

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 6](#_Toc490393500)

[1.1. DESCRIPCIÓN GENERAL DEL PROYECTO 6](#_Toc490393501)

[1.2. OBJETIVO DEL PROYECTO 6](#_Toc490393502)

[1.3. ENMARQUE DEL PROYECTO 7](#_Toc490393503)

[2. ESTADO DEL ARTE 8](#_Toc490393504)

[2.1. INGENIERÍA BIOMÉDICA 8](#_Toc490393505)

[2.1.1. Descripción 8](#_Toc490393506)

[2.1.2. Historia 8](#_Toc490393507)

[2.2. BIOSEÑALES 8](#_Toc490393508)

[2.2.1. Descripción 8](#_Toc490393509)

[2.2.2. Naturaleza 11](#_Toc490393510)

[2.2.3. Señales fisiológicas 11](#_Toc490393511)

[2.2.4. Tipos y clasificación 11](#_Toc490393512)

[2.3. ELECTROMIOGRAFÍA (EMG) 12](#_Toc490393513)

[2.3.1. Descripción 12](#_Toc490393514)

[2.3.2. Señales 12](#_Toc490393515)

[2.3.3. Digitalización 12](#_Toc490393516)

[2.3.4. Tipos de sensores comerciales 12](#_Toc490393517)

[2.4. FPGA 15](#_Toc490393518)

[2.4.1. Arquitectura 15](#_Toc490393519)

[2.4.2. Tecnologías 15](#_Toc490393520)

[2.5. COMUNICACIONES 15](#_Toc490393521)

[2.5.1. Serie (RS232/UART) 15](#_Toc490393522)

[2.5.2. Bluetooth 15](#_Toc490393523)

[3. DESCRIPCIÓN DEL DISPOSITIVO 16](#_Toc490393524)

[3.1. DE0 NANO 16](#_Toc490393525)

[3.1.1. Descripción general 16](#_Toc490393526)

[3.1.2. ADC 16](#_Toc490393527)

[3.1.3. Cyclone 16](#_Toc490393528)

[3.2. MYOWARE 16](#_Toc490393529)

[3.3. HC-05 (MÓDULO BLUETOOTH) 16](#_Toc490393530)

[4. HERRAMIENTAS DE DESARROLLO 17](#_Toc490393531)

[4.1. DESARROLLO FPGA 17](#_Toc490393532)

[4.1.1. ModelSim 17](#_Toc490393533)

[4.1.2. Quartus 19](#_Toc490393534)

[4.2. CAPTURA DE DATOS 22](#_Toc490393535)

[4.2.1. CoolTerm 23](#_Toc490393536)

[4.2.2. GNU Octave 24](#_Toc490393537)

[4.2.3. Python 25](#_Toc490393538)

[5. TRABAJO DESARROLLADO 28](#_Toc490393539)

[5.1. DESCRIPCIÓN GENERAL 28](#_Toc490393540)

[5.2. ESPECIFICACIONES 29](#_Toc490393541)

[5.2.1. Especificaciones ADC128S022 29](#_Toc490393542)

[5.3. DISEÑO 31](#_Toc490393543)

[5.3.1. Diseño sistema 31](#_Toc490393544)

[5.3.2. FPGA: Esquema general 35](#_Toc490393545)

[5.3.3. FPGA: módulo control ADC 36](#_Toc490393546)

[5.3.4. FPGA: módulo principal 38](#_Toc490393547)

[5.3.5. FPGA: módulo transmisión UART 40](#_Toc490393548)

[5.3.6. FPGA: PLL 43](#_Toc490393549)

[5.3.7. PC: Script captura de datos 43](#_Toc490393550)

[5.4. IMPLEMENTACIÓN 43](#_Toc490393551)

[5.4.1. Control de ADC 43](#_Toc490393552)

[5.4.2. Transmisión UART 44](#_Toc490393553)

[5.4.3. Sistema completo 45](#_Toc490393554)

[5.4.4. Resultados 49](#_Toc490393555)

[5.5. PROBLEMAS 49](#_Toc490393556)

[5.6. FUTUROS DESARROLLOS 49](#_Toc490393557)

[6. CONCLUSIONES 50](#_Toc490393558)

[7. REFERENCIAS 51](#_Toc490393559)

[ANEXO A 52](#_Toc490393560)

# 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 EO4CE22F17C6N) 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 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

