Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Control de Dispositivos Externos, desde una FPGA, vía Bluetooth

Autor:

Prados Sesmero, Carlos

Tutor:

Rodríguez Trelles, Francisco José de Andrés
Departamento de Tecnología Electrónica

Valladolid, Junio de 2018

Resumen

En el Departamento de Tecnología Electrónica de la Universidad de Valladolid se ha planteado el desarrollo de una estación meteorológica inteligente basada en dispositivos FPGA, incluyendo la conexión inalámbrica entre varias estaciones, con sensores remotos y con dispositivos externos.

Este proyecto presenta el diseño e implantación del sistema de comunicación entre los distintos elementos que constituyen la citada estación, así como los dispositivos externos.

El diseño está basado en dispositivos FPGA, en sensores y en los elementos necesarios para la comunicación vía Bluetooth.

Se ha realizado un diseño estructurado y jerárquico, empleando el software de diseño Lattice Diamond. Como lenguaje de descripción de hardware se ha empleado VHDL, el cual se ha complementado con herramientas externas para realizar la implantación de máquinas de estados finitos y aplicaciones móviles.

Finalmente, se ha implementado el sistema físico completo y se ha comprobado su correcto funcionamiento, documentando todo ello a través de LaTeX.

Abstract

In the Electronic Technology Department of the University of Valladolid the approach of an intelligent meteorological station based on FPGA devices has taken place, including wireless connection between several stations, remote sensors and external devices.

This project presents the design and development of the communication system between the different elements that make up the station, as well as the external devices.

The design is based on FPGA devices, sensors and the necessary elements for communication through Bluetooth.

A structured and hierarchical design has been made, using the Lattice Diamond design software. VHDL has been used like hardware description language, which has been complemented with external tools to facilitate the implementation of finite state machines and mobile applications.

Finally, the complete physical system has been implemented and its correct functioning has been verified, documenting all this through LaTeX.

Palabras Clave

Bluetooth - Comando AT - Comunicación - FPGA - VHDL.

Keywords

Bluetooth - AT Comand - Comunication - FPGA - VHDL.

Terminología empleada

Se presentan a continuación una serie de definiciones útiles a lo largo de la memoria del proyecto, mostradas cada una de ellas alfabéticamente ordenadas. A lo largo de dicha memoria se marcarán las palabras pertenecientes a esta lista con el carácter "*".

ATCommand: Orden o instrucción que se utiliza para configurar los parámetros de los diferentes módulos (entre los que encontramos los módulos de Bluetooth), pudiendo determinar las diferentes funciones del mismo. Los parámetros que se pueden modificar son, entre otros: velocidad de transmisión de datos por el puerto serie, frecuencia del canal de comunicación inalámbrico, modo de transmisión transparente, energía transmitida al módulo, etc. Originalmente se conocían como comandos Hayes, convirtiéndose en un estándar abierto de comandos para configurar y parametrizar módems.

Banda ISM: Bandas reservadas internacionalmente para uso no comercial de radiofrecuencia electromagnética en áreas industriales, científicas y médicas. Incluye la frecuencia de 433MHz que utilizaremos en este trabajo.

Big Endian: Formato de almacenamiento o emisión de datos de tal manera que se almacena o envía el dato más significativo en primer lugar.

Bluetooth: Especificación industrial para Redes Inalámbricas de Área Personal (WPAN*) que posibilita la transmisión de datos entre diferentes dispositivos mediante un enlace de radiofrecuencia en la banda ISM. Es decir, ofrece la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

Bus I2C: Inter-Integrated Circuit. Es un bus serie de datos que se utiliza internamente para la comunicación entre diferentes partes de un circuito entre un controlador y circuitos periféricos integrados.

Bus SPI: Serial Peripheral Interface. Es un estándar de comunicaciones usado principalmente para la transferencia de información entre circuitos integrados en

equipos electrónicos. Sirve para controlar casi cualquier dispositivo electrónico digital que acepte una comunicación sincrónica.

Comunicación: Intercambio de información entre dos o más dispositivos. Los pasos básicos son la elaboración del mensaje por parte de un emisor, su codificación, la transmisión de la señal, su recepción, la decodificación del mensaje y finalmente, su interpretación por parte de un receptor. La comunicación se puede desarrollar a partir de un medio de propagación físico o de manera inalámbrica. Esta última utiliza la modulación de ondas electromagnéticas a través del espacio, aportando una mayor flexibilidad y movilidad de los dispositivos que se comunican.

FPGA: Una FPGA o matriz de puertas programables es un dispositivo programable que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada in situ mediante un lenguaje de descripción especializado. La gran flexibilidad de poder cambiar su configuración hace que su coste sea menor tanto para pequeños lotes de fabricación como para prototipado. Las FPGA fueron inventadas en el año 1984, como evolución a las PAL y los cPLD. Cualquier circuito de aplicación específica puede ser implementado en una FPGA, siempre y cuando esta disponga de los recursos necesarios.

Interfaz: Conexión funcional entre dos dispositivos, que proporciona una comunicación entre distintos niveles permitiendo el intercambio de información.

JTAG: Joint Test Action Group. Se refiere a los pines o puertos de acceso a test, utilizados para testear PCBs utilizando escaneo de límites.

Little Endian: Formato de almacenamiento o emisión de datos de tal manera que se almacena o envía el dato menos significativo en primer lugar.

Luts4: Componente principal de los Slices. Cada uno de ellos es capaz de realizar una función de 4 variables.

Memoria Flash: Deriva de las siglas EEPROM y permite la lectura y escritura de múltiples posiciones de memoria en una misma operación, consiguiendo grandes velocidades de funcionamiento. Es una memoria no volátil.

Memoria RAM: Memoria de acceso aleatorio. Es capaz de leer o escribir en una posición de memoria con un tiempo de espera igual para cualquier posición. Es una memoria volátil.

MIT App Inventor: Entorno de desarrollo de software creado para la elaboración de aplicaciones destinadas al sistema operativo Android. El usuario puede enlazar una serie de bloques para crear la lógica de la aplicación.

PCB: Placa de circuito impreso. Es la superficie constituida por caminos, pistas o buses de material conductor laminadas sobre una base no conductora. Se utiliza para conectar eléctricamente a través de las pistas conductoras, y sostener mecánicamente, por medio de la base, un conjunto de componentes electrónicos.

PDU: Protocol Data Units. Mensaje con información intercambiado entre entidades pares (del mismo nivel) dialogando mediante el protocolo definido de ese nivel.

PFU: Programmable Function Units. Son unidades de funciones programables que se utilizan en FPGAs para la generación de memorias o lógica. A la hora de la generación de memorias se realizarán mediante memoria RAM distribuida. Cada PFU está formada por 4 Slices*. Cada PFU tendrá la capacidad de generar una función de 7 variables, 2 funciones de 6 variables, 4 funciones de 5 variables o 8 funciones de 4 variables.

PLD: Programmable Logic Device. Circuito integrado que puede programarse para realizar funciones complejas. Se compone de grupos de puertas AND y OR, desarrollándose la combinación deseada para el desarrollo de un circuito específico.

Protocolo: Descripción de la forma de realizar determinadas funciones para garantizar la compatibilidad en el nivel correspondiente. Se definen una serie de reglas que determinan como realizar estas funciones (información intercambiada, formato del mensaje, establecimiento del diálogo, etc.).

Radiofrecuencia: Término que se aplica a la porción menos energética del espectro electromagnético, situada entre los 3Hz y 300GHz.

SDU: Service Data Unit. Mensaje intercambiado con información entre entidades de niveles adyacentes. En el caso de esta práctica el intercambio de información se produce entre la FPGA y el módulo HC-12.

Sistema embebido: Sistema de computación diseñado para realizar una o pocas funciones dedicadas, frecuentemente en un sistema de computación en tiempo real, es decir, están diseñados normalmente para aplicaciones específicas.

Slice: Componente principal de las PFUs. Cada uno de ellos posee 2 Luts4*, es decir, tendrá la capacidad de generar una función de 5 variables o 2 funciones de 4 variables.

SPP: Interfaz de alto nivel para su uso entre dispositivos Bluetooth. Define el formato utilizado a la hora de enviar paquetes de datos.

VHDL: Very High Speed Integrated Circuit, Hardware Description Language. Lenguaje de especificación definido por el IEEE utilizado para describir circuitos digitales y para la automatización de diseño electrónico. Este lenguaje fue creado por el departamento de defensa de EEUU con el objetivo de simular circuitos eléctricos digitales.

Wishbone Bus: Bus estándar destinado a permitir que diferentes partes de un circuito integrado se comuniquen entre sí. Está diseñado como un "bus lógico" con código abierto.

WPAN: Red de Área Personal que permite la comunicación entres distintos dispositivos cercanos al punto de acceso.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y Objetivos

El objetivo fundamental de este TFG es la implantación de comunicaciones inalámbricas, entre los elementos de una estación meteorológica (sensores, dispositivos de procesamiento y visualización de información). Esta estación se ha desarrollado en otro TFG, que ha sido desarrollado en paralelo con éste. Evidentemente, aunque se trata de dos trabajos independientes, están relacionados entre sí.

También se va a desarrollar la implementación de la comunicación entre la estación meteorológica y el exterior. La estación meteorológica está basada en dispositivos FPGA, incorporando sensores de temperatura y humedad, mientras que la comunicación se realizará mediante módulos de Bluetooth.

Cuando se pretende realizar un sistema digital, se presentan diferentes opciones, entre las que se encuentra la implementación a través de microcontroladores, PLDs* o FPGAs*. Este último es un dispositivo reprogramable (característica que le añade una gran flexibilidad al flujo de diseño), el coste de desarrollo y adquisición son relativamente bajos para pequeñas cantidades de dispositivos y el tiempo de desarrollo es, dentro de lo que cabe, asequible para gran cantidad de aplicaciones.

La utilización de estos dispositivos permite una gran flexibilidad en cada aplicación. Sin embargo, se presenta un problema, una vez realizadas todas las operaciones específicas de cualquier aplicación, la información queda retenida en el dispositivo. Algunos dispositivos pueden requerir los resultados obtenidos en otros, para lo cual se tiene que establecer una comunicación entre ambos y procediendo al paso de información.

La comunicación se puede establecer de dos modos diferentes: mediante cable (SPI* o I2C*) o de manera inalámbrica (Bluetooth*, Infrarrojo, GSM, etc.). Este proyecto se ha decantado por una comunicación inalámbrica, vía Bluetooth, que permite una mayor flexibilidad, un ahorro de componentes y una mayor movilidad, facilitando la utilización de dispositivos móviles. En contrapartida podemos

destacar una peor calidad de servicio (en relación con la velocidad de transferencia y una mayor tasa de error), una limitación de cobertura y un riesgo de cara a la seguridad de nuestros datos.

Para la aplicación a la que se encuentra destinada este proyecto, las ventajas de la comunicación inalámbrica son mucho mayores que los inconvenientes, destacando como ventaja principal el aprendizaje de los formatos y protocolos utilizados por los dispositivos Bluetooth.

El proyecto que se presenta en este documento consiste en el establecimiento de la comunicación entre dos dispositivos FPGA utilizando como vía de comunicación la radiofrecuencia*, con ayuda de dispositivos Bluetooth. En modo de ampliación, se desarrolla una aplicación app móvil, en donde se visualizan los datos enviados, de manera que se pueda observar de una manera más visual y efectiva la información deseada en el teléfono móvil, aparato que toma cada vez más presencia en el día a día de cualquier persona.

Para el desarrollo de la comunicación mediante Bluetooth se necesita un módulo de transmisión de datos. Se busca, para ello, un dispositivo que permita versatilidad de desarrollo y facilidad de adaptación a diferentes usos, situaciones y modos de trabajo. Este módulo debe permitir una comunicación transparente, para conseguir una mejor adaptación.

El proyecto consiste en lo siguiente: Una FPGA, denominada origen, realiza una serie de operaciones (en el caso de este proyecto realiza una toma de humedad y temperatura); una vez realizadas, se procede al paso de información hacia otro dispositivo, denominado destino, el cual trata la información, mostrándola de una manera adecuada.

En el caso de modificar los parámetros del módulo de Bluetooth utilizado, hay que basarse en el uso de los comandos AT. Para ello se pone como ejemplo el uso de diferentes comandos y se realiza un estudio de los resultados obtenidos.

Por último, se realizan una serie de aplicaciones para el sistema operativo Android, donde se visualizan los resultados obtenidos del tratamiento de información.

La organización del proyecto es la siguiente:

En este capítulo se plantean los objetivos. Con estos objetivos realizamos un estudio de los componentes, seleccionando los dispositivos necesarios. Posteriormente se desarrolla el proyecto de una manera incremental. Este proceso consiste en realizar un estudio sobre la comunicación y establecer un ejemplo de comunicación

básica entre dos FPGA como continuación a una adecuación de los materiales a utilizar. Como siguiente paso se realiza el paso de información relativo a la estación meteorológica, con una posible modificación de los parámetros de la comunicación. Como fin del proceso se diseñan diferentes aplicaciones móviles para la visualización del estado de la estación en dispositivos Android.

Para finalizar, se realiza un estudio de los recursos empleados por la FPGA, un estudio económico global y se destacan una serie de conclusiones y líneas futuras de desarrollo. En los Anexos podemos encontrar información útil sobre los códigos en VHDL desarrollados, sobre la estación meteorológica (relación con el TFG "Estación Meteorológica Basada en FPGA"), manuales de la utilización de los dispositivos y de las aplicaciones móviles desarrolladas y Datasheet de los dispositivos utilizados que, debido a su extensión, no se han incluido completos.

Capítulo 2

Selección y análisis de los componentes

En este capítulo se realiza una selección de los componentes que se van a emplear.

Para el desarrollo de este proyecto se plantean una serie de dispositivos necesarios, entre los que se encuentran un módulo de Bluetooth (para la comunicación entre diferentes dispositivos), un sensor de temperatura y humedad y varias FPGAs. Estos elementos serán utilizados como pilares básicos para la realización de todos los procesos. Sin embargo utilizaremos una serie de dispositivos sin los cuales resultaría imposible conseguir los objetivos propuestos, los cuales explicaremos más adelante.

Se buscarán los dispositivos más adecuados en cada caso, tratando de encontrar el equilibrio entre funcionalidad, facilidad de utilización y precio.

A lo largo de este apartado del proyecto se tratará de dar solución a la búsqueda de diferentes dispositivos, entre los que se encuentran los anteriormente descritos y los elementos auxiliares, como puede ser el cableado, los analizadores lógicos, los conectores, etc.

