

**Universidad Carlos III de Madrid**

**Escuela Politécnica Superior**

**Departamento Tecnología Electrónica**



Trabajo de Fin de Grado

Grado en Ingeniería en Tecnologías Industriales

UART

UNIVERSAL ASYNCHRONOUS RECEIVER-  
TRANSMITTER

**Autor:** Borja Díaz Mulas

**Tutor:** Luis Mengibar Pozo





## AGRADECIMIENTOS

En primer lugar quiero dar las gracias a mis padres y a mi hermana por hacer posibles todos estos años de estudio.

También me gustaría dar las gracias a mi novia por todo el apoyo recibido durante la realización del proyecto.

Por último dar las gracias a mi tutor Luis Mengibar Pozo por ayúdame durante la realización del proyecto y por la libertad a la hora de desarrollarlo según mi criterio.



## RESUMEN

En este proyecto se ha desarrollado e implementado sobre FPGA un dispositivo electrónico conocido con el nombre de UART (*Universal Asynchronous Receiver-Transmitter*). Es un dispositivo hardware capaz de traducir datos con formato paralelo a formato serie con el objeto de permitir una comunicación entre sistemas electrónicos. Actualmente es un circuito ampliamente incluido en microcontroladores. Diseñaremos el dispositivo empleando el lenguaje de descripción de hardware VHDL. Para comenzar con el desarrollo será necesario un estudio de este tipo de comunicación serie y el análisis de una serie de dispositivos que ya incorporan la UART. Además emplearemos la UART del dispositivo PIC18F2525 de Microchip como ejemplo, descomponiéndolo en diferentes bloques y analizándolo detenidamente. Para conseguir realizar el diseño de forma satisfactoria será necesario incrementar nuestros conocimientos de VHDL y familiarizarnos con una serie de herramientas de diseño, principalmente el entorno de desarrollo de Altera. Una vez diseñado el dispositivo, se ejecutarán una serie de simulaciones por bloques, para comprobar el funcionamiento del dispositivo antes de implementar el diseño en una FPGA (*Field Programmable Gate Array*). Para comprobar el correcto funcionamiento del circuito diseñado, además de realizar simulaciones temporales (*post-layout*), se ha implementado este sobre una placa de desarrollo (DE0-Nano de Terasic) incluyendo la UART diseñada y un circuito de prueba que permite la comunicación con un PC.

## ABSTRACT

In this project we will develop an electronic device known by the name UART (Universal Asynchronous Receiver-Transmitter). We will design the device using a hardware description language called VHDL. To start with the development, a study of this type of serial communication and some existing devices will be necessary. In addition we will employ one of these devices as an example, breaking it down into different blocks and analyzing them carefully. To do so, it will be necessary to increase our knowledge of VHDL and become familiar with the Altera development environment. Upon the completion of the device, we will run simulations of the blocks to check the performance of the device before implementing the design on an FPGA (Field Programmable Gate Array). Finally, we will execute a practical example on a development board to demonstrate the correct design performance.



## ÍNDICE DE CONTENIDOS

<b>1 INTRODUCCIÓN</b>	<b>12</b>
<b>2 OBJETIVOS PROPUESTOS EN EL PROYECTO</b>	<b>13</b>
<b>3 ¿EN QUÉ CONSISTE LA COMUNICACIÓN SERIE?</b>	<b>15</b>
<b>3.1 CLASIFICACIÓN POR TIPO DE SINCRONÍA</b>	<b>15</b>
<b>3.2 CLASIFICACIÓN POR EL SENTIDO DE LA COMUNICACIÓN</b>	<b>16</b>
<b>4 TIPOS DE COMUNICACIÓN SERIE</b>	<b>17</b>
<b>4.1 I2C <i>INTER-INTEGRATED CIRCUIT (INTER-CIRCUITOS INTEGRADOS)</i></b>	<b>17</b>
<b>4.2 SPI (SERIAL PERIPHERAL INTERFACE)</b>	<b>18</b>
<b>4.3 COMUNICACIÓN SERIE SÍNCRONA (USART)</b>	<b>19</b>
<b>4.4 COMUNICACIÓN SERIE ASÍNCRONA (USART)</b>	<b>20</b>
<b>5 DISPOSITIVOS UART CON MAYOR RELEVANCIA</b>	<b>22</b>
<b>5.1 UART 8250</b>	<b>22</b>
<b>5.2 UART 16550</b>	<b>23</b>
<b>6 UART (UNIVERSAL ASYNCHRONOUS RECEIVER-TRANSMITTER). COMUNICACIÓN SERIE ASÍNCRONA.</b>	<b>25</b>
<b>6.1 UART. DEFINICIÓN</b>	<b>25</b>
<b>6.2 TRANSMITIR Y RECIBIR DATOS EN FORMATO SERIE</b>	<b>25</b>
<b>6.2.1 TRANSMISIÓN DE UN CARÁCTER</b>	<b>26</b>
<b>6.3 BLOQUES GENERALES QUE COMPONENT LA UART</b>	<b>27</b>
<b>6.3.1 RECEPTOR</b>	<b>27</b>
<b>6.3.2 TRANSMISOR</b>	<b>28</b>
<b>6.4 COMPONENTES BÁSICOS DE LA UART</b>	<b>28</b>
<b>6.5 CONTROL DE FLUJO</b>	<b>29</b>
<b>6.5.1 CONTROL DE FLUJO POR HARDWARE COMPLETO</b>	<b>29</b>
<b>6.5.2 CONTROL HARDWARE ÚNICAMENTE CON RTS Y CTS</b>	<b>31</b>
<b>6.5.3 CONTROL DE FLUJO POR SOFTWARE (XON- XOFF)</b>	<b>31</b>
<b>6.6 ESTÁNDARES DE COMUNICACIÓN SERIE</b>	<b>32</b>
<b>6.6.1 RS-232</b>	<b>32</b>
<b>6.6.2 RS-422</b>	<b>37</b>



6.6.3 RS-485	38
6.6.4 COMPARACIÓN ENTRE LOS ESTÁNDARES DE COMUNICACIÓN SERIE ASÍNCRONA MÁS REPRESENTATIVOS.	40
<b>6.7 VHDL</b>	<b>41</b>
<b>7 CARACTERÍSTICAS DE LA PLACA DE DESARROLLO</b>	<b>42</b>
<b>7.1 INTRODUCCIÓN</b>	<b>42</b>
<b>7.2 DISPOSITIVOS DE LA PLACA DE DESARROLLO</b>	<b>43</b>
7.2.1 FPGA	44
7.2.2 SISTEMAS DE CONFIGURACIÓN	45
7.2.3 PUERTOS DE EXPANSIÓN	46
7.2.4 DISPOSITIVOS DE MEMORIA	46
7.2.5 DISPOSITIVOS DE ENTRADA Y SALIDA	47
7.2.6 SENSOR G	49
7.2.7 CONVERTIDOR A/D	49
7.2.8 SISTEMA DE RELOJ	50
7.2.9 FUENTES DE ALIMENTACIÓN	50
<b>8 QUARTUS II</b>	<b>51</b>
<b>8.1 MODELSIM-ALTERA STARTER EDITION</b>	<b>52</b>
<b>9 DISPOSITIVO ELEGIDO PARA REALIZAR EL PROYECTO</b>	<b>53</b>
<b>9.1 DESCRIPCIÓN DE LA USART DEL MICROCONTROLADOR PIC18F2525</b>	<b>53</b>
<b>9.2 UART: USART EN MODO ASÍNCRONO</b>	<b>54</b>
9.2.1 GENERADOR DE BAUDIOS	55
9.2.2 TRANSMISOR ASÍNCRONO	56
9.2.3 RECEPTOR ASÍNCRONO	58
<b>10 DESCRIPCIÓN DEL DISPOSITIVO IMPLEMENTADO EN VHDL</b>	<b>61</b>
<b>10.1 BRG O BAUD RATE GENERATOR</b>	<b>62</b>
10.1.1 ENTIDAD	63
10.1.2 FUNCIONAMIENTO DEL GENERADOR DE BAUDIOS	63
10.1.3 PUERTOS Y SEÑALES	64
10.1.4 RESULTADOS DE LA SÍNTESIS	65
10.1.5 SIMULACIÓN RTL	65
10.1.6 SIMULACIÓN LÓGICA TEMPORAL A NIVEL DE PUERTAS	66

<b>10.2 TX_TRANSFER</b>	<b>67</b>
10.2.1 ENTIDAD	67
10.2.2 FUNCIONAMIENTO DEL TRANSMISOR TX_TRANSFER	67
10.2.3 PUERTOS Y SEÑALES	68
10.2.4 RESULTADOS DE LA SÍNTESIS	70
10.2.5 TRANSMISIÓN DE DOS CARACTERES: SIMULACIÓN RTL	70
10.2.6 SIMULACIÓN LÓGICA TEMPORAL A NIVEL DE PUERTAS	71
<b>10.3 RX_RECEIVER</b>	<b>72</b>
10.3.1 ENTIDAD	72
10.3.2 FUNCIONAMIENTO DEL RECEPTOR	73
10.3.3 PUERTOS Y SEÑALES	74
10.3.4 RESULTADOS DE LA SÍNTESIS	76
10.3.5 RECEPCIÓN DE DOS CARACTERES: SIMULACIÓN RTL	76
10.3.6 SIMULACIÓN LÓGICA TEMPORAL A NIVEL DE PUERTAS	77
<b>10.4 TECLADO</b>	<b>77</b>
10.4.1 ENTIDAD	78
10.4.2 FUNCIONAMIENTO DEL TECLADO	78
10.4.3 PUERTOS Y SEÑALES	80
10.4.4 RESULTADOS DE LA SÍNTESIS	82
10.4.5 PULSACIÓN DEL TECLADO: SIMULACIÓN RTL	82
10.4.6 SIMULACIÓN LÓGICA TEMPORAL A NIVEL DE PUERTAS	83
<b>10.5 SPBRGVALUE</b>	<b>84</b>
10.5.1 ENTIDAD	84
10.5.2 FUNCIONAMIENTO DEL BLOQUE SPBRGVALUE	84
10.5.3 PUERTOS Y SEÑALES	85
10.5.4 RESULTADOS DE LA SÍNTESIS	85
 <b>11 PRESUPUESTO</b>	 86
<b>11.1 GASTOS DE HARDWARE</b>	<b>86</b>
<b>11.2 GASTOS DE SOFTWARE</b>	<b>87</b>
<b>11.3 COSTES DE PERSONAL</b>	<b>87</b>
<b>11.4 COSTES TOTALES</b>	<b>88</b>
 <b>13 ENTORNO SOCIOECONÓMICO</b>	 89
<b>14 OBJETIVOS ALCANZADOS</b>	<b>90</b>
<b>15 PROYECTOS FUTUROS</b>	<b>91</b>



# 16 DEMOSTRACIÓN DE FUNCIONAMIENTO DE LA UART

## 92

<b>16.1 DISPOSITIVOS EMPLEADOS</b>	<b>92</b>
16.1.1 DISPOSITIVO CONVERTIDOR USB-UART	92
16.1.2 DISPOSITIVO CONVERTIDOR UART-BLUETOOTH (HC-05)	93
16.1.3 BATERÍA USB	93
<b>16.2 PRUEBA DE FUNCIONAMIENTO</b>	<b>94</b>
16.2.1 CONFIGURACIÓN DEL TERMINAL Y EL DISPOSITIVO UART-BLUETOOTH	95
<b>17 CONCLUSIÓN</b>	<b>98</b>
<b>18 BIBLIOGRAFÍA</b>	<b>100</b>
<b>19 ANEXO A: CÁLCULO DEL BAUD RATE</b>	<b>102</b>
<b>19.1 FORMA DE CALCULAR EL BAUD RATE</b>	<b>102</b>
<b>19.2 EJEMPLO DE CÁLCULO DEL BAUD RATE</b>	<b>103</b>
<b>19.3 CÁLCULO DEL ERROR DEL BAUD RATE</b>	<b>104</b>
<b>19.4 EJEMPLO DE CÁLCULO DEL ERROR DEL BAUD RATE</b>	<b>105</b>
<b>19.5 TABLAS CON LOS CÁLCULOS PARA BAUD RATES ESTÁNDAR</b>	<b>107</b>
19.5.1 BRGH=0 BRG16=0. MODO DE OCHO BITS	107
19.5.2 BRGH=0 BRG16=1. MODO DE 16 BITS	109
19.5.3 BRGH=1 BRG16=0. MODO DE 8 BITS	111
19.5.4 BRGH=1 BRG16=1. MODO DE 16 BITS	113



## ÍNDICE DE FIGURAS

Figura 1: Lectura y escritura en comunicación I2C .....	17
Figura 2: Ejemplo de comunicación SPI .....	19
Figura 3: Transmisión serie síncrona.....	19
Figura 4: Transmisión del byte “sync” para mantener la sincronización, ante la ausencia de datos.....	20
Figura 5: Secuencia de transmisión de un carácter .....	21
Figura 6: UART 8250.....	22
Figura 7: UART 16550.....	23
Figura 8: Transmisión de un carácter a través de la UART .....	26
Figura 9: Transmisión y recepción.....	27
Figura 10: Control de flujo por hardware completo .....	30
Figura 11: Control hardware RTS CTS.....	31
Figura 12: Control de flujo por software .....	32
Figura 13: DB-25 y DE-9.....	33
Figura 14: Conector DB-25 .....	34
Figura 15: Conector DE-9 .....	36
Figura 16: Conexión RS-422 con cuatro líneas .....	37
Figura 17: Ejemplo de comunicación serie mediante el estándar RS-485 .....	39
Figura 18: Placa de desarrollo DE0-Nano .....	42
Figura 19: Dispositivos que incluye la placa de desarrollo.....	43
Figura 20: FPGA .....	44
Figura 21: Diferencia de consumo entre la FPGA Cyclone IV E y el modelo anterior .....	45
Figura 22: USB Blaster y memoria EPCS64.....	45
Figura 23: Puertos de propósito general.....	46
Figura 24: RAM .....	47
Figura 25: Memoria EEPROM .....	47
Figura 26: LEDs .....	48
Figura 27: Pulsadores .....	48
Figura 28: Acelerómetro .....	49
Figura 29: ADC.....	49
Figura 30: Página principal de Quartus II .....	51
Figura 31: Menú principal de ModelSim .....	52
Figura 32: Transmisor asíncrono .....	56
Figura 33: Transmisión de dos caracteres.....	57
Figura 34: Receptor asíncrono .....	58
Figura 35: Recepción de dos caracteres.....	59
Figura 36: Esquema de la UART implementada .....	61
Figura 37: Resultados de la síntesis de la UART .....	62
Figura 38: Bloque correspondiente al generador de baudios.....	63
Figura 39: Resultados de la síntesis del generador de baudios .....	65
Figura 40: Simulación RTL del generador de baudios .....	65

Figura 41: Simulación a nivel de puertas del generador de baudios .....	66
Figura 42: Bloque que contiene las entradas y salidas del transmisor .....	67
Figura 43: Resultados de la síntesis del dispositivo TX_TRANSFER.....	70
Figura 44: Transmisión de dos palabras de nueve bits .....	71
Figura 45: Inicio de la transmisión. ....	71
Figura 46: Simulación con retrasos .....	72
Figura 47: Entidad del receptor.....	72
Figura 48: Resultados de la síntesis para el dispositivo RX_RECEIVER.....	76
Figura 49: Simulación del receptor .....	76
Figura 50: Simulación con retrasos de los componentes.....	77
Figura 51: Teclado matricial .....	77
Figura 52: Bloque correspondiente al teclado .....	78
Figura 53: Esquema de conexión del teclado matricial.....	78
Figura 54: Diagrama de estados del teclado .....	79
Figura 55: Resultados de la síntesis para el TECLADO.....	82
Figura 56: Simulación que muestra la pulsación de una tecla .....	82
Figura 57: Simulación que muestra la señal DECISEGUNDO activarse .....	83
Figura 58: Simulación temporal de la pulsación de una tecla.....	83
Figura 59: Bloque SPBRGVALUE .....	84
Figura 60: Resultados de la síntesis para el dispositivo SPBRGVALUE.....	85
Figura 61: Dispositivo convertidor USB-UART.....	92
Figura 62: Dispositivo UART-bluetooth.....	93
Figura 63: Batería USB.....	93
Figura 64: Bloque del ordenador .....	94
Figura 65: Bloque de la placa de desarrollo .....	95
Figura 66: Configuración del terminal.....	96
Figura 67: Conexión del dispositivo UART-Bluetooth para su configuración .....	97
Figura 68: Entradas y salidas del generador de baudios.....	102
Figura 69: Muestreo de los bits en el medio.....	103



## ÍNDICE DE TABLAS

Tabla 1: Recopilación de pines del conector DB-25 .....	35
Tabla 2: Convenciones de tensión.....	36
Tabla 3: Recopilación de pines del conector DE-9 .....	37
Tabla 4: Comparativa de los estándares de comunicación serie .....	40
Tabla 5: Fórmulas para calcular el valor máximo de los registros SPBRGH y SPBRG .....	55
Tabla 6: Registros que utiliza el transmisor asíncrono.....	58
Tabla 7: Registros que emplea el receptor asíncrono.....	60
Tabla 8: Puertos que forman parte de la entidad del bloque BRG .....	64
Tabla 9: Señales del bloque BRG .....	64
Tabla 10: Puertos que forman parte de la entidad del bloque TX_TRANSFER .....	68
Tabla 11: Señales del bloque TX_TRANSFER .....	69
Tabla 12: Puertos que forman parte de la entidad del bloque RX_RECEIVER .....	74
Tabla 13: Señales del bloque RX_RECEIVER .....	75
Tabla 14: Puertos que forman parte de la entidad del bloque TECLADO .....	80
Tabla 15: Señales del bloque TECLADO.....	81
Tabla 16: Puertos que forman parte de la entidad del bloque SPBRGVALUE.....	85
Tabla 17: Señales del bloque TECLADO.....	85
Tabla 18: Gastos de hardware.....	86
Tabla 19: Gastos de software .....	87
Tabla 20: Costes de personal .....	87
Tabla 21: Costes totales .....	88
Tabla 22: Fórmulas para calcular el baud rate en función de la configuración del dispositivo	103
Tabla 23: Cálculo de "n" para cada baud rate para frecuencias de 50 y 40 MHz .....	107
Tabla 24: Cálculo de "n" para cada baud rate para frecuencias de 20 y 10 MHz .....	107
Tabla 25: Cálculo de "n" para cada baud rate para frecuencias de 8 y 4 MHz .....	108
Tabla 26: Cálculo de "n" para cada baud rate para frecuencias de 2 y 1 MHz .....	108
Tabla 27: Cálculo de "n" para cada baud rate para frecuencias de 50 y 40 MHz .....	109
Tabla 28: Cálculo de "n" para cada baud rate para frecuencias de 20 y 10 MHz .....	109
Tabla 29: Cálculo de "n" para cada baud rate para frecuencias de 8 y 4 MHz .....	110
Tabla 30: Cálculo de "n" para cada baud rate para frecuencias de 2 y 1 MHz .....	110
Tabla 31: Cálculo de "n" para cada baud rate para frecuencias de 50 y 40 MHz .....	111
Tabla 32: Cálculo de "n" para cada baud rate para frecuencias de 20 y 10 MHz .....	111
Tabla 33: Cálculo de "n" para cada baud rate para frecuencias de 8 y 4 MHz .....	112
Tabla 34: Cálculo de "n" para cada baud rate para frecuencias de 2 y 1 MHz .....	112
Tabla 35: Cálculo de "n" para cada baud rate para frecuencias de 50 y 40 MHz .....	113
Tabla 36: Cálculo de "n" para cada baud rate para frecuencias de 20 y 10 MHz .....	113
Tabla 37: Cálculo de "n" para cada baud rate para frecuencias de 8 y 4 MHz .....	114
Tabla 38: Cálculo de "n" para cada baud rate para frecuencias de 2 y 1 MHz .....	114



## 1 Introducción

Este proyecto consiste en la creación de una UART, *Universal Asynchronous Receiver Trasmitter*, en español, Transmisor-Receptor-Asíncrono-Universal; que es un chip de ciertos sistemas digitales, cuyo principal objetivo es convertir los datos recibidos en forma paralela, a forma serie, con el fin de comunicarse con otro sistema externo. A su vez el dispositivo es capaz de recibir datos en formato serie y convertirlos a formato paralelo.

Durante este trabajo, se desarrollarán los diferentes pasos que se deben seguir para su creación. Estos pasos consisten en el estudio de la UART de un microcontrolador determinado, en este caso el PIC18F2525 de Microchip; y después el diseño de nuestra UART en lenguaje VDHL.

Además de esto, una vez implementada la UART sobre una FPGA, se comprobará su funcionamiento y se hará una demostración de ello. Esta demostración consistirá en mantener una comunicación serial entre una FPGA y el terminal de un ordenador.

Por último, durante este proyecto, se hará un presupuesto donde se exponga cuánto le costaría a un grupo de personas desarrollar esta UART, teniendo en cuenta los gastos de personal, de software y de hardware.



## 2 Objetivos propuestos en el proyecto

Durante la realización del proyecto se plantean una serie de objetivos que desarrollaremos a continuación. Estos objetivos no solo engloban una serie de metas a cumplir sino que también se incluye una serie de conocimientos a adquirir para poder cumplir los objetivos del proyecto.

El principal objetivo del proyecto es el diseño e implementación de una UART en el lenguaje de descripción de hardware VHDL. Para alcanzar este objetivo será necesario el cumplimiento de una serie de requisitos entre los que destacan los siguientes:

- Realizar un estudio previo de los componentes que conforman una UART y comprender su funcionamiento para conseguir elaborar una implementación funcional.
- Realizar un estudio de dispositivos UART que han sido relevantes en el pasado para tener un conocimiento más certero del funcionamiento del dispositivo.
- Ampliar los conocimientos en VHDL para conseguir implementar una UART lo más eficiente posible.
- Realizar un estudio de las partes de una FPGA y más concretamente de la placa de desarrollo utilizada para implementar el proyecto. En este caso emplearemos la placa DE0-Nano de Terasic, la cual incorpora la FPGA Cyclone IV de Altera.
- Aprender a utilizar el software necesario para implementar el diseño en la FPGA. En este caso emplearemos la herramienta Quartus II Web Edition suministrada por Altera.
- Emplear Quartus II para implementar el diseño de la UART en VHDL y corregir errores de código, tanto sintácticos como funcionales.
- Durante la realización del proyecto incorporar mejoras significativas en el desarrollo de la UART. Partiendo de una UART con funcionalidad básica incorporar noveno bit, diferentes *baud rates*, etc.
- Realizar una serie de simulaciones mediante la herramienta Modelsim-Altera para comprobar el funcionamiento de los diferentes componentes de la UART. También será necesaria la realización de una simulación de todo el circuito en general. Se obtendrá una primera simulación funcional y después se realizarán simulaciones post-layout, que tienen en cuenta los retrasos de los componentes internos de la FPGA.
- Después de las simulación probar el diseño en la placa y comprobar su correcta funcionalidad con un ejemplo sencillo



Después de implementar la UART en la FPGA será necesaria la creación de un circuito de test que demuestre el correcto intercambio de información mediante la UART.

- Implementar un caso práctico que demuestre el funcionamiento de la UART comunicando la FPGA con otro dispositivo. En concreto emplearemos un dispositivo USB- UART para comunicar la FPGA con el terminal de un ordenador. Además obtendremos un interfaz de comunicación inalámbrica añadiendo dos dispositivos *UART-bluetooth*.
- Emplearemos un teclado matricial para enviar caracteres al terminal. Por tanto será necesario implementar en VHDL un diseño que permita el correcto funcionamiento de este teclado.

### 3 ¿En qué consiste la comunicación serie?

La comunicación serie consiste en el envío de bits de información de manera secuencial a través de una única línea. Este tipo de comunicación difiere completamente de la comunicación en paralelo que consiste en enviar simultáneamente un conjunto de datos a través de varias líneas. De esta manera se conseguiría una velocidad de transmisión mayor, ya que se enviaría un grupo de bits a la vez, en vez de uno.

Pero si la velocidad de transferencia de datos en paralelo es mucho más rápida, ¿por qué se utiliza la transmisión de datos serie? Algunas respuestas se dan a continuación: Para realizar la comunicación de datos en paralelo se requiere gran cantidad de hilos conductores, pues debe ser establecido un hilo para cada bit de datos, además de las señales de control. Esto encarece notablemente la comunicación en función de la distancia. La comunicación serie requiere 2, 3 o 4 hilos.

- Una entrada salida/serie puede ser transmitida a través de pares de cobre, cable coaxial, fibra óptica, vía radio o vía satélite, lo que proporciona comunicación con equipos remotos (redes locales) o muy remotos (Internet a través de las redes telefónicas y de datos).
- La comunicación paralela no posee el alto grado de estandarización que ha alcanzado la comunicación serie, lo que permite la intercomunicación entre equipos, por ejemplo mediante RS232, USB o *FireWire*.

Existen una gran cantidad de protocolos serie en la actualidad. Todos ello se puede clasificar en función del tipo de sincronía que presentan y la dirección de transmisión de los bits.

La forma en la que se indica cuándo empieza un carácter y cuándo llega cada bit, es una forma de clasificación de los tipos de comunicación serie. Bajo este tipo de diferenciación identificamos dos tipos de comunicación serie. La comunicación síncrona y la comunicación asíncrona.

También hay que tener en cuenta el sentido de transmisión de esos bits. En este caso encontramos tres tipos de comunicación. La comunicación *simplex*, *semi-duplex* y *full-duplex*.

#### 3.1 Clasificación por tipo de sincronía

- Comunicación síncrona: En este tipo de comunicación existe un maestro que marca el ritmo de la comunicación. Este maestro es el encargado de transmitir la señal de reloj al dispositivo esclavo. De esta manera se envía un bit con cada ciclo de reloj y ambos dispositivos conocen el momento exacto de transmisión.
- Comunicación asíncrona: en este caso no hay un intercambio de reloj entre ninguno de los dispositivos. Esto no significa que no haya sincronización entre los dispositivos.



Para que exista la comunicación ambos dispositivos deben compartir la misma velocidad de transmisión de los datos. Para ello se emplea una misma tasa de bits por segundo (*baud rate*).

### 3.2 Clasificación por el sentido de la comunicación

- *Simplex*: en este caso la comunicación solo se puede establecer en un solo sentido a través de la misma línea.
- *Semi-duplex*: en la comunicación *half-duplex* o *semi-duplex* la comunicación se puede establecer en una misma línea y ambos sentidos pero no simultáneamente. Por tanto se alterna la transmisión de datos entre los dispositivos que interviene en la comunicación.
- *Full-dúplex*: en esta ocasión se permite la comunicación en ambas direcciones y simultáneamente. Por tanto tiene que haber al menos dos líneas de transmisión de datos. Una para transmitir y otra para recibir.

## 4 Tipos de comunicación serie

Antes de comenzar con la descripción detallada de la UART es importante hacer una breve introducción de los protocolos de comunicación serie más extendidos. Esto es bastante relevante ya que es útil a la hora de identificar las ventajas y los inconvenientes del sistema de transmisión de datos que vamos a implementar.

Entre todos los sistemas de comunicación serie existentes cabe destacar los siguientes protocolos:

### 4.1 I2C *Inter-Integrated Circuit (Inter-Circuitos Integrados)*

I2C es un bus serie diseñado por Philips en el año 1992 aunque su versión más popular apareció en el año 2000. Su velocidad estándar es de 100 Kbit/s aunque permite velocidades de 3.4Mbits/s. Es un bus muy empleado para la comunicación de microcontroladores y periféricos en sistemas integrados, por tanto, muy empleado en la industria.

Presenta dos líneas para transmitir la información: una para transmitir el reloj (SCL) y otra para la trasmisión de datos (SDA). Los dispositivos conectados al bus pueden ser maestros o esclavos y pueden intercambiarse el rol de maestro entre los distintos dispositivos que lo permitan (multimáestro). Dado que solo tiene una línea de transmisión de datos la comunicación no puede ser *full-duplex*, pero si *semi-duplex*.