### Historia

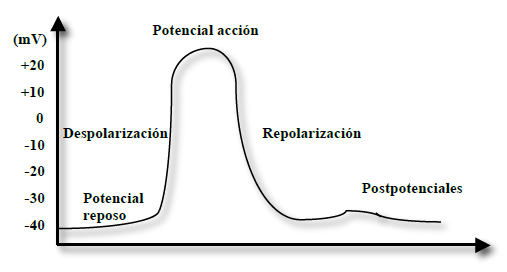
## **BIOSEÑALES**

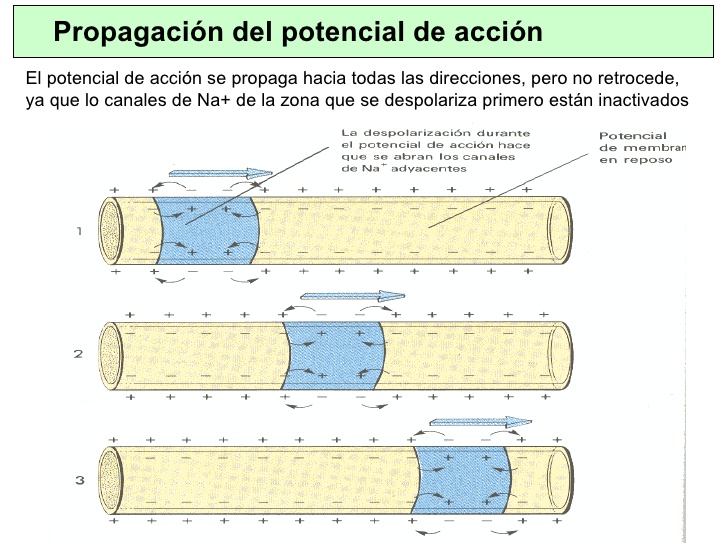
### Descripción

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 |

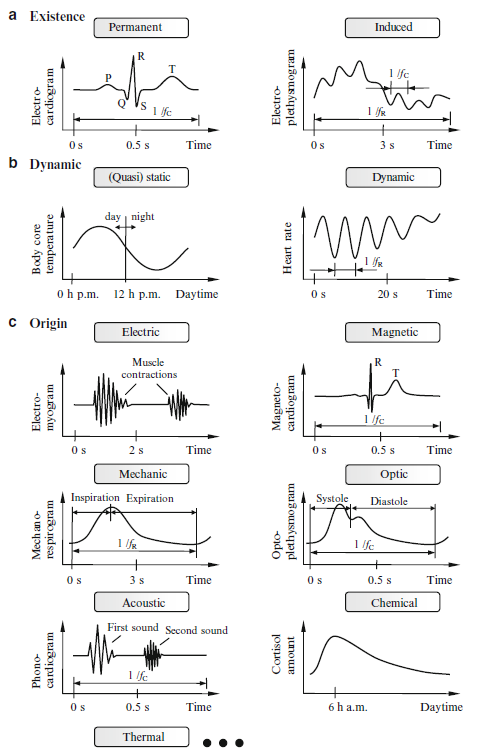




### Naturaleza

### Señales fisiológicas

### Tipos y clasificación

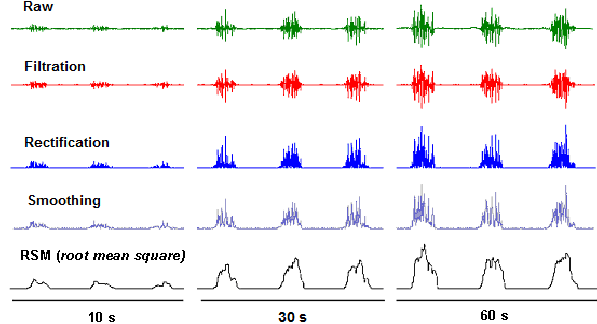


## **ELECTROMIOGRAFÍA (EMG)**

### Descripción

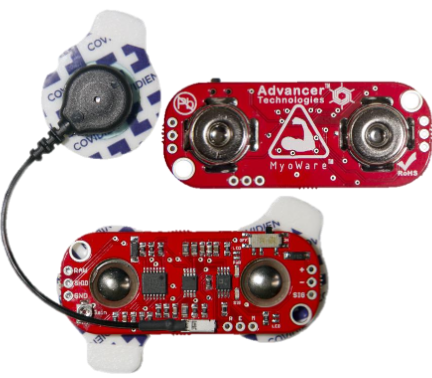
### Señales

### 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)

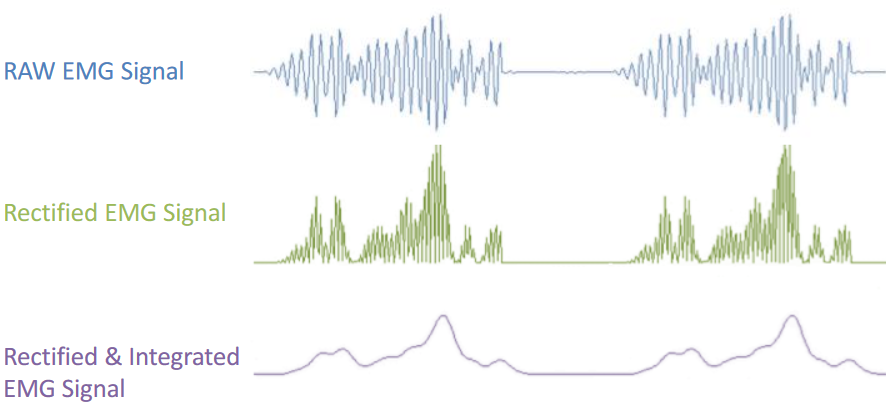
Se trata de un sistema embebido comercial, el cual integra todos los componentes necesarios para la adquisición y el tratamiento de la señal.

El tamaño es reducido y dispone de varios modos de obtención de señal:

* “en crudo”
* 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

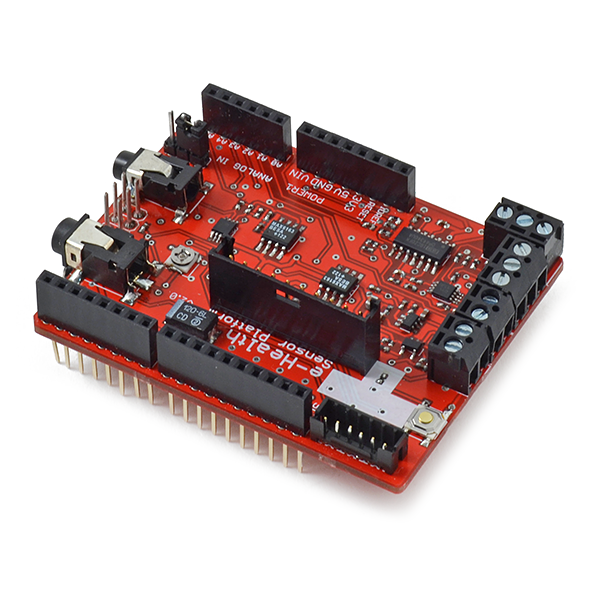
*Desventajas*

* 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

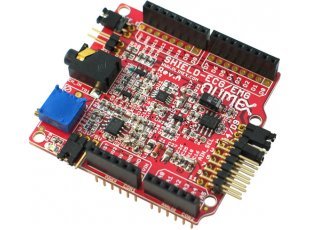
*Desventajas*

* 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 al ADC de la DE0-Nano.

*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 un protoboard, 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**

### Arquitectura

### Tecnologías

## **COMUNICACIONES**

### Serie (RS232/UART)

### Bluetooth

# DESCRIPCIÓN DEL DISPOSITIVO

## **DE0 NANO**

### Descripción general

### ADC

### Cyclone

## **MYOWARE**

## **HC-05 (MÓDULO BLUETOOTH)**

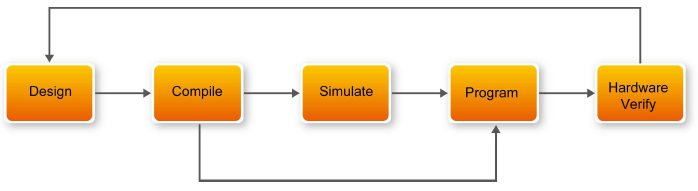
# HERRAMIENTAS DE DESARROLLO

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.

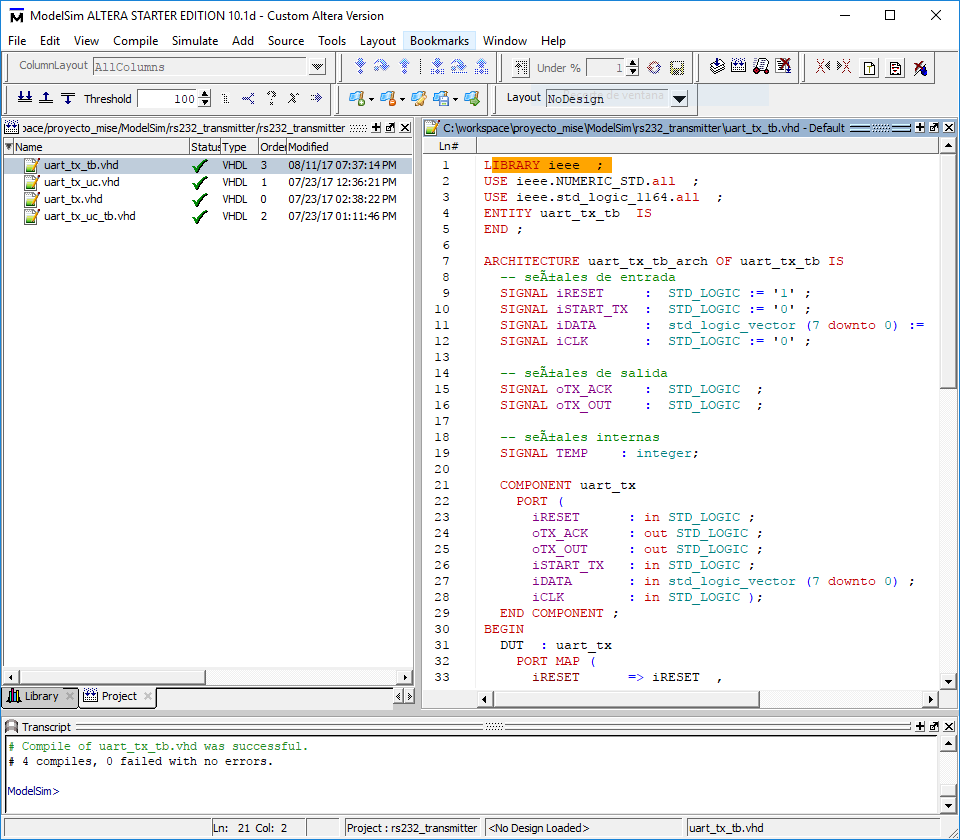
Una vez diseñado el circuito, se ha de compilar y programar en la FPGA para comprobar su correcto funcionamiento. Si bien la simulación del circuito diseñado no es un paso obligatorio, sí es altamente recomendable. Esto descarta posibles fallos inesperados del sistema.

### ModelSim

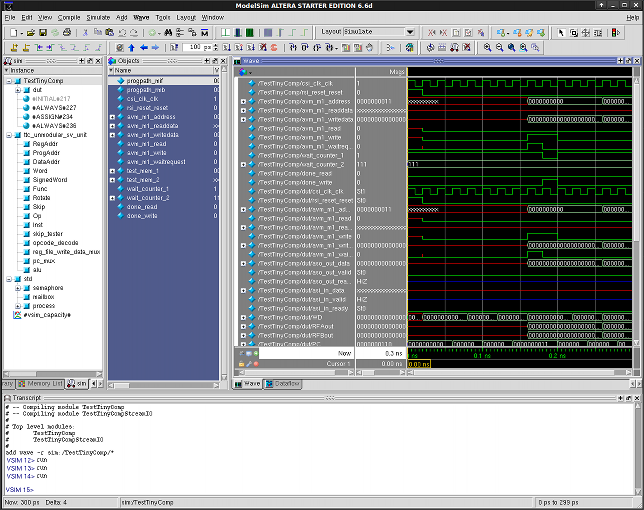
<https://www.mentor.com/products/fv/modelsim/> ¡! MOVER A REFERENCIAS