En primer lugar realizaremos un estudio del mercado, en donde explicaremos los principales componentes posibles, destacando ventajas e inconvenientes. Posteriormente se seleccionarán los más adecuados, concluyendo con una explicación detallada de dichos componentes.

1. FPGA

Una FPGA es un dispositivo programable que realiza la lógica en base a memorias. Esto supone una gran ventaja frente a las PLDs cuando se requiere una gran velocidad de procesamiento en los casos que realizamos gran cantidad de funciones sencillas (sumas de productos con pocas variables). Una FPGA consiste en un circuito integrado, que en ocasiones se sitúa en una placa de circuito impreso con una serie de componentes auxiliares que facilitarán el acceso a los pines de la FPGA, alimentarán al dispositivo, generarán relojes/osciladores, etc.

Entre las aplicaciones a destacar donde más se utilizan las FPGA destacamos las siguientes: procesamiento digital de señales (DSP), radio definido por software, sistemas aeroespaciales y de defensa, sistema de imágenes para medicina, diseño de procesadores, etc.

Podemos encontrar diferentes fabricantes que ofrecen dispositivos FPGA con diferentes características. Entre los principales, encontramos los siguientes:

- Xilinx: Uno de los grandes líderes en la fabricación de este ámbito, en concreto es la compañía de tecnología que creó las FPGA. Posee las mayores familias de productos de FPGA, incluyendo dispositivos de un enorme rango de precios y capacidades, dependiendo de su aplicación final (automovilismo, defensa, aeroespacial, etc.).
- Altera: Desarrolla una plataforma de supercomputación con CPUs con FPGA integradas.
- Lattice Semiconductor: Es el líder en tecnología no volátil, es decir, no pierde su configuración en caso de fallo energético. Ofrece dispositivos programables basados en RAM combinados con memoria no volátil reprogramable.
- Microsemi: Desarrolla FPGA basadas en tecnología Flash reprogramable.

Por supuesto, hay más fabricantes que desarrollan dispositivos con características similares. La elección de que FPGA es la más adecuada es complicada. Sin embargo, el Departamento de Tecnología Electrónica de la Universidad de Valladolid, y en concreto el tutor encargado de este proyecto, dispone de diferentes dispositivos de la familia Lattice Semiconductor.

Entre las FPGA disponibles se destacan la MachXO2-1200ZE y la MachXO2-7000HE (versión superior a la 1200ZE), las cuales incluyen una serie de componentes adicionales en una placa desarrollada por el fabricante, que nos permiten realizar conexiones, toma de relojes, etc.

En un primer momento se plantea la utilización únicamente del dispositivo MachXO2-1200ZE, sin embargo, debido a la demanda posterior de capacidad de cálculo, será necesario el uso de la MachXO2-7000HE. Las características de ambas FPGA son muy similares, con alguna pequeña modificación y mejora en la versión posterior. Explicaremos de una manera extensa y detallada las características de la MachXO2-1200ZE..

La familia MachXO2 ofrece a los diseñadores de PLD de baja densidad, una mezcla sin precedentes de integración alta del sistema, bajo consumo y bajo costo en un solo dispositivo. Aporta los beneficios de una integración mayor del sistema, una solidez del sistema mejorado y un menor consumo de electricidad estática.

1.1. MachXO2-1200ZE.

La placa que utilizaremos para este trabajo será la denominada MachXO2-1200ZE Breakout Board, de la familia Lattice Semiconductor.

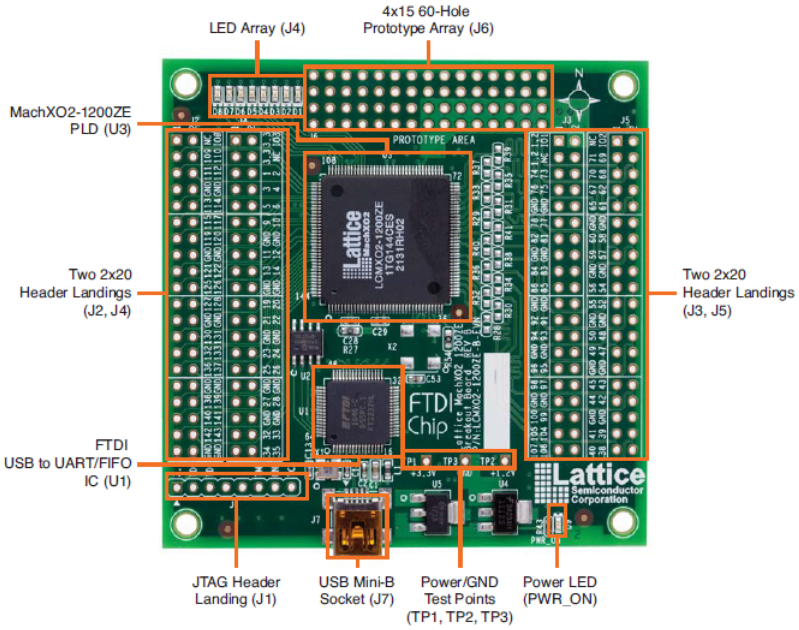


FIGURA 1.1. MachXO2-1200ZE

Esta placa contiene los siguiente componentes y circuitos (los cuales los podemos observar en la Figura 1.1).

- MachXO2-1200ZE PLD (ZE significa Ultra bajo consumo).
- Conector de corriente y programación USB mini-B.
- 8 leds.
- 60 puertos en el área de prototipo.
- 4 secciones de pines generales para I/O, JTAG* y fuente de alimentación (x20 pines por sección).
- Sección de pines para testeo JTAG (x8 pines)
- Carriles de alimentación de 3.3V y 1.2V.

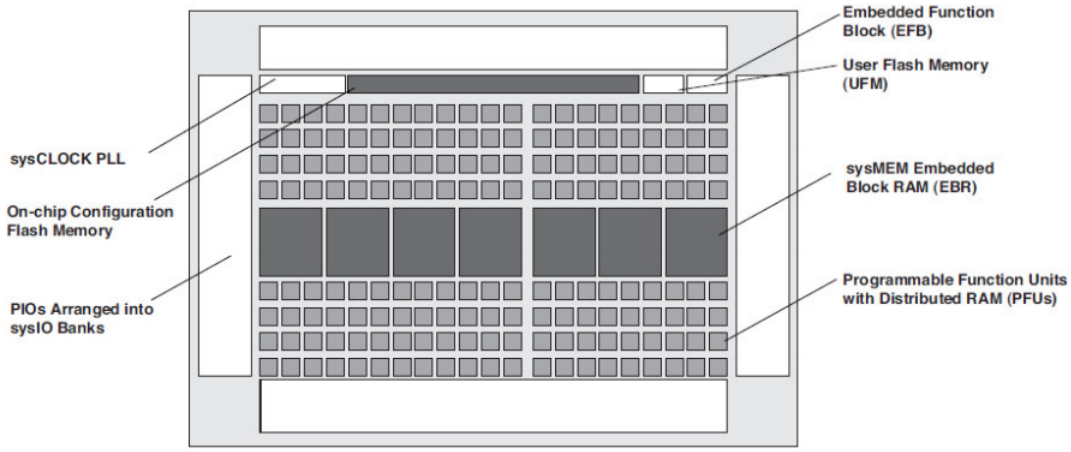


FIGURA 1.2. Arquitectura de la familia MachX02

Para poder explicar los componentes que tiene la placa que utilizaremos en este proyecto de una manera más visual y detallada disponemos del esquema mostrado en la Figura 1.2, donde se muestran los componentes del circuito integrado de la FPGA únicamente, sin incluir el resto de componentes situados en la placa.

Los componentes mostrados son los siguientes:

- Bloque PLL único (La MachXO2-7000HE dispondrá de dos bloques): Este bloque es un sintetizador de frecuencia, es decir, obtiene frecuencias de oscilación a partir de otras. Normalmente se tomará como frecuencia de referencia una de las dos aportadas por los osciladores presentes en la placa, estos dos osciladores/relojes son los siguientes:
  - Reloj interno de 2.08 MHz: Se encuentra dentro del propio circuito integrado FPGA. Será un oscilador poco preciso de manera que nos será útil cuando no sea necesaria una precisión elevada. Este valor está establecido por defecto, pudiendo variar dicho valor.
  - Reloj externo de 50MHz: Funcionará como un oscilador/reloj mucho más exacto que el interno, situado en la misma placa y conectado a la propia FPGA a partir de dos pines, uno de entrada al oscilador que funcionará como entrada de activación (pin 32 de la FPGA), y otra saliente del oscilador que será la propia señal del oscilador de 50 MHz (situada en el pin 27 de la FPGA).

- Memoria Flash* de configuración: Almacenará la última descripción funcional del dispositivo cargada. Puesto que es una memoria Flash (memoria no volátil), la configuración del dispositivo permanecerá intacta a pesar de que desconectemos la placa de la alimentación. Esta memoria es interna al dispositivo, lo que se traduce en una mayor velocidad y en una mayor seguridad frente a copias.
- Puertos I/O: Serán puertos de entradas/salidas que se encuentran organizados en bancos, los cuales pueden tener diferentes tensiones de alimentación.
- Lógica de propósito general: Se organizan en PFUs* (las cuales se organizan en Slices*, las que se organizan en Luts4*), las cuales se utilizan para la generación de lógica o memorias (mediante la producción de memoria RAM distribuida, no dedicada).
- Bloques de funciones preconstruidas: Estos bloques serán los siguientes: I2C (primario y secundario), SPI y Timer/Counter. Para la conexión entre estos bloques de funciones preconstruidas y la lógica de propósito general, se dispone de un bus estándar llamado Wishbone Bus*, que funcionará como interface entre ambos. Actuando la lógica de propósito general como maestro de la comunicación y siendo los bloques prefabricados los esclavos. Se puede observar la comunicación entre los bloques preconstruidos y la lógica de propósito general de una manera más visual en la Figura 1.3.

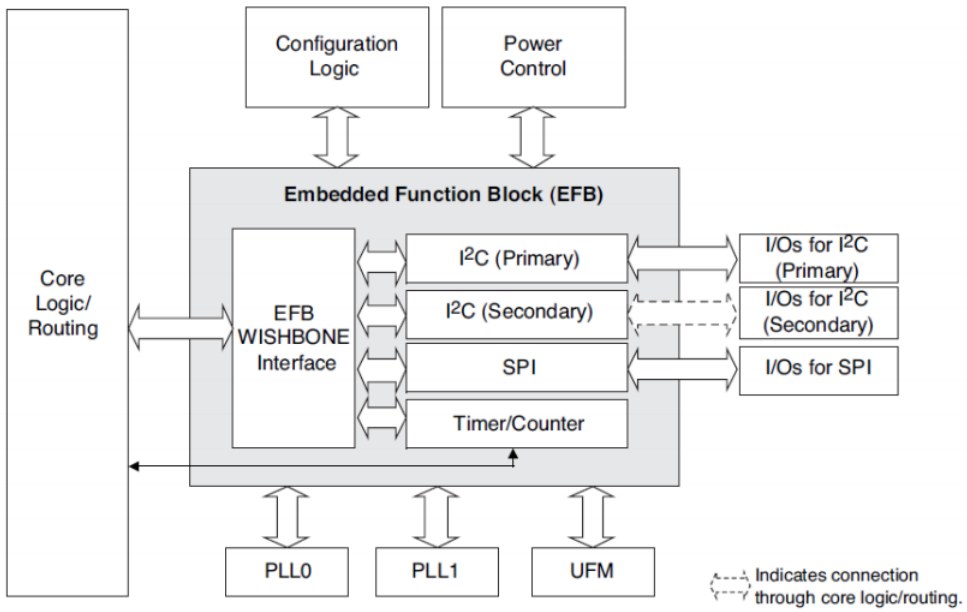


FIGURA 1.3. Interface de los bloques Hardware incorporados

- Memoria Flash de Usuario: Al igual que la memoria de configuración será una memoria no volátil que funcionará como memoria de propósito general.
- Memoria RAM* dedicada o embebida: Serán bloques de memoria RAM ya desarrollados por el fabricante y estarán únicamente destinados a funcionar como memoria RAM.

Es decir, tenemos dos posibilidades de memoria RAM, una de ellas será dedicada (viene incorporada en la propia FPGA), mientras que la otra será distribuida (será necesario su desarrollo a través de las PFUs).

Desde el punto de vista de la estructura, nuestro dispositivo va a ser una FPGA, pero desde el punto de vista de la configuración (programación o descripción del comportamiento) nuestro dispositivo funcionará como un dispositivo no volátil (debido a la memoria Flash de configuración).

Una característica a tener en cuenta del dispositivo a utilizar en este proyecto es la existencia del JTAG o TAP (Test Access Port). El TAP está formado por 4 pines (TDI, TDO, TMS y TCK) y por una circuitería interna con el objetivo del testeo de circuitos integrados.

Para terminar con la parte Hardware de nuestra placa MachXO2 Breakout Board, cabe destacar que se encuentran incorporadas dos placas de circuito impreso PCBs* con los siguientes componentes (véase Figura 1.4):

- 8 Pulsadores.
- 8 Interruptores o Microswitch.
- 2 Displays 7 segmentos con sus correspondientes resistencias limitadoras de corriente.
- 6 conexiones a pines: Se encuentras situadas en la misma PCB que los displays y nos servirán para poder conectar diferentes módulos al dispositivo.

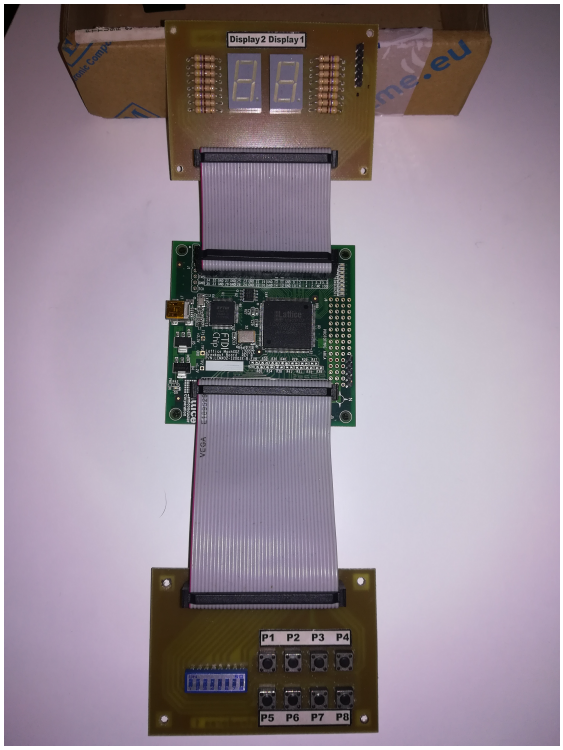


FIGURA 1.4. Placa MachX02 Breakout Board con las PCBs incorporadas

1.2. Software utilizado.

Desde el punto de vista Software, el único requerimiento que necesitaremos para describir el comportamiento deseado de nuestro dispositivo será el programa Lattice Diamond, el cual es aportado por el fabricante de la placa MachXO2. Sin embargo, en los casos en los que necesitemos desarrollar una máquina de estados finitos utilizaremos el programa Qfsm, el cual nos permite exportar dichas

máquinas con su consiguiente creación de código en diferentes lenguajes, entre los que se encuentra VHDL..

