analog Output Pins:	2 (DAC)
Total DC Output Current on all I/O lines:	130 mA
DC Current for 3.3V Pin:	800 mA
DC Current for 5V Pin:	theoretical 1A, recommended 800 mA
Flash Memory:	512 KB
SRAM:	96 KB (64 + 32 KB)
Clock speed:	84 MHz
Debug access:	JTAG/SWD connector

El poseer un procesador ARM de 32 bit a 84 Mhz, permite realizar cosas mucho más rápidamente", comenta el co-fundador de Arduino Massimo Banzi ya que puede leer muchos sensores y procesar los datos con mayor rapides. Tener un procesador más rápido, y el uso del DMA, puede aumentar la estabilidad, capacidad de respuesta y la precisión, sin mermar la capacidad de proceso". El nuevo Arduino Due (fig 17), ofrece dos puertos USB, uno para programación y comunicación, y otro que va a actuar como cliente o como Host, o utilizar un ratón USB o teclado. Esto es algo ventajoso para para desarrollos con esta placa.

Gracias a su nuevo chip Atmel, Arduino Due da un paso de gigante en con los conversores analógicos a digital, permitiendo a diseñadores llevar sus creaciones al límite. "Mucha gente ha construido instrumentos científicos

utilizando Arduino en el pasado, con el DUE, van a disponer de entradas y salidas analógicas de 12-bit" explica Massimo. La tasa teórica de muestreo se ha visto multiplicada hasta unos increíbles 1000ksps (kilomuestras por segundo). En comparación, Arduino Uno, Leonardo y Mega 2560, poseían una tasa de muestro de 15ksps.

Análisis de las de tarjetas de adquisición de datos basadas en tecnologías libres

Luego de haber hecho el estudio y visto la tabla comparativa y funcionamiento de cada una se realiza el análisis de cuál sería la opción más apropiada para implementarla en el dispositivo de adquisición de datos a desarrollar.

La diferencia mas importante entre los cuatro, es que todas las I/O del Arduino, DUE trabajan a 3.3V, mientras que el resto de modelos lo hacen a 5V, un voltaje más común a la hora de encontrar los sensores y actuadores que utilizaremos en nuestro dispositivo de adquisición de datos con interfaz virtual, sin embargo el Arduino DUE tiene una capacidad de memoria y sobre todo, una velocidad de proceso muy superior a sus hermanos "pequeños", indudablemente se puede evidenciar en las características principales de los cuatro Arduino en el Cuadro 1 –Comparativa de Tarjetas de Adquisición de datos.

Con un primer vistazo entre los cuatro modelos, lo primero que se encuentra es la gran diferencia de memoria disponible que hay entre ellos, mientras que el Arduino UNO y Arduino Leonardo tienen 32k, el Arduino Mega 2560 tiene 256k y el Arduino DUE 512k, la memoria es muchas veces más importante que la velocidad, sobre todo cuando se utiliza librerías complejas, por ejemplo una simple demostración que utilice las librerías para

el manejo de pantallas UTFT ocupa más del 80% de la memoria disponible en el Arduino UNO y no es posible de compilar en el Arduino Leonardo (usaría el 104% de la memoria), mientras que en el Arduino Mega 2560 solamente se utiliza el 11% de la memoria, dejándo cerca de un 90% de sus 256K para el código.

En la velocidad se encuentra una gran diferencia entre los tres primeros modelos y el Arduino DUE, este funciona con un reloj 5 veces más rápido que los anteriores y además su procesador es mucho más potente, utiliza un ARM de 32-bit, pero la desventaja es que se encuentra con muchos problemas de incompatibilidad en el código de las librerías que se suelen encontrar en la red, por ejemplo las librerías estándar de los ejemplos de las pantallas LED que utilizan UTFT no se pueden compilar en este modelo sin modificarlas considerablemente.

Otro punto es la cantidad de pines de I/O Digitales disponibles que se pueden utilizar para PWM, con 16 pines PWM disponibles o un Arduino DUE con 12, tanto el Arduino UNO con 6 o el Arduino Leonardo con 7, se nos quedarán cortos si el proyecto es mínimamente complejo, para el desarrollo del dispositivo de adquisición de datos propuesto no es necesario PWM así que no sería una variable a tomar en cuenta.

También hay que tener en cuenta el número de pines de I/O Digitales estándar disponibles, estos se utilizan para conectarse con sensores u otro tipo de periféricos de entrada o salida,. Pero para el Dispositivo de Adquisición de datos con Interfaz Virtual en desarrollo no se necesita disponer de tantas entradas y salidas digitales ya que la interfaz virtual con el computador ahorra teclado y display, pudiendo así usar Arduino Uno o Leonardo que disponen mas de 10 entrada/salida digitales mas que suficiente para.

Cuando se tiene que entrar en contacto con el mundo de los sensores, la mayoría necesita conectarse a pines analógicos de entrada, en este caso

también el Arduino Mega 2560 gana con 16 pines disponibles, le sigue el Arduino Leonardo con 20 y en última posición el Arduino UNO con 6, para el Dispositivo de Adquisición de datos con Interfaz Virtual que realizara la lectura de hasta 5 señales analógicas el Arduino UNO seria el mas recomendable y de bajo costo ya que posee 6 entradas Analógicas.

Tomando en cuenta las distintas características del análisis para el diseño del dispositivo de adquisición de datos con Interfaz Virtual se define como el mas acorde el Arduino UNO para la implementación.

Definición del protocolo de comunicación del dispositivo de adquisición de datos.

Para la definición del protocolo de comunicación con el computador se toma en cuenta las diferentes librerías que usa la tarjeta Arduino UNO de tal manera de seleccionar la mas eficiente para interactuar con el dispositivo de adquisición de datos a implementar donde se debe seleccionar los pines analógicos y digitales a usar y la configuración respectiva que debe tener el código de programación pertinente y la conexión entre las partes se muestra asi (Fig18)

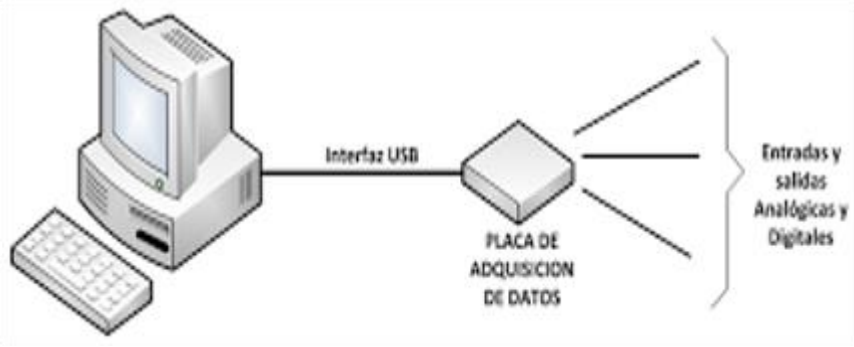


Figura 18 Esquema de comunicación del dispositivo de adquisición de datos. **Elaborado para mostrar como son las etapas y comunicación del dispositivo**

Comunicación por puerto serie: Las funciones de manejo del puerto serie deben ir precedidas de la palabra "Serial" aunque no necesitan ninguna declaración en la cabecera del programa. Por esto se consideran funciones base del lenguaje.25 Estas son las funciones para transmisión serial:

- begin(), available(), read(), flush(), print(), println(), write()

Dentro la comunicación serie tenemos:

UART(recepción-transmisión asíncrona universal) : es uno de los protocolos serie más utilizados. La mayoría de los microcontroladores disponen de hardware UART. Usa una línea de datos simple para transmitir y otra para recibir datos. Comúnmente, 8 bits de datos son transmitidos de la siguiente forma: un bit de inicio, a nivel bajo, 8 bits de datos y un bit de parada a nivel alto. UART se diferencia de SPI y I2C en que es asíncrono y los otros están sincronizados con señal de reloj. La velocidad de datos UART está limitado a 2Mbps

SPI : es otro protocolo serie muy simple. Un maestro envia la señal de reloj, y tras cada pulso de reloj envía un bit al esclavo y recibe un bit de éste. Los nombres de las señales son por tanto SCK para el reloj, MOSI para el Maestro Out Esclavo In, y MISO para Maestro In Esclavo Out. Para controlar más de un esclavo es preciso utilizar SS (selección de esclavo).

I2C : es un protocolo síncrono. I2C usa solo 2 cables, uno para el reloj (SCL) y otro para el dato (SDA). Esto significa que el maestro y el esclavo envían datos por el mismo cable, el cuál es controlado por el maestro, que crea la señal de reloj. I2C no utiliza selección de esclavo, sino direccionamiento.
Dos o más señales a través del mismo cable pueden causar conflicto, y ocurrirían problemas si un dispositivo envía un 1 lógico al mismo tiempo que

otro envía un 0. Por tanto el bus es “cableado” con dos resistencia para poner el bus a nivel alto, y los dispositivos envían niveles bajos. Si quieren enviar un nivel alto simplemente lo comunican al bus.

Comunicaciones serie con Firmata

Firmata es un protocolo de comunicaciones que permite al ordenador (PC o laptop) comunicar con uno o varios Arduinos y controlar sus microprocesadores desde el ordenador. La librería *Firmata* proporciona los protocolos de comunicación serie (métodos) que permitirán comunicar al PC con el Arduino. Usando Firmata podrás controlar servos, motores, pantallas, LEDs, etc desde tu PC a través de uno o varios Arduinos. La tabla siguiente muestra los protocolos o métodos más usuales de *Firmata*:

Cuadro 3 Comandos del protocolo firmata .Elaborado a partir de información en http://firmata.org

	Método	**Descripción**
Común	*begin*	Inicializa la librería *Firmata*
	printVersion	Envía versión del protocolo al PC
	setFirmwareVersion	Establece la versión del firmware
Enviar mensajes	*sendAnalog*	Envía un mensaje analógico
	sendDigitalPortPair	Envía el valor de un pin digital
	sendsysex	Envía un comando con un array de bytes
	sendString	Envía un string al PC
Recibir mensajes	*available*	Comprueba que hay mensajes en el buffer de entrada
	processInput	Procesa los mensajes entrants
	attach	Asocia una función a un cierto tipo de mensaje de entrada
	detach	Disocia una función de un cierto tipo de mensaje de entrada

El protocolo *Firmata* esta evolucionado constantemente; y adaptándose a las nuevas tecnología.

Luego de evaluar los protocolos de comunicación de arduino se define como el mas acorde a la implementación sobre el dispositivo de adquisición de datos y la interfaz virtual para visualizar el comportamiento, medición, generación y análisis de las señales de la implementación en desarrollo, al protocolo serie Firmata, ya que nos permite en tiempo real controlar y modificar el uso de las entradas y salidas digitales, como también las entradas analógicas.

Diseño de una interfaz virtual amigable, para manipular la tarjeta de adquisición de datos a implementar.

Para la elaboración de la interfaz virtual del dispositivo de adquisición de datos se ha utilizado el software de "código abierto" MyOpenLab, versión 3.0.3.3. Se trata de una plataforma de simulación de sistemas y circuitos. El entorno ha sido desarrollado con el lenguaje de programación de alto nivel Java (R) que permite la creación, modificación y utilización de diferentes librerías de componentes para la simulación de múltiples aplicaciones sobre una interface desarrollada para la conexión y la edición entre sus componentes.

Características generales del entorno de desarrollo de MyOpenLab:

•Facilidad de uso

•Amplia biblioteca de funciones tanto para manejo de señales analógicas como digitales.

•Tratamiento de los tipos de datos y operaciones con estos.

•Realización de las aplicaciones mediante el uso de bloques de función.

•Facilidad para crear pantallas de visualización que recojan el estado de las variables y eventos de las simulaciones.

•Posibilidad de ampliación de su librería de componentes.

•Posibilidad de creación de sub-modelos que se pueden encapsular a su vez en otros sub-modelos.

Aplicaciones: Dado su carácter abierto (open source), las aplicaciones se van ampliando constantemente ya que la plataforma deja de crecer y evolucionar en todo momento. En este caso, lo más interesante para la interfaz virtual es la creación de circuitos y sistemas tanto analógicos como digitales. Pero existen más posibles aplicaciones:

Entorno de trabajo : El diseño de las posibles aplicaciones se realiza en dos campos de trabajo diferenciados. Por un lado, el Panel Circuito (fig 19) que se encarga de la elaboración de los diagramas de flujo de programación, el diseño y la interconexión de los componentes creando un circuito bastante visual de cómo están relacionados los elementos que intervienen en la aplicación. Por otro lado, el Panel Frontal (fig 19), se trata de un panel de visualización que representa mediante imágenes interactivas los elementos y componentes que intervienen en dicha aplicación. Es decir, se trata de visualizar la aplicación y poder interactuar con ella teniendo un control de los diferentes estímulos de entrada y de salida

En la parte principal de MyOpenLab se encuentran cuatro áreas diferencias que serán usadas continuamente:

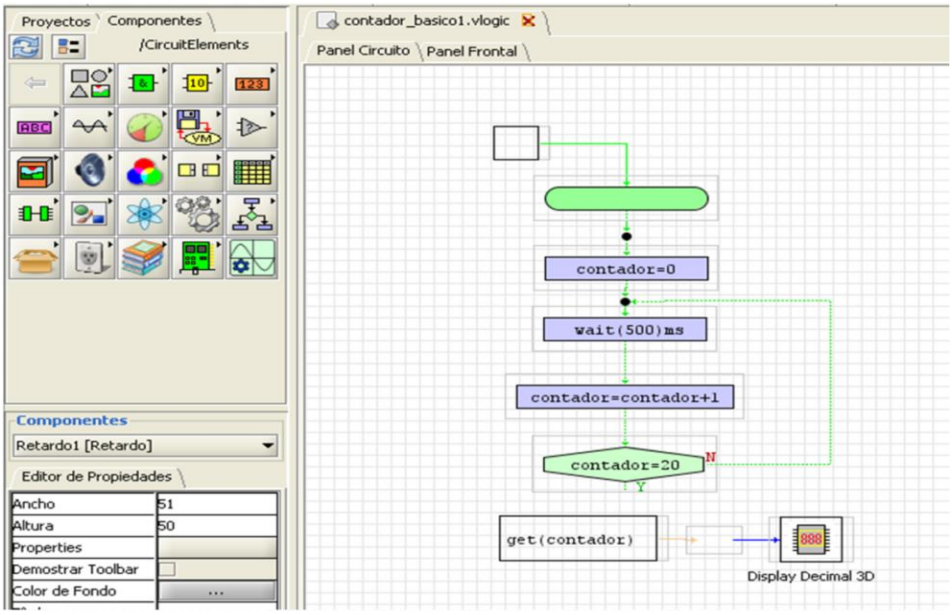


Figura 19 Panel de circuito y panel Frontal de MyOpenLAb. Tomado de : http://www.myopenlab.de/startseite.html

El área de Proyectos (Fig 20), donde se esquematiza los diferentes proyectos con sus sub-proyectos y carpetas. Con ello se facilita la búsqueda y acceso a las aplicaciones realizadas, los componentes y sub-componentes.