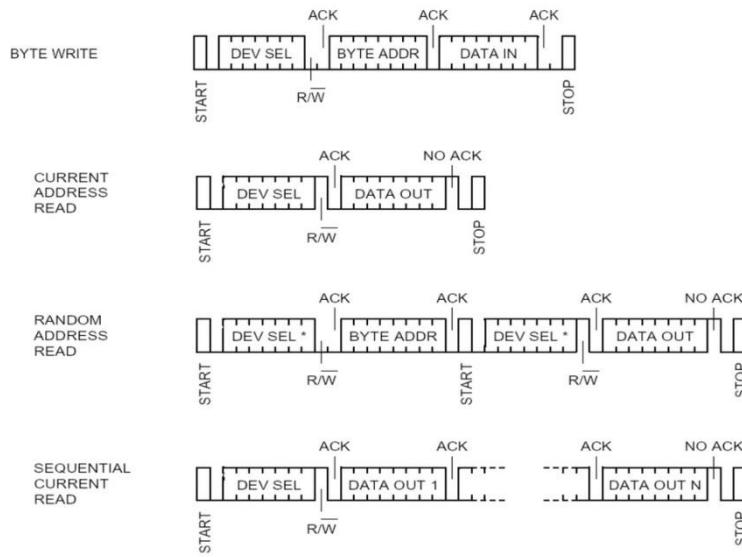


Figura 1: Lectura y escritura en comunicación I2C

Fuente: <https://www.maximintegrated.com/en/app-notes/index.mvp/id/4649>



El maestro comienza la comunicación con una condición de start, seguida de un byte compuesto por la dirección del dispositivo esclavo y un bit de que determina una operación de lectura os escritura. El esclavo correspondiente responde enviando un bit de ACK. A continuación el maestro transmite un byte que determina el registro del esclavo que se va a leer o escribir. Tras un nuevo bit ACK se comienza con la transmisión de datos en paquetes de ocho bits seguidos de un bit ACK. Cuando la transmisión finaliza el maestro envía una condición de STOP.

El estado de la línea de reloj es controlado por el maestro, aunque el esclavo puede forzarlo a nivel bajo por motivos de baja velocidad o para detener la transferencia.

## 4.2 SPI (Serial Peripheral Interface)

El bus SPI es un estándar de comunicación empleado principalmente para el intercambio de información entre circuitos integrados de sistemas electrónicos. Es un estándar válido para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de datos en serie controlados por un reloj.

Nos encontramos ante un protocolo síncrono. La trasmisión de datos se consigue mediante cuatro líneas:

- SCLK : A través de esta línea el maestro manda la señal de reloj a los dispositivos esclavos.
- MOSI (*Master Output Slave Input*): Es una línea de intercambio de datos. Concretamente la salida del maestro y la entrada del esclavo.
- MISO (*Master Input Slave Output*): Es otra línea de transmisión de datos. A diferencia de la línea MOSI, en este caso nos encontramos ante la salida del esclavo y la entrada del maestro.
- SS (chip select): mediante esta línea el maestro activa el esclavo con el que se intercambia la información.

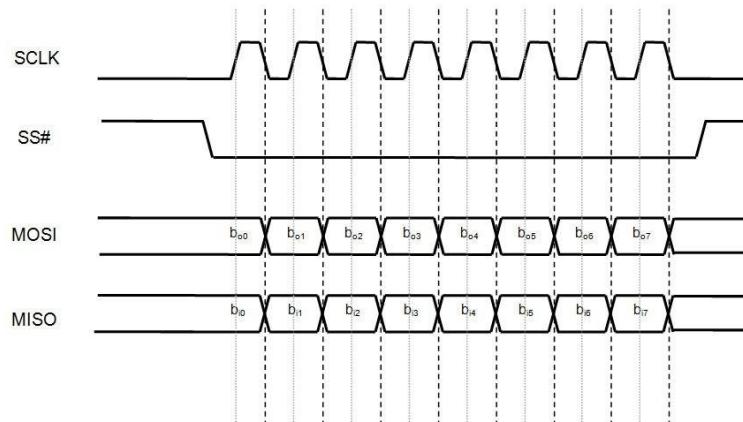


Figura 2: Ejemplo de comunicación SPI

Fuente: <http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/>

La cadena de bits se transmite con los pulsos del reloj, de modo que con cada pulso el master envía un bit. Para que se inicie la transmisión el master pone a nivel bajo la línea SS del esclavo. De esta forma el dispositivo esclavo se activa y recibe el primer bit con el siguiente pulso. Los pulsos del reloj pueden estar configurados de tal forma que la escritura en MOSI se haga con el flanco de subida y se haga la lectura con el flanco de bajada. También se puede emplear otras polaridades.

Este bus permite una comunicación full dúplex, mayor velocidad que el I2C y no está limitado a la transferencia de bloques de ocho bits. Además su implementación en hardware es relativamente fácil y emplea una única línea específica para cada esclavo (SS).

Por otro lado utiliza más líneas que la comunicación por I2C y solo funciona en distancias cortas a diferencia de protocolos como el RS-232, RS-485 y otros buses industriales.

### 4.3 Comunicación serie síncrona (USART)

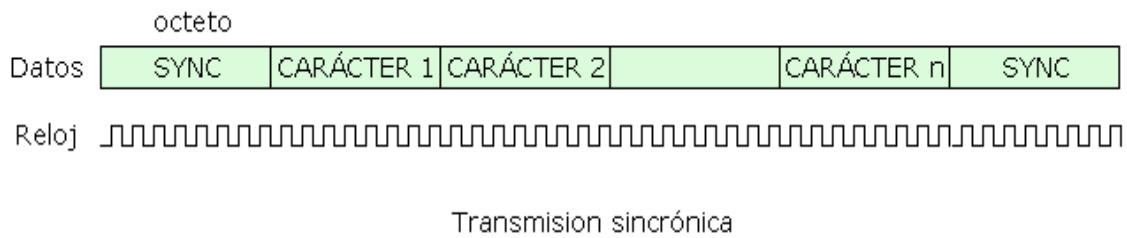


Figura 3: Transmisión serie síncrona

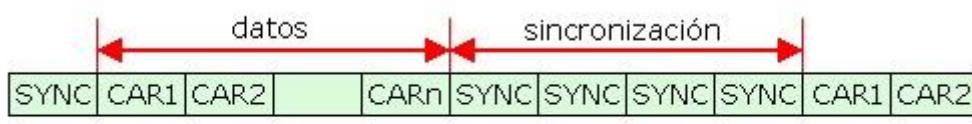
Fuente: <http://perso.wanadoo.es/pictob/comserie.htm>

La comunicación serie síncrona es más eficiente que cualquier tipo de comunicación asíncrona en términos de velocidad de transmisión.

Cuando se transmite información de manera síncrona lo primero que se envía es un byte de sincronización (sync). Este byte de sincronización realiza la misma función que el bit de inicio de la transmisión serie asíncrona, indicando al receptor que va a recibir una serie de datos. Este byte también determina con qué frecuencia será muestreada la señal en el receptor y por tanto, permite sincronizar los relojes del transmisor y el receptor. Por regla general la mayoría de los dispositivos ejecutan una resincronización para evadir posibles desviaciones del reloj. Esta sincronización se produce cada uno o dos segundos, introduciendo el byte “sync” dentro periódicamente entre los datos enviados.

El byte de sincronización debe diferenciarse de los datos del usuario para posibilitar que el receptor lo identifique. En código ASCII se suele emplear el byte 10010110.

Existen casos en los que se definen dos bytes distintos de sincronización. Esto sería necesario si por cualquier error el carácter “sync” se desvirtuara y el receptor no lo detectase. El segundo byte de sincronización permitiría la inicialización del receptor.



Inserción automática de caracteres de sincronismo

Figura 4: Transmisión del byte “sync” para mantener la sincronización, ante la ausencia de datos

Fuente: <http://perso.wanadoo.es/pictob/comserie.htm>

Cabe destacar que cuando se realiza una transmisión de manera asíncrona es necesario mantener el sincronismo entre transmisor y receptor cuando no se envían caracteres. Para ello se envía de manera continua caracteres de sincronismo para que no se pierda la comunicación.

#### 4.4 Comunicación serie asíncrona (USART)

La principal característica de la comunicación serie asíncrona consiste en que no se intercambia la señal de reloj entre emisor y receptor. Dado que no se comparte el reloj, la comunicación se puede iniciar en cualquier momento. Esto conlleva que cada dispositivo opera con su propio reloj y por tanto se debe acordar una velocidad de transmisión de datos o *baud rate*.

En este caso la comunicación se compone de dos líneas de datos:

TD: es la línea de transmisión de datos

RD: es la línea de recepción de datos

Estas son las líneas principales pero se pueden incorporar un número mayor de líneas para realizar control de flujo por hardware o por software

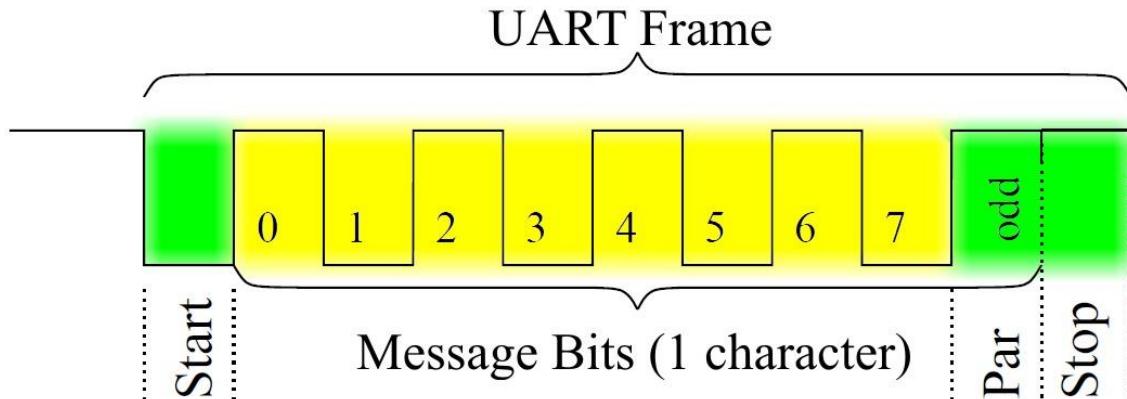


Figura 5: Secuencia de transmisión de un carácter

Fuente: <https://netwerkt.wordpress.com/2011/01/02/an-alternative-uart-that-does-not-need-special-quartz-frequencies/>

Cada carácter transmitido se compone de los siguientes bits:

- Bit de START: es el bit encargado de marcar el inicio de la comunicación. Este bit presenta valor lógico 0.
- Bits de datos: estos bits componen la información que interesa transmitir. Generalmente suelen ser 7 u ocho datos, aunque hay otras configuraciones.
- Bit de paridad: este bit es útil para detectar errores en la transmisión de la información. Se puede emplear tanto paridad par como impar. Si al sumar los bits en el receptor no se obtiene la paridad elegida, los datos recibidos no se corresponden con el dato transmitido.
- Bit de parada: este bit determina la finalización de la transmisión de un carácter. Este bit tiene valor lógico 1.

## 5 Dispositivos UART con mayor relevancia

Existen una serie de circuitos integrados que implementan una comunicación serie asíncrona. Entre estos dispositivos destacan los nombrados a continuación:

### 5.1 UART 8250

En la siguiente imagen podemos ver una UART 8250:



Figura 6: UART 8250

Fuente: [https://ca.wikipedia.org/wiki/UART\\_8250](https://ca.wikipedia.org/wiki/UART_8250)

El UART 8250 (Wikipedia, UART 8250, 2014) es un circuito integrado diseñado para establecer una interfaz de comunicación serie. Este dispositivo fue fabricado por National Semiconductor. Se incorporaba con frecuencia en los PCs y en periféricos relacionados, como pueden ser impresoras y módems. Este chip incorpora, entre otros componentes, un generador de baudios. Esto le permite operar empleando tanto *bit rates* comunes como de propósito especial, pudiéndose utilizar diferentes frecuencias de reloj para un mismo *baud rate*.

El nombre del chip incorpora letras como sufijos, determinando así la correspondiente revisión del dispositivo. De hecho el dispositivo 8250 original fue rápidamente precedido por las versiones 8250A y 8250B corrigiendo algunos errores del chip. Concretamente el chip 8250 podía repetir la transmisión de un carácter si la línea *Clear To Send* (CTS) fuera activada asíncronamente durante el primer intento de transmisión.

Debido a la amplia incorporación al mercado del chip, otros fabricantes empezaron a fabricar chips compatibles. Western Digital implementó el chip WD8250 bajo varios nombres como "*Async Communications Interface Adapter*" y "*Async Communications Element*". Estos chip fueron sustituidos por el dispositivo 16450 y 16450A, que mejoraron la comunicación serie permitiendo velocidades mayores.

## 5.2 UART 16550

En la siguiente imagen podemos apreciar el esquema de una UART 16550:

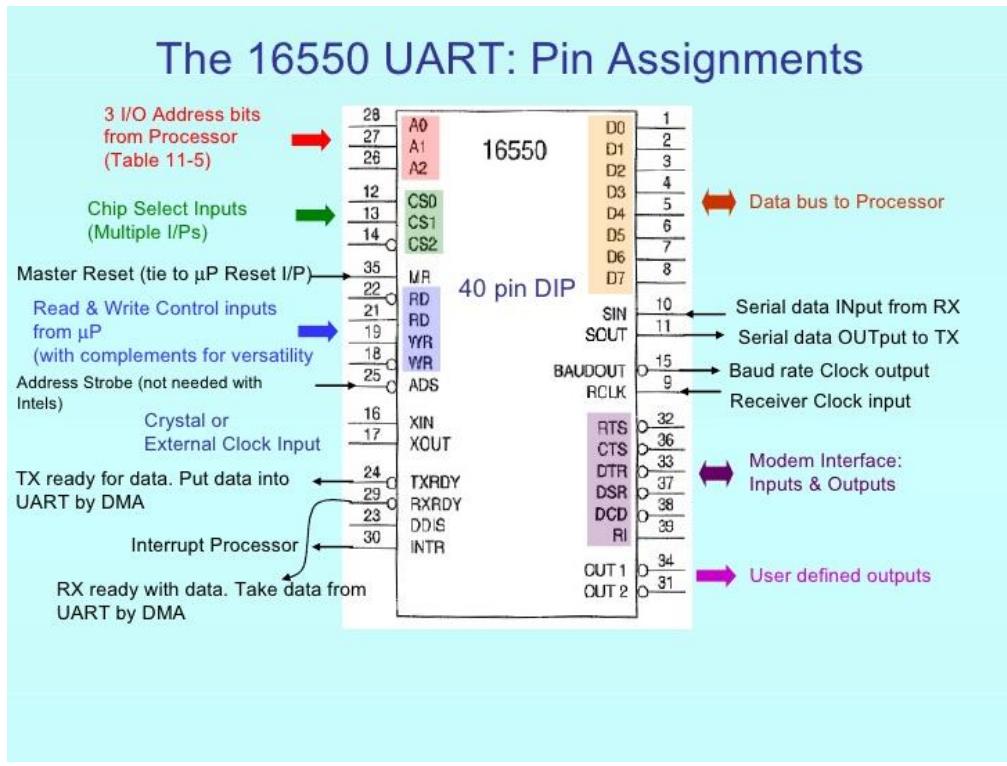


Figura 7: UART 16550

Fuente: [http://www.slideshare.net/gurls\\_on\\_mars/uart-16550](http://www.slideshare.net/gurls_on_mars/uart-16550)

El dispositivo UART 16550 (Wikipedia, 16550 UART, 2015) es un circuito integrado diseñado para implementar una interfaz de comunicación serie. Se empleó de manera frecuente para implementar el puerto serie de PCs IBM donde se conecta a una interfaz RS232 para módems, ratones, impresoras y periféricos similares.

Este componente fue fabricado originalmente por National Semiconductor. De forma similar otros fabricantes diseñaron numerosos circuitos integrados con distintos niveles de compatibilidad con respecto al original.

El reemplazo de los dispositivos UART 8250 por el circuito 16550, fue una actualización habitual entre los propietarios de una PC IBM, XT y ordenadores compatibles, cuando los módems de alta velocidad estuvieron disponibles. A velocidades superiores a 9600 baudios se descubrió que el puerto serie no era capaz de manejar un flujo continuo de datos sin perder caracteres. El intercambio del dispositivo 8250 por el dispositivo 16550 y la incorporación de sistemas de software para controlar la memoria FIFO (*First Input First Output*) que este incorpora, supuso una mejora de la estabilidad y la fiabilidad en conexiones de alta velocidad.

Uno de los inconvenientes de los modelos anteriores, como el 8250 y el 16450, fue que se producía una interrupción por cada byte recibido. Por tanto con el incremento de la velocidad



de transmisión las interrupciones eran cada vez más frecuentes. Por tanto existía un riesgo evidente de que un byte recibido fuera sobrescrito debido a un retraso en las interrupciones.

Para superar este defecto, la UART 16550 incorpora una memoria FIFO con una interrupción programable disparada en el primer, el cuarto, el octavo o el catorceavo bit recibido.

El dispositivo UART 16550 original presentaba un error que impedía el funcionamiento de esta memoria FIFO. Posteriormente National Semiconductor lanzó la actualización 16550A que corregía este inconveniente.



## **6 UART (Universal Asynchronous Receiver-Transmitter ). Comunicación serie asíncrona.**

En este capítulo se estudia el protocolo de comunicación serie utilizado por la UART. En primer lugar se explicará que es la UART, cuál es su forma de transmitir datos y los diferentes dispositivos que la componen. Para finalizar se explicará en qué consiste el control de flujo y se definirán los estándares que avalan este tipo de comunicación serie.

### **6.1 UART. Definición**

Una UART (transmisor/receptor asíncrono universal) es un dispositivo hardware capaz de traducir datos con formato paralelo a formato serie. Los dispositivos UART son empleados comúnmente en conjunción con estándares de comunicación como EIA, RS-232, RS-422 o RS-485. Se denominan universales ya que el formato y la velocidad de transmisión de los datos es configurable. Los niveles eléctricos de corriente y tensión son controlados por un driver externo a la UART.

Frecuentemente, la UART se presenta como un circuito integrado individual, empleado para implementar una interfaz de comunicación serie en un ordenador o en un dispositivo periférico. Actualmente es un circuito ampliamente incluido en microcontroladores.

### **6.2 Transmitir y recibir datos en formato serie**

La UART es capaz de transmitir bytes de datos enviando bits individuales de forma secuencial. El dispositivo receptor se encarga de reensamblar estos bits en un byte completo. Cada UART posee un registro de desplazamiento, necesario para la conversión de datos de paralelo a serie. Evidentemente la transmisión de datos a través de una sola línea es menos costosa que una transmisión en paralelo con múltiples líneas.

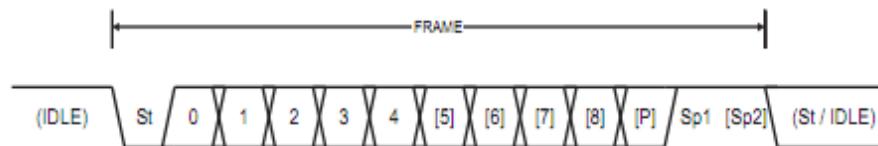
Normalmente la UART no genera o recibe las señales externas empleadas entre distintos dispositivos. Se emplean dispositivos externos para convertir los niveles lógicos de la UART a niveles diversos de tensión y corriente. Estándares como el RS-232, RS422 y RS-485 presentan niveles distintos de tensión para los mismos estados lógicos.

La comunicación puede ser simplex (comunicación en una sola dirección), full-dúplex (dos dispositivos transmiten y reciben al mismo tiempo) y half-duplex (los dispositivos se turnan para transmitir y recibir).

Dado que en la UART la comunicación se realiza mediante la transmisión de caracteres, se explica, en el siguiente apartado, este proceso.

### 6.2.1 Transmisión de un carácter

Siempre que no se esté produciendo una transmisión de datos la línea de salida estará a nivel alto. Esto se debe a una herencia por parte del telégrafo que se mantenía con tensión para mostrar que la línea y el transmisor no estaban dañados. El envío de un carácter se completa con la secuencia mostrada en la siguiente figura:



- St** Start bit, always low.
- (n)** Data bits (0 to 8).
- P** Parity bit. Can be odd or even.
- Sp** Stop bit, always high.
- IDLE** No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.

Figura 8: Transmisión de un carácter a través de la UART

Fuente: <http://electronics.stackexchange.com/questions/24551/seeking-mcu-with-9-data-bit-uart>

- Cada transmisión se inicia con un bit de START. En este caso la línea tendrá un nivel lógico de cero.
- A continuación se enviará la información requerida bit a bit. Frecuentemente se puede programar el número de bits que queremos transmitir. Se suele poder elegir en un rango de 5 a 9 bits dependiendo de la UART empleada. A pesar de esto lo más habitual es la transmisión de 8 bits de datos.
- Opcionalmente se puede transmitir un bit de paridad siempre y cuando no se haya elegido la transmisión de 9 bits de datos.
- Para finalizar la transmisión se envía un bit de STOP. A diferencia del bit de START nos encontramos con un nivel lógico de uno.

Generalmente el primer bit transmitido se corresponde con el bit menos significativo. La condición de START señala al receptor que la transmisión de un carácter se ha iniciado. Si se emplea bit de paridad este se transmitirá al final de los bits de datos. Los siguientes bits (uno o dos) siempre presentan un estado lógico alto y se corresponden con los bits de STOP. Estos indican al dispositivo receptor que la transmisión del carácter ha finalizado. Dado que siempre hay, como mínimo, un bit de START con nivel bajo y un bit de STOP con nivel bajo, nos garantizamos al menos dos cambios de la señal entre cada carácter. De esta manera se asegura la correcta comunicación entre dispositivos.

### 6.3 Bloques generales que componen la UART

La UART se compone principalmente de dos bloques. Un transmisor y un receptor. Dependiendo de la UART podemos encontrar distintos bloques que la componen pero estos son propios de cualquier UART.

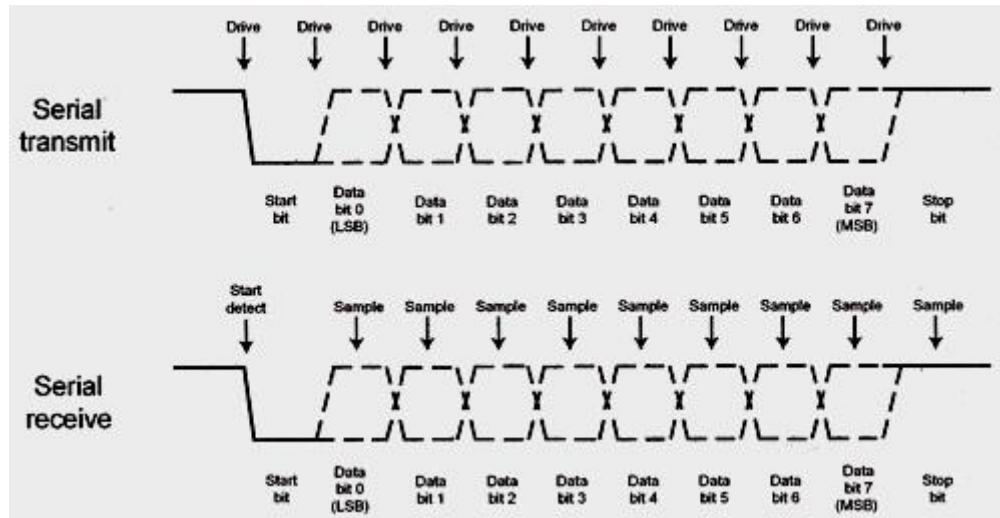


Figura 9: Transmisión y recepción

Fuente: <http://www.embedded.com/design/other/4025995/Implementing-your-MCU-based-system-s-serial-UART-in-software>

En la figura anterior podemos apreciar un carácter enviado por el transmisor y otro que detecta el receptor. Estos dos bloques se describen en los siguientes apartados.

#### 6.3.1 Receptor

Todas las operaciones realizadas por el hardware de la UART son comandadas por un reloj cuya frecuencia es múltiplo de la velocidad de transmisión de los bits, típicamente 8 veces el *bit rate* aunque se permiten otras velocidades. El receptor comprueba el estado de la línea de entrada en cada pulso de reloj, esperando el comienzo marcado por el bit de START. Una vez finalizado el bit de START el estado de la línea es muestreado y el resultado se introduce en un registro con desplazamiento. Una vez que se han transmitido todos los bits de datos, el contenido del registro de desplazamiento está en disposición del sistema. La UART tiene la opción de activar un *flag* indicando que hay un nuevo dato disponible. Además, algunas UARTs implementadas en sistemas embebidos pueden generar una interrupción en el procesador que permite la transferencia de los datos recibidos.

La comunicación UART no tiene ninguna forma de transmitir una cierta temporización más allá de las líneas de transmisión de datos. Frecuentemente, la UART resincroniza su reloj interno en cada cambio de byte. Dado que se sincroniza de esta manera, el receptor confía en que la



transmisión de datos sea al *bit rate* previamente acordado. Por tanto la UART se sincroniza en el flanco de bajada del bit de START y después lee en el centro de cada bit de datos que se espera recibir. Este sistema funciona siempre que la velocidad de los bits transmitidos sea lo suficientemente precisa para que el bit de STOP se muestre en el centro con una exactitud aceptable.

Está bastante estandarizado que la UART almacene un carácter mientras recibe el siguiente. Este “doble buffer” permite a un receptor tener todo el tiempo que se emplea para transmitir un carácter, para poder leer otro carácter ya recibido. Muchas UARTs presentan una pequeña memoria FIFO entre el receptor y la interfaz del dispositivo que utiliza los datos. Esto le otorga más tiempo al equipo que incorpora la UART para generar una interrupción y así asegurar que no se pierda ningún dato recibido.

### 6.3.2 Transmisor

La forma de operar para la transmisión de datos es más simple que la recepción. Tan pronto como se escribe un carácter en el registro con desplazamiento, se genera un bit de START. A continuación el registro con desplazamiento va transmitiendo los bits a la línea de salida en concordancia con el *bit rate* prestablecido. Para concluir se transmite el bit de paridad de forma opcional y el bit de STOP. Como la transmisión de un único carácter puede durar un tiempo considerable en comparación con la velocidad de una CPU, la UART mantendrá un *flag* activo. Este *flag* indica que la UART está ocupada y que por tanto, el sistema no debe depositar un nuevo bit en el transmisor hasta que la transmisión anterior se complete.

## 6.4 Componentes básicos de la UART

Una vez conocido el funcionamiento básico de la UART, estamos en disposición de definir los componentes básicos que componen tanto el receptor como el transmisor.

Dentro del transmisor y el receptor podemos encontrar componentes más concretos:

- Un generador de reloj (Baud Rate Generator): normalmente genera un reloj cuya velocidad es múltiplo de la velocidad de transmisión (*bit rate*). De esta manera se puede muestrear en el medio de cada bit transmitido.
- Dos registros con desplazamiento, uno para el receptor y otro para el transmisor. No se puede emplear el mismo para las dos funciones ya que en una comunicación full dúplex se necesitan los dos dispositivos.
- Sistema de control para el transmisor y el receptor.
- Lógica para el control de las lecturas y escrituras. Interrupciones, flags, etc.



- Buffers para el transmisor y el receptor. En este caso hablamos de dispositivos opcionales, pero muy útiles. Otorgan tiempo de reacción al equipo que incorpora la UART, dándoles más margen de tiempo para leer y escribir datos.
- Buses paralelos para permitir escribir datos en la UART y permitir leer los datos que esta proporciona.
- Memoria FIFO (First Input, First Output).

## 6.5 Control de flujo

Dentro del control de flujo podemos encontrar gran cantidad de alternativas en función del diseñador.

Existen diferentes tipos de control de flujo:

- Hardware completo
- Control hardware únicamente con RTS y CTS
- Software (Xon- Xoff)
- Sin control de flujo

Tanto para el control de flujo por hardware como para el software, la conexión mínima necesaria para la comunicación de dos dispositivos es la siguiente:

- La línea TxD del transmisor con la línea RxD del receptor.
- La línea RxD del receptor con la línea TxD del receptor.
- Las líneas de tierra de ambos dispositivos deben estar conectadas.

En los siguientes apartados se explicarán los distintos tipos de control de flujo.

### 6.5.1 Control de flujo por hardware completo

En este tipo de control de flujo se emplean los puertos RTS/CTS y DTR/DSR. Estos puertos trabajan de forma conjunta siendo un par de entrada y otro de salida. El primer para de líneas es RTS (*Request to Send*) y CTS (*Clear to Send*). Cuando el receptor de uno de los dispositivos está listo para recibir datos, cambia la línea RTS ha estado alto. Este valor será leído por el transmisor del otro dispositivo en el puerto CTS, indicándole que puede enviar datos. El

siguiente par de líneas es DTR (*Data Terminal Ready*) y DSR (*Data Set Ready*). Estas líneas se utilizan principalmente para comunicación por modem, permiten al puerto serie y modem indicarse mutuamente su estado. Por ejemplo, cuando un modem se encuentra preparado para que un PC le envíe datos, cambia la línea DTR ha estado alto indicando que se ha realizado una conexión por vía telefónica. Este valor se lee en el puerto DSR del ordenador, que empieza a enviar datos. Como regla general, las líneas DTR/DSR se utilizan para indicar que el sistema está listo para la comunicación, mientras que las líneas RTS/CTS se utilizan para paquetes individuales de datos.