Lattice Diamond tiene una serie de características que lo hacen muy útil, como por ejemplo: Funciona como un entorno de programación (para el lenguaje VHDL), permite crear bloques por cada uno de los códigos realizados lo que genera una modularidad de los circuitos generados, permite jerarquizar bloques (unos pertenecientes a otros), etc.

2. Módulo de Bluetooth

Para realizar una comunicación entre diferentes dispositivos FPGA, será necesario disponer de módulos de Bluetooth. Estos módulos normalmente incluyen un microcontrolador integrado, lo que resulta en una mayor facilidad de utilización y en la posibilidad de un modo de transmisión transparente en algunos casos.

Se ha realizado un estudio del mercado, encontrando diferentes elementos que pueden ser útiles a la hora de desarrollar el presente proyecto. Podemos destacar como principales módulos encontrados:

- Módulo RN-42: Este elemento se puede conectar a un dispositivo existente (como una FPGA), con una mayor facilidad de integrar en un sistema embebido* (principalmente está destinado a este propósito). Como ventajas podemos destacar que la tensión utilizada se ajusta a las especificaciones de la FPGA seleccionada, las dimensiones son muy pequeñas y el consumo en reposo es muy bajo. En contrapartida podemos destacar su escaso rango de transmisión, el cual es de 20 metros únicamente, y un elevado precio en relación con los objetivos de este proyecto (varía entre 10 y 30€ dependiendo del vendedor). De esta manera hemos descartado este dispositivo para el presente proyecto.
- Bluetooth Low Energy 4.0: Utiliza una banda de frecuencia ISM 2.4GHz (compatible con el teléfono móvil). Nos permite una comunicación maestro/esclavo con una transmisión de datos transparente. Dispone de un control de parámetros gracias a los comandos AT con 40 canales de comunicación y es compatible con nuestra FPGA. Las desventajas de este producto se basan en un bajo rango de transmisión (cerca de 70 metros) y en un precio superior al resto de dispositivos encontrados (12.50€ aproximadamente). Debido a sus desventajas, no utilizaremos dicho dispositivo.

- Módulo Bluetooth HC-05: Utiliza una banda de frecuencia ISM 2.4GHz (compatible con el teléfono móvil). Permite dos modos de funcionamiento, el modo transparente y el modo de utilización de comandos AT. Funciona como maestro/esclavo y se destaca por su reducido precio y gran funcionalidad.
- Módulo Bluetooth HC-12: Utiliza un rango de frecuencia de entre 433.4 a 473MHz (muy inferior a las utilizadas por el teléfono móvil). Esto supone una desventaja a la hora de realizar una aplicación móvil para este dispositivo, sin embargo, su reducida frecuencia nos permite un mayor alcance de transmisión (1km en abierto). Dispone de las características del módulo HC-05 con una mejora de las mismas. Su precio es reducido y su utilidad elevada.
- Módulo Bluetooth HC-06: Utiliza una banda de frecuencia ISM 2.4GHz (compatible con el teléfono móvil). Dispone únicamente de 4 pines y solo opera de modo esclavo, por lo que no será utilizando en este proyecto. Las prestaciones son inferiores a las del módulo HC-05.

Planteándonos los dispositivos anteriores, nos hemos decantado por la utilización de los módulos HC-05 y HC-12. El primero de ellos será utilizado a la hora de realizar una aplicación móvil, puesto que ambos utilizan un mismo rango de frecuencias, garantizando la compatibilidad. El segundo será utilizado para la comunicación entre diferentes FPGAs, las cuales no necesariamente tienen que estar próximas, permitiendo, gracias a su elevado alcance de transmisión, una mayor versatilidad.

2.1. Módulo HC-12 .

El dispositivo HC-12, llamado HC-12 Wireless Serial Port Comunication Module, es un módulo de transmisión de datos embebido (empotrado), inalámbrico y multicanal de nueva generación, cuyas dimensiones son de 27.8x14.4x4 mm. Posee un microcontrolador dentro del módulo, de manera que el usuario no necesita programar el módulo de manera independiente, consistiendo toda la transmisión en recibir y mandar datos por puerto serie.

El módulo HC-12 será el encargado de generar el canal de comunicación entre los dos dispositivos que deseemos conectar. En el caso de que queramos comunicar dos FPGAs serán necesarios dos módulos HC-12, uno de los cuales se comportará como emisor de la información y otro como receptor.

Entrando un poco más al detalle en el módulo de bluetooth, el HC-12 podrá tener gran cantidad de aplicaciones, entre las que se encuentran todo tipo de aplicaciones inalámbricas que se nos ocurran, desde el control inalámbrico de robots hasta el control remoto industrial y telemedida.

Entre las **características** más importantes que podemos definir de este módulo destacan las siguientes:

- Transmisión inalámbrica a larga distancia (1km en espacio abierto).
- Trabaja en un rango de frecuencia de entre 433.4 hasta 473MHz (Distinguiendo más de 100 canales de comunicación), trabaja con diferentes potencias de transmisión, siendo la máxima de 20dBm y trabaja en diferentes modos de trabajo (3).
- Permite la comunicación con dispositivos externos a través de un puerto serie.
- Gestión de información en paquetes.

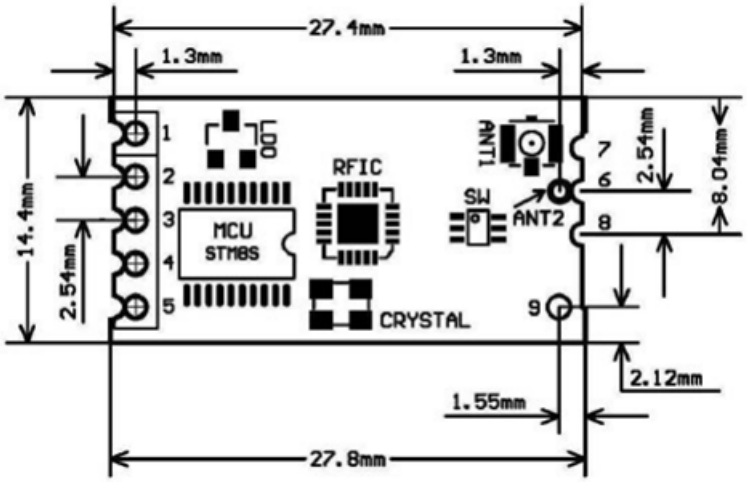


FIGURA 2.1. Dimensiones y pines del HC-12

El módulo tendrá la apariencia mostrada en la Figura 2.1:

La separación de los pines será de 2.54mm, idóneo para poder realizar una PCB fácilmente. Como podemos observar en la Figura 2.1, el módulo tiene 9 pines (uno no conectado), siendo los cuales:

- Pin 1 (Vcc): Tensión de entrada en continua de 3.2-5.5V.
- Pines 2, 7 y 8 (GND).

- Pin 3 (RXD): Puerto de entrada UART, con débil pull-up.
- Pin 4 (TXD): Puerto de salida UART.
- Pin 5 (SET): Control de ajuste de parámetros, válido para nivel bajo.
- Pin 6 (ANT): Pin de antena de 433MHz.

Cabe destacar que los pines 3, 4 y 5 poseen una resistencia de 1k interna y que es recomendable utilizar un diodo conectado en serie con el pin 1 cuando el módulo está trabajando durante un largo periodo de tiempo y bajo una tensión de entrada superior a 4.5V, en nuestro caso trabajaremos con una tensión de 3.3V por lo que no será necesario.

El **funcionamiento** será el siguiente: Un dispositivo envía datos por el puerto serie, conectado al puerto de entrada RXD del módulo, que recibe dichos datos, de esta manera se envían los datos vía aire automáticamente, cuyo destinatario será otro dispositivo. El módulo HC-12 utiliza un modo de transmisión de datos trasparente, teniendo 3 diferentes tipos de trasmisión (FU1, FU2 y FU3), los cuales describiremos más adelante.

Un factor a tener en cuenta es la posible pérdida de información a la hora de la transmisión de datos, es decir, considerando las interferencias del ambiente y otros factores, algunos bytes de información se pueden perder.

Por defecto, la transmisión transparente está establecida para trabajar en modo **FU3** (estado de máxima velocidad). En este modo el consumo de corriente será de 16mA. Otra característica a destacar es que el módulo puede ajustar automáticamente la velocidad en baudios de la transmisión inalámbrica de acuerdo a la velocidad en baudios del puerto serie, según una tabla aportada por el fabricante.

Cuanto mayor sea la distancia de comunicación, la velocidad en baudios del puerto serie deberá ser menor y viceversa.

El modo **FU1** es un modo de ahorro de energía relativo, siendo el consumo de corriente de 3.6mA. En este caso podemos variar la velocidad en baudios del puerto serie (8 posibilidades), pero no la velocidad de transmisión vía aire, que será constante e igual a 250.000bps.

El modo **FU2** es un modo de ahorro de energía, siendo el consumo de corriente de 80uA. Podremos variar la velocidad en baudios entre 3 posibilidades, pero la velocidad de transmisión vía aire será la misma que en el modo FU1.

El **comando AT*** se usa para configurar los parámetros del módulo y elegir sus funciones. Cabe destacar que una vez configurado el módulo, los parámetros permanecerán constantes incluso en caso de fallo energético.

Para poder configurar el dispositivo deberemos entrar en el comando AT, para ello pondremos el Pin 5 (SET) a nivel bajo y posteriormente introduciendo instrucciones del comando, las cuales nos servirán para testear el funcionamiento del módulo, cambiar la velocidad en baudios de trasmisión por el puerto serie, cambiar el canal de comunicación inalámbrico (128 posibilidades de frecuencia), cambiar el modo de transmisión (FU1, FU2 o FU3), configurar la energía transmitida (para asegurar la comunicación para diferentes distancias de comunicación) y obtención de información de los parámetros y del módulo. Todos los diferentes comandos están descritos de una manera más explícita en el datasheet del fabricante.

En este proyecto vamos a trabajar con el modo de funcionamiento FU3, el cual trabaja con paquetes de datos de un carácter, es decir, un byte. La línea de emisión se encuentra permanentemente a nivel alto (debido a la resistencia pull-up) hasta que se produce un paso de información en forma de paquete de datos. Este paquete tendrá un bit de arranque a nivel bajo, seguido del byte de datos. Posteriormente la línea volverá a su estado natural (nivel alto), véase la Figura 2.2.

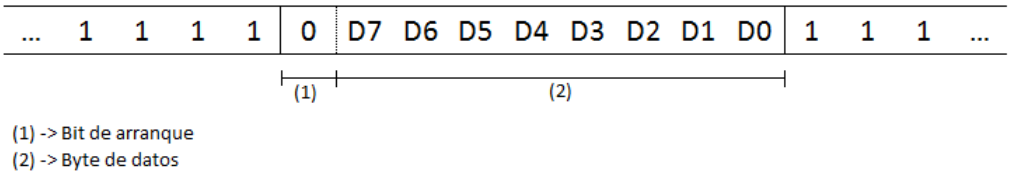


FIGURA 2.2. Paquete de datos del HC-12

De esta manera debemos adaptar nuestro diseño al formato del paquete establecido por el módulo HC-12.

2.2. Módulo HC-05.

El módulo HC-05 dispone de una fácil utilización del Bluetooth SPP. Para ello dispone de un modo de funcionamiento transparente y un modo de configuración de parámetros (Comandos AT). Como características principales podemos destacar:

- Utiliza una frecuencia de emisión de 2.4GHz (al igual que el teléfono móvil), permitiendo una correcta comunicación con gran cantidad de dispositivos.
- Tensión de alimentación entre 1.8 y 3.6V.
- Velocidad de transmisión por defecto de 38.4kHz, con posibilidad de ser configurado con otras frecuencias (véase datasheet del fabricante).
- Emisión de información en paquete de datos al igual que el módulo HC-12 (Figura 2.2).

En la Figura 2.3 podemos observar sus dimensiones y alguno de sus principales pines.

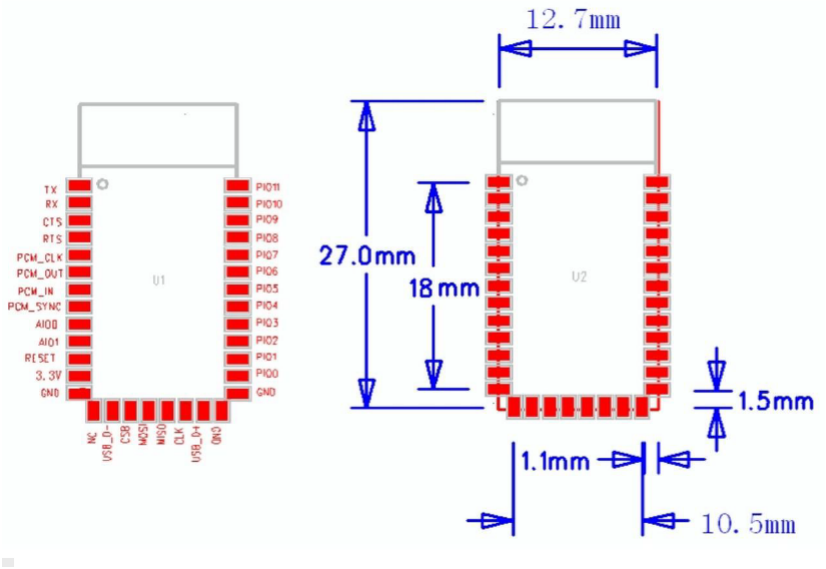


FIGURA 2.3. Dimensiones del HC-05

A la hora de adquirir el módulo HC-05, hemos obtenido un módulo con la apariencia y características de la Figura .