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>

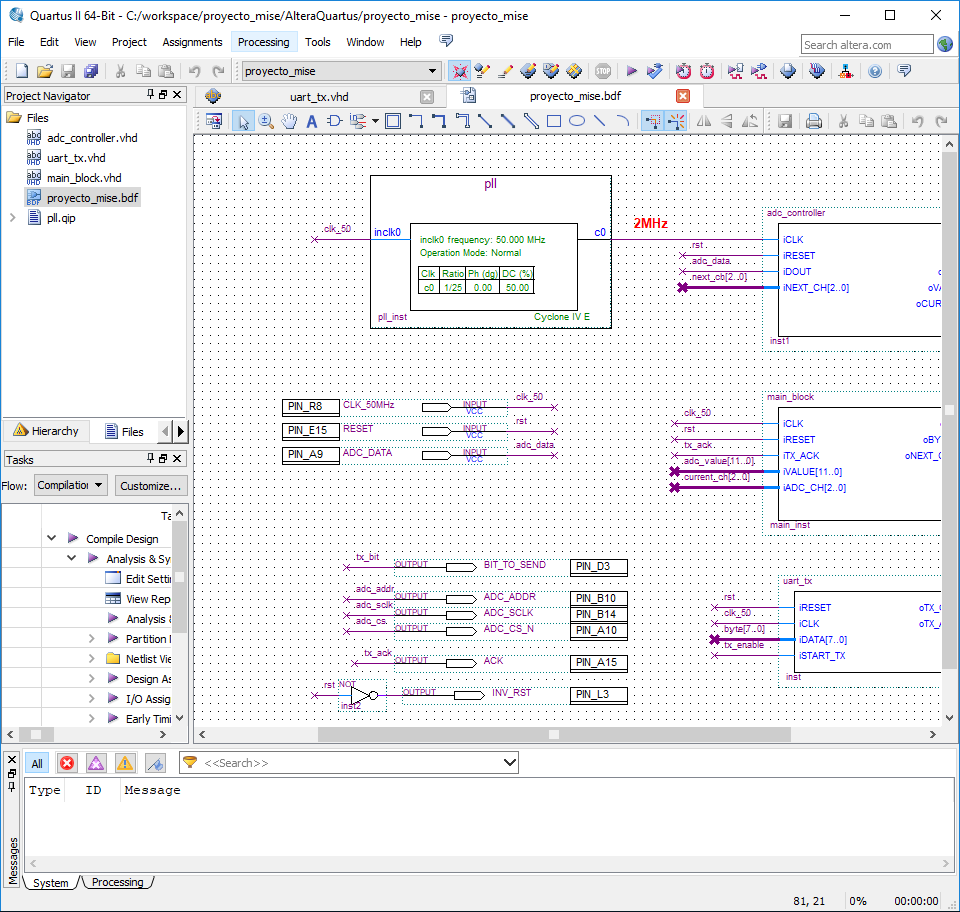
<https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/intro_to_quartus2.pdf>

¡! MOVER A REFERENCIAS

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

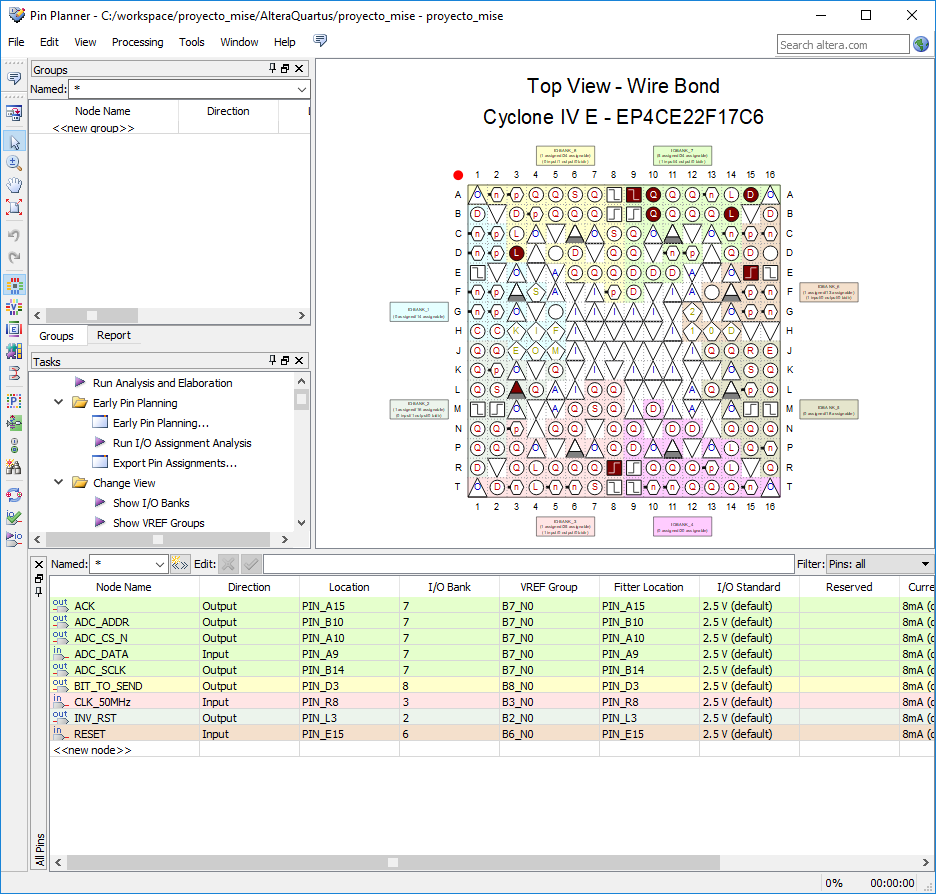
Quartus II es un software de diseño PLD para FPGAs de Altera, el cual es adecuado para diseños de FPGA de alta densidad, diseños PFGA de bajo coste, y diseños CPLD (*Complex Programmable Logic Devices*).

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.

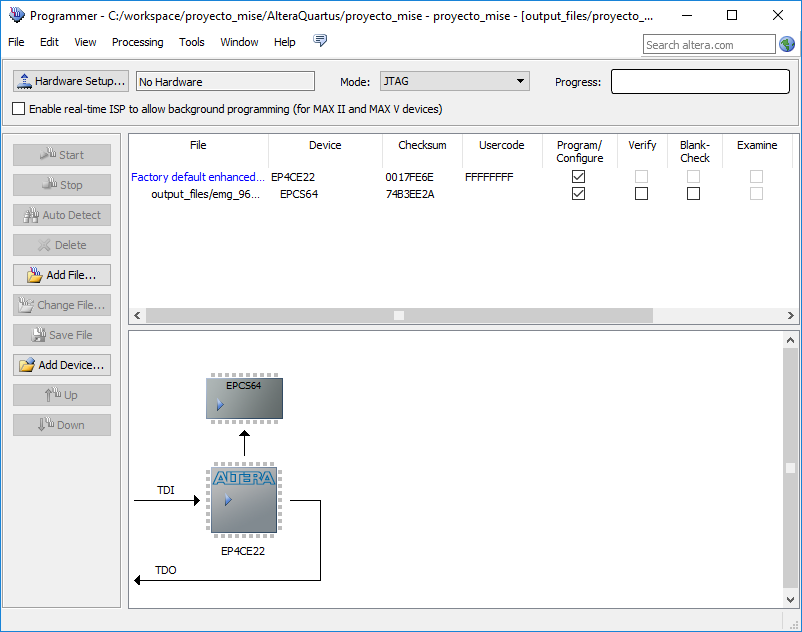


Una vez finalizado el diseño completo del sistema, se procede a compilar el sistema completo, lo cual genera un fichero “.sof”. Quartus realiza una serie de pasos y comprobaciones durante la compilación:

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.

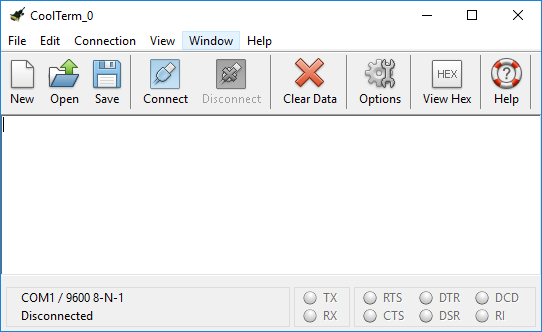
Se implementará también un pequeño script en Octave, al ser Matlab una herramienta muy utilizada para el procesado de datos y tareas matemáticas, alguien que tomará este proyecto para continuar una línea de estudio o trabajo, podría necesitar dicho script. La decisión de Octave sobre Matlab es el hecho 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.