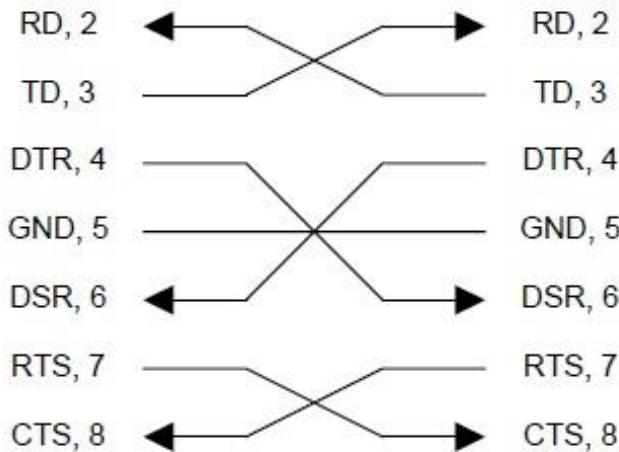


Figura 10: Control de flujo por hardware completo

Fuente: <http://www.monografias.com/trabajos104/puerto-serie-del-pc/puerto-serie-del-pc.shtml>

En la anterior fotografía podemos ver como conectar los puertos de dos UART para mantener un control de flujo por hardware completo.

### 6.5.2 Control hardware únicamente con RTS y CTS

Esta forma de control de flujo es similar al control de flujo completo, salvo que en este caso no se emplean las señales DTR Y DSR. Por tanto el control de Flujo queda relegado a la utilización de las señales RTS y CTS. El esquema de conexiones se muestra en la figura 11.

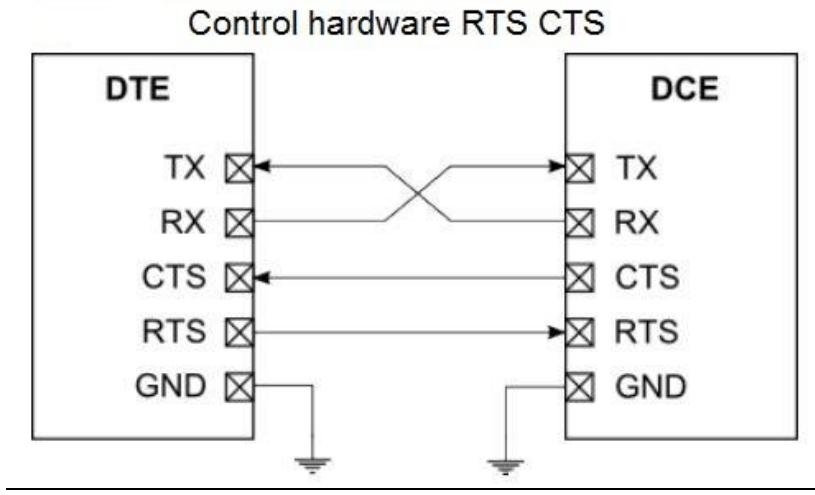


Figura 11: Control hardware RTS CTS

Fuente: <http://www.avrfreaks.net/forum/uart-flow-control-between-dte-dce>

### 6.5.3 Control de flujo por software (Xon- Xoff)

El control de flujo por software consiste en la utilización de los caracteres Xon y Xoff para la detención y reanudación de envío de bytes. Este tipo de control es ampliamente utilizado en informática (ordenadores, impresoras, etc.)

En las tablas del código ASCII el carácter Xon es el numero decimal 17, mientras que el Xoff es el número 19. Con frecuencia al carácter Xon se le conoce con el nombre de DC1 y el carácter Xoff se le conoce con el nombre de DC3.

Es una buena forma de controlar el flujo de datos si transmitimos caracteres alfanuméricos, dado que estos caracteres no coinciden con los caracteres de control en el código ASCII. En cambio tiene sus inconvenientes para datos binarios, dado que alguno de los bytes puede coincidir con los caracteres de control.

El funcionamiento se ve reflejado en la figura 12.

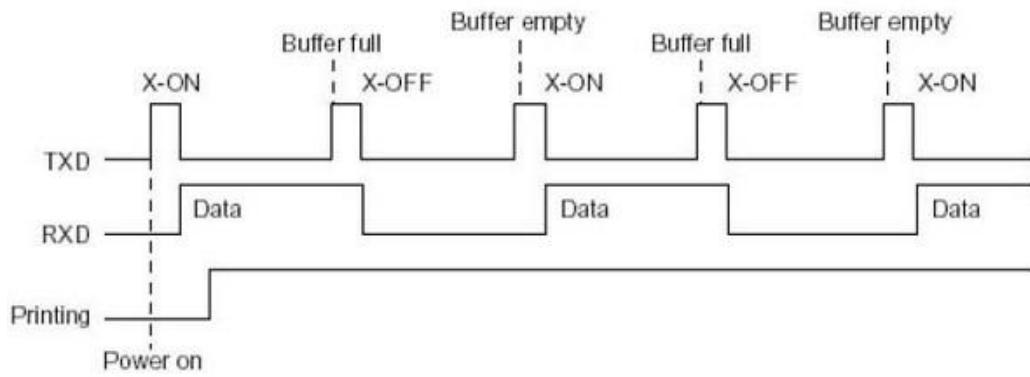


Figura 12: Control de flujo por software

Fuente: <http://www.starasia.com/InterfaceFAQ.asp>

Cuando el buffer del dispositivo receptor está próximo a llenarse, el propio dispositivo envía el carácter Xoff. El dispositivo emisor al recibir el mensaje detiene la transmisión de datos. Hay que tener en cuenta que desde el envío del carácter Xoff, hasta la detención del dispositivo de emisión, aun puede llegar algún dato. Por tanto no se debe esperar a que el buffer este completamente lleno para mandar el carácter de parada.

Para que el flujo de datos se reanude, el emisor debe recibirle carácter Xon. Este carácter lo envía el dispositivo receptor cuando existe espacio suficiente en su buffer.

Una de las principales ventajas de esta forma de controlar el flujo de datos es que no se necesitan líneas adicionales. Los caracteres de control son enviados por las líneas de transmisión de datos propias de la UART (RxT y TxT)

## 6.6 Estándares de comunicación serie

Hasta ahora solo hemos comentado el dispositivo UART desde un punto de vista digital, pero También es interesante tener un cierto conocimiento de las señales eléctricas que engloban ciertos estándares de comunicación serie.

Entre estos estándares destacan el RS-232, el RS-422 y el RS-485.

### 6.6.1 RS-232

También es conocido como EIA/TIA RS-232C. Este estándar describe una norma para el intercambio de datos binarios entre un DTE (*Data Terminal Equipment*, “Equipo Terminal de datos”) y un DCE (*Data Communication Equipment*, “Equipo de Comunicación de Datos”).

Existen casos en los que se requiere la conexión de dos DTE. Esto sucede cuando interesa conectar dos computadores.

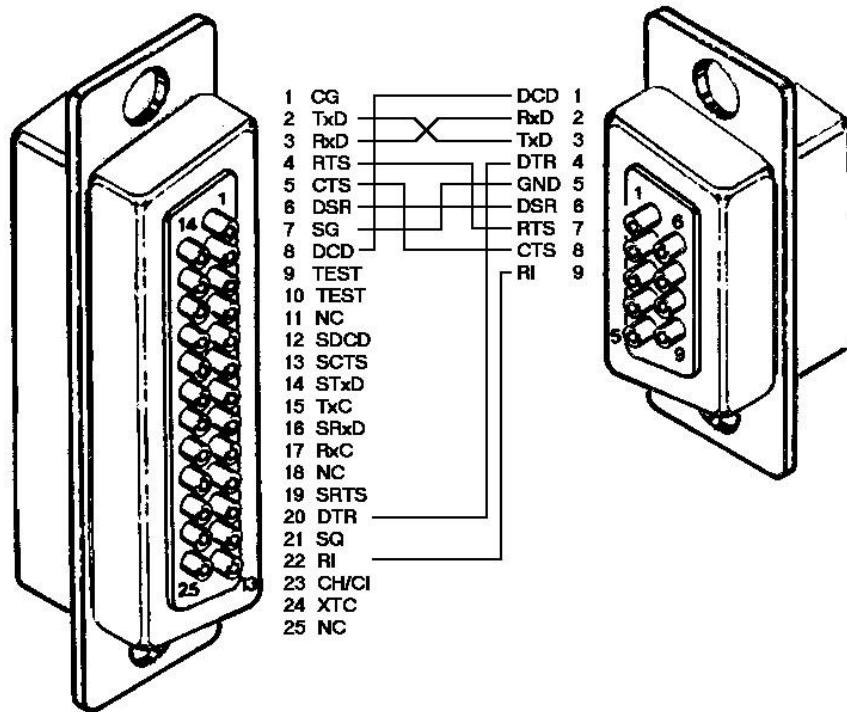


Figura 13: DB-25 y DE-9

Fuente: <http://laurent.granjon.free.fr/connectique/rs232.htm>

El estándar RS-232 consiste en un conector del tipo DB-25 (25 pines), aunque posteriormente se creó un conector de 9 pines (DE-9) bastante más barato y más extendido para ciertos periféricos como puede ser un ratón de ordenador. La correspondencia de puertos entre los dos conectores se puede apreciar en la figura 13.

La interfaz RS-232 está diseñada para transmitir datos en distancias cortas, de hasta 15 metros y con velocidades de comunicación bajas, no mayores a 20 kbps. Aun así la norma también se emplea con velocidades superiores y resultados aceptables. La interfaz trabaja con comunicación síncrona o asíncrona y con tipos de línea *simplex*, *semi-duplex* o *full-duplex*.

Las líneas de *handshaking* de la norma RS-232 se emplean para resolver problemas tales como el direccionamiento de los datos en un instante determinado. Si un dispositivo de los que están conectados a una interfaz RS-232 procesa los datos a una velocidad menor deben activarse las señales de *handshaking* relacionadas con el control de flujo. De esta forma al dispositivo más lento le dará tiempo a procesar la información. Estas líneas son la RTS y la CTS. En su diseño el estándar no concebía que estas líneas se empleasen para este uso, pero dada su gran utilidad se implementó en interfaces posteriores.

A continuación se muestran los dos conectores del estándar que incorporan estas y otras líneas de control.

**Conecotor DB-25 y convenciones de tensión**

El conector DB-25 forma parte del estándar RS-232. En la siguiente imagen podemos ver la forma de este conector.



Figura 14: Conecotor DB-25

Fuente: <https://yerayastor.wordpress.com/2012/09/22/practica-l-conectores-db/>

Los puertos que indica el estándar se muestran en la tabla 1 que aparece a continuación:

<b>PIN</b>	<b>EIA</b>	<b>CCITT / V.24</b>	<b>E/S</b>	<b>Función DTE-DCE</b>
1	CG	AA 101		Tierra del Chasis
2	TD	BA 103	Salida	Datos Transmitidos
3	RD	AA 104	Entrada	Datos Recibidos
4	RTS	CA 105	Salida	Solicitud de Envío
5	CTS	CB 106	Entrada	Listo para Enviar
6	DSR	CC 107	Entrada	Equipo de Datos Listo
7	SG	AB 102	---	Tierra de Señal
8	DCD	CF 109	Entrada	Portadora Detectada
9*			Entrada	Test de Voltaje Positivo
10*			Entrada	Test de Voltaje Negativo
11				(no se usa)
12+	SCDC	SCF 122	Entrada	Portadora Detectada-Secundario
13+	SCTS	SCB 121	Entrada	Listo para Enviar-Secundario
14+	SBA 118		Salida	Datos Transmitidos-Secundario
15#	TC	DB 114	Entrada	Reloj de Transmisión
16+	SRD	SBB 119	Entrada	Datos Recibidos-Secundario
17#	RC	DD 115	Entrada	Reloj de Recepción
18				(no se usa)
19+	SRTS	SCA 120	Salida	Solicitud de Envío Secundario
20	DTR	CD 108,2	Salida	Terminal de Datos Listo
21*	SQ	CG 110	Entrada	Calidad de Señal
22	RI	CE 125	Entrada	Indicador de Timbre
23*	DSR	CH 111	Salida	Equipo de Datos Listo
		CI 112	Salida	Selector de Tasa de Datos
24*	XTC	DA 113	Salida	Reloj de Transmisión Externo
25*			Salida	Ocupado

Tabla 1: Recopilación de pines del conector DB-25

En la tabla 1 los caracteres adyacentes al número de pin simbolizan lo siguiente:

- (\*) se emplea con poca frecuencia.
- (+) usado únicamente si se implementa el canal secundario.
- (#) se emplea únicamente con interfaces síncronas.

Para los propósitos de la RS-232 estándar, una conexión se define mediante un cable desde un dispositivo al otro. Hay 25 pines definidos en la especificación completa, pero es muy probable que se empleen menos de la mitad de estos en una interfaz determinada. La causa es simple,

una interfaz *full-duplex* puede obtenerse con la conexión de tres pines de cada dispositivo únicamente.

Las convenciones de tensión empleadas en el protocolo se muestran en la tabla 2.

Tensión	Señal	Nivel Lógico	Control
+3 a +15	Espacio	0	On
-3 a -15	Marca	1	Off

Tabla 2: Convenciones de tensión

En este caso los valores lógicos se invierten con respecto a los de tensión. Es decir, el valor lógico positivo se corresponde con la tensión negativa y el cero lógico con la tensión positiva.

### Conektor DE-9

Debido a la gran cantidad de pines del conector DB-25 y a la escasa utilización de algunos de ellos, apareció un conector de nueve pines alternativo con prácticamente las mismas funcionalidades.



Figura 15: Conektor DE-9

Fuente: <http://www.eletrodex.com.br/conector-db9-femea-180-solda-placa.html>

En la figura 15 se muestra el aspecto físico de los conectores DB9, mientras que en la tabla 3 se indican cada uno de los pines que se utilizan.

PIN	Nombre	RS232	V.24	E/S	Descripción
1	CD	CF	109	entrada	<i>Carrier Detect</i> , detección de portadora
2	RXD	BB	104	entrada	<i>Receive Data</i> , recepción de datos
3	TXD	BA	103	salida	<i>Transmit Data</i> , transmisión de datos
4	DTR	CD	108.2	salida	<i>Data Terminal Ready</i> , terminal de datos preparado
5	GND	AB	102	tierra	<i>System Ground</i> ó <i>Signal Ground</i> , tierra de señal
6	DSR	CC	107	entrada	<i>Data Set Ready</i> , dispositivo preparado
7	RTS	CA	105	salida	<i>Request to Send</i> , petición de envío
8	CTS	CB	106	entrada	<i>Clear to Send</i> , preparado para transmitir
9	RI	CE	125	entrada	<i>Ring Indicator</i> , indicador de llamada entrante

Tabla 3: Recopilación de pines del conector DE-9

En este caso los estándares de tensión del anterior conector se mantienen.

### 6.6.2 RS-422

El RS-422 (ANSI/TIA/EIA-422-B) es un estándar técnico que especifica características eléctricas de un circuito de señales digitales de voltaje balanceado. El estándar RS-422 es una interfaz diferencial que define niveles de voltaje y especificaciones eléctricas de transmisor receptor. En la figura 16 podemos ver los puertos con tensión balanceada.

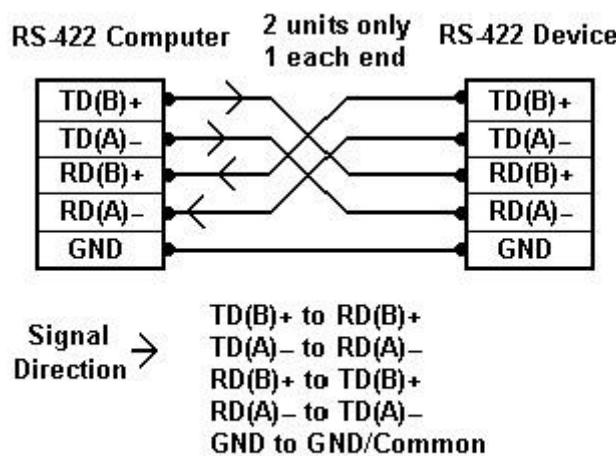


Figura 16: Conexión RS-422 con cuatro líneas

Fuente: <http://www.bb-elec.com/Learning-Center/All-White-Papers/Serial/FAQ-Connect-RS-422-Devices.aspx>



En una interfaz diferencial, los niveles lógicos son definidos por la diferencia de voltaje entre una pareja de señales de entrada o de salida. En contraste, una interfaz de una sola terminación, por ejemplo RS-232, define los niveles lógicos como la diferencia de voltaje entre una señal eléctrica y una conexión a tierra común. Las interfaces diferenciales tienen típicamente mayor inmunidad al ruido o a picos de tensión que pudieran tener lugar en las líneas de comunicación. Las interfaces diferenciales tienen también capacidades de conducción mayores, lo que permite cables más largos.

La interfaz RS-422 alcanza velocidades de transmisión de hasta 10 Mbps a 12 metros, y puede alcanzar longitudes de 1200 metros con una velocidad de 100 kbps. El RS-422 también define las características eléctricas del transmisor y del receptor, permitiendo 1 transmisor y hasta 32 receptores a la vez en la línea. Los niveles de la señal RS-422 van desde 0 a +5 voltios. El estándar RS-422 no define un conector físico concreto.

Un uso común de la EIA-422 es para extender conexiones RS-232. Sistemas de automatización de Broadcast y equipos de post-producción y edición lineal utilizan RS-422 para controlar remotamente los lectores y grabadores ubicados en la sala central de equipos.

Para implementar la norma RS-422 se emplea cable de par trenzado, frecuentemente con cada par blindado. Esto permite una mayor protección ante ruidos e interferencia y por tanto una mayor distancia de transmisión de datos.

### 6.6.3 RS-485

El estándar RS-485 (TIA/EIA-485-A o RS-485) es compatible con las versiones anteriores a RS-422; sin embargo, está optimizado para aplicaciones de línea compartida o multipunto. La salida del controlador RS-422/485 puede estar activa (habilitada) o en tercer estado (inhabilitada en alta impedancia). Esta capacidad permite que múltiples puertos sean conectados en un bus multipunto y sondeados selectivamente.

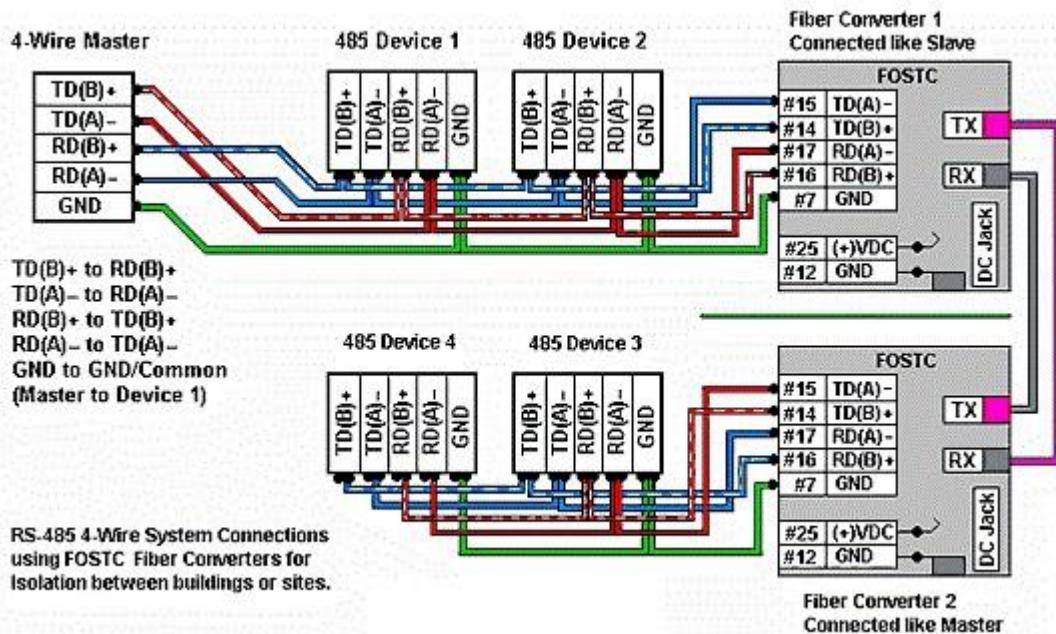


Figura 17: Ejemplo de comunicación serie mediante el estándar RS-485

Fuente: <http://www.bb-elec.com/Learning-Center/All-White-Papers/Industrial-Automation/FAQ-Connect-RS-422-Devices.aspx>

El RS-485 permite longitudes de cable de hasta 1200 metros y velocidades de transmisión de hasta 35 Mbps. Los niveles de señal para RS-485 son los mismos que los definidos para RS-422. RS-485 tiene características eléctricas que permiten a 32 transmisores y 32 receptores estar conectados a la vez a una línea. Esta interfaz es ideal para entornos de comunicaciones multipunto o de red. La salida de tercer estado del controlador RS-485 permite retirar la presencia eléctrica del transmisor/receptor de la línea sin problemas. Sólo uno de los controladores puede estar activo cada vez, y los otros controladores deben tener la salida en tercer estado. La señal RTS de control de salida del módem controla el estado del dispositivo. Algunos sistemas de software de comunicación se refieren al RS-485 como un modo de transferencia RTS activado o RTS bloqueado.

La interfaz RS-485 puede ser cableada de dos formas: con dos cables o con cuatro cables. El modo de conexión mediante dos cables no permite comunicación *full-duplex*, y requiere que los datos sean transferidos en un solo sentido cada vez. Para operaciones *half-duplex*, los dos pines de transmisión deben estar conectados a los dos pines de recepción (TD+ a RD+ y TD- a RD-). El modo de conexión mediante cuatro cables permite la transferencia de datos *full-duplex*. El estándar RS-485 no define un diagrama de conexiones ni un conjunto de señales de control del módem. El RS-485 tampoco define un tipo de conector físico.

Presenta las siguientes aplicaciones:

- La norma RS-485 se usa con frecuencia en las UARTs para la comunicación de datos de baja velocidad dentro de cabinas de aviones. Lo emplean algunas unidades de control de pasajeros y equipos de monitoreo de unidades fotovoltaicas. Requiere poco

cableado y se puede compartir entre varios asientos. Por tanto se reduce el peso del sistema.

- También se emplea en sistemas grandes de sonido, como en los sistemas de conciertos de música y producciones de teatro. Se emplea de forma general para enlazar micrófonos.
- Se emplea para sistemas de automatización de edificios, dada la simplicidad de cableado y larga longitud del cable.
- La norma RS-485 se emplea fundamentalmente en plantas de producción automatizadas.

#### **6.6.4 Comparación entre los estándares de comunicación serie asíncrona más representativos.**

En la tabla 4 se resumen las principales características de los principales protocolos de comunicación serie asíncrona.

Especificaciones	RS-232	RS-422	RS-485
Modo de operación	No balanceado	Diferencial	Diferencial
Número total de transmisores y receptores en una única línea. Solo un transmisor activo a la vez en la norma RS-485	1 Transmisor	1 Transmisor	32 Trasmisores
	1 Receptor	10 Receptores	32 Receptores
Máxima longitud del cable	15 metros	1200 metros	1200 metros
Máxima velocidad de transmisión	160 kbit/s. Pueden alcanzarse velocidades superiores a 1Mbit/s	10 Mbit/s	35 Mbit/s hasta 10 metros y 100 kbit/s en 1200 metros

*Tabla 4: Comparativa de los estándares de comunicación serie*

Los estándares RS-422 y RS-485 utilizan un esquema diferencial, por lo que presentan una mayor inmunidad al ruido permitiendo mayores distancias de comunicación. Por este motivo



se suelen emplear en entornos industriales. Por el contrario el estándar RS-232 por su sencillez se emplea en comunicaciones a corta distancia en sistemas empotrados.

Por otro lado los estándares RS-422 y RS-485 permiten transmitir un volumen de datos mayor y un mayor numero de dispositivos en la interfaz. En cambio el estándar RS-232 permite un volumen de datos mas modesto y una interfaz de comunicación entre dos dispositivos únicamente.

## 6.7 VHDL

VHDL es una lenguaje definido por el IEEE (Institute of Electrical and Electronics Engineers) empleado para la descripción de circuitos digitales. VHDL es un acrónimo que combina VHSIC y HDL, donde VHSIC son las siglas de *Very High Speed Integrated Circuit* y HDL son a su vez las siglas de *Hardware Description Language*.

Se emplea principalmente para describir hardware digital, sintetizable en dispositivos como PLDs (*Programmable Logic Device*), FPGAs (*Field Programmable Gate Array*), ASICs y similares.

Existen diferentes formas de diseñar el mismo circuito en VHDL y es deber del diseñador elegir la más adecuada:

- Funcional: se describe cómo se comporta el circuito. Esta descripción se engloba en procesos que se ejecutan en paralelo entre sí. A su vez estos procesos se ejecutan de manera concurrente con asignaciones de señales e instanciaciones de componentes.
- Flujo de datos: únicamente se describen asignaciones concurrentes de señales.
- Estructural: se forma un circuito mediante la instanciación de componentes. La unión de estos componentes da lugar a diseños de jerarquía superior, al conectar puertos de estos componentes con señales internas del circuito y puertos de jerarquía superior.
- Mixta: combinación de alguna de las anteriores e incluso de todas.

Cabe destacar que en los entornos de desarrollo que emplean el leguaje de descripción de hardware VHDL podemos encontrar plantillas para la implementación de bancos de pruebas, máquinas de estado, filtros digitales, etc. Por lo tanto el VHDL es una gran alternativa para el desarrollo de circuitos digitales.

## 7 Características de la placa de desarrollo

En este capítulo se describe la placa de desarrollo utilizada en este trabajo. Concretamente, se ha utilizado la placa DEO-Nano, cuyo principal componente es la FPGA de Altera EP4CE22F17C6N de la familia Cyclone IV.

### 7.1 Introducción

La placa DEO-Nano introduce una plataforma de desarrollo FPGA de tamaño compacto para un amplio rango de proyectos de diseño portables.

La placa de desarrollo DEO-Nano es ideal para procesadores embebidos ligeros. Incorpora una potente FPGA Cyclone IV (con 22320 elementos lógicos), 32 MB de SDRAM, 2 Kb de memoria EEPROM, y un dispositivo de memoria de 64 bits configurable. Para conectar sensores analógicos la placa DEO-Nano incluye un ADC de 12 bits con 8 canales, y también incorpora un acelerómetro de tres ejes.

La placa de desarrollo incorpora de manera integrada un USB Blaster para la programación de la FPGA. Puede ser alimentada por este mismo USB o por una fuente de energía externa. La placa incluye puertos de expansión para incorporar placas de Terasic u otros dispositivos como motores y actuadores. Incluye dos pulsadores, ocho LEDs y cuatro interruptores DIP.

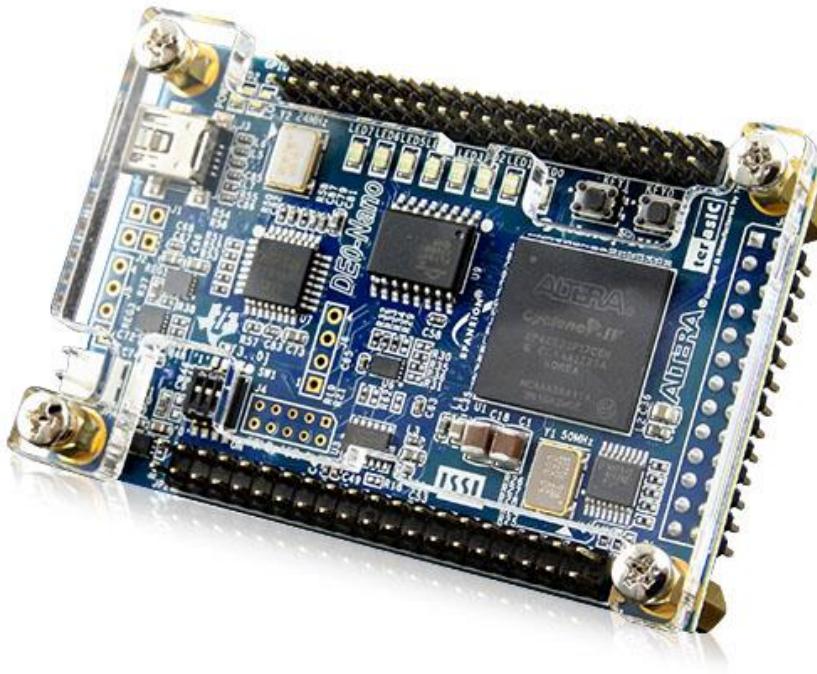


Figura 18: Placa de desarrollo DEO-Nano

Fuente: [https://www.altera.com/en\\_US/pdfs/literature/ug/DE0\\_Nano\\_User\\_Manual\\_v1.9.pdf](https://www.altera.com/en_US/pdfs/literature/ug/DE0_Nano_User_Manual_v1.9.pdf)

## 7.2 Dispositivos de la placa de desarrollo

La placa de desarrollo DE0 Nano incorpora los siguientes dispositivos representados en el esquema de la figura 19.