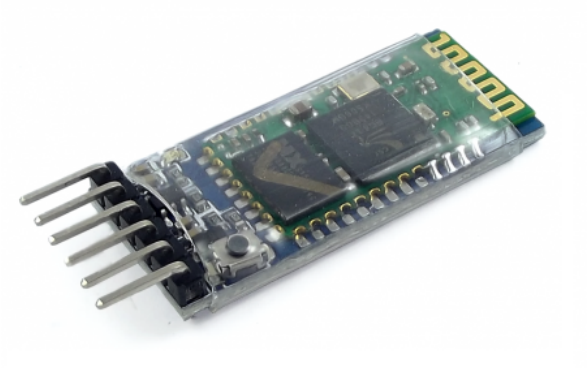


FIGURA 2.4. Módulo HC-05

Donde se pueden observar los 6 pines disponibles (GND, VCC, RXD, TXD, EN, y STATE), siendo los cinco primeros los equivalentes al módulo HC-12.

3. Sensor de Temperatura y Humedad

En el caso del sensor de temperatura y humedad se plantean tres posibilidades:

- Módulo SHT21: Es un sensor de humedad y temperatura de Sensirion. Es un sensor poco conocido y se carece de información abundante y fiable sobre el mismo.
- DHT11: Sensor de humedad/temperatura de bajo costo y de media precisión.
- DHT22: Sensor de humedad/temperatura de bajo costo y de alta precisión.

Todos ellos proporciona señales calibradas y linealizadas. Sin embargo, el rango y precisión de temperaturas y humedades del sensor DHT22 es superior al resto, por lo que será dicho módulo el elegido para este sistema.

3.1. Sensor DHT22.

El sensor DHT22* es un sensor de temperatura y humedad de bajo costo con una interfaz digital. El sensor está calibrado y no requiere de componentes adicionales por lo que basta con conectarlo para comenzar a tomar mediciones de humedad relativa y temperatura. Este aspecto nos será muy útil y nos servirá para reducir la dificultad de la toma de temperatura y humedad.

El sensor DHT22 utiliza un protocolo de comunicación serie propio que ocupa sólo una conexión en uno de sus pines. Las principales características del módulo son las siguientes:

- Alimentación entre 3.3 y 5V.
- Tiene una precisión de decimales.
- Tiempo de muestreo de 2 segundos.
- Rango de valores desde -40ºC hasta 80ºC de temperatura y desde 0 % hasta 99.9 % de humedad relativa.

Como se observa en la Figura 3.1, dicho sensor tiene 4 pines, los cuales son (de izquierda a derecha):

1. VDD: Alimentación del dispositivo.
2. DATA: Señal de datos.
3. NULL.
4. GND: Referencia de tensión, tierra.

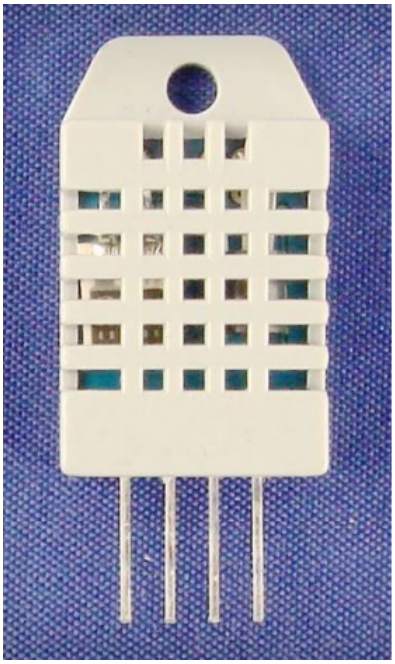


FIGURA 3.1. Sensor DHT22

Su funcionamiento se basa en un sensor capacitivo de humedad y un termistor para medir el aire circundante. Muestra los datos mediante una señal digital en el pin de datos (no hay pines analógicos).

Su utilización requiere una cuidadosa sincronización para la toma de datos. Este sensor presenta gran cantidad de ventajas (como es su precio), presentando como inconvenientes que no tiene un interface estándar (por lo que sus lecturas son algo complicadas de obtener) y que sólo se pueden obtener nuevos datos cada 2 segundos, de manera que es un sensor óptimo para aplicaciones de baja frecuencia y que requieran poca velocidad de toma de datos.

El diseño para el control y lectura de este módulo forma parte de un trabajo de fin de grado diferente, realizado por un compañero. Partiremos de esta manera con la información del dispositivo ya codificada y con un formato conocido, el cual se muestra en la Figura 3.2.

La información saliente del módulo tiene un formato peculiar y tendrá los siguientes componentes:

- 1 byte que expresa la parte más significativa de la humedad.
- 1 byte que expresa la parte menos significativa de la humedad.
- 1 byte que expresa la parte más significativa de la temperatura.
- 1 byte que expresa la parte menos significativa de la temperatura.
- 1 byte de redundancia (suma de presión y temperatura) para comprobar errores.

8 bits	8 bits	8 bits	8 bits	8 bits
Parte alta de humedad	Parte baja de humedad	Parte alta de temperatura	Parte baja de temperatura	Redundancia

FIGURA 3.2. Formato de la información aportada por el sensor

4. Otros componentes y herramientas

Serán necesarios una serie de componentes auxiliares que nos sirvan de apoyo a lo largo del desarrollo del proyecto. Estos componentes son los siguientes:

- Cable USB Mini-B: Será el encargado de alimentar la placa MachXO2 y de cargar la configuración de la FPGA para que realice las funciones deseadas.
- Analizador lógico de 24MHz y 8 canales. Será el encargado de realizar el estudio de determinadas señales (hasta 8 simultáneamente), para lo cual se utilizará el programa Logic 1.2.17 a través del muestreo con una determinada frecuencia preestablecida. Para poder ser utilizado necesitaremos otros 3 componentes diferentes (los cuales se muestran en la Figura 4.1):

- Cable conector USB Mini-B.
- Cableado para la conexión de señales DuPont.
- Test Clips

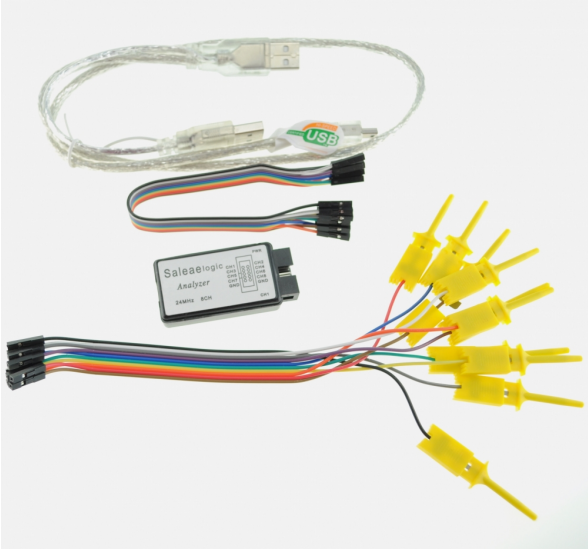


FIGURA 4.1. Componentes auxiliares

Para utilizar el dispositivo, lo conectaremos al ordenador a través del cable USB y, estableciendo las señales deseadas, observaremos los resultados a través del software "Logic 1.2.17".

Desarrollo del Proyecto

A lo largo de este proyecto se van a desarrollar diferentes actividades con el ánimo de cumplir los objetivos fijados y conseguir de manera óptima todos los requerimientos y especificaciones definidas. Se buscará en cada caso la mejor alternativa desde el punto de vista funcional y económico, buscando soluciones sofisticadas con un reducido consumo de recursos.

El desarrollo se ha estructurado de la siguiente forma: En primer lugar se desarrollan una serie de acciones sobre los componentes utilizados a lo largo del proyecto, con el objetivo de facilitar su colocación sobre la placa MachXO2 y sobre las PCBs anidadas a la misma.

En segundo lugar se diseña y se desarrolla un ejemplo que nos sirva de base para el diseño final. Este ejemplo consistirá en la emisión, desde una FPGA origen hacia una destino, de un byte de información sobre que interruptor se encuentra accionado, siguiendo las pautas establecidas por el protocolo de comunicación que utiliza el módulo de Bluetooth HC-12.

Seguidamente se realiza el diseño final de comunicación entre dos FPGAs para el traspaso de información indicativa de temperatura y humedad. Para ello se deberán enviar 40 bits de información, es decir, 5 bytes. Debido a esto se deberá desarrollar una metodología de emisión de diferentes paquetes de datos de un byte. Tanto la toma de temperatura y humedad como su representación forma parte del trabajo de fin de grado de un compañero, de tal manera que no nos centraremos en esa parte, pudiendo encontrar información útil sobre diferentes aspectos de esa parte en el Anexo B..

Con el objetivo de ampliar las posibilidades y facilitar futuros desarrollos sobre el módulo HC-12, se desarrolla una metodología para la modificación de los parámetros del mismo (velocidad de comunicación, potencia transmitida, etc.). Para ello nos basaremos en el uso de los comandos AT, los cuales vienen especificados en el datasheet del dispositivo y nos ayudarán a modificar determinados aspectos del funcionamiento del mismo para su aplicación en diferentes entornos.

Por último se diseñará una serie de aplicaciones en Android para su posterior uso en el teléfono móvil. Estas aplicaciones van a consistir en la obtención de información sobre la temperatura y humedad del ambiente donde se encuentra una FPGA origen de la información. En este caso se utilizará el módulo HC-05 puesto que la versión HC-12 no es compatible con el Bluetooth de dispositivos móviles.

Capítulo 3

Estudio de la comunicación

Para conseguir los objetivos propuestos, es fundamental la correcta comunicación entre dos dispositivos FPGA. Para ello se necesitan dos elementos básicos:

1. El propio dispositivo FPGA: Permite realizar una descripción del comportamiento del circuito a desarrollar para establecer la comunicación.
2. El módulo Bluetooth HC-12: Es el encargado de, recibiendo una serie de datos por uno de sus pines, enviar la información vía Bluetooth hacia un módulo par conectado a un dispositivo FPGA destino.

La comunicación entre dos dispositivos FPGA se va a basar en el esquema de la Figura 3.0.1.

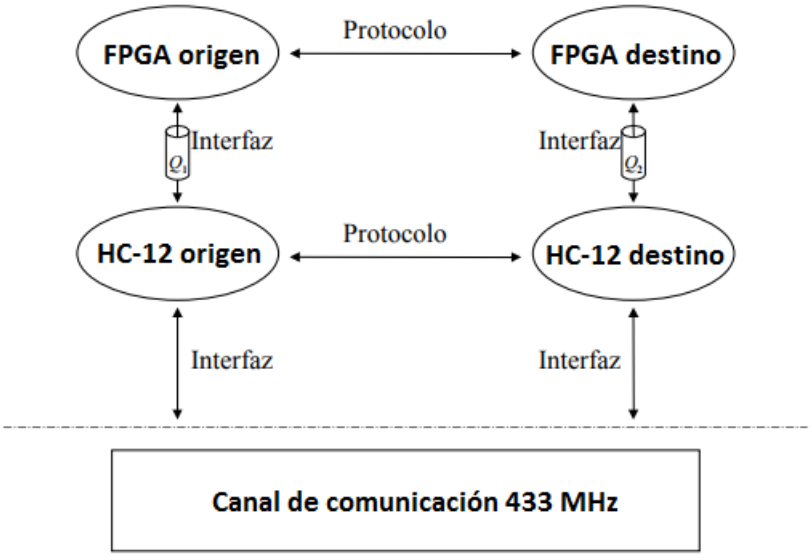


FIGURA 3.0.1. Comunicación entre FPGA

En dicha Figura se observan las dos FPGAs a conectar, una funcionando como origen y otra como destino de la información. Para establecer una correcta

comunicación es necesario que cada una de las FPGA se encuentren conectadas a un módulo HC-12. Una vez realizada una correcta descripción del comportamiento del circuito a desarrollar (lo realizaremos posteriormente) se procede a la emisión de información.

El flujo de información es el siguiente: Una FPGA origen envía un mensaje SDU* hacia su correspondiente módulo HC-12 a través de un interfaz* definido. El módulo emite la información, es decir, un mensaje PDU*, a través de la frecuencia 433MHz de la banda ISM*. El módulo destino recibirá la información, que la transmitirá hacia la FPGA destino, con su posterior tratamiento de la información.

El interfaz entre la FPGA y el módulo HC-12 estará definido por las características de configuración del módulo y las especificaciones de transmisión de información entre ambos, es decir, el módulo requiere de una transmisión de información serie de los datos, gestionando dichos datos en paquetes. Este interfaz es muy básico debido a que el módulo HC-12 funcionará en modo transparente, emitiendo los datos recibidos sin ninguna modificación. De esta manera el interfaz definirá la velocidad de transmisión de datos, la potencia transmitida, etc.

El protocolo* entre FPGAs quedará definido por la manera de enviar los datos hacia el módulo, es decir, deberán tener tanto la FPGA origen como la de destino una forma común de tratar los datos, siendo de esta manera posible una correcta comunicación. Este proceso se realizará mediante una descripción funcional compatible entre ambos dispositivos.

Es el propio módulo HC-12 el encargado de gestionar la emisión y recepción de datos a través del canal de comunicación (mediante radiofrecuencia, a 433MHz). Bastará con que los módulos origen y destino estén configurados en el mismo modo de funcionamiento (FU1, FU2 o FU3), con la misma velocidad de transmisión, con el mismo canal de comunicación inalámbrico y con la misma energía transmitida al módulo. Consiguiendo de esta manera una compatibilidad entre módulos absoluta.

Para conseguir los objetivos especificados me voy a basar en la utilización del lenguaje VHDL*. Este lenguaje nos permite realizar una descripción del comportamiento de un circuito especificado, es decir, vamos a desarrollar una descripción funcional del circuito requerido para que una FPGA sea capaz de transmitir, apoyándose en el funcionamiento del módulo HC-12, una serie de datos hacia otro dispositivo equivalente en el destino.

Capítulo 4

Configuración del Hardware

Para comenzar a realizar el proyecto se requiere la configuración de todos los componentes que vamos a utilizar a lo largo del desarrollo de cada una de las partes de dicho proyecto. Como hemos descrito en apartados anteriores necesitaremos 3 componentes: una FPGA, un módulo bluetooth HC-12 y un sensor DHT22. Cada uno de ellos requiere de una serie de modificaciones para conseguir los objetivos fijados.

4.1. Módulo bluetooth HC-12

A la hora de obtener el módulo, lo hemos recibido sin ningún componente adicional, como se observa en la Figura 4.1.1.

