

MÁSTER UNIVERSITARIO EN

INGENIERÍA DE SISTEMAS EMPOTRADOS



|  |
| --- |
| **Electromiografía: diseño e implementación en un sistema embebido** |

|  |
| --- |
| **Alumno: Xabier Figueroa Sánchez** |
| **Director: Andoni Arruti** |
| **Instructor en la empresa: ¿?** |

**Proyecto Fin de Máster, septiembre de 2017**

Contenido

[1. INTRODUCCIÓN 7](#_Toc491370283)

[1.1. DESCRIPCIÓN GENERAL DEL PROYECTO 7](#_Toc491370284)

[1.2. OBJETIVO DEL PROYECTO 7](#_Toc491370285)

[1.3. ENMARQUE DEL PROYECTO 8](#_Toc491370286)

[2. ESTADO DEL ARTE 9](#_Toc491370287)

[2.1. INGENIERÍA BIOMÉDICA Y ELECTROMEDICINA 9](#_Toc491370288)

[2.1.1. Descripción 9](#_Toc491370289)

[2.1.2. Historia 9](#_Toc491370290)

[2.2. BIOSEÑALES 9](#_Toc491370291)

[2.2.1. Descripción 9](#_Toc491370292)

[2.2.2. Naturaleza 12](#_Toc491370293)

[2.2.3. Señales fisiológicas 12](#_Toc491370294)

[2.2.4. Tipos y clasificación 12](#_Toc491370295)

[2.3. ELECTROMIOGRAFÍA (EMG) 13](#_Toc491370296)

[2.3.1. Descripción 13](#_Toc491370297)

[2.3.2. Señales 13](#_Toc491370298)

[2.3.3. Digitalización 13](#_Toc491370299)

[2.3.4. Tipos de sensores comerciales 13](#_Toc491370300)

[2.4. FPGA 16](#_Toc491370301)

[2.4.1. Introducción 16](#_Toc491370302)

[2.4.2. Arquitectura 16](#_Toc491370303)

[2.4.3. Tecnologías 18](#_Toc491370304)

[2.4.4. Fabricantes 20](#_Toc491370305)

[2.5. COMUNICACIONES 20](#_Toc491370306)

[2.5.1. Comunicación serie 20](#_Toc491370307)

[2.5.2. Bluetooth 23](#_Toc491370308)

[3. DESCRIPCIÓN DEL DISPOSITIVO 27](#_Toc491370309)

[3.1. DE0 NANO 27](#_Toc491370310)

[3.1.1. Descripción general 27](#_Toc491370311)

[3.1.2. Arquitectura 28](#_Toc491370312)

[3.1.3. Conexiones 29](#_Toc491370313)

[3.1.4. Altera Cyclone IV EP4CE22 31](#_Toc491370314)

[3.1.5. ADC128S022 32](#_Toc491370315)

[3.2. MYOWARE 33](#_Toc491370316)

[3.3. HC-05 (MÓDULO BLUETOOTH) 34](#_Toc491370317)

[4. HERRAMIENTAS DE DESARROLLO 37](#_Toc491370318)

[4.1. DESARROLLO FPGA 37](#_Toc491370319)

[4.1.1. ModelSim 37](#_Toc491370320)

[4.1.2. Quartus 39](#_Toc491370321)

[4.2. CAPTURA DE DATOS 42](#_Toc491370322)

[4.2.1. CoolTerm 43](#_Toc491370323)

[4.2.2. GNU Octave 44](#_Toc491370324)

[4.2.3. Python 45](#_Toc491370325)

[5. TRABAJO DESARROLLADO 48](#_Toc491370326)

[5.1. DESCRIPCIÓN GENERAL 48](#_Toc491370327)

[5.2. ESPECIFICACIONES 49](#_Toc491370328)

[5.2.1. Especificaciones ADC128S022 49](#_Toc491370329)

[5.3. DISEÑO 50](#_Toc491370330)

[5.3.1. Diseño sistema 50](#_Toc491370331)

[5.3.2. FPGA: Esquema general 54](#_Toc491370332)

[5.3.3. FPGA: módulo control ADC 55](#_Toc491370333)

[5.3.4. FPGA: módulo principal 57](#_Toc491370334)

[5.3.5. FPGA: módulo transmisión UART 59](#_Toc491370335)

[5.3.6. FPGA: PLL 62](#_Toc491370336)

[5.3.7. PC: Script captura de datos 62](#_Toc491370337)

[5.3.8. HW: circuito externo 65](#_Toc491370338)

[5.4. IMPLEMENTACIÓN 66](#_Toc491370339)

[5.4.1. Control de ADC 66](#_Toc491370340)

[5.4.2. Transmisión UART 68](#_Toc491370341)

[5.4.3. Sistema completo 69](#_Toc491370342)

[5.4.4. Resultados 76](#_Toc491370343)

[5.5. PROBLEMAS 78](#_Toc491370344)

[5.6. FUTUROS DESARROLLOS 78](#_Toc491370345)

[6. CONCLUSIONES 79](#_Toc491370346)

[7. REFERENCIAS 80](#_Toc491370347)

[ANEXOS 81](#_Toc491370348)

[Anexo A: Simulaciones VHDL 81](#_Toc491370349)

[Anexo B: Manual de Interfaz gráfica 81](#_Toc491370350)

[Anexo C: Comandos AT 81](#_Toc491370351)

[Anexo D: Conexión PC con HC-05 81](#_Toc491370352)

[Anexo E: Conexiones de electrodos 81](#_Toc491370353)

# INTRODUCCIÓN

## **DESCRIPCIÓN GENERAL DEL PROYECTO**

Hoy en día la biomedicina es una rama muy importante dentro de la propia medicina. En los últimos años se está llevado a cabo una rápida evolución, lo cual conlleva una mejora sustancial en la calidad de vida de las personas.

Gracias a las investigaciones y los desarrollos de sistemas muy sofisticados, es posible detectar problemas de salud con mayor antelación, e incluso detectar problemas que hasta ahora eran difícilmente detectables.

Esto es posible gracias a que los sistemas electrónicos son cada día más potentes, lo que dota a los investigadores y desarrolladores de posibilidades que hasta hace pocos años nadie podía imaginar.

El ser humano produce un sinfín de señales eléctricas, las cuales se denominan. Las bioseñales más estudiadas son:

* ECG (electrocardiograma)
* EEG (electroencefalograma)
* EMG (electromiograma)
* MMG (mecanomiograma)
* EOG (electrooculografía)
* GSR (respuesta galvánica de la piel)
* MEG (magnetoencefalograma)

El estudio de estas señales puede ayudar al diagnóstico y a una prematura detección de anomalías y enfermados, las cuales, pueden ser tratadas con tiempo suficiente.

El conocimiento del funcionamiento del cuerpo humano y las señales que este produce, ayuda en el diseño e implementación de sistemas que puedan trabajar como parte del propio cuerpo humano. En estos casos, se puede sustituir un elemento no funcional del cuerpo por un sistema electrónico que cumpla con las mismas especificaciones que dicho elemento.

Existen ejemplos de sustitución de elementos no funcionales (o incluso inexistentes) por elemento electrónicos como, por ejemplo, crear extremidades mecánicas del cuerpo humano conectadas a los músculos/nervios del ser humano para reaccionar ante unos determinados impulsos.

Es por todo esto, que poder obtener y guardar las bioseñales para su posterior estudio es muy importante para la investigación biomédica.

## **OBJETIVO DEL PROYECTO**

El presento proyecto tiene como objetivo el diseño e implementación de un prototipo que sea capaz de obtener las señales eléctricas producidas por los músculos del cuerpo humano y guardarlas en un servidor (PC) para que puedan ser estudiadas.

Para ello se dispone de una placa de desarrollo *DE0-Nano* de *Terasic*, la cual integra un FPGA de Altera (Cyclone IV EP4CE22F17C6N) como elemento principal.

Pretende diseñarse un sistema que sea portable, lo cual exige el cumplimiento de otros objetivos implícitos:

* El sistema tiene que ser autónomo, es decir, tiene que trabajar a pilas o con baterías.
* El sistema tiene que comunicarse con el PC de manera inalámbrica.
* El tamaño ha de ser lo más reducido posible.
* El almacenaje de los datos ha de ser de forma que el posterior manejo de estos datos sea sencillo.

## **ENMARQUE DEL PROYECTO**

¿?¿?¿?¿?¿?¿?¿?¿?¿¿?¿?¿?¿?

# ESTADO DEL ARTE

## **INGENIERÍA BIOMÉDICA**

### Descripción

La ingeniería biomédica se puede definir como el hecho de aplicar principios y técnicas de la ingeniería en el ámbito de la medicina. Sus funciones son la mejora de métodos de prevención, diagnóstico, tratamiento y rehabilitación a través del diseño e implementación de dispositivos tecnológicos para el campo de la medicina, como puede ser, prótesis, equipos de diagnóstico, sistemas de medición, etc.

La ingeniería biomédica se encuentra en pleno crecimiento, y es una de las ramas que más crecimiento ha experimentado. Esto es gracias al gran avance tecnológico que se está realizando en las diferentes áreas de la ingeniería. Un ejemplo claro, es el hecho de que los sistemas embebidos e informáticos son cada vez más potentes, por lo que hoy en día es posible estudiar parámetros con mayor fiabilidad que con una alta resolución.

### Historia

Los primeros indicios de la ingeniería biomédica se remontan más de 3000 años atrás, donde una prótesis de un dedo del pie fue encontrada en Egipto. Pero cuando realmente se hace oficial su comienzo es entre los años 1890 y 1930, cuando el desarrollo de la instrumentación eléctrica y electrónica produjo la invención de nuevas tecnologías. En este periodo se obtiene los primeros registros de biopotenciales. A continuación, se citan alguno se los hitos más importantes dentro de ese período (referencias-> PDF tema 1):

* 1888: August Waller obtiene el primer registro de electrocardiograma.
* 1895: comienza a utilizarse los rayos-x para el diagnóstico de fracturas óseas y dislocaciones. Un año más tarde se comercializa el primer sistema de este tipo.
* 1903: Willem Einthoven obtiene un registro de electrocardiograma mediante galvanómetro de hilo.
* 1911: primer electrocardiógrafo comercial.
* 1930: primer electroencefalógrafo. Posibilidad de visualizar prácticamente todos los órganos mediante utilización de sustancias radiopacas.

En la década de los 50 se las aplicaciones de ingeniería biomédica empiezan a utilizar ordenadores. En esta época también se desarrollan sistemas de diagnóstico por ultrasonidos. Cabe mencionar, que, durante estos años, Paul Zoll desarrolla el primer marcapasos externo (1952), y pocos años más tarde (1958) ya se dispone de marcapasos que pueden ser implantados.

En los años 60 se desarrollan los primeros sistemas telemétricos, los cuales son implantados en unidades de cuidados intensivos.

Durante los 70, aparecen el TAC (escáner de tomografía axial computarizada), RMN (resonancia magnética nuclear) y PET (tomografía por emisión de positrones).

Unos años más tarde, aparecen sistemas de endoscopia y se realiza cirugía por láser (años 80). Y es en los 90 cuando empieza a implantarse la robótica en la medicina.

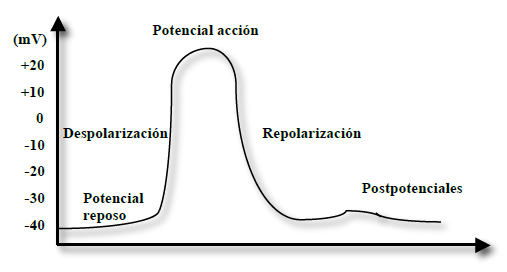
## **BIOSEÑALES**

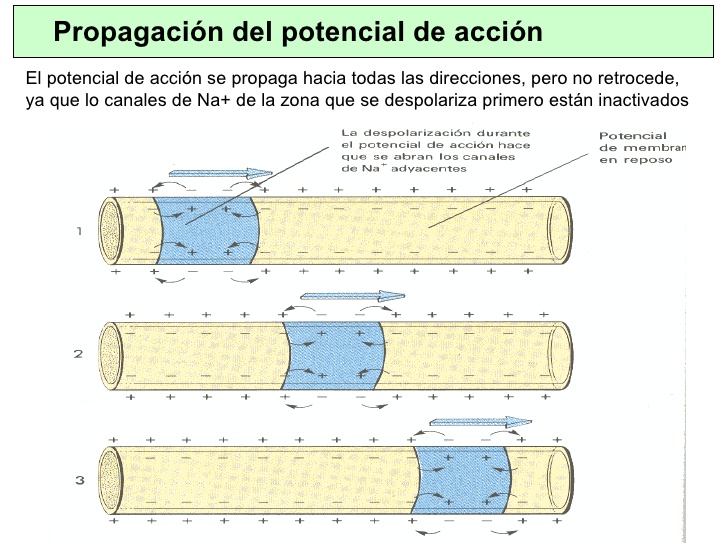
### Descripción

Algunos tipos de células son capaces de producir potenciales bioeléctricos debido a la actividad electroquímica de sus membranas como, por ejemplo, las nerviosas y musculares. Ya que cada tipo genera una actividad eléctrica diferente a las demás, el hecho de medir esta actividad proporciona información sobre su funcionamiento. Las disfunciones se pueden revelar en las señales bioeléctricas, por ello se puede obtener información a partir de estos registros.

Rangos de algunas de las bioseñales más usuales:

|  |  |  |  |
| --- | --- | --- | --- |
| Bioseñal | Definición | Rango amplitud | Rango frecuencia |
| Electrocardiograma  (ECG) | Actividad eléctrica cardíaca | 0.5 - 4 mV | 0.01-250 Hz |
| Electroencefalograma  (EEG) | Actividad eléctrica cerebral | 5 - 300 μV | DC-150 Hz |
| Ectrogastrograma  (EGG) | Actividad eléctrica gástrica | 10 µV – 1 mV | DC-1 Hz |
| Electromiograma  (EMG) | Actividad eléctrica muscular | 0.1 – 5 mV | DC-10 kHz |
| Electroneurograma  (ENG) | Actividad eléctrica nerviosa | 0.01 – 3 mV | DC-1 kHz |
| Electrooculograma  (EOG) | Potencial retina-córnea | 50 - 3500 µV | DC-50 Hz |
| Electrorretinograma  (ERG) | Actividad eléctrica de la  retina | 0 - 900 µV | DC-50 Hz |
| Fonocardiograma  (PCG) | Sonidos cardíacos | 80 dB (rango  dinámico)  100 µPa (umbral) | 5-2000 Hz |
| Flujo sanguíneo | Flujo sanguíneo | 1 – 300 ml/s | DC – 20 Hz |
| Gasto cardíaco | Cantidad de sangre  bombeada por el corazón en  unidad de tiempo | 4 – 25  litros/minuto | DC – 20 Hz |
| pH sanguíneo | Medida del pH en sangre | 6.8 – 7.8 unid. pH | DC – 2 Hz |
| Plestimografía | Medida de cambios de  volumen | Depende del  órgano medido | DC-30 Hz |
| Pneumotacografía | Medida del flujo respiratorio | 0-600  litros/minuto | DC-40 Hz |
| Potencial acción | Potencial característico de  diferentes tipos de células | 100 mV | 2 kHz |
| Potenciales evocados  (EP) | Respuestas cerebrales  evocadas por estímulos  sensoriales | 0.1 - 10 µV | 0.5 – 3 kHz |
| Presión arterial | Presión sanguínea arterial | 10 – 400 mm Hg | DC – 50 Hz |
| Presión venosa | Presión sanguínea venosa | 0 – 50 mm Hg | DC – 50 Hz |
| Respuesta galvánica  de la piel (GSR) | Potencial generado por la  actividad de las glándulas  sudoríparas de la mano | 1 – 500 kΩ | 1 -1 Hz |

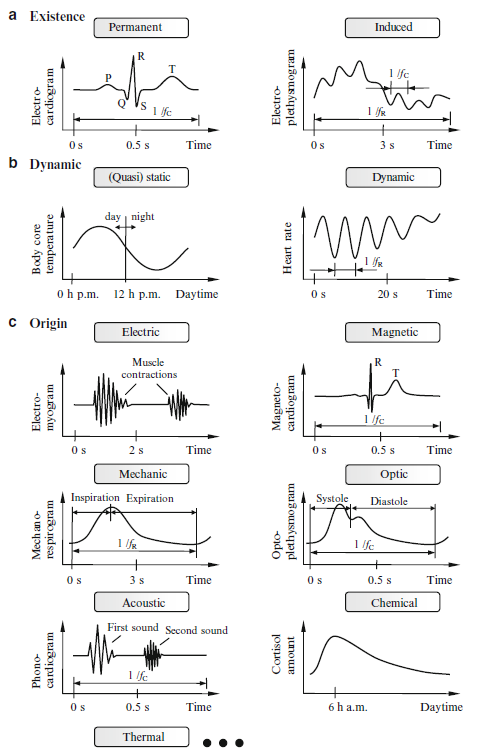




### Naturaleza

### Señales fisiológicas

### Tipos y clasificación

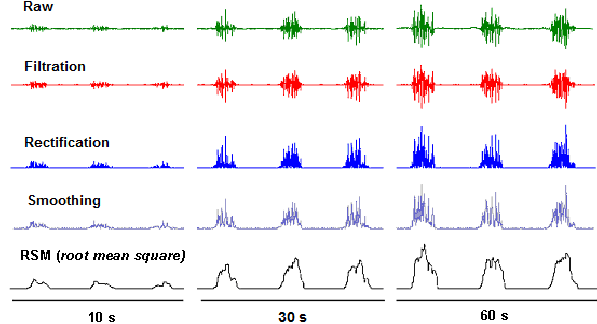


## **ELECTROMIOGRAFÍA (EMG)**

### Descripción

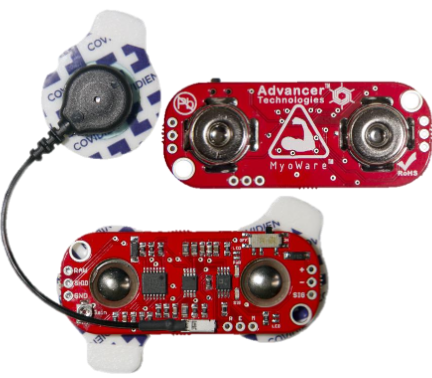
### Señales

### Digitalización



### Tipos de sensores comerciales

A continuación, se exponen varios de los múltiples sistemas para la adquisición de señales electromiográficas. Se trata de realiza un estudio de que existe actualmente en el mercado, para poder decidir cuál de ellos se ajusta mejor a las necesidades de este proyecto.

MyoWare (SparkFun)

Se trata de un sistema embebido comercial, el cual integra todos los componentes necesarios para la adquisición y el tratamiento de la señal.

El tamaño es reducido y dispone dos modos de obtención de señal: sin procesar, y rectificada e integrada.

El último modo es el más interesante para la integración con otros sistemas embebidos.

La ganancia es ajustable a través de un potenciómetro.

*Precio:* ~ 45€

*Ventajas*

* tamaño reducido
* circuito comercial probado

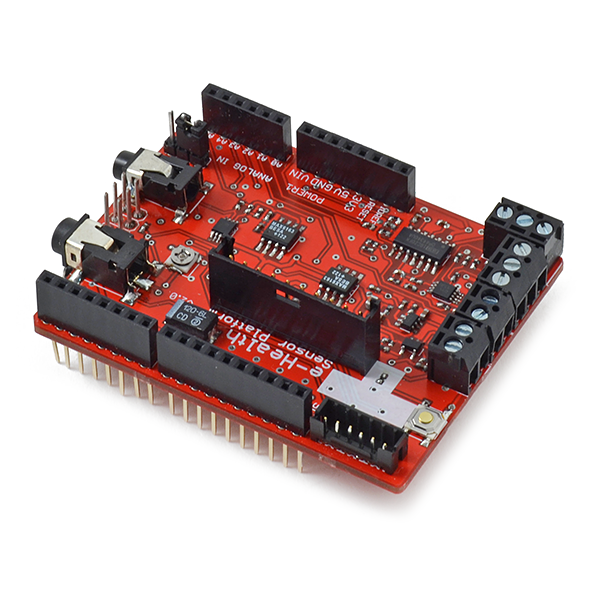
*Desventajas*

* Electrodos muy juntos (cables con electrodos son un extra)

MOVER A REFERENCIAS

<https://www.sparkfun.com/products/13723>

<http://cdn.sparkfun.com/datasheets/Sensors/Biometric/MyowareUserManualAT-04-001.pdf>

Sensor electromiográfico para e-Health Platform (Cooking Hacks)

Se trata de un sistema embebido comercial de muy altas prestaciones. Está diseñado para obtener todo tipo de señales del cuerpo humano. Se pueden adquirir extensiones para conectividad inalámbrica.

*Precio:* 240€ + 48€ (sensores)

*Ventajas*

* Sistema muy completo

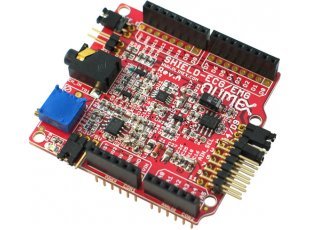
*Desventajas*

* Sistema completo muy caro
* Prototipo muy grande
* Demasiados sensores para lo que se pretende en este proyecto

MOVER A REFERENCIAS

<https://www.cooking-hacks.com/ehealth-sensor-shield-biometric-medical-arduino-raspberry-pi>

<https://www.cooking-hacks.com/electromyography-sensor-emg>

Olimex Shield-EKG-EMG

Está pensado para trabajar como shield para Arduino, pero puede ser utilizado por cualquier otro sistema embebido. La señal de salida de este sistema se puede conectar directamente a un microcontrolar o FPGA, ya que esta está tratada.

*Precio:* 20€ (shield) + 10€ (cables) + 1€xElectrodos

*Ventajas*

* Precio razonable
* Circuito comercial probado

*Desventajas*

* Prototipo muy grande

MOVER A REFERENCIAS

<https://www.olimex.com/Products/Duino/Shields/SHIELD-EKG-EMG/>

<https://www.olimex.com/Products/Duino/Shields/SHIELD-EKG-EMG/resources/SHIELD-EKG-EMG.pdf>

Conclusiones

A continuación, se presenta una tabla comparativa a modo de resumen.

|  |  |  |
| --- | --- | --- |
| **Sistema** | **Ventajas** | **Desventajas** |
| *Circuito propio* | Barato.  Personalizable (hecho a medida). | Prototipo grande.  Tiempo necesario para su diseño e implementación. |
| *MyoWare* | Tamaño reducido.  Circuito comercial probado. | Electrodos muy juntos (cables con electrodos son un extra). |
| *e-Health Platform* | Sistema muy completo.  Circuito comercial probado. | Sistema completo muy caro.  Prototipo muy grande.  Demasiados sensores para lo que se pretende en este proyecto. |
| *Olimex Shield-EKG-EMG* | Precio razonable.  Circuito comercial probado. | Prototipo muy grande. |

* El sistema e-Health queda descartado ya que es muy caro, y aunque es un sistema muy completo, las prestaciones que ofrece son demasiadas para lo que se pretende en este proyecto.
* Diseñar y construir un circuito propio es una opción válida (ver capítulo X.X). El mayor de los problemas es el tiempo necesario para su diseño, implementación y testeo. También hay que tener en cuenta que, al tratarse de un prototipo, se ha de implementar en una protoboard y en una placa de prototipado con soldaduras, por lo que su tamaño es será grande y muchos cables quedarán a la vista, los cuales se pueden mover y desconectar con mucha facilidad.
* Olimex Shield-EKG-EMG y MyoWare son dos sistemas que, dadas sus prestaciones, encajan en el desarrollo de este proyecto. Los dos sistemas nos aportan una señal que se puede conectar directamente a la placa DE0-Nano. El tamaño reducido hace que MyoWare sea más práctico, y al tratar de implementar un sistema sin inalámbrico y portable, hacen que el sensor que vaya a implementarse sea este último.