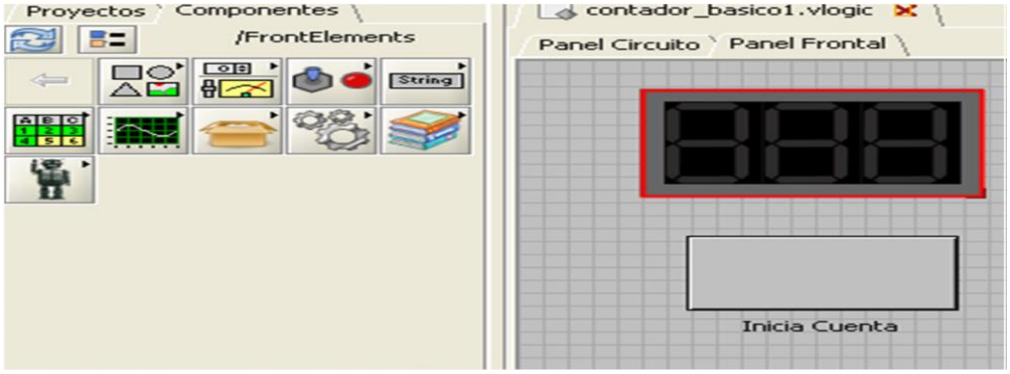


Figura 20 Paneles Generales de MyOpenLab tomado de : http://www.myopenlab.de/startseite.html

El área de Componentes, donde se ubican los elementos y componentes de las diferentes librerías incluidas con el software. Dependiendo del panel que se esté usando en cada momento, se podrán visualizar diferentes componentes para el Panel Circuito o Panel Frontal.

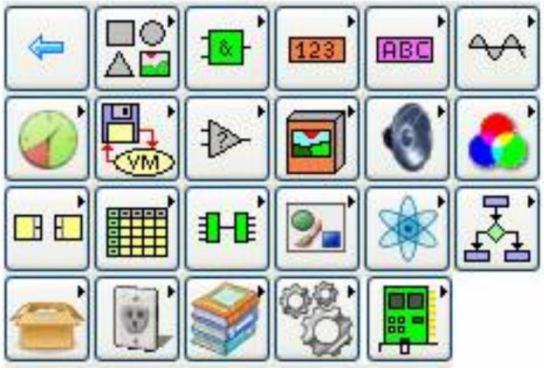


Figura 21 .-Panel de Circuito MyOpeLab tomado de : http://www.myopenlab.de/startseite.html

Para el Panel Circuito, las librerías incluidas son, en este orden :

- Elementos de decoración.
- Operadores digitales.
- Operadores numéricos.
- Tratamiento de caracteres.
- Elementos analógicos.
- Utilidades.
- Ficheros de entrada/salida.
- Comparaciones.
- Tratamiento de imágenes.
- Tratamiento de sonidos.
- Color.
- Pines de entrada/salida.
- Vectores y matrices.
- Agrupación de elementos.
- Objetos gráficos “canvas”.
- Librería de física.
- Librería de diagrama de flujo.
- Librería de extras.
- Librería de conexiones entre aplicaciones.
- Librería definida por el usuario.
- Librería de automatización.
- *Interfaces*.

Para el Panel Frontal, las librerías que se establecen, en siguiente orden, son:

Figura 22 Panel Frontal MyOpenLab. Tomado de: http://www.myopenlab.de/startseite/screenshots.html

- Elementos de decoración.
- Elementos de visualización numérica.
- Elementos de activación digital.
- Elementos de entrada y salida de cadenas de caracteres.
- Vectores y matrices de datos.
- Elementos de visualización gráfica.
- Elementos extras.
- Elementos de automatización.
- Elementos de librería de usuario.
- Robot 2D.

Desarrollo del interfaz virtual para el dispositivo de adquisición de datos:

El desarrollo de la interfaz esta diferenciada en cuatro areas, en primer lugar, es necesaria un área de inicio en el cual se pueda seleccionar el número del puerto de conexión USB para conectar la tarjeta de adquisición de datos de “Arduino” con la interfaz en el ordenador personal. Para ello se verifica previamente de cuál es el puerto indicado para la tarjeta. Además, como también está disponible en un componente “Arduino” de “MyOpenLab”, una entrada que sirva como arranque del interfaz, se utiliza esta área para colocar aquí el botón “Start”. Con todo ello ya sería suficiente para poder

usar el componente virtual sin tener ningún problema, fuera de los que pueda tener el diseño.

Figura 23 Interfaz para selección del Puerto de comunicación. Elaborado con la interfaz grafica de MyOpeLab para el proyecto.

Por otro lado, el desarrollo de los elementos que intervienen propiamente dicho en el dispositivo de adquisición de datos, estos son: El área para entradas, tanto analógicas como digitales.

Para las entradas, analógicas y digitales, se necesita la visualización gráfica de las señales procedentes de sus pines para poder comprender cada señal y todas a la vez. Por ello, se diseña un bloque que contenga una gráfica de tiempos para cada señal. Para las señales de entrada digital se dispone de los seis pines que vienen configurados en la tarjeta "Arduino", y para las entradas analógicas se elaborara solamente dos de las seis entradas. Esto es simplemente por el hecho de visualizar 3 señales digitales inicialmente, sin embargo será útil contar con la visualización de alguna señal analógica como puede ser la señal de alimentación del circuito que se quiera testear y los sensores de temperatura y presencia de luz para la demostración en el área de instrumentación Electronica .

Para el correcto estudio de la señal gráfica contaremos con funciones específicas para las gráficas como es la opción de visualizarlas o no, poder

parar el tiempo para un mejor análisis o refrescar cada señal empezando el tiempo de nuevo sobre el eje X. Ver (Fig 25)

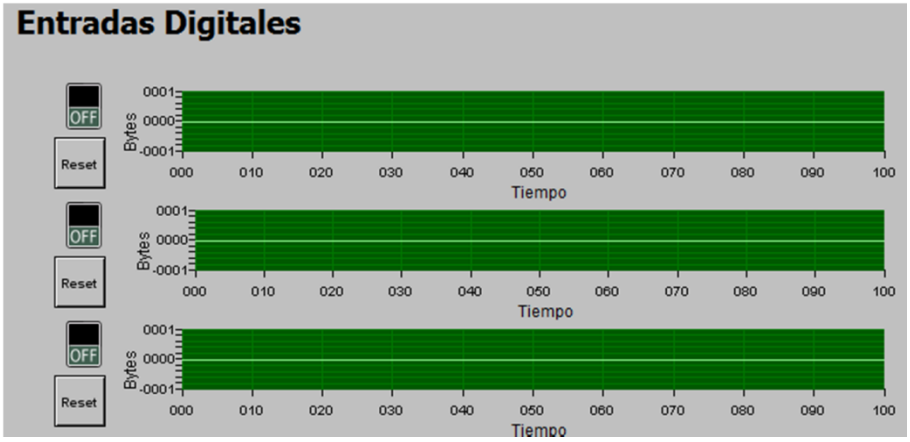


Figura 24 Interfaz Virtual para Señales Digitales. Elaborado para la interfaz Virtual con MyOpenLab

Todo lo anterior, es común para ambos tipos de entradas, analógicas y digitales, sin embargo en el área de entradas analógicas será más útil contar con un visor numérico que nos permita comprobar con mayor exactitud cuál su posición en el tiempo.

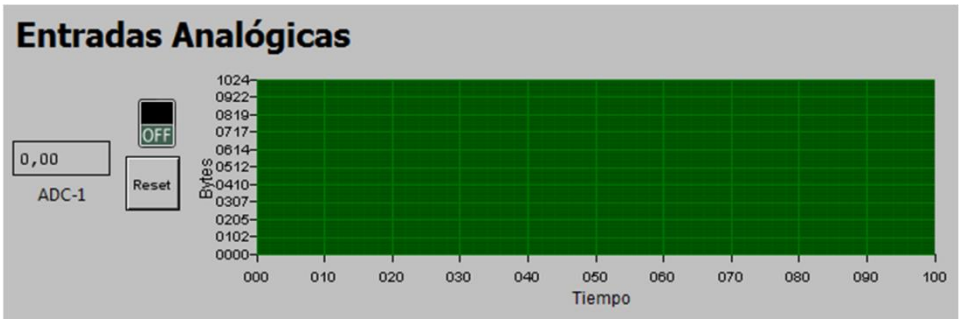


Figura 25 Interfaz Virtual para Señales Analógicas. Elaborado para la interfaz Virtual con MyOpenLab

Por último, el área de salidas digitales permitirá controlar de forma virtual los posibles circuitos lógicos que se analicen. Su visualización gráfica no es de suma importancia ya que contamos con las gráficas para señales de entrada digital, pero si es interesante poder gobernar señales de salida de circuitos lógicos a partir del control virtual de sus entradas. Es decir, con la visualización de las señales digitales podremos decidir cómo gobernar e intervenir en el circuito lógico para poder controlarlo de forma que no sea necesaria el uso de elementos físicos para tal acción y de esta manera realizar la demostración planteada con circuitos en el área de circuitos digitales. Una vez comprobado el lugar asignado para las entradas y salidas digitales, con la tarjeta de datos. Abriendo el programa “MyOpenLab” y a la vez se crea el modelo lógico que le da la funcionalidad a la parte gráfica del panel con los datos adquiridos por el hardware.

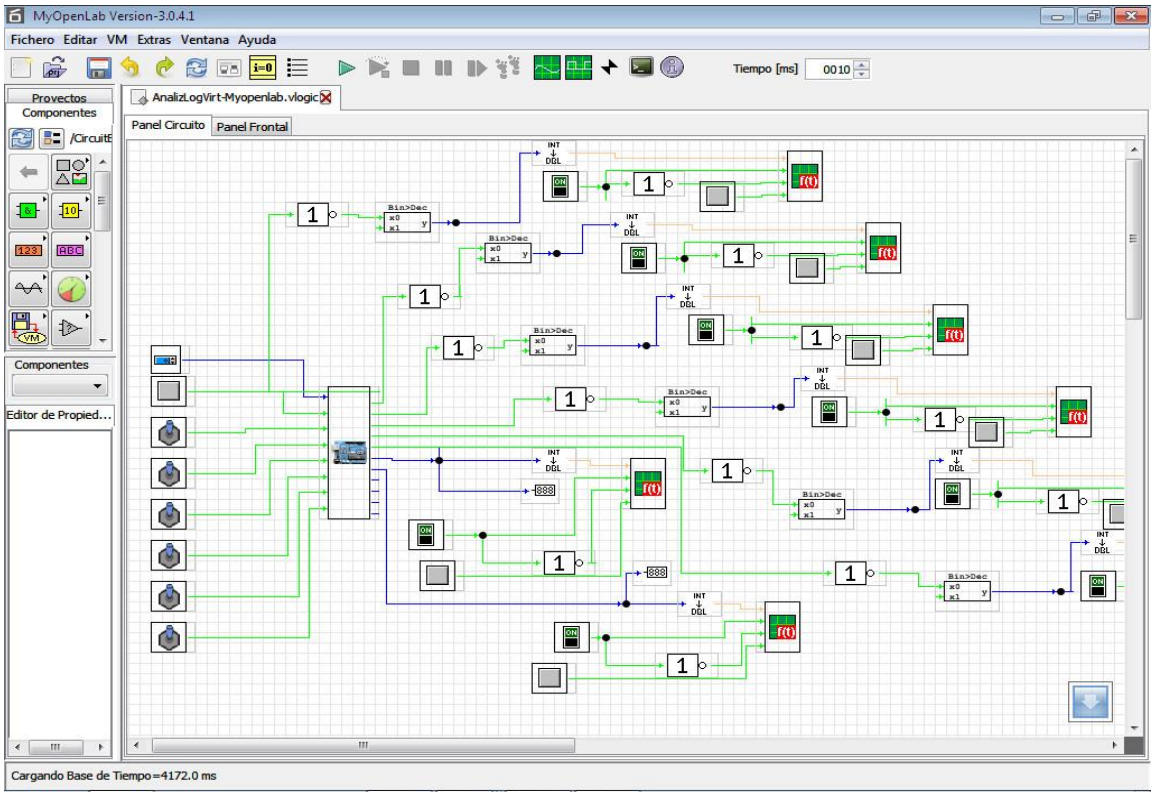
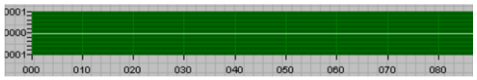


Figura 26 Bloque Lógico de la interfaz Virtual Diseñada. Elaborado para la interfaz Virtual con MyOpenLab

Con el entendimiento claro del funcionamiento de ambas herramientas, se podrá diseña la interfaz prevista y se pueden realizar cambios a conveniencia Para ello se necesitan diferentes elementos o componentes gráficos con los que elaborar nuestro interfaz virtual en el Panel Frontal de “MyOpenLab”.
Elementos como gráficos, botones, interruptores, paneles numéricos, elementos de señalización etc. se irán colocando en el panel de forma clara y ordenada dando forma a lo que queremos que sea nuestra pantalla del analizador.



Una vez estructurada la visión del interfaz se realizan las conexiones necesarias entre estos elementos interactivos y variables, donde el resultado del bloque lógico y el bloque visual permite definir lo que el usuario del dispositivo de adquisición de datos va a manipular y monitorear mediante un computador conectado a la tarjeta de adquisición de datos. ver (fig 28).

Figura 27 Interfaz Virtual del Dispositivo de Adquisición de datos. Elaborado para la interfaz Virtual con MyOpenLab

Construccion del sistema integrado con los componentes definidos, con el fin de realizar una demostración en el área de Instrumentación (lectura de sensores) y Sistemas Digitales.(manipular compuertas lógicas).

Para procesar los datos se requiere que la placa Arduino conozca magnitudes del mundo real, como pueden ser de luz, temperatura, etc y para

ello la placa cuentan con entradas analógicas con las que, a través, de sensores, medimos dichas magnitudes y también cuenta con entradas digitales que manejan valores de 0 (OFF) o 5 (ON).

Cuando se plantea que un ordenador o microcontrolador en este caso, es un sistema digital y las magnitudes que deseamos medir son analógicas, por ello necesitamos un sistema que pase de analógico a digital, el cual llamaremos ADC (Analog digital coverter), como es lógico Arduino cuenta con un ADC.

Este conversor ADC va conectado a un multiplexor, para que así, con uno solo tener varías entradas, como es en el caso de las placas Arduino que normalmente cuenta con 6 de estas.