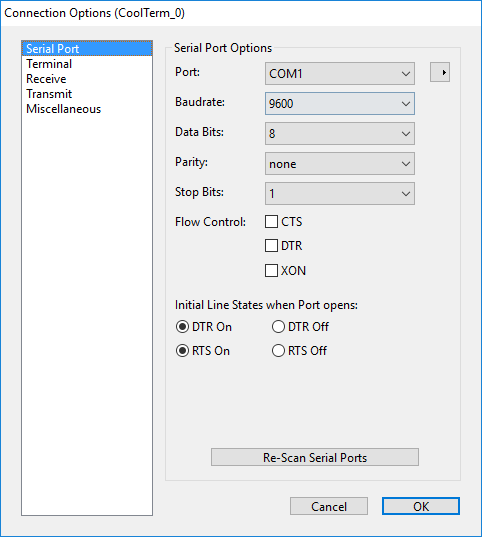
### CoolTerm

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

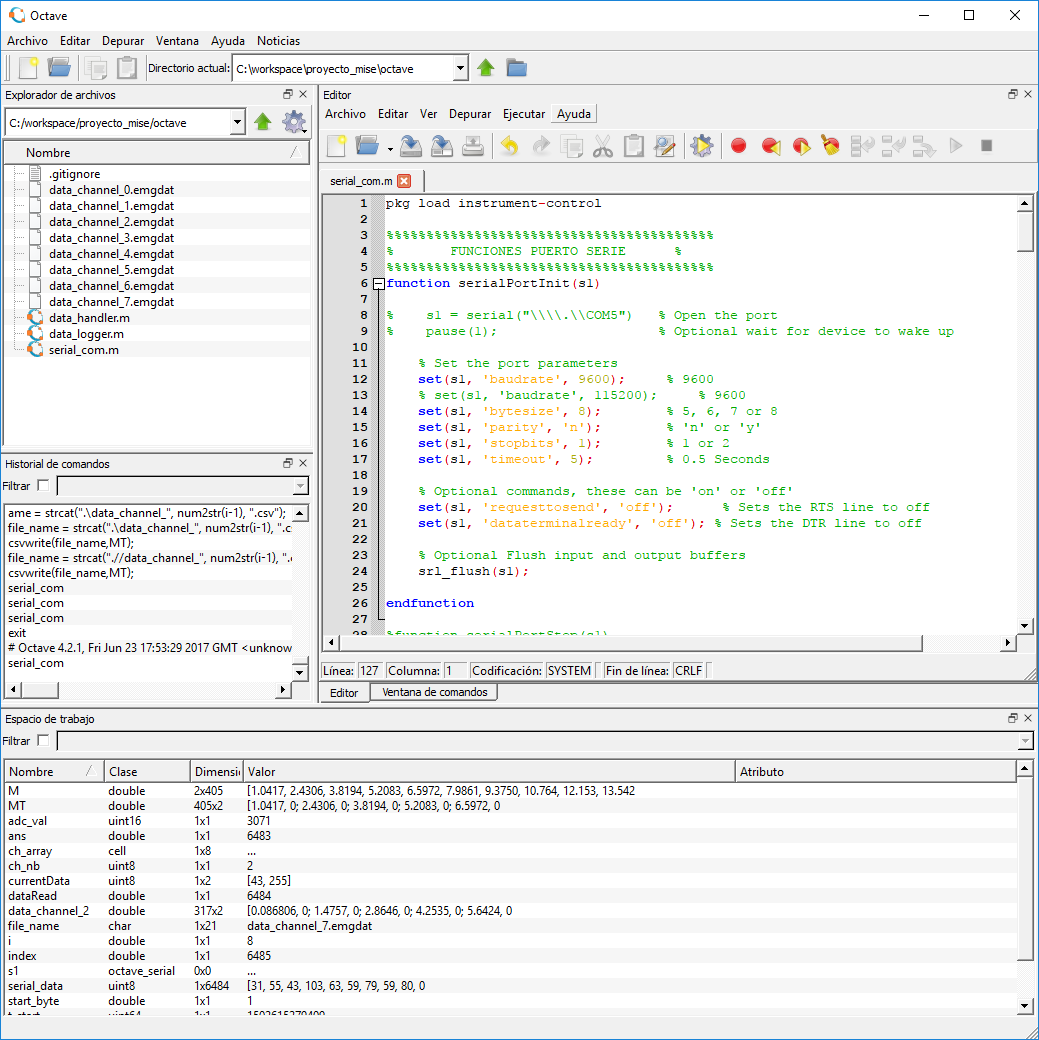
<https://www.gnu.org/software/octave/>

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.



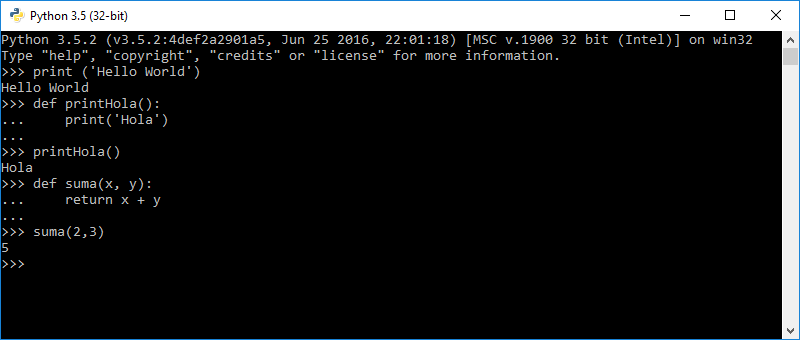
### Python

<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.

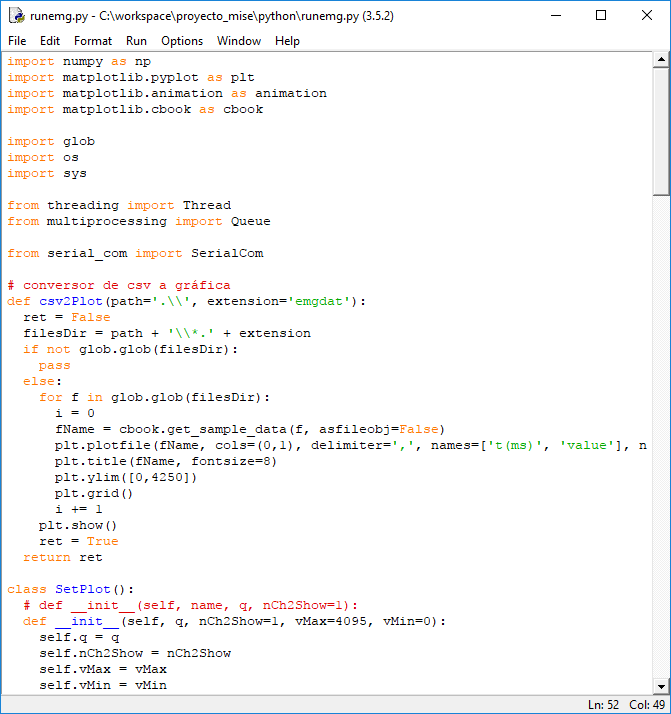
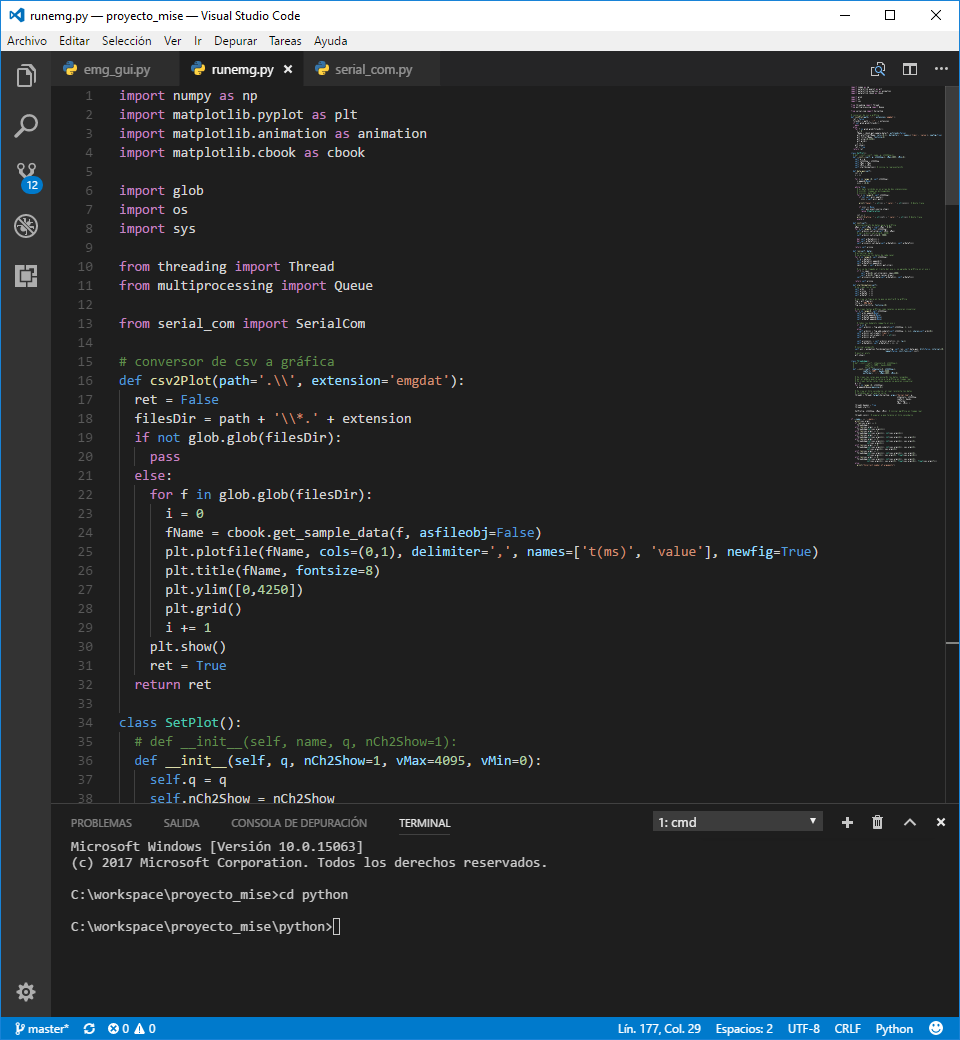
Es un lenguaje de muy alto nivel, muy simple de usar. Permite dividir el código en módulos para ser usados por otros programas Python. Python se distribuye con una amplia colección de módulos ya integrados para el uso básico de programas. Algunos de estos módulos proporcionan elementos como E/S de ficheros,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