## **FPGA**

Este apartado tiene como objetivo la realización de una pequeña introducción en el mundo de las FPGAs. No se pretende dar una explicación en profundidad, el propósito situar al lector en el contexto del proyecto.

<http://www.fpgacenter.com/fpga/index.php>

http://www.fpga4fun.com/FPGAinfo1.html

### Introducción

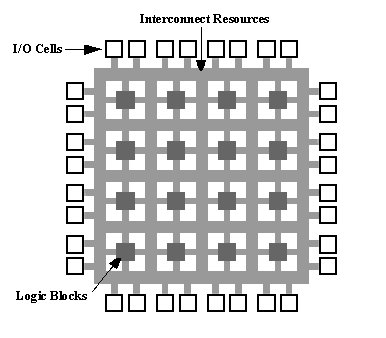
Las FPGAs (*Field Programmable Gate Array*) es un dispositivo digital que puede ser configurado para realizar cualquier tipo de tarea. Contiene bloques lógicos, cuya interconexión y funcionalidad es configurable. Se puede decir que es un circuito integrado, que a diferencia de otros con funciones ya establecidas y no fijas, puede ser reprograma y re configurado para la funcionalidad que se desee. Las FPGAs pueden integrar desde tareas muy simples (por ejmplo, un contador, o un simple bloque lógico) hasta funciones tan complejas como un microprocesador.

Una de las características más importantes de las FPGAs es la habilidad para realizar procesado en paralelo. Esto significa que puede realizar múltiples transacciones al mismo tiempo. El hecho de poder realizar tareas en paralelo, hace que estos dispositivos sean capaces de implementar funciones críticas en tiempo.

Los propios fabricantes de FPGAs distribuyen funciones integradas que han sido optimizadas para una familia específica de FPGAs. Estas funciones son conocidas como IP (*Intellectual Property*). Funciones más avanzadas y complejas pueden requerir de un pago para su uso.

### Arquitectura

Una FPGA consiste básicamente en celdas lógicas, bloques de E/S, interconexiones y líneas de reloj. Todas las FPGAs actuales también implementan recursos adicionales como, por ejemplo, bloques RAM. Cada fabricante tiene su propia arquitectura específica, pero en general, todos los diseños se basan en los componentes previamente mencionados.



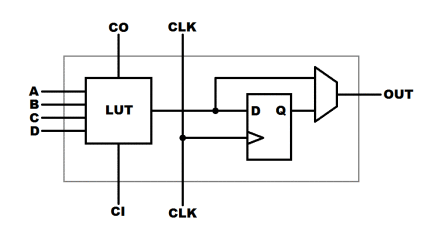
<https://en.wikipedia.org>

Celdas lógicas

Forman la principal estructura de las FPGAs. Su arquitectura básica consiste en una LUT (*Lookup Table)*, un Flip Flop y un multiplexor 2 a 1.

El LUT es un pequeño bloque de memoria que implementa cualquier función lógica. La salida de LUT de conecta con el flip flop para mantener el estado de manera síncrona. El multiplexor sirve para puentear el flip flop en caso de ser necesario.

Adicionalmente, las celdas lógicas pueden tener líneas extra para la conexión con sus celdas vecinas. Estas líneas, la cuales son muy rápidas, se denominan “*carry in”* y *“carry out”, y* permiten crear funciones aritméticas (como contadores y sumadores) de manera eficiente.



Bloques E/S

Los pines de las FPGA se pueden dividir en dos categorías:

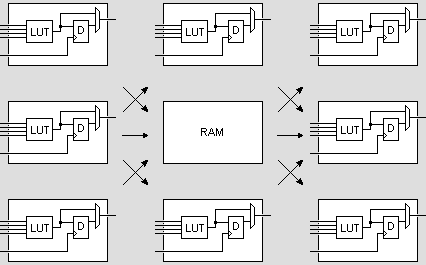
* Pines de usuario: los pines estándar de E/S que pueden ser configurados por el usuario. Pueden ser configurados para trabajar como entrada, salida o entrada y salida. Cada pin está conectado a un bloque E/S dentro de la FPGA. Estos bloques están alimentados a través de los pines de alimentación (VCCIO).
* Pines dedicados: estos pines no son configurables y tienen una función específica. Pueden ser pines de alimentación, pines de configuración o pines para reloj.

Interconexiones

Las interconexiones entre bloques de la FPGA son programables y configurables. Esto dota a las FPGA de mucha flexibilidad, ya que estas conexiones han de ser entre dispositivos cercanos y lejanos. Por lo general, es el software el que realiza estás tareas, pero también es posible añadir condiciones para las conexiones.

Bloques RAM

Los bloques RAM están distribuidos en la FPGA y controlados por los elementos lógicos. Son usados para necesidades de almacenaje temporal durante la operación de circuitos lógicos. Estos bloques pueden soportar acceso múltiple o simple.



Gracias al acceso múltiple, varias aplicaciones pueden leer/escribir operaciones en la RAM. Ésta es una buena solución para la transferencia de datos entre bloques que tenga diferentes señales de reloj.

Líneas globales y de reloj

Las FPGA generalmente son utilizadas para diseños síncronos. Estos diseños se basan en una señal de reloj que sincroniza la señal del flip flop, por lo que el reloj debe dispara todos los flips flops del sistema al mismo tiempo, si no, aparecen problemas eléctricos y problemas de tiempos.

Para evitar estos problemas, se utilizan conexiones internas especiales llamadas *“líneas globales”*. Gracias a estas conexiones, se proporciona un acceso simultáneo de la señal de reloj a toso los flip flops de la FPGA.

### Tecnologías

Las FPGAs pueden ser programadas con tecnología diferentes. Es decisión de fabricante decidir qué método se implementará en cada uno de sus dispositivos.

El método de programación tiene un impacto en el rendimiento de la FPGA, por lo que conociendo en qué consiste cada una de ellas, se puede tomar una decisión más acertada acerca de la FPGA que se ajuste a las necesidades del diseño de un sistema.

A continuación, se explica brevemente las tecnologías más comunes.

http://www.oldcitypublishing.com/FullText/JAPEDfulltext/JAPED5.1-2fulltext/JAPEDv5n1-2p129-143Abbasi.pdf

SRAM

Es la tecnología más utiliza hoy en día por los fabricantes de FPGAs. Utiliza bloques de memoria SRAM para guardar la información de configuración de los bloques lógicos de la FPGA, así como sus interconexiones.

Esta tecnología permite reprogramar el dispositivo tantas veces como sea necesario, y además esta velocidad es muy rápida. Cabe destacar también, el costo de fabricación es menor que otras tecnologías.

El problema de esta tecnología radica en ser volátil, por lo que cuando se deje de alimentar el dispositivo, perderá toda la información que contiene. Otro problema es que necesita una mayor área para ser implantado, por lo que el dispositivo será más grande.

Antifuse

Cuando el dispositivo no está programado, el antifusible tiene una resistencia muy alta, por lo que se considera circuito abierto. La programación se realiza estableciendo una tensión elevada entre dos líneas que se cruzan. Esta tensión es superior a la rigidez dieléctrica del óxido y el campo eléctrico lo rompe. Entonces salta un pequeño arco entre las dos pistas, que las funde parcialmente, soldándolas, estableciendo así una conexión permanente.

Utilizando esta tecnología los dispositivos pueden ser programados una única vez.

https://es.wikipedia.org/wiki/Antifusible

FLASH

Esta tecnología se basa en memorias no volátiles, pero a su vez permite la reprogramación. Utiliza transistores MOS, que trabajan por parejas para almacenar el estado de cada bit.

Con esta tecnología se pueden volver a reprogramar el dispositivo, pero tiene un límite. Se necesita una tensión alta para poder reprogramarlos, y no permite la reconfiguración parcial. Cabe destacar, que la fabricación con esta tecnología es compleja.

A continuación, se muestra una tabla a modo de resumen

|  |  |  |  |
| --- | --- | --- | --- |
|  | SRAM | Antifuse | FLASH |
| Reprogramable | Sí | No | Sí |
| Velocidad de reprogramación | Rápido | --- | 3 veces más lento que SRAM |
| Volátil | Sí | No | No |
| Necesita fichero de configuración externo | Sí | No | No |
| Bueno para prototipado | Sí | No | Sí |
| Tamaño de celda de configuración | Grande  (6 transistores) | Muy pequeño | Medio-Pequeño  (2 transistores) |
| Consumo | Medio | Bajo | Medio |
| Tecnología | CMOS | CMOS+ | EECMOS |
| Asociadas a | FPGAs | FPGAs | SPLDs y CPLDs |

<https://www.dc.uba.ar/materias/disfpga/2012/c1/descargas/Introduccion.pdf>

Debido a que en este proyecto se ha realizar un prototipo, y debido a los muchos cambios que se han de realizar durante el proceso de desarrollo, se utilizará un FPGA con tecnología SRAM.

### Fabricantes

En la actualizan existen dos compañías que tiene casi la totalidad del mercado de las FPGAs. Estas son Xilinx y Altera. Existen muchas otras como Lattice Semiconductor, Actel o QuickLogic, pero en este apartado se hará una pequeña presentación de las dos más importantes.

* Xilinx: es la compañía que intentó las FPGAs. Es la compañía líder en el mercado de las FPGAs. Tienen una buena relación calidad-precio. La serie Spartan cubre el mercado de baja y media gana, mientras que Virtez cubre la alta gama. Cuenta con un software para diseño y síntesis llamado ISE.
* Altera: la gran competidora de Xilinx. Sus familias Cyclone, Arria y Stratix cubren el mercado de baja, media y alta gama respectivamente. Cuenta con un software para diseño y síntesis llamado Quartus II.

## **COMUNICACIONES**

En este proyecto, el envío y recepción de datos entre el PC y el sistema embebido será de manera inalámbrica. Para ello, se ha de decidir el protocolo que se usará para esta comunicación.

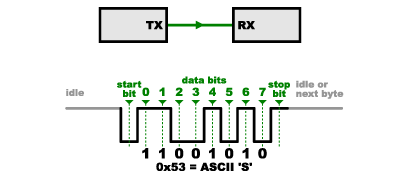
La comunicación inalámbrica es aquella que prescinde de cables y propaga la información a través del espacio, basándose en la modulación de ondas electromagnéticas.

### Comunicación serie

La comunicación serie consiste en el envío de datos bit a bit, de forma secuencial, a través de un canal de comunicación o un bus. La comunicación en paralelo en cambio, consiste en enviar todos los bits al mismo tiempo. La ventaja de usar comunicación serie es que son necesarias un número menor de líneas que en la comunicación en paralele, en la cual hacen falta tantas líneas de comunicación como bits a enviar. Otro problema de la comunicación paralelo es que, debido al gran número de bit a enviar al mismo tiempo, pueden surgir problemas de desincronización. La ventaja de la comunicación paralelo es clara, al poder enviar más bits a la vez, la comunicación es más rápida.

La comunicación serie a su vez puede ser síncrona o asíncrona. La comunicación asíncrona necesita un solo hilo, ya que cada sistema tiene su propio reloj. Su sincronización se realiza mediante bits de start y stop. La comunicación asíncrona, por el contrario, cuenta con una señal de reloj común, la cual es generada por el *maestro*. Debido a que la señal de reloj se ha de compartir, este sistema necesita de dos hilos. Cabe destacar, que los sistemas síncronos permiten trabajar a velocidades mayores.





Otra clasificación para las comunicaciones series es la dirección en la que se realiza la comunicación, pudiendo esta esta bidireccional simultánea (Full-duplex) o bidireccional multiplexada (Half-duplex). Si se trata de Full-duplex, la comunicación en las dos direcciones es simultánea, por lo que son necesarios dos hilos (TX y RX). En el caso de Half-duplex, la comunicación no puede realizarse en ambas direcciones simultáneamente, por lo que existe un solo hilo (único canal), y se ha de establecer un protocolo (primero en un sentido, y luego en el otro).

Cuando se habla de comunicaciones serie en sistemas embebidos, existen tres protocolos que son los más extendidos para aplicaciones que necesiten comunicaciones a corta distancia. Estos son UART, SPI e I2C.

http://www.comm.pub.ro/dicm/C7\_Serial\_Bus.pdf

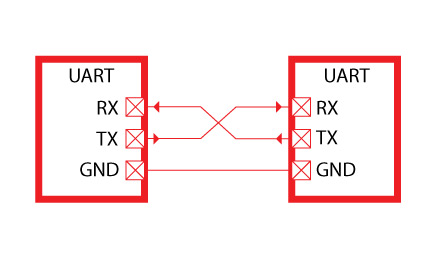
<https://www.swarthmore.edu/NatSci/echeeve1/Class/e91/Lectures/E91(10)Serial.pdf>

<http://www.quilix.com/sites/default/files/comparison_between_SPI_I2C_UART.pdf>

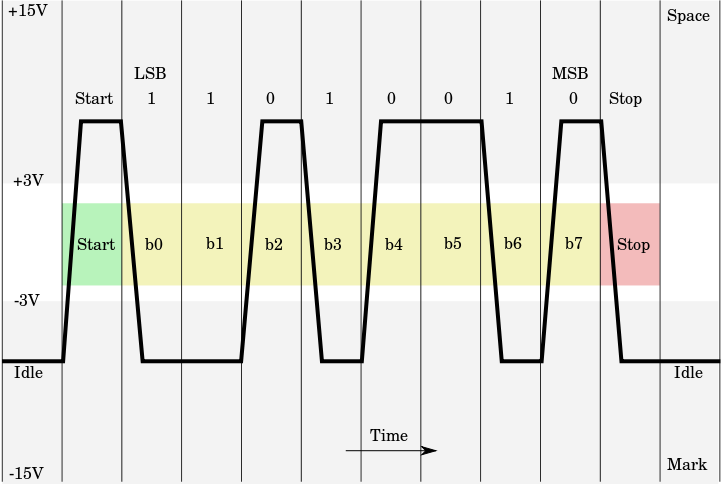
http://www.rfwireless-world.com/Terminology/UART-vs-SPI-vs-I2C.html

UART (*Universal Asynchronous Receiver-Transmitter*)

Es uno de los protocolos más utilizados, debido a que la gran mayoría de microcontroladores disponen de hardware dedicado para UART. Consiste en dos hilos, uno para enviar (TX) y otro para recibir (RX). Si la comunicación se realiza en un solo sentido, se puede omitir uno de los dos hilos. LA comunicación es uno a uno, es decir, sólo dos sistemas se pueden comunicar.



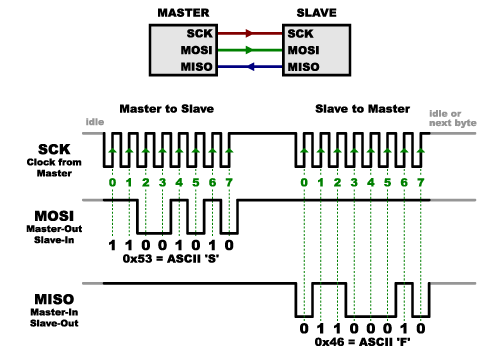
Generalmente se envían tramas de un byte (8 bit). El formato de la transmisión es el siguiente: bit de start, 8 bits de datos y bit de stop. Se puede añadir un bit para controlar la paridad a nivel hardware (parity bit).



Se trata de un sistema Full-duplex y asíncrono. La frecuencia de transmisión de bit (*baud rate*) ha de ser establecida de antemano. La velocidad máxima es de 230Kbps a 460Kbps.

SPI (*Serial Peripheral Interface*)

Es un tipo de comunicación Full-duplex y síncrona. El maestro envía la señal de reloj al esclavo a través de la línea SCK. Cuenta con dos hilos para la comunicación, que son MOSI (*Master Out Slave In*) y MISO (*Master In Slave Out*). Este porotocolo permite la comunicación con más de un sistema. Para ellos se añade una línea llamada SS (*Slave Select*) por cada dispositivo conectado, mediante la cual se dice a qué dispositivo se quiere enviar el mensaje. Los bits son enviados en cada pulso de reloj.

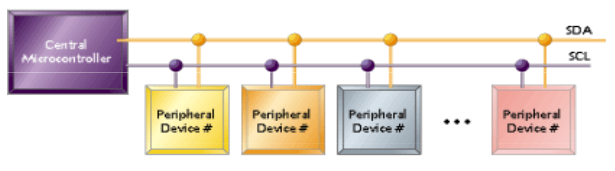


Permite la comunicación a velocidades de hasta 20Mbps.

I2C (*Inter Integrated Circuit*)

I2C es un protocolo síncrono y Full-duplex. Consiste en dos hilos, uno de datos (SDA) y otro de reloj (SCL). Es un protocolo que permite la conexión de múltiples esclavos sin añadir ninguna línea extra.

http://www.comm.pub.ro/dicm/C7\_Serial\_Bus.pdf



El protocolo funciona de la siguiente manera:

1. El maestro envía la condición de start (S).
2. El maestro encía la dirección del esclavo (7 bits).
3. El maestro envía el bit para determinar si se procede a una lectura o a una escritura.
4. El transmisor (esclavo o maestro) envía un bit ACK.
5. El transmisor envía 1 byte de dato.
6. El receptor envía un bit ACK por cada byte recibido.
7. Para escrituras, el maestro (transmisor) envía un stopbit (P) tras el último byte de dato. Para lecturas el maestro (receptor) no envía ACK tras el último byte, tan sólo envía la señal de stop (P) para comunicar al esclavo que la transmisión ha terminado.



Permite la comunicación a velocidades de hasta 3,4Mbps.

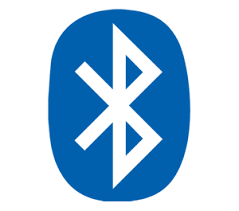
Conclusión

Para este proyecto se ha decidido utilizar tecnología Bluetooth, ya que es la más sencilla de implementar, teniendo en cuenta que existen módulos comerciales (ver capítulo X.X) que reciben datos a través de protocolo serie (UART), y los envían a través de Bluetooth. En este caso el sistema embebido será el encargado de enviar datos al módulo Bluetooth e través de UART, y éste a su vez será quien renvíe la información al PC a través de Bluetooth.

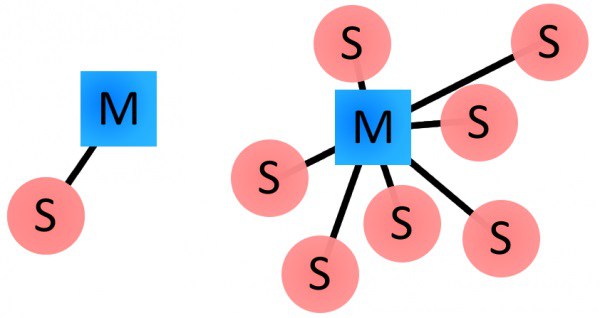
### Bluetooth

<https://www.bluetooth.com>

https://learn.sparkfun.com/tutorials/bluetooth-basics

Bluetooth es un estándar de tecnología inalámbrica para el intercambio de datos a través de distancias cortas procedentes de dispositivos electrónicos móviles y fijos y PANs (*Personal Area Networks*). Esta tecnología utiliza un enlace por radiofrecuencia en la banda ISM de 2.4 a 2.485GHz. Es un protocolo seguro y encaja muy bien en aplicaciones de corto alcance, bajo consumo y bajo coste.

Los dispositivos Bluetooth pueden trabajar como maestros o esclavos. Un esclavo sólo puede conectarse a un maestro, sin embargo, un maestro puede conectarse a varios esclavos (hasta 7), o permitir que estos se conecten.



Clases

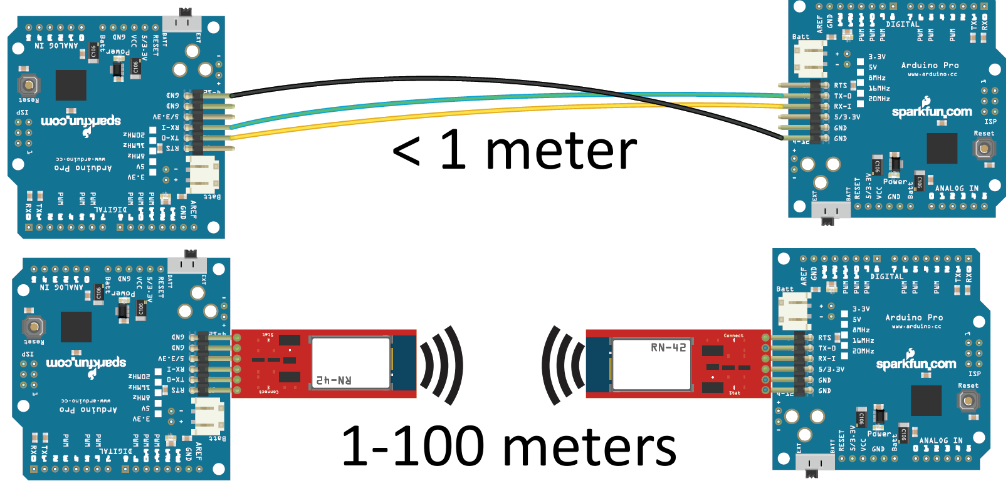
Todos los dispositivos con Bluetooth tienen la capacidad de comunicarse entre sí siempre que estén al alance unos de otros. Los dispositivos no tienen que encontrarse en una orientación determinada, ni siquiera han de estar en la misma habitación para poder comunicarse. Los dispositivos se clasifican en función de su potencia de transmisión (alcance). Cabe destacar, que los dispositivos de una clase pueden comunicarse con los de otra clase distinta, siempre y cuando estén en el rango de alcance.

|  |  |  |  |
| --- | --- | --- | --- |
| Clase | Potencia max. permitida (mW) | Potencia max. permitida (dBm) | Alcance (m) |
| 1 | 100 | 20 | 100 |
| 2 | 2.5 | 4 | 10 |
| 3 | 1 | 0 | 1 |
| 4 | 0.5 | -3 | 0.5 |

Perfiles Bluetooth

Los perfiles Bluetooth son protocolos adicionales, basados en el Bluetooth básico, que definen mejor qué tipo de dato se transmite. Mientras las especificaciones Bluetooth definen cómo funciona la tecnología, los perfiles definen cómo se usa. Para que dos dispositivos Bluetooth sean compatibles, tienen que soportar los mismos perfiles. A continuación, se enumeran algunos ejemplos.

* *SPP (Serial Port Profile)*: este perfil sustituye una comunicación serie con cables (tipo RS-232 o UART). Con este perfil, cada dispositivo conectado es capaz de enviar y recibir datos como si dos líneas RX y TX estuvieran conectados entre ellos.