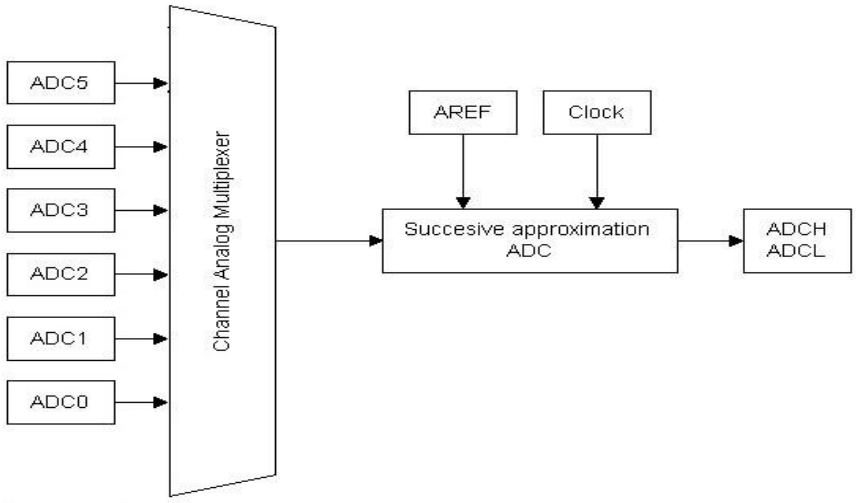


Figura 28 Bloque de ADC en Arduino. Tomado de : https://en.wikipedia.org/wiki/Successive_approximation_ADC

El tiempo de muestreo, es el tiempo que transcurres entre dos mediciones consecutivas, es fundamental para la adquisición de datos y se suele expresar en frecuencia. Siempre que se mida un sistema como puede ser el sonido, se requiere que la frecuencia de muestro sea superior a la

frecuencia del sistema, de no ser así, la representación que se obtiene con el muestreo, del sistema, no será correcta.

El tiempo de muestreo según sus características: es de 112 micro segundos, lo que equivale a una frecuencia de muestreo de 8.928 kHz. Otro factor a tener en cuenta del ADC es su resolución, para ello se habla de los bits de resolución que tiene, los bits permiten hacer combinaciones, cuantos mas tenga mayor numero de combinaciones pueden hacerse, cada una de estas combinaciones muestra una medida del sistema analógico, la cantidad de medidas que tenemos serán 2 elevado al numero de bits de resolución. Puesto que Arduino UNO, tiene un ADC de 10 bits de resolución tendrá 1024 combinaciones.

Pero hay un segundo factor que interviene en la resolución, es el rango de medida, El rango de Voltaje de las placas Arduino por defecto es de 5v o 3.3v. Con esto se deduce que en voltios hay una resolución igual a;

$$RESOLUCIÓN = \frac{5V}{1024} = 4.88\,mV$$

En ocasiones,se utilizan sensores cuyo rango de voltaje no será igual a [0, 5v]. En primer lugar, se tiene en cuenta, que no debe introducirse en la placa Arduino voltajes mayores de 5v o 3.3v, partiendo de esto se cambia el límite superior del rango de voltaje. De manera interna o externa.

La manera interna de cambiarlo, es utilizando software únicamente, para ello la función analogReference(). Esta función permite cambiar el límite superior del rango de voltaje, en la placa Arduino UNO por ejemplo, al introducir como parámetro INTERNAL el límite de voltaje superior será de 1.1v. En la placa Arduino UNO introduciendo como parámetro INTERNAL1V1 y INTERNAL2V56 siendo el nuevo límite 1.1v y 2.56v respectivamente.

Conexión de entradas analógicas en Arduino

Se dispone de un sensor analógico que proporciona una señal analógica entre 0V a 5V. El esquema de conexión es el siguiente:

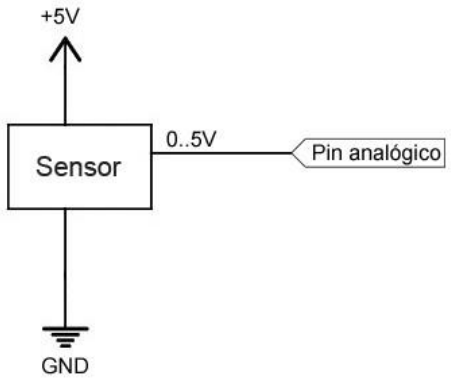


Figura 29 Esquema de pin Analogico

El código para realizar la lectura es mediante AnalogRead() y almacenamos el valor devuelto.

```
1.  const int sensorPin = A0;     // seleccionar la entrada para el sensor
2.  int sensorValue; // variable que almacena el valor raw (0 a 1023)
3.  void setup() {
4.  }
5.  void loop() {
6.  sensorValue = analogRead(sensorPin);    // realizar la lectura
7.  //mandar mensaje a puerto serie en función del valor leido
8.  if (sensorValue > 512) {
9.  Serial.println("Mayor que 2,5V");
10. }
11. else {
12. Serial.println("Menor que 2,5V");
13. }
14. delay(1000);
```

15. }

Lectura de valores mayores de 5V

En caso de necesitar leer una entrada de nivel de tensión superior, por ejemplo de 12V, debemos realizar una adaptación de tensión. La forma mejor de realizar la adaptación es emplear un simple divisor de tensión.

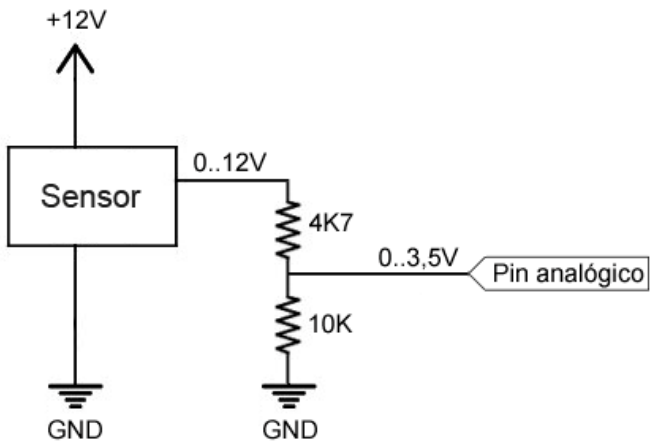


Figura 30 Esquema de Adaptación de Entrada a pin analógica. Elaborado para mostrar la adaptación

Con esta configuración el pin digital de Arduino recibe una tensión que varía entre 0 a 3,5V, como se hizo referencia, se estara perdiendo precisión relativa. Una opción sera ajustar las resistencias para que los límites estén lo más cercanos posible a 0 y 5V, o usar otro divisor de tensión para alimentar el pin Aref.

Los valores de las resistencias a emplear dependen del voltaje a tomar lectura, y de la impedancia del sensor. En general, deben cumplir las siguientes condiciones:

- Deben convertir la señal en un rango inferior pero similar a la tensión de alimentación.
- Deben ser muy superiores a la impedancia equivalente del dispositivo a medir.
- Deben despreciables respecto a la impedancia de la entrada Arduino.
- Deben limitar la corriente que circula por ellas para minimizar pérdidas.
- Deben ser capaces de disipar la potencia que van a soportar.

Calculo del Circuito Divisor Tensión para adquirir datos en prácticas de Laboratorio:

Las entradas analógicas de un Arduino pueden medir hasta 5V (cuando se utiliza la tensión de referencia analógica incorporada). Incluso cuando sólo se conecta a un circuito de 5V, debe utilizar las resistencias para ayudar a proteger el Arduino de cortocircuitos o sobrecargas de tensión inesperadas.

Se trata de un circuito divisor de tensión que consta de dos resistencias (R1 y R2) en serie que, se encargan de dividir el voltaje de entrada, para adaptarlo a la ventana de tensiones que pueden leer las entradas analógicas del Arduino (5V).

El divisor entrega una tensión al pin analógico de Arduino que éste convierte en un formato digital que puede ser procesada por el microcontrolador. En

este caso, la tensión entrada después de pasar por el divisor de tensión descrito (R1 y R2), se aplica al pin A0 .

El circuito con los valores mostrados para R1 de 1MΩ en serie con R2 de 100KΩ representa una impedancia de entrada de 1MΩ + 100kΩ = 1’1M, que reduciendo es = 11, factor de división que es adecuado para la medición de voltajes de DC hasta aproximadamente 55V.

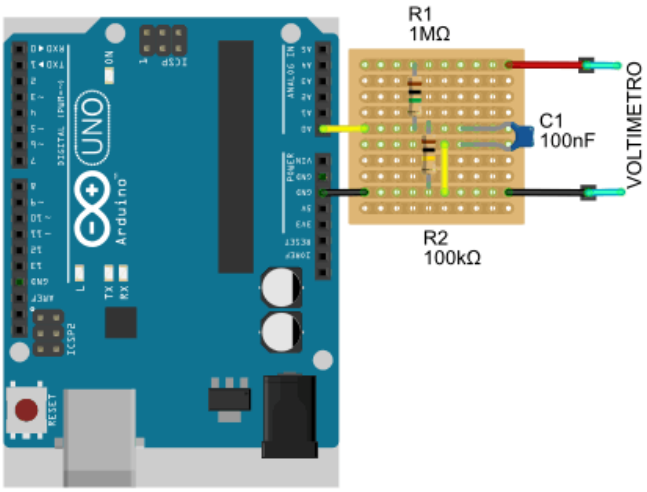


Figura 31 Diagrama de circuito adaptador a entrada de arduino. Elaborado con http://www.fritzing.org

El circuito mostrado divide el voltaje de entrada conectado al pin análogo Arduino, equivalente a la tensión de entrada dividido por 11, por lo tanto el máximo es de 55V ÷ 11 = 5V, por seguridad se da un margen de medición de voltaje de 0-30V DC.

Principios de funcionamiento del circuito divisor de tensión:

Si un voltímetro tiene una baja impedancia de entrada baja, digamos 10kΩ y se está midiendo un voltaje a extremos de una resistencia de 10kΩ, el

multímetro está cambiando efectivamente el valor de la resistencia a 5KΩ (ya que dos resistencias de 10kΩ en paralelo = resistencia 5kΩ). Por consiguiente, el voltímetro ha cambiado los parámetros del circuito, y realmente, está leyendo una tensión errónea. Esas dos resistencias forman un divisor de potencial que se utiliza para bajar el voltaje que se mide a un nivel que Arduino pueda leer, además de ser la impedancia de entrada.

Para medir voltaje DC típicamente tendrá una impedancia de entrada de 10MΩ o superior. Esto significa que la resistencia entre las dos sondas o terminales del multímetro es de 10MΩ o más.

Así pues, es deseable una alta impedancia de entrada para un voltímetro (o multímetro en la escala de voltaje). Cuanto mayor impedancia de entrada, menos probable es que el multímetro influya o cambie lo que esta midiendo del circuito. Al medir (con un multímetro que tiene una impedancia de entrada de 10 millones de ohmios) el voltaje a extremos de un componente en un circuito, es el mismo que la conexión de una resistencia de 10MΩ en paralelo con el circuito; no influye.

Así que, con una alta impedancia de entrada el circuito divisor de tensión, y la impedancia de este “voltímetro” no influye en el circuito que se esté bajo prueba.

La fórmula para calcular los valores en un divisor de tensión es:

$$Vout = (R2 / (R1 + R2)) * Vin$$

Por lo tanto, puesto que Arduino admite un Vmax de 5V en sus entradas analógicas, si el divisor está funcionando correctamente, entonces el Vout será de un máximo de 5V, y para poder calcular la tensión máxima de entrada al circuito usaremos:

Vmax = 5,0 / (R2 / (R1 + R2))

Codigo Arduino para adquirir mediciones de Voltaje:

```
// Tesis de grado IUPSM Leopoldo Marcano
// Adquirir Volatjes DC con Arduino UNO
// Utiliza el monitor Serial para mostrar los valores.
float v1 = 4.98; // valor real de la alimentacion de Arduino, Vcc
float r1 = 1000000; // 1M
float r2 = 100000; // 100K
void setup() {
Serial.begin(9600);
Serial.println("---------------------");
Serial.println("aquirir datos");
Serial.print("Maximo Voltage: ");
Serial.print((int)(v1 / (r2 / (r1 + r2))));
Serial.println("V");
Serial.println("---------------------");
Serial.println("");
delay(2000);
}
void loop() {
float v = (analogRead(0) * v1) / 1024.0;
float v2 = v / (r2 / (r1 + r2));
Serial.print("V: ");
Serial.println(v2);
delay(10);
}
```

Cómo funciona el Código.

Para medir la tensión en la rutina *loop()*, se utiliza la función analógica *analogRead(0)*, para leer la entrada analógica 0 en este caso. El valor que

nos devuelve, es un entero dentro del rango de 0 a 1023, por dicho motivo lo se ajusta en la ventana de 0 a 5 que, es el margen de Arduino. Dicho valor leído se multiplica por el nivel real de alimentación y divide por 1024.

Transmisión de la información adquirida por el sistema electrónico a través del puerto serie USB en computadora

Arduino posee como principal característica la habilidad de comunicarse con nuestra computadora a través del puerto serie. Esto se conoce como comunicación serial. Debido a que el uso de este puerto ha quedado un poco en desuso a favor de la tecnología USB, Arduino cuenta con un convertidor de Serial a USB que permite a nuestra placa ser reconocida por nuestra computadora como un dispositivo conectado a un puerto COM aún cuando la conexión física sea mediante USB.

La forma más simple que existe para establecer la comunicación serial con Arduino se expresa en el siguiente código ;.

```
1
2 void setup(){
3   Serial.begin(9600);
4 }
5  void loop(){
6   Serial.println('1');
7   delay(1000);
8 }
```

En la función setup inicializa la comunicación serial con la sentencia Serial.begin(9600).

El 9600 indica el baud rate, o la cantidad de baudios que manejará el puerto serie. Se define baudio como una unidad de medida, usada en telecomunicaciones, que representa el número de símbolos por segundo en un medio de transmisión ya sea analógico o digital.

Siempre que se necesita comunicación con Arduino vía puerto serie se necesita invocar la sentencia Serial.begin(9600).

Comandos de comunicación serial presentes en el desarrollo :

Serial.begin(rate): Abre el puerto serie y fija la velocidad en baudios para la transmisión de datos en serie. El valor típico de velocidad para comunicarse con el ordenador es 9600, aunque otras velocidades pueden ser soportadas.

```
void setup()

{
Serial.begin(9600);    // abre el Puerto serie
}    // configurando la velocidad en 9600 bps
```

Nota: Cuando se utiliza la comunicación serie los pines digitales 0 (RX) y 1 (TX) no pueden utilizarse para otros propósitos.

Serial.println(data):Imprime los datos en el puerto serie, seguido por un retorno de carro y salto de línea. Este comando toma la misma forma que Serial.print (), pero es más fácil para la lectura de los datos en el Monitor Serie del software.

```
Serial.println(analogValue);    // envía el valor 'analogValue'
al puerto
```

El siguiente ejemplo toma de una lectura analógica del *pin 0* y envía estos datos al ordenador cada segundo.

```
void setup()
{Serial.begin(9600);    // configura el puerto serie a 9600bps}
void loop(){
Serial.println(analogRead(0)); // envía valor analógico
delay(1000);     // espera 1 segundo
}
```

Demostración con el dispositivo en el área de Sistemas Digitales:

En un primer lugar, para comprobar el funcionamiento correcto de las entradas digitales usaremos un integrado de puertas lógicas AND (modelo 74ls08), de tal forma que al manipular sus dos entradas se modifique su salida y por tanto su visualización gráfica observarla en la interfaz virtual del dispositivo de adquisición de datos.