Figura 4.1.1. Módulo HC-12 de partida

Será imprescindible la colocación de una antena en uno de los pines especificados para ello, con el objetivo de aumentar la distancia de emisión de datos y

garantizar la correcta comunicación a corta distancia. Por esta razón, hemos decidido soldar una pequeña antena en el dispositivo, en el pin ANT2 (pin destinado a la soldadura de la antena).

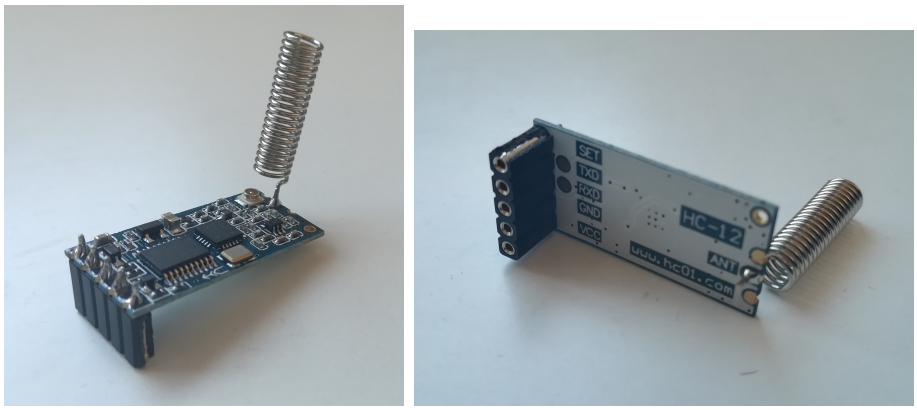


FIGURA 4.1.2. Módulo HC-12 modificado

Será también necesaria la colocación de zócalos SIP (pines) en los 5 pines principales (SET, TXD, RXD, GND y VCC) para poder acoplar el dispositivo a la placa de la FPGA de una manera óptima. Para ello soldaremos los pines, que serán macho/hembra. Estas modificaciones se pueden observar en la Figura 4.1.2, obteniendo como resultado final el mostrado en dicha imagen.

4.2. Módulo DHT22

Partiendo del módulo cuyo funcionamiento se encuentra explicado en el apartado 3.1, se requiere una serie de acciones para facilitar el acoplamiento y conexión de dicho sensor a la placa de la FPGA..

Recordemos que el sensor DHT22 dispone de 4 pines, de esta manera, se ha utilizado un adaptador de pines, de tal forma que obtenemos un acceso fácil a los puertos de alimentación (Vdd y Gnd) y puerto de datos. Esta adaptación se puede observar en la Figura 4.2.1, donde se consigue un acceso sencillo sobre los pines que vamos a utilizar.

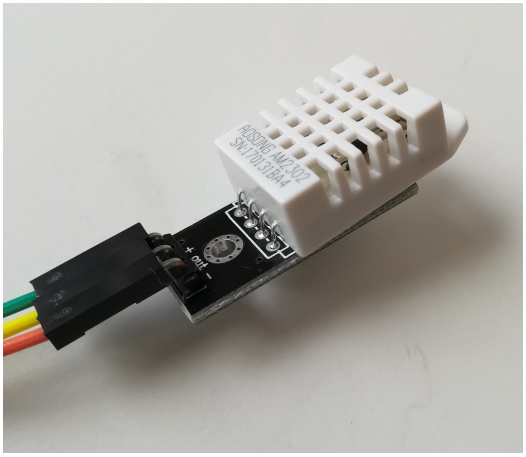


FIGURA 4.2.1. Adaptador del sensor DHT22

Partiendo del acoplamiento realizado (realizado mediante soldadura), podemos separar con una determinada distancia nuestro sensor de nuestra placa FPGA, gracias a una serie de cables conectados a los pines del dispositivo.

En el otro extremo de los cables conectados al módulo, disponemos de tres zócalos SIP hembra (uno por cable) que nos permitirá conectar el dispositivo a la placa FPGA..

4.3. FPGA

Las modificaciones realizadas sobre este dispositivo nos permitirán no solo el establecimiento de la conexión entre la placa FPGA y los módulos, sino también la adición de determinados pulsadores, interruptores, acceso a pines y displays de 7 segmentos.

Esto último no debe preocuparnos, debido a las placas MachXO2 Breakout Board disponibles ya incorporan dos PCBs con sus correspondientes elementos (descritos en el apartado de componentes 1.1).

Es decir, no se requerirá ninguna modificación en la placa MachXO2 para poder conectar correctamente el módulo DHT22, puesto que en una de las PCBs anidadas a la placa hay disponibles 6 pines conectados directamente a pines de la FPGA. Estos pines serán utilizados para la conexión del sensor de temperatura y humedad.

Para poder conectar el módulo de Bluetooth a la placa serán necesarias una serie de acciones. El módulo tiene 5 puertos principales (SET, TXD, RXD, GND y VCC), los cuales se ha decidido colocar en determinados puertos del área de prototipo. Para ello será necesario conectar manualmente los puertos y los pines destinados a alimentación y los puertos y los pines de la FPGA. Esta conexión se ha realizado mediante soldadura y se puede observar en la Figura 4.3.1:

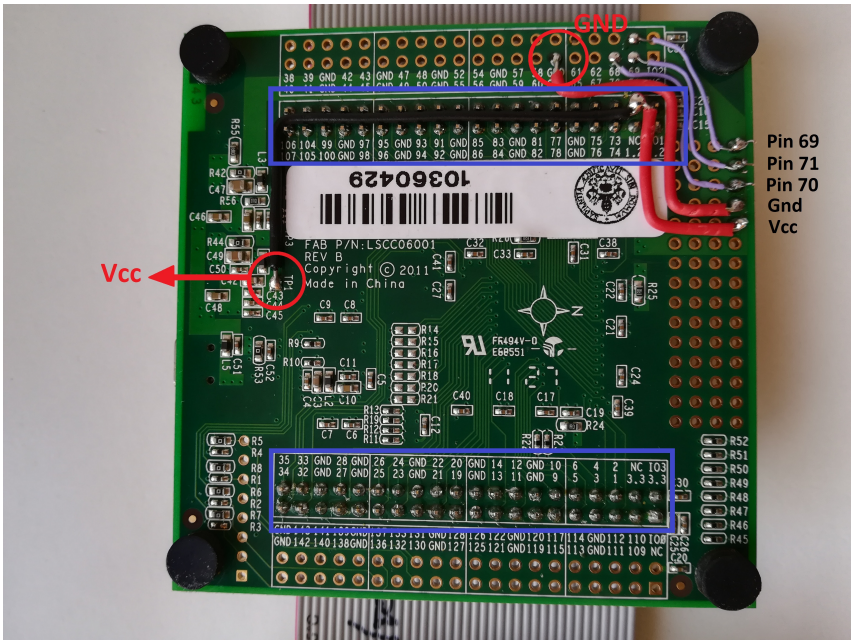


FIGURA 4.3.1. Conexiones realizadas en la placa MachXO2

Observamos en la Figura 4.3.1 dos recuadros azules, los cuales indican la conexión entre la placa MachXO2 y las PCBs. Se muestra también que hemos conectado la tensión de alimentación Vcc, de 3.3V, mediante el cable negro a uno de los pines marcados por el recuadro azul (para tener más cerca dicho pin a la zona de prototipo donde vamos a conectar el módulo HC-12).

Mediante los cables rojos hemos procedido a la conexión entre la fuente de alimentación (Vcc y Gnd) y los puertos del área prototipo en los cuales está destinada la colocación de los pines de alimentación del módulo HC-12.

Mediante los cables morados hemos realizado la conexión entre los pines 69, 70 y 71 de la FPGA a los puertos del área de prototipo en los cuales está destinada la colocación de los pines SET, RXD y TDX del módulo HC-12 respectivamente.

Simultáneamente a la conexión de los cables en los puertos del área de prototipo hemos realizado una soldadura de pines macho/macho, a los cuales se anclarán los pines propios del módulo HC-12 (hembra), véase la Figura 4.3.2.

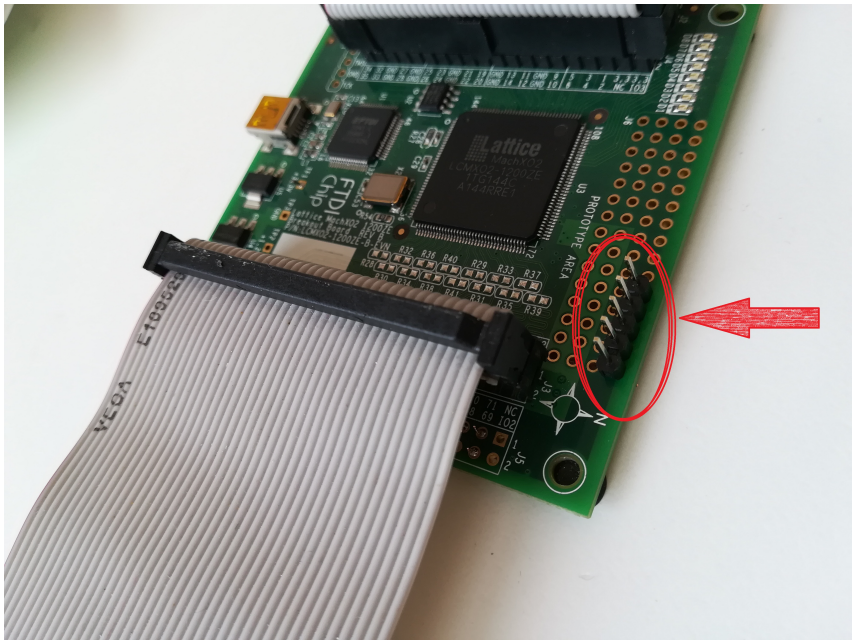


FIGURA 4.3.2. Conexión de los pines en la placa MachXO2

En resumen, la conexión entre la FPGA y el sensor de temperatura y humedad se realizará gracias a los pines situados en una de las placas de circuito impreso anidadas a la placa MachXO2, mientras que la conexión entre la FPGA y el módulo de Bluetooth se realizará gracias a unos pines colocados en el área de prototipo de la placa.

Capítulo 5

Comunicación básica

El proyecto, complejo, que se debe realizar, se ha estructurado de forma que, se ha dividido en varias secciones más sencillas que, además, permitirán establecer conclusiones sobre cuales son las soluciones más adecuada.

Recordemos que el objetivo final es la comunicación entre dos dispositivos FPGA, de tal manera que se envían 32 bits de información procedentes del sensor de temperatura y humedad (más 8 bits de redundancia que sirven para comprobar errores). Este último envía la información respectiva con un periodo de 2 segundos.

Para comenzar, se va a desarrollar un ejemplo que permita la comunicación entre dos dispositivos FPGA. Recordemos que se parte de dos placas MachXO2, a las cuales se han conectado los módulo HC-12 y el sensor de temperatura y humedad. Para este primer ejemplo solo es necesaria la conexión del módulo de Bluetooth a la placa, ya que únicamente se desea la transmisión de información entre placas.

Para realizar una comunicación entre los dos dispositivos deberemos realizar una descripción del comportamiento del circuito a desarrollar mediante el lenguaje VHDL (descripción funcional) o bien mediante la descripción estructural (descripción de la estructura del circuito). De aquí en adelante realizaremos una mezcla de ambas, de tal manera que realizaremos descripciones del comportamiento de determinados componentes del circuito y descripciones estructurales para permitir la conexión entre dichos componentes.

Como hemos explicado en el apartado 1.1, nuestra placa tiene anidado, entre otros componentes, 8 micro interruptores. En esta sección del proyecto fijamos como objetivos los siguientes apartados:

1. Detección del interruptor más prioritario. El interruptor cuyo número de identificación sea mayor en valor tendrá una mayor prioridad (P8 > P7 > ... > P1).
2. Emisión del número de identificación por una única linea para la utilización del módulo de Bluetooth.

3. Emisión de la información en paquetes, con su correspondiente recepción.
4. Transformación de la información recibida por una única linea para conseguir el número de identificación.
5. Tratamiento de la información, es decir, mostrar el número de identificación por uno de los displays.

Los interruptores tienen cada uno de ellos una codificación en binario, igual su codificación binaria menos uno, es decir, el interruptor 8 tendrá una codificación de "111". Esta codificación está vigente para todos los interruptores salvo para el número 1, el cual es un interruptor de habilitación, activo a nivel alto. La codificación de salida equivale a la codificación del interruptor de mayor prioridad. En el caso de que ninguno de los interruptores esté activo la codificación de salida vale "000", mientras que si el interruptor de habilitación se encuentra a nivel bajo, no habrá codificación de salida, y por tanto no se enviará ningún mensaje. Se pueden observar los interruptores en la Figura 5.0.1.

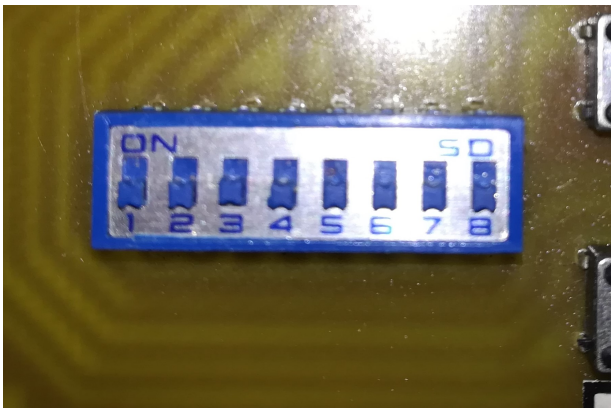


FIGURA 5.0.1. Interruptores

Para conseguir estos objetivos es necesario establecer, previamente, un protocolo de emisión de datos común entre FPGAs (como indicábamos en el capítulo 3). Este protocolo establece no solo la manera de elaborar e interpretar el mensaje, sino también un formato de mensaje.

Es decir, la línea normalmente está a nivel alto (debido a una configuración pull-up). Cuando la línea cambia a nivel bajo (bit de arranque) se indica el inicio del mensaje, el cual tiene un tamaño de 8 bits. Se decide diseñar un formato de mensaje donde se añade un bit de paridad, el cual indica una información redundante y sirve para comprobar el buen funcionamiento de la transmisión. En

caso de haber un número par de "1s" en el campo de datos del mensaje, el bit de paridad valdrá "0", mientras que si hay un número impar de "1s", el bit de paridad valdrá "1". Es decir, el formato es el mostrado en la Figura 2.2, situada la sección 2.1, con un contenido específico.