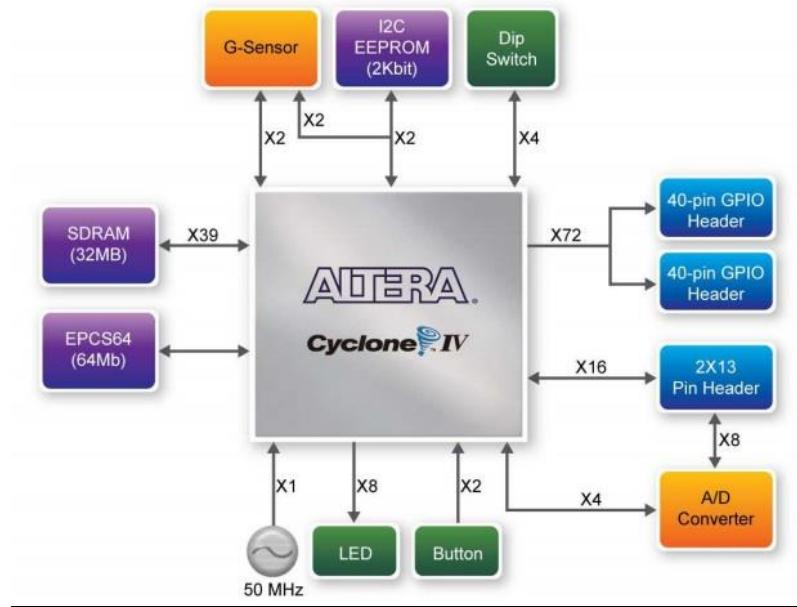


Figura 19: Dispositivos que incluye la placa de desarrollo

Fuente: [https://www.altera.com/en\\_US/pdfs/literature/ug/DE0\\_Nano\\_User\\_Manual\\_v1.9.pdf](https://www.altera.com/en_US/pdfs/literature/ug/DE0_Nano_User_Manual_v1.9.pdf)

Como podemos apreciar la placa de desarrollo incorpora una FPGA de la familia Cyclone IV y una serie de periféricos. Entre los periféricos podemos encontrar un acelerómetro, un convertidor A/D, una memoria no volátil con una interfaz I2C, una memoria RAM, una serie de interruptores y LEDs y una serie de puertos de propósito general.

Además la placa incorpora un oscilador de 50 MHz.

En los siguientes apartados se procede a describir la FPGA y los periféricos anteriormente nombrados.

### 7.2.1 FPGA

Como hemos comentado anteriormente la placa incorpora una FPGA Cyclone IV EP4CE22F17C6N. En la figura 20 podemos apreciar la forma de las FPGAs de la familia y la disposición de sus pines.

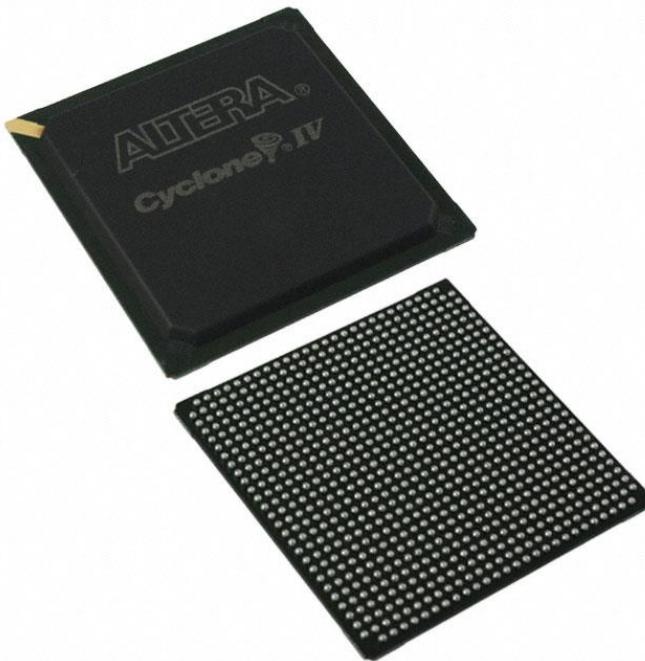


Figura 20: FPGA

Fuente: <http://uk.farnell.com/altera/ep4ce40f23c6n/fpga-cyclone-iv-40k-484fbga/dp/2100842>

Se caracteriza por ser una FPGA de bajo costo y muy bajo consumo de potencia. Solo presenta la posibilidad de incorporar dos fuentes de alimentación.

Presenta varias opciones de voltaje para alimentar el núcleo. Podemos elegir entre 1.2 V y 1.0 voltios para optimizar el rendimiento y la potencia según las necesidades.

La familia Cyclone IV presenta las siguientes características:

- Puede albergar hasta 115K LE (logic elements). En nuestro caso la FPGA presenta 23000 logic elements.
- Incorpora hasta 3.8 megabytes de memoria RAM
- Presenta hasta 266 multiplicadores 18 por 18 integrados
- La familia Cyclone IV Disminuye el consumo en un 25 % con respecto a los modelos Cyclone III. Esta disminución se ve representada en el gráfico de la figura 21.

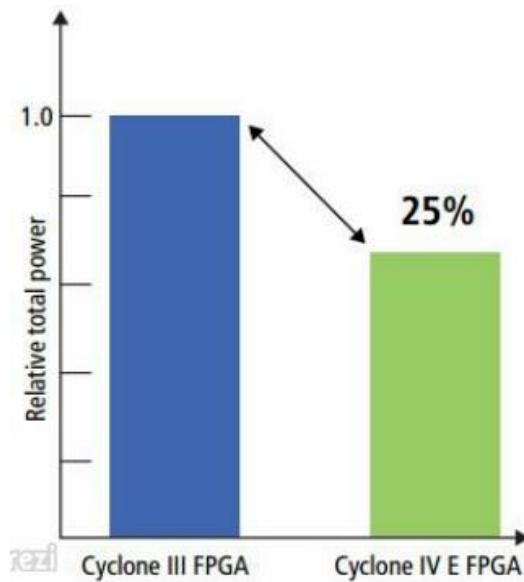


Figura 21: Diferencia de consumo entre la FPGA Cyclone IV E y el modelo anterior

Fuente: [https://www.altera.com/en\\_US/pdfs/literature/hb/cyclone-iv/cyiv-51001.pdf](https://www.altera.com/en_US/pdfs/literature/hb/cyclone-iv/cyiv-51001.pdf)

Los dispositivos I/O de Cyclone IV incorporan bus de espera programable, resistencias de pull-up programables, retardo programable y control de velocidad de respuesta programable para optimizar la integridad de la señal.

### 7.2.2 Sistemas de configuración

La placa de desarrollo incorpora dos mecanismos para configurar la FPGA. En la figura 22 podemos apreciar una representación esquemática de los dos sistemas de configuración.

- Circuito para programar la FPGA USB- Blaster. Integrado en la placa.
- Memoria EPICS64.



Figura 22: USB Blaster y memoria EPCS64

Fuente: [https://www.altera.com/en\\_US/pdfs/literature/ug/DE0\\_Nano\\_User\\_Manual\\_v1.9.pdf](https://www.altera.com/en_US/pdfs/literature/ug/DE0_Nano_User_Manual_v1.9.pdf)

El circuito USB Blaster es necesario para cargar el proyecto en VHDL en la placa. En este caso el circuito viene integrado en nuestra placa de desarrollo. Por otro lado también viene integrado un sistema de configuración serial para la memoria EPROM. Esta memoria es no volátil y permite que el proyecto cargado en esta se ejecute cuando la placa recibe alimentación.

### 7.2.3 Puertos de expansión

La placa utilizada dispone de dos zonas con puertos de expansión. Entre las dos zonas suman 80 pines. De estos 80 pines, 72 son puertos de propósito general, 2 son pines con 5 voltios, otros dos presentan 3,3 voltios y cuatro pines conectados a tierra. En la figura 23 podemos apreciar un esquema con los nombres de los puertos de propósito general asociados a sus pines correspondientes.

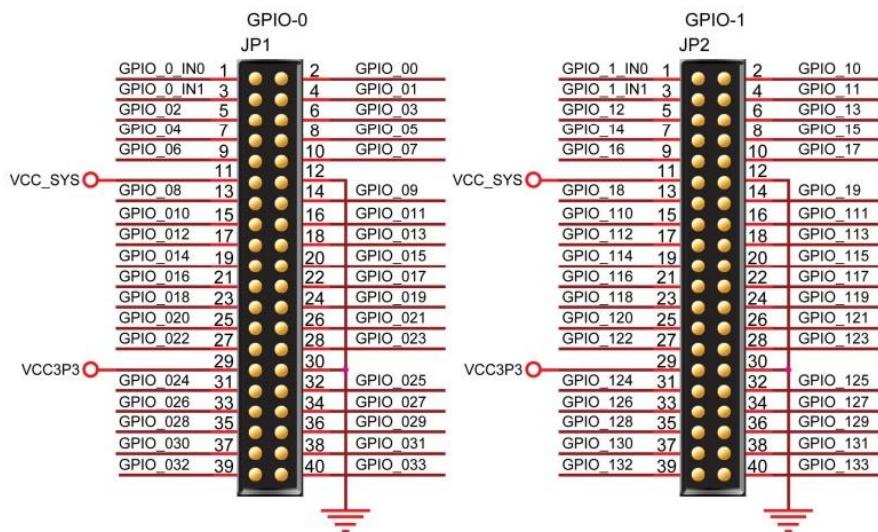


Figura 23: Puertos de propósito general

Fuente: [https://www.altera.com/en\\_US/pdfs/literature/ug/DE0\\_Nano\\_User\\_Manual\\_v1.9.pdf](https://www.altera.com/en_US/pdfs/literature/ug/DE0_Nano_User_Manual_v1.9.pdf)

En estos puertos se realizará la conexión de un teclado matricial y un dispositivo USB-UART

### 7.2.4 Dispositivos de memoria

La placa DEO Nano dispone de dos dispositivos de memoria, que pueden ser utilizados por los circuitos implementados. Una memoria SDRAM y una memoria EEPROM con interfaz serie I2C. La memoria SDRAM es una memoria volátil de mayor capacidad para poder almacenar información mientras la placa recibe alimentación. La memoria EEPROM tiene una menor capacidad, pero es una memoria no volátil que permite almacenar datos aunque la placa deje de recibir alimentación. Las figuras 24 y 25 muestran esquemáticamente las conexiones de las memorias con la FPGA.

- Memoria SDRAM de 32 MB



Figura 24: RAM

Fuente: [https://www.altera.com/en\\_US/pdfs/literature/ug/DE0\\_Nano\\_User\\_Manual\\_v1.9.pdf](https://www.altera.com/en_US/pdfs/literature/ug/DE0_Nano_User_Manual_v1.9.pdf)

- Memoria EEPROM de 2Kb con interfaz I2C

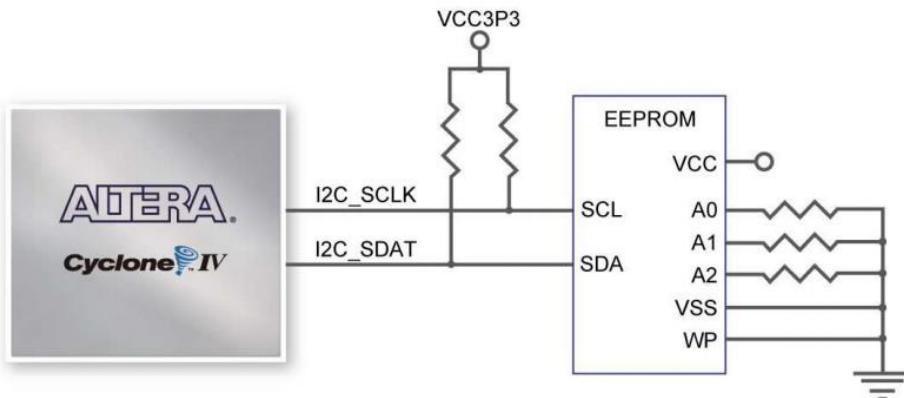


Figura 25: Memoria EEPROM

Fuente: [https://www.altera.com/en\\_US/pdfs/literature/ug/DE0\\_Nano\\_User\\_Manual\\_v1.9.pdf](https://www.altera.com/en_US/pdfs/literature/ug/DE0_Nano_User_Manual_v1.9.pdf)

### 7.2.5 Dispositivos de entrada y salida

La placa cuenta con varios dispositivos de entrada/salida que permiten interactuar con el diseño implementado en la FPGA. Dentro de estos dispositivos están los LEDs que nos servirán para llevar a cabo la prueba de funcionamiento del circuito. También podemos encontrar los interruptores DIP. Emplearemos uno de estos interruptores como RESET de la UART. En las figuras 26 y 27 se presentan los esquemas de conexiones de los LEDs y los interruptores pulsadores.

- Ocho LEDs verdes

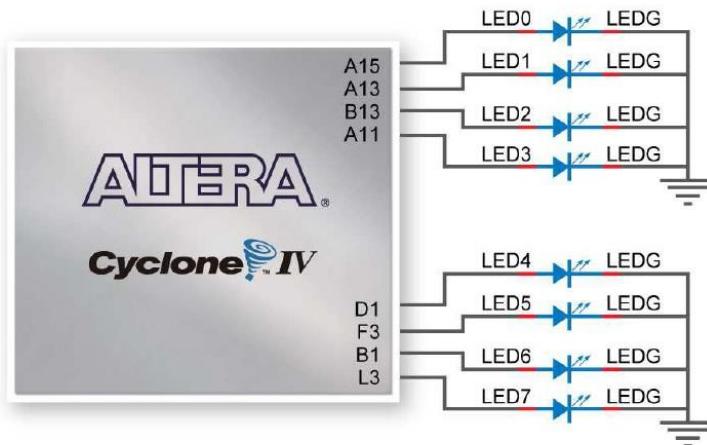


Figura 26: LEDs

Fuente: [https://www.altera.com/en\\_US/pdfs/literature/ug/DE0\\_Nano\\_User\\_Manual\\_v1.9.pdf](https://www.altera.com/en_US/pdfs/literature/ug/DE0_Nano_User_Manual_v1.9.pdf)

Como podemos apreciar los LEDs son de cátodo común. Dado que emplearemos los LEDs para demostrar el funcionamiento de la UART lo debemos tener en cuenta.

- Dos pulsadores

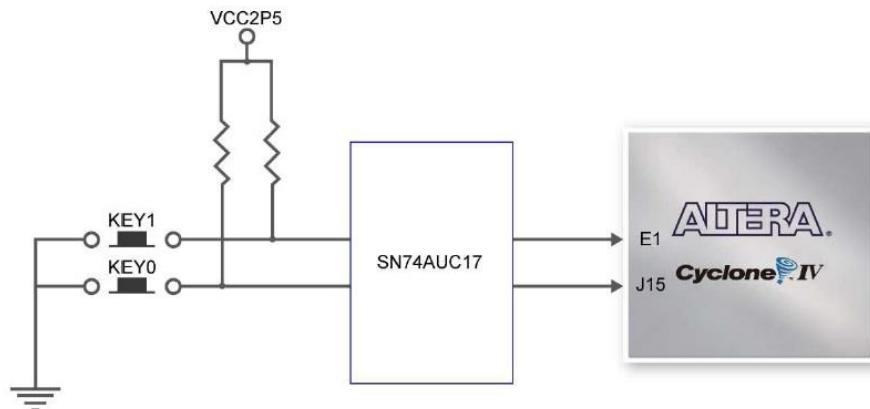


Figura 27: Pulsadores

Fuente: [https://www.altera.com/en\\_US/pdfs/literature/ug/DE0\\_Nano\\_User\\_Manual\\_v1.9.pdf](https://www.altera.com/en_US/pdfs/literature/ug/DE0_Nano_User_Manual_v1.9.pdf)

Los pulsadores presentan resistencias de *pull-up*. Por tanto proporcionarán un nivel lógico uno cuando no estén pulsados y un nivel lógico 0 cuando estén pulsados.

- Cuatro interruptores DIP

### 7.2.6 Sensor G

La placa de desarrollo incorpora un acelerómetro capaz de medir la aceleración en un rango de más menos 16 g. presenta dos interfaces de comunicación serie, SPI e I2C. Mantiene un consumo de 23  $\mu$ A en modo medición y 0,1  $\mu$ A en modo espera. A continuación se incluye la figura 28, con el esquema de conexiones del acelerómetro con la FPGA.

- ADI ADXL345, acelerómetro de tres ejes de alta resolución (13 bits).

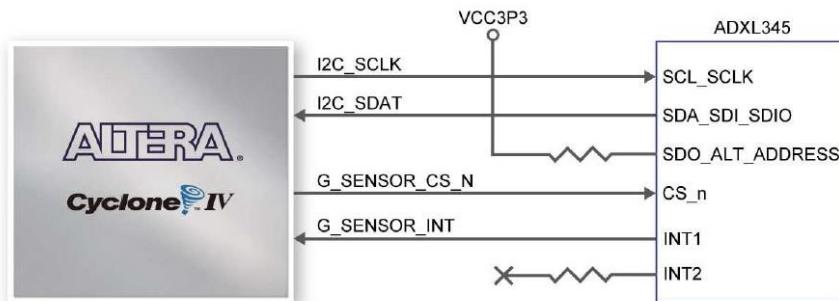


Figura 28: Acelerómetro

Fuente: [https://www.altera.com/en\\_US/pdfs/literature/ug/DE0\\_Nano\\_User\\_Manual\\_v1.9.pdf](https://www.altera.com/en_US/pdfs/literature/ug/DE0_Nano_User_Manual_v1.9.pdf)

### 7.2.7 Convertidor A/D

El convertidor A/D presenta una resolución de doce bits y es capaz de medir señales analógicas en ocho canales distintos. Entre otras interfaces presenta una interfaz de comunicación serie SPI que nos permite comunicarnos fácilmente con el dispositivo. El dispositivo consigue la conversión A/D mediante aproximaciones sucesivas. En la figura 29 se aprecia el esquema de conexiones del convertidor A/D con la FPGA.

- NS ADC128S022 Convertidor ADC de doce bits, ocho canales.

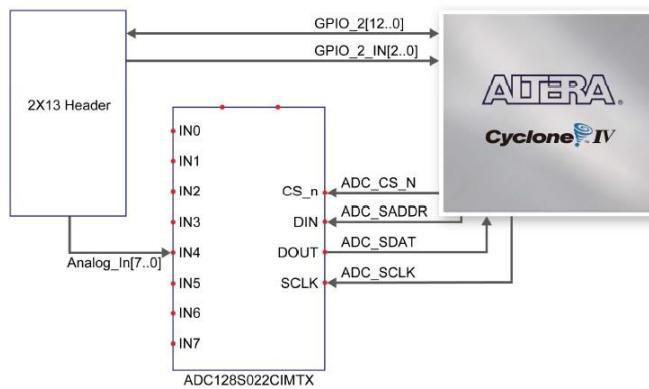


Figura 29: ADC

Fuente: [https://www.altera.com/en\\_US/pdfs/literature/ug/DE0\\_Nano\\_User\\_Manual\\_v1.9.pdf](https://www.altera.com/en_US/pdfs/literature/ug/DE0_Nano_User_Manual_v1.9.pdf)



### 7.2.8 Sistema de reloj

El dispositivo DE0-Nano incorpora un oscilador de 50 Mhz. Este oscilador será preescalado en función de la velocidad de transmisión de datos de la UART.

- Oscilador integrado de 50 MHz

### 7.2.9 Fuentes de alimentación

El dispositivo de desarrollo presenta dos fuentes de alimentación para la placa. Además de configurar la FPGA, el puerto USB permite alimentarla. Por otro lado el dispositivo cuenta con dos pines que permiten una alimentación de 5 voltios. Estos dos pines permiten incorporar al diseño una batería externa.

Ademas la placa de desarrollo incorpora una serie de pines de alimentación para dispositivos externos que se incorporen a la placa.

- Puerto USB tipo mini A-B (5 voltios)
- Dos pines de 5 voltios. Uno por cada zona de puertos de propósito general (2 zonas)
- Dos pines de alimentación externa. Alimentación en un rango entre 3.6 y 5.7 voltios.

## 8 Quartus II

Quartus II es una herramienta de software proporcionada por Altera para la síntesis y análisis de diseños descritos en lenguajes de descripción de hardware. En la figura 30 se puede apreciar la distribución del entorno de desarrollo.

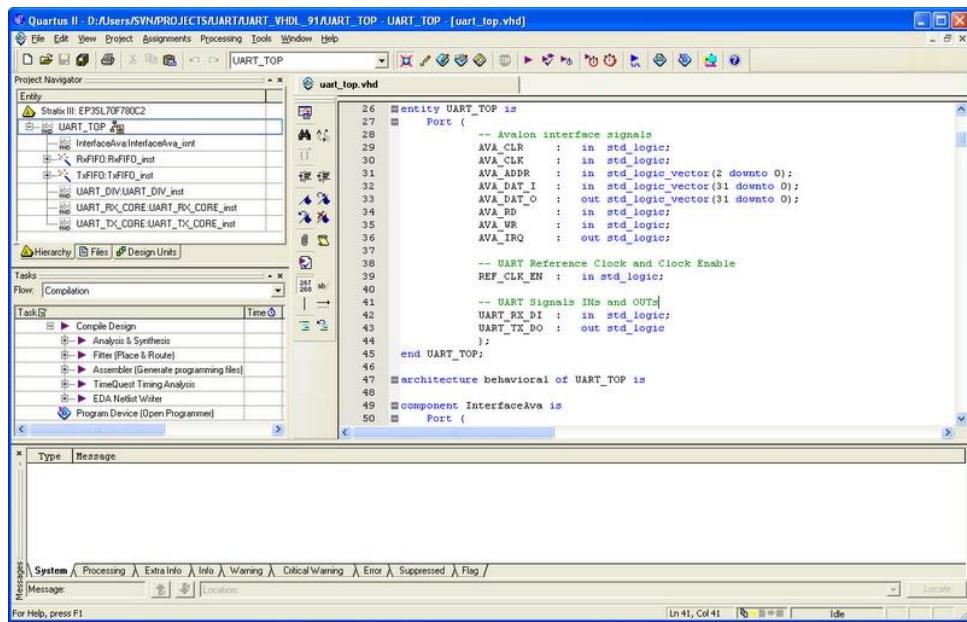


Figura 30: Página principal de Quartus II

Quartus II permite al desarrollador compilar sus diseños depurando errores, realizar análisis temporales, generar diagramas RTL y configurar el dispositivo de destino.

Además permite la simulación de los diseños digitales mediante la herramienta Modelsim. Son posibles tanto simulaciones funcionales, como simulaciones teniendo en cuenta el retardo de los componentes de la FPGA.

Podemos encontrar dos versiones distintas de Quartus II:

- La Edición de Suscripción con licencia de pago, versión recomendada para empresas, grandes desarrolladores y para FPGAs de alta gama.
- Por otro lado encontramos la edición Web que puede ser descargada de manera gratuita. Esta edición permite la compilación y la programación de un conjunto limitado de FPGAs. Entre este conjunto se incluye la gama Cyclone, indicada para pequeños desarrolladores. Únicamente se requiere una licencia gratuita que se renueva ilimitadamente.

En nuestro caso, dado que disponemos de una placa de desarrollo que incorpora una FPGA de la gama Cyclone (Cyclone IV) emplearemos la edición web.

Cabe destacar que la herramienta dispone de todo lo necesario para la implementación de nuestro circuito. Disponemos de diversas utilidades, que agilizan el desarrollo, que nombramos a continuación:

- Disponemos de una amplia librería de bloques como pueden ser registros, multiplexores, etc.
- La herramienta incorpora una gran cantidad de plantillas en VHDL que se pueden ajustar a las necesidades de nuestro diseño.
- Dentro del entorno de diseño podemos encontrar herramientas para la descripción de bancos de pruebas, esenciales para las simulaciones.

A continuación se describe la herramienta principal para realizar simulaciones a partir de bancos de pruebas.

## 8.1 ModelSim-Altera Starter Edition

Además de las funcionalidades anteriores Quartus II incorpora la herramienta ModelSim-Altera Starter Edition de forma gratuita. Esta herramienta nos permitirá, junto a Quartus II, realizar todas las simulaciones de los distintos componentes que forman nuestro diseño.

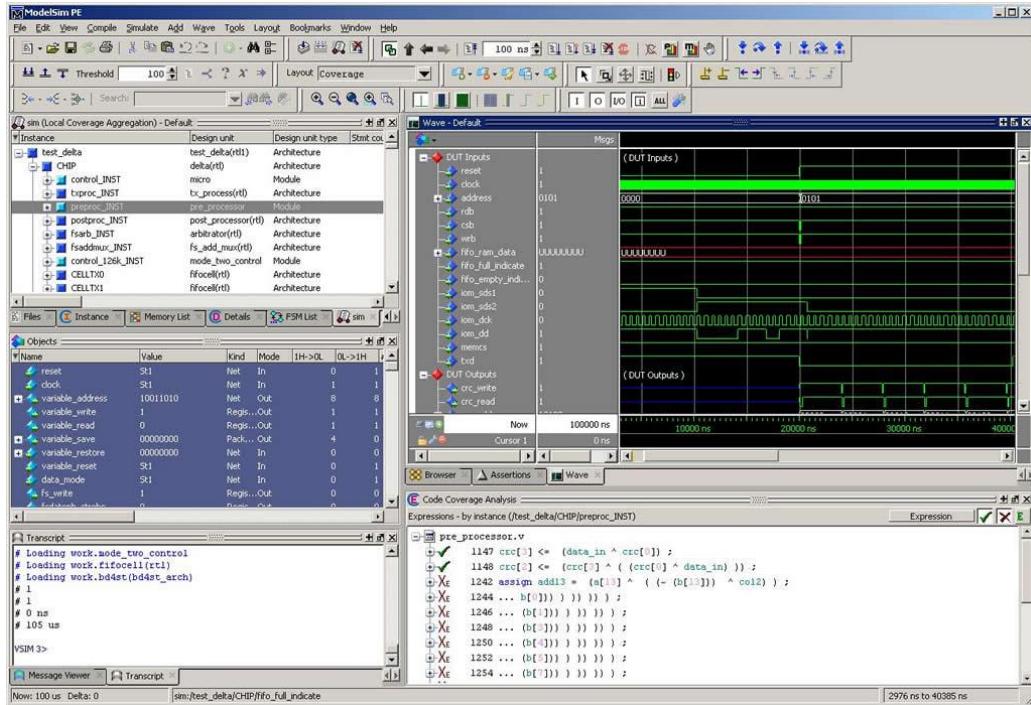


Figura 31: Menú principal de ModelSim

## 9 Dispositivo elegido para realizar el proyecto

En este proyecto vamos a emplear el PIC18F2525 de Microchip, como base para ser capaces de diseñar una UART. Hemos optado por esta elección dado que consideramos que este microcontrolador presenta una USART con la complejidad adecuada para poder abarcar el proyecto. Cabe remarcar que este dispositivo solo será una base y que durante el proyecto se realizarán una serie de modificaciones y simplificaciones con respecto al dispositivo de origen.

Serán de gran utilidad los cronogramas y tablas de la USART presentes en la documentación del microcontrolador (PIC18F2525/2620/4525/4620 Data Sheet, 2004). Las tablas las emplearemos para localizar cada una de las entradas y salidas que componen la entidad del proyecto. Además descubriremos gran cantidad de señales necesarias en el momento de describir hardware en VHDL. Además los cronogramas serán de gran utilidad para comprender de una forma más gráfica, el funcionamiento de algunos de los componentes del proyecto.

### 9.1 Descripción de la USART del microcontrolador PIC18F2525

La USART es uno de los dos módulos de comunicación serie presentes en el microcontrolador PIC18F2525. Se puede configurar como un sistema de comunicación full dúplex asíncrono que interactúa con diversos periféricos, como un terminal CRT o un ordenador personal. También se puede configurar como una interfaz de comunicación *half-duplex* síncrono que puede comunicarse con otros periféricos como ADCs y DACs, memorias EEPROM, etc.

La USART implementa dispositivos adicionales, incluyendo un detector y calibrador automático de *baud rate*, activación automática ante la recepción de un carácter y la posibilidad de transmitir 12 bits de parada.

La USART puede ser configurada en los siguientes modos:

- Modo asíncrono (*full-dúplex*):

Presenta autoactivación ante la recepción de un carácter.

Autocalibración de *baud rate*.

Posibilita la transmisión de 12 bits de parada.

- Modo síncrono (*half-duplex*):

En modo maestro, con la posibilidad de elegir la polaridad del reloj

En modo esclavo, con la posibilidad de elegir la polaridad del reloj



Las operaciones del módulo USART son controladas a través de tres registros:

- TXSTA: registro de estado y control de transmisión
- RCSTA: registro de estado y control de recepción
- BAUDCON: registro de control de *baud rate*

De estos tres registros podemos concluir que las tres partes principales que componen la USART son el transmisor, el receptor y el generador de baudios.

## 9.2 UART: USART en modo asíncrono

En este proyecto solo describiremos en VHDL la parte de la USART correspondiente a la comunicación asíncrona, dato que abarcar ambos modos de funcionamiento se sale de los objetivos del proyecto.