En primer lugar hay que alimentar el circuito integrado de las puertas lógicas situado en una placa protoboard. Existen dos modos, mediante una fuente externa o directamente a través de la tarjeta “Arduino” conectada a su vez al ordenador. Puede conectarse a 3,3 o 5V, este caso 5V (considerar características eléctricas) serán cableados a la patilla 14 del integrado y a su vez en este caso la tierra que ofrece también “Arduino” a la patilla 7

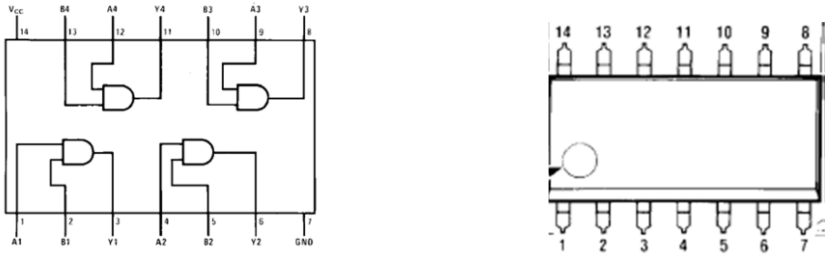


Figura 32 Esquema del integrado 74ls08 con compuertas lógicas.Tomado de: https://es.wikibooks.org/wiki/Circuito_integrado_7408

Una vez conectado, se cablean las entradas y salidas del integrado. Por un lado las dos entradas del integrado (por ejemplo pin 13 y 12) irán conectadas

a dos de los pines asignados para las salidas digitales que interactuaran mediante los interruptores. Por otro lado, la salida de la puerta AND (en este caso, pin 11) irá conectada a una de las entradas asignadas digitales para su visualización.

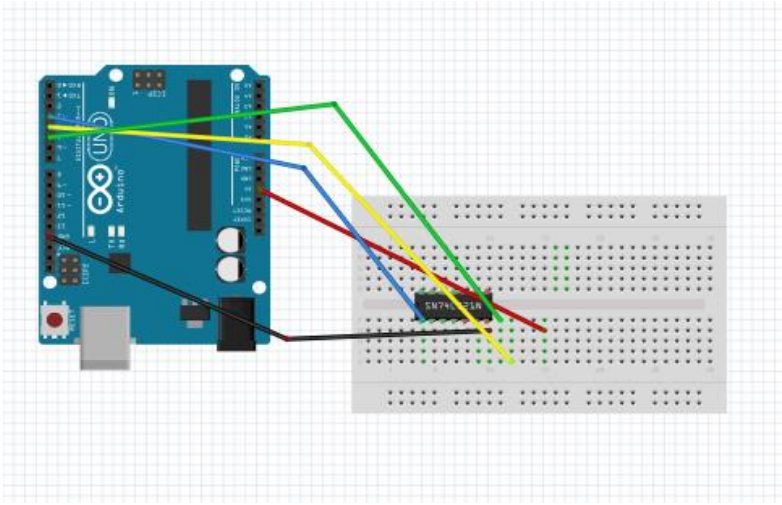


Figura 33 Conexión para demostración puertas logicas AND con el dispositivo de adquisición de datos. . Elaborado con http://www.fritzing.org

Una vez conectado se activará la gráfica correspondiente a la señal digital asignada al pin de la salida de la puerta lógica. Y con los interruptores de las salidas digitales apagados, el resultado gráfico será una línea recta de valor 0. Si y solo si están activo ambos interruptores de las salidas digitales correspondientes a las entradas de la puerta AND, se muestra un pulso o flanco ascendente de valor y la representación de la función lógica AND es la siguiente:

Cuadro 4. Tabla de la verdad del 74ls08. Elaborado a partir del resultado con el dispositivo portátil

Y = AB

Inputs		Output
A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

H = HIGH Logic Level
L = LOW Logic Level

Demostración con el dispositivo en el área de Instrumentación:

En la instrumentación la mayoría de sensores transforman las variables física de entorno en una señal analógica, para ello se plantea un montaje de un divisor de tensión de dos resistencias, una de ellas una LDR (variable con la luz). A continuación el circuito propuesto:

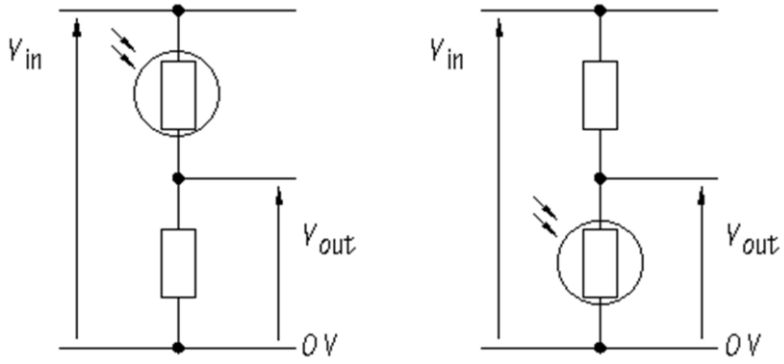


Figura 34 Esquema del circuito con Sensor de luz. Elaborado para mostrar la conexión de la fotocelda

Obsérvese el simple cambio de posición de la LDR ya que afectará a su visualización gráfica. Es decir, en el primer caso, si la luz incide sobre ella, la diferencia de potencial aumentará en la salida y en el otro caso, si la luz incide sobre ella la diferencia de potencial en la salida será menor ya que la mayoría de la corriente circulará a tierra a través de la LDR.

Para el montaje del primer caso con la tarjeta “Arduino”, conectaremos el pin de 5V de la tarjeta a la patilla de la LDR que marca la tensión de entrada y la tierra GND a la patilla de la segunda resistencia que marca 0V. Por otro lado, entre una y otra resistencia conectaremos a una de las gráficas de las entradas analógicas para visualizar la variación de datos que permite según la cantidad de luz.

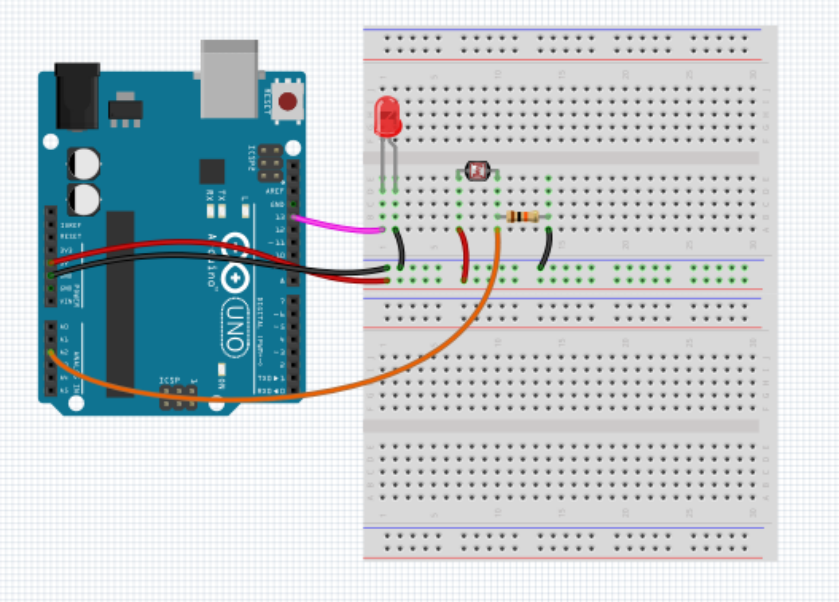


Figura 35 Conexión para demostración con sensor de luz con el dispositivo de adquisición de datos. Elaborado con http://www.fritzing.org

Una vez conectado, se inicia el interfaz y el marcador numérico de datos comenzará oscilar, en este caso sobre los 600 bytes. Si interrumpimos el paso de luz, la cantidad de datos disminuirá hasta los 200bytes

aproximadamente. La cantidad de datos no es fija ya que la luz va variando constantemente.

En el montaje contrario ocurrirá lo opuesto, evidentemente. Simplemente cambiando de posición las conexiones entre alimentación y tierra, conseguiremos invertir la cantidad de datos que se muestre en el gráfico en función de la cantidad de luz.

CONCLUSIONES

Durante la realización del “Dispositivo portátil de adquisición de datos con interfaz virtual basado en tecnologías libres y a partir de cada problema que surgía, al que se le daba solución buscando diferentes alternativas para su funcionamiento, se obtuvieron los siguientes resultados:

Se diseñó un prototipo, que permite la medición, generación y análisis de señales en una práctica teniendo como limitantes rangos de voltajes, amperaje y frecuencia de muestreo con lo cual se determinó que su utilidad mayormente seria a circuitos tanto digitales como en áreas de instrumentación electrónica.

En la selección de la tarjeta de adquisición de datos se evidencio el poder que presenta estas tarjetas Arduino y las grandes posibilidades de uso en todas las áreas de la computación física, automatización y control pudiendo ampliar sus bondades con esquemas de adaptación de señales como el desarrollado y presentar un dispositivo con funciones específicas.

Con el desarrollo de la interfaz virtual se evidencio el poder que tiene el desarrollo de código embebido a través de la plataforma MyOpenLab comparable con su par de software propietario Labview, ya que con el protocolo firmata que permite a MyOpenLab comunicarse directamente con el microcontrolador presente en Arduino y gobernar a voluntad las entrada y salidas tanto digitales como analógicas de una forma transparente y utilizando una lógica grafica de su interfaz de desarrollo.

Se utilizó Tecnologías Libres , que permite que el docente o estudiante se apodere del proyecto para lograr mejoras o adaptaciones..

Se diseño pensando en poder ser útil en todas las asignaturas que el estudiante recorre por su formación, tomando en cuenta que el mismo puede y debe ir mejorando según los objetivos pedagógicos que se persigan.

Después de tener el Proyecto en fase de desarrollo y habiendo ya subsanando los problemas de la manera mas factible que, además buscando siempre una aplicación a nuestra problemática actual en la formación. Concluimos además que pudiera ser una herramienta complementaria para capacitaciones E-learning en esta nueva era digital.

RECOMENDACIONES

Existe aspectos importantes en el desarrollo de este dispositivo que pueden mejorarse Debido a que con un dispositivo mas versátil con mayor capacidad de muestreo y ampliando la interfaz gráfica virtual con módulos específicos para asignaturas puntuales se podría obtener un producto beneficioso, útil y accesible que además de utilizar una tarjeta como arduino uno como el corazón principal del hardware, pudiendo mejorarse y escalar su funcionalidad con shield o tarjeta adaptadores mas complejas y de mayor rango de medidas.

Gracias a que esta enmarcado en tecnologías libres su desarrollo pudiera ampliarse con facilidad sin requerir pagar licenciamiento de uso por software de desarrollo en nuevas interfaces virtuales como ocurre con entornos similares como Labview donde se necesita pagar por su de manera legal.

Debido a que cada dia los teléfonos tactiles, tabletas y dispositivos móviles son mas frecuente en nuestro entorno, pudiera enfocarse la portabilidad de la interfaz virtual a estos entornos ya que la base del desarrollo esta enmarcado en un lenguaje multiplataforma como lo es Java, lenguaje por el cual la plataforma MyOpenLab genera sus intefaces virtuales.

REFERENCIAS BIBLOGRAFICAS

Oscar Torrente Artero, Arduino Curso Practico de Formación, Mexico, Alfa Omega Grupo Editorial, 2013

MyOpenLab, (2014), [Programa de Computacion en Linea], Disponible en : http://myopenlab.de/startseite.html

Arduino, Open Hardware (2016) [Programa de Computación, Esquemas y firmware en Linea]. Disponible en : https://www.arduino.cc/en/Main/Software

José Manuel Ruiz Gutiérrez, (s.f). Guia de Usuario MyOpenLab [Documento en linea] disponible : http://sourceforge.net/projects/myopenlab3/files/Distribution%203.0.3.3/Guia_usuario_MyOpenLab_3010.pdf/download

Fritzing . open-source hardware initiative (2016) [Programa de Computacion en Linea], Disponible en : http://fritzing.org/

A. Manuel, D. Biel, J. Olivé, J. Prat, Instrumentación Virtual: Adquisición, procesado y análisis de señales. Barcelona: Ediciones UPC, 2001.

Richard M. Stallman. (2004). Software libre para una sociedad libre. Madrid. Traficantes de Sueños.

Hurtado, J. (2007). El Proyecto de Investigación. Metodología de la Investigación Holística. Caracas: Quirón.

Instituto Universitario Politécnico “Santiago Mariño”. (2006). Manual de trabajo especial de grado. Caracas.

Virgilio Rosendo Pérez Pérez (2011) Sistema Domótica y de Entretenimiento: Hardware Libre y Software de Código Abierto

Jordi Girona Salgado (2005) Laboratorio de Electrónica: Curso Básico Barcelona – España Ediciones de La Universidad deCataluña.