Para el desarrollo del problema propuesto en este apartado se ha decidido que la transmisión de la información se apoye en el formato de mensaje de la Figura 5.0.2. Puesto que tenemos 8 pulsadores bastaría con un campo de datos de un tamaño de 3 bits (2^3= 8), aunque el campo de datos tiene siempre un tamaño de 8 bits, debido a que el módulo de Bluetooth utiliza esta configuración.

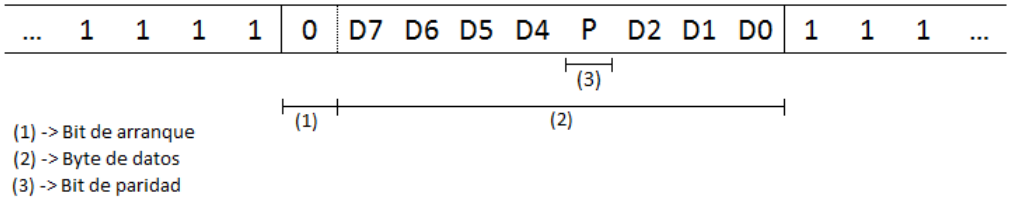


FIGURA 5.0.2. Formato del mensaje de envío de interruptor más prioritario

Como podemos observar, dentro del campo de datos se encuentran los tres bits que indican el interruptor (D2-D0), el bit de paridad, que sirve para la detección de errores y cuatro bits inutilizados (D7-D4).

Podemos observar otro detalle, el protocolo que se establece entre dispositivos FPGA es un protocolo orientado a bits, donde no se representan símbolos de ningún código. También se puede concluir que el formato del mensaje es de "Principio y Cuenta", es decir, tiene una cabecera inicial en la cual se establece el sincronismo y se indica el inicio de los datos. Puesto que el receptor conoce el protocolo, y por tanto el tamaño del campo de datos, dicho receptor será conocedor en todo momento de la finalización del mensaje.

Para el desarrollo del circuito electrónico necesario para el funcionamiento del conjunto, es necesario desarrollar un reloj de frecuencia apropiada. El módulo HC-12 viene preconfigurado con un modo de funcionamiento FU3 (véase apartado 2.1), el cual tiene una velocidad de transmisión de 9600 baudios. El cambio en el modo de funcionamiento se producirá de una manera muy sencilla desde microcontroladores como puede ser Arduino, pero se complica en dispositivos como son las FPGAs.

De esta manera, se necesita generar un **Reloj de 9600Hz**, para lo cual ha sido desarrollado el esquema de la Figura 5.0.3.

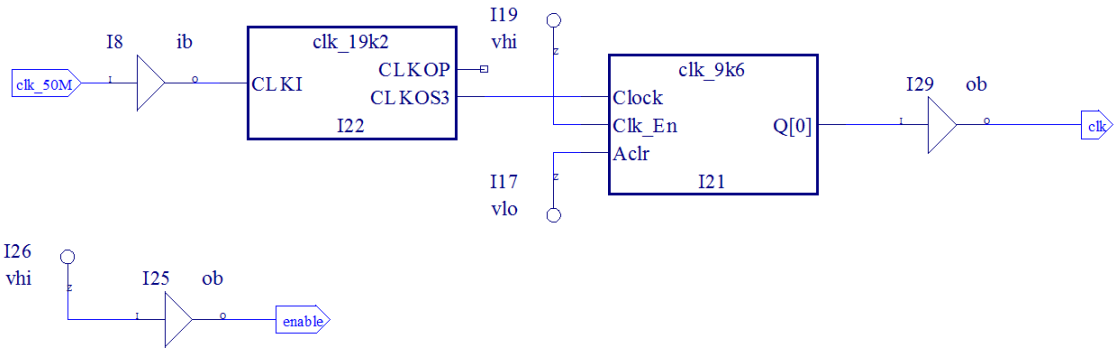


FIGURA 5.0.3. Generación del reloj de 9.6kHz

Donde se observan dos partes: En una de ellas se habilita el reloj externo mediante la salida "enable" a nivel alto, la cual va dirigida al pin correspondiente; En la otra parte se tiene como entrada el reloj externo, de 50MHz, siendo los bloques "clk_19k2" y "clk_9k6" circuitos que modifican la señal de entrada.

El primer bloque ha sido generado gracias al PLL (sintetizador de frecuencia), el cual genera, a partir de una señal de 50MHz en este caso, una señal de 19.2kHz que servirá como señal de reloj del siguiente bloque. No generaremos la señal de 9.6kHz directamente debido a que el sintetizador de frecuencia de nuestra placa no tiene tanta capacidad. El segundo bloque es un contador, que únicamente contará dos bits distintos (0 y 1), de tal manera que en cada flanco de subida de la señal de reloj (19.2kHz) se producirá una transición, consiguiendo una señal de una frecuencia 2 veces menor a la de entrada, es decir, de 9.6kHz.

Ambos bloques han sido generados gracias a la herramienta IPexpress, la cual nos permite entre otras cosas realizar estas acciones, y además viene instalada en el software utilizado (Lattice Diamond). Para comprobar su buen funcionamiento nos hemos apoyado en la utilización de un analizador lógico. Obteniendo el resultado deseado mostrado en la Figura 5.0.4.

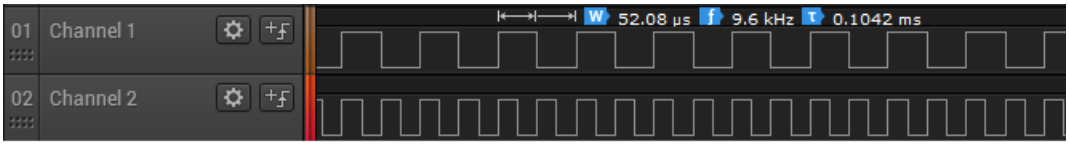


FIGURA 5.0.4. Señal de reloj

Siendo la señal del canal 1 la señal de reloj final de 9.6kHz y la señal del canal 2 la previamente generada de 19.2kHz.

Para conseguir los objetivos propuestos en este apartado, será necesaria una **Decodificación de los interruptores**, es decir, debemos desarrollar un circuito que, teniendo como entrada los interruptores, genere una salida binaria que indique el código del interruptor accionado más significativo. Para ello se ha desarrollado el bloque de la Figura 5.0.5.

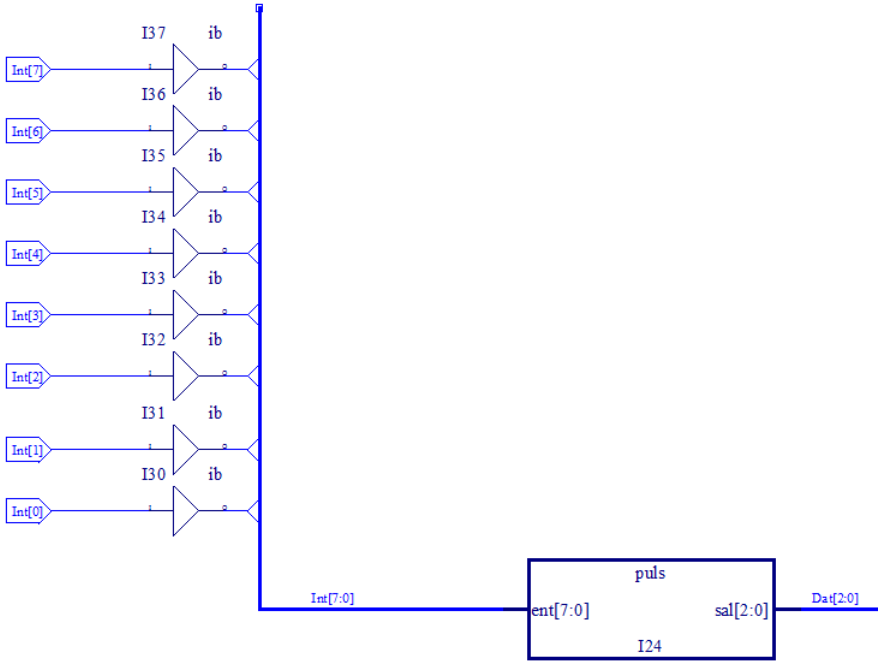


FIGURA 5.0.5. Decodificador de los interruptores

Como se puede observar en dicha imagen, tenemos como entradas los 8 interruptores de la placa (los cuales se configurarán en los pines respectivos), y como salida, un bus de 3 bits que indicará el código en binario del interruptor más prioritario.

Este circuito se ha desarrollado a partir de la descripción funcional mediante el lenguaje VHDL. Es decir, tendrá el código de la Figura A.1.1, localizada en el Anexo A.

Para comprobar el funcionamiento del bloque será necesaria una simulación del mismo. Obteniendo el resultado mostrado en la Figura 5.0.6.

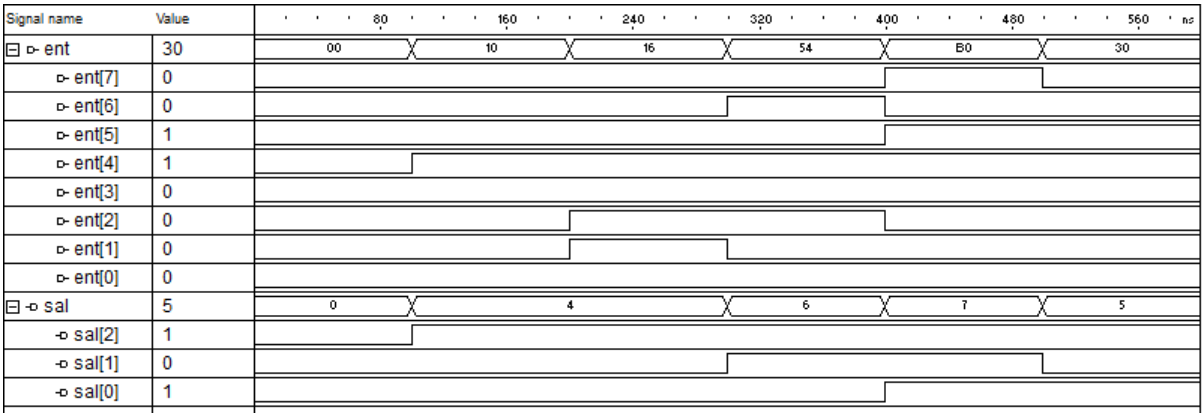


FIGURA 5.0.6. Simulación del Decodificador de Interruptores

Donde podemos observar que efectivamente se cumplen los objetivos del bloque, es decir, la salida indicará el interruptor accionado más prioritario.

Una vez que el receptor reciba el mensaje, será necesaria una conversión para poder visualizarla. Una manera para conseguirlo es mediante la utilización de uno de los displays que se encuentran en una de las placas de circuito impreso anidadas a la placa MachXO2.

Es necesario entonces el desarrollo de un **Convertidor de Binario (BCD) a código del display de 7 segmentos**. Este bloque tendrá 4 bits de entrada (por si es necesario convertir un número mayor o igual a $2^3 = 8$) y tendrá 7 salidas (una por cada led o segmento del display). Esquemáticamente, tendrá la apariencia de la Figura 5.0.7. Donde se puede observar que cada cable del bus de salida se conecta (mediante un buffer de salida) a un segmento del display (con sus correspondientes pines).

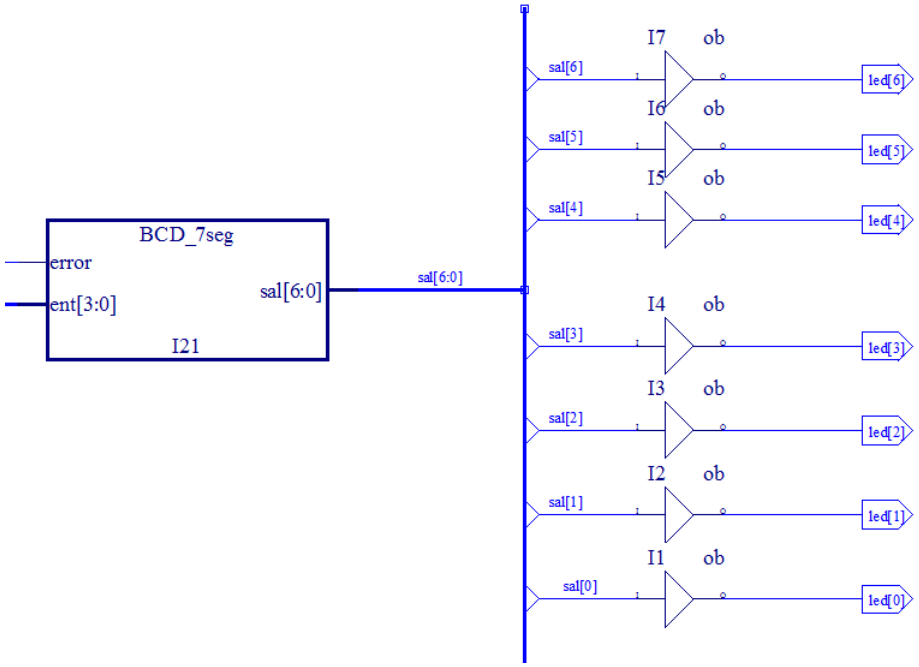


FIGURA 5.0.7. Convertidor BCD a 7 segmentos

Este circuito se ha desarrollado también a partir de la descripción funcional mediante el lenguaje VHDL. Es decir, tendrá el código de la Figura A.1.2, localizada en el Anexo A.

Para comprobar el funcionamiento del bloque será necesaria una simulación del mismo. Obteniendo el resultado mostrado en la Figura 5.0.8.