Como ya hemos descrito en apartados anteriores la UART emplea un estándar de comunicación NRZ (*Non-Return-to-Zero*). Este estándar consiste en el envío de un bit de START, ocho o nueve bits de datos y un bit de STOP. El formato más común es el de ocho bits. Esta USART incorpora un dispositivo con dos registros de ocho bits, denominado generador de *baud rate*, el cual se emplea para derivar frecuencias estándar de *baud rate* del reloj.

El primer bit en ser transmitido es el menos significativo. El transmisor y receptor son funcionalmente independientes, pero emplean el mismo formato de datos y el mismo *baud rate*. El generador de baudios genera un reloj que es 4, 16, o 64 veces más rápido que el registro de desplazamiento presente en el transmisor y el receptor. El dispositivo no incorpora el control de paridad por hardware, pero puede ser implementada en software y almacenada en el noveno bit.

Cuando se opera en modo asíncrono, el módulo USART incorpora los siguientes elementos importantes:

- Generador de Baudios
- Circuito de muestreo
- Transmisor asíncrono
- Receptor asíncrono
- Autoactivación ante la llegada de un bit de sincronización
- Posibilidad de transmisión de 12 bits de parada
- Autodetección de *baud rate*

A continuación se puede apreciar una descripción de los elementos más importantes de la USART del microcontrolador PIC18F2525

### 9.2.1 Generador de Baudios

El generador de baudios presenta dos registros de 8 bits y puede funcionar tanto en modo asíncrono como síncrono. Por defecto funciona con un único registro de ocho bits, poniendo a uno el registro BRG16 para emplear los dos registros.

El par de registros de ocho bits (SPBRGH y SPBRG) registra el valor máximo de un temporizador que determina el periodo final de la salida del generador de baudios. Los registros BRGH y BRG16 también determinan el *baud rate* final.

La siguiente tabla muestra las fórmulas para calcular el *baud rate* en diferentes modos de funcionamiento.

TABLE 18-1: BAUD RATE FORMULAS

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-bit/Asynchronous	Fosc/[64 (n + 1)]
0	0	1	8-bit/Asynchronous	
0	1	0	16-bit/Asynchronous	
0	1	1	16-bit/Asynchronous	
1	0	x	8-bit/Synchronous	
1	1	x	16-bit/Synchronous	Fosc/[4 (n + 1)]

Legend: x = Don't care, n = value of SPBRGH:SPBRG register pair

Tabla 5: Fórmulas para calcular el valor máximo de los registros SPBRGH y SPBRG

Fuente: <http://ww1.microchip.com/downloads/en/DeviceDoc/39626e.pdf>

Dada la frecuencia del reloj del dispositivo y el *baud rate* deseado, puede ser calculado el valor máximo que será almacenado en los registros SPBRGH y SPBRG. Además a partir de las formulas se puede calcular el error que se va producir comparando el *baud rate* deseado y el obtenido realmente.

Para reducir este error es aconsejable poner el registro BRGH o BRG16 a uno. También es interesante este método para obtener un *baud rate* más lento con frecuencias de reloj altas.

### 9.2.2 Transmisor asíncrono

El diagrama del bloque transmisor se muestra en la figura 32.

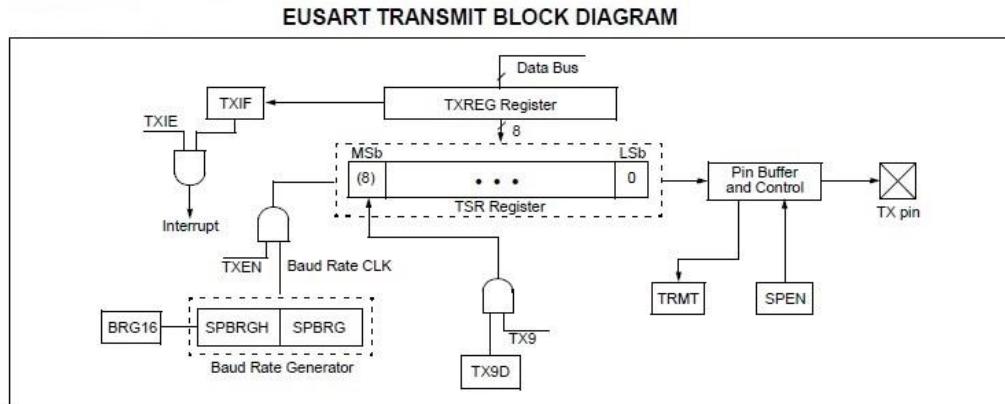


Figura 32: Transmisor asíncrono

Fuente: <http://ww1.microchip.com/downloads/en/DeviceDoc/39626e.pdf>

La parte principal del módulo transmisor es el registro con desplazamiento (TSR register). Este registro obtiene los datos de un registro que funciona como un buffer que almacena el byte que entra al dispositivo transmisor. Se denomina TXREG register. El registro TXREG se carga con datos software, provenientes del procesador del microcontrolador. El registro TSR no se carga hasta que el bit de STOP de la anterior transmisión no hay sido transmitido. En el momento en el que el bit de STOP es transmitido, el registro TSR se carga con los datos del registro TXREG, en el caso de que contengan datos.

Una vez el registro TXREG transfiere los datos al registro TSR, el registro TXREG está vacío y el *flag* TXIF adquiere valor lógico uno. La interrupción generada por este *flag* puede ser activada o desactivada con el *enable* TXIE. A pesar de esto, aunque la señal TXIE este desactivada, el *flag* TXIF será activado o desactivado independientemente. Por otro lado el *flag* TXIF no se pone a cero inmediatamente en el momento en el que se carga el registro TXREG. Se pone a cero en el segundo ciclo de instrucción siguiente a la instrucción de carga. Poner a cero inmediatamente el *flag* TXIF daría lugar a obtener resultados erróneos.

Mientras que el *flag* TXIF indica el estado del registro TXREG, el *flag* TRMT indica el estado del registro TSR. TRMT es un bit de solo lectura que se pone a uno cuando el registro TSR está vacío. Este *flag* no tiene ninguna interrupción asociada, por tanto el usuario debe muestrearlo para determinar si el registro TSR está vacío.

Para establecer una transmisión asíncrona se necesitan los siguientes pasos, resumidos en la figura 33:

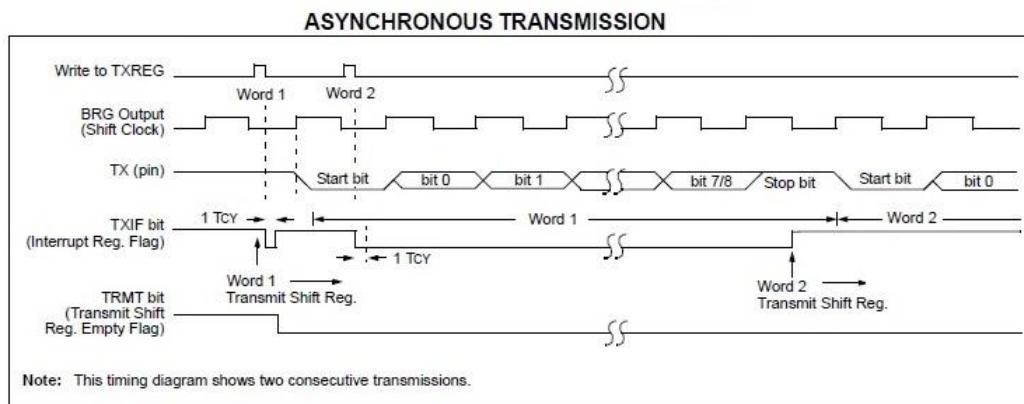


Figura 33: Transmisión de dos caracteres

Fuente: <http://ww1.microchip.com/downloads/en/DeviceDoc/39626e.pdf>

1. Inicializar el registro SPBRGH: SPBRG con un valor para obtener el *baud rate* deseado. Poner a uno o a cero los registros BRGH y BRG16, como sea requerido para obtener el *baud rate* deseado.
2. Activar el dispositivo serial asíncrono poniendo a cero el bit SYNC y poniendo a uno el bit SPEN.
3. Si se desean activar las interrupciones, activar el bit TXIE
4. Si se desea un noveno bit de transmisión, activar el registro TX9 (presente en el byte TXSTA).
5. Activar la transmisión activando el bit TXEN (TXSTA), lo que también activara el flag TXIF.
6. Si hemos elegido el modo de transmisión con 9 bits, el noveno dato debe ser cargado en el registro TX9D (TXSTA).
7. Cargar el registro TXREG para comenzar la transmisión

A continuación se incluyen los registros encargados de controlar la transmisión asíncrona:

REGISTERS ASSOCIATED WITH ASYNCHRONOUS TRANSMISSION								
Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON	GIE/GIEH	PEIE/GIEL	TMROIE	INTOIE	RBIE	TMR0IF	INT0IF	RBIF
PIR1	PSP1F <sup>(1)</sup>	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE1	PSP1E <sup>(1)</sup>	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
IPR1	PSP1P <sup>(1)</sup>	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
TXREG	EUSART Transmit Register							
TXSTA	CSRC	TX9	TXEN	SYNC	SENDDB	BRGH	TRMT	TX9D
BAUDCON	ABDOVF	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN
SPBRGH	EUSART Baud Rate Generator Register High Byte							
SPBRG	EUSART Baud Rate Generator Register Low Byte							

**Legend:** — = unimplemented locations read as '0'. Shaded cells are not used for asynchronous transmission.

**Note 1:** These bits are unimplemented on 28-pin devices and read as '0'.

*Tabla 6: Registros que utiliza el transmisor asíncrono*

Fuente: <http://ww1.microchip.com/downloads/en/DeviceDoc/39626e.pdf>

### 9.2.3 Receptor asíncrono

El diagrama del bloque receptor se muestra en la figura 34:

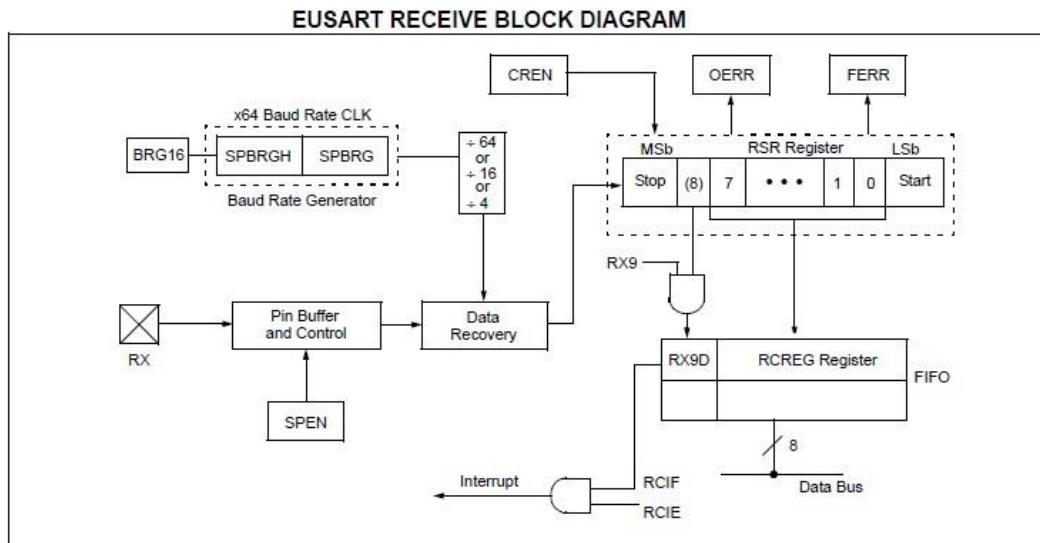


Figura 34: Receptor asíncrono

Fuente: <http://ww1.microchip.com/downloads/en/DeviceDoc/39626e.pdf>

Los datos se reciben en el pin RX. La parte principal del bloque es un registro con desplazamiento (RSR register) que opera a una velocidad cuatro, diecisésis o sesenta y cuatro veces el *baud rate*. De esta forma muestrea el bit recibido en el momento óptimo para evitar fallos. Por otro lado el registro RCREG se encargará de agrupar los ocho o nueve bits cuando se completa la recepción de un byte. En este caso el bloque receptor incorpora una memoria FIFO para acumular los bytes recibidos.

El *flag* RCIF se pone a uno cuando la memoria FIFO tiene algún byte y se pone a cero cuando todos los bytes han sido leídos. El igual que en el transmisor, el *flag* RCIF está asociado a una interrupción que se activa y desactiva con la señal RCIE.

Para que se produzca una recepción se deben realizar los siguientes pasos:

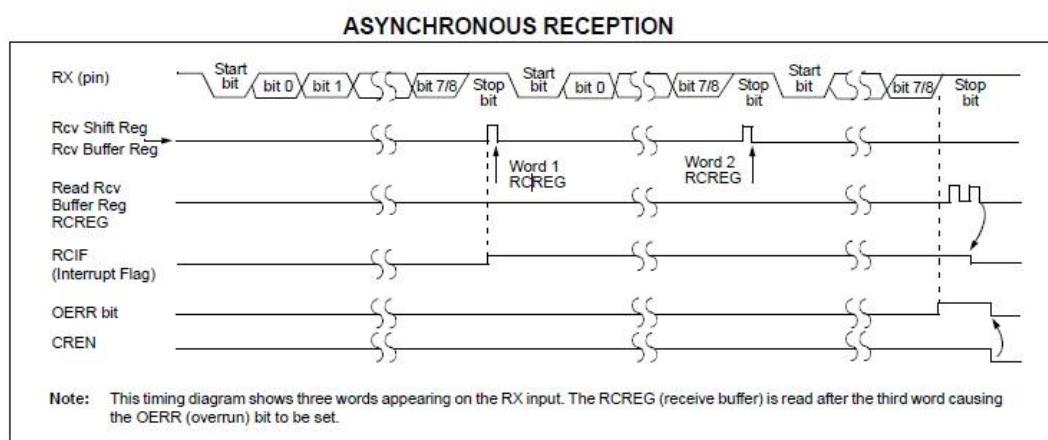


Figura 35: Recepción de dos caracteres

Fuente: <http://ww1.microchip.com/downloads/en/DeviceDoc/39626e.pdf>

1. Inicializar los registros SPBRGH y SPBRG con el valor para obtener el *baud rate* deseado.
2. Habilitar el puerto serie asíncrono poniendo a cero el registro SYNC y poniendo a uno el registro SPEN.
3. Si se requieren interrupciones poner a uno RCIE.
4. Si se desea recibir un noveno bit, poner a uno el bit RX9.
5. Habilitar la recepción poniendo el registro CREN a uno.
6. El *flag* RCIF se pondrá a uno cuando la recepción de un byte se complete y se activará una interrupción, si el registro de habilitación RCIE tiene valor lógico uno.
7. Se lee el registro RCSTA para obtener el valor del 9 bit en caso de que este habilitado, y se comprueba si algún error ha ocurrido.



8. Se obtienen los 8 bits recibidos, leyendo el registro RCREG.
9. Si algún error ocurrido, se borra el error poniendo a cero el bit CREN

Los registros encargados de controlar el bloque receptor se indican en la tabla 7:

**REGISTERS ASSOCIATED WITH ASYNCHRONOUS RECEPTION**

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON	GIE/GIEH	PEIE/GIEL	TMROIE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
PIR1	PSP1F <sup>(1)</sup>	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE1	PSP1E <sup>(1)</sup>	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
IPR1	PSP1P <sup>(1)</sup>	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
RCREG	EUSART Receive Register							
TXSTA	CSRC	TX9	TXEN	SYNC	SENDDB	BRGH	TRMT	TX9D
BAUDCON	ABDOVF	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN
SPBRGH	EUSART Baud Rate Generator Register High Byte							
SPBRG	EUSART Baud Rate Generator Register Low Byte							

**Legend:** — = unimplemented locations read as '0'. Shaded cells are not used for asynchronous reception.

**Note 1:** These bits are unimplemented on 28-pin devices and read as '0'.

*Tabla 7: Registros que emplea el receptor asíncrono*

Fuente: <http://ww1.microchip.com/downloads/en/DeviceDoc/39626e.pdf>

## 10 Descripción del dispositivo implementado en VHDL

A continuación vamos a realizar una descripción de cada uno de los bloques que componen la UART que hemos diseñado. Antes de comenzar con cada bloque vamos a analizar un esquema global de todo el proyecto en conjunto. Así podremos ver de una manera sencilla los bloques que componen el proyecto, así como las conexiones entre los diferentes bloques.

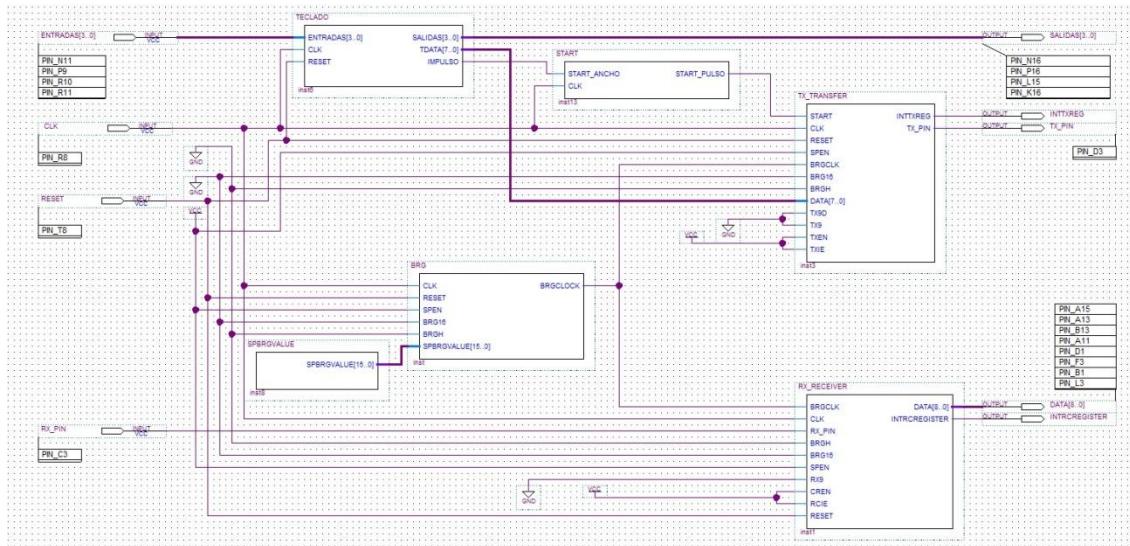


Figura 36: Esquema de la UART implementada

Como se puede apreciar en el esquema de la figura 36, podemos diferenciar los siguientes bloques:

- BRG o BAUD RATE GENERATOR
- TX\_TRANSFER
- RX\_RECEIVER
- TECLADO
- START
- SPBRGVALUE

Es interesante comentar que los bloques que componen propiamente la UART son los bloques BRG, TX\_TRANSFER y RX\_RECEIVER. El resto de los bloques se emplean para mostrar el funcionamiento de la UART. Los bloques TECLADO y START se emplean para mandar información al transmisor (TX\_TRANSFER) e iniciar la comunicación. Para finalizar, el bloque SPBRGVALUE se emplea para configurar el *baud rate* del bloque BRG.

En global la UART obtiene los siguientes resultados al realizar la síntesis de todos los bloques:

Flow Summary	
Flow Status	Successful - Mon Jul 27 12:55:20 2015
Quartus II 64-Bit Version	14.0.0 Build 200 06/17/2014 SJ Web Edition
Revision Name	uart
Top-level Entity Name	uart
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	176 / 22,320 (< 1 %)
Total combinational functions	162 / 22,320 (< 1 %)
Dedicated logic registers	122 / 22,320 (< 1 %)
Total registers	122
Total pins	23 / 154 ( 15 % )
Total virtual pins	0
Total memory bits	0 / 608,256 ( 0 % )
Embedded Multiplier 9-bit elements	0 / 132 ( 0 % )
Total PLLs	0 / 4 ( 0 % )

Figura 37: Resultados de la síntesis de la UART

Como podemos comprobar en la imagen 37, el empleo de recursos de la UART es mínimo. El empleo de elementos lógicos no alcanza el 1%. Por tanto sería un dispositivo interesante para incorporar a otros proyectos dado que su consumo de recursos es escaso.

A continuación se realiza una descripción de cada bloque individualmente.

## 10.1 BRG o Baud Rate Generator

El generador de baudios se resume en un dispositivo que se encarga de preescalar el reloj principal. Mediante el reloj que viene incorporado en el dispositivo (ya sea un microcontrolador o una FPGA) obtenemos una señal de reloj con una frecuencia inferior. Además no solo nos permite reducir la frecuencia del reloj principal. Además nos permite seleccionar la frecuencia que nosotros queremos obtener mediante la introducción de un número calculado mediante una serie de fórmulas preestablecidas. Esto nos permite conseguir el *baud rate* requerido para la correcta sincronización de los dispositivos que realizan la comunicación. A continuación se realiza una descripción detallada del generador de baudios.

### 10.1.1 Entidad

En la figura 38 se muestra un esquema de la entidad del generador de baudios. La salida BRGCLOCK proporciona a la UART una señal de reloj con una frecuencia acorde el *baud rate* elegido.

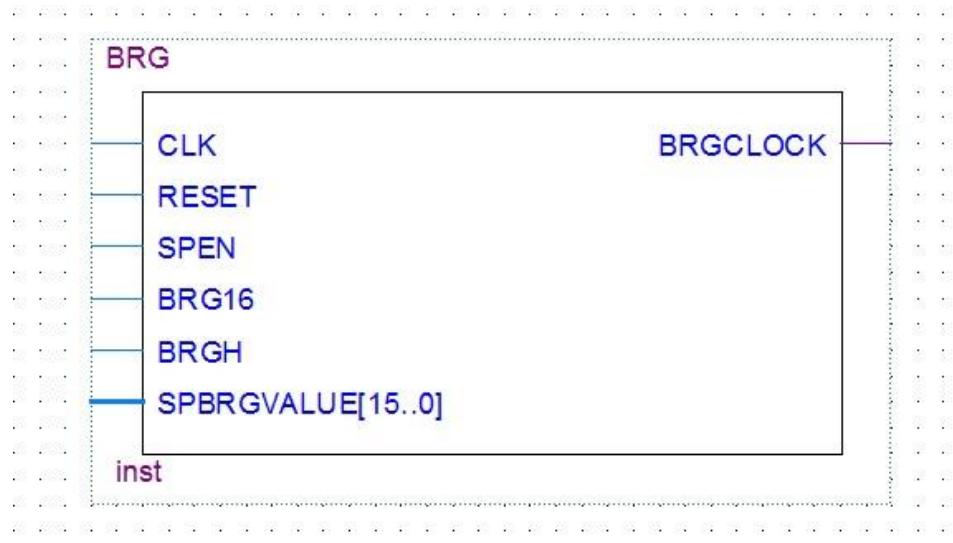


Figura 38: Bloque correspondiente al generador de baudios

### 10.1.2 Funcionamiento del generador de baudios

El Baud Rate Generator se sirve de un contador para obtener la frecuencia del reloj de salida. Cuando la cuenta de este contador alcanza la mitad del valor de la entrada SPBRGVALUE, la señal BRGCLOCK cambia de 1 a 0 o de 0 a 1, según corresponda. Así obtenemos un reloj con el periodo adecuado.

Cabe destacar que el generador de baudios genera un reloj 4, 16 o 64 veces con más frecuencia que la frecuencia de transmisión. Esto es necesario sobre todo en el receptor dado que necesitamos muestrear cada bit que llega en el medio. Por tanto necesitamos un reloj, por parte del generador de baudios, que nos permita contar ciclos de reloj y muestrear justo en el flanco del reloj que este en el medio del bit transmitido. Esto permite reducir errores.



### 10.1.3 Puertos y señales

Los diferentes puertos y señales de la entidad del generador de baudios se resumen en la tabla 8 y 9.

Puertos	E/S	Tipo	descripción
CLK	entrada	std_logic	Reloj del circuito.
RESET	entrada	std_logic	Reset del circuito
SPEN	entrada	std_logic	Habilita el funcionamiento del puerto serie
BRG16	entrada	std_logic	Determina si se emplea un temporizador de 8 o 16 bits para calcular el <i>baud rate</i>
BRGH	entrada	std_logic	Determina la velocidad del BRG
SPBRGVALUE	entrada	std_logic_vector(15 downto 0)	Valor calculado a partir de las formulas del <i>baud rate</i>
BRGCLOCK	salida	std_logic	Reloj con la frecuencia deseada. Determinada por el valor SPBRGVALUE

Tabla 8: Puertos que forman parte de la entidad del bloque BRG

señales		Tipo	descripción
SPBRGH		unsigned(7 downto 0)	Registro más significativo donde se almacena la cuenta del temporizador
SPBRG		unsigned(7 downto 0)	Registro menos significativo donde se almacena la cuenta del temporizador
UNSPBRGVALUE		unsigned(15 downto 0)	Señal donde se almacena el valor del registro SPBRGVALUE menos 1
S_BRGCLOCK		std_logic	Señal necesaria para generar la salida BRGCLOCK. Son idénticas.

Tabla 9: Señales del bloque BRG

#### 10.1.4 Resultados de la síntesis

Como podemos comprobar en los resultados de la síntesis la cantidad de elementos lógicos empleados es reducida. Estos resultados se indican en la figura 39.

Flow Summary	
Flow Status	Successful - Mon Jul 27 13:04:10 2015
Quartus II 64-Bit Version	14.0.0 Build 200 06/17/2014 SJ Web Edition
Revision Name	BRG
Top-level Entity Name	BRG
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	58 / 22,320 (< 1 %)
Total combinational functions	58 / 22,320 (< 1 %)
Dedicated logic registers	17 / 22,320 (< 1 %)
Total registers	17
Total pins	22 / 154 (14 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 39: Resultados de la síntesis del generador de baudios

#### 10.1.5 Simulación RTL

En la simulación de la figura 40 podemos observar cómo cambia la salida BRGCLOCK en función del valor que tenga la entrada SPBRGVALUE. También podemos apreciar como el contador SPBRG cuanta una serie de ciclos de reloj hasta que cambia la salida BRGCLOCK.

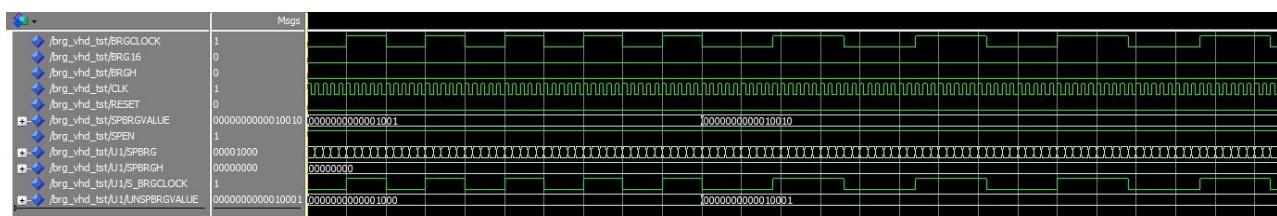


Figura 40: Simulación RTL del generador de baudios

### 10.1.6 Simulación lógica temporal a nivel de puertas

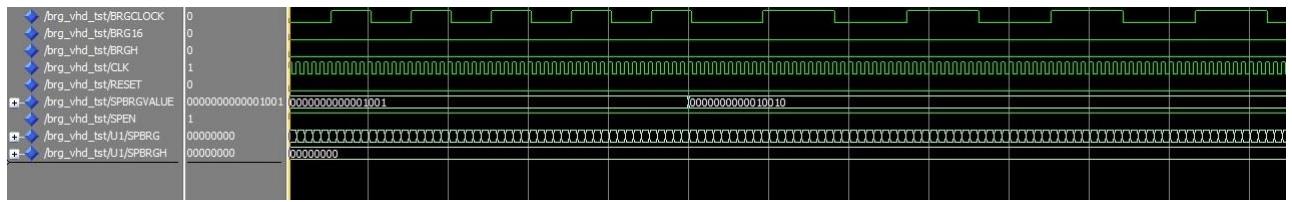


Figura 41: Simulación a nivel de puertas del generador de baudios

En la simulación de la figura 41 se puede apreciar los retrasos en la señal de salida, producidos por los componentes de la FPGA.

## 10.2 TX\_TRANSFER

Este bloque se encarga de realizar la transmisión de información. Se encarga de generar un bit de START que comience la transmisión. Después genera los bits de datos y finaliza con el bit de STOP. En este caso nuestro transmisor cuenta con la posibilidad de enviar ocho o nueve bits. Además se pueden configurar, junto al BRG, una serie de velocidades de transmisión para reducir el error en las transmisiones.

A continuación se muestra un resumen detallado del bloque TX\_TRANSFER.

### 10.2.1 Entidad

En el esquema de la figura 42 se aprecian los puertos que componen la entidad del bloque TX\_TRANSFER. Es preciso destacar el puerto TX\_PIN, a través del cual se realiza la transmisión de datos en formato serie.