* *HID (Human Interface Device)*: este perfil es el utilizado para conectar dispositivos que actúan como interfaces de entrada de usuario (teclados, ratones, micrófonos, etc.).
* *HFP y HSP*: *Hands-Free Profile* y  *Headset Profile.* Este perfil es usado en los sistemas de audio manos libres.

Versiones

Esta tecnología fue concebida en 1994, y desde entonces ha estado en constante evolución. La última versión de Bluetooth es la 5.0 (2016-2017), pero todavía son muy utilizadas versiones anteriores.

|  |  |
| --- | --- |
| Versión | Ancho de banda (Mbits/s) |
| 1.2 | 1 |
| 2.0 +EDR | 3 |
| 3.0 + HS | 24 |
| 4.0 | 32 |
| 5.0 | 50 |

Comparación inalámbrica

Bluetooth es uno de los muchos protocolos inalámbricos que existen. A continuación, se muestra una tabla comparativa con algunos de estos protocolos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre | Bluetooth Classic | Bluetooth 4.0 Low Energy (BLE) | ZigBee | WiFi |
| Estándar IEEE | 802.15.1 | 802.15.1 | 802.15.4 | 802.11 (a, b, g, n) |
| Frecuencia (GHz) | 2.4 | 2.4 | 0.868, 0.915, 2.4 | 2.4, 5 |
| Velocidad max. de bit sin procesar (Mbps) | 1-3 | 1 | 0.250 | 11 (b), 54 (g), 600 (n) |
| Rendimiento típico de dato (Mbps) | 0.7-2.1 | 0.27 | 0.2 | 7 (b), 25 (g), 150 (n) |
| Alcance máximo (m) | 10 (clase 2), 100 (clase 1) | 50 | 10-100 | 100-250 |
| Consumo relativo | Medio | Muy bajo | Muy bajo | Alto |
| Ejemplo de vida de batería | Días | Meses-Años | Meses-Años | Horas |
| Tamaño de red | 7 | Indefinido | 64,000+ | 255 |

Bluetooth en el proyecto

En este proyecto se hará uso de un módulo Bluetooth con las siguientes características (ver capítulo X.X):

* Bluetooth 2.0
* Clase 2
* Perfil SSP

# DESCRIPCIÓN DEL DISPOSITIVO

## **DE0 NANO**

En este apartado se explicará brevemente la plataforma DE0-Nano de *Terasic*. Se hará mención a los diferentes dispositivos que lo componen, y se tratará de explicar los dispositivos que son utilizados en este proyecto.

DE0 Nano User Manual v1.9

### Descripción general

DE0-Nano es una plataforma de desarrollo FPGA de tamaño reducido, la cual encaja perfectamente en proyectos que necesiten un diseño compacto y portable.

Esta placa cuenta con una FPGA Cyclone IV de Altera (con 22,230 elementos lógicos), 32MB de SDRAM, 2Kb EEPROM, y un dispositivo de memoria de 64Mb configurable. Para conectarse con el mundo real incluye un convertidor A/D de 12-bit y 8 canales de National Semiconductor, y también cuenta con un acelerómetro de 13-bit y 3 ejes de Analog Devices.

DE0-Nano incluye un USB Blaster integrado para la programación de la FPGA, y la placa puede ser alimentada a través de tu puerto USB o a través de una fuente externa. La plataforma incluye pines de expansión, que pueden ser usador para conectar otros dispositivos, como sensores o motores. Las salidas y entradas incluyen 2 pulsadores, 8 LEDs y 4 dip-switches.

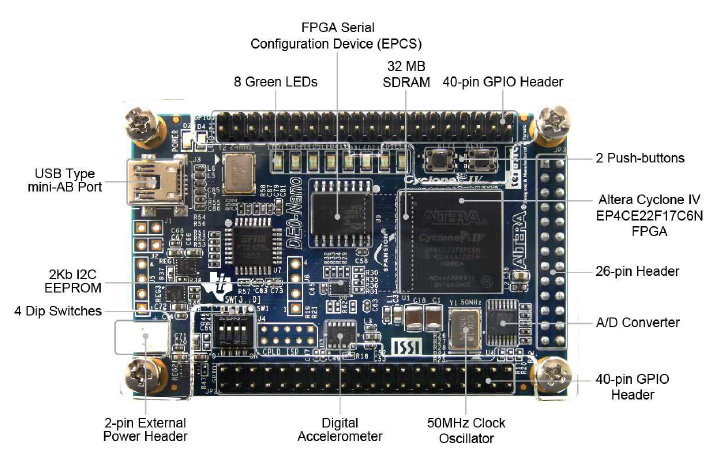
La tensión de trabajo de esta plataforma es de 3.3V

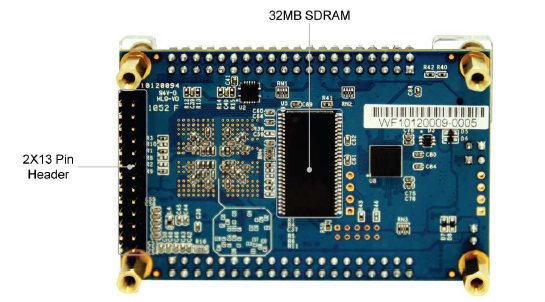
A continuación de muestra una lista de los dispositivos integrados en esta plataforma:

* Dispositivo destacado
  + Altera Cyclone® IV EP4CE22F17C6N FPGA
  + 153 maximum FPGA I/O pins
* Elementos de configuración
  + Circuito para programación USB-Blaster integrado
  + Spansion EPCS64
* Pines de expansión
  + Dos headers de 40-pines (GPIOs) proporcionan 72 pines de E/S, pines de alimentación de 5V, dos pines de alimentación 3.3V y cuatro pines de tierra.
* Dispositivos de memoria
  + 32MB SDRAM
  + 2Kb I2C EEPROM
* Entradas y salidas generales de usuario
  + 8 LEDs verdes
  + 2 pulsadores con anti-rebotes
  + DIP switch de 4 posiciones
* G-Sensor
  + ADI ADXL345, acelerómetro de 3 ejes con alta resolución (13-bit)
* Conversor A/D
  + NS ADC128S022, 8 canales, resolución 12-bit
* Sistema de reloj
  + Reloj oscilador de 50 MHz integrado
* Alimentación
  + Puerto USB tipo mini-AB (5V)
  + Pin de 5V DC por cada header GPIO (2 pines 5V DC)
  + Alimentación externa a través de conector de dos pines (3.6-5.7V)

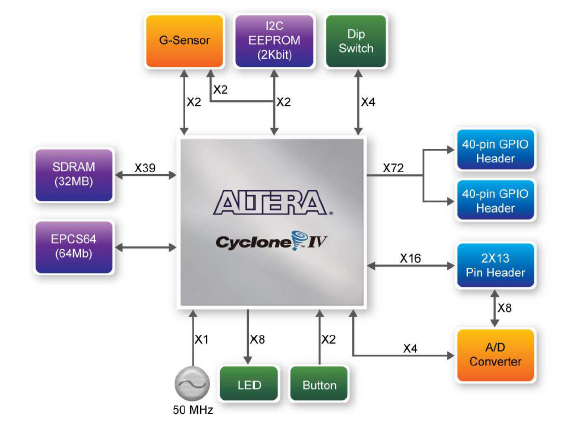
### Arquitectura

Las fotos de la placa DE-Nano se muestran en la figura X.X y la figura X.X. En ellas se representa el layout de la placa y se indica la localización de todos sus componentes, así como la de los conectores.





La figura X.X muestra el diagrama de bloques de la placa. Para proveer máxima flexibilidad para el usuario, todas las conexiones se realizan a través de la FPGA Cyclone IV. De este modo, el usuario puede configurar la FPGA para implementar cualquier diseño.



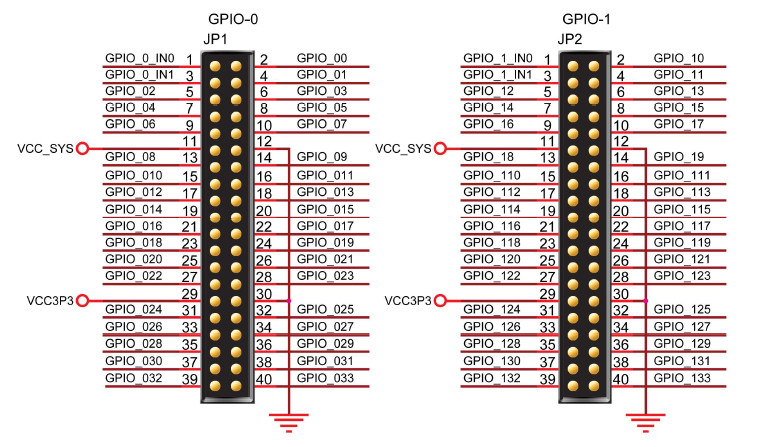
Debido a que la FPGA tiene memoria volátil, es necesario un EEPROM de arranque que permita volcar el programa, previamente cargado, en la FPGA. Para esto, DE=-Nano cuenta con EPCS64, el cual almacena el programa en su memoria interna no volátil para cárgalo en la FPGA cada vez que se alimenta el sistema.



### Conexiones

Conectores de expansión de 40 pines

A continuación, se muestran las conexiones hacia el exterior que ofrece la placa DE0-Nano a través de los conectores de 40 pines.

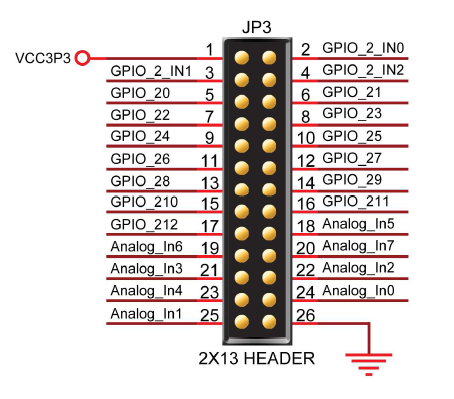


En la imagen se pueden apreciar 4 tipos diferentes de pines:

* GPIO\_X: pines de E/S de propósito general. Cada uno de estos pines está conectado directamente con pines de E/S de la FPGA
* VCC\_SYS: pines de alimentación de 5V DC.
* VCC3P3: pines de alimentación de 3.3V DC
* GND: pines de conexiones a tierra.

Convertidos A/D y conectores de expansión 2x13

La siguiente figura muestra la relación de pines del conector 2x13, el cual sirve de entrada para las señales analógicas.



Los tipos de pines mostrados en la figura son:

* GPIO\_2\_INx: pines de entrada a la FPGA. Cada uno de estos pines está conectado directamente a la FPGA.
* GPIO\_X: pines de E/S de propósito general. Cada uno de estos pines está conectado directamente con pines de E/S de la FPGA
* VCC3P3: pines de alimentación de 3.3V DC
* Analog\_Inx: entradas analógicas. Estos pines están conectados directamente al ADC128S022.

A continuación, se muestra la conexión entre el conector de expansión 2x13, la FPGA y el conversor A/D:



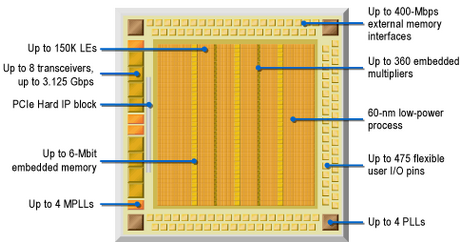
### Altera Cyclone IV EP4CE22

<https://www.altera.com/products/fpga/cyclone-series/cyclone/support.html>

https://www.altera.com/products/fpga/cyclone-series/cyclone-iv/overview.html

La familia Cyclone IV de altera es una línea de FPGAs que combina el bajo coste con el bajo consumo. Esta gama requiere sólo de dos alimentaciones para funcionar, lo cual simplifica la distribución de las líneas de alimentación y ahorra espacio en la placa. Están fabricados con una tecnología de 60-nm.

<https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyc/cyc_c51002.pdf>



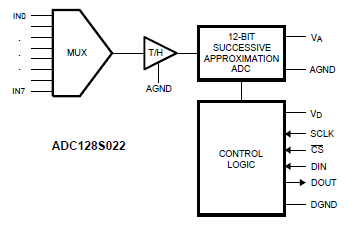
https://www.altera.com/content/dam/altera-www/global/en\_US/pdfs/literature/pt/cyclone-iv-product-table.pdf

|  |  |
| --- | --- |
| LEs (K) | 22 |
| M9K bloques de memoria | 66 |
| Memoria embebida (Kb) | 594 |
| Multiplicadores 18 x 18 | 66 |
| Líneas de reloj global | 20 |
| PLLs | 4 |
| Bancos de E/S de usuario | 8 |
| E/S de usuario máximas | 153 |
| Voltajes de E/S admitidos (V) | 1.2, 1.5, 1.8, 2.5, 3.3 |
| Canales LVDs emulados | 52 |

### ADC128S022

El ADC128S002 es un conversor analógico-digital de ocho canales de baja potencia, con resolución de 12 bits y tecnología CMOS, para rangos de conversiones de 50 ksps a 200 ksps. El conversor está basado en una arquitectura de registro de aproximaciones sucesivas con un circuito interno de *track-and-hold*. Puede ser configurado para aceptar hasta ocho señales de entrada.

La comunicación entre el ADC128S022 y el microcontrolador o FPGA se realiza a través de 4 líneas: señal de reloj, chip select, datos de entrada y datos de salida.



## **MYOWARE**

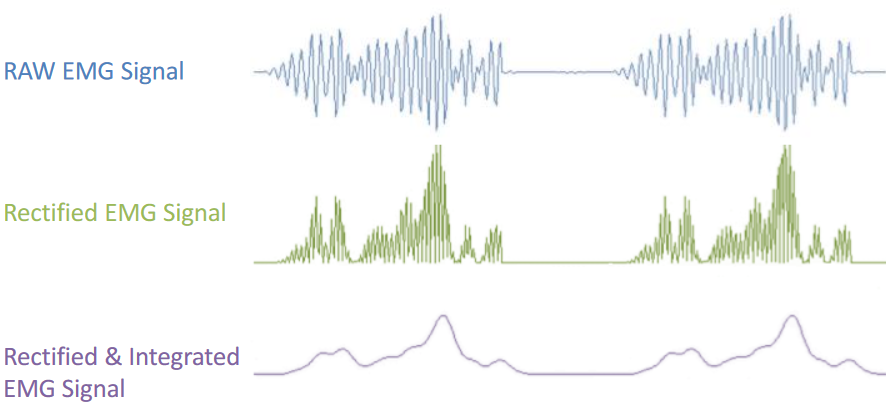
<https://github.com/AdvancerTechnologies/MyoWare_MuscleSensor/blob/master/Documents/AT-04-001.pdf>

MyoWare es un sensor de *Advancer Technolgies*, el cual monitoriza la actividad muscular, midiendo el potencial eléctrico generado por las fibras musculares, lo que se conoce como electromiografía (EMG).

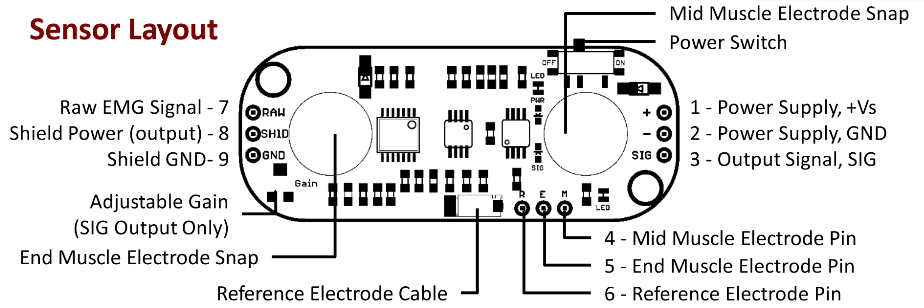
Debido a que las señales generadas por los músculos son muy débiles (ver capítulo X.X), estas han de ser amplificadas. MyoWare cuenta con un potenciómetro mediante el cual es posible ajustar la ganancia. La relación entre el valor del potenciómetro y la ganancia se define mediante la siguiente fórmula:

MyoWare ofrece la posibilidad de obtener las señales EMG en dos formatos diferente, ambas amplificadas a través de la ganancia.

* El primer método es obtener la señal sin tratar sin tratar. La señal sin tratar está centrada alrededor de un offset de +Vs/2. Esta señal no está rectificada ni integrada.
* El segundo método obtiene la señal tratada, es decir, rectificada e integrada (conocida como señal envolvente de EMG). Esta señal trabaja bien con los ADC de los microcontroladores.

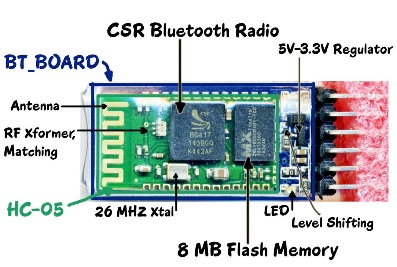


MyoWare incorpora en la propia placa conectores para electrodos, lo cual evita tener cables extra si el proyecto no lo requiere. El problema de esto, es que para la distancia entre los electrodos es fija, por lo que esta distancia puede no ser válida para todos los músculos. Para estos casos, MyoWare cuenta con conectores para conectar estos electrodos a través de cables.



## **HC-05 (MÓDULO BLUETOOTH)**

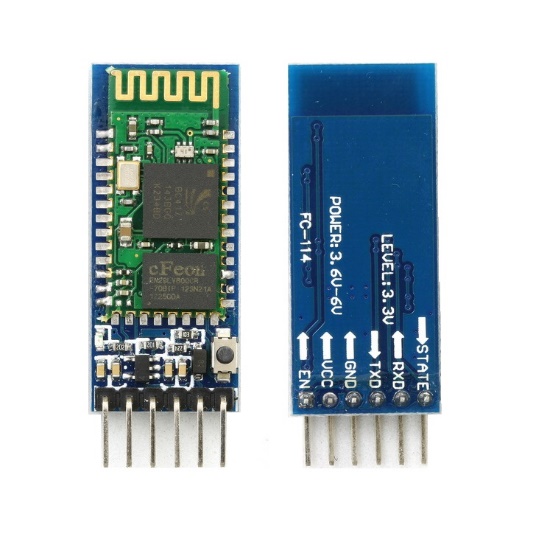
HC-05 es un módulo Bluetooth que proporciona una conexión inalámbrica fiable y sencilla de utilizar. Implementa el protocolo SPP (ver capítulo X.X) y funciona con Bluetooth v2.0+EDR. Incorpora un chip Bluetooth “*CSR Bluecore 04”* con tecnología CMOS y con AFH (*Adaptative Frequency Hopping*).



A diferencia de su hermano HC-06, este módulo puede trabajar como maestro o esclavo. La configuración del HC-05 se realiza mediante comandos Hayes o AT (ver anexo X.X). Este dispositivo incorpora un LED para indicar el estado de la conexión y el emparejamiento.

Hay que tener en cuenta, que, aunque la alimentación de módulo sean 5V, las líneas TX y RX trabajan a 3.3V.

A continuación, se explica brevemente la función de cada pin.

* EN: Si se pone a ‘1’ antes de alimentar el módulo, se fuerza el modo comandos AT.
* VCC: 3.6 – 6 V
* GND: Tierra
* TX: Línea de transmisión (HC-05 a dispositivo). Esta línea trabaja con 3.3V.
* RX: Línea de recepción (dispositivo a HC-05). Esta línea trabaja con 3.3V.
* STATE: a ‘1’ si está conectado.

<http://www.electronica60norte.com/mwfls/pdf/newBluetooth.pdf>

El HC-05 permite configurar muchos parámetros gracias a los comandos AT. Para poder configurar el módulo, es posible utilizar la línea ‘EN’ o, simplemente pulsando el pulsador que incorpora. Una vez el módulo entra en el modo configuración (modo AT), te han de enviar los comandos a través de protocolo UART, utilizando los pines TX y RX. Cada comando se envía con “AT+” como prefijo (ver anexo X.X). A continuación, se enumeran los comandos más básicos:

* AT, test de comunicación
* AT+VERSION, versión del Firmware
* AT+NAME, Programa el nombre que queremos presentar cuando alguien nos busque
* AT+UART, configura la comunicación serie
* AT+PSWD, cambiar código de vinculación
* AT+ROLE, nos informa de si está configurado como Maestro 1, o como esclavo 0.

Las principales características de este módulo son las siguientes: http://tienda.bricogeek.com/modulos-bluetooth/800-modulo-bluetooth-hc-05.html

* Protocolo Bluetooth: v2.0.
* Frecuencia: banda ISM de 2,4 GHz.
* Modulación: GFSK
* Potencia de transmisión: menos de 4dBm, Clase 2.
* Sensibilidad: Menos de -84dBm en el 0,1% BER.
* Ratio asíncronos: 2.1Mbps (Max) / 160 kbps.
* Síncrono: 1Mbps / 1Mbps.
* Perfil: SPP.
* Fuente de alimentación: + 3.3VDC 50mA. (soporta de 3.3 a 6V)
* Temperatura de trabajo: -5 ° C a 45 ° C.

# HERRAMIENTAS DE DESARROLLO

En este capítulo se realizará una breve descripción de las herramientas de desarrollo empleadas para la realización de este proyecto.

Todas las herramientas utilizadas en este proyecto son muy extensas, pero en este apartado sólo se hará referencia a las funcionalidades más significativas que ofrece cada herramienta y que haya sido utilizada para la realización.

## **DESARROLLO FPGA**

El flujo de diseño de una FPGA sigue el siguiente esquema:



El primer paso en el diseño de una FPGA comienza con diseño a través de esquemas o a través de lenguajes de descripción hardware (HDL), como Verilog o VHDL. Este paso es donde el diseñador crea el circuito digital que será implementado dentro de la FPGA.

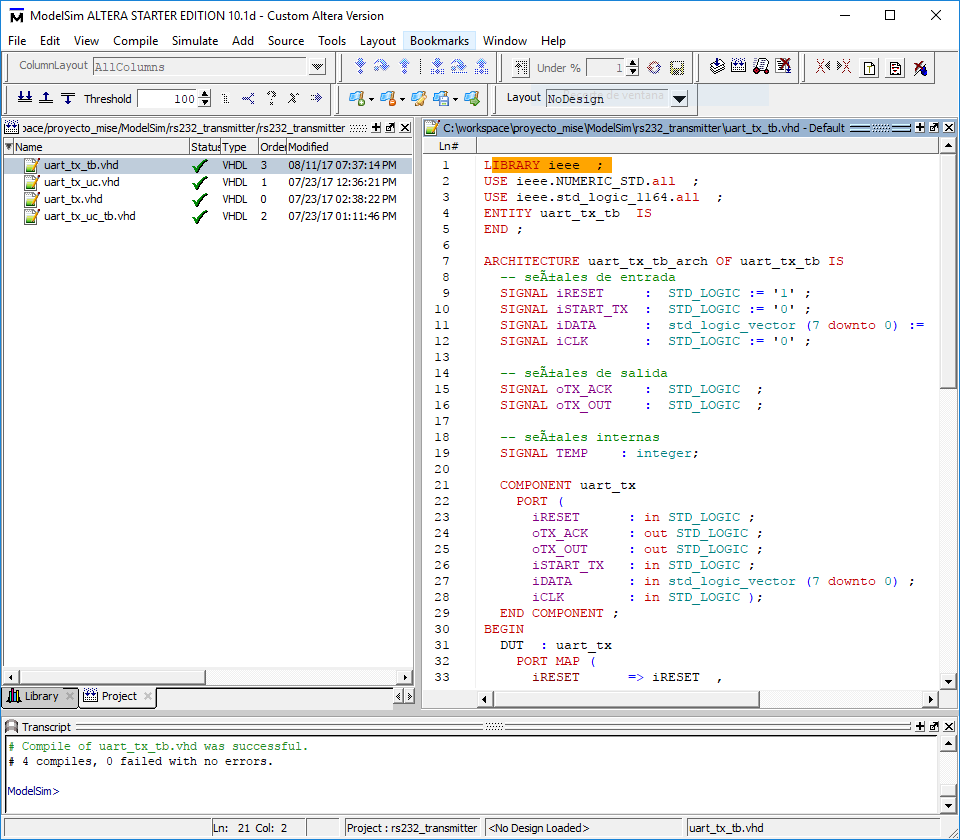
Una vez diseñado el circuito, se ha de compilar y programar en la FPGA para comprobar su correcto funcionamiento. Si bien la simulación del circuito diseñado no es un paso obligatorio, sí es altamente recomendable. Esto descarta posibles fallos inesperados del sistema.