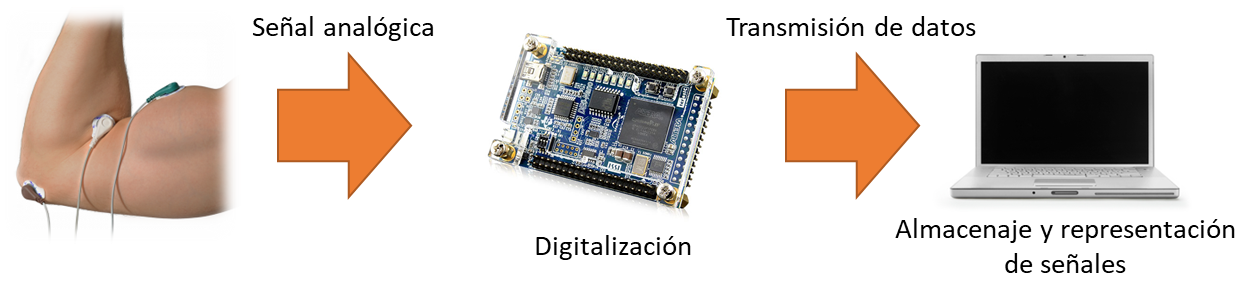
Python cuenta con una comunidad de desarrolladores muy extensa, lo que le da un gran soporte y hace que sea muy potente, ya que todas las librerías que se crean para Python son libres.

# 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.

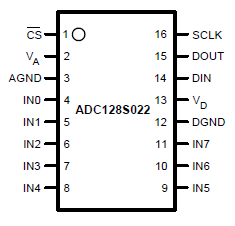
## **ESPECIFICACIONES**

### 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.



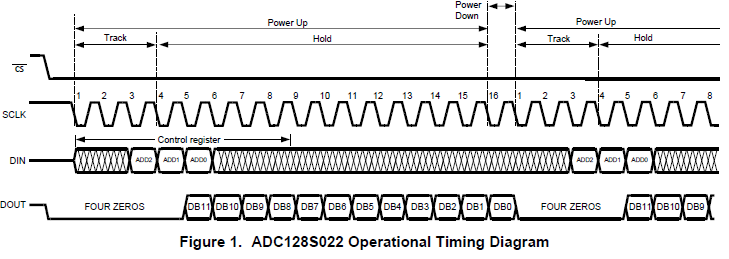
|  |  |  |  |
| --- | --- | --- | --- |
| PIN | | TIPO | DESCRIPCIÓN |
| NOMBRE | Número |
| CS\_n | 1 | Digital I/O | Chip select (negado). La conversión comienza con el flanco descendente de CS\_n. La conversión continua mientras CS\_n se mantenga en nivel bajo. |
| IN0 – IN7 | 4-11 | Analog I/O | 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:

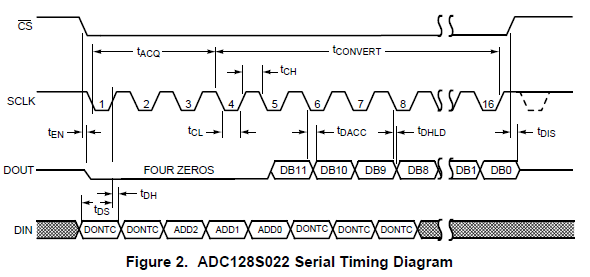
* Entradas: DOUT
* Salidas: CS\_n, DIN, SCLK.

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 en intermediario entre los dos anteriores. Es el único dotado con cierta inteligencia. Sus funciones son:
  + Enviar al “ADC Controller” en 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 (Startbit, 8bits de dato, Stopbit), y cada conversión supone el envío de dos bytes al PC.

La fórmula anterior describe el periodo de envío de datos, pero es interesante por 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)

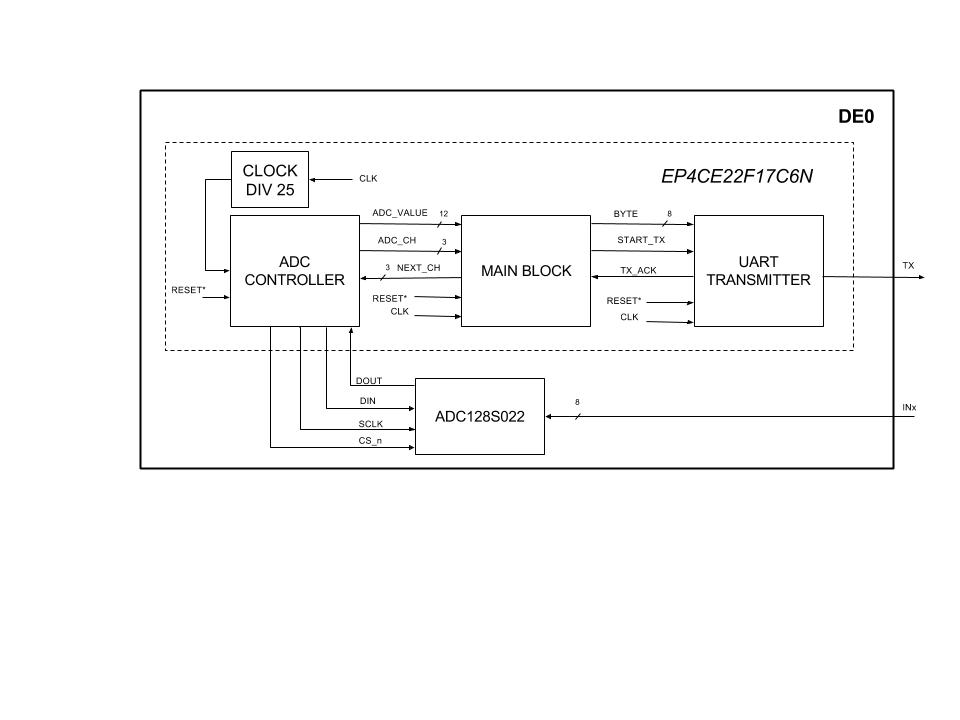
Se han considerado diferentes formatos en los cuales almacenar los datos. Tras un filtro previo, la decisión final quedó entre formato “excel” o “csv”. El problema que tiene “Excel” frente a “csv” es que su manejo a través de aplicaciones de escritorio o scripts (Matlab, Octave, Python…) es más completo, 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 “CSV” para cada canal. Esto supone que por cada sesión de trabajo se obtendrán 8 ficheros, con el nombre “data\_channel\_X.csv” (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.