.CREUS SOLE, Antonio. Instrumentación Industrial. México: Alfaomega-Marcombo, 1992. 4 ed

W. Cooper. Editorial (1982)'Instrumentación Electrónica Moderna y Técnicas de Medición' de Prentice Hall

ANEXOS

ANEXO A

CODIGO FUENTE DEL FIRMWARE PARA EL MICROCONTROLADOR ATMEGA 328P

arduino/Arduino
Code
Issues 622
Pull requests 105
Wiki
Pulse
Graphs
Arduino/hardware/arduino/avr/bootloaders/atmega/ATmegaBOOT_168.c
94af627 on 14 Jul 2015
@iMartyn iMartyn Bootloaders: wrong #ifdefs, should be defined() not just tested (i.e.…
3 contributors
@iMartyn
@ffissore
@cmaglie
1058 lines (916 sloc) 29.5 KB

```
/*************************************************************/
/* Serial Bootloader for Atmel megaAVR Controllers        */
/*                                                           */
/* tested with ATmega8, ATmega128 and ATmega168           */
/* should work with other mega's, see code for details    */
/*                                                           */
/* ATmegaBOOT.c                                           */
/*                                                           */
/*                                                           */
/* 20090308: integrated Mega changes into main bootloader */
/*           source by D. Mellis                          */
/* 20080930: hacked for Arduino Mega (with the 1280       */
/*           processor, backwards compatible)             */
/*           by D. Cuartielles                            */
/* 20070626: hacked for Arduino Diecimila (which auto-    */
/*           resets when a USB connection is made to it)  */
/*           by D. Mellis                                 */
/* 20060802: hacked for Arduino by D. Cuartielles         */
/*           based on a previous hack by D. Mellis        */
/*           and D. Cuartielles                           */
/*                                                           */
/* Monitor and debug functions were added to the original */
/* code by Dr. Erik Lins, chip45.com. (See below)         */
/*                                                           */
/* Thanks to Karl Pitrich for fixing a bootloader pin     */
/* problem and more informative LED blinking!             */
/*                                                           */
/* For the latest version see:                            */
/* http://www.chip45.com/                                 */
/*                                                           */
/* ------------------------------------------------------ */
/*                                                           */
/* based on stk500boot.c                                  */
/* Copyright (c) 2003, Jason P. Kyle                      */
/* All rights reserved.                                   */
/* see avr1.org for original file and information         */
/*                                                           */
/* This program is free software; you can redistribute it */
```

```
/* and/or modify it under the terms of the GNU General    */
/* Public License as published by the Free Software       */
/* Foundation; either version 2 of the License, or        */
/* (at your option) any later version.                    */
/*                                                        */
/* This program is distributed in the hope that it will   */
/* be useful, but WITHOUT ANY WARRANTY; without even the  */
/* implied warranty of MERCHANTABILITY or FITNESS FOR A   */
/* PARTICULAR PURPOSE.  See the GNU General Public        */
/* License for more details.                              */
/*                                                        */
/* You should have received a copy of the GNU General     */
/* Public License along with this program; if not, write  */
/* to the Free Software Foundation, Inc.,                 */
/* 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA */
/*                                                        */
/* Licence can be viewed at                               */
/* http://www.fsf.org/licenses/gpl.txt                    */
/*                                                        */
/* Target = Atmel AVR m128,m64,m32,m16,m8,m162,m163,m169, */
/* m8515,m8535. ATmega161 has a very small boot block so  */
/* isn't supported.                                       */
/*                                                        */
/* Tested with m168                                       */
/**********************************************************/


/* some includes */
#include <inttypes.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>
#include <avr/wdt.h>
#include <util/delay.h>

/* the current avr-libc eeprom functions do not support the ATmega168 */
/* own eeprom write/read functions are used instead */
#if !defined(__AVR_ATmega168__) || !defined(__AVR_ATmega328P__) || !defined(__AVR_ATmega328__)
#include <avr/eeprom.h>
#endif

/* Use the F_CPU defined in Makefile */

/* 20060803: hacked by DojoCorp */
/* 20070626: hacked by David A. Mellis to decrease waiting time for auto-reset */
/* set the waiting time for the bootloader */
/* get this from the Makefile instead */
/* #define MAX_TIME_COUNT (F_CPU>>4) */

/* 20070707: hacked by David A. Mellis - after this many errors give up and launch application */
#define MAX_ERROR_COUNT 5

/* set the UART baud rate */
/* 20060803: hacked by DojoCorp */
//#define BAUD_RATE   115200
#ifndef BAUD_RATE
#define BAUD_RATE   19200
#endif


/* SW_MAJOR and MINOR needs to be updated from time to time to avoid warning message from AVR Studio */
/* never allow AVR Studio to do an update !!!! */
#define HW_VER  0x02
#define SW_MAJOR 0x01
#define SW_MINOR 0x10
```

```
/* Adjust to suit whatever pin your hardware uses to enter the bootloader */
/* ATmega128 has two UARTS so two pins are used to enter bootloader and select UART */
/* ATmega1280 has four UARTS, but for Arduino Mega, we will only use RXD0 to get code */
/* BL0... means UART0, BL1... means UART1 */
#ifdef __AVR_ATmega128__
#define BL_DDR  DDRF
#define BL_PORT PORTF
#define BL_PIN  PINF
#define BL0     PINF7
#define BL1     PINF6
#elif defined __AVR_ATmega1280__
/* we just don't do anything for the MEGA and enter bootloader on reset anyway*/
#else
/* other ATmegas have only one UART, so only one pin is defined to enter bootloader */
#define BL_DDR  DDRD
#define BL_PORT PORTD
#define BL_PIN  PIND
#define BL      PIND6
#endif


/* onboard LED is used to indicate, that the bootloader was entered (3x flashing) */
/* if monitor functions are included, LED goes on after monitor was entered */
#if defined __AVR_ATmega128__ || defined __AVR_ATmega1280__
/* Onboard LED is connected to pin PB7 (e.g. Crumb128, PROBOmega128, Savvy128, Arduino Mega) */
#define LED_DDR  DDRB
#define LED_PORT PORTB
#define LED_PIN  PINB
#define LED      PINB7
#else
/* Onboard LED is connected to pin PB5 in Arduino NG, Diecimila, and Duomilanuove */
/* other boards like e.g. Crumb8, Crumb168 are using PB2 */
#define LED_DDR  DDRB
#define LED_PORT PORTB
#define LED_PIN  PINB
#define LED      PINB5
#endif


/* monitor functions will only be compiled when using ATmega128, due to bootblock size constraints */
#if defined(__AVR_ATmega128__) || defined(__AVR_ATmega1280__)
#define MONITOR 1
#endif


/* define various device id's */
/* manufacturer byte is always the same */
#define SIG1        0x1E // Yep, Atmel is the only manufacturer of AVR micros.  Single source :(

#if defined __AVR_ATmega1280__
#define SIG2        0x97
#define SIG3        0x03
#define PAGE_SIZE   0x80U       //128 words

#elif defined __AVR_ATmega1281__
#define SIG2        0x97
#define SIG3        0x04
#define PAGE_SIZE   0x80U       //128 words

#elif defined __AVR_ATmega128__
#define SIG2        0x97
#define SIG3        0x02
#define PAGE_SIZE   0x80U       //128 words

#elif defined __AVR_ATmega64__
#define SIG2        0x96
#define SIG3        0x02
```

```
#define PAGE_SIZE     0x80U        //128 words

#elif defined __AVR_ATmega32__
#define SIG2      0x95
#define SIG3      0x02
#define PAGE_SIZE     0x40U        //64 words

#elif defined __AVR_ATmega16__
#define SIG2      0x94
#define SIG3      0x03
#define PAGE_SIZE     0x40U        //64 words

#elif defined __AVR_ATmega8__
#define SIG2      0x93
#define SIG3      0x07
#define PAGE_SIZE     0x20U        //32 words

#elif defined __AVR_ATmega88__
#define SIG2      0x93
#define SIG3      0x0a
#define PAGE_SIZE     0x20U        //32 words

#elif defined __AVR_ATmega168__
#define SIG2      0x94
#define SIG3      0x06
#define PAGE_SIZE     0x40U        //64 words

#elif defined __AVR_ATmega328P__
#define SIG2      0x95
#define SIG3      0x0F
#define PAGE_SIZE     0x40U        //64 words

#elif defined __AVR_ATmega328__
#define SIG2      0x95
#define SIG3      0x14
#define PAGE_SIZE     0x40U        //64 words

#elif defined __AVR_ATmega162__
#define SIG2      0x94
#define SIG3      0x04
#define PAGE_SIZE     0x40U        //64 words

#elif defined __AVR_ATmega163__
#define SIG2      0x94
#define SIG3      0x02
#define PAGE_SIZE     0x40U        //64 words

#elif defined __AVR_ATmega169__
#define SIG2      0x94
#define SIG3      0x05
#define PAGE_SIZE     0x40U        //64 words

#elif defined __AVR_ATmega8515__
#define SIG2      0x93
#define SIG3      0x06
#define PAGE_SIZE     0x20U        //32 words

#elif defined __AVR_ATmega8535__
#define SIG2      0x93
#define SIG3      0x08
#define PAGE_SIZE     0x20U        //32 words
#endif


/* function prototypes */
void putch(char);
char getch(void);
```

```
void getNch(uint8_t);
void byte_response(uint8_t);
void nothing_response(void);
char gethex(void);
void puthex(char);
void flash_led(uint8_t);

/* some variables */
union address_union {
        uint16_t word;
        uint8_t  byte[2];
} address;

union length_union {
        uint16_t word;
        uint8_t  byte[2];
} length;

struct flags_struct {
        unsigned eeprom : 1;
        unsigned rampz  : 1;
} flags;

uint8_t buff[256];
uint8_t address_high;

uint8_t pagesz=0x80;

uint8_t i;
uint8_t bootuart = 0;

uint8_t error_count = 0;

void (*app_start)(void) = 0x0000;


/* main program starts here */
int main(void)
{
        uint8_t ch,ch2;
        uint16_t w;

#ifdef WATCHDOG_MODS
        ch = MCUSR;
        MCUSR = 0;

        WDTCSR |= _BV(WDCE) | _BV(WDE);
        WDTCSR = 0;

        // Check if the WDT was used to reset, in which case we dont bootload and skip straight to the code. woot.
        if (! (ch &  _BV(EXTRF))) // if its a not an external reset...
                app_start();  // skip bootloader
#else
        asm volatile("nop\n\t");
#endif

        /* set pin direction for bootloader pin and enable pullup */
        /* for ATmega128, two pins need to be initialized */
#ifdef __AVR_ATmega128__
        BL_DDR &= ~_BV(BL0);
        BL_DDR &= ~_BV(BL1);
        BL_PORT |= _BV(BL0);
        BL_PORT |= _BV(BL1);
#else
        /* We run the bootloader regardless of the state of this pin.  Thus, don't
        put it in a different state than the other pins.  --DAM, 070709
        This also applies to Arduino Mega -- DC, 080930
```

```
	BL_DDR &= ~_BV(BL);
	BL_PORT |= _BV(BL);
	*/
#endif


#ifdef __AVR_ATmega128__
	/* check which UART should be used for booting */
	if(bit_is_clear(BL_PIN, BL0)) {
		bootuart = 1;
	}
	else if(bit_is_clear(BL_PIN, BL1)) {
		bootuart = 2;
	}
#endif

#if defined __AVR_ATmega1280__
	/* the mega1280 chip has four serial ports ... we could eventually use any of them, or not? */
	/* however, we don't wanna confuse people, to avoid making a mess, we will stick to RXD0, TXD0 */
	bootuart = 1;
#endif

	/* check if flash is programmed already, if not start bootloader anyway */
	if(pgm_read_byte_near(0x0000) != 0xFF) {

#ifdef __AVR_ATmega128__
	/* no UART was selected, start application */
	if(!bootuart) {
		app_start();
	}
#else
	/* check if bootloader pin is set low */
	/* we don't start this part neither for the m8, nor m168 */
	//if(bit_is_set(BL_PIN, BL)) {
	//	app_start();
	//  }
#endif
	}

#ifdef __AVR_ATmega128__
	/* no bootuart was selected, default to uart 0 */
	if(!bootuart) {
		bootuart = 1;
	}
#endif


	/* initialize UART(s) depending on CPU defined */
#if defined(__AVR_ATmega128__) || defined(__AVR_ATmega1280__)
	if(bootuart == 1) {
		UBRR0L = (uint8_t)(F_CPU/(BAUD_RATE*16L)-1);
		UBRR0H = (F_CPU/(BAUD_RATE*16L)-1) >> 8;
		UCSR0A = 0x00;
		UCSR0C = 0x06;
		UCSR0B = _BV(TXEN0)|_BV(RXEN0);
	}
	if(bootuart == 2) {
		UBRR1L = (uint8_t)(F_CPU/(BAUD_RATE*16L)-1);
		UBRR1H = (F_CPU/(BAUD_RATE*16L)-1) >> 8;
		UCSR1A = 0x00;
		UCSR1C = 0x06;
		UCSR1B = _BV(TXEN1)|_BV(RXEN1);
	}
#elif defined __AVR_ATmega163__
	UBRR = (uint8_t)(F_CPU/(BAUD_RATE*16L)-1);
	UBRRHI = (F_CPU/(BAUD_RATE*16L)-1) >> 8;
	UCSRA = 0x00;
```

```
        UCSRB = _BV(TXEN)|_BV(RXEN);
#elif defined(__AVR_ATmega168__) || defined(__AVR_ATmega328P__) || defined (__AVR_ATmega328__)

#ifdef DOUBLE_SPEED
        UCSR0A = (1<<U2X0); //Double speed mode USART0
        UBRR0L = (uint8_t)(F_CPU/(BAUD_RATE*8L)-1);
        UBRR0H = (F_CPU/(BAUD_RATE*8L)-1) >> 8;
#else
        UBRR0L = (uint8_t)(F_CPU/(BAUD_RATE*16L)-1);
        UBRR0H = (F_CPU/(BAUD_RATE*16L)-1) >> 8;
#endif

        UCSR0B = (1<<RXEN0) | (1<<TXEN0);
        UCSR0C = (1<<UCSZ00) | (1<<UCSZ01);

        /* Enable internal pull-up resistor on pin D0 (RX), in order
        to supress line noise that prevents the bootloader from
        timing out (DAM: 20070509) */
        DDRD &= ~_BV(PIND0);
        PORTD |= _BV(PIND0);
#elif defined __AVR_ATmega8__
        /* m8 */
        UBRRH = (((F_CPU/BAUD_RATE)/16)-1)>>8;   // set baud rate
        UBRRL = (((F_CPU/BAUD_RATE)/16)-1);
        UCSRB = (1<<RXEN)|(1<<TXEN);  // enable Rx & Tx
        UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);  // config USART; 8N1
#else
        /* m16,m32,m169,m8515,m8535 */
        UBRRL = (uint8_t)(F_CPU/(BAUD_RATE*16L)-1);
        UBRRH = (F_CPU/(BAUD_RATE*16L)-1) >> 8;
        UCSRA = 0x00;
        UCSRC = 0x06;
        UCSRB = _BV(TXEN)|_BV(RXEN);
#endif

#if defined __AVR_ATmega1280__
        /* Enable internal pull-up resistor on pin D0 (RX), in order
        to supress line noise that prevents the bootloader from
        timing out (DAM: 20070509) */
        /* feature added to the Arduino Mega --DC: 080930 */
        DDRE &= ~_BV(PINE0);
        PORTE |= _BV(PINE0);
#endif


        /* set LED pin as output */
        LED_DDR |= _BV(LED);


        /* flash onboard LED to signal entering of bootloader */
#if defined(__AVR_ATmega128__) || defined(__AVR_ATmega1280__)
        // 4x for UART0, 5x for UART1
        flash_led(NUM_LED_FLASHES + bootuart);
#else
        flash_led(NUM_LED_FLASHES);
#endif

        /* 20050803: by DojoCorp, this is one of the parts provoking the
                system to stop listening, cancelled from the original */
        //putch('\0');

        /* forever loop */
        for (;;) {

        /* get character from UART */
        ch = getch();
```

```
	/* A bunch of if...else if... gives smaller code than switch...case ! */

	/* Hello is anyone home ? */
	if(ch=='0') {
		nothing_response();
	}


	/* Request programmer ID */
	/* Not using PROGMEM string due to boot block in m128 being beyond 64kB boundry  */
	/* Would need to selectively manipulate RAMPZ, and it's only 9 characters anyway so who cares.  */
	else if(ch=='1') {
		if (getch() == ' ') {
			putch(0x14);
			putch('A');
			putch('V');
			putch('R');
			putch(' ');
			putch('I');
			putch('S');
			putch('P');
			putch(0x10);
		} else {
			if (++error_count == MAX_ERROR_COUNT)
				app_start();
		}
	}


	/* AVR ISP/STK500 board commands  DON'T CARE so default nothing_response */
	else if(ch=='@') {
		ch2 = getch();
		if (ch2>0x85) getch();
		nothing_response();
	}


	/* AVR ISP/STK500 board requests */
	else if(ch=='A') {
		ch2 = getch();
		if(ch2==0x80) byte_response(HW_VER);		// Hardware version
		else if(ch2==0x81) byte_response(SW_MAJOR);	// Software major version
		else if(ch2==0x82) byte_response(SW_MINOR);	// Software minor version
		else if(ch2==0x98) byte_response(0x03);		// Unknown but seems to be required by avr studio 3.56
		else byte_response(0x00);			// Covers various unnecessary responses we don't care
about
	}


	/* Device Parameters  DON'T CARE, DEVICE IS FIXED  */
	else if(ch=='B') {
		getNch(20);
		nothing_response();
	}


	/* Parallel programming stuff  DON'T CARE  */
	else if(ch=='E') {
		getNch(5);
		nothing_response();
	}


	/* P: Enter programming mode  */
	/* R: Erase device, don't care as we will erase one page at a time anyway.  */
	else if(ch=='P' || ch=='R') {
		nothing_response();
```

```
	}


	/* Leave programming mode  */
	else if(ch=='Q') {
		nothing_response();
#ifdef WATCHDOG_MODS
		// autoreset via watchdog (sneaky!)
		WDTCSR = _BV(WDE);
		while (1); // 16 ms
#endif
	}


	/* Set address, little endian. EEPROM in bytes, FLASH in words  */
	/* Perhaps extra address bytes may be added in future to support > 128kB FLASH.  */
	/* This might explain why little endian was used here, big endian used everywhere else.  */
	else if(ch=='U') {
		address.byte[0] = getch();
		address.byte[1] = getch();
		nothing_response();
	}


	/* Universal SPI programming command, disabled.  Would be used for fuses and lock bits.  */
	else if(ch=='V') {
		if (getch() == 0x30) {
			getch();
			ch = getch();
			getch();
			if (ch == 0) {
				byte_response(SIG1);
			} else if (ch == 1) {
				byte_response(SIG2);
			} else {
				byte_response(SIG3);
			}
		} else {
			getNch(3);
			byte_response(0x00);
		}
	}


	/* Write memory, length is big endian and is in bytes  */
	else if(ch=='d') {
		length.byte[1] = getch();
		length.byte[0] = getch();
		flags.eeprom = 0;
		if (getch() == 'E') flags.eeprom = 1;
		for (w=0;w<length.word;w++) {
			buff[w] = getch();                        // Store data in buffer, can't keep up with serial data stream whilst