### ModelSim

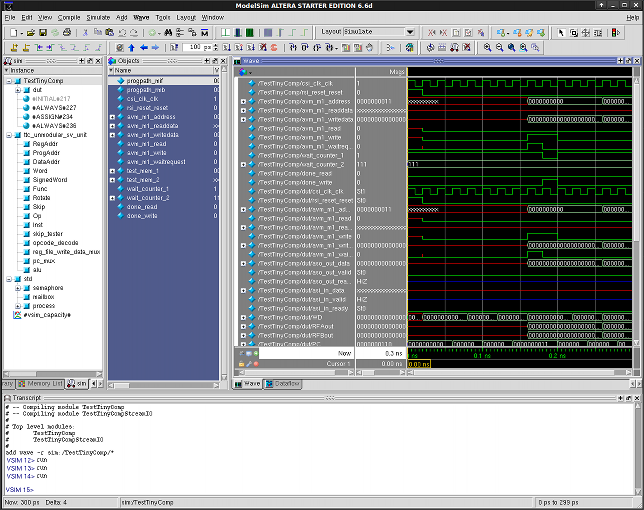
<https://www.mentor.com/products/fv/modelsim/> ¡! MOVER A REFERENCIAS

https://www.microsemi.com/document-portal/doc\_view/131619-modelsim-user

ModelSim es un entorno de simulación para lenguajes HDL (Lenguaje de Descripción Hardware) utilizado tanto en desarrollos para FPGAs como para ASICs. Se pueden simular códigos en lenguajes populares como VHDL y Verilog.



Esta herramienta permite desarrollar (integra compilador) y posteriormente simular el código HDL. Para ello permite crear fácilmente un testbench a partir de un fichero HDL, tras lo cual se puede iniciar una simulación. En esta simulación, se pueden ver representadas todas las señales, tanto externas como internas, previamente programadas. Cabe destacar, que incluye un debugger, lo cual facilita mucho la tarea de depuración de código.



En este punto, el circuito diseñado utilizando HDL no es dependiente de la FPGA a programar, es decir, el circuito es integrable en múltiples FPGAs.

### Quartus

<https://www.altera.com/products/design-software/fpga-design/quartus-prime/support.html>

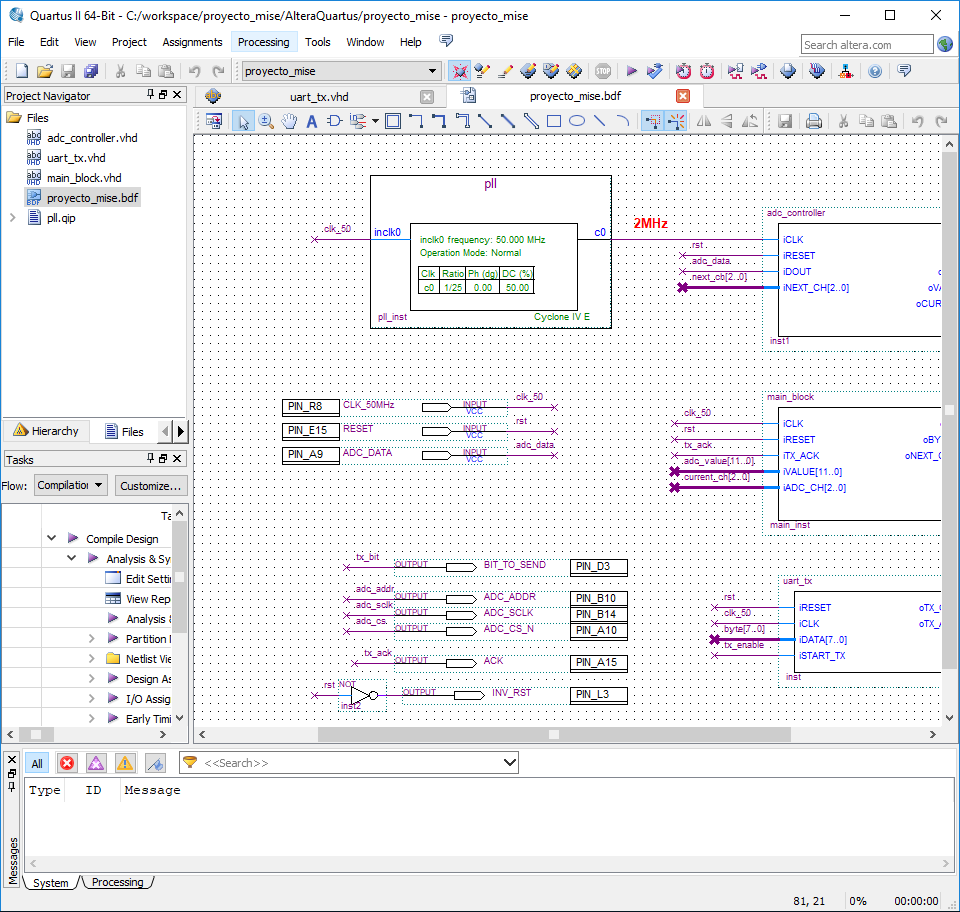
<https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/intro_to_quartus2.pdf>

¡! MOVER A REFERENCIAS

https://www.altera.co.jp/ja\_JP/pdfs/literature/hb/qts/archives/quartusii\_handbook\_archive\_130.pdf

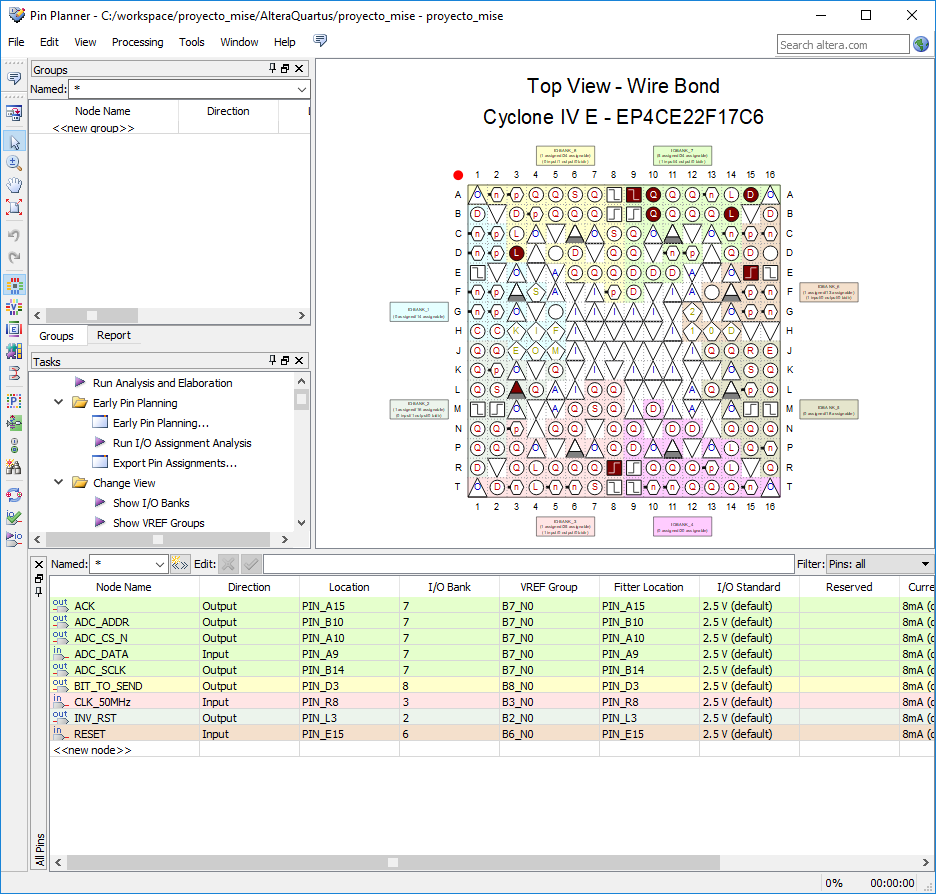
Quartus II es un software de diseño PLD para FPGAs de Altera, el cual es adecuado para diseños de FPGA de alta densidad, diseños PFGA de bajo coste, y diseños CPLD (*Complex Programmable Logic Devices*).

Quartus II permite el diseño de circuitos a través de esquema utilizando bloques y/o componentes individuales (puertas lógicas, contadores, registros, etc).



Los bloques, pueden haber sido creados a través de un fichero HDL. Quartus II también proporciona la posibilidad de crear bloques a través de plantillas, o a través de circuitos complejos en los cuales sólo hay que introducir los parámetros deseados para el funcionamiento deseado (*MegaWizard Plug-In Manager*), lo cual es muy útil a la hora de crear nuevos módulos sin tener que diseñar el circuito entero en HDL.

Quartus II utiliza una interfaz muy intuitiva para la asignación de pines (*Pin Planner*). Al mismo tiempo que se le asigna a cada señal su pin correspondiente, se puede ver un esquema de la FPGA y su relación de pines.

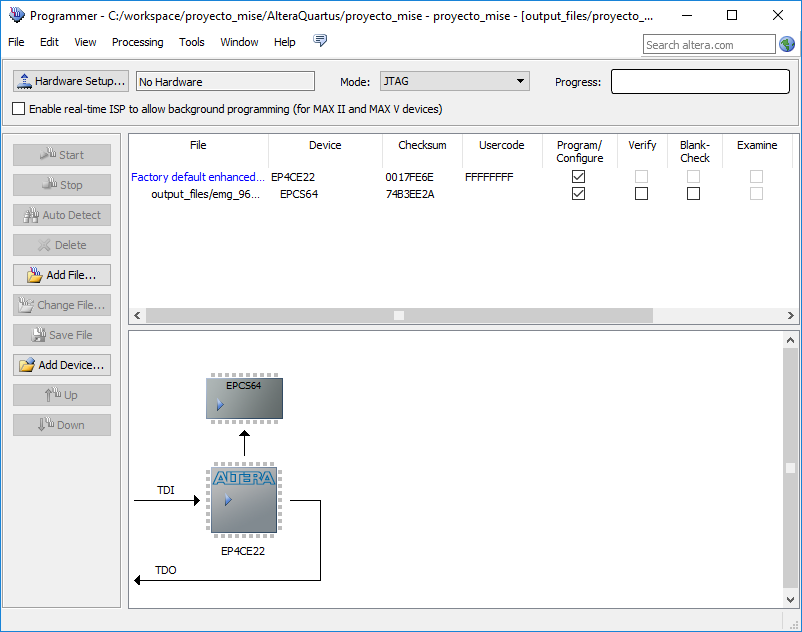


Una vez finalizado el diseño completo del sistema, se procede a compilar el sistema completo, lo cual genera un fichero “.sof”. Quartus realiza una serie de pasos y comprobaciones durante la compilación:

http://quartushelp.altera.com/13.0/mergedProjects/comp/comp/comp\_view\_comp.htm

* Analysis & Synthesis: comprueba los ficheros de diseño y errores de diseño globales. Esta acción también realiza una síntesis para minimizar la lógica del diseño, y realiza un mapeo para implementar el diseño lógico utilizando los recursos del sistema.
* FItter (Place & Route): ajusta la lógica del diseño al dispositivo. Se realizan acciones de optimización.
* Assembler (Generate programming files): convierte el *Fitter* generado, la lógica de celdas y la asignación de pines en una imagen programable del dispositivo.
* TimeQuest Timing Analysis: analiza, depura y valida el rendimiento de toda lógica del diseño.
* EDA NEtlist Writer: genera ficheros de salida de tipo *netlist* para su uso con otras herramientas EDA.

Tras la generación del fichero de compilación, éste puede ser volcado a la FPGA usando la herramienta *Programmer* dentro de Quartus.



## **CAPTURA DE DATOS**

Para realizar la captura de datos procedentes de la FPGA, se ha de diseñar e implementar un programa o script, el cual sea capaz de recoger estos datos, para guardarlos y posteriormente mostrarlos en una gráfica.

Existen muchas opciones para realizar esta tarea, pero se ha de intentar seleccionar la opción más flexible y que sea más sencilla, ya que el objetico de este proyecto no se centra en este apartado. Una característica importante a la hora de seleccionar la opción correcta, es el hecho de que la trata de los datos recogidos sea fácil de manejar, y que ofrezca funciones para realizar operaciones con estos datos.

Para la captura de datos, bastaría con crear un script o programa que se ejecute mediante consola, pero es interesante tener la posibilidad de crear una GUI (*Graphical User Interface*) de manera sencilla para poder configurar las opciones de la captura, representación y almacenaje de los datos.

A continuación, se enumeran algunas de las opciones que han sido consideradas:

* Matlab: Matlab es una herramienta muy potente en cuanto al manejo de datos. Funciona mediante scripts, aunque existe la posibilidad de añadir librerías para crear GUI, pero es más limitado comparado con otras opciones. Es una opción muy interesante si los datos recogidos han de ser procesador matemáticamente. Uno de los principales problemas es que Matlab no tiene licencia gratuita. Compatible con Linux
* VisualStudio (C#): Es una herramienta de desarrollo de software de Microsoft. Es totalmente gratuita y cuenta con muchas librerías para diferentes tareas que se quieran realizar (comunicación serie, representación de gráficos, etc). No es una herramienta creada para realizar tareas matemáticas muy potentes, aunque es capaz de soportarlo. Muy buena herramienta para la creación de GUI. En su última versión, permite realizar compilaciones para cualquier sistema operativo, pero ha de realizarse una compilación diferente por cada uno de ellos.
* GNU Octave: muy similar a Matlab, pero se distribuye bajo licencia gratuita. Es multiplataforma y cuenta con muchas de las funcionalidades que ofrece Matlab, y los scripts escritos en Octave se pueden ejecutar en Matlab sin apenas modificaciones, es más lento y menos potente que Matlab. La creación de GUI es bastante limitada.
* Python: lenguaje de programación multiplataforma y gratuito. Es muy potente y flexible, ya que existen librerías gratuitas para cualquier propósito. Permite la ejecución de scripts, así como la creación de GUI de manera muy sencilla. Es un lenguaje fácilmente integrable con otros programas. Cuenta con unas librerías que lo hacen muy similar a Matlab (y Octave). Gracias a librerías como ‘*matplotlib*’ y ´*numpy*’ se pueden representar gráficas y realizar tareas matemáticas de manera muy sencilla. Cuenta con una gran comunidad online.

Tras un estudio previo, se ha decido utilizar Octave y Python.

Python es elegido por ser un lenguaje más sencillo que los demás, y por contar con un gran soporte online y de muchas librerías. Python es hoy en día uno de los lenguajes más utilizados para tareas científicas, además es multiplataforma y la implementación de una pequeña GUI es muy sencilla.

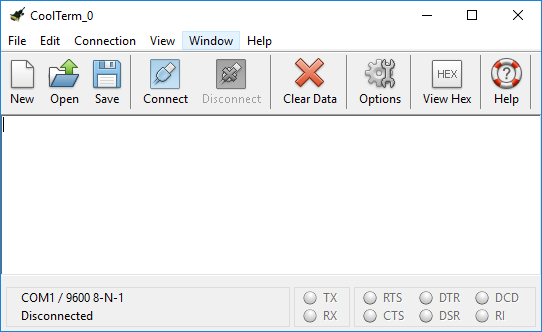
Se implementará también un pequeño script en Octave, al ser Matlab una herramienta muy utilizada para el procesado de datos y tareas matemáticas, alguien que tomará este proyecto para continuar una línea de estudio o trabajo, podría necesitar dicho script. La decisión de Octave sobre Matlab es el hecho de que el primero tiene licencia gratuita. La migración de Octave a Matlab no necesita apenas de cambios.

Python será la herramienta principal la que soporte una GUI y de más opciones a la hora de configurar el procesado de los datos. Python mostrará gráficas en tiempo real, mientras que Octave se limitará a almacenar los datos en archivos y representarlos cuando termine la recepción de datos.

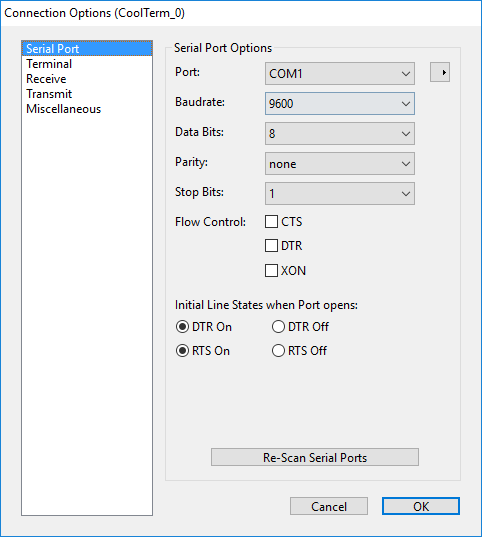
### CoolTerm

http://freeware.the-meiers.org/CoolTerm\_ReadMe.txt.html

Es una aplicación muy simple de Terminal para puerto serie y cuenta con una licencia gratuita. Esta herramienta será utilizada durante el desarrollo del proyecto, para comprobar que los datos que se reciben son correctos antes de la completa integración con la aplicación final.



La configuración para la comunicación es muy sencilla y es muy similar a cualquier otra herramienta de Terminal para la comunicación serie.



### GNU Octave

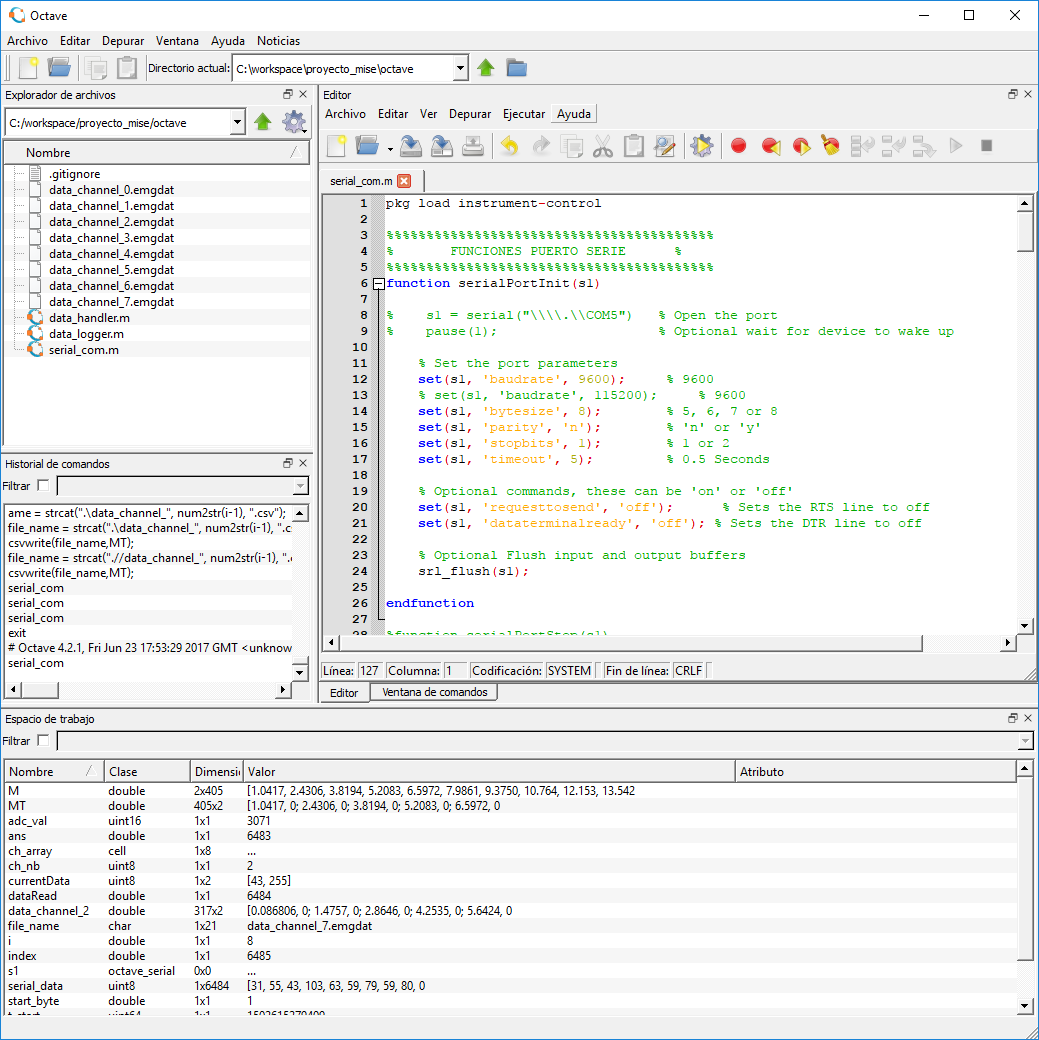
<https://www.gnu.org/software/octave/>

Es un programa libre para realizar cálculos numéricos. Es el equivalente libre de Matlab.

Es un lenguaje de alto nivel principalmente destinado a cálculos numéricos. Generalmente es utilizado para resolver ecuaciones lineales y no lineales, álgebra lineal numérica, análisis estadístico y para realizar otro tipo de experimentos numéricos.

En su última versión, cuenta con una interfaz gráfica (GUI), la cual incluye un editor de código, un depurador, un explorador de archivos y un intérprete de lenguaje.

El intérprete de comandos permite ejecutar comandos de modo interactivo. Tiene una sintaxis de lenguaje propia, siendo esta prácticamente idéntica a Matlab, por lo que es compatible con muchos scripts escritos en Matlab. Los scripts no compatibles, generalmente, necesitan muy pocas modificaciones para ser poder ser ejecutados en Octave.