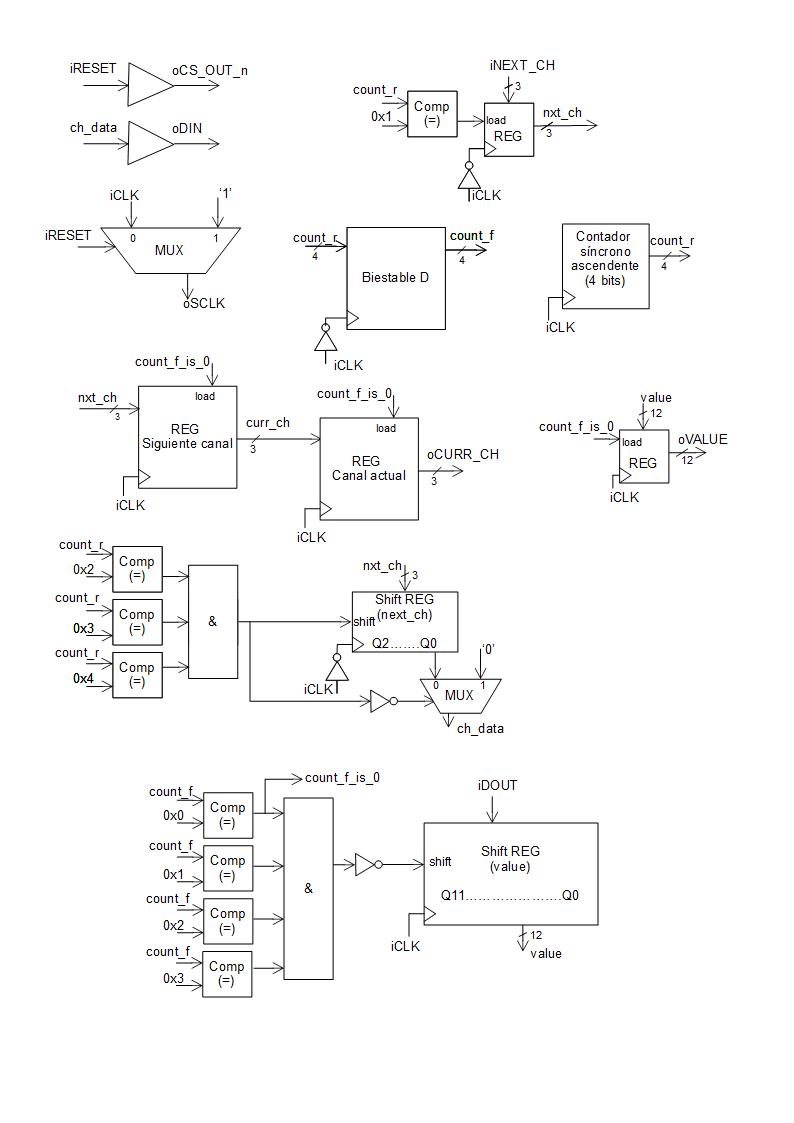
La UART envía los datos serie a través de *TX*.

### FPGA: módulo control ADC

SIMULACIONES MODELSIM AQUÍ O EN ANEXOS??

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.

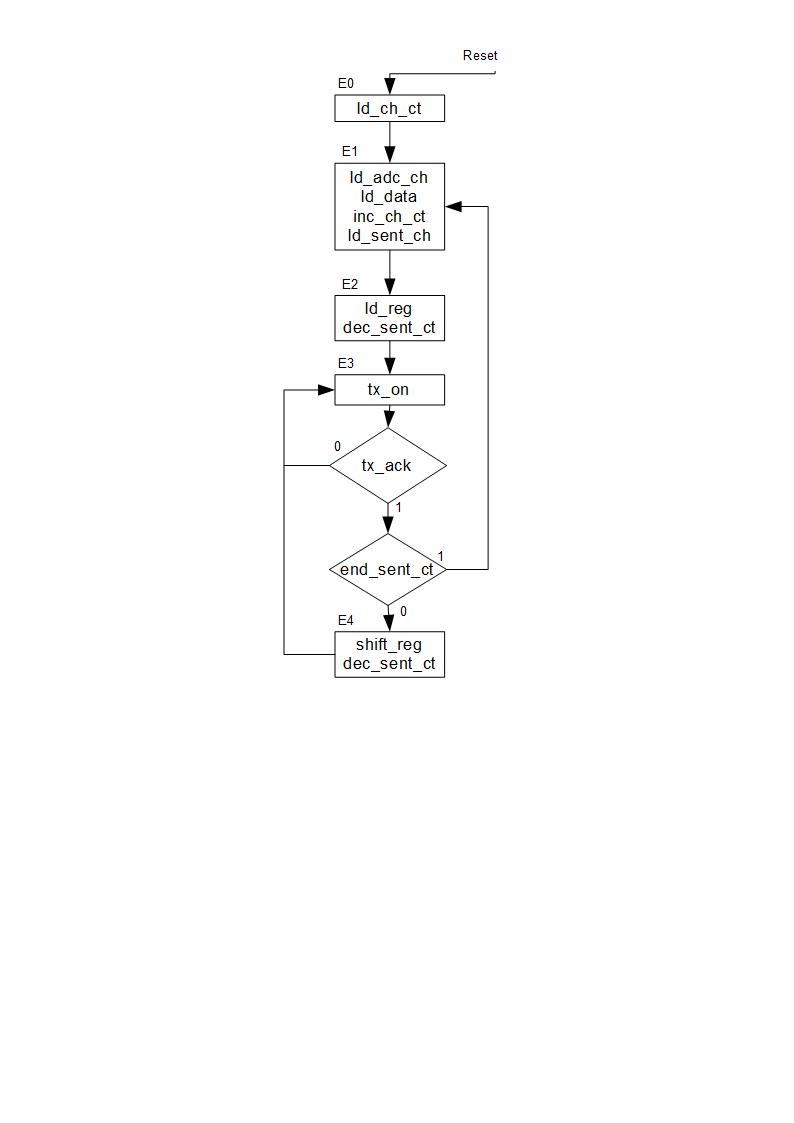
Los datos son almacenados en *value* durante cada flanco ascendente para no leer un dato durante su cambio.

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

### FPGA: módulo principal

SIMULACIONES MODELSIM AQUÍ O EN ANEXOS??

Unidad de control

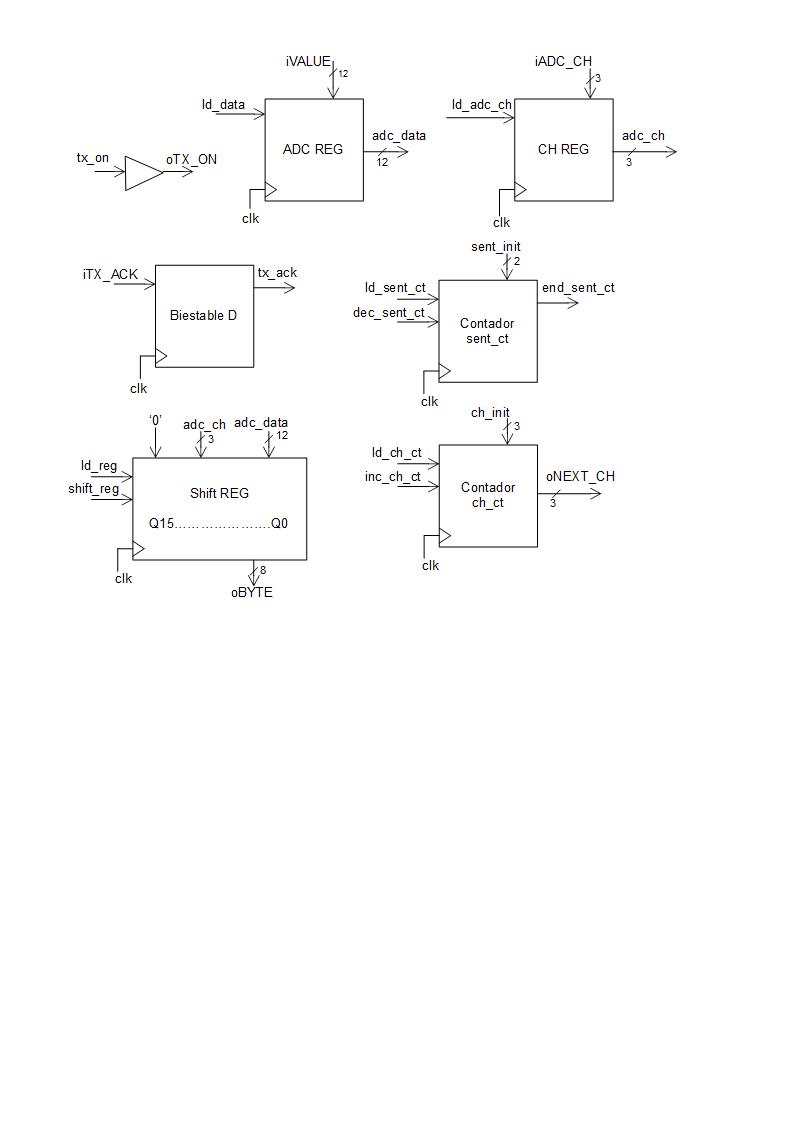


El contador de canal se inicia junto al inicio del sistema.

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*.