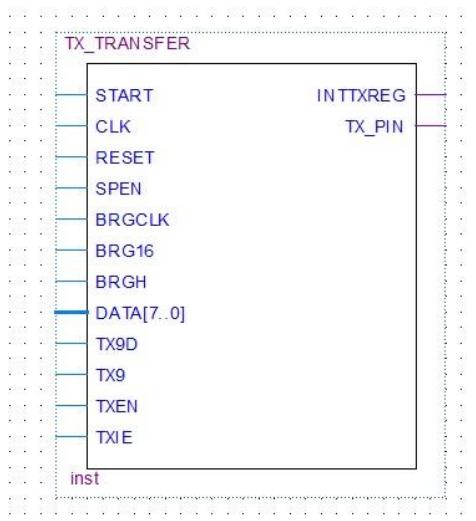


Figura 42: Bloque que contiene las entradas y salidas del transmisor

### 10.2.2 Funcionamiento del transmisor TX\_TRANSFER

El dispositivo transmisor cuenta con dos registros que almacenan datos de la transmisión. *El* primero es el registro TXREG\_REGISTER que actúa como un buffer. Almacena la palabra a enviar hasta que esta es solicitada por el registro TSR\_REGISTER. Este registro se encarga de transmitir los bits uno a uno a través del pin TX. Para coordinar el buen funcionamiento de estos dos registros se emplea las señales TXIF y TMRT. La señal TXIF indica cuando el registro TXREG\_REGISTER está vacío y por tanto puede cargar una palabra. La señal TMRT muestra el momento en el que el registro TSR\_REGISTER está vacío. Cuando esto sucede el registro carga la palabra proveniente del registro TXREG\_REGISTER. Cabe destacar que el registro



TXREG\_REGISTER recibe un valor cuando la UART recibe una señal de START. En caso de que el registro TXREG\_REGISTER esté lleno no se tendrá en cuenta esta señal.

En función de las entradas BRG16 y BRGH el transmisor transmitirá un bit cada 4, 16 o 64 ciclos del reloj BRGCLK.

Activando la entrada TX9 el dispositivo es capaz de transmitir 9 bits de datos.

El dispositivo genera una interrupción cuando el registro que funciona como buffer está vacío.

### 10.2.3 Puertos y señales

Los diferentes puertos y señales de la entidad TX\_TRANSFER se resumen en las tablas 10 y 11.

Puertos	E/S	Tipo	Descripción
START	entrada	std_logic	Inicia la transmisión de información
CLK	entrada	std_logic	Reloj
RESET	entrada	std_logic	Resetea la transmisión
SPEN	entrada	std_logic	Habilita la transmisión serie
BRGCLK	entrada	std_logic	Reloj generado por el BRG necesario en el registro de desplazamiento (TSR_Register)
BRG16	entrada	std_logic	Este bit determina el número de ciclos que tienen que pasar en BRGCLK para enviar un dato. Determina el valor de COUNTMAX
BRGH	entrada	std_logic	Determina la velocidad con la que el registro de desplazamiento envía datos. Determina el valor de COUNTMAX
DATA	entrada	std_logic_vector(7 downto 0)	Ocho bits que se transmiten
TX9D	entrada	std_logic	Noveno dato
TX9	entrada	std_logic	Habilita el noveno dato a transmitir
TXEN	entrada	std_logic	Habilita el reloj del BRG
TXIE	entrada	std_logic	Habilita la interrupción INTTXREG
INTTXREG	salida	std_logic	Interrupción generada cuando el registro inicial (TXREG_REGISTER) está vacío
TX_PIN	salida	std_logic	Salida TX del puerto serie

Tabla 10: Puertos que forman parte de la entidad del bloque TX\_TRANSFER



Señales		Tipo	Descripción
TSR_REGISTER		unsigned (10 downto 0)	Este es el registro con desplazamiento. Engloba los bits de START y STOP. Además engloba el noveno dato
TXREG_REGISTER		unsigned (7 downto 0)	Este registro carga los ocho datos provenientes del exterior. Después los carga en TSR_REGISTER
S_BRGCLK		std_logic	Es igual a (TXEN and BRGCLK)
TMRT		std_logic	Se pone a uno cuando el registro de desplazamiento está vacío
TXIF		std_logic	Se pone a uno cuando el TXREG_REGISTER está vacío
COUNT		unsigned (5 downto 0)	Contador que cuenta flacos del reloj BRGCLK hasta alcanzar el valor COUNTMAX. En ese momento se envía un bit.
COUNTMAX		unsigned (5 downto 0)	Valor máximo que alcanza el registro COUNT en función del valor de las entradas BRG16 y BRGH
INDEX		integer range 0 to 10	Contador que indica el bit a enviar
INDEXMAX		integer range 0 to 10	Valor máximo que alcanza el registro INDEX en función de si se envían ocho bits o nueve bits
STOP		std_logic	Señal que determina el final de una transmisión. Se corresponde con el bit de STOP
S1		std_logic	La señal S1 es necesaria para el generador de impulsos. Este generador convierte la señal STOP en un impulso de un solo ciclo de reloj
S2		std_logic	La señal S2 es necesaria para el generador de impulsos. Este generador convierte la señal STOP en un impulso de un solo ciclo de reloj
IMP_STOP		std_logic	Esta señal manda un impulso cada vez que STOP se pone a uno.(tiene un retardo de un ciclo de reloj con respecto a STOP)
U1		std_logic	La señal U1 es necesaria para tener únicamente un reloj. Convierten la señal BRGCLK en un impulso de ancho igual a CLK (IMP_BRGCLK)
U2		std_logic	La señal U2 es necesaria para tener únicamente un reloj. Convierten la señal BRGCLK en un impulso de ancho igual a CLK (IMP_BRGCLK)
IMP_BRGCLK		std_logic	Esta señal es un impulso de ancho igual al periodo del reloj CLK. Se obtiene cada vez que la señal BRGCLK presenta un flanco de reloj positivo

Tabla 11: Señales del bloque TX\_TRANSFER

#### 10.2.4 Resultados de la síntesis

Como es de esperar el empleo de elementos lógicos es inferior a un 1 %, al igual que la UART completa. El número de elementos lógicos empleados por el Bloque TX\_TRANSFER se puede apreciar en la figura 43.

Flow Summary	
Flow Status	Successful - Mon Jul 27 13:25:45 2015
Quartus II 64-Bit Version	14.0.0 Build 200 06/17/2014 SJ Web Edition
Revision Name	TX_TRANSFER
Top-level Entity Name	TX_TRANSFER
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	51 / 22,320 (< 1 %)
Total combinational functions	36 / 22,320 (< 1 %)
Dedicated logic registers	36 / 22,320 (< 1 %)
Total registers	36
Total pins	21 / 154 (14 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 43: Resultados de la síntesis del dispositivo TX\_TRANSFER

#### 10.2.5 Transmisión de dos caracteres: Simulación RTL

En la simulación de la figura 44 podemos apreciar la transmisión de dos palabras de nueve bits cada una. También se puede apreciar el funcionamiento de las señales TMRT y TXIF que se ponen a uno cuando sus respectivos registros están vacíos. También podemos comprobar el buen funcionamiento de la señal STOP, que se pone a uno para determinar el final de la transmisión de una palabra.

Además podemos comprobar que cada bit transmitido tiene la misma duración.

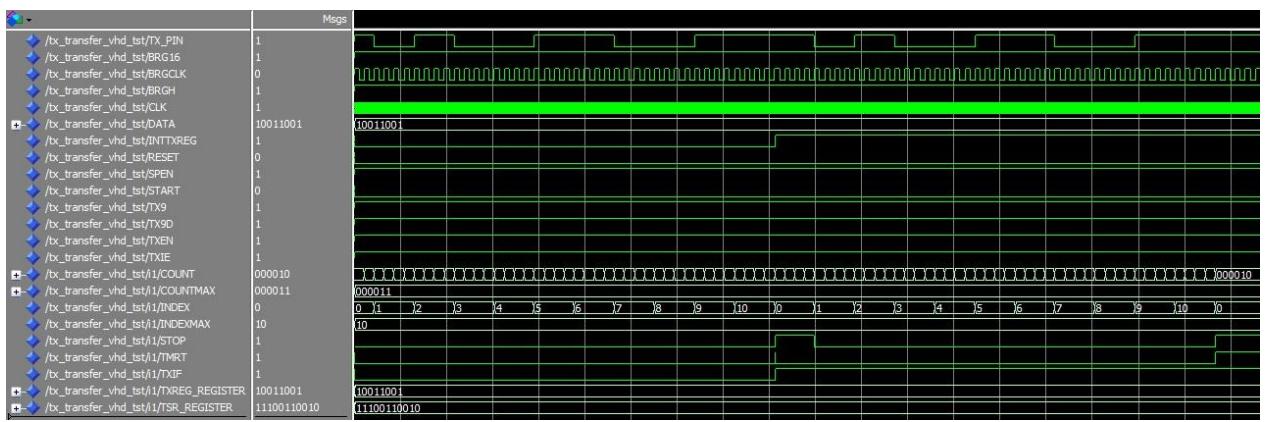


Figura 44: Transmisión de dos palabras de nueve bits

En la simulación de la figura 45 podemos observar la entrada de dos bits de START. Podemos apreciar como la señal TXIF pasa de uno a cero al cargarse datos en el registro TXREG\_REGISTER. Además se observa como la señal TMRT pasa de uno a cero cuando se carga en el registro TSR\_REGISTER los datos provenientes del registro TXREG\_REGISTER; con el consiguiente cambio de la señal TXIF de cero a uno al vaciarse el registro TXREG\_REGISTER.

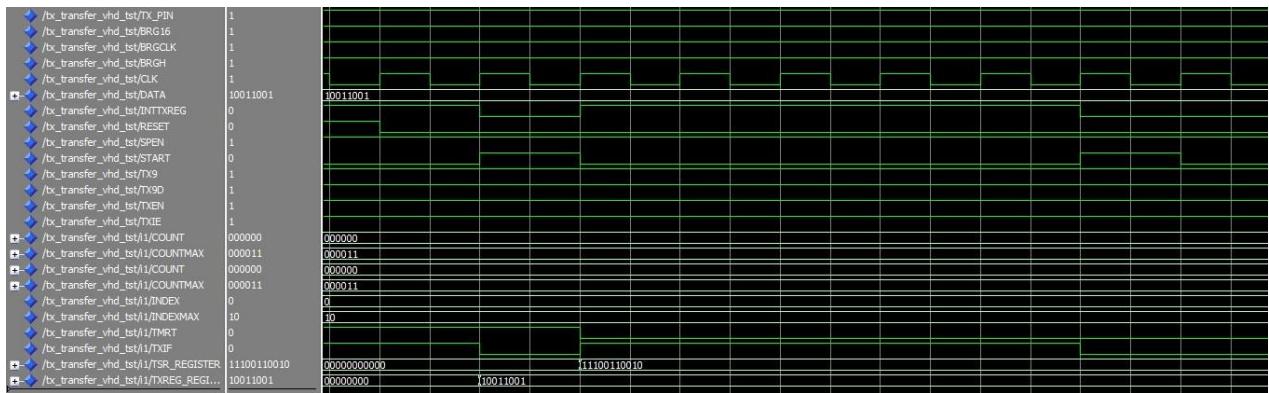


Figura 45: Inicio de la transmisión.

### 10.2.6 Simulación lógica temporal a nivel de puertas

En la simulación de la figura 46 se puede apreciar que la salida del pin TX es equivalente a la salida de la simulación RTL. Además la interrupción generada, en ambas simulaciones se produce simultáneamente.

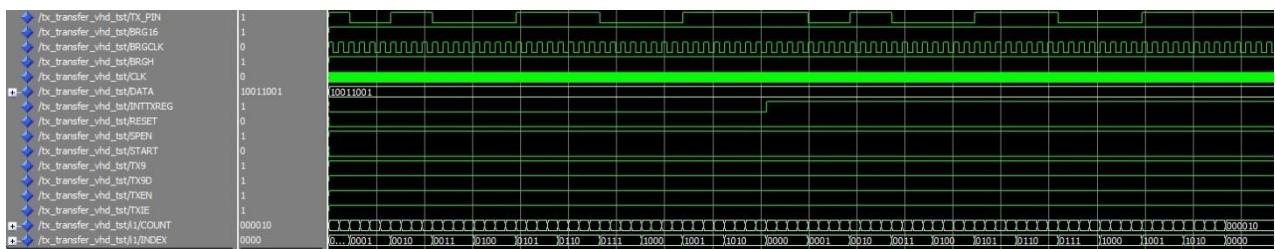


Figura 46: Simulación con retrasos

## 10.3 RX\_RECEIVER

El receptor se encarga de recibir los bits provenientes de otro dispositivo UART. El puerto RX permanece a uno hasta que llega el bit de START. A partir de ese suceso el dispositivo empieza a almacenar cada bit hasta recibir el bit final o de STOP. Finalmente el dispositivo guarda los bits de datos en un registro, para que estos puedan ser leídos.

En los siguientes apartados se proporciona una descripción detallada del bloque receptor.

### 10.3.1 Entidad

En la figura 47 podemos apreciar el conjunto de puertos que componen la entidad del bloque RX\_RECEIVER. Dentro de estos puertos destaca el puerto RX\_PIN donde se produce la recepción de los datos en formato serie. También destaca el puerto DATA donde se obtienen los datos en formato paralelo cuando el receptor termina la conversión.

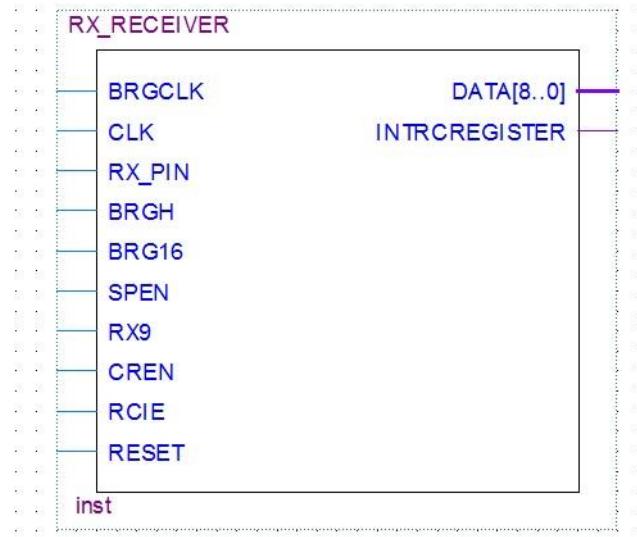


Figura 47: Entidad del receptor



### 10.3.2 Funcionamiento del receptor

En el caso del receptor, la recepción comienza cuando aparece el bit de START en el puerto RX. Al igual que en los dos dispositivos anteriores, cabe tener en cuenta que la señal BRGCLK tiene una frecuencia 4, 16 o 64 veces superior a la frecuencia de recepción de bits. De tal forma debemos contar el número de flancos de reloj de la señal BRGCLK para muestrear los bits recibidos en el medio de cada bit. Para esto se empleara el registro COUNT que contará los flancos de reloj de la señal anterior y muestreará cuando se alcance la mitad del valor máximo (4,16 o 64) almacenado en COUNTMAX.

Por otro lado el registro INDEX indica en cada momento el bit que se muestrea. El registro INDEXMAX indica el máximo número de bits a muestrear y sean 8 o 9, en función de la entrada RX9.

Cabe destacar que el receptor cuenta con una interrupción que se activa cada vez que se concluye una transmisión.

La recepción concluye cuando se recibe el bit de STOP. En ese momento se activa la interrupción y se almacena la palabra recibida en el registro RC\_REGISTER.

### 10.3.3 Puertos y señales

Las tablas 12 y 13 muestran una descripción de los puertos y señales del bloque RX\_RECEIVER.

Puerto	E/S	Tipo	Descripción
CLK	entrada	std_logic	Reloj
BRGCLK	entrada	std_logic	Reloj procedente del BRG. Es necesario para muestrear los bits en el pin RX
RX_PIN	entrada	std_logic	Pin donde se recibe cada bit en formato serie
BRGH	entrada	std_logic	Junto con la señal BRG16 nos permite seleccionar cuantos ciclos del reloj BRGCLOCK tienen que pasar para muestrear un bit del pin RX
BRG16	entrada	std_logic	Junto con la señal BRGH nos permite seleccionar cuantos ciclos del reloj BRGCLOCK tienen que pasar para muestrear un bit del pin RX
SPEN	entrada	std_logic	Habilita la comunicación serie
RX9	entrada	std_logic	Habilita la recepción de un noveno bit
CREN	entrada	std_logic	Habilita el funcionamiento del receptor
RCIE	entrada	std_logic	Habilita la interrupción INTRCREGISTER
RESET	entrada	std_logic	Reset del dispositivo
DATA	salida	std_logic_vector(8 downto 0)	Salida de los ocho o nueve bits recibidos por el puerto RX
INTRCREGISTER	salida	std_logic	Interrupción que señala la recepción completa de ocho o nueve bits

Tabla 12: Puertos que forman parte de la entidad del bloque RX\_RECEIVER



Señales	Tipo	Descripción
RSR_REGISTER	unsigned (10 downto 0)	Registro con desplazamiento. Almacena los bits que van llegando del puerto RX
RC_REGISTER	unsigned (7 downto 0)	Registro final. Almacena los bits recibidos cuando se completa la recepción de una palabra
S_RX9D	std_logic	Registro que almacena el noveno bit
RX_FLAG	std_logic	Determina si el registro de desplazamiento está funcionando. Si 1 funciona, si no, no.
COUNT	unsigned (5 downto 0)	Contador que cuenta cada flanko del reloj BRGCLK. Es una señal imprescindible para preescalar este reloj y así muestrear los bits recibidos en el medio.
COUNTMAX	unsigned (5 downto 0)	Este registro almacena el máximo valor de la cuenta COUNT. Alcanza un valor de 4, 16 o 64 en función de las entradas BRGH y BRG16
INDEX	integer range 0 to 10	Contador que determina el bit a leer
INDEXMAX	integer range 0 to 10	Cuenta máxima que alcanza INDEX en función de si se reciben 8 o 9 bits. Viene determinado por la entrada RX9
STOP	std_logic	Señal que se activa cuando se alcanza el bit de STOP en la transmisión
S1	std_logic	Tanto S1 como S2 son señales necesarias para el generador de impulsos. Este generador convierte la señal STOP en un impulso de un ciclo de reloj
S2	std_logic	Tanto S1 como S2 son señales necesarias para el generador de impulsos. Este generador convierte la señal STOP en un impulso de un ciclo de reloj
IMP_STOP	std_logic	Esta señal manda un impulso cada vez que STOP se pone a uno.(tiene un retardo de un ciclo de reloj con respecto a STOP)
RCIF	std_logic	Este registro indica si el registro RC_REGISTER está lleno o no.
T1	std_logic	Tanto T1 como T2 son señales necesarias para tener únicamente un reloj. Convierten la señal BRGCLK en un impulso de ancho igual a CLK
T2	std_logic	Tanto T1 como T2 son señales necesarias para tener únicamente un reloj. Convierten la señal BRGCLK en un impulso de ancho igual a CLK
IMP_BRGCLK	std_logic	Esta señal nos permite tener un único reloj en el dispositivo (CLK) ya que convierte el reloj BRGCLK en un enable que funciona con el reloj CLK

Tabla 13: Señales del bloque RX\_RECEIVER

#### 10.3.4 Resultados de la síntesis

Como se puede apreciar en la figura 48 el empleo de elementos lógicos es escaso. Esto lógico dado que el consumo de elementos de la UART completa no supera el 1 %.

Flow Summary	
Flow Status	Successful - Mon Jul 27 13:29:54 2015
Quartus II 64-Bit Version	14.0.0 Build 200 06/17/2014 SJ Web Edition
Revision Name	RX_RECEIVER
Top-level Entity Name	RX_RECEIVER
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	54 / 22,320 ( < 1 % )
Total combinational functions	44 / 22,320 ( < 1 % )
Dedicated logic registers	35 / 22,320 ( < 1 % )
Total registers	35
Total pins	20 / 154 ( 13 % )
Total virtual pins	0
Total memory bits	0 / 608,256 ( 0 % )
Embedded Multiplier 9-bit elements	0 / 132 ( 0 % )
Total PLLs	0 / 4 ( 0 % )

Figura 48: Resultados de la síntesis para el dispositivo RX\_RECEIVER

#### 10.3.5 Recepción de dos caracteres: Simulación RTL

En la simulación de la figura 49 se puede apreciar la recepción de dos caracteres de forma seguida. En este caso cada carácter contiene ocho bits, ya que el bit RX9 tiene valor lógico cero. Se puede apreciar como la señal RCIF se pone a uno cuando se ha completado la recepción de un carácter. También se observa como la señal RX\_FLAG se pone a uno cuando el registro RSR\_REGISTER está ocupado.

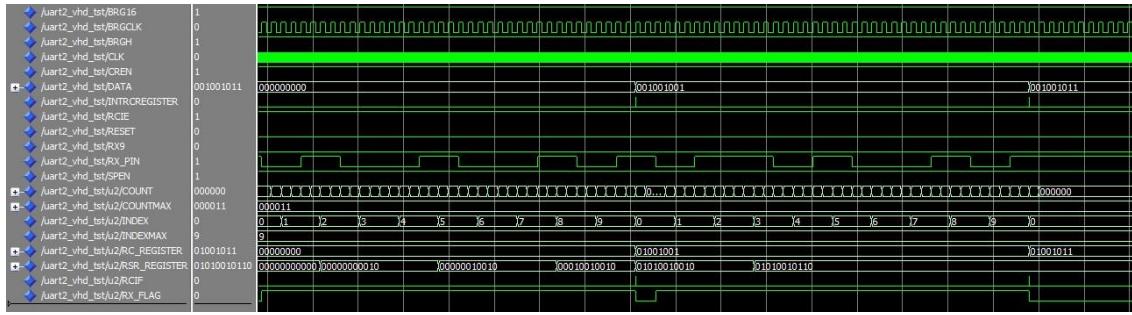


Figura 49: Simulación del receptor

Por otro lado también podemos comprobar cómo se producen únicamente cuatro incrementos en el contador COUNT por cada bit leído. Además podemos apreciar en el bit de STOP, como se produce el muestreo en el medio.

### 10.3.6 Simulación lógica temporal a nivel de puertas

En la figura 50 podemos comprobar como los dos caracteres consecutivos se reciben correctamente. Cabe destacar la interrupción que marca la recepción completa de cada palabra.

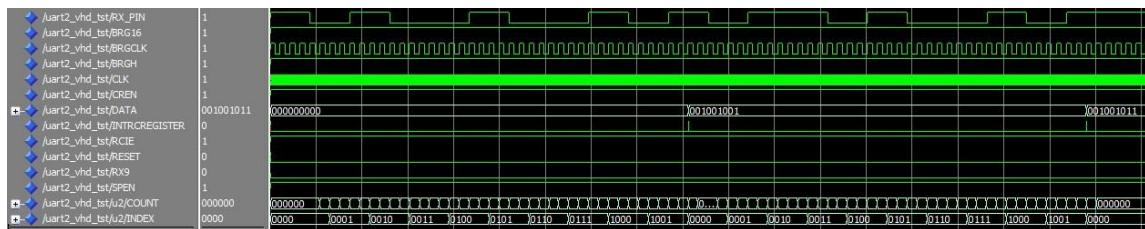


Figura 50: Simulación con retrasos de los componentes

Destacar que la interrupción se produce justo en la mitad del bit de STOP, lo que señala que se ha muestreado este bit justo en el medio.

## 10.4 TECLADO

Para probar el buen funcionamiento de la UART vamos a emplear un teclado matricial para enviar información al transmisor y que este la envíe. El teclado tendrá dos funciones:

- Según la tecla pulsada se enviará un carácter distinto
- Iniciará la comunicación enviando un impulso de START al transmisor

En la figura 51 encontramos un tipo de teclado matricial.

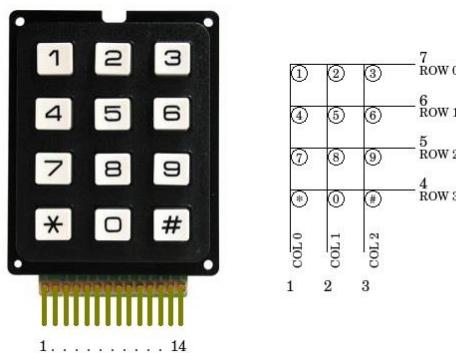


Figura 51: Teclado matricial

### 10.4.1 Entidad

En la figura 52 se muestra en un esquema la entidad del teclado. Cabe destacar los puertos ENTRADAS y SALIDAS necesarios para el correcto muestreo del teclado. También destacar el puerto IMPULSO, que marca cuando se inicia la transmisión de datos en la UART.



Figura 52: Bloque correspondiente al teclado

## 10.4.2 Funcionamiento del teclado

Hay diferentes métodos para muestrear un teclado matricial. Hemos decidido implementar el siguiente método.

Nuestro teclado matricial presenta cuatro filas y cuatro columnas conectadas como en la figura 53.

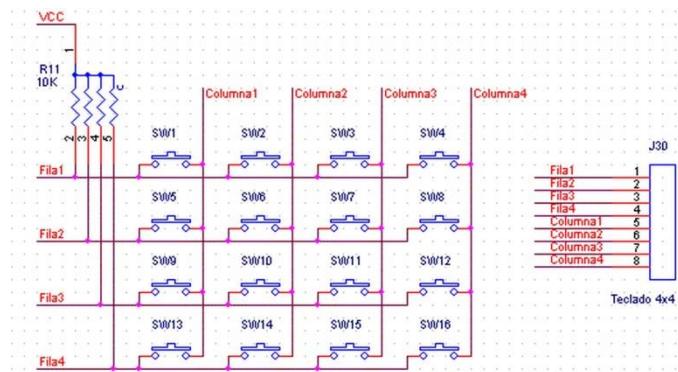


Figura 53: Esquema de conexión del teclado matricial

Fuente: [http://arantxa.ii.uam.es/~gdrivera/sed/sist\\_des/gdr\\_arm.htm](http://arantxa.ii.uam.es/~gdrivera/sed/sist_des/gdr_arm.htm)

En este caso tendremos cuatro filas que se corresponderán con puertos de entrada de la FPGA y cuatro columnas que se corresponden con puertos de salida. Cabe remarcar el empleo de resistencias de *pull-up* para evitar rebotes al pulsar el dispositivo. Estas resistencias vienen incorporadas en la FPGA.

Par saber que tecla se pulsa en un momento dado, se inducen ciertas salidas en las columnas del teclado. Cada salida se repite con una frecuencia de 100 Hz. En cada salida, se induce un cero en una de las columnas, de tal manera que si alguien pulsa un botón de esa columna, en la fila correspondiente aparecerá un cero y se podrá identificar la tecla en función de la entrada.

Para implementar lo anterior en VHDL empleamos la variable PRSCL como preescalador del reloj. De esta manera obtenemos una señal que nos permite inducir un cero en cada columna con una frecuencia de 400 Hz, obteniendo una frecuencia de muestreo del teclado completo de 100 Hz.

Por otro lado empleamos una serie de decodificadores para obtener el código ASCII de las teclas del teclado, en función de las salidas inducidas y las entradas recibidas.

Para asignar el carácter en código ASCII a la salida del bloque TECLADO y para enviar el impulso que inicie la transmisión de este carácter se emplea una máquina de estados. Esta máquina cuenta con tres estados como se muestra a en la figura 54.



Figura 54: Diagrama de estados del teclado

El estado S1 es un a estado de reposo. En este estado solo se pone el puerto de salida, IMPULSO, a cero. Se pasa al estado S2 si se produce la pulsación de un botón. Esto se sabe dado que alguno de los cuatro bits menos significativos de la señal CODIGO\_ASCII se pone a cero. En el estado S2 se asigna la señal CODIGO\_ASCII al puerto de salida TDATA. Este será el puerto de salida que lea el transmisor. Se pasa al estado S3 cuando pase un ciclo de reloj. En el estado S3 se pone a uno la señal impulso. De esta forma el transmisor recibe una señal diciendo que se ha pulsado el botón. Finalmente pasamos del estado S3 al S1 cuando pase un decisegundo. Se establece este tiempo para evitar que la pulsación del botón produzca un envío repetitivo del mismo carácter.

### 10.4.3 Puertos y señales

Las tablas 14 y 15 muestran los puertos y señales del bloque TECLADO.