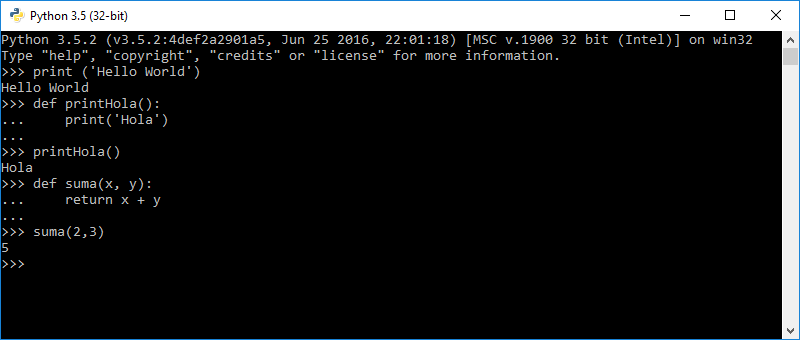
### Python

<https://www.python.org/>

Python es un lenguaje de programación interpretado que se basa en favorecer un código legible. Python tiene una licencia de código abierto, la cual es compatible con licencia GNU. Cabe destacar que es un lenguaje multiplataforma.

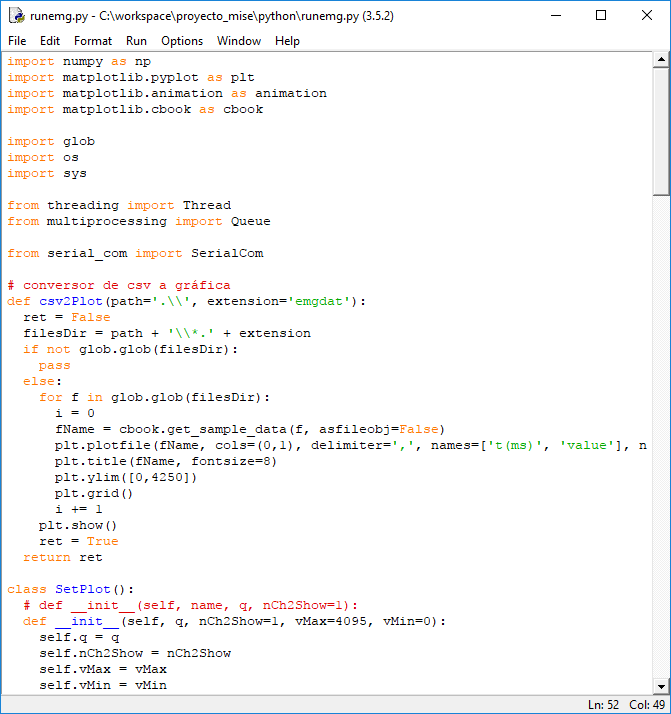
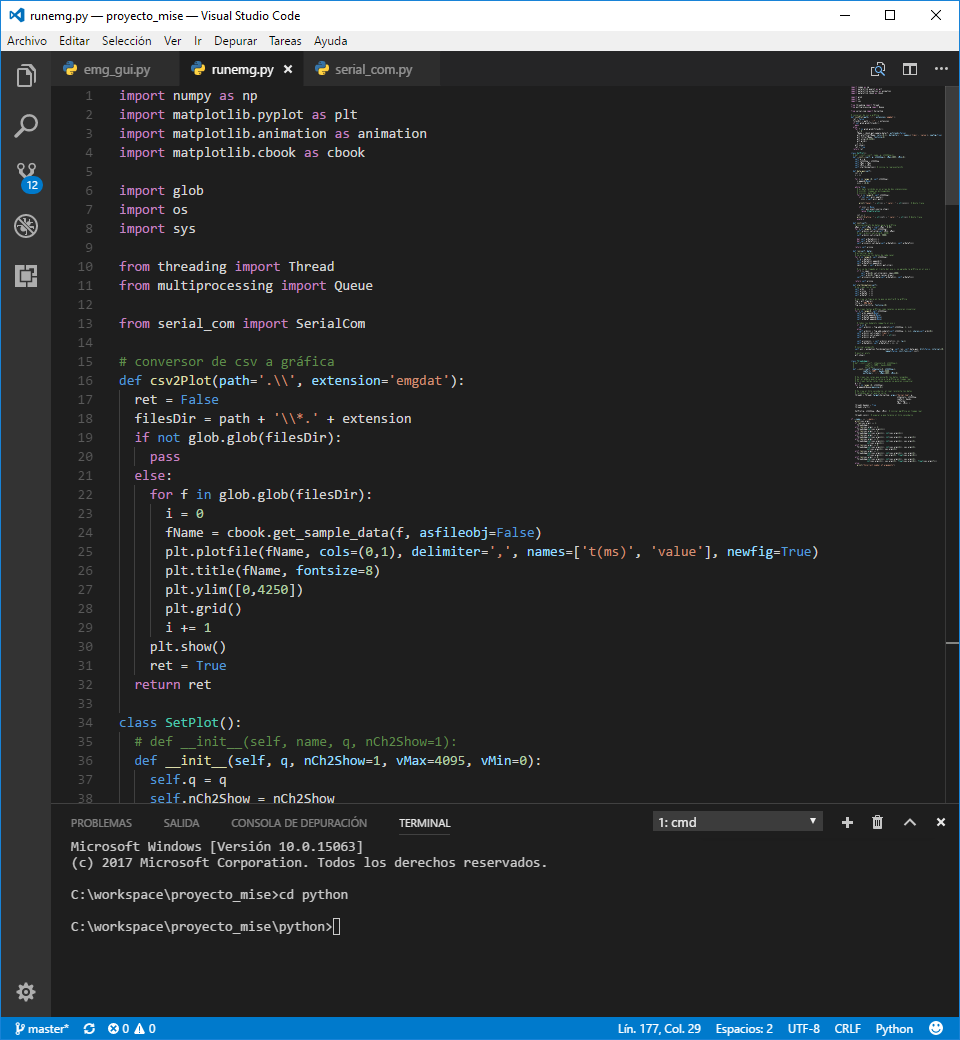
Es un lenguaje de muy alto nivel, muy simple de usar. Permite dividir el código en módulos para ser usados por otros programas Python. Python se distribuye con una amplia colección de módulos ya integrados para el uso básico de programas. Algunos de estos módulos proporcionan elementos como E/S de ficheros, llamadas al sistema, sockets, e incluso interfaces para GUI.

Python es un lenguaje interpretado, lo que puede ahorras mucho tiempo durante el desarrollo de un programa porque no necesita compilación ni linkado. El intérprete puede usarse interactivamente, lo que hace que sea fácil el experimentar con las características del lenguaje. También puede usarse como una calculadora de escritorio.



Python permite que los programas sean escritos de manera compacta y legible. Los programas en Python suelen ser más cortos que sus equivalentes en C, C++ o Java.

Python también se distribuye con una IDLE para el desarrollo de scripts, aunque existen opciones más interesantes para el desarrollo de programas en Python. Para este proyecto, se utilizará Python en conjunto con *Visual Studio Code* de *Miscrosoft*.



Para este proyecto son necesarios módulos para representación de gráficos, para manejo de datos numéricos, para comunicación serie y para implementación de una pequeña GUI. A continuación, se ofrece una breve descripción de las librerías más importantes utilizadas:

<https://matplotlib.org/>

<http://www.numpy.org/>

<http://pythonhosted.org/pyserial/>

https://docs.python.org/3.5/library/tk.html

* Matplotlib: es un módulo para generación de gráficos en 2D y 3D de alta calidad. Los gráficos generados son interactivos y puede ser generados en multitud de formatos. Destaca por tener una librería (*pyplot*) muy parecida a Matlab.
* NumPy: es el módulo por excelencia para computación científica en Python. Contiene, entre otras cosas, un potente objeto de tipo array de dimesión N, y unas capacidades de manejo de algebra lineal y transformadas de Fourier.
* PySerial: este módulo encapsula el acceso del puerto serie. Soporta todo tipo de configuración (baudrate, stopt bits, etc.).
* Tkinter: es el módulo básico para el desarrollo de interfaces gráficas de usuario. Permite la creación de elementos (botones, etiquetas, cajas de texto, etc.) muy fácilmente.

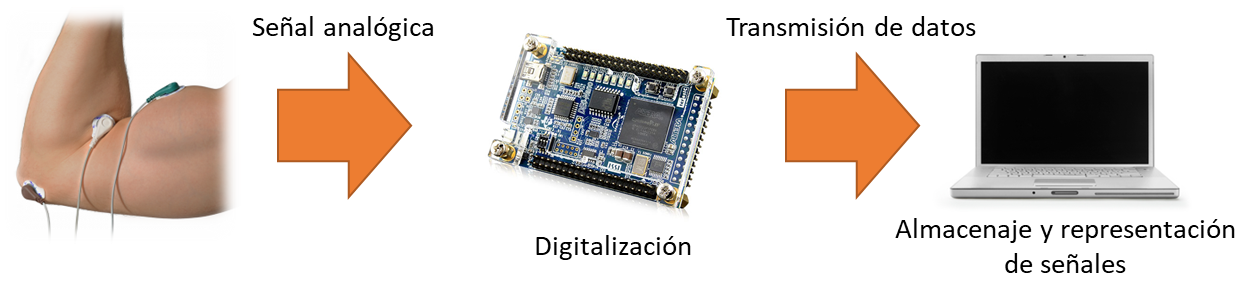
Python cuenta con una comunidad de desarrolladores muy extensa, lo que le da un gran soporte y hace que sea muy potente, ya que todas las librerías que se crean para Python son libres.

# TRABAJO DESARROLLADO

## **DESCRIPCIÓN GENERAL**

El objetivo de sistema desarrollado es la captura de las señales eléctricas emitidas por un músculo, y su posterior almacenaje y representación en un ordenador. Para ello, se utilizará la placa de desarrollo *DE0-Nano* de *Terasic*. Esta placa contiene varios módulos embebidos, de los cuales, para este proyecto, se hará uso de:

* Altera Cyclone® IV EP4CE22F17C6N (FPGA)
* ADC128S022 (Conversor A/D de 8 canales)



Las señales analógicas procedentes del músculo son llevadas al ADC. El ADC es controlado por la FPGA, la cual establecerá qué canal y en qué momento (teniendo en cuenta las especificaciones de tiempo del propio ADC) ha de ser muestreado.

La FPGA, aparte de gobernar el ADC, será la encargada de empaquetar los datos recibidos, para su posterior envío al ordenador. Este envío se realizará mediante una comunicación serie (RS232).

La comunicación entre FPGA y ordenador será inalámbrica a través de Bluetooth. Para ello, se dispondrá de un módulo HC-05 y un PC con conexión Bluetooth, o en su defecto, con un módulo Bluetooth en formato USB.

El ordenador es responsable de desempaquetar los datos recibidos y guardarlos en un fichero para su posterior estudio.

Cabe destacar, que este sistema ha de ser flexible, por lo que su diseño ha de ser modular. Al propio ADC podría ser conectada cualquier señal analógica (teniendo en cuenta las limitaciones de la placa), y los datos podrían ser enviados a cualquier sistema con conexión UART.

Por ejemplo, se podrían sustituir los sensores EMG por unos sensores de monitorización cardíaca, y el ordenador, por un dispositivo móvil con sistema Android.

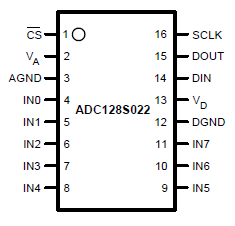
## **ESPECIFICACIONES**

### Especificaciones ADC128S022

El ADC es capaz de leer hasta 8 entradas analógicas, con una resolución de 12bits cada una de ellas. Para esto es necesario seguir las especificaciones dadas por el fabricante en cuestión de tiempos.

A continuación, se explicará cuáles son los parámetros a tener en cuenta para el diseño del controlador de este dispositivo.

En primer lugar, se han de estudiar los pines conectados a las entradas y salidas de la FPGA o a los headers. La siguiente tabla contiene una breve descripción de los pines a tener en cuenta.



|  |  |  |  |
| --- | --- | --- | --- |
| PIN | | TIPO | DESCRIPCIÓN |
| NOMBRE | Número |
| CS\_n | 1 | Digital I/O | Chip select (negado). La conversión comienza con el flanco descendente de CS\_n. La conversión continua mientras CS\_n se mantenga en nivel bajo. |
| IN0 – IN7 | 4-11 | Analog I/O | Entradas analógicas. Estas señales tienen un rango de 0V a Vref |
| DIN | 14 | Digital I/O | Entrada digital de datos. El Registro de Control es cargado a través de este pin en los flancos de subida del pin SCLK |
| DOUT | 15 | Digital I/O | Salida digital de datos. Los datos muestreados son enviados a través de este pin en los flancos de bajada del pin SCLK. |
| SCLK | 16 | Digital I/O | Entrada digital del reloj. El rango de frecuencias de trabajo para esta entrada es de 0.8 MHz a 3.2 MHz. Este reloj controla directamente los procesos de conversión y lectura. |

En la placa DE-Nano, los pines IN0-IN7 están conectados al header 2X13. Estos pines han de conectarse directamente con el exterior (señales analógicas que se desean muestrear).

Desde el punto de vista de la FPGA, los demás pines son tratados con entradas o salidas:

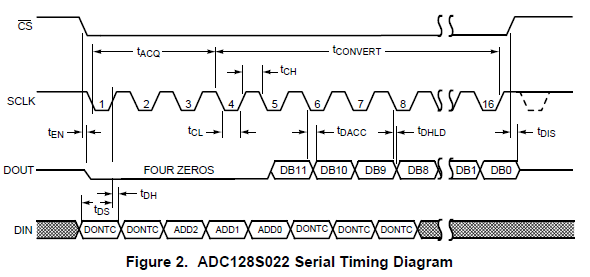
* Entradas: DOUT
* Salidas: CS\_n, DIN, SCLK.

En segundo lugar, se ha de respetar los tiempos especificados por el propio fabricante. A continuación, se muestra el diagrama de tiempos.



Se puede observar que cada 16 pulsos de reloj se repite la secuencia, por lo que se puede considerar al conjunto de 16 pulsos con un ciclo.

* Pin DIN: a través de esta línea se ha de especificar qué canal se quiere muestrear. Los datos del canal especificado en el ciclo actual, serán entregador por el ADC en el siguiente ciclo. Los datos del canal serán leídos durante los flancos de subida de los pulsos 3, 4 y 5 (MSB en primer lugar).
* Pin DOUT: a través de este pin se enviarán los datos muestreados. Los datos enviados en el ciclo actual corresponden a los datos de canal que se envió por la línea DIN en el ciclo anterior. Los datos del canal serán enviados durante los flancos de bajada de los pulsos 5 a 16 (MSB en primer lugar).



## **DISEÑO**

### Diseño sistema

El diseño se ha realizado teniendo en cuenta que la mayor limitación del sistema en su conjunto es el tiempo de procesado de un dato de ADC: desde que se envía el canal que se quiere muestrear hasta que el dato es procesado por el PC.

Para que el sistema sea más rápido, la comunicación entre PC y FPGA será sólo en un sentido. La FPGA enviará los datos muestreados al PC, y éste tan sólo se limitará a procesar dichos datos. En ningún caso permite la comunicación del PC hacia la FPGA, ya que esto ralentizaría la obtención de datos.

Debido a que el PC sólo obtiene datos procedentes de la FPGA, está tendrá que enviar los datos del ADC junto con el canal al que corresponde esos datos. Si esto no fuese así, el PC no sería capaz de relacionar el dato recibido con un canal en concreto.

Teniendo en cuenta lo antes mencionado, el número de bits a enviar son 15 (12 del valor muestreado y 3 del número de canal), lo que hace un total de dos bytes por cada dato muestreado.

El siguiente diagrama muestra el orden de estos bits:

**Byte MSB**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | 0 | NCanal2 | NCanal1 | NCanal0 | Dato11 | Dato10 | Dato9 | Dato8 |

**Byte LSB**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | Dato7 | Dato6 | Dato5 | Dato4 | Dato3 | Dato2 | Dato1 | Dato0 |

El PC tendrá que desempaquetar estos datos para obtener el número de canal y el dato muestreado antes de almacenarlos.

Se pretende que el sistema sea modular, por lo que en la FPGA se implementa tres módulos con funciones muy definidas:

* ADC Controller: este módulo será el encargado de comunicarse con el ADC. Enviará el número de canal que ha de muestrear y recogerá los datos enviados por el ADC (valores de conversión)
* UART TX: será el encargado de enviar los bytes por línea serie.
* Main Block: el módulo principal. Este módulo será el intermediario entre los dos anteriores. Es el único dotado con cierta inteligencia. Sus funciones son:
  + Enviar al “ADC Controller” el número de canal que se quiere muestrear.
  + Recibir del “ADC Controller” los datos y el número de canal al que corresponden esos datos, para a continuación empaquetarlos para su envío a través de la UART.
  + Enviar a “UART TX” el byte que se quiere enviar en cada momento.

En este sistema, se muestrearán todos los canales. Se comienza obteniendo los datos del canal 0 y se continua de forma ascendente hasta el canal 7. Una vez alcanzado este canal, se vuelve a comenzar del canal 0. Es decir, se muestrean todos los canales por igual de forma circular.

Configuración UART

El siguiente paso consiste en seleccionar la correcta configuración de la UART, así como la velocidad del reloj del ADC.

La siguiente tabla muestra la configuración de la UART:

|  |  |
| --- | --- |
| UART | |
| Bits | 8 |
| Stop | 1 |
| Parity | 0 |
| Muestreo | |
| Bytes por dato | 2 |
| Canales | 8 |

Con los datos arriba presentados, se calcula el periodo de muestreo de los canales. Cada byte enviado consta de 10 bits (Start bit, 8 bits de dato, Stop bit), y cada conversión supone el envío de dos bytes al PC.

La fórmula anterior describe el periodo de envío de datos, pero es interesante calcular cuál es el periodo de muestreo de cada canal. Para ello se modifica la fórmula, sabiendo que se disponen de 8 canales.

La siguiente tabla muestra los períodos calculados para las diferentes velocidades.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Velocidad  (baudios) | Tiempo por bit (s) | Tiempo por byte (s) | Periodo muestreo un canal (s) | Periodo muestreo todo los canales (s) |
| 115200 | 8,68056E-06 | 8,68056E-05 | 0,000173611 | 0,001388889 |
| 76800 | 1,30208E-05 | 0,000130208 | 0,000260417 | 0,002083333 |
| 57600 | 1,73611E-05 | 0,000173611 | 0,000347222 | 0,002777778 |
| 56000 | 1,78571E-05 | 0,000178571 | 0,000357143 | 0,002857143 |
| 38400 | 2,60417E-05 | 0,000260417 | 0,000520833 | 0,004166667 |
| 19200 | 5,20833E-05 | 0,000520833 | 0,001041667 | 0,008333333 |
| 14400 | 6,94444E-05 | 0,000694444 | 0,001388889 | 0,011111111 |
| 9600 | 0,000104167 | 0,001041667 | 0,002083333 | 0,016666667 |

Reloj ADC

Como ya se ha explicado antes, el ADC permite un rango de trabajo de reloj entre 0.8 MHz y 3.0 MHz.

Teniendo en cuenta que el reloj integrado en la placa DE0-Nano es de 50MHz, se puede obtener una señal de reloj dentro del rango de trabajo del ADC de manera muy sencilla a través de un PLL: basta con dividir la frecuencia entre 25 para obtener una señal de 2 MHz.

Esta velocidad no es limitante para el sistema, ya que la velocidad de la UART será más lenta incluso trabajando a 115200 (ver tabla X.X). A continuación, se realizan los cálculos que lo demuestran.

El ADC trabaja en ciclos de 16 bits (cada bit es una señal de reloj). Cada ciclo de ADC tiene un periodo:

\* 86,8 : periodo de envío de un byte por UART a 115200 baudios

La fórmula anterior muestra el tiempo que necesita el ADC para un ciclo entero. Hay que tener en cuenta, que el canal que se quiere muestrear se le envía en el ciclo anterior al ciclo en el que se obtienen los datos, por lo que desde que se selecciona el canal y se obtienen los datos son necesarios dos ciclos.

\* 86,8 : periodo de envío de un byte por UART a 115200 baudios

Debido a que el ADC es más rápido la UART, se muestrearán varios datos de un mismo canal de manera continua. Hasta que un dato completo no es transmitido a través de la UART, no son leídos los datos del siguiente canal del ADC. A efectos prácticos no tiene ninguna importancia, ya que se procesará siempre un dato de cada canal.

Procesado de datos en PC (servidor)

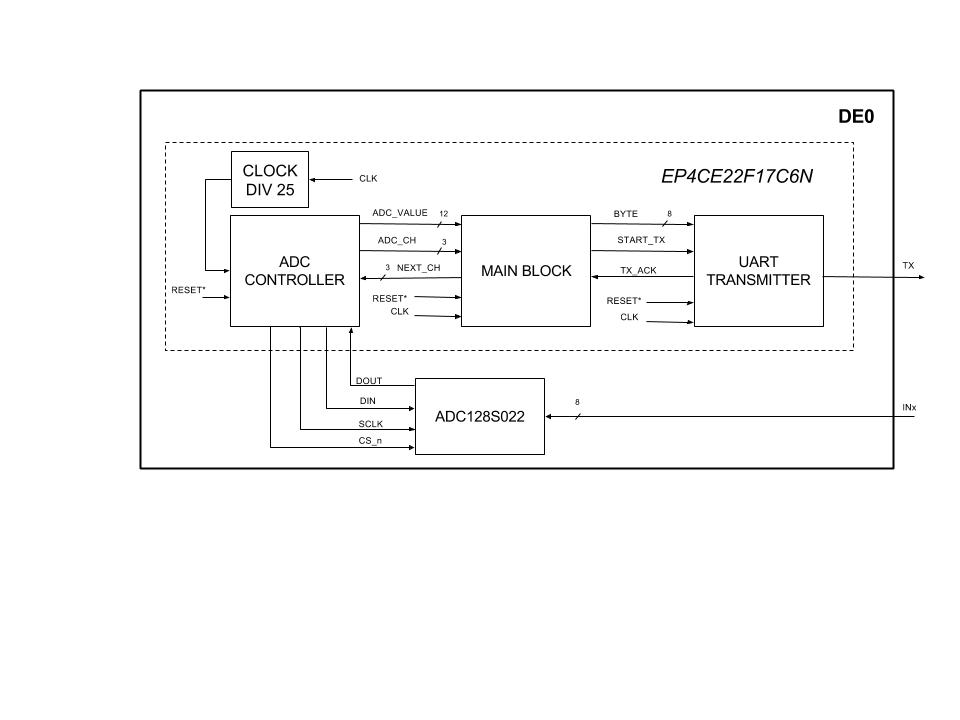
Se han considerado diferentes formatos en los cuales almacenar los datos. Tras un filtro previo, la decisión final quedó entre formato “EXCEL” o “CSV”. El problema que tiene “EXCEL” frente a “CSV”, es que su manejo a través de aplicaciones de escritorio o scripts (Matlab, Octave, Python…) es más complejo, mientras que casi todos los lenguajes de procesado tienen funciones para manejo de ficheros “CSV”.

El formato en el que se almacenan los datos recibidos será en un fichero tipo “CSV” para cada canal. Esto supone que por cada sesión de trabajo se obtendrán 8 ficheros: “chX.emgdat” (X corresponde al número de canal).

Los datos son almacenados en dos columnas. La primera corresponde al tiempo transcurrido desde el inicio de la captura de datos y el procesado del dato a almacenar. La segunda por su parte será el valor del dato recibido. La gestión del tiempo transcurrido correrá a cargo del servidor (PC).

### FPGA: Esquema general

El siguiente esquema muestra el diseño completo de la FPGA



Se puede observar que el sistema completo trabajará con dos frecuencias diferentes:

* 50 MHz: Main Block, UART TX.
* 2 MHz (50MHz/25): ADC Controller.