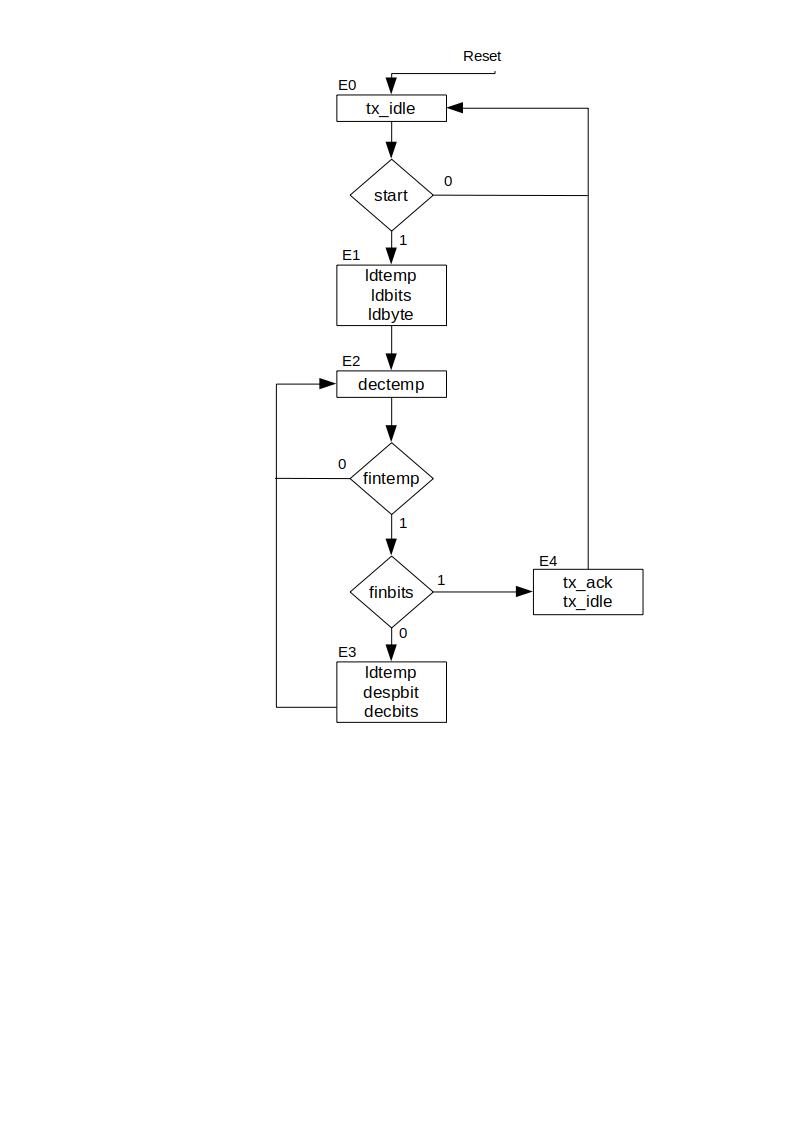
El registro de desplazamiento es de 16 bits. Este registro sirve para separar los 16 bits en dos bytes diferentes. Cuando la señal *shift\_reg* se activa, se desplaza un byte completo. Cada vez que la señal *ld\_reg* se activa, el registro carga la siguiente información:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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

SIMULACIONES MODELSIM AQUÍ O EN ANEXOS??

Unidad de control

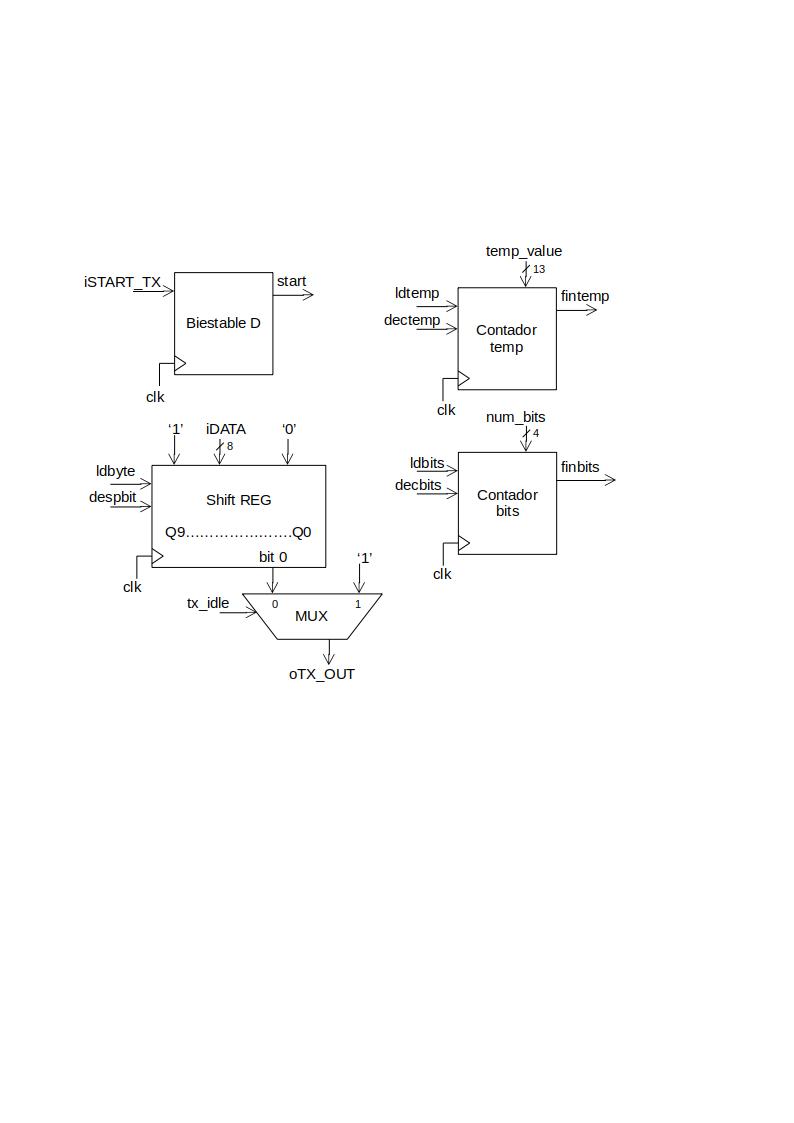


Mientras no reciba la señal de que *start* el módulo permanece en un estado idle(*tx\_idle* activado).

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.

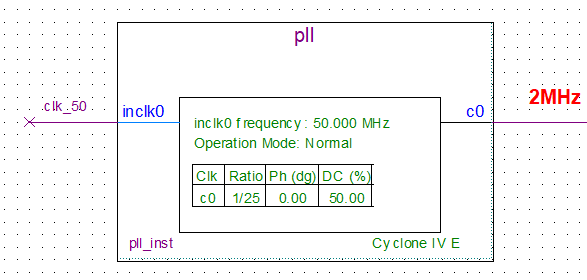
El contador descendente *temp* es el encargado de contar el tiempo entre pulsos. Se carga con el valor *temp\_value* cuando se activa la señal *ldtemp*. En cada pulso de reloj en el cual *dectemp* se encuentre activo, se decrementará el valor. Una vez alcanzado el valor ‘0’, se activa la señal *fintemp*.

*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

Diagrama de flujo

PASAR ESQUEMA DEL CUADERNO A ORDENADOR

## **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.

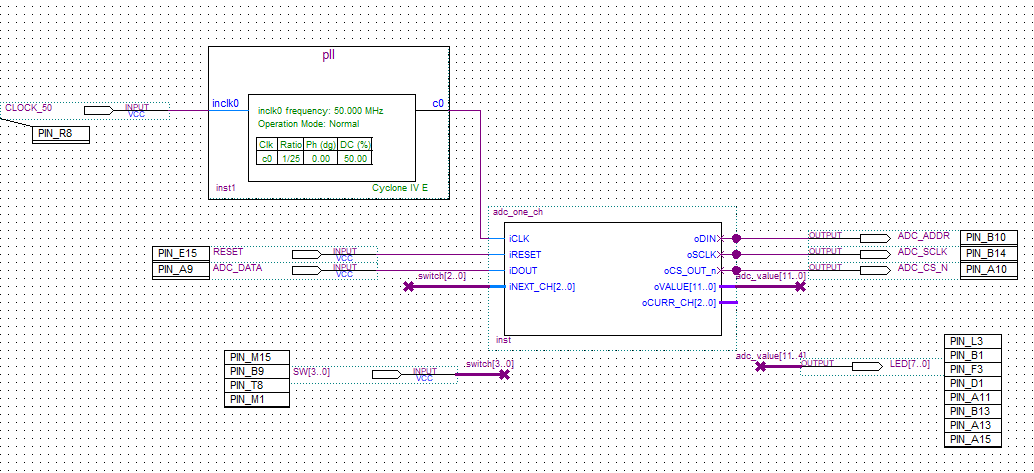
INSERTAR ESQUEMA Y FOTO

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.