Puertos	E/S	Tipo	Descripción
ENTRADAS	entrada	std_logic_vector(3 downto 0)	Cuatro bits de entrada que se conectan a las filas del teclado. Determinan la tecla que se ha pulsado en función de la salida.
SALIDAS	salida	out std_logic_vector(3 downto 0)	Cuatro bits de salida que se conectan a las columnas del teclado. Determinan la tecla que se ha pulsado en función de la entrada.
TDATA	salida	out std_logic_vector(7 downto 0)	Salida donde se puede leer el código ASCII correspondiente a la tecla pulsada
CLK	entrada	std_logic	Reloj
RESET	entrada	std_logic	Reset del circuito
IMPULSO		std_logic	Manda un pulso de un decisecondo cuando pulsamos una tecla. Está controlado por una máquina de estados para enviarlo un ciclo de reloj después de la asignación del código ASCII correspondiente

Tabla 14: Puertos que forman parte de la entidad del bloque TECLADO



Señales		Tipo	Descripción
CUENTA		unsigned (1 downto 0)	Contador que cuenta de 0 a 3. Este contador se incrementa cuando la variable PRSCL alcanza el valor correspondiente. Cada vez que se incrementa el valor del contador, se cambia el puerto SALIDAS
S_SALIDAS		std_logic_vector(3 downto 0)	Esta señal cambia cuando se incrementa el contador CUENTA. Esta señal cambia para poder muestrear cada columna del teclado. Esta señal se asigna al puerto SALIDAS
CODIGO_ASCII		unsigned(7 downto 0)	Esta señal adjudica el código ASCII correspondiente en función del valor de los puertos ENTRADAS Y SALIDAS.
STATE		STATE_TYPE	A esta señal se le pueden asignar los estados S1, S2 y S3.
PRSCL		unsigned(16 downto 0)	Preescala el reloj CLK haciendo posible que se muestre una pulsación del teclado a 100 Hz
DECISEGUNDO		std_logic	Señal que indica que ha pasado un decisegundo desde que se alcanzó el estado S2. Hace posible enviar un único carácter cada vez que se pulsa una tecla.
TIEMPO		unsigned (22 downto 0)	Contador que cuenta ciclos de reloj hasta que pasa un decisegundo.

Tabla 15: Señales del bloque TECLADO

#### 10.4.4 Resultados de la síntesis

Al igual que en el resto de apartados la utilización de elementos lógicos de la FPGA es escaso.

Flow Summary	
Flow Status	Successful - Mon Jul 27 13:19:16 2015
Quartus II 64-Bit Version	14.0.0 Build 200 06/17/2014 SJ Web Edition
Revision Name	TECLADO
Top-level Entity Name	TECLADO
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	100 / 22,320 (< 1 %)
Total combinational functions	100 / 22,320 (< 1 %)
Dedicated logic registers	45 / 22,320 (< 1 %)
Total registers	45
Total pins	19 / 154 ( 12 % )
Total virtual pins	0
Total memory bits	0 / 608,256 ( 0 % )
Embedded Multiplier 9-bit elements	0 / 132 ( 0 % )
Total PLLs	0 / 4 ( 0 % )

Figura 55: Resultados de la síntesis para el TECLADO

#### 10.4.5 Pulsación del teclado: Simulación RTL

En la simulación mostrada en la figura 56, podemos apreciar como al pulsar una de las teclas se envía un carácter al transmisor. Forzamos la pulsación de una tecla poniendo los cuatro bits “1011” en el bus de entrada. Al forzar esta circunstancia se le asigna a la señal CODIGO\_ASCII el código ASCII correspondiente a la entrada y la salida. Posteriormente al puerto de salida TDATA se le asigna el valor de la señal CODIGO\_ASCII. Despues de esto se alcanza el estado S2 en el que el puerto de salida IMPULSO se pone a uno. Esto da lugar a que se inicie la transmisión del carácter pulsado.

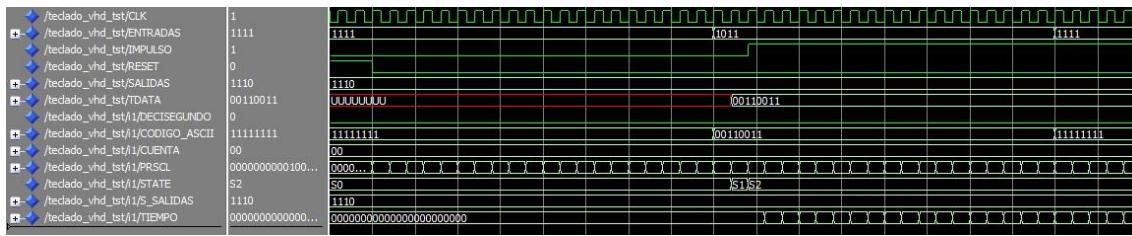


Figura 56: Simulación que muestra la pulsación de una tecla

En esta simulación de la figura 57 podemos apreciar cómo se pasa del estado S2 al S0. Esto se produce cuando se pone a uno la señal DECISEGUNDO. Esta señal depende de un contador

que se inicia cuando se alcanza el estado S2. Al llegar a una cuenta que equivale a un tiempo de un decisecondo, se cambia de estado y se pone a cero la salida IMPULSO.

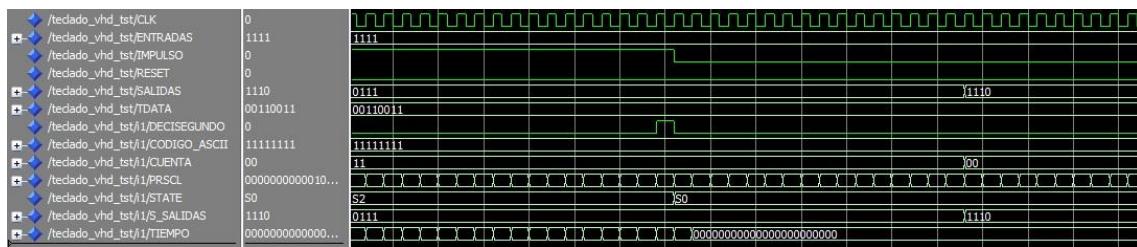


Figura 57: Simulación que muestra la señal DECISEGUNDO activarse

#### 10.4.6 Simulación lógica temporal a nivel de puertas

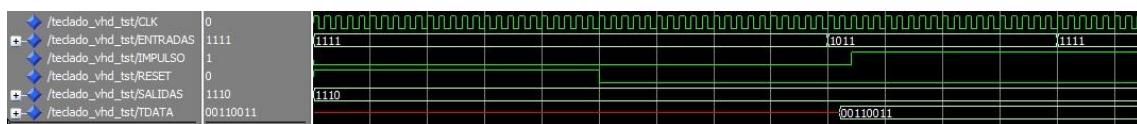


Figura 58: Simulación temporal de la pulsación de una tecla

En esta simulación se muestra como al pulsar el botón (al forzar la entrada a “1011”) la variable TDATA adquiere el valor correspondiente al código ASCII. Posteriormente la señal IMPULSO se pone a uno para iniciar la transmisión.

## 10.5 SPBRGVALUE

El boque SPBRGVALUE se emplea para cargar un valor en la entrada SPBRGVALUE del generador de baudios. De esta forma el generador de baudios genera un reloj con la frecuencia adecuada.

En los siguientes apartados se muestra una descripción detallada del bloque SPBRGVALUE

### 10.5.1 Entidad

En la figura 59 se muestra un esquema de la entidad del bloque SPBRGVALUE. El único puerto del bloque carga el valor necesario, al generador de baudios, para que funcione a una determinada velocidad de transmisión.



Figura 59: Bloque SPBRGVALUE

### 10.5.2 Funcionamiento del bloque SPBRGVALUE

Este dispositivo se encarga únicamente de asignar al dispositivo BRG el valor correcto para poder obtener un reloj con la frecuencia adecuada. Por tanto en función del *baud rate* que necesitemos haremos que el dispositivo cargue en la entrada SPBRGVALUE un valor u otro.

Podemos encontrar en el Anexo A los valores correctos a asignar en función de la configuración del dispositivo BRG. El valor a introducir se denomina “n” en estas tablas.

Dentro de este bloque empleamos casting para convertir una señal *integer* (DEC\_SPBRGVALUE) donde introducimos el valor “n”, en la salida SPBRGVALUE de tipo std\_logic. De esta manera convertimos el número decimal que designa el *baud rate*, en la salida necesaria para que el dispositivo BRG genere el reloj adecuado.

### 10.5.3 Puertos y señales

Las tablas 16 y 17 muestran los puertos y señales del bloque SPBRGVALUE.

Puertos	E/S	Tipo	Descripción
SPBRGVALUE	salida	std_logic_vector(15 downto 0)	Salida conectada a la entrada SPBRGVALUE del dispositivo BRG. Carga un valor para obtener el <i>baud rate</i> adecuado.

Tabla 16: Puertos que forman parte de la entidad del bloque SPBRGVALUE

Señales		Tipo	Descripción
DEC_SPBRGVALUE		integer range 0 to 65535	Valor cargado en la salida SPBRGVALUE. Introducimos el valor "n" de las tablas del Anexo A.

Tabla 17: Señales del bloque TECLADO

### 10.5.4 Resultados de la síntesis

Como podemos observar en la figura 60 el empleo de elementos lógicos es igual a cero.

Flow Summary	
Flow Status	Successful - Mon Jul 27 13:44:26 2015
Quartus II 64-Bit Version	14.0.0 Build 200 06/17/2014 SJ Web Edition
Revision Name	SPBRGVALUE
Top-level Entity Name	SPBRGVALUE
Family	Cyclone IV E
Device	EP4CE22F17C7
Timing Models	Final
Total logic elements	0 / 22,320 ( 0 % )
Total combinational functions	0 / 22,320 ( 0 % )
Dedicated logic registers	0 / 22,320 ( 0 % )
Total registers	0
Total pins	16 / 154 ( 10 % )
Total virtual pins	0
Total memory bits	0 / 608,256 ( 0 % )
Embedded Multiplier 9-bit elements	0 / 132 ( 0 % )
Total PLLs	0 / 4 ( 0 % )

Figura 60: Resultados de la síntesis para el dispositivo SPBRGVALUE



## 11 Presupuesto

En este presupuesto se hace una estimación de los gastos necesarios para la realización de este proyecto. Vamos a tener en cuenta los siguientes gastos:

- Gastos de hardware

Costes de placas de desarrollo, módulos acoplados, ordenador, cableado, etc.

- Gasto de software

Gastos de plataformas de desarrollo y simulación, coste de sistema operativo.

- Costes de personal

Recursos humanos necesarios para la realización del proyecto.

### 11.1 Gastos de hardware

En este apartado vamos a destacar los gastos relativos a los dispositivos físicos que hemos empleado para realizar el proyecto.

Entre estos dispositivos destacan el ordenador que hemos empleado. También hay que destacar la placa de desarrollo DE0 Nano de Altera.

Por otro lado también hemos empleado una serie de módulos adicionales como puede ser el convertidor USB-UART que hemos empleado.

Concepto	Precio
Ordenador portátil. Netbook	250 €
Placa de desarrollo DE0 Nano de Altera	90 €
Convertidor USB UART	5 €
Total	345 €

*Tabla 18: Gastos de hardware*

## 11.2 Gastos de software

Dentro de los gastos de software debemos incluir la licencia de Windows 7 y las licencias de los programas de desarrollo necesarios para describir hardware en la placa. Dentro de los programas de desarrollo debemos incluir Quartus II Edición Web y el software de simulación que esta misma plataforma incorpora, Altera-Modelsim. En este caso tanto la versión web de Quartus II como Modelsim son gratuitos para la familia de FPGAs Cyclone, por lo tanto tienen coste cero.

Por otro lado hemos empleado diversas herramientas de Microsoft Office para la correcta elaboración del proyecto.

Concepto	Precio
Microsoft Windows 7 Professional	146,95 €
Quartus II Edición Web	0 €
Altera-Modelsim	0 €
Microsoft Office 2010 Professional	393,07 €
Total	540 €

Tabla 19: Gastos de software

## 11.3 Costes de personal

En el coste de mano de obra debemos tener en cuenta la necesidad de un ingeniero junior para el desarrollo del proyecto y un ingeniero senior para resolver posibles inconvenientes que surjan a lo largo del desarrollo.

Concepto	Precio/hora	Horas empleadas	Precio total
Ingeniero junior	9,48 €	600	5.688,00 €
Ingeniero senior	11,59 €	200	2.318,00 €
Total			8.006,00 €

Tabla 20: Costes de personal

## 11.4 Costes totales

Costes del proyecto	
Costes de hardware	345 €
Costes de software	540 €
Costes de personal	8.006 €
Costes totales	8.891 €

*Tabla 21: Costes totales*

Cabe remarcar que los precios de los productos adquiridos incluyen el impuesto del valor añadido (IVA).

Los sueldos de los ingenieros son brutos por lo que incluyen impuestos. Se ha tomado como base de los sueldos, un estudio de la página Tusalario.es.



## 13 Entorno socioeconómico

El dispositivo UART que hemos diseñado en VHDL se puede clasificar como un IP Core o núcleo de propiedad intelectual. Esto se debe a que nuestro diseño esta implementado en un lenguaje de descripción de hardware, por lo que no es algo tangible. En diseño electrónico un IP Core es una unidad de lógica un dispositivo *layout* o una celda reusable que es propiedad intelectual de una parte. Un IP Core puede ser propiedad de una único propietario o puede ser otorgada una licencia a otros propietarios. El termino de IP Core deriva de la obtención de la patente o los derechos de *copyrjght* del diseño. Se pueden emplear como bloques dentro de diseños de ASICs o de lógica en FPGAs.

Actualmente existen una serie de páginas web que se encargan de comercializar este tipo de dispositivos y entre ellos la UART. Un ejemplo de ello lo encontramos en la página “Logicbricks” (<http://www.logicbricks.com/>). En esta página se comercializan licencias de UART de duración limitada. La licencia durante un año tiene un coste de 500 €, la licencia por dos años 900 € y la licencia por tres años 1200 €. Además también podemos encontrar páginas web con diseños de lógica digital complemento gratuitos. Es el caso de la web “Opencores” (<http://opencores.com/>) que ofrece alternativas de hardware libre.

Con estos datos obtenemos como conclusión que existe un mercado de lógica digital donde el dispositivo diseñado durante el proyecto podría ser comercializado obteniendo un beneficio real.

Desde otra perspectiva cabe destacar que la UART es avalada por una gran cantidad de estándares. Entre ellos destaca el estándar RS-232, RS-422 y RS-485. Cada uno de estos estándares son descritos detalladamente en el capítulo 6 del proyecto. Por tanto estamos ante un dispositivo altamente estandarizado.



## 14 Objetivos alcanzados

Al inicio del proyecto nos marcamos una serie de metas a alcanzar. Entre estas metas destaca el diseño e implementación de una UART en VHDL y el desarrollo de un ejemplo práctico que demuestre el buen funcionamiento de la UART.

Como se ha podido demostrar en apartados anteriores tanto por el estudio de la UART como por el desarrollo en VHDL, se puede comprobar que el diseño e implementación se ha conseguido con éxito.

Hemos conseguido una implementación correcta de la UART gracias al establecimiento de diferentes hitos que se han desarrollado de manera secuencial. Los hitos han sido los siguientes:

- Se ha realizado un pequeño estudio de la comunicación serie, así como de los diferentes tipos de comunicación.
- Hemos realizado un estudio exhaustivo del funcionamiento de la comunicación serie asíncrona.
- Hemos analizado el funcionamiento de diversos dispositivos ya existentes. Entre ellos destacar la UART del microcontrolador que empleamos como base para el proyecto.
- Se ha procedido a la división de la UART en diferentes bloques para su posterior implementación.
- Hemos implementado cada bloque empleando los conocimientos sobre VHDL adquiridos durante el grado. Además ha sido necesario profundizar en estos conocimientos.
- Para finalizar ha sido necesario utilizar la herramienta Quartus II de Altera para implementar nuestro desarrollo en la FPGA. Cabe destacar que esto sirve de ayuda para conocer el entorno de desarrollo de Altera y así complementar el conocimiento ya adquirido en el grado utilizando el ISE de Xilinx.

Por otro lado cabe destacar que con la incorporación del teclado matricial y el empleo de los LEDs de la placa hemos conseguido crear un ejemplo práctico del funcionamiento de la UART. Además se han incorporado dos dispositivos que convierten la comunicación serial en *bluetooth*, por lo que en el ejemplo cuenta con comunicación inalámbrica. Podemos encontrar los detalles del ejemplo práctico en el capítulo 15.



## 15 Proyectos futuros

Una vez alcanzados los objetivos propuestos al inicio del proyecto es interesante indagar en posibles trabajos, en los cuales, se pueda incorporar nuestro dispositivo. Debido a que nuestro diseño es una interfaz de comunicación, existen gran cantidad de posibilidades. A continuación vamos a enumerar y explicar brevemente una serie de posibilidades a la hora de emplear nuestro dispositivo en otros trabajos:

- Red de sensores inalámbrica:

Mediante el empleo de la UART diseñada en VHDL podemos incorporar a nuestra FPGA un dispositivo UART-WIFI. Mediante este dispositivo nos podemos comunicar inalámbricamente con sensores mediante el uso de un router. No solo podríamos comunicarnos con sensores inalámbricos cercanos, si no que nos podríamos comunicar con sensores lejanos a través de internet.

- UART con control de flujo y tolerante a fallos:

Siguiendo con el desarrollo de la UART podríamos incorporar un sistema de control de flujo. También convendría incorporar una memoria en el receptor para que el control de flujo tenga sentido. Además se podría diseñar en VHDL un sistema para controlar errores como el error de paridad.

- Cámara de infrarrojos: actualmente existen en el mercado cámaras de fotografía gobernadas mediante una interfaz UART. Empleando nuestro dispositivo e incorporando un sistema de descompresión de JPEG podríamos obtener una cámara digital de bajo coste. Además encontramos en el mercado cámaras que incorporan sensores de infrarrojos por lo que las posibilidades aumentan.
- Controlar dispositivos mediante nuestro *smartphone*: incorporando a nuestra UART un dispositivo UART-bluetooth, podemos enviar instrucciones mediante nuestro Smartphone a una FPGA. Estos nos permite incorporar la FPGA a cualquier dispositivo y controlarlo mediante nuestro móvil.

## 16 Demostración de funcionamiento de la UART

Para determinar el buen funcionamiento de la UART se va a realizar un ejemplo práctico. Para ello se va a implementar el diseño desarrollado en la placa de desarrollo DE0 Nano de Altera.

El proyecto consistirá en enviar una serie de caracteres en código ASCII desde un ordenador. Mediante la UART implementada en la FPGA de la placa, recibiremos estos caracteres y los representaremos en ocho LEDs de la placa. Además podremos enviar caracteres al ordenador desde la FPGA empleando el teclado incorporado.

Para llevar a cabo el ejemplo práctico necesitaremos emplear una serie de dispositivos electrónicos.

### 16.1 Dispositivos empleados

Para poder demostrar el buen funcionamiento de la UART desarrollada en VHDL es necesario emplear una serie de dispositivos que sean capaces de interactuar con ella. En los siguientes apartados se hace una breve descripción de estos dispositivos, así como de la fuente de alimentación de la placa de desarrollo.

#### 16.1.1 Dispositivo convertidor USB-UART

Mediante este dispositivo conseguimos convertir los datos provenientes de la UART a datos compatibles con el protocolo USB. Dado que implementamos una UART en la FPGA y que el ordenador incorpora un USB este dispositivo es necesario para establecer la comunicación. En este caso se producirá la comunicación entre la UART de este dispositivo y la UART que nosotros hemos desarrollado e implementado en la FPGA.

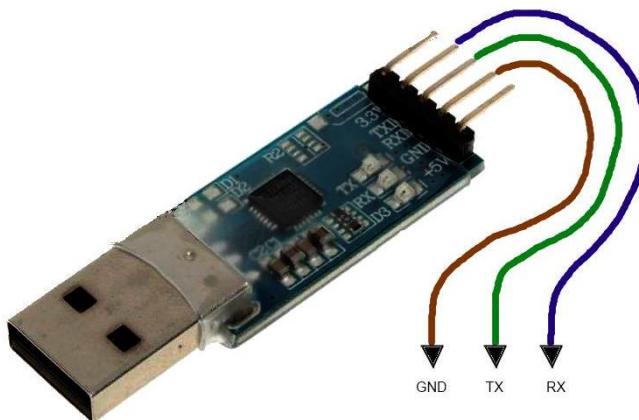


Figura 61: Dispositivo convertidor USB-UART

En la figura 61 se muestra una imagen del dispositivo USB-UART con sus puertos más representativos.

### 16.1.2 Dispositivo convertidor UART-bluetooth (HC-05)

Mediante este dispositivo conseguimos que la comunicación entre la placa de desarrollo y el ordenador sea inalámbrica. Mediante este dispositivo convertimos los datos recibidos de la UART implementada en la FPGA enviándolos inalábricamente al ordenador y viceversa.

Empleamos dos de estos dispositivos. Uno conectado a la placa de desarrollo y otro conectado al ordenador mediante el dispositivo convertidor USB-UART.



Figura 62: Dispositivo UART-bluetooth

En la figura 62 se puede ver una imagen del dispositivo UART-bluetooth

### 16.1.3 Batería USB

Se emplea una batería para alimentar la placa de desarrollo y el dispositivo convertidor UART-bluetooth incorporado. De esta manera el dispositivo es completamente inalámbrico. En este caso se elige alimentar la placa de desarrollo a través del puerto USB. En la figura 63 se muestra una imagen de la batería.



Figura 63: Batería USB

## 16.2 Prueba de funcionamiento

Como hemos comentado en apartados anteriores, para hacer la prueba de funcionamiento vamos a comunicar la placa de desarrollo donde implementamos la UART, con un ordenador. Para ello vamos a emplear los dispositivos anteriormente descritos, dando lugar a una comunicación inalámbrica.

Además de los dispositivos anteriores será necesaria la utilización de otros recursos. Será imprescindible la utilización de un terminal en el ordenador, que permita establecer la comunicación serie. En este caso emplearemos el terminal Termite de CompuPhase. También será necesario configurar los LEDs de la placa de desarrollo, para que representen los datos recibidos en la UART implementada.

Dentro del ejemplo práctico es interesante diferenciar los dos bloques que intervienen en la comunicación:

- El ordenador:

En el ordenador empleamos un terminal para enviar y recibir datos en formato serial. Dentro de este bloque empleamos el convertidor USB-UART. Es necesario ya que el ordenador no incorpora ningún puerto serie y por tanto necesitamos enviar los datos a través del puerto USB y convertirlos. En este bloque también incorporamos uno de los convertidores UART-*bluetooth*. Este bloque se conecta directamente al convertidor anterior. Permite generar una comunicación inalámbrica con el bloque que se explicara a continuación. En la imagen 64 se muestran los dispositivos que componen el bloque del ordenador.

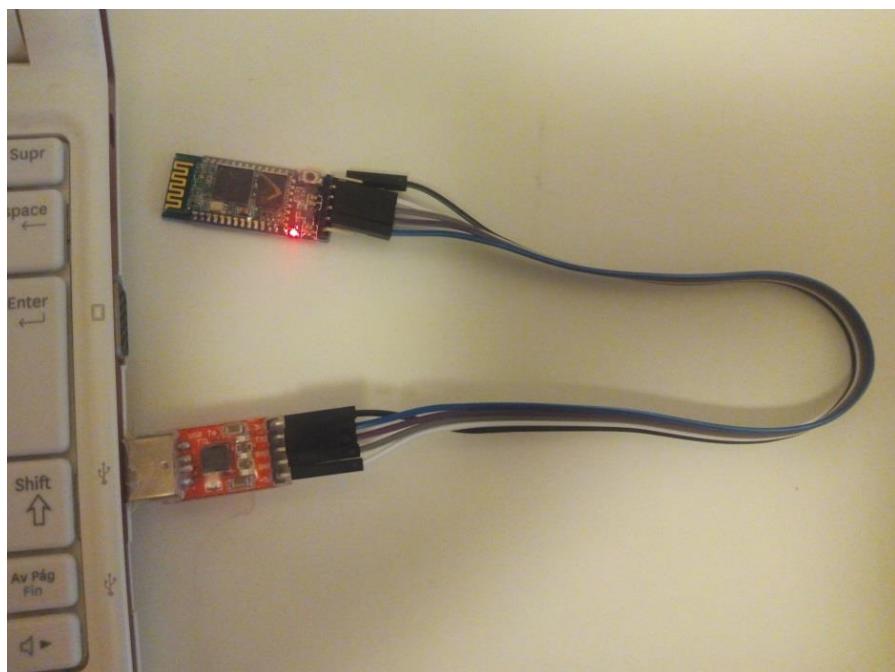


Figura 64: Bloque del ordenador

- Bloque de la placa de desarrollo:

Dentro de la placa de desarrollo implementamos la UART que hemos desarrollado durante el proyecto. Debemos asegurarnos que los datos recibidos en esta UART sean representados en los LEDs que incorpora la propia placa de desarrollo. Además debemos asignar dos puertos de propósito general de la placa de desarrollo para la entrada Rx y para la salida Tx. Estos puertos se conectarán al convertidor UART-*bluetooth* para establecer la comunicación inalámbrica. También será necesario conectar el teclado a los puertos de propósito general. Para alimentar este bloque emplearemos la batería USB que alimentara el puerto USB de la placa de desarrollo. En la figura 65 se muestra los dispositivos que componen el bloque de la placa de desarrollo.

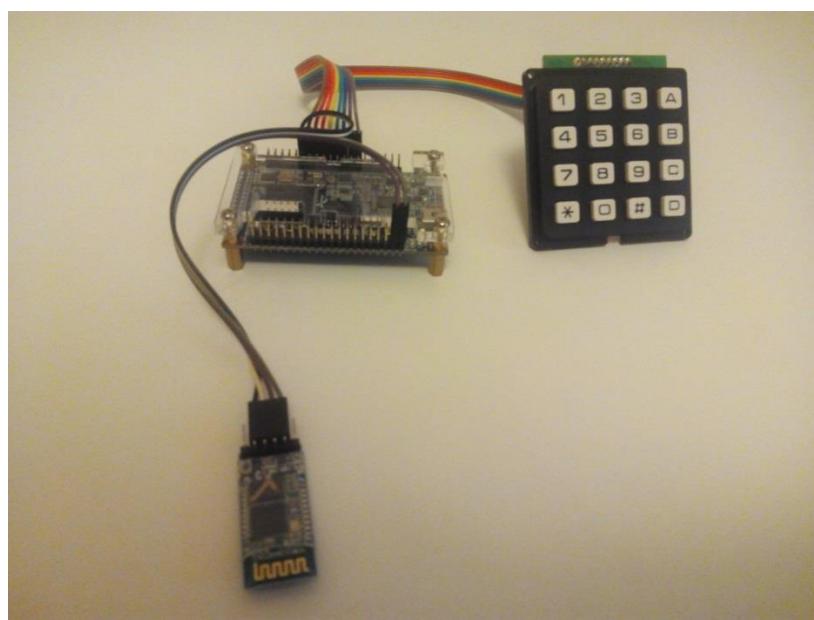


Figura 65: Bloque de la placa de desarrollo

### 16.2.1 Configuración del terminal y el dispositivo UART-bluetooth

Para que se pueda producir la comunicación entre las diferentes UARTs del ejemplo práctico, es necesario que todas ellas tengan configurada la misma velocidad de transmisión. En este caso hemos elegido una velocidad de transmisión de 9600 baudios para este ejemplo, aunque podríamos haber empleado cualquier otra velocidad. Esta es la velocidad que hemos implementado en la UART que hemos desarrollado por lo que es necesario que el resto de dispositivos funcionen acorde a esta velocidad.

Estos son los dispositivos a configurar:

- Terminal del ordenador:

El terminal debe presentar la misma velocidad de transmisión que el resto de UARTs del circuito. Por tanto debemos elegir una velocidad de transmisión de 9600 baudios. Además debemos configurar otros parámetros como el puerto a emplear y el número de bits de STOP que en este caso será uno. En la imagen 66 de muestra una imagen con la configuración adecuada para el terminal.

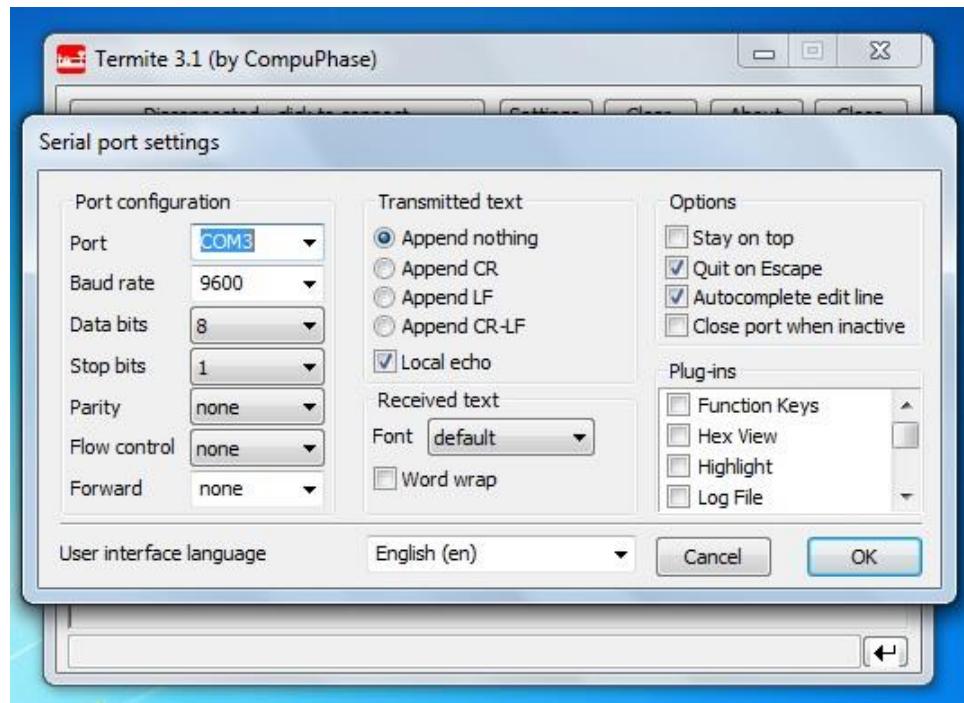


Figura 66: Configuración del terminal

- Configurar dispositivo UART-bluetoot:

Es necesario configurar el *baud rate* de cada uno de los dispositivos UART-Bluetooth. Para ello debemos conectar el dispositivo a al ordenador mediante el dispositivo USB-UART como muestra la figura 67.