programming pages
		}
		if (getch() == ' ') {
			if (flags.eeprom) {                    //Write to EEPROM one byte at a time
				address.word <<= 1;
				for(w=0;w<length.word;w++) {
#if defined(__AVR_ATmega168__)  || defined(__AVR_ATmega328P__) || defined(__AVR_ATmega328__)
					while(EECR & (1<<EEPE));
					EEAR = (uint16_t)(void *)address.word;
					EEDR = buff[w];
					EECR |= (1<<EEMPE);
					EECR |= (1<<EEPE);
#else
					eeprom_write_byte((void *)address.word,buff[w]);
#endif
```

```
                address.word++;
            }
        }
        else {                                  //Write to FLASH one page at a time
            if (address.byte[1]>127) address_high = 0x01;   //Only possible with m128, m256 will need
3rd address byte. FIXME
            else address_high = 0x00;
#if defined(__AVR_ATmega128__) || defined(__AVR_ATmega1280__) || defined(__AVR_ATmega1281__)
            RAMPZ = address_high;
#endif
            address.word = address.word << 1;        //address * 2 -> byte location
            /* if ((length.byte[0] & 0x01) == 0x01) length.word++;  //Even up an odd number of bytes */
            if ((length.byte[0] & 0x01)) length.word++; //Even up an odd number of bytes
            cli();                              //Disable interrupts, just to be sure
#if defined(EEPE)
            while(bit_is_set(EECR,EEPE));                   //Wait for previous EEPROM writes to
complete
#else
            while(bit_is_set(EECR,EEWE));                   //Wait for previous EEPROM writes to
complete
#endif
            asm volatile(
                "clr   r17          \n\t"   //page_word_count
                "lds   r30,address\n\t"   //Address of FLASH location (in bytes)
                "lds   r31,address+1    \n\t"
                "ldi   r28,lo8(buff)        \n\t"   //Start of buffer array in RAM
                "ldi   r29,hi8(buff)        \n\t"
                "lds   r24,length   \n\t"   //Length of data to be written (in bytes)
                "lds   r25,length+1         \n\t"
                "length_loop:               \n\t"   //Main loop, repeat for number of words in block

                "cpi   r17,0x00     \n\t"   //If page_word_count=0 then erase page
                "brneno_page_erase   \n\t"
                "wait_spm1:                 \n\t"
                "lds   r16,%0               \n\t"   //Wait for previous spm to complete
                "andi r16,1         \n\t"
                "cpi   r16,1        \n\t"
                "breqwait_spm1      \n\t"
                "ldi   r16,0x03     \n\t"   //Erase page pointed to by Z
                "sts   %0,r16               \n\t"
                "spm                \n\t"
#ifdef __AVR_ATmega163__
                ".word 0xFFFF               \n\t"
                "nop                \n\t"
#endif
                "wait_spm2:                 \n\t"
                "lds   r16,%0               \n\t"   //Wait for previous spm to complete
                "andi r16,1         \n\t"
                "cpi   r16,1        \n\t"
                "breqwait_spm2      \n\t"

                "ldi   r16,0x11     \n\t"   //Re-enable RWW section
                "sts   %0,r16               \n\t"
                "spm                \n\t"
#ifdef __AVR_ATmega163__
                ".word 0xFFFF               \n\t"
                "nop                \n\t"
#endif
                "no_page_erase:         \n\t"
                "ld    r0,Y+        \n\t"   //Write 2 bytes into page buffer
                "ld    r1,Y+        \n\t"

                "wait_spm3:                 \n\t"
                "lds   r16,%0               \n\t"   //Wait for previous spm to complete
                "andi r16,1         \n\t"
                "cpi   r16,1        \n\t"
                "breqwait_spm3      \n\t"
```

```
                                "ldi    r16,0x01    \n\t"   //Load r0,r1 into FLASH page buffer
                                "sts    %0,r16              \n\t"
                                "spm                \n\t"

                                "inc    r17             \n\t"   //page_word_count++
                                "cpi r17,%1                 \n\t"
                                "brlo same_page\n\t"    //Still same page in FLASH
                                "write_page:                \n\t"
                                "clr    r17             \n\t"   //New page, write current one first
                                "wait_spm4:                 \n\t"
                                "lds    r16,%0              \n\t"   //Wait for previous spm to complete
                                "andi r16,1         \n\t"
                                "cpi    r16,1           \n\t"
                                "breqwait_spm4      \n\t"
#ifdef __AVR_ATmega163__
                                "andi r30,0x80      \n\t"   // m163 requires Z6:Z1 to be zero during page write
#endif
                                "ldi    r16,0x05    \n\t"   //Write page pointed to by Z
                                "sts    %0,r16              \n\t"
                                "spm                \n\t"
#ifdef __AVR_ATmega163__
                                ".word 0xFFFF               \n\t"
                                "nop                \n\t"
                                "ori    r30,0x7E    \n\t"   // recover Z6:Z1 state after page write (had to be zero during
write)
#endif
                                "wait_spm5:                 \n\t"
                                "lds    r16,%0              \n\t"   //Wait for previous spm to complete
                                "andi r16,1         \n\t"
                                "cpi    r16,1           \n\t"
                                "breqwait_spm5      \n\t"
                                "ldi    r16,0x11    \n\t"   //Re-enable RWW section
                                "sts    %0,r16              \n\t"
                                "spm                \n\t"
#ifdef __AVR_ATmega163__
                                ".word 0xFFFF               \n\t"
                                "nop                \n\t"
#endif
                                "same_page:                 \n\t"
                                "adiw       r30,2           \n\t"   //Next word in FLASH
                                "sbiw r24,2         \n\t"   //length-2
                                "breqfinal_write    \n\t"   //Finished
                                "rjmp length_loop\n\t"
                                "final_write:               \n\t"
                                "cpi    r17,0           \n\t"
                                "breqblock_done\n\t"
                                "adiw       r24,2           \n\t"   //length+2, fool above check on length after short page
write
                                "rjmp write_page    \n\t"
                                "block_done:                \n\t"
                                "clr    __zero_reg__        \n\t"   //restore zero register
#if defined(__AVR_ATmega168__) || defined(__AVR_ATmega328P__) || defined(__AVR_ATmega328__) ||
defined(__AVR_ATmega128__) || defined(__AVR_ATmega1280__) || defined(__AVR_ATmega1281__)
                                : "=m" (SPMCSR) : "M" (PAGE_SIZE) :
"r0","r16","r17","r24","r25","r28","r29","r30","r31"
#else
                                : "=m" (SPMCR) : "M" (PAGE_SIZE) : "r0","r16","r17","r24","r25","r28","r29","r30","r31"
#endif
                                );
                        /* Should really add a wait for RWW section to be enabled, don't actually need it since we
never */
                        /* exit the bootloader without a power cycle anyhow */
                }
                putch(0x14);
                putch(0x10);
        } else {
                if (++error_count == MAX_ERROR_COUNT)
```

```
                        app_start();
                }
        }


        /* Read memory block mode, length is big endian.  */
        else if(ch=='t') {
                length.byte[1] = getch();
                length.byte[0] = getch();
#if defined(__AVR_ATmega128__) || defined(__AVR_ATmega1280__)
                if (address.word>0x7FFF) flags.rampz = 1;               // No go with m256, FIXME
                else flags.rampz = 0;
#endif
                address.word = address.word << 1;       // address * 2 -> byte location
                if (getch() == 'E') flags.eeprom = 1;
                else flags.eeprom = 0;
                if (getch() == ' ') {                   // Command terminator
                        putch(0x14);
                        for (w=0;w < length.word;w++) {                 // Can handle odd and even lengths okay
                                if (flags.eeprom) {                     // Byte access EEPROM read
#if defined(__AVR_ATmega168__) || defined(__AVR_ATmega328P__) || defined(__AVR_ATmega328__)
                                        while(EECR & (1<<EEPE));
                                        EEAR = (uint16_t)(void *)address.word;
                                        EECR |= (1<<EERE);
                                        putch(EEDR);
#else
                                        putch(eeprom_read_byte((void *)address.word));
#endif
                                        address.word++;
                                }
                                else {

                                        if (!flags.rampz) putch(pgm_read_byte_near(address.word));
#if defined(__AVR_ATmega128__) || defined(__AVR_ATmega1280__)
                                        else putch(pgm_read_byte_far(address.word + 0x10000));
                                        // Hmmmm, yuck  FIXME when m256 arrvies
#endif
                                        address.word++;
                                }
                        }
                        putch(0x10);
                }
        }


        /* Get device signature bytes  */
        else if(ch=='u') {
                if (getch() == ' ') {
                        putch(0x14);
                        putch(SIG1);
                        putch(SIG2);
                        putch(SIG3);
                        putch(0x10);
                } else {
                        if (++error_count == MAX_ERROR_COUNT)
                                app_start();
                }
        }


        /* Read oscillator calibration byte */
        else if(ch=='v') {
                byte_response(0x00);
        }


#if defined MONITOR
```

```
	/* here come the extended monitor commands by Erik Lins */

	/* check for three times exclamation mark pressed */
	else if(ch=='!') {
		ch = getch();
		if(ch=='!') {
		ch = getch();
		if(ch=='!') {
			PGM_P welcome = "";
#if defined(__AVR_ATmega128__) || defined(__AVR_ATmega1280__)
			uint16_t extaddr;
#endif
			uint8_t addrl, addrh;

#ifdef CRUMB128
			welcome = "ATmegaBOOT / Crumb128 - (C) J.P.Kyle, E.Lins - 050815\n\r";
#elif defined PROBOMEGA128
			welcome = "ATmegaBOOT / PROBOmega128 - (C) J.P.Kyle, E.Lins - 050815\n\r";
#elif defined SAVVY128
			welcome = "ATmegaBOOT / Savvy128 - (C) J.P.Kyle, E.Lins - 050815\n\r";
#elif defined __AVR_ATmega1280__
			welcome = "ATmegaBOOT / Arduino Mega - (C) Arduino LLC - 090930\n\r";
#endif

			/* turn on LED */
			LED_DDR |= _BV(LED);
			LED_PORT &= ~_BV(LED);

			/* print a welcome message and command overview */
			for(i=0; welcome[i] != '\0'; ++i) {
				putch(welcome[i]);
			}

			/* test for valid commands */
			for(;;) {
				putch('\n');
				putch('\r');
				putch(':');
				putch(' ');

				ch = getch();
				putch(ch);

				/* toggle LED */
				if(ch == 't') {
					if(bit_is_set(LED_PIN,LED)) {
						LED_PORT &= ~_BV(LED);
						putch('1');
					} else {
						LED_PORT |= _BV(LED);
						putch('0');
					}
				}

				/* read byte from address */
				else if(ch == 'r') {
					ch = getch(); putch(ch);
					addrh = gethex();
					addrl = gethex();
					putch('=');
					ch = *(uint8_t *)((addrh << 8) + addrl);
					puthex(ch);
				}

				/* write a byte to address  */
				else if(ch == 'w') {
```

```
                                ch = getch(); putch(ch);
                                addrh = gethex();
                                addrl = gethex();
                                ch = getch(); putch(ch);
                                ch = gethex();
                                *(uint8_t *)((addrh << 8) + addrl) = ch;
                        }

                        /* read from uart and echo back */
                        else if(ch == 'u') {
                                for(;;) {
                                        putch(getch());
                                }
                        }
#if defined(__AVR_ATmega128__) || defined(__AVR_ATmega1280__)
                        /* external bus loop  */
                        else if(ch == 'b') {
                                putch('b');
                                putch('u');
                                putch('s');
                                MCUCR = 0x80;
                                XMCRA = 0;
                                XMCRB = 0;
                                extaddr = 0x1100;
                                for(;;) {
                                        ch = *(volatile uint8_t *)extaddr;
                                        if(++extaddr == 0) {
                                                extaddr = 0x1100;
                                        }
                                }
                        }
#endif

                        else if(ch == 'j') {
                                app_start();
                        }

                } /* end of monitor functions */

        }
        }
    }
    /* end of monitor */
#endif
    else if (++error_count == MAX_ERROR_COUNT) {
        app_start();
    }
    } /* end of forever loop */

}


char gethexnib(void) {
    char a;
    a = getch(); putch(a);
    if(a >= 'a') {
        return (a - 'a' + 0x0a);
    } else if(a >= '0') {
        return(a - '0');
    }
    return a;
}


char gethex(void) {
    return (gethexnib() << 4) + gethexnib();
}
```

```
void puthex(char ch) {
	char ah;

	ah = ch >> 4;
	if(ah >= 0x0a) {
		ah = ah - 0x0a + 'a';
	} else {
		ah += '0';
	}

	ch &= 0x0f;
	if(ch >= 0x0a) {
		ch = ch - 0x0a + 'a';
	} else {
		ch += '0';
	}

	putch(ah);
	putch(ch);
}


void putch(char ch)
{
#if defined(__AVR_ATmega128__) || defined(__AVR_ATmega1280__)
	if(bootuart == 1) {
		while (!(UCSR0A & _BV(UDRE0)));
		UDR0 = ch;
	}
	else if (bootuart == 2) {
		while (!(UCSR1A & _BV(UDRE1)));
		UDR1 = ch;
	}
#elif defined(__AVR_ATmega168__) || defined(__AVR_ATmega328P__) || defined (__AVR_ATmega328__)
	while (!(UCSR0A & _BV(UDRE0)));
	UDR0 = ch;
#else
	/* m8,16,32,169,8515,8535,163 */
	while (!(UCSRA & _BV(UDRE)));
	UDR = ch;
#endif
}


char getch(void)
{
#if defined(__AVR_ATmega128__) || defined(__AVR_ATmega1280__)
	uint32_t count = 0;
	if(bootuart == 1) {
		while(!(UCSR0A & _BV(RXC0))) {
			/* 20060803 DojoCorp:: Addon coming from the previous Bootloader*/
			/* HACKME:: here is a good place to count times*/
			count++;
			if (count > MAX_TIME_COUNT)
				app_start();
		}

		return UDR0;
	}
	else if(bootuart == 2) {
		while(!(UCSR1A & _BV(RXC1))) {
			/* 20060803 DojoCorp:: Addon coming from the previous Bootloader*/
			/* HACKME:: here is a good place to count times*/
			count++;
			if (count > MAX_TIME_COUNT)
```

```
                        app_start();
                }

                return UDR1;
        }
        return 0;
#elif defined(__AVR_ATmega168__)  || defined(__AVR_ATmega328P__) || defined (__AVR_ATmega328__)
        uint32_t count = 0;
        while(!(UCSR0A & _BV(RXC0))){
                /* 20060803 DojoCorp:: Addon coming from the previous Bootloader*/
                /* HACKME:: here is a good place to count times*/
                count++;
                if (count > MAX_TIME_COUNT)
                        app_start();
        }
        return UDR0;
#else
        /* m8,16,32,169,8515,8535,163 */
        uint32_t count = 0;
        while(!(UCSRA & _BV(RXC))){
                /* 20060803 DojoCorp:: Addon coming from the previous Bootloader*/
                /* HACKME:: here is a good place to count times*/
                count++;
                if (count > MAX_TIME_COUNT)
                        app_start();
        }
        return UDR;
#endif
}


void getNch(uint8_t count)
{
        while(count--) {
#if defined(__AVR_ATmega128__) || defined(__AVR_ATmega1280__)
                if(bootuart == 1) {
                        while(!(UCSR0A & _BV(RXC0)));
                        UDR0;
                }
                else if(bootuart == 2) {
                        while(!(UCSR1A & _BV(RXC1)));
                        UDR1;
                }
#elif defined(__AVR_ATmega168__)  || defined(__AVR_ATmega328P__) || defined (__AVR_ATmega328__)
                getch();
#else
                /* m8,16,32,169,8515,8535,163 */
                /* 20060803 DojoCorp:: Addon coming from the previous Bootloader*/
                //while(!(UCSRA & _BV(RXC)));
                //UDR;
                getch(); // need to handle time out
#endif
        }
}


void byte_response(uint8_t val)
{
        if (getch() == ' ') {
                putch(0x14);
                putch(val);
                putch(0x10);
        } else {
                if (++error_count == MAX_ERROR_COUNT)
                        app_start();
        }
}
```

```
void nothing_response(void)
{
	if (getch() == ' ') {
		putch(0x14);
		putch(0x10);
	} else {
		if (++error_count == MAX_ERROR_COUNT)
			app_start();
	}
}

void flash_led(uint8_t count)
{
	while (count--) {
		LED_PORT |= _BV(LED);
		_delay_ms(100);
		LED_PORT &= ~_BV(LED);
		_delay_ms(100);
	}
}


/* end of file ATmegaBOOT.c */
```