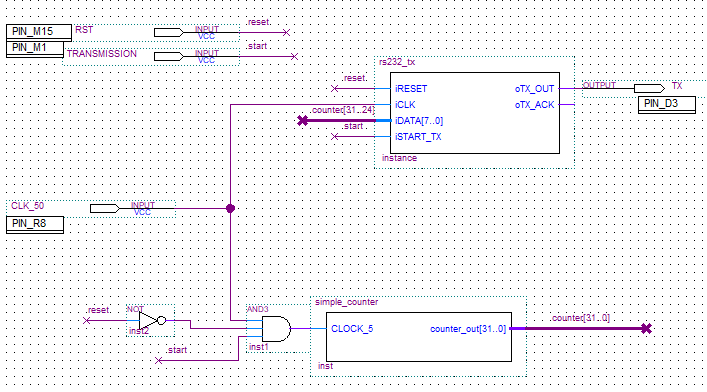
La señal *iSTART\_TX*, la cual indica que se puede comenzar la transmisión del dato, se simula mediante el switch 0 integrado en la placa DE-Nano.

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.



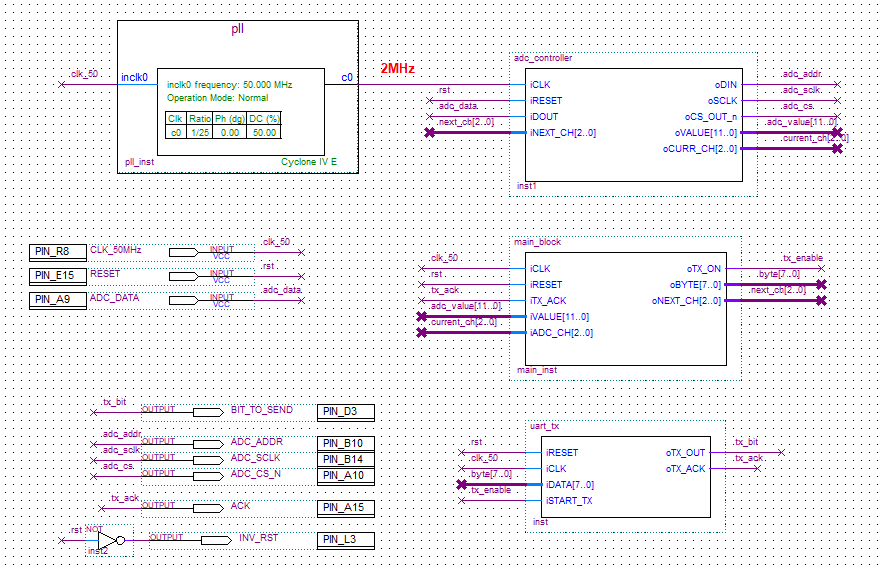
El resultado del test ha sido el correcto. Cuando la señale *start* está activa, y la señal reset no está activa, se reciben en la aplicación *CoolTerm* del PC valores ascendentes.

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

INSERTAR IMAGEN DEL HYPERTERMINAL

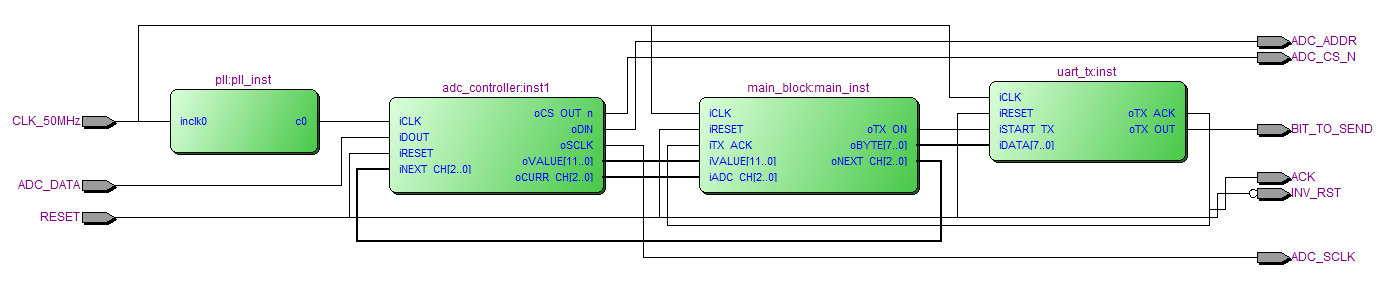
### Sistema completo

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\_D3 |
|  |  |  | INV\_RST | PIN\_L3 |



*INV\_RST* es la negación de la señal de reset. Esta señal es conectada al LED7 integrado en la placa DE0-Nano, 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.

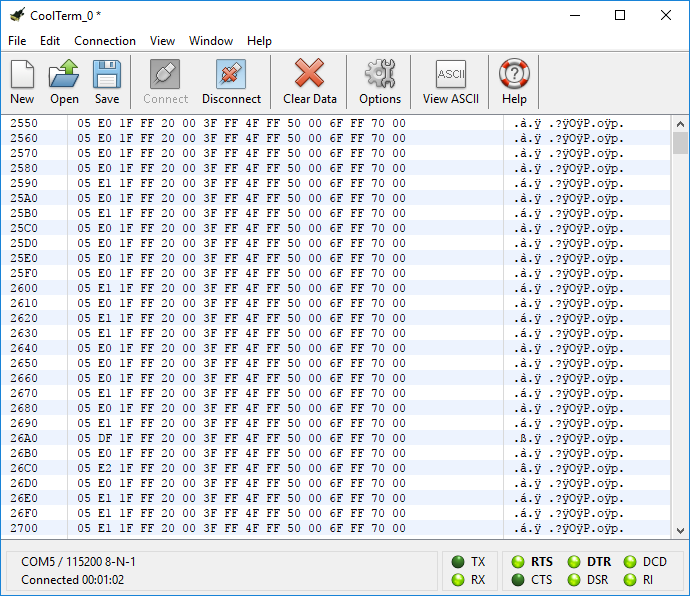
El sistema completo se ha implementado en una breadboard. En esta breadboard se han realizado conexiones con la placa DE0-Nano, así como con la conexión serie hacia el PC.

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.

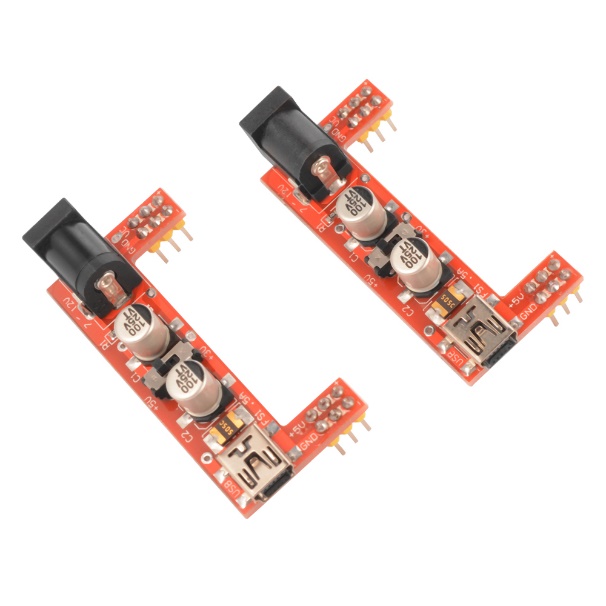
|  |  |  |
| --- | --- | --- |
| Terminal | Canal  (3 bits) | Valor  (12 bits) |
| 05E0 | 0 | 5E0 |
| 1FFF | 1 | FFF |
| 2000 | 2 | 000 |
| 3FFF | 3 | FFF |
| 4FFF | 4 | FFF |
| 5000 | 5 | 000 |
| 6FFF | 6 | FFF |
| 7000 | 7 | 000 |

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. En este sistema la configuración ha sido la siguiente:

* NAME: EMG\_BL
* UART: 9600, 1, 0 (bauds, StopBit, Parity)
* Modo esclavo

Llegados a este punto, hay que tener en cuenta que la placa DE0-Nano trabaja con 3,3V, y el módulo HC-05 a 5V. Debido a esto, el sistema completo necesita una tensión de entrada de 5V, para después ser capaz de obtener otra señal de 3,3V a partir de la primera. Esto es posible mediante un circuito regulador de tensión. En ese sistema, siendo un prototipo, se ha decido utilizar un regulador de tensión comercial, el cual aporta dos niveles de tensión diferentes: 5V y 3,3V.

ESQUEMA ELÉCTRICO Y FOTO

IMÁGENES DE LA CONEXIÓN BL EN EL PC

Integración con aplicación de escritorio

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

PANTALLAZOS DE LA APP DE ESCRITORIO Y EXPLICACIÓN

### Resultados

INSERTAR IMAGEN GRÁFICA PC Y DEL OSCILOSCOPIO Y ANALIZAR LAS FASES DE LA SEÑAL (POTENCIAL DE ACCIÓN, POLARIZACIÓN…)

MOSTRAR FICHERO CSV

## **PROBLEMAS**

## **FUTUROS DESARROLLOS**

# CONCLUSIONES

# REFERENCIAS

# ANEXO A