Comunicación ADC Controller – ADC128S022

“ADC Controller” suministra su mismo reloj al ADC a través de “SCLK”. Este módulo también controla el “chip select” del ADC.

A través de la línea *DIN,* “ADC Controller” envía el siguiente canal a muestrear, y a través de *DOUT* se leerá el valor muestreado.

Comunicación Main Block - ADC Controller

El bloque principal envía al control de ADC el siguiente canal que se quiere muestrear.

El control del ADC a su vez, enviará al bloque principal el valor del ADC y el número de canal al que corresponde dichos datos. Estos datos se envían de manera independiente, ya que “Main Block” es el encargado de empaquetar estos datos de la manera que corresponda.

Comunicación Main Block – UART TX

El módulo principal envía el byte que el módulo UART debe comunicar al PC. Sólo en byte es enviado al módulo UART, ya que es éste es el encargado de empaquetar dicho byte con la configuración que corresponda (paridad, stopbit…) y transmitirlo a la velocidad que corresponda.

Cuando la UART haya enviado un dato, lo hará saber al módulo principal a través de *TX\_ACK*, de esta manera, “Main Block” sabe que puede enviar el siguiente byte.

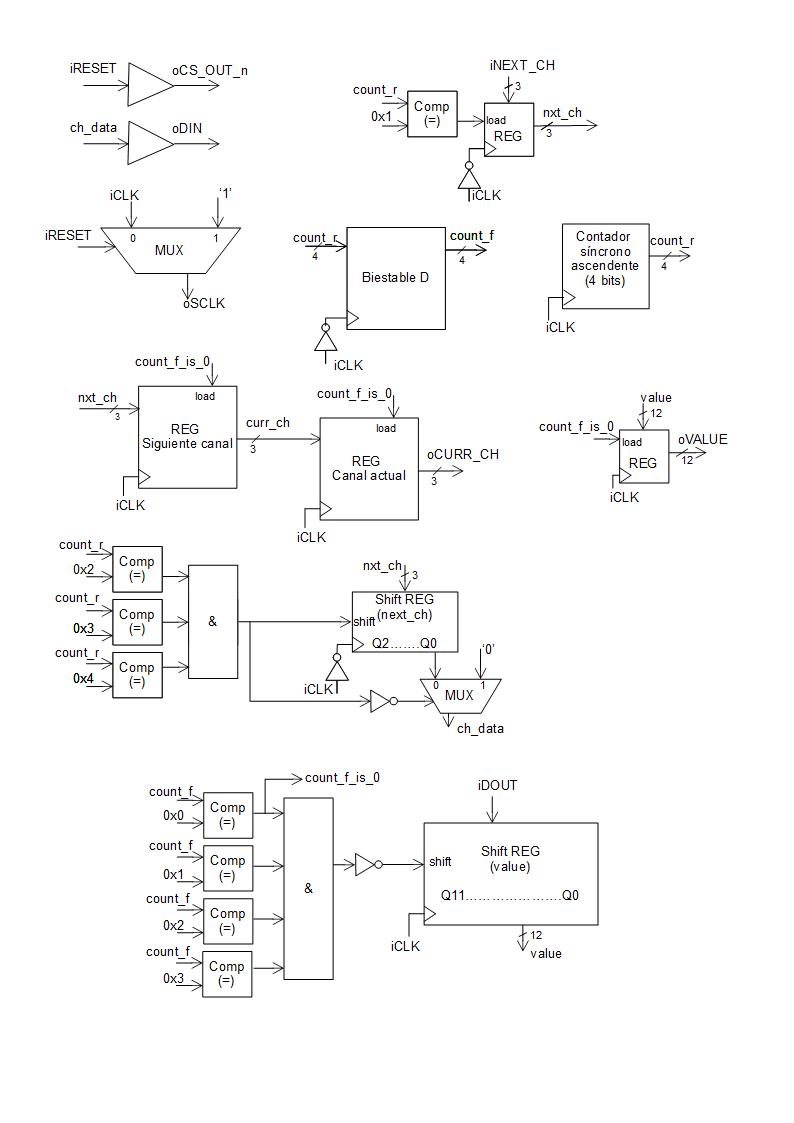
La línea *START\_TX* hará saber a la UART que puede recoger el dato de la línea *BYTE* para transmitirlo hacia el PC.

La UART envía los datos serie a través de *TX*.

### FPGA: módulo control ADC

El módulo ADC no tiene unidad de control, ya que no es necesaria. Se ha implementado tan solo utilizando unidad de proceso.

Unidad de proceso



La imagen anterior muestra los diferentes elementos que componen el módulo “ADC Controller”. Este módulo se ha diseñado teniendo en cuenta las especificaciones técnicas dadas por el fabricante del ADC128S022 (ver X.X).

El multiplexor *MUX* se encarga de enviar la señal de reloj hacia el ADC128S022 siempre que el reset no esté activo. En ese caso, la señal que se envía será siempre ‘1’. De esta manera, el ADC128S022 no realizará conversiones en estado reset.

Para enviar los datos del siguiente canal a muestrear, y para la lectura correcta de los datos de ADC es necesaria la cuenta de pulsos, ya que se definen unos tiempos para el envío y la recepción de bits.

Las especificaciones definen que el número de canal a muestrear ha de enviarse a través de *oDIN* en los flancos ascendentes de *iSCLK*. Sin embargo, los datos recibidos a través de *iDOUT* han de leerse en los flancos descendentes de *iSCLK*. Es por esto, que se ha implementado un contador síncrono ascendente, que cuenta los flancos ascendentes de *iSCLK (count\_r),* y un biestable D, el cual en los flancos descendentes tomará el valor de *count\_r.* De esta manera *count\_*f funcionará como un contador de flancos descendentes. Ambos contadores son de 4 bits (de 0 a 15) con overflow, es decir, cada vez que el contador llegue a 15 y se vea incrementado, volverá a comenzad en 0.

Gracias a estos contadores, se puede establecer en qué momentos se cargan señales en registros.

*oVALUE* (valor de conversión ADC) será cargado cada vez que se haya ejecutado un ciclo de ADC (1 ciclo = 16 pulsos de reloj), es decir, que *count\_f* sea 0.

El número de canal a muestrear en el siguiente ciclo (*iNEXT\_CH*) será cargado cuando *count\_r* sea igual a 1. Esto es así, ya que en siguiente flanco comienza la transmisión del canal que se desea muestrear.

Como ya se ha explicado en anteriores apartados, los datos que se reciben del ADC en ciclo actual, pertenecen a los datos del canal que se ha enviado durante el ciclo anterior. Es por esto, que se ha de guardar el número de canal del ciclo anterior para poder enviar por *oCURR\_CH* el canal al que corresponden los datos leídos. Para ello, se implementan dos registros, los cuales son cargados cuando *count\_f* es igual a 0. El Primer registro almacena el número de canal a trasmitir, y el segundo registro contiene canal al cual pertenecen los datos recibidos desde el ADC en el ciclo actual.

Los registros para la comunicación con el ADC (envío y recepción de datos) tienen un funcionamiento muy similar. Ambos se realizan a través de un registro de desplazamiento.

* Registro desplazamiento *next\_ch*: este registro es el encargado de enviar el número de canal del cual se quiere obtener la conversión en el siguiente ciclo. Las propias especificaciones explican que el número de canal (3 bits) ha de enviarse durante los flancos ascendentes 2, 3 y 4 (de msb a lsb). Durante los demás flancos, se ha de mandar ‘0’.

Para ello, en los flancos ascendentes 2, 3 y 4, se desplaza el registro.

A la salida del registro de desplazamiento, se coloca un multiplexor, para que la salida sea ‘0’ en caso de que los flancos sean diferentes a 2, 3 y 4.

* Registro desplazamiento *value*: este registro funciona de manera similar al anterior. La diferencia es que trabaja con los flancos contrarios al registro *next\_ch*.

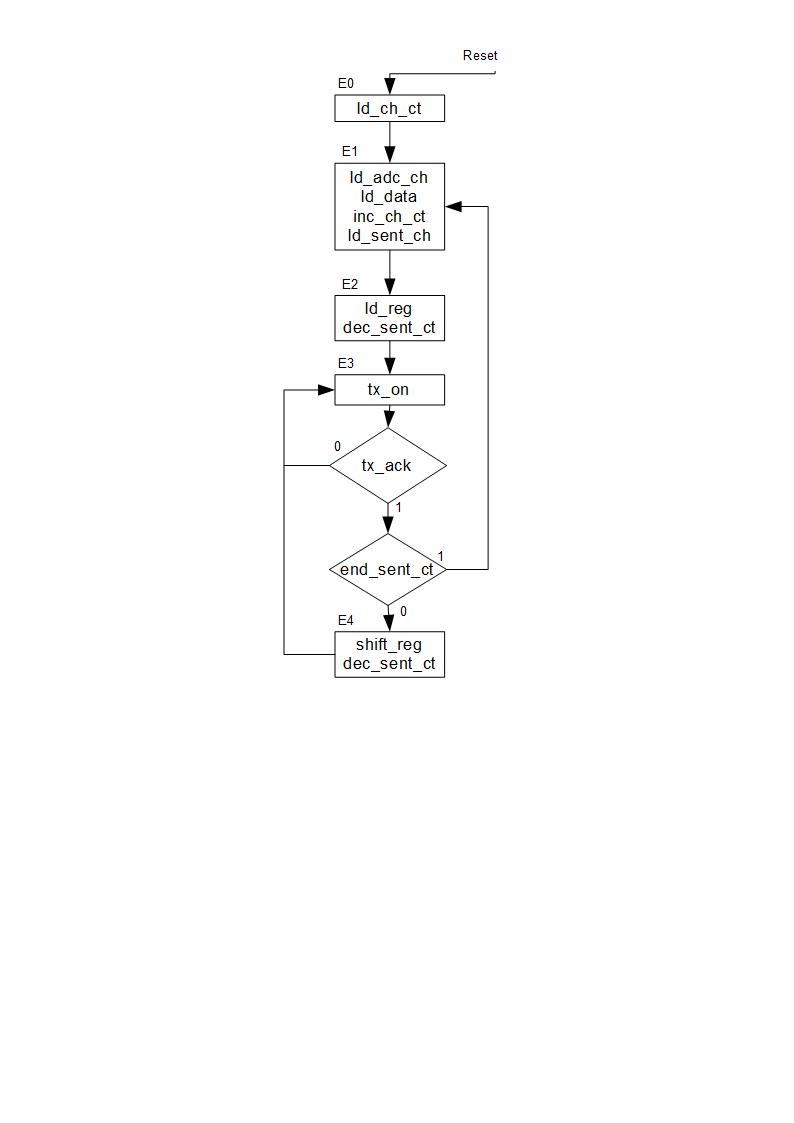
Los datos de conversión se reciben en los flancos descendentes de 4 a 15. Por ello, siempre que *count\_f* sea diferente a 0, 1, 2 o 3, se desplazará el registro.

Los datos son almacenados en *value* durante cada flanco ascendente para no leer un dato durante su cambio.

El valor recibido durante los pulsos 0, 1, 2 y 3 es ignorado.

### FPGA: módulo principal

Unidad de control

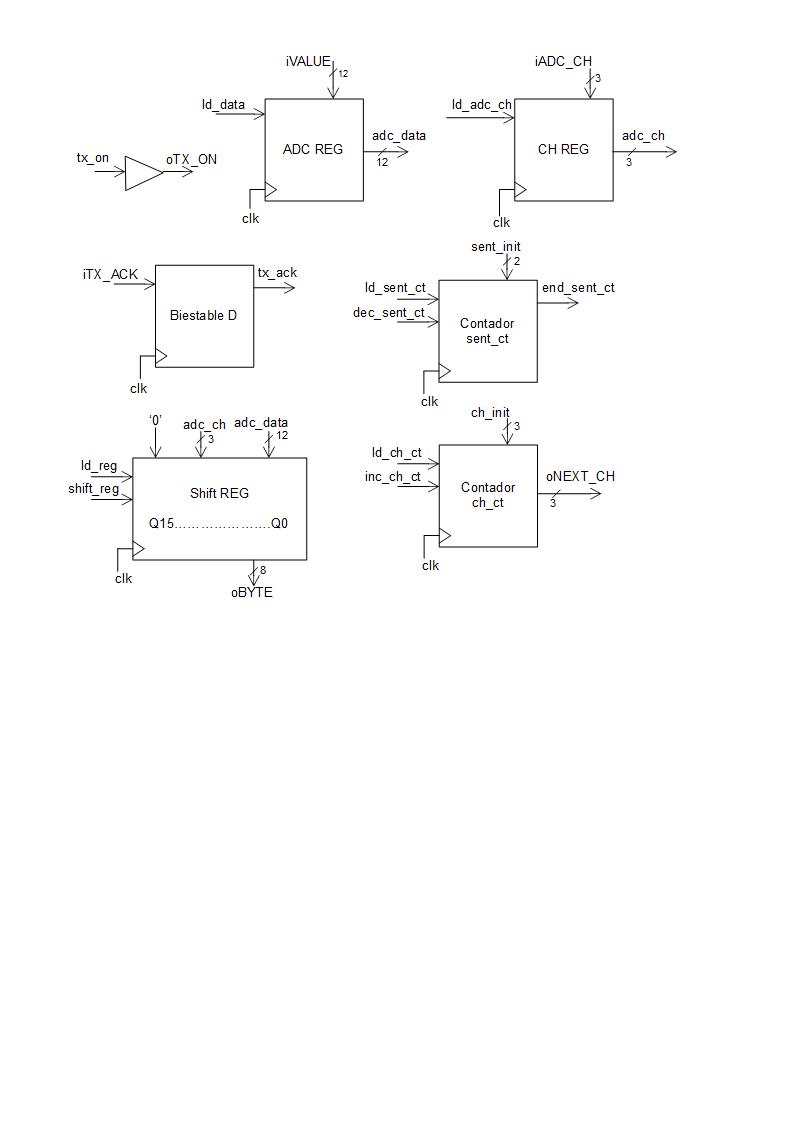


El contador de canal se inicia junto al inicio del sistema.

Cada vez que se envía un dato completo (*n* número de bytes) al módulo UART, se incrementa el número de canal a muestrear, y se cargan todos los datos recibidos desde el módulo ADC.

Para el envío de los datos se activa la señal *tx\_on* para que el módulo ADC sepa que puede enviar nuevos datos. Se espera a que se active la señal *tx\_ack* la cual indica que se puede enviar el byte siguiente. Si no se han enviado todos los bytes correspondientes a los datos actuales, se carga el nuevo byte y se espera a que la UART lo transmita. Si se han enviado todos los bytes, se vuelve a cargar nueva información.

Unidad de proceso



Los registros *ADC REG* y *CH REG* cargan la conversión del ADC y el número de canal al que corresponden los datos respectivamente, cuando su señal de carga (*ld\_data* y *ld\_adc\_ch*) está activa.

El Biestable D, sincroniza la señal *iTX\_ACK*.

El contador ascendente de 3 bits *ch\_ct* lleva la cuenta del siguiente canal que ha de ser muestreado. Con la señal *ld\_ch\_ct* carga el valor inicial *ch\_init*. Cuando la señal *inc\_ch\_ct* el contador se incrementa en uno. Este contador permite overflow, es decir, cuando su valor llega a ‘111’, vuelva a empezar en ‘000’.

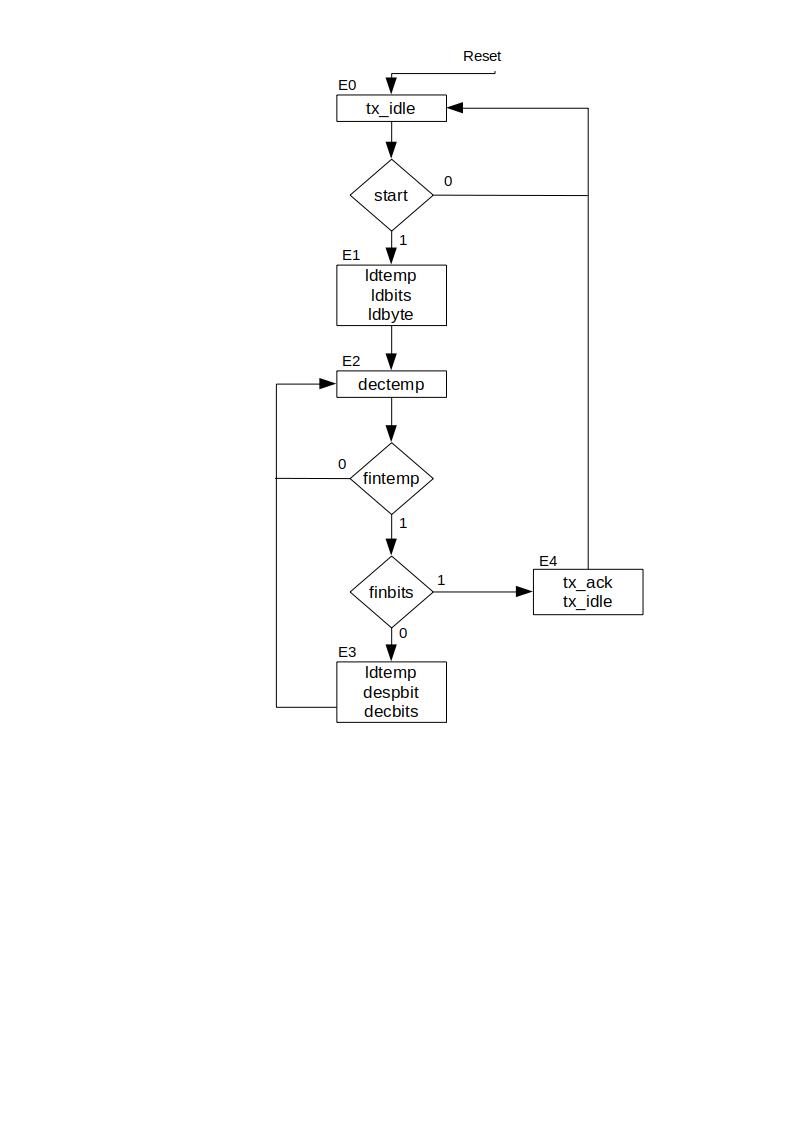
El contador descendente de 2 bits *sent\_ct* sirve para contar el número de bytes que faltan por ser enviados a través de la UART. Su valor se inicializa con *sent\_init* cuando la señal *ld\_sent\_ct* se activa. El contador es decrementado con la señal *dec\_sent\_ct*.

El registro de desplazamiento es de 16 bits. Este registro sirve para separar los 16 bits en dos bytes diferentes. Cuando la señal *shift\_reg* se activa, se desplaza un byte completo. Cada vez que la señal *ld\_reg* se activa, el registro carga la siguiente información:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **bit** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|  | 0 | adc\_ch | | | adc\_data | | | | | | | | | | | |

### FPGA: módulo transmisión UART

Unidad de control

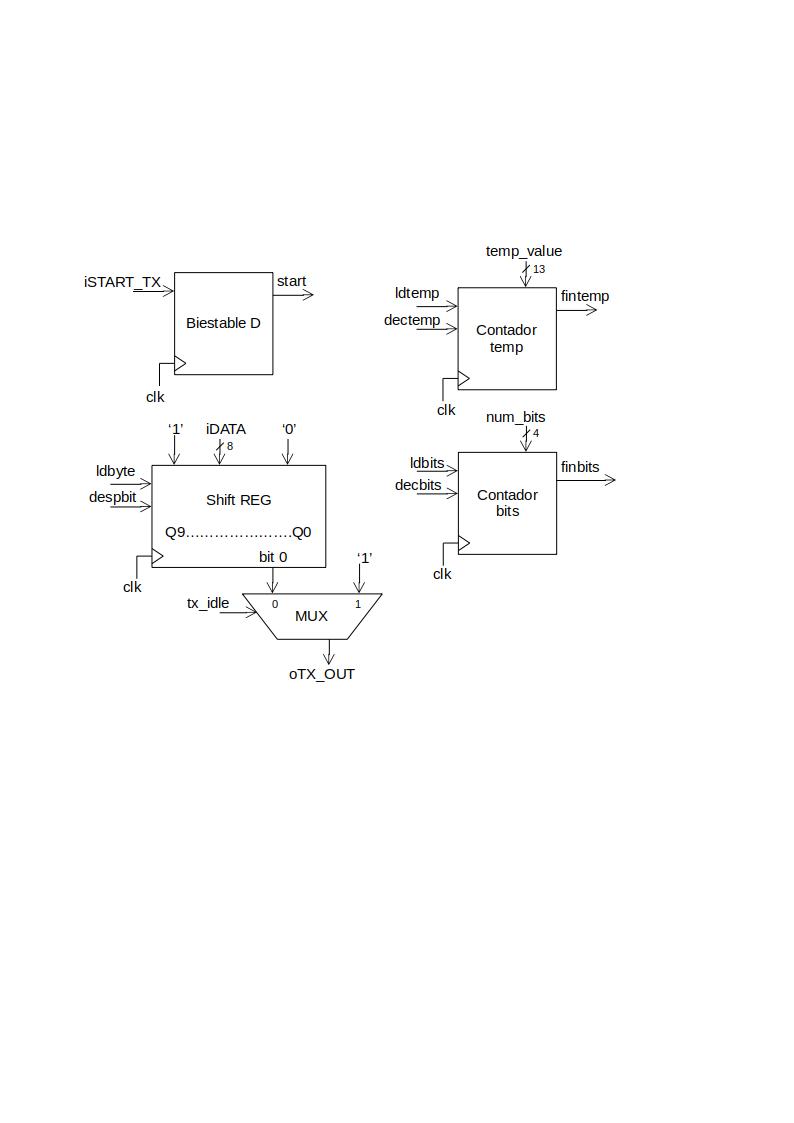


Mientras no reciba la señal de que *start* el módulo permanece en un estado idle(*tx\_idle* activado).

Cuando se recibe la señal que indica que se puede comenzar la transmisión (*start*). Se carga el byte a enviar, a la vez que el contador de bits y el temporizador de envío de un bit.

Cuando finaliza el tiempo de envío de un bit, se carga el siguiente bit a enviar, y se vuelve a iniciar la temporización. Cuando todos los bits son enviados, se activa la señal *tx\_ack* (incida que terminó la transmisión), y se vuelve al estado idle.

Unidad de control



El biestable D sirve para sincronizar la señal de entrada *iSTART\_TX* con el resto del módulo.

El contador descendente *temp* es el encargado de contar el tiempo entre pulsos. Se carga con el valor *temp\_value* cuando se activa la señal *ldtemp*. En cada pulso de reloj en el cual *dectemp* se encuentre activo, se decrementará el valor. Una vez alcanzado el valor ‘0’, se activa la señal *fintemp*.

El valor de *temp\_value* es un valor constante. Es el número de ciclos de reloj necesarios para transmitir un bit. Sabiendo que el reloj de la FPGA es de 50MHz (20ns cada ciclo de reloj), la siguiente tabla muestra el valor de *temp\_value* para el correcto funcionamiento de cada velocidad.