ANEXO B

CODIGO FUENTE DEL PROTOCOLO FIRMATA

```
/*
  Firmata is a generic protocol for communicating with microcontrollers
  from software on a host computer. It is intended to work with
  any host computer software package.

  To download a host software package, please clink on the following link
  to open the download page in your default browser.

  https://github.com/firmata/arduino#firmata-client-libraries

  Copyright (C) 2006-2008 Hans-Christoph Steiner.  All rights reserved.
  Copyright (C) 2010-2011 Paul Stoffregen.  All rights reserved.
  Copyright (C) 2009 Shigeru Kobayashi.  All rights reserved.
  Copyright (C) 2009-2015 Jeff Hoefs.  All rights reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  See file LICENSE.txt for further informations on licensing terms.

  Last updated by Jeff Hoefs: April 11, 2015
*/

#include <Servo.h>
#include <Wire.h>
#include <Firmata.h>

#define I2C_WRITE                   B00000000
#define I2C_READ                    B00001000
#define I2C_READ_CONTINUOUSLY       B00010000
#define I2C_STOP_READING            B00011000
#define I2C_READ_WRITE_MODE_MASK    B00011000
#define I2C_10BIT_ADDRESS_MODE_MASK B00100000
#define MAX_QUERIES                 8
#define REGISTER_NOT_SPECIFIED      -1

// the minimum interval for sampling analog input
#define MINIMUM_SAMPLING_INTERVAL 10


/*==============================================================================
 * GLOBAL VARIABLES
 *============================================================================*/

/* analog inputs */
int analogInputsToReport = 0; // bitwise array to store pin reporting

/* digital input ports */
byte reportPINs[TOTAL_PORTS];       // 1 = report this port, 0 = silence
byte previousPINs[TOTAL_PORTS];     // previous 8 bits sent

/* pins configuration */
byte pinConfig[TOTAL_PINS];         // configuration of every pin
byte portConfigInputs[TOTAL_PORTS]; // each bit: 1 = pin in INPUT, 0 = anything else
int pinState[TOTAL_PINS];           // any value that has been written

/* timer variables */
unsigned long currentMillis;        // store the current value from millis()
unsigned long previousMillis;       // for comparison with currentMillis
unsigned int samplingInterval = 19; // how often to run the main loop (in ms)

/* i2c data */
struct i2c_device_info {
  byte addr;
  int reg;
```

```
  byte bytes;
};

/* for i2c read continuous more */
i2c_device_info query[MAX_QUERIES];

byte i2cRxData[32];
boolean isI2CEnabled = false;
signed char queryIndex = -1;
// default delay time between i2c read request and Wire.requestFrom()
unsigned int i2cReadDelayTime = 0;

Servo servos[MAX_SERVOS];
byte servoPinMap[TOTAL_PINS];
byte detachedServos[MAX_SERVOS];
byte detachedServoCount = 0;
byte servoCount = 0;

boolean isResetting = false;

/* utility functions */
void wireWrite(byte data)
{
#if ARDUINO >= 100
  Wire.write((byte)data);
#else
  Wire.send(data);
#endif
}

byte wireRead(void)
{
#if ARDUINO >= 100
  return Wire.read();
#else
  return Wire.receive();
#endif
}

/*==============================================================================
 * FUNCTIONS
 *============================================================================*/

void attachServo(byte pin, int minPulse, int maxPulse)
{
  if (servoCount < MAX_SERVOS) {
    // reuse indexes of detached servos until all have been reallocated
    if (detachedServoCount > 0) {
      servoPinMap[pin] = detachedServos[detachedServoCount - 1];
      if (detachedServoCount > 0) detachedServoCount--;
    } else {
      servoPinMap[pin] = servoCount;
      servoCount++;
    }
    if (minPulse > 0 && maxPulse > 0) {
      servos[servoPinMap[pin]].attach(PIN_TO_DIGITAL(pin), minPulse, maxPulse);
    } else {
      servos[servoPinMap[pin]].attach(PIN_TO_DIGITAL(pin));
    }
  } else {
    Firmata.sendString("Max servos attached");
  }
}

void detachServo(byte pin)
{
  servos[servoPinMap[pin]].detach();
```

```
  // if we're detaching the last servo, decrement the count
  // otherwise store the index of the detached servo
  if (servoPinMap[pin] == servoCount && servoCount > 0) {
    servoCount--;
  } else if (servoCount > 0) {
    // keep track of detached servos because we want to reuse their indexes
    // before incrementing the count of attached servos
    detachedServoCount++;
    detachedServos[detachedServoCount - 1] = servoPinMap[pin];
  }

  servoPinMap[pin] = 255;
}

void readAndReportData(byte address, int theRegister, byte numBytes) {
  // allow I2C requests that don't require a register read
  // for example, some devices using an interrupt pin to signify new data available
  // do not always require the register read so upon interrupt you call Wire.requestFrom()
  if (theRegister != REGISTER_NOT_SPECIFIED) {
    Wire.beginTransmission(address);
    wireWrite((byte)theRegister);
    Wire.endTransmission();
    // do not set a value of 0
    if (i2cReadDelayTime > 0) {
      // delay is necessary for some devices such as WiiNunchuck
      delayMicroseconds(i2cReadDelayTime);
    }
  } else {
    theRegister = 0;  // fill the register with a dummy value
  }

  Wire.requestFrom(address, numBytes);  // all bytes are returned in requestFrom

  // check to be sure correct number of bytes were returned by slave
  if (numBytes < Wire.available()) {
    Firmata.sendString("I2C: Too many bytes received");
  } else if (numBytes > Wire.available()) {
    Firmata.sendString("I2C: Too few bytes received");
  }

  i2cRxData[0] = address;
  i2cRxData[1] = theRegister;

  for (int i = 0; i < numBytes && Wire.available(); i++) {
    i2cRxData[2 + i] = wireRead();
  }

  // send slave address, register and received bytes
  Firmata.sendSysex(SYSEX_I2C_REPLY, numBytes + 2, i2cRxData);
}

void outputPort(byte portNumber, byte portValue, byte forceSend)
{
  // pins not configured as INPUT are cleared to zeros
  portValue = portValue & portConfigInputs[portNumber];
  // only send if the value is different than previously sent
  if (forceSend || previousPINs[portNumber] != portValue) {
    Firmata.sendDigitalPort(portNumber, portValue);
    previousPINs[portNumber] = portValue;
  }
}

/* -----------------------------------------------------------------------------
 * check all the active digital inputs for change of state, then add any events
 * to the Serial output queue using Serial.print() */
void checkDigitalInputs(void)
{
```

```
  /* Using non-looping code allows constants to be given to readPort().
   * The compiler will apply substantial optimizations if the inputs
   * to readPort() are compile-time constants. */
  if (TOTAL_PORTS > 0 && reportPINs[0]) outputPort(0, readPort(0, portConfigInputs[0]), false);
  if (TOTAL_PORTS > 1 && reportPINs[1]) outputPort(1, readPort(1, portConfigInputs[1]), false);
  if (TOTAL_PORTS > 2 && reportPINs[2]) outputPort(2, readPort(2, portConfigInputs[2]), false);
  if (TOTAL_PORTS > 3 && reportPINs[3]) outputPort(3, readPort(3, portConfigInputs[3]), false);
  if (TOTAL_PORTS > 4 && reportPINs[4]) outputPort(4, readPort(4, portConfigInputs[4]), false);
  if (TOTAL_PORTS > 5 && reportPINs[5]) outputPort(5, readPort(5, portConfigInputs[5]), false);
  if (TOTAL_PORTS > 6 && reportPINs[6]) outputPort(6, readPort(6, portConfigInputs[6]), false);
  if (TOTAL_PORTS > 7 && reportPINs[7]) outputPort(7, readPort(7, portConfigInputs[7]), false);
  if (TOTAL_PORTS > 8 && reportPINs[8]) outputPort(8, readPort(8, portConfigInputs[8]), false);
  if (TOTAL_PORTS > 9 && reportPINs[9]) outputPort(9, readPort(9, portConfigInputs[9]), false);
  if (TOTAL_PORTS > 10 && reportPINs[10]) outputPort(10, readPort(10, portConfigInputs[10]), false);
  if (TOTAL_PORTS > 11 && reportPINs[11]) outputPort(11, readPort(11, portConfigInputs[11]), false);
  if (TOTAL_PORTS > 12 && reportPINs[12]) outputPort(12, readPort(12, portConfigInputs[12]), false);
  if (TOTAL_PORTS > 13 && reportPINs[13]) outputPort(13, readPort(13, portConfigInputs[13]), false);
  if (TOTAL_PORTS > 14 && reportPINs[14]) outputPort(14, readPort(14, portConfigInputs[14]), false);
  if (TOTAL_PORTS > 15 && reportPINs[15]) outputPort(15, readPort(15, portConfigInputs[15]), false);
}

// -----------------------------------------------------------------------------
/* sets the pin mode to the correct state and sets the relevant bits in the
 * two bit-arrays that track Digital I/O and PWM status
 */
void setPinModeCallback(byte pin, int mode)
{
  if (pinConfig[pin] == IGNORE)
    return;

  if (pinConfig[pin] == I2C && isI2CEnabled && mode != I2C) {
    // disable i2c so pins can be used for other functions
    // the following if statements should reconfigure the pins properly
    disableI2CPins();
  }
  if (IS_PIN_DIGITAL(pin) && mode != SERVO) {
    if (servoPinMap[pin] < MAX_SERVOS && servos[servoPinMap[pin]].attached()) {
      detachServo(pin);
    }
  }
  if (IS_PIN_ANALOG(pin)) {
    reportAnalogCallback(PIN_TO_ANALOG(pin), mode == ANALOG ? 1 : 0); // turn on/off reporting
  }
  if (IS_PIN_DIGITAL(pin)) {
    if (mode == INPUT) {
      portConfigInputs[pin / 8] |= (1 << (pin & 7));
    } else {
      portConfigInputs[pin / 8] &= ~(1 << (pin & 7));
    }
  }
  pinState[pin] = 0;
  switch (mode) {
    case ANALOG:
      if (IS_PIN_ANALOG(pin)) {
        if (IS_PIN_DIGITAL(pin)) {
          pinMode(PIN_TO_DIGITAL(pin), INPUT);    // disable output driver
          digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
        }
        pinConfig[pin] = ANALOG;
      }
      break;
    case INPUT:
      if (IS_PIN_DIGITAL(pin)) {
        pinMode(PIN_TO_DIGITAL(pin), INPUT);    // disable output driver
        digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
        pinConfig[pin] = INPUT;
      }
```

```
      break;
    case OUTPUT:
      if (IS_PIN_DIGITAL(pin)) {
        digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable PWM
        pinMode(PIN_TO_DIGITAL(pin), OUTPUT);
        pinConfig[pin] = OUTPUT;
      }
      break;
    case PWM:
      if (IS_PIN_PWM(pin)) {
        pinMode(PIN_TO_PWM(pin), OUTPUT);
        analogWrite(PIN_TO_PWM(pin), 0);
        pinConfig[pin] = PWM;
      }
      break;
    case SERVO:
      if (IS_PIN_DIGITAL(pin)) {
        pinConfig[pin] = SERVO;
        if (servoPinMap[pin] == 255 || !servos[servoPinMap[pin]].attached()) {
          // pass -1 for min and max pulse values to use default values set
          // by Servo library
          attachServo(pin, -1, -1);
        }
      }
      break;
    case I2C:
      if (IS_PIN_I2C(pin)) {
        // mark the pin as i2c
        // the user must call I2C_CONFIG to enable I2C for a device
        pinConfig[pin] = I2C;
      }
      break;
    default:
      Firmata.sendString("Unknown pin mode"); // TODO: put error msgs in EEPROM
  }
  // TODO: save status to EEPROM here, if changed
}

void analogWriteCallback(byte pin, int value)
{
  if (pin < TOTAL_PINS) {
    switch (pinConfig[pin]) {
      case SERVO:
        if (IS_PIN_DIGITAL(pin))
          servos[servoPinMap[pin]].write(value);
        pinState[pin] = value;
        break;
      case PWM:
        if (IS_PIN_PWM(pin))
          analogWrite(PIN_TO_PWM(pin), value);
        pinState[pin] = value;
        break;
    }
  }
}

void digitalWriteCallback(byte port, int value)
{
  byte pin, lastPin, mask = 1, pinWriteMask = 0;

  if (port < TOTAL_PORTS) {
    // create a mask of the pins on this port that are writable.
    lastPin = port * 8 + 8;
    if (lastPin > TOTAL_PINS) lastPin = TOTAL_PINS;
    for (pin = port * 8; pin < lastPin; pin++) {
      // do not disturb non-digital pins (eg, Rx & Tx)
      if (IS_PIN_DIGITAL(pin)) {
```

```
        // only write to OUTPUT and INPUT (enables pullup)
        // do not touch pins in PWM, ANALOG, SERVO or other modes
        if (pinConfig[pin] == OUTPUT || pinConfig[pin] == INPUT) {
          pinWriteMask |= mask;
          pinState[pin] = ((byte)value & mask) ? 1 : 0;
        }
      }
      mask = mask << 1;
    }
    writePort(port, (byte)value, pinWriteMask);