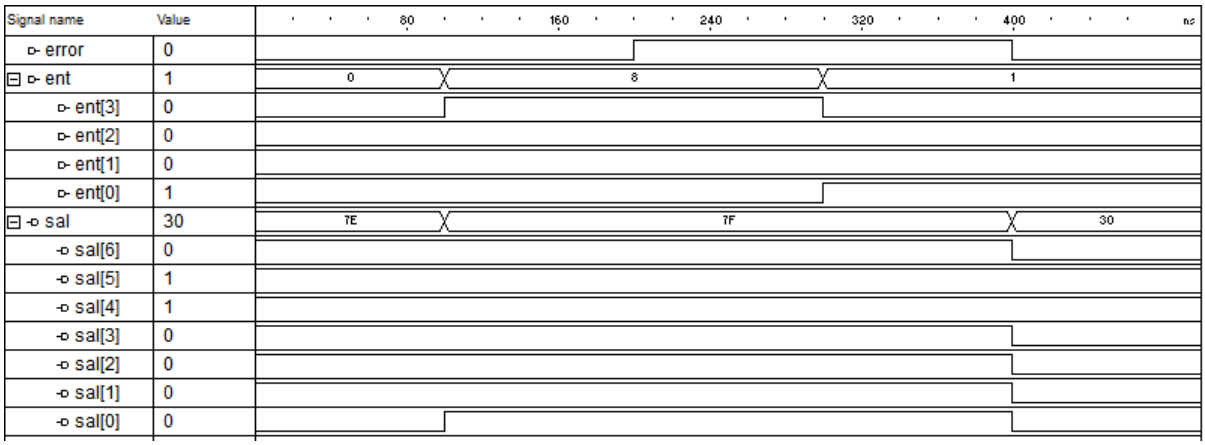


FIGURA 5.0.8. Simulación del Convertidor BCD-7segmentos

Aunque el estudio de su comportamiento por medio de esta simulación no es muy visual, se han planteado los casos en donde es más fácil de observar. Por ejemplo para representar el '0' será necesario activar todos los leds salvo el último (el led central), mientras que para representar el '8' será necesaria la activación de todos los leds.

En el caso de recibir la señal de error es necesario que la salida se mantenga constante. Esto se debe a que si recibimos una señal incorrecta no se debe mostrar en ningún momento, prevaleciendo la última señal válida.

El resto de componentes del circuito tienen aplicaciones específicas y por lo tanto no podrán ser utilizados en todos los casos.

A la hora de desarrollar el problema propuesto planteamos dos alternativas para la emisión de los datos:

1. Sistema por tiempo: La información es enviada en determinados instantes de tiempo, normalmente en tiempos equiespaciados. Serán sistemas mucho más manejables, aunque puede haber redundancias. Hay un mayor volumen de datos, con la ventaja de que la pérdida de información no es importante.
2. Sistemas por eventos: La información es enviada cuando haya un cambio en la entrada. En este caso obtendremos un menos volumen de datos, con la desventaja de que un pequeño fallo puede suponer una enorme pérdida de información.

Ambos sistemas tienen ventajas e inconvenientes, y su utilización depende de la aplicación a la que sea destinado. Para comprobar cual es el método óptimo para el desarrollo del proyecto final procederemos al desarrollo de ambos sistemas.

5.1. Sistema por eventos

En este caso, la información se envía cuando haya un cambio en la entrada. Como estamos trabajando con señales digitales, no hay umbral en la entrada para el cambio en la salida, es decir, el cambio en la entrada produce inequívocamente la emisión de un nuevo dato hacia el receptor.

Se presenta la ventaja de que, ante una posible entrada estática (sin cambio durante una gran cantidad de tiempo), la información no es enviada, pero con una serie de inconvenientes que iremos observando a la hora del desarrollo del circuito.

Como ya se ha explicado, tenemos una señal de reloj de 9.6kHz, un decodificador del interruptor activado más prioritario y una señal de activación del circuito (interruptor 1). Es necesario una parte más en el circuito propio del emisor: Un bloque que detecte el cambio en la entrada y, si está habilitado el circuito, que envíe por el pin RXD del módulo de Bluetooth el mensaje correspondiente siguiendo el protocolo establecido, y por tanto, el formato de mensaje que establece dicho protocolo.

Para ello, el bloque debe tener las entradas y salidas que se muestran en la Figura 5.1.1. Este bloque funciona como un **Multiplexor Especial**, es decir, envía el mensaje según el formato establecido. Para ello envía en primer lugar el bit de arranque "0", y posteriormente el dato con un tamaño de un byte.

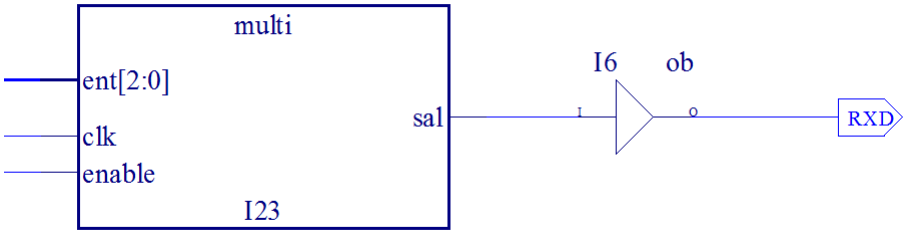


FIGURA 5.1.1. Multiplexor especial

Una manera de desarrollar este multiplexor especial es mediante la descripción funcional mediante VHDL, obteniendo el código de la Figura A.1.3 (situada en el Anexo A). Observamos en este caso que el código resulta muy engorroso en el caso de que el formato del mensaje fuera complejo, teniendo el código una estructura muy personal y resultando de mayor complejidad de comprensión para terceras personas. Es por tanto necesaria una simulación para comprobar el buen funcionamiento del sistema, obteniendo el resultado mostrado en la Figura 5.1.2.

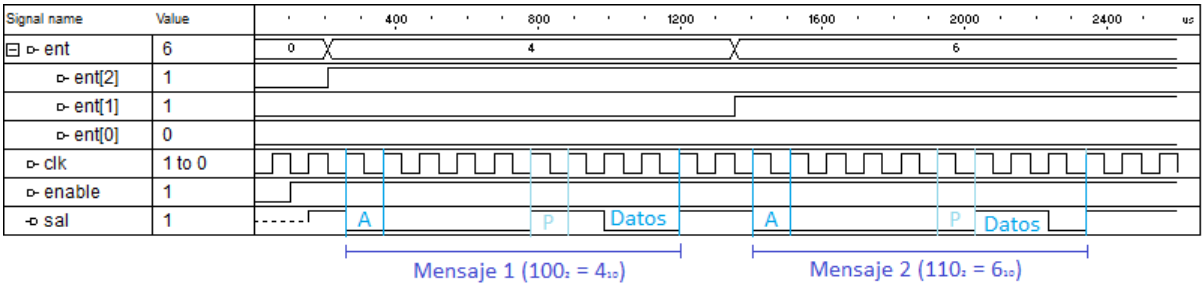


FIGURA 5.1.2. Simulación del multiplexor especial

Para que el sistema detecte el cambio en la entrada, es necesaria una habilitación del sistema (habilitado a nivel alto). Una vez que se produce el cambio se comienza a transmitir el mensaje por medio del bit de arranque (A). Seguidamente se transmiten los 8 bits de datos (Datos), donde se incluye el bit de paridad (P), útil para la detección de fallos.

La señal "anterior" (señal auxiliar para detectar el cambio en la entrada) se actualiza cuando se termina el campo de datos, de tal manera que garantizamos que no se produzca la emisión de un nuevo dato mientras enviamos el dato previo. Garantizamos también que el receptor sea capaz de distinguir entre mensajes consecutivos. En nuestra aplicación final este factor no va a ser de alta importancia, puesto que el sensor de temperatura y humedad tendrá una frecuencia de cambio muy baja (actualiza los datos cada 2 segundos aproximadamente).

Se debe realizar con una misma metodología, un **Demultiplexor Especial** que sea compatible con el multiplexor, o que realice las funciones inversas, en definitiva, que siga el mismo protocolo. Es necesario por tanto el desarrollo de un bloque con las entradas y salidas mostradas en la Figura 5.1.3.

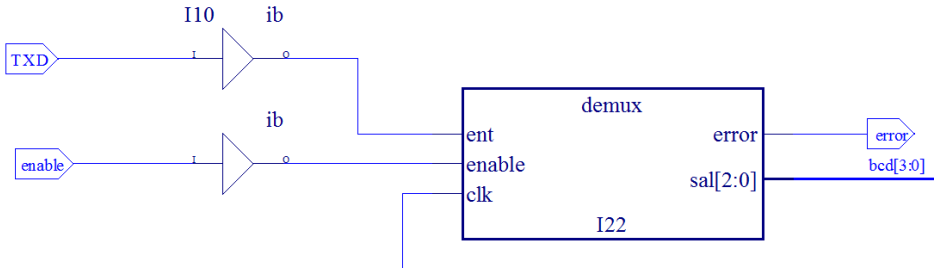


FIGURA 5.1.3. Demultiplexor Especial

Donde la entrada "TXD" es la entrada de datos procedente del módulo HC-12, la entrada "enable" es la entrada procedente del interruptor 1 (interruptor de habilitación), "clk" es la señal de 9600Hz, la salida "error" indica si se ha producido un fallo en la transmisión del mensaje con una comprobación gracias al bit de paridad, y por último, el bus "bcd" es el número de identificación del interruptor más prioritario accionado codificado en binario.

Para ello se ha generado el código que se muestra en la Figura A.1.4 que encontraremos en el Anexo A.

Para comprobar el funcionamiento del bloque hemos realizado una simulación del mismo, obteniendo el resultado de la Figura 5.1.4.

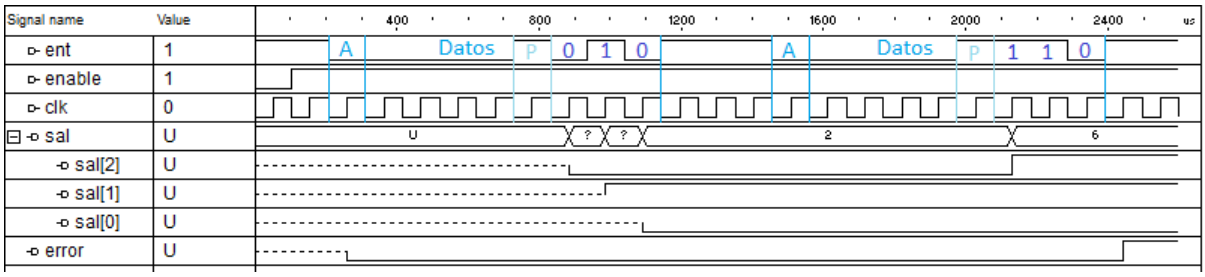


FIGURA 5.1.4. Simulación del Demultiplexor Especial

Se puede observar que, por la entrada, se reciben diferentes mensajes. El primero de ellos es el número 2, recibiéndolo de esta manera en binario (010). Puesto que esta codificación tiene un número impar de unos, el bit de paridad vale "1" (como se puede observar en la gráfica). En el segundo caso se recibe el número 6 (110), por lo que el bit de paridad deberá valer "0". Por el contrario se recibe un bit de paridad "1", por lo que se concluye que ha habido un error en la transmisión del mensaje, activándose de esta manera la señal de error.

Los bits de datos se van transmitiendo por las líneas correspondientes en los flancos de subida, siempre y cuando la señal de habilitación se encuentre a nivel alto. En el ejemplo mostrado en la gráfica de la Figura 5.1.4, los datos procedentes de la entrada se actualizan en los flancos de bajada, por lo que se garantiza una correcta recepción del mensaje.

Existe la posibilidad de que los datos se actualicen justo en el flanco de subida, de manera que una pequeña variación en los tiempos de llegada de los datos daría como consecuencia una incorrecta recepción de los datos. Este fallo tiene mayores

consecuencias en el caso de ser un sistema dirigido por eventos, como explicamos anteriormente, produciéndose una pérdida importante de datos.

Una posible solución es aumentar la frecuencia de muestreo de los datos y de esta forma garantizar la separación entre diferentes bits. Esta solución se considera que es la más adecuada, por lo que es desarrollada más adelante en este proyecto, una vez comprobado el funcionamiento del sistema.

Desarrollados los circuitos necesarios, los cuales formarán parte de un circuito general, se procede a realizar la interconexión de componentes adecuada.

En primer lugar, vamos a desarrollar el circuito propio del **Emisor**, es decir, el circuito que contiene la FPGA origen de la información. Este circuito está compuesto por los componentes: Multiplexor especial, Generador de reloj de 9600Hz y Decodificador de los Interruptores. Se obtiene de esta manera el resultado de la Figura 5.1.5.

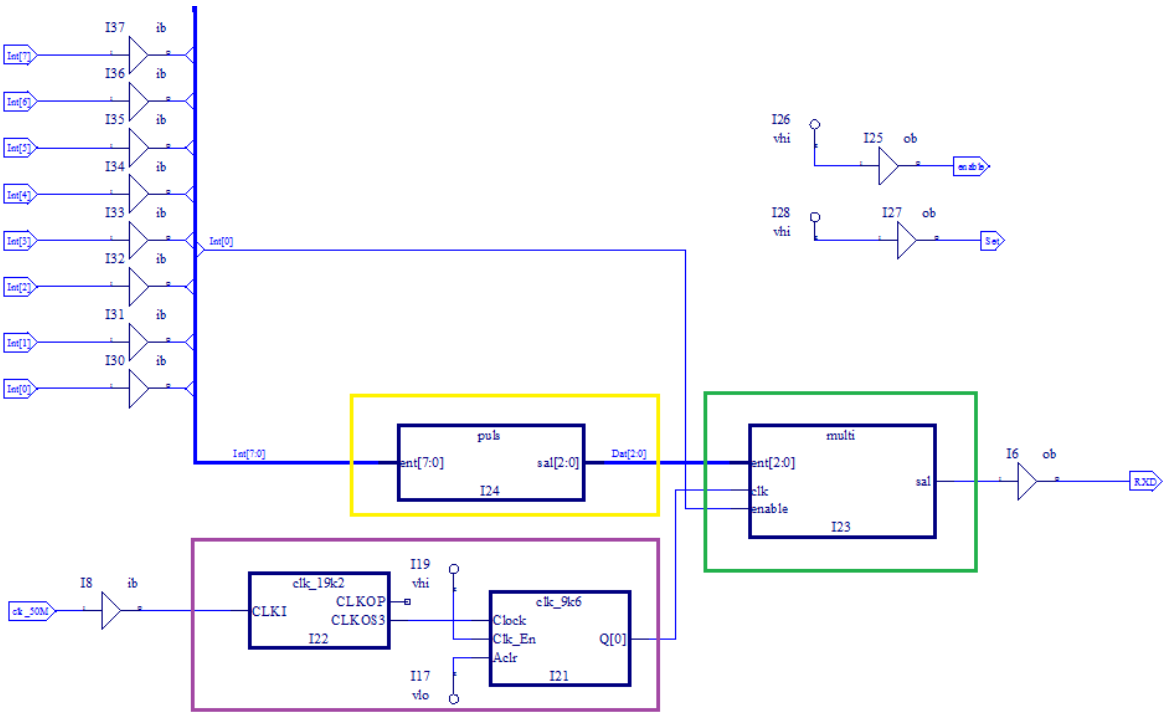


FIGURA 5.1.5. Circuito del Emisor de la comunicación básica

Su funcionamiento consiste en decodificar el número de interruptor más prioritario activado, obteniendo un número binario de 3 bits. Este número binario se trata en el multiplexor especial, que detecta cambios en la entrada, generando una

salida que va destinada al pin correspondiente del módulo HC-12 según el formato establecido en su protocolo. La señal "SET" debe estar siempre a nivel alto para no cambiar su modo de funcionamiento, al igual que la habilitación del reloj de 50MHz, el cual entra al sistema por un determinado puerto, pasando el mismo por una modulación de frecuencia para obtener un reloj de 9600Hz. El interruptor 1 funciona como habilitador del sistema, en caso de no estar habilitado, no habrá transmisión.