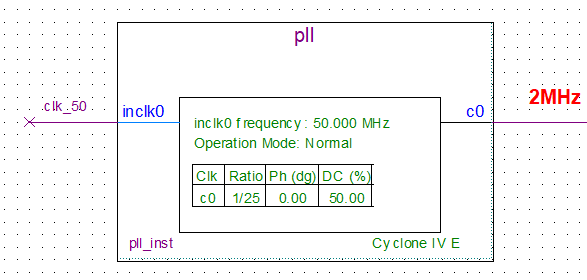
|  |  |  |  |
| --- | --- | --- | --- |
| Velocidad  (baudios) | Tiempo por bit (s) | Ciclos de reloj necesarios para transmitir un bit | Redondeo |
| 115200 | 8,68056E-06 | 434,0277778 | 434 |
| 76800 | 1,30208E-05 | 651,0416667 | 651 |
| 57600 | 1,73611E-05 | 868,0555556 | 868 |
| 56000 | 1,78571E-05 | 892,8571429 | 893 |
| 38400 | 2,60417E-05 | 1302,083333 | 1302 |
| 19200 | 5,20833E-05 | 2604,166667 | 2604 |
| 14400 | 6,94444E-05 | 3472,222222 | 3472 |
| 9600 | 0,000104167 | 5208,333333 | 5208 |

*Contador bits* es un contador descendente de 4 bits, el cuál sirve para contabilizar los bits que han sido transmitidos a través de la UART. SU valor se carga a *num\_bits* con la señal *ldbits*, y decrementa su valor cuando la señal *decbits* está activa. Al llega a ‘0’, se activa la señal *finbits*.

El registro de desplazamiento es de 10 bits. Este registro carga *iDATA* (byte a enviar) cuando la señal *ldbyte* está activa. Cuando la señal *despbit* se activa, se desplaza un bit a la derecha. El bit 9 es el startbit, y el bit 0 el stopbit (ver configuración UART en X.X). El registro tiene el siguiente formato:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **bit** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|  | 1 | iDATA | | | | | | | | 0 |

### FPGA: PLL



El PLL es implementado directamente en Quartus, gracias a la funcionalidad *MegaWizard Plug-In Manager.*

Se ha creado este nuevo basándose en el bloque ya existente *ALTPLL*, el cual ha sido modificado para generar una señal de reloj de 2 MHz partiendo de una entrada de reloj de 50 MHz.

* Ratio: 1/25
* Fase (grados): 0
* Duty cycle (%): 0

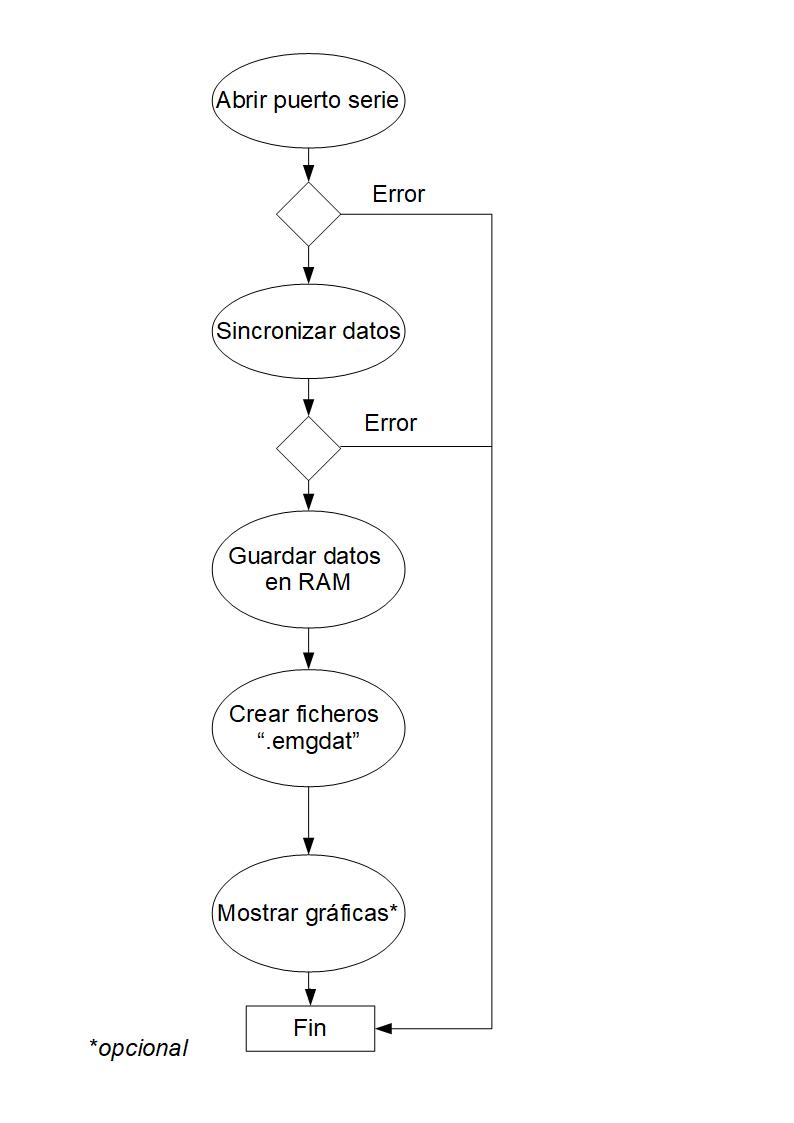
### PC: Script captura de datos

El script de captura de datos ha de ser capaz de abrir el puerto serie y guardar los datos recibidos en un fichero de tipo “CSV”.

Para ello, una vez abierto el puerto serie, tiene que sincronizar los datos recibidos de la FPGA, ya que, como se ha comentado, para obtener los datos de un canal se han de leer dos bytes.

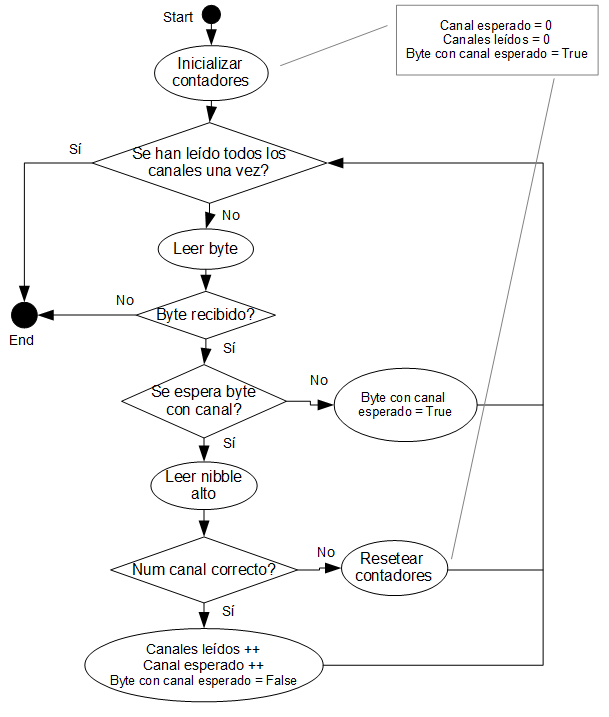
Cuando ha conseguido sincronizarse (reconocer qué byte es el primero), los datos recibidos se guardan en RAM. Se crean vectores para cada canal, ya que del primero byte recibido se obtiene el número de canal. Los datos no se guardan directamente en el fichero CSV. Esto es así, ya que se ahorra tiempo de apertura, escritura y cierre de un documento. Junto a el dato recibido, se guarda el momento (tiempo en ms) en el que se ha recibido este dato

Los datos son volcados a ficheros (un fichero por cada canal) CSV una vez finalizada la recepción de datos.



El dato de cada canal se compone de dos bytes (ver capítulo X.X), es por ello que la sincronización ha de buscar cuál es el primero de los bytes de cada canal. El siguiente un diagrama muestra cómo se realiza esta sincronización.

E

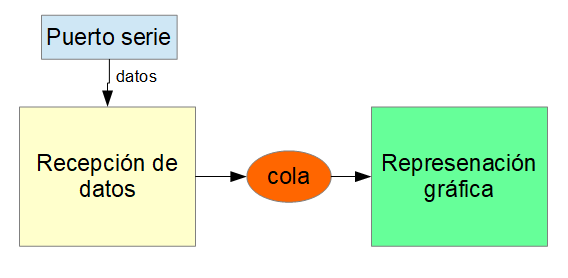


En el diagrama de flujo se puede observar que, una vez leído el primer canal válido, se buscará volver a leer otro canal cada dos bytes, por sólo se comprobará el número de canal cada dos bytes.

Los datos serán almacenados en vectores (RAM) hasta que no se reciba ningún dato por el puerto serie. En ese momento, se volcaran los datos leídos en los ficheros “.emgdat”.

Como se ha comentado con anterioridad, se realizarán dos scripts, uno en Octave y otro en Python. El script de Octave cumplirá con lo descrito hasta este punto.

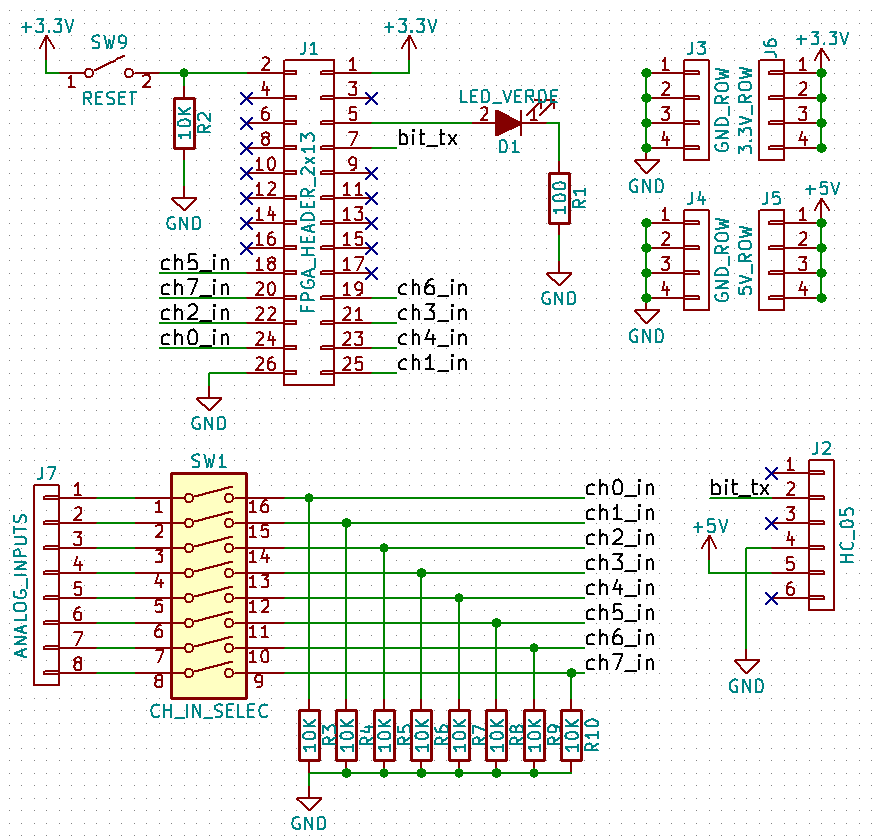
El script en Python, dará la posibilidad de visualizar los datos en tiempo real. Para ello el script se divide en dos hilos (*threads*), uno para recibir los datos, y otro para mostrar estos datos en tiempo real. La comunicación entre los dos hioes se realiza a través de colas. Cada vez que se recibe un dato del puerto serie, se calcula el tiempo transcurrido desde el inicio de la recepción de mensajes hasta este instante. El valor de ADC y el tiempo transcurrido se envían al hilo encargado de la representación en tiempo real, el cual, mostrará en dato. Es importante destacar, que sólo se encolará un dato, si la cola está vacía, es decir, la cola será de tamaño 1. Esto es así, debido a que la representación necesita mucho tiempo para procesar un dato en comparación en tiempo necesario para la recepción. Si se encolaran todos los datos, la representación tendría un retardo. Debido a que no todos los datos recibidos son representados, la resolución de la gráfica en tiempo real será mucho menor que la representación a través de los ficheros “*.emg.dat*”. Se podrán visualizar múltiples canales en tiempo real, por lo que se creará una cola para los datos de cada canal a representar.



### HW: circuito externo

El sistema se ha diseñado teniendo en cuenta que ha de ser portable. Por ello, la conexión de los componentes externos se realiza mediante una placa extensora. Esta placa será conectada directamente con los pines de expansión 2x13 de la placa DE0-Nano, de esta manera, queda todo el sistema integrado, sin que existan componentes independientes (a excepción de los sensores MyoWare).

La figura X.X muestra el diagrama eléctrico de la placa.



Las entradas analógicas son conectadas al conector J7, el cual es lleva todas sus líneas al DIP-Switch SW1. Si el interruptor asociado a cada línea no está activo (OFF), la señal analógica de ese canal será 0 a la entrada de la FPGA. Si el interruptor está activo (ON), la señal analógica será la misma que en el conector J7. DE esta manera, los canales que no son utilizados no quedan al aire, lo cual podría generar voltajes no deseados.

Los conectores J3, J4, J5 y J6, son alimentaciones del sistema. Se han creado estos conectores, para que cualquier dispositivo externo pueda ser alimentado desde la propia placa.

## **IMPLEMENTACIÓN**

La implementación del sistema se ha hecho de manera incremental. Primero se probaron módulos independientemente, para después integrar el sistema completo

### Control de ADC

Para comprobar el correcto funcionamiento del módulo de control ADC, se ha montado un circuito de test.

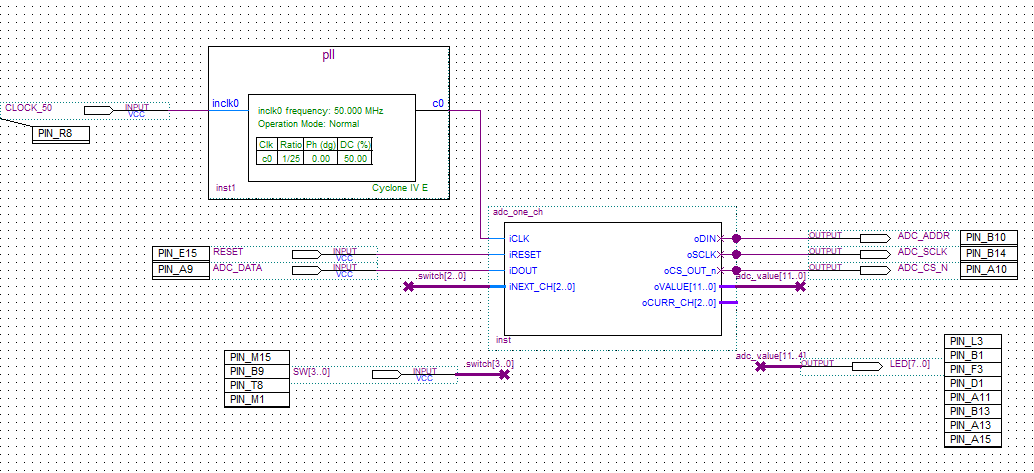
INSERTAR ESQUEMA Y FOTO

La entrada analógica *IN0* del ADC128S022 se ha conectado a un divisor de tensión controlado por potenciómetro y así poder probar el sistema con una entrada analógica variable. La señal reset es contralada por un switch externo.

El canal a muestrear es controlado por los switches integrados en placada DE-Nano.

El valor de salida del ADC se lleva a los LED integrados en la propia placa DE-Nano. Al existir sólo 8 LEDS y siendo la salida del ADC de 12 bits, se han conectado los bits más significativos con los LEDs.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LED | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Bit ADC | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 |

La figura siguiente muestra el sistema de test implementado en Quartus.

El resultado del test ha sido satisfactorio. Al variar el valor del potenciómetro, se observa que los LED varían en función del valor del potenciómetro.

Se ha comprobado el módulo conectando del potenciómetro a las demás entradas del ADC (*IN1-IN7*) y seleccionando el canal correspondiente a través de los switches integrados. El resultado obtenido ha sido el correcto: al seleccionar el canal en el cual está conectado el potenciómetro, se observa que los LEDs varían.

La comprobación del correcto valor de *oCURR\_CH* se realiza con el sistema completo, ya que en este punto es difícil comprobar que los datos del ADC en *oVALUE* corresponde con el canal *oCURR\_CH*. Esto es debido a que el canal a muestrear no varía de manera constante.

### Transmisión UART

El siguiente esquema muestra el circuito de test creado para la comprobación del correcto funcionamiento de este módulo

INSERTAR ESQUEMA Y FOTO

Para realizar el test del módulo UART se implementa un contador ascendente de 32 bits. Los 8 bits más significativos del contador serán la entrada *iDATA* del módulo UART, de esta manera se simula la entrada de un byte de dato. El contador sólo incrementa su valor cuando la señal de reset no está activada, y cuando la señal de *start* está activa.

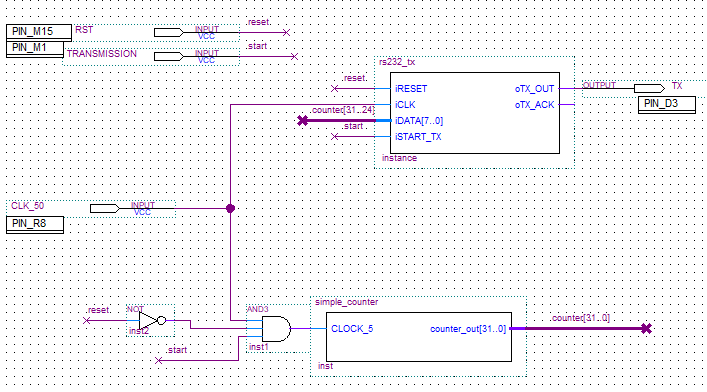
La señal *iSTART\_TX*, la cual indica que se puede comenzar la transmisión del dato, se simula mediante el switch 0 integrado en la placa DE-Nano.

La señal de reset se conecta a un switch externo.

La señal *oTX\_ACK* es testeada con el sistema completo, ya que su activación y desactivación es muy rápida, y su conexión a un LED para su testeo visual no es posible.

*oTX\_OUT* es enviada al puerto serio a través de su conexión con un módulo RS232.

A continuación, se muestra el sistema implementado en Quartus.



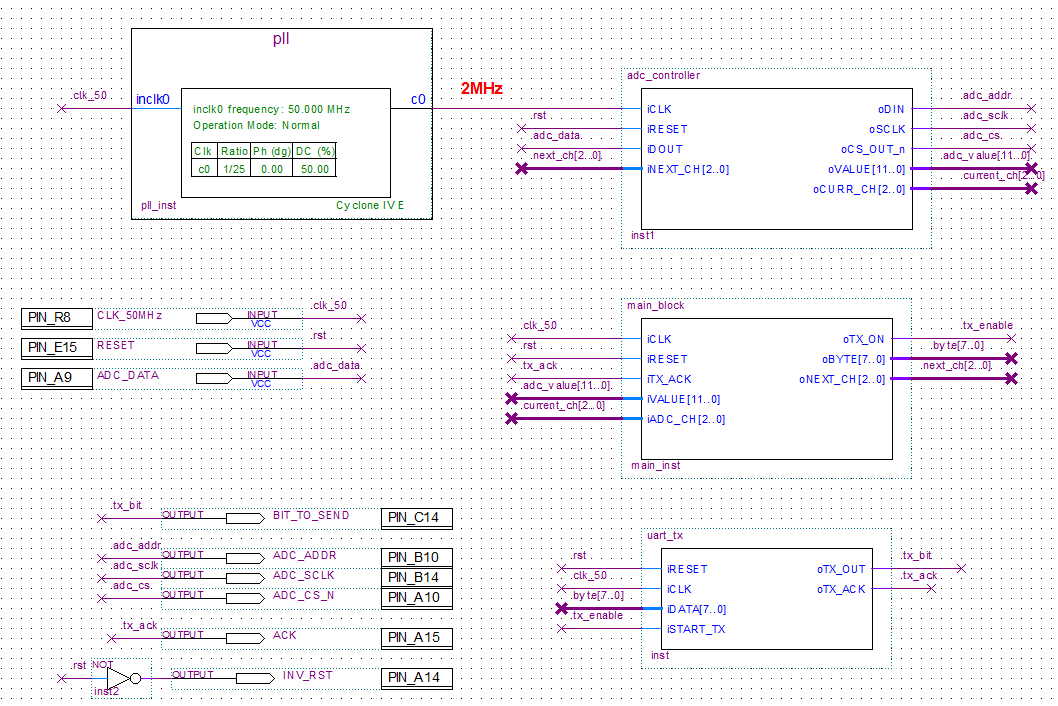
El resultado del test ha sido el correcto. Cuando la señale *start* está activa, y la señal reset no está activa, se reciben en la aplicación *CoolTerm* del PC valores ascendentes.

Cuando se desactiva la señal *start* o cuando se activa la señal reset, no se recibe ningún dato.

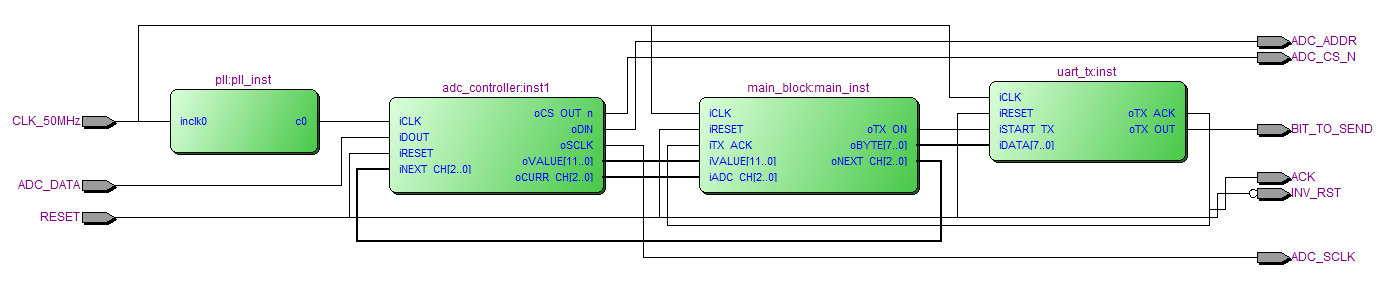
INSERTAR IMAGEN DEL HYPERTERMINAL

### Sistema completo

El módulo principal no ha sido testeado independientemente. Comprobado el correcto funcionamiento de los módulos UART y ADC, y antes los correctos resultados de la simulación mediante ModelSim, se ha procedido a su integración completa con el sistema.



A continuación, se muestra es sistema completo implementado en Quartus.



La asociación de pines de la FPGA es la siguiente.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Entradas | |  | Salidas | |
| Nombre | Pin |  | Nombre | Pin |
| ADC\_DATA | PIN\_A9 |  | ACK | PIN\_A15 |
| CLK\_50MHz | PIN\_R8 |  | ADC\_ADDR | PIN\_B10 |
| RESET | PIN\_E15 |  | ADC\_CS\_N | PIN\_A10 |
|  |  |  | ADC\_SCLK | PIN\_B14 |
|  |  |  | BIT\_TO\_SEND | PIN\_C14 |
|  |  |  | INV\_RST | PIN\_A14 |

*INV\_RST* es la negación de la señal de reset. Esta señal es conectada al LED externo, de esta manera se crea una señalización visual que indica si es sistema está en estado reset o está activo.