  }
}


// -----------------------------------------------------------------------------
/* sets bits in a bit array (int) to toggle the reporting of the analogIns
 */
//void FirmataClass::setAnalogPinReporting(byte pin, byte state) {
//}
void reportAnalogCallback(byte analogPin, int value)
{
  if (analogPin < TOTAL_ANALOG_PINS) {
    if (value == 0) {
      analogInputsToReport = analogInputsToReport & ~ (1 << analogPin);
    } else {
      analogInputsToReport = analogInputsToReport | (1 << analogPin);
      // prevent during system reset or all analog pin values will be reported
      // which may report noise for unconnected analog pins
      if (!isResetting) {
        // Send pin value immediately. This is helpful when connected via
        // ethernet, wi-fi or bluetooth so pin states can be known upon
        // reconnecting.
        Firmata.sendAnalog(analogPin, analogRead(analogPin));
      }
    }
  }
  // TODO: save status to EEPROM here, if changed
}

void reportDigitalCallback(byte port, int value)
{
  if (port < TOTAL_PORTS) {
    reportPINs[port] = (byte)value;
    // Send port value immediately. This is helpful when connected via
    // ethernet, wi-fi or bluetooth so pin states can be known upon
    // reconnecting.
    if (value) outputPort(port, readPort(port, portConfigInputs[port]), true);
  }
  // do not disable analog reporting on these 8 pins, to allow some
  // pins used for digital, others analog.  Instead, allow both types
  // of reporting to be enabled, but check if the pin is configured
  // as analog when sampling the analog inputs.  Likewise, while
  // scanning digital pins, portConfigInputs will mask off values from any
  // pins configured as analog
}

/*==============================================================================
 * SYSEX-BASED commands
 *============================================================================*/

void sysexCallback(byte command, byte argc, byte *argv)
{
  byte mode;
  byte slaveAddress;
  byte data;
  int slaveRegister;
  unsigned int delayTime;
```

```
switch (command) {
  case I2C_REQUEST:
    mode = argv[1] & I2C_READ_WRITE_MODE_MASK;
    if (argv[1] & I2C_10BIT_ADDRESS_MODE_MASK) {
      Firmata.sendString("10-bit addressing not supported");
      return;
    }
    else {
      slaveAddress = argv[0];
    }

    switch (mode) {
      case I2C_WRITE:
        Wire.beginTransmission(slaveAddress);
        for (byte i = 2; i < argc; i += 2) {
          data = argv[i] + (argv[i + 1] << 7);
          wireWrite(data);
        }
        Wire.endTransmission();
        delayMicroseconds(70);
        break;
      case I2C_READ:
        if (argc == 6) {
          // a slave register is specified
          slaveRegister = argv[2] + (argv[3] << 7);
          data = argv[4] + (argv[5] << 7);  // bytes to read
        }
        else {
          // a slave register is NOT specified
          slaveRegister = REGISTER_NOT_SPECIFIED;
          data = argv[2] + (argv[3] << 7);  // bytes to read
        }
        readAndReportData(slaveAddress, (int)slaveRegister, data);
        break;
      case I2C_READ_CONTINUOUSLY:
        if ((queryIndex + 1) >= MAX_QUERIES) {
          // too many queries, just ignore
          Firmata.sendString("too many queries");
          break;
        }
        if (argc == 6) {
          // a slave register is specified
          slaveRegister = argv[2] + (argv[3] << 7);
          data = argv[4] + (argv[5] << 7);  // bytes to read
        }
        else {
          // a slave register is NOT specified
          slaveRegister = (int)REGISTER_NOT_SPECIFIED;
          data = argv[2] + (argv[3] << 7);  // bytes to read
        }
        queryIndex++;
        query[queryIndex].addr = slaveAddress;
        query[queryIndex].reg = slaveRegister;
        query[queryIndex].bytes = data;
        break;
      case I2C_STOP_READING:
        byte queryIndexToSkip;
        // if read continuous mode is enabled for only 1 i2c device, disable
        // read continuous reporting for that device
        if (queryIndex <= 0) {
          queryIndex = -1;
        } else {
          // if read continuous mode is enabled for multiple devices,
          // determine which device to stop reading and remove it's data from
          // the array, shifiting other array data to fill the space
          for (byte i = 0; i < queryIndex + 1; i++) {
```

```
          if (query[i].addr == slaveAddress) {
            queryIndexToSkip = i;
            break;
          }
        }

        for (byte i = queryIndexToSkip; i < queryIndex + 1; i++) {
          if (i < MAX_QUERIES) {
            query[i].addr = query[i + 1].addr;
            query[i].reg = query[i + 1].reg;
            query[i].bytes = query[i + 1].bytes;
          }
        }
        queryIndex--;
      }
      break;
    default:
      break;
  }
  break;
case I2C_CONFIG:
  delayTime = (argv[0] + (argv[1] << 7));

  if (delayTime > 0) {
    i2cReadDelayTime = delayTime;
  }

  if (!isI2CEnabled) {
    enableI2CPins();
  }

  break;
case SERVO_CONFIG:
  if (argc > 4) {
    // these vars are here for clarity, they'll optimized away by the compiler
    byte pin = argv[0];
    int minPulse = argv[1] + (argv[2] << 7);
    int maxPulse = argv[3] + (argv[4] << 7);

    if (IS_PIN_DIGITAL(pin)) {
      if (servoPinMap[pin] < MAX_SERVOS && servos[servoPinMap[pin]].attached()) {
        detachServo(pin);
      }
      attachServo(pin, minPulse, maxPulse);
      setPinModeCallback(pin, SERVO);
    }
  }
  break;
case SAMPLING_INTERVAL:
  if (argc > 1) {
    samplingInterval = argv[0] + (argv[1] << 7);
    if (samplingInterval < MINIMUM_SAMPLING_INTERVAL) {
      samplingInterval = MINIMUM_SAMPLING_INTERVAL;
    }
  } else {
    //Firmata.sendString("Not enough data");
  }
  break;
case EXTENDED_ANALOG:
  if (argc > 1) {
    int val = argv[1];
    if (argc > 2) val |= (argv[2] << 7);
    if (argc > 3) val |= (argv[3] << 14);
    analogWriteCallback(argv[0], val);
  }
  break;
case CAPABILITY_QUERY:
```

```
      Firmata.write(START_SYSEX);
      Firmata.write(CAPABILITY_RESPONSE);
      for (byte pin = 0; pin < TOTAL_PINS; pin++) {
        if (IS_PIN_DIGITAL(pin)) {
          Firmata.write((byte)INPUT);
          Firmata.write(1);
          Firmata.write((byte)OUTPUT);
          Firmata.write(1);
        }
        if (IS_PIN_ANALOG(pin)) {
          Firmata.write(ANALOG);
          Firmata.write(10); // 10 = 10-bit resolution
        }
        if (IS_PIN_PWM(pin)) {
          Firmata.write(PWM);
          Firmata.write(8); // 8 = 8-bit resolution
        }
        if (IS_PIN_DIGITAL(pin)) {
          Firmata.write(SERVO);
          Firmata.write(14);
        }
        if (IS_PIN_I2C(pin)) {
          Firmata.write(I2C);
          Firmata.write(1);  // TODO: could assign a number to map to SCL or SDA
        }
        Firmata.write(127);
      }
      Firmata.write(END_SYSEX);
      break;
    case PIN_STATE_QUERY:
      if (argc > 0) {
        byte pin = argv[0];
        Firmata.write(START_SYSEX);
        Firmata.write(PIN_STATE_RESPONSE);
        Firmata.write(pin);
        if (pin < TOTAL_PINS) {
          Firmata.write((byte)pinConfig[pin]);
          Firmata.write((byte)pinState[pin] & 0x7F);
          if (pinState[pin] & 0xFF80) Firmata.write((byte)(pinState[pin] >> 7) & 0x7F);
          if (pinState[pin] & 0xC000) Firmata.write((byte)(pinState[pin] >> 14) & 0x7F);
        }
        Firmata.write(END_SYSEX);
      }
      break;
    case ANALOG_MAPPING_QUERY:
      Firmata.write(START_SYSEX);
      Firmata.write(ANALOG_MAPPING_RESPONSE);
      for (byte pin = 0; pin < TOTAL_PINS; pin++) {
        Firmata.write(IS_PIN_ANALOG(pin) ? PIN_TO_ANALOG(pin) : 127);
      }
      Firmata.write(END_SYSEX);
      break;
  }
}

void enableI2CPins()
{
  byte i;
  // is there a faster way to do this? would probaby require importing
  // Arduino.h to get SCL and SDA pins
  for (i = 0; i < TOTAL_PINS; i++) {
    if (IS_PIN_I2C(i)) {
      // mark pins as i2c so they are ignore in non i2c data requests
      setPinModeCallback(i, I2C);
    }
  }
```

```
  isI2CEnabled = true;

  Wire.begin();
}

/* disable the i2c pins so they can be used for other functions */
void disableI2CPins() {
  isI2CEnabled = false;
  // disable read continuous mode for all devices
  queryIndex = -1;
}

/*==============================================================================
 * SETUP()
 *============================================================================*/

void systemResetCallback()
{
  isResetting = true;

  // initialize a defalt state
  // TODO: option to load config from EEPROM instead of default

  if (isI2CEnabled) {
    disableI2CPins();
  }

  for (byte i = 0; i < TOTAL_PORTS; i++) {
    reportPINs[i] = false;    // by default, reporting off
    portConfigInputs[i] = 0;  // until activated
    previousPINs[i] = 0;
  }

  for (byte i = 0; i < TOTAL_PINS; i++) {
    // pins with analog capability default to analog input
    // otherwise, pins default to digital output
    if (IS_PIN_ANALOG(i)) {
      // turns off pullup, configures everything
      setPinModeCallback(i, ANALOG);
    } else {
      // sets the output to 0, configures portConfigInputs
      setPinModeCallback(i, OUTPUT);
    }

    servoPinMap[i] = 255;
  }
  // by default, do not report any analog inputs
  analogInputsToReport = 0;

  detachedServoCount = 0;
  servoCount = 0;

  /* send digital inputs to set the initial state on the host computer,
   * since once in the loop(), this firmware will only send on change */
  /*
  TODO: this can never execute, since no pins default to digital input
        but it will be needed when/if we support EEPROM stored config
  for (byte i=0; i < TOTAL_PORTS; i++) {
    outputPort(i, readPort(i, portConfigInputs[i]), true);
  }
  */
  isResetting = false;
}

void setup()
{
  Firmata.setFirmwareVersion(FIRMATA_MAJOR_VERSION, FIRMATA_MINOR_VERSION);
```

```
  Firmata.attach(ANALOG_MESSAGE, analogWriteCallback);
  Firmata.attach(DIGITAL_MESSAGE, digitalWriteCallback);
  Firmata.attach(REPORT_ANALOG, reportAnalogCallback);
  Firmata.attach(REPORT_DIGITAL, reportDigitalCallback);
  Firmata.attach(SET_PIN_MODE, setPinModeCallback);
  Firmata.attach(START_SYSEX, sysexCallback);
  Firmata.attach(SYSTEM_RESET, systemResetCallback);

  Firmata.begin(57600);
  systemResetCallback();  // reset to default config
}

/*==============================================================================
 * LOOP()
 *============================================================================*/
void loop()
{
  byte pin, analogPin;

  /* DIGITALREAD - as fast as possible, check for changes and output them to the
   * FTDI buffer using Serial.print()  */
  checkDigitalInputs();

  /* STREAMREAD - processing incoming messagse as soon as possible, while still
   * checking digital inputs.  */
  while (Firmata.available())
    Firmata.processInput();

  // TODO - ensure that Stream buffer doesn't go over 60 bytes

  currentMillis = millis();
  if (currentMillis - previousMillis > samplingInterval) {
    previousMillis += samplingInterval;
    /* ANALOGREAD - do all analogReads() at the configured sampling interval */
    for (pin = 0; pin < TOTAL_PINS; pin++) {
      if (IS_PIN_ANALOG(pin) && pinConfig[pin] == ANALOG) {
        analogPin = PIN_TO_ANALOG(pin);
        if (analogInputsToReport & (1 << analogPin)) {
          Firmata.sendAnalog(analogPin, analogRead(analogPin));
        }
      }
    }
    // report i2c data for all device with read continuous mode enabled
    if (queryIndex > -1) {
      for (byte i = 0; i < queryIndex + 1; i++) {
        readAndReportData(query[i].addr, query[i].reg, query[i].bytes);
      }
    }
  }
}
```


ARDUINO
ÖPENLAB
D2
D3
D4
D5
D6
D7
- GND
+ VCC
A0
A1
D8
D9
Instituto Universitario Politécnico
"Santiago Mariño"
Extensión Maturín
TRABAJO DE GRADO
ING ELECTRONICA
TSU. LEOPOLDO MARCANO
Febrero 2016
DISPOSITIVO DE ADQUISICIÓN DE DATOS CON
INTERFAZ VIRTUAL