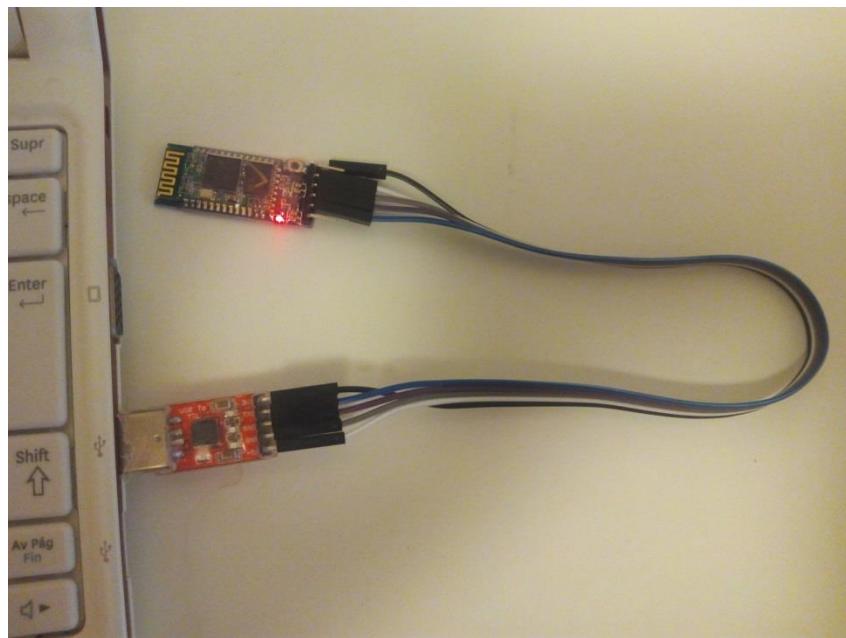


Figura 67: Conexión del dispositivo UART-Bluetoot para su configuración

Se emplea el terminal del ordenador para enviar comandos AT que configuren el dispositivo con el *baud rate* que se emplea en la demostración. En el terminal elegiremos un *baud rate* de 9600 baudios para realizar la configuración. Previamente deberemos haber puesto el dispositivo en modo de configuración, para lo que pulsaremos el botón presente mientras se conecta a la alimentación.



## 17 Conclusión

En este proyecto se ha conseguido implementar una UART mediante VHDL, un lenguaje de descripción de hardware ampliamente empleado en Europa y en todo el mundo.

Para ello hemos necesitado emplear gran cantidad de conocimientos adquiridos durante la carrera. Desde las técnicas necesarias para la correcta búsqueda y uso de información, hasta la técnica de simplificar problemas complejos en bloques más simples que se emplea en cada uno de los campos del grado. Además se han empleado conocimientos más específicos adquiridos durante el estudio de la intensificación en electrónica, como puede ser el VHDL o la capacidad de crear lógica digital.

Durante el desarrollo del proyecto ha sido necesario adquirir diversos conocimientos además de los ya adquiridos. Ha sido necesario incrementar los conocimientos en VHDL además de emplear entornos de desarrollo como Quartus II de Altera. Remarcar que durante la carrera se empleó el ISE de Xilinx por lo que ha sido interesante extrapolar los conocimientos adquiridos a otro entorno de desarrollo. Además ha sido necesario un gran estudio para realizar las simulaciones, tanto ideales como con retardos con la herramienta Altera-Modelsim.

Por otro lado se ha conseguido crear un dispositivo desde cero, empleando hojas de catálogo de otros dispositivos similares y empleando conocimientos dispersos en la red. A pesar de esto se ha conseguido que el dispositivo estable una comunicación con cualquier otro dispositivo UART presente en el mercado. Por lo tanto se ha creado un dispositivo nuevo bajo un estándar bien definido y completamente funcional.

Además el dispositivo puede emplearse en cualquier FPGA del mercado debido a su implementación en VHDL. Debido a su escaso empleo de celdas lógicas se puede implementar en casi cualquier FPGA de bajo coste. Además gracias a la incorporación de un generador de baudios puede trabajar con un gran rango de relojes.

Cabe tener en cuenta que hay una gran cantidad de electrónica en el mercado que incorpora una interfaz UART por lo que hemos posibilitado la comunicación de una FPGA con una gran gama de dispositivos. Además existen dispositivos UART-bluetooth y UART-Wifi que permiten incorporar a nuestro diseño, estándares muy extendidos de comunicación inalámbrica tanto a corta como a larga distancia. Podemos comunicar nuestro FPGA a través de internet vía Wifi o con otro dispositivo cercano vía bluetooth.

Debido al punto anterior se nos abre un amplio marco de proyectos futuros. Podemos realizar proyectos tan diversos como incorporar nuestro diseño a un microprocesador diseñado en VHDL, diseñar una red de sensores inalámbrica o conectar una serie de FPGAs y microcontroladores entre sí para que intercambien información.

Por tanto, debido a la gran cantidad de conocimientos reforzados y adquiridos; al buen funcionamiento del dispositivo desarrollado; a su gran versatilidad; a la capacidad de incorporar estándares de comunicación muy extendido en la actualidad y al gran marco de



proyectos que se nos abre a continuación podemos concluir que el desarrollo del proyecto ha sido más que satisfactorio.



## 18 Bibliografía

- [1] <http://digital.ni.com/public.nsf/allkb/039001258CEF8FB686256E0F005888D1>
- [2] [http://catarina.udlap.mx/u\\_dl\\_a/tales/documentos/lem/morales\\_h\\_oe/capitulo3.pdf](http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/morales_h_oe/capitulo3.pdf)
- [3] <http://perso.wanadoo.es/pictob/comserie.htm>
- [4] [http://es.wikipedia.org/wiki/Puerto\\_serie](http://es.wikipedia.org/wiki/Puerto_serie)
- [5] [http://es.wikipedia.org/wiki/UART\\_8250](http://es.wikipedia.org/wiki/UART_8250)
- [6] [http://en.wikipedia.org/wiki/16550\\_UART](http://en.wikipedia.org/wiki/16550_UART)
- [7] <http://electronics.stackexchange.com/questions/37814/usart-uart-rs232-usb-spi-i2c-ttl-etc-what-are-all-of-these-and-how-do-th>
- [8] <http://www.cprogramdevelop.com/2301056/>
- [9] [http://www.swarthmore.edu/NatSci/echeeve1/Class/e91/Lectures/E91%2810%29\\_Serial.pdf](http://www.swarthmore.edu/NatSci/echeeve1/Class/e91/Lectures/E91%2810%29_Serial.pdf)
- [10] [http://www.comm.pub.ro/dicm/C7\\_Serial\\_Bus.pdf](http://www.comm.pub.ro/dicm/C7_Serial_Bus.pdf)
- [11] <http://geekeninformacion.blogspot.com.es/2013/01/interfaces-de-comunicacion-spi-i2c-uart.html>
- [12] [http://es.wikipedia.org/wiki/Serial\\_Peripheral\\_Interface](http://es.wikipedia.org/wiki/Serial_Peripheral_Interface)
- [13] <http://es.wikipedia.org/wiki/I%C2%B2C>
- [14] [http://www.cimco.com/docs/cimco\\_dnc-max/v7/es/#SerialComStandards](http://www.cimco.com/docs/cimco_dnc-max/v7/es/#SerialComStandards)
- [15] <http://es.wikipedia.org/wiki/RS-232>
- [16] [http://es.wikipedia.org/wiki/UART\\_8250](http://es.wikipedia.org/wiki/UART_8250)
- [17] <http://digital.ni.com/public.nsf/allkb/039001258CEF8FB686256E0F005888D1>
- [18] <http://whatis.techtarget.com/definition/UART-Universal-Asynchronous-Receiver-Transmitter>
- [19] <https://www.youtube.com/watch?v=fMmcSpgOtJ4>



[20][https://www.youtube.com/watch?v=5Rna\\_pRyVU0](https://www.youtube.com/watch?v=5Rna_pRyVU0)

[21]<http://opencores.com/>

[22]<http://www.logicbricks.com/>

[23]<http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=593&PartNo=4>

[24]<http://ww1.microchip.com/downloads/en/DeviceDoc/39626e.pdf>

## 19 Anexo A: Cálculo del Baud Rate

En este capítulo se calculan los valores necesarios para obtener la configuración adecuada del generador de baudios en función del *baud rate* elegido. Entre estos valores destaca el cálculo del valor a introducir en el generador de baudios (El valor que se introduce en el bloque SPBRGVALUE). Por otro lado se calcula el error que se comete con las fórmulas empleadas y se confecciona una tabla con todos los valores de configuración para una serie de velocidades estándar.

### 19.1 Forma de calcular el Baud Rate

La UART que se desarrolla en este proyecto incorpora un generador de baudios que nos permite seleccionar la velocidad con la que se transmiten y se reciben los bits de información. Para seleccionar una velocidad concreta es necesario que se calculen los parámetros a introducir dentro del generador. Dependiendo del modo en el que el la UART vaya a funcionar el cálculo de estos parámetros se realiza con una fórmula distinta. En la figura 68 se muestra un esquema de la entidad del generador de baudios.



Figura 68: Entradas y salidas del generador de baudios

Hay que tener en cuenta que el generador de baudios cambia su funcionamiento en función de las señales BRGH y BRG16. La señal BRGH ayuda a determinar la selección de la fórmula para calcular el valor a cargar en la entrada SPBRGVALUE del generador. La señal BRG16 también ayuda a determinar lo anterior, pero además selecciona si el generador funciona con un registro de 8 bits o 16 bits. Este registro funciona como un contador, que cuenta ciclos de reloj hasta que se alcanza la mitad del valor cargado en SPBRGVALUE. En ese momento se produce un flanco en el reloj del generador. En función de si se utilizan 8 bits o 16 bits el dispositivo podrá cargar velocidades más lentas o con mayor precisión.

Como hemos señalado durante el trabajo el generador de baudios produce un reloj con una frecuencia 4, 16 o 64 veces superior a la frecuencia con la que se transmite o se recibe un bit. Esto es necesario ya que nos interesa recibir los bits de tal manera que podamos muestrearlos en el medio. Así evitamos posibles errores producidos por muestrear el bit cuando está

cambiando al siguiente. Evidentemente si el reloj del generador funciona con una frecuencia 64 veces superior a la frecuencia del *baud rate*, muestrearemos más en el medio que si solo es 16 veces superior.

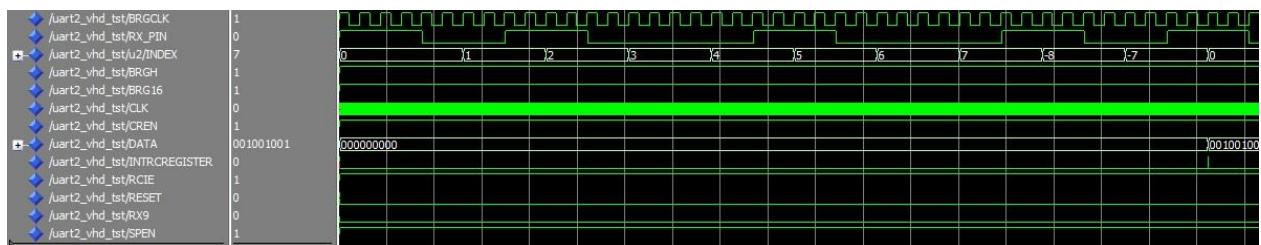


Figura 69: Muestreo de los bits en el medio

Como podemos apreciar en la figura 69 el reloj generado por el generador de baudios BRGCLK tiene una frecuencia cuatro veces superior a la frecuencia de recepción de bits. Como se ve reflejado por la señal INDEX el muestreo coincide con el flanco del reloj BRGCLK, que está en el medio del bit.

La tabla 22 muestra un resumen con los modos de funcionamiento y las fórmulas en función de las variables BRGH y BRG16:

Configuración		Modo del generador	Fórmula del baud rate
BRG16	BRGH		
0	0	8 bits	$FOSC/[64 (n + 1)]$
0	1	8 bits	$FOSC/[16 (n + 1)]$
1	0	16 bits	
1	1	16 bits	$FOSC/[4 (n + 1)]$

Tabla 22: Fórmulas para calcular el baud rate en función de la configuración del dispositivo

El valor denominado como “n” es el valor que hay que cargar en la entrada SPBRGVALUE del generador de baudios para obtener el *baud rate* adecuado.

## 19.2 Ejemplo de cálculo del baud rate

Vamos a realizar un ejemplo práctico del cálculo del valor de la entrada SPBRGVALUE. Para ello vamos a configurar el generador de baudios de la siguiente manera:

- BRGH: 0
- BRG16: 0
- *baud rate*: 9600 baudios
- Frecuencia del reloj: 50 MHz

Dado que la señal BRGH es igual a cero y la señal BRG16 es igual a cero, debemos emplear la siguiente fórmula como se puede apreciar en la tabla anterior.



$$\text{Baud Rate} = \frac{\text{frecuencia del reloj}}{[64(n + 1)]}$$

Despejamos “n”:

$$n = \frac{\text{frecuencia del reloj}}{\text{Baud Rate} * 64} - 1$$

$$n = \frac{50 * 10^6}{9600 * 64} - 1 = 80,380208$$

Por tanto introduciríamos un valor de 80 en la entrada SPBRGVALUE, en binario.

En este caso el generador de baudios tendría un modo de funcionamiento de 8 bits, por lo que la entrada SPBRGVALUE debe ser un numero binario de como máximo 8 bits. Esto se debe a que solo se emplea el registro SPBRG, de ocho bits, para realizar la temporización del generador; dejando el registro SPBRGH sin utilizar.

Por lo anterior siempre es interesante comprobar que el valor “n” no supera los ocho bits.

El valor “80” en decimal equivale al valor “101000” en binario, y como podemos comprobar no supera los ocho bits.

### 19.3 Cálculo del error del baud rate

Debido a las características del generador de baudios tenemos que realizar cálculos para determinar el error que se obtiene al aplicar un *baud rate* determinado. Como hemos observado en cálculos anteriores, al despejar “n” se obtiene un valor con decimales que hay que redondear para introducirlo en la entrada SPBRGVALUE. Esto da lugar a que el *baud rate* no sea exacto. Por lo tanto es necesario cuantificar este error para determinar si será relevante en la transmisión de datos.

Primero hay que calcular el valor de “n” para un *baud rate* teórico. Para ello seleccionamos la fórmula en función de la configuración y despejamos “n”. El valor obtenido para “n” presentara decimales que no hay que redondear.

A continuación calculamos un *baud rate* real a partir del valor de n obtenido.

Para finalizar empleamos la siguiente fórmula:

$$Error(\%) = \frac{Baud\ Rate\ Real - Baud\ Rate\ teórico}{Baud\ Rate\ teórico} * 100$$

De esta forma obtenemos el error.

## 19.4 Ejemplo de cálculo del error del baud rate

En este caso vamos a partir del ejemplo anterior:

- BRGH: 0
- BRG16: 0
- *baud rate*: 9600 baudios
- Frecuencia del reloj: 50 MHz
- $n = 80.380208$

Fórmula:

$$Baud\ Rate = \frac{\text{frecuencia del reloj}}{[64(n + 1)]}$$

Dado que ya hemos calculado “n” vamos a calcular el *Baud Rate Real*:

$$Baud\ Rate\ Real = \frac{\text{frecuencia del reloj}}{[64(n + 1)]}$$

$$Baud\ Rate\ Real = \frac{50 * 10^6}{[64(80,380208 + 1)]} = 9600,00003932$$

A continuación empleamos la fórmula:

$$Error(\%) = \frac{Baud\ Rate\ Real - Baud\ Rate\ teórico}{Baud\ Rate\ teórico} * 100$$



$$Error(\%) = \frac{9600,00003932 - 9600}{9600} * 100 = 0\%$$

En este caso el error es completamente despreciable.

## 19.5 Tablas con los cálculos para baud rates estándar

Las siguientes tablas muestran todos los datos relativos al *baud rate* en función del tipo de configuración y de la frecuencia del reloj.

### 19.5.1 BRGH=0 BRG16=0. Modo de ocho bits

Baud rate teórico (baudios/s)	BRGH=0 BRG16=0					
	50MHz			40MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	300,019201	0,006400	2603	300,048008	0,016003	2082
1200	1200,076805	0,006400	650	1199,616123	-0,031990	520
2400	2396,472393	-0,146984	325	2403,846154	0,160256	259
9600	9645,061728	0,469393	80	9615,384615	0,160256	64
19200	19054,878049	-0,755843	40	18939,393939	-1,357323	32
57600	55803,571429	-3,118800	13	56818,181818	-1,357323	10
115200	111607,142857	-3,118800	6	125000,000000	8,506944	4

Tabla 23: Cálculo de "n" para cada baud rate para frecuencias de 50 y 40 MHz

Baud rate teórico (baudios/s)	BRGH=0 BRG16=0					
	20MHz			10MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	299,904031	-0,031990	1041	299,904031	-0,031990	520
1200	1201,923077	0,160256	259	1201,923077	0,160256	129
2400	2403,846154	0,160256	129	2403,846154	0,160256	64
9600	9469,696970	-1,357323	32	9765,625000	1,725260	15
19200	19531,250000	1,725260	15	19531,250000	1,725260	7
57600	62500,000000	8,506944	4	52083,333333	-9,577546	2
115200	104166,666667	-9,577546	2	156250,000000	35,633681	0

Tabla 24: Cálculo de "n" para cada baud rate para frecuencias de 20 y 10 MHz



Baud rate teórico (baudios/s)	BRGH=0 BRG16=0					
	8MHz			4MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	299,760192	-0,079936	416	300,480769	0,160256	207
1200	1201,923077	0,160256	103	1201,923077	0,160256	51
2400	2403,846154	0,160256	51	2403,846154	0,160256	25
9600	9615,384615	0,160256	12	8928,571429	-6,994048	6
19200	17857,142857	-6,994048	6	20833,333333	8,506944	2
57600	62500,000000	8,506944	1	62500,000000	8,506944	0
115200	125000,000000	8,506944	0	62500,000000	-45,746528	0

Tabla 25: Cálculo de "n" para cada baud rate para frecuencias de 8 y 4 MHz

Baud rate teórico (baudios/s)	BRGH=0 BRG16=0					
	2MHz			1MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	300,480769	0,160256	103	300,480769	0,160256	51
1200	1201,923077	0,160256	25	1201,923077	0,160256	12
2400	2403,846154	0,160256	12	2232,142857	-6,994048	6
9600	10416,666667	8,506944	2	7812,500000	-18,619792	1
19200	15625,000000	-18,619792	1	15625,000000	-18,619792	0
57600	31250,000000	-45,746528	0	#DIV/0!	#DIV/0!	-1
115200	#DIV/0!	#DIV/0!	-1	#DIV/0!	#DIV/0!	-1

Tabla 26: Cálculo de "n" para cada baud rate para frecuencias de 2 y 1 MHz

Destacar que al presentar un modo de funcionamiento de ocho bits no tienen sentido valores de "n" mayores de 256 .

### 19.5.2 BRGH=0 BRG16=1. Modo de 16 bits

Baud rate teórico (baudios/s)	BRGH=0 BRG16=1					
	50MHz			40MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	299,990400	-0,003200	10416	300,012000	0,004000	8332
1200	1200,076805	0,006400	2603	1200,192031	0,016003	2082
2400	2400,153610	0,006400	1301	2399,232246	-0,031990	1041
9600	9585,889571	-0,146984	325	9615,384615	0,160256	259
19200	19171,779141	-0,146984	162	19230,769231	0,160256	129
57600	57870,370370	0,469393	53	58139,534884	0,936693	42
115200	115740,740741	0,469393	26	113636,363636	-1,357323	21

Tabla 27: Cálculo de "n" para cada baud rate para frecuencias de 50 y 40 MHz

Baud rate teórico (baudios/s)	BRGH=0 BRG16=1					
	20MHz			10MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	299,976002	-0,007999	4166	300,048008	0,016003	2082
1200	1199,616123	-0,031990	1041	1199,616123	-0,031990	520
2400	2399,232246	-0,031990	520	2403,846154	0,160256	259
9600	9615,384615	0,160256	129	9615,384615	0,160256	64
19200	19230,769231	0,160256	64	18939,393939	-1,357323	32
57600	56818,181818	-1,357323	21	56818,181818	-1,357323	10
115200	113636,363636	-1,357323	10	125000,000000	8,506944	4

Tabla 28: Cálculo de "n" para cada baud rate para frecuencias de 20 y 10 MHz



Baud rate teórico (baudios/s)	BRGH=0 BRG16=1					
	8MHz			4MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	299,940012	-0,019996	1666	300,120048	0,040016	832
1200	1199,040767	-0,079936	416	1201,923077	0,160256	207
2400	2403,846154	0,160256	207	2403,846154	0,160256	103
9600	9615,384615	0,160256	51	9615,384615	0,160256	25
19200	19230,769231	0,160256	25	19230,769231	0,160256	12
57600	55555,555556	-3,549383	8	62500,000000	8,506944	3
115200	125000,000000	8,506944	3	125000,000000	8,506944	1

Tabla 29: Cálculo de "n" para cada baud rate para frecuencias de 8 y 4 MHz

Baud rate teórico (baudios/s)	BRGH=0 BRG16=1					
	2MHz			1MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	299,760192	-0,079936	416	300,480769	0,160256	207
1200	1201,923077	0,160256	103	1201,923077	0,160256	51
2400	2403,846154	0,160256	51	2403,846154	0,160256	25
9600	9615,384615	0,160256	12	8928,571429	-6,994048	6
19200	17857,142857	-6,994048	6	20833,333333	8,506944	2
57600	62500,000000	8,506944	1	62500,000000	8,506944	0
115200	125000,000000	8,506944	0	62500,000000	-45,746528	0

Tabla 30: Cálculo de "n" para cada baud rate para frecuencias de 2 y 1 MHz

Al presentar un modo de funcionamiento de 16 bits no tienen sentido valores de "n" mayores de 65535.

### 19.5.3 BRGH=1 BRG16=0. Modo de 8 bits

Baud rate teórico (baudios/s)	BRGH=1 BRG16=0					
	50MHz			40MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	299,990400	-0,003200	10416	300,012000	0,004000	8332
1200	1200,076805	0,006400	2603	1200,192031	0,016003	2082
2400	2400,153610	0,006400	1301	2399,232246	-0,031990	1041
9600	9585,889571	-0,146984	325	9615,384615	0,160256	259
19200	19171,779141	-0,146984	162	19230,769231	0,160256	129
57600	57870,370370	0,469393	53	58139,534884	0,936693	42
115200	115740,740741	0,469393	26	113636,363636	-1,357323	21

Tabla 31: Cálculo de "n" para cada baud rate para frecuencias de 50 y 40 MHz

Baud rate teórico (baudios/s)	BRGH=1 BRG16=0					
	20MHz			10MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	299,976002	-0,007999	4166	300,048008	0,016003	2082
1200	1199,616123	-0,031990	1041	1199,616123	-0,031990	520
2400	2399,232246	-0,031990	520	2403,846154	0,160256	259
9600	9615,384615	0,160256	129	9615,384615	0,160256	64
19200	19230,769231	0,160256	64	18939,393939	-1,357323	32
57600	56818,181818	-1,357323	21	56818,181818	-1,357323	10
115200	113636,363636	-1,357323	10	125000,000000	8,506944	4

Tabla 32: Cálculo de "n" para cada baud rate para frecuencias de 20 y 10 MHz



Baud rate teórico (baudios/s)	BRGH=1 BRG16=0					
	8MHz			4MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	299,940012	-0,019996	1666	300,120048	0,040016	832
1200	1199,040767	-0,079936	416	1201,923077	0,160256	207
2400	2403,846154	0,160256	207	2403,846154	0,160256	103
9600	9615,384615	0,160256	51	9615,384615	0,160256	25
19200	19230,769231	0,160256	25	19230,769231	0,160256	12
57600	55555,555556	-3,549383	8	62500,000000	8,506944	3
115200	125000,000000	8,506944	3	125000,000000	8,506944	1

Tabla 33: Cálculo de "n" para cada baud rate para frecuencias de 8 y 4 MHz

Baud rate teórico (baudios/s)	BRGH=1 BRG16=0					
	2MHz			1MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	299,760192	-0,079936	416	300,480769	0,160256	207
1200	1201,923077	0,160256	103	1201,923077	0,160256	51
2400	2403,846154	0,160256	51	2403,846154	0,160256	25
9600	9615,384615	0,160256	12	8928,571429	-6,994048	6
19200	17857,142857	-6,994048	6	20833,333333	8,506944	2
57600	62500,000000	8,506944	1	62500,000000	8,506944	0
115200	125000,000000	8,506944	0	62500,000000	-45,746528	0

Tabla 34: Cálculo de "n" para cada baud rate para frecuencias de 2 y 1 MHz

#### 19.5.4 BRGH=1 BRG16=1. Modo de 16 bits

Baud rate teórico (baudios/s)	BRGH=1 BRG16=1					
	50MHz			40MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	299,997600	-0,000800	41666	300,003000	0,001000	33332
1200	1199,961601	-0,003200	10416	1200,048002	0,004000	8332
2400	2400,153610	0,006400	5207	2399,808015	-0,007999	4166
9600	9600,614439	0,006400	1301	9596,928983	-0,031990	1041
19200	19201,228879	0,006400	650	19193,857965	-0,031990	520
57600	57603,686636	0,006400	216	57471,264368	-0,223499	173
115200	114678,899083	-0,452345	108	114942,528736	-0,223499	86

Tabla 35: Cálculo de "n" para cada baud rate para frecuencias de 50 y 40 MHz

Baud rate teórico (baudios/s)	BRGH=1 BRG16=1					
	20MHz			10MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	299,994000	-0,002000	16666	300,012000	0,004000	8332
1200	1199,904008	-0,007999	4166	1200,192031	0,016003	2082
2400	2400,384061	0,016003	2082	2399,232246	-0,031990	1041
9600	9596,928983	-0,031990	520	9615,384615	0,160256	259
19200	19230,769231	0,160256	259	19230,769231	0,160256	129
57600	57471,264368	-0,223499	86	58139,534884	0,936693	42
115200	116279,069767	0,936693	42	113636,363636	-1,357323	21

Tabla 36: Cálculo de "n" para cada baud rate para frecuencias de 20 y 10 MHz



Baud rate teórico (baudios/s)	BRGH=1 BRG16=1					
	8MHz			4MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	299,985001	-0,005000	6666	300,030003	0,010001	3332
1200	1199,760048	-0,019996	1666	1200,480192	0,040016	832
2400	2400,960384	0,040016	832	2398,081535	-0,079936	416
9600	9615,384615	0,160256	207	9615,384615	0,160256	103
19200	19230,769231	0,160256	103	19230,769231	0,160256	51
57600	57142,857143	-0,793651	34	58823,529412	2,124183	16
115200	117647,058824	2,124183	16	111111,111111	-3,549383	8

Tabla 37: Cálculo de "n" para cada baud rate para frecuencias de 8 y 4 MHz

Baud rate teórico (baudios/s)	BRGH=1 BRG16=1					
	2MHz			1MHz		
	Baud rate real	Error %	n	Baud rate real	Error %	n
300	299,940012	-0,019996	1666	300,120048	0,040016	832
1200	1199,040767	-0,079936	416	1201,923077	0,160256	207
2400	2403,846154	0,160256	207	2403,846154	0,160256	103
9600	9615,384615	0,160256	51	9615,384615	0,160256	25
19200	19230,769231	0,160256	25	19230,769231	0,160256	12
57600	55555,555556	-3,549383	8	62500,000000	8,506944	3
115200	125000,000000	8,506944	3	125000,000000	8,506944	1

Tabla 38: Cálculo de "n" para cada baud rate para frecuencias de 2 y 1 MHz