La señal *tx\_ack*, se conecta al LED0 para generar una señal visual cada vez que un byte es transmitido. Los datos se envían tan rápido, que no es fácil distinguir visualmente cuándo un dato ha sido transmitido, pero se ha considera que puede ser interesante como comprobación visual, o para futuros desarrollos.

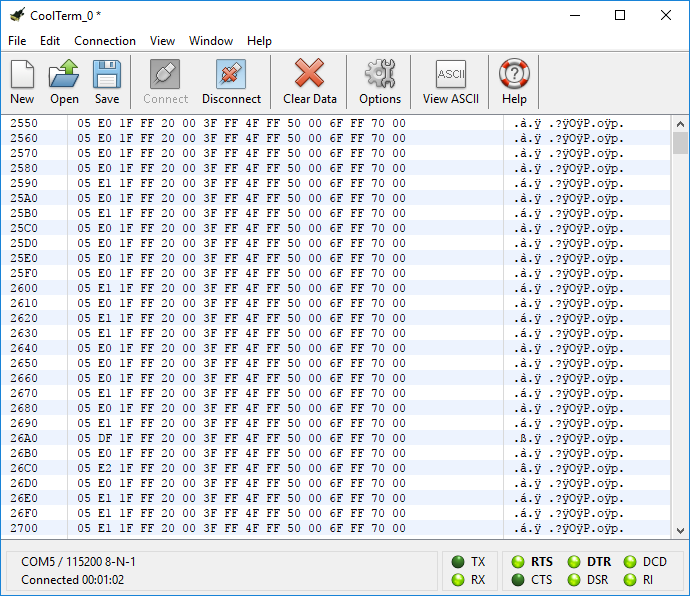
Inicialmente, el sistema completo se ha implementado en una protoboard. En ella se han realizado conexiones con la placa DE0-Nano, así como con la conexión serie hacia el PC.

En primer lugar, se ha testeado el sistema usando un potenciómetro como generador de señal analógica. En este punto, la conexión serie se realiza mediante cable, ya que lo principal es asegurar que el sistema probado hasta ahora, sigue funcionando una vez conectados todos los módulos.

A continuación, se muestra una tabla con las conexiones de las entradas analógicas del ADC128S022.

|  |  |
| --- | --- |
| Entrada analógica | Señal |
| IN0 | Potenciómetro |
| IN1 | VCC |
| IN2 | GND |
| IN3 | VCC |
| IN4 | VCC |
| IN5 | GND |
| IN6 | VCC |
| IN7 | GND |

Los resultados obtenidos en la aplicación terminal (*CoolTerm*) son los siguientes.



Como se ha explicado con anterioridad, los datos de un canal son representados en dos bytes Siendo el primer bit siempre 0, los siguientes 3 bits representan el canal (ver tabla X.X).

Se comprueba que los datos recibidos corresponden a las conexiones realizadas.

|  |  |  |
| --- | --- | --- |
| Terminal | Canal  (3 bits) | Valor  (12 bits) |
| 05E0 | 0 | 5E0 |
| 1FFF | 1 | FFF |
| 2000 | 2 | 000 |
| 3FFF | 3 | FFF |
| 4FFF | 4 | FFF |
| 5000 | 5 | 000 |
| 6FFF | 6 | FFF |
| 7000 | 7 | 000 |

Integración con conexión Bluetooth

Tras la comprobación del correcto funcionamiento del sistema completo, se ha procedido a sustituir la comunicación serie a través de la placa RS232, por el módulo Bluetooth HC-05.

En el apartado X.X se explica cómo se ha de configurar este módulo. Se han configurado dos módulos diferentes para poder intercambiar las velocidades de manera más rápida. En este sistema las configuraciones han sido las siguientes:

* NAME: BL\_01
* UART: 115200, 1, 0 (bauds, StopBit, Parity)
* Modo esclavo
* NAME: BL\_02
* UART: 38400, 1, 0 (bauds, StopBit, Parity)
* Modo esclavo

La conexión entre HC-05 y PC se realiza sin ningún problema y se reciben datos en el terminal de la misma manera que se hace con la conexión a través de cable.



Integración con aplicación de escritorio

Como se ha explicado en otro apartado, en Octave se ha creado un script muy simple, que sirve de basa para futuros desarrollos sobre esta herramienta. El código en Python ofrece más opciones, por lo que en a partir de este punto se tratará sólo este script.

Los datos recibidos son guardados en el PC en ficheros “.emgdat”. Estos ficheros se crean bajo la ruta especificada por el usuario en una carpeta creada por el script, la cual tiene por nombre la fecha y hora de la recogida de datos con el siguiente formato. Los ficheros a su vez, al igual que la carpeta creada, tienen por nombre la fecha y hora, más el canal al que corresponden los datos. El formato es el siguiente:

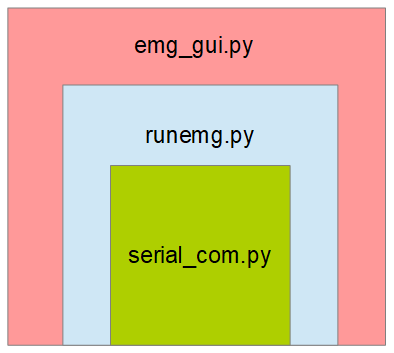
*RutaUser/AAAAMMDDHHMMSS/AAAAMMDDHHMMSS \_emg\_chX.emgdat*

El script permite la visualización de gráficas en tiempo real, así como visualizar gráficas desde fichero compatibles (*.emgdat*) almacenados en el PC. Los

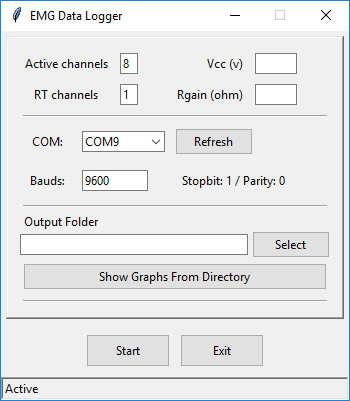
El código en Python se ha divido en tres scripts diferentes:

* serial\_com.py: capa encargada de la recepción de los datos y su posterior almacenaje en los ficheros *“.emgdat”*
* runemg.py: capa para representación gráfica de los datos.
* emg\_gui.py: interfaz gráfica para introducir los datos necesarios para la recepción de datos y su representación

Se podría decir que el código tiene tres niveles o capas. Cada capa para funcionar, necesita a las que se encuentren por debajo suyo, es decir, *emg\_gui* para poder trabajar necesita de *runemg,* y esta a su vez necesita de *serial\_com*. Pero gracias a esta diferenciación de capas, *serial\_com* puede ser ejecutada sin necesidad de ninguna otra, y *runemg* sólo necesita de *serial\_com* para poder ser lanzada.



La aplicación escritorio aporta una interfaz más amigable para la configuración de la conexión serie, y la representación y almacenaje de los datos recibidos.



A continuación, se explican las posibilidades de configuración que ofrece esta interfaz.

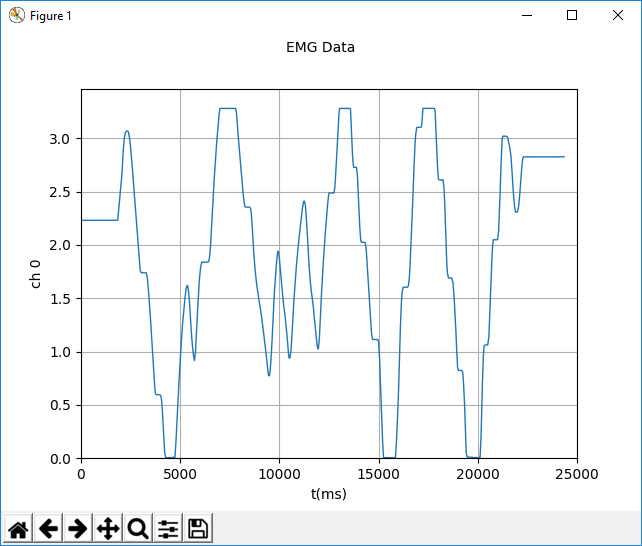
* Active channels: número de canales de los cuales se quiere almacenar sus datos. Nunca podrá ser menor que 1.
* RT channels: canales a mostrar en tiempo real. Su valor nunca podrá ser menor que 1 ni menor que el número de canales activos.
* Vcc: tensión de referencia de los sensores. Si la casilla se deja en blanco, o se escribe 0, se toma como referencia la resolución del ADC, en este caso, de 0 a 4095.
* Rgain: valor de la resistencia de ganancia del sensor MyoWare. Al escribir el valor de la resistencia de ganancia (en ohmnios), los datos mostrados y almacenados serán el voltaje real del músculo. Si se deja en blanco o se escribe 0, la ganancia será 1.
* COM: lista de los puertos series disponibles. El botón “*Refresh*” actualiza la lista.
* Bauds: velocidad de la comunicación con el puerto serie.
* Output Folder: path donde se guardarán los ficheros “.emgdat”. Es posible escribir la ruta directamente en la casilla en blanco, o se puede buscar a través de un navegador pulsando “Select”.
* Show Graphs From Directory: seleccionar la ruta donde se encuentran los ficheros “*.emgdat*”. Se mostrarán las gráficas de todos los ficheros compatibles existentes en dicha ruta.

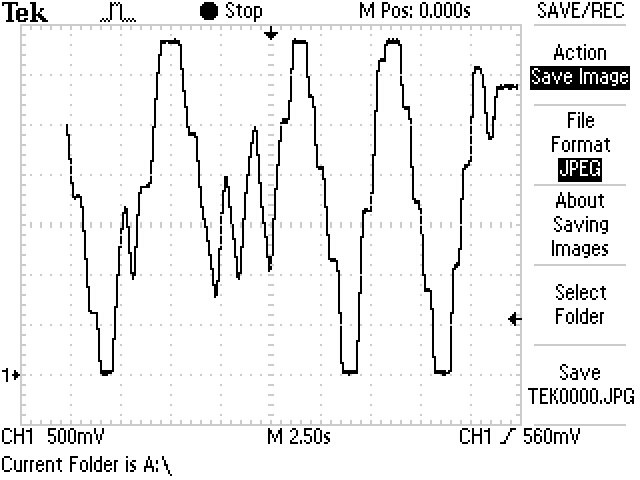
En el Anexo X.X se explica cómo utilizar el script sin la interfaz gráfica.

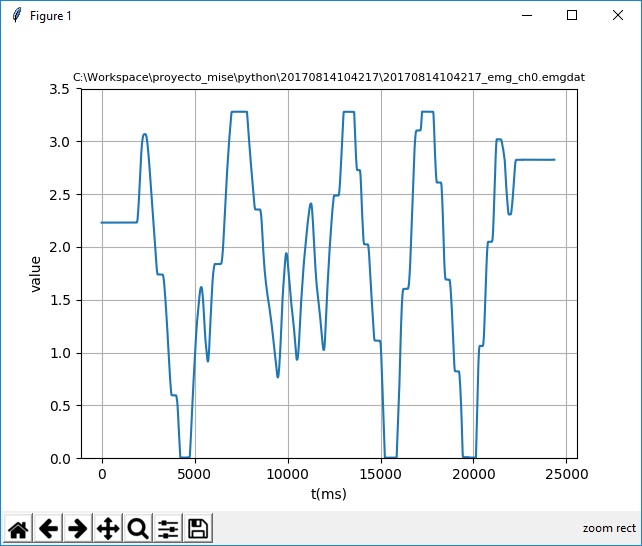
Se ha realizado un prueba con un potenciómetro conectado al canal 0, para comprobar que los datos mostrados en tiempo real y la gráfica generada desde fichero “*.emgdat*” coinciden con la señal medida en el osciloscopio.

La configuración ha sido la siguiente:

* Vcc: 3.3
* Rgain: -
* Bauds: 9600





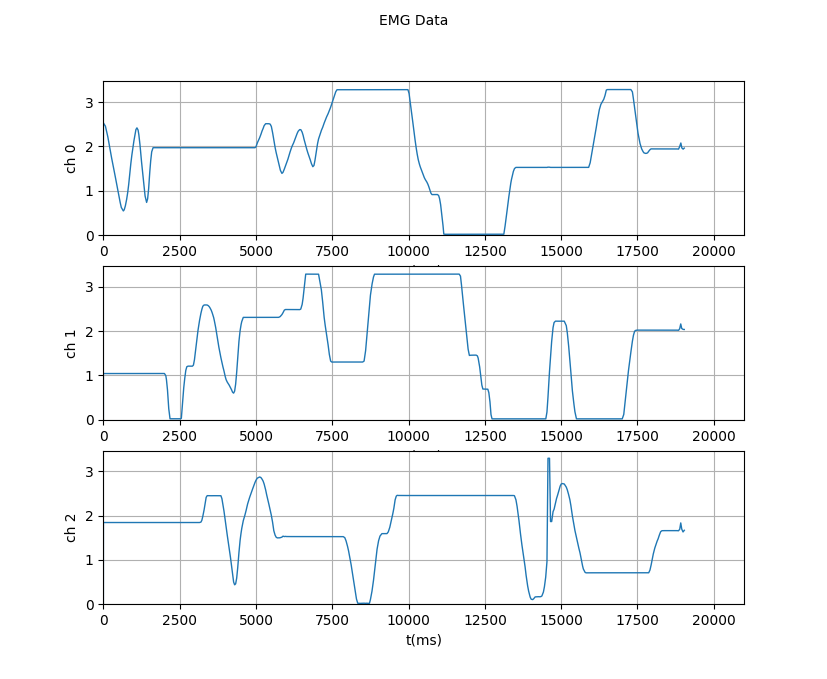


Se comprueba que todas las gráficas son correctas, por lo que se da por validada la aplicación de escritorio.

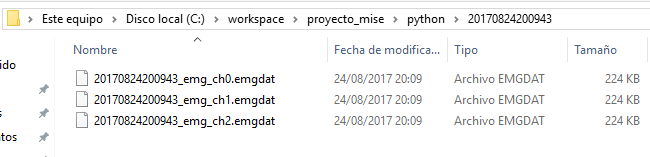
Una vez probado el correcto funcionamiento de todo el sistema hasta este punto, se han incorporado otros dos potenciómetros para hacer un total de 3. Con esta nueva configuración, se vuelve a probar el sistema para comprobar que el muestreo de todos los canales se hace correctamente.

La configuración ha sido la siguiente:

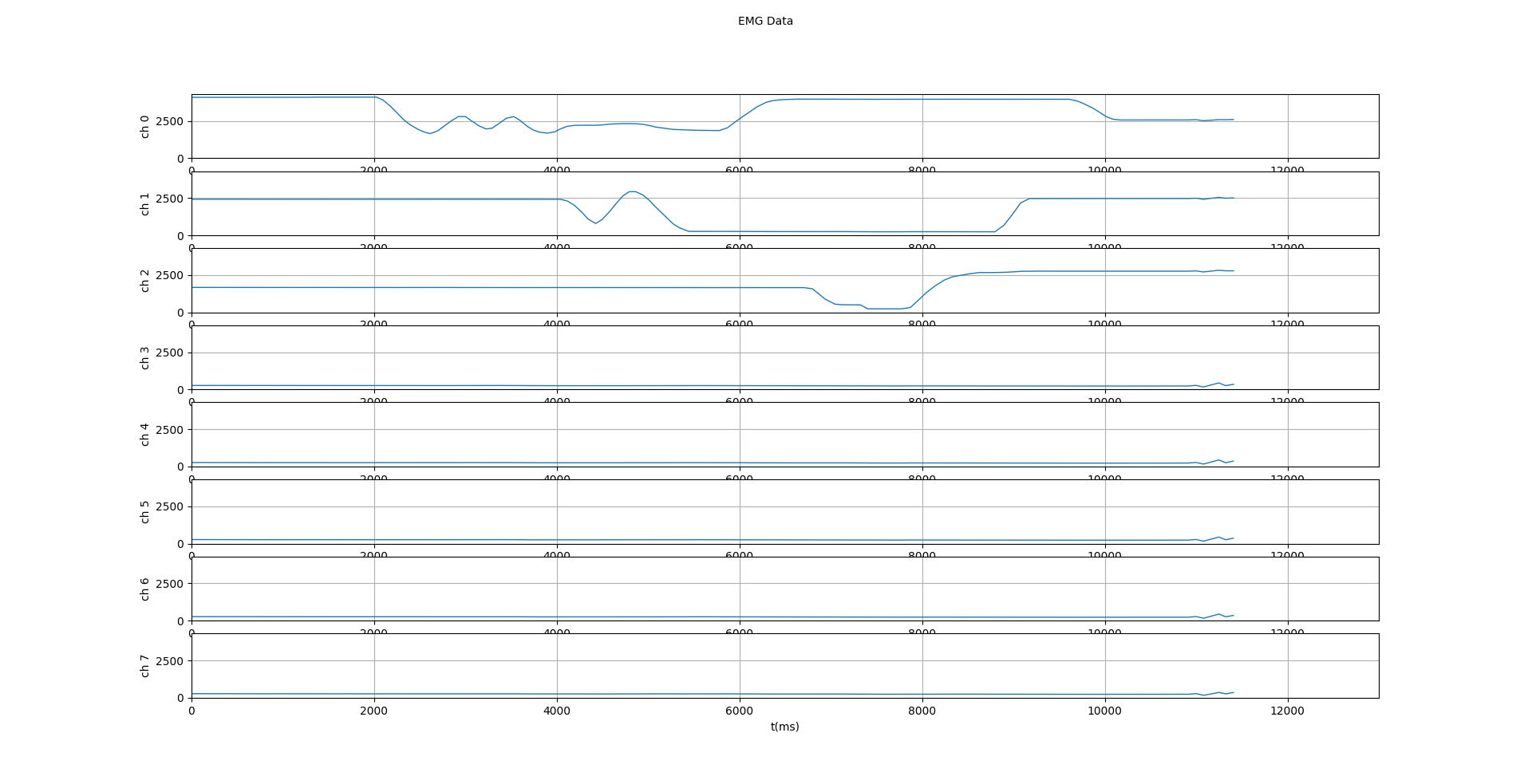
* Active channels: 3
* RT channels: 3
* Vcc: 3.3
* Rgain: -
* Bauds: 38400



Los ficheros “.*emgdat*” son creados correctamente.



Cabe destacar que se han probado diferentes combinaciones de canales con conexión a los potenciómetros. El resultado ha sido correcto en todos los casos.

También se ha probado a mostrar los 8 canales en tiempo real, pero la representación es muy lenta debido a que se tienen que dibujar todos los puntos y mantener todos esos puntos en la gráfica.

Integración placa extensión  
FALTA ESTA PARTE

### Resultados

Se han ido integrando todas las partes del sistema poco a poco y se ha comprobado que todo funciona correctamente. Este es el momento de integrar los sensores MyoWare y observar que todo funciona correctamente.

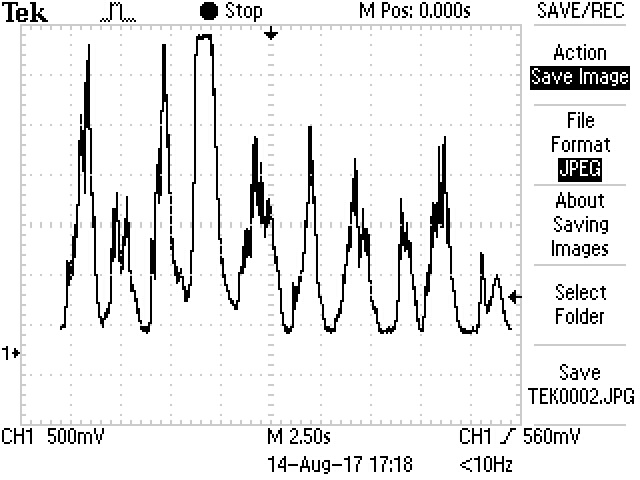
Se ha conectado un sensor en el bíceps izquierdo. Tras varias pruebas, se ha ajustado el potenciómetro de manera que el sistema funciona sin saturaciones.

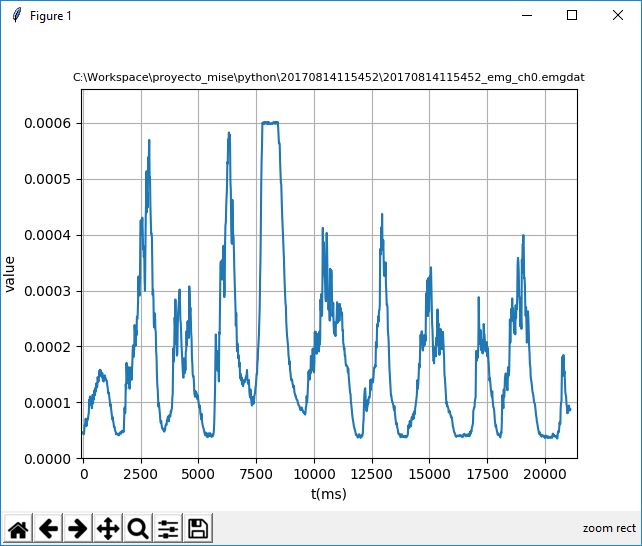
La configuración es la siguiente:

* Active channels: 1
* RT channels: 1
* Vcc: 3.3
* Rgain (ohms): 25820
* Bauds: 38400

Para este test, se ha proporcionado el valor de Rgain en la interfaz, de esta manera, se obtiene el voltaje real medido en el músculo mediante la fórmula de la ganancia del sensor MyoWare.







Las figuras X muestran los datos obtenidos de la medición de la contracción del bíceps izquierdo. Se puede comprobar que el valor máximo obtenido es el calculado previamente (600µV aprox.).

Se observa una saturación alrededor de los 7500 ms. Esta saturación se debe a la inestabilidad de los propios cables de conexión. A diferencia de los demás picos, este no tiene pequeñas subidas y bajas antes de llegar a su máximo.

MOSTRAR FICHERO CSV

## **PROBLEMAS**

Se han tenido muchos problemas de inestabilidad con las señales. Esto puede ser debido a los cables o malas conexiones entre los componentes. También puede ser debido a que la placa DE-nano no es capaz de suministrar la potencia suficiente para alimentar correctamente a todos los elementos externos (sensores, módulo HC-05, etc.).

Otro de los grandes problemas ha sido la constante pérdida de la señal Bluetooth del ordenador. En este caso, el desarrollo de este proyecto se ha realizado en ordenadores sin un módulo Bluetooth ya incorporado, por lo que se ha realizado conectando un USB con conexión Bluetooth. La calidad de este dispositivo es bastante dudosa.

## **FUTUROS DESARROLLOS**

Cambiar la velocidad de la UART dinámicamente a través de los DIP-SWITCH incorporados en DE0-Nano. De esta manera no es necesario volver a cargar el programa con la velocidad desea.

Alimentar los sensores a través de alimentación independiente de DE0-Nano, a través de pilas.

Evitar la alimentación a través de cable USB.

Comunicación con APP Smartphone o Tablet.

# CONCLUSIONES

# REFERENCIAS

# ANEXOS

## Anexo A: Simulaciones VHDL

Módulo control ADC

Módulo principal

Módulo trasmisión UART

## Anexo B: Manual de Interfaz gráfica

## Anexo C: Comandos AT

## Anexo D: Conexión PC con HC-05

## Anexo E: Conexiones de electrodos