Los componentes en los que se divide el circuito vienen dado por las siguientes partes:

- Parte amarilla: Decodificador de los pulsadores
- Parte morada: Generador del reloj de 9.6kHz.
- Parte verde: Multiplexor especial que generará el mensaje.

Una manera eficaz de comprobar el funcionamiento del sistema consiste en cargar la FPGA con la descripción estructural realizada (Figura 5.1.5), realizándolo de una manera sencilla gracias al programa utilizado. Previamente se ha realizado una correcta puesta a punto de pines, enlazando cada entrada y cada salida con los puertos adecuados del dispositivo.

Una vez que el dispositivo tiene cargado el circuito, utilizando un analizador lógico, se puede observar las señales de los ejemplos mostrados en la Figura 5.1.6. Estas señales se generan cuando el interruptor de habilitación se encuentra activado y cuando se produce un cambio en el interruptor activado más significativo.

FIGURA 5.1.6. Señales enviadas hacia el módulo de Bluetooth de un sistema dirigido por eventos

En los ejemplos anteriores se muestran dos mensajes. En el primero se transmite el código 4 ("100", cuyo bit de paridad es 1, es decir, número impar de unos),

mientras que en el segundo se transmite el interruptor 6 ("110", con bit de paridad 0, es decir, número par de unos). Podemos concluir de esta manera que la descripción realizada para el dispositivo origen es la correcta.

En segundo lugar, se desarrolla el circuito propio del **Receptor**, es decir, el circuito que contiene la FPGA destino de la información. Este circuito está compuesto por los componentes: Demultiplexor especial, Generador de reloj de 9600Hz y Convertidor de BCD al código de display 7 segmentos. Se obtiene de esta manera el resultado de la Figura 5.1.7.

Su funcionamiento consiste en la recepción de un mensaje procedente del HC-12, el cual tiene el formato establecido por el protocolo. El demultiplexor especial trata este mensaje, identificando cada campo y convirtiéndolo de manera que se consigue el dato en un bus de 3 bits. Este dato se transforma, gracias al convertidor BCD-7segmentos, a una serie de señales que permiten observar, por el display, el número del interruptor más prioritario accionado.

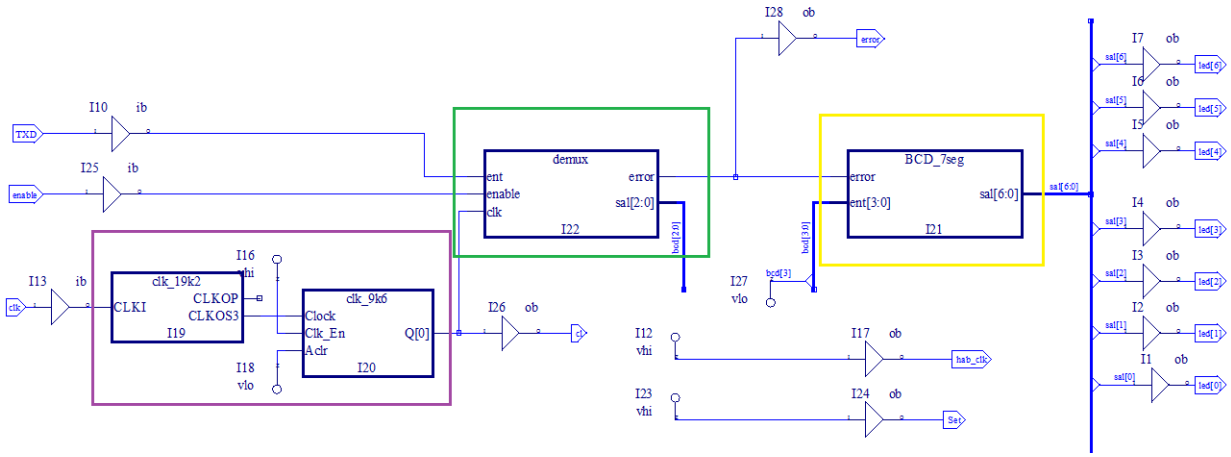


FIGURA 5.1.7. Circuito del Receptor de la comunicación básica

La señal "SET" debe estar siempre a nivel alto para no cambiar su modo de funcionamiento, al igual que la habilitación del reloj de 50MHz, el cual entra al sistema por un determinado puerto, pasando el mismo por una modulación de frecuencia para obtener un reloj de 9600Hz. El interruptor 1 funciona como habilitador del sistema, en caso de que no se encuentre habilitado, no habrá conversión. En caso de error en la recepción del mensaje, se encenderá un led en el dispositivo y la salida permanecerá constante hasta recibir otro dato.

Los componentes del circuito serán por tanto:

- Parte morada: Generador del reloj de 9.6kHz.
- Parte verde: Demultiplexor especial que genera el vector de salida.
- Parte amarilla: Convertidor de BCD a código del display 7 segmentos.

Posteriormente se debe comprobar el funcionamiento del sistema como conjunto. Para ello vamos a cargar las descripciones realizadas en dos diferentes dispositivos FPGAs con sus módulos correspondientes conectados, es decir, cargaremos la descripción del circuito emisor en uno de los dispositivos y la descripción del circuito receptor en el otro. De esta manera se ha comprobado que funciona correctamente, cumpliendo los requisitos definidos y, por tanto, los objetivos.

Destacamos las siguientes **conclusiones** del sistema dirigido por eventos:

- La tolerancia a fallos es elevada cuando la cobertura entre emisor y receptor es aceptable, aún así se pueden producir grandes pérdidas de información en un sistema como el nuestro. Por ejemplo, si por algún motivo la cobertura es débil, existe una gran posibilidad de pérdida de información, de tal manera que si el dispositivo emisor envía información en ese instante (por un cambio en la entrada) el receptor tomará valores erróneos incluso cuando la cobertura vuelva a ser aceptable.
- En un sistema múltiplemente cambiante la emisión de información se producirá con una frecuencia elevada, pudiendo incluso solaparse los mensajes. Sería necesario en ese caso, una serie de bits de parada para la distinción de diferentes mensajes.
- En el caso anterior se obtiene una gran cantidad de información. Si nuestro objetivo es mostrar la información por algún display, pantalla o dispositivo alternativo, es necesario un tratamiento de los datos (p.e. el cálculo de la media aritmética durante un determinado periodo de tiempo), ya que en caso contrario la información varia mucho y el ojo humano no es capaz de percibir los cambios.

5.2. Sistema por tiempo

En este tipo de sistemas, la información es enviada en determinados instantes de tiempo, normalmente en tiempos equiespaciados. Hay un mayor volumen de datos, con la ventaja de que la pérdida de información no es excesivamente importante.

Recordemos que el sensor de humedad y temperatura envía la información cada 2 segundos aproximadamente, de manera que una buena alternativa sería la realización de un sistema que funcione con unas características similares al sistema por eventos, pero que envíe la información con un periodo de 2 segundos. Es necesario, por tanto, el desarrollo de una serie de componentes (pertenecientes al circuito del emisor) que envíen un pulso de inicio cada 2 segundos, es decir, un **Generador de pulsos**. Estos componentes son los mostrados en la Figura 5.2.1.

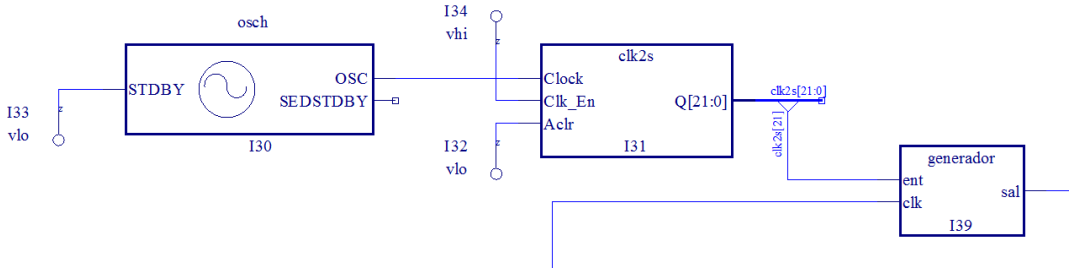


FIGURA 5.2.1. Generador de un pulso cada 2s

En dicha imagen se puede observar tres componentes. En el primero (localizado a la izquierda) se genera una señal de 2.08MHz, es decir, es el bloque correspondiente al reloj interno, el cual describimos en el apartado de descripción de componentes (1.1) y el cual se encuentra activado a nivel bajo.

La salida del primer bloque entra en el segundo, el cual es un contador, generado gracias a la herramienta IPexpress del programa utilizado. El contador está gobernado por un reloj, es decir, por la señal de 2.08MHz, la cual se divide entre 2^{22} en el caso de que se escoja como señal de salida el bit más significativo generado (2.08MHz / 2^{22} = 0.5Hz = 2s).

El último bloque es una descripción funcional de un generador de pulsos. Cuando en la señal de entrada hay un flanco de subida, se genera un pulso en la salida de un tamaño algo superior que la señal de reloj que utiliza el sistema (9600Hz), es decir, de menor frecuencia. El código desarrollado se muestra en la Figura A.1.5, localizada en el Anexo A. Este bloque detecta el cambio en la entrada gracias a una variable auxiliar llamada "anterior" y genera el pulso durante una serie de ciclos de reloj.

Se ha decidido que la realización del sistema se producirá mediante una alternativa al Multiplexor realizado anteriormente. Esta alternativa consiste en una

máquina de estados finitos (FSM) que controle el acceso a una memoria en la cual se almacena el dato a enviar, es decir, un **Controlador de la memoria**.

El inicio de la emisión de datos se produce gracias al pulso generado cada 2 segundos y es, por tanto, la entrada a la máquina de estados finitos. La máquina es de tipo Moore y tiene tantos estados como bits tiene nuestro mensaje (más el estado de la línea normalmente a "1"). Se puede observar dicha máquina en la Figura 5.2.2, donde las salidas son dos: Reset de la memoria y Habilitación de la memoria, respectivamente.

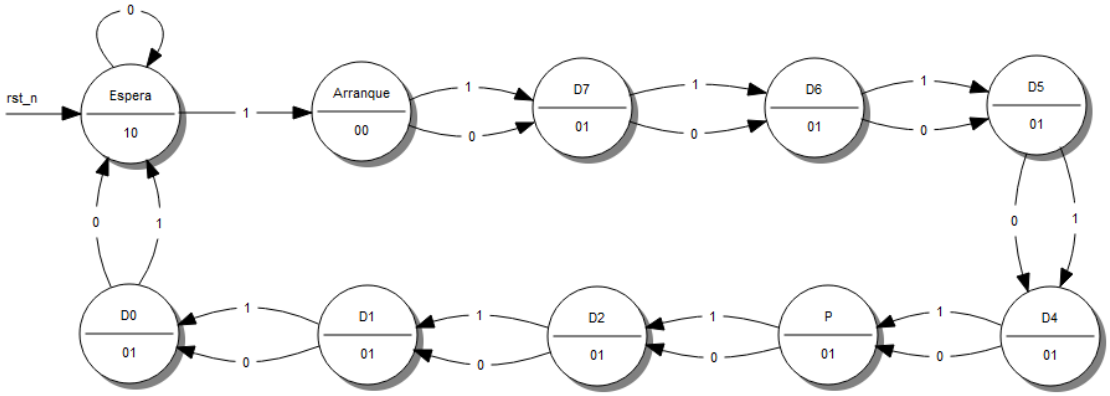


FIGURA 5.2.2. FSM controladora de la memoria del emisor

Tanto el reset como el habilitador actúan sobre el puntero a memoria, no sobre el contenido. De tal manera que cuando el reset esté activado se emita por defecto un "1" (línea sin transmitir datos), cuando la memoria no esté habilitada, pero no esté reseteada, se transmita un "0" (inicio de datos) y cuando la memoria esté habilitada se transmita un dato en cada transición de reloj. Cada estado indica, por tanto, un bit del formato definido del mensaje.

Será necesario realizar una **Memoria** que funcione correctamente con el control por parte de la máquina de estados finitos realizada, la cual debe tener la estructura de la Figura 5.2.3. Es decir, tendrá como señales de entrada un reset y un habilitador, cuyo valor depende de la FSM creada previamente. Como se ha dicho anteriormente, es importante tener claro que la señal de reset no vacía el contenido de la memoria, sino que pone el puntero de lectura en el primer dato a enviar (D7). Este puntero "cnt_rd" va recorriendo la memoria estableciendo diferentes salidas en cada transición de reloj.

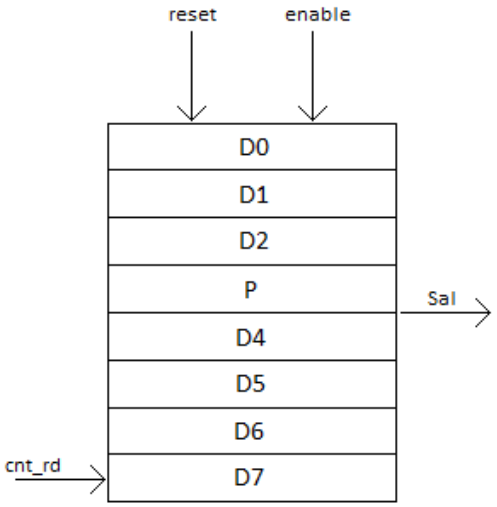


FIGURA 5.2.3. Memoria de 8 bits del emisor

Para describir el funcionamiento de la memoria hemos realizado una descripción funcional mediante VHDL que tendrá el código de la Figura A.1.6 localizado en el Anexo A, donde la señal "regis" es el contenido de la memoria y la variable "cuenta" es el puntero que indica la dirección de memoria a leer.

Una vez desarrollados estos bloques y partes del circuito podremos interconectarlas entre sí, añadiendo partes previamente desarrolladas, formando el circuito del **Emisor**. El resultado obtenido es el que se muestra en la Figura 5.2.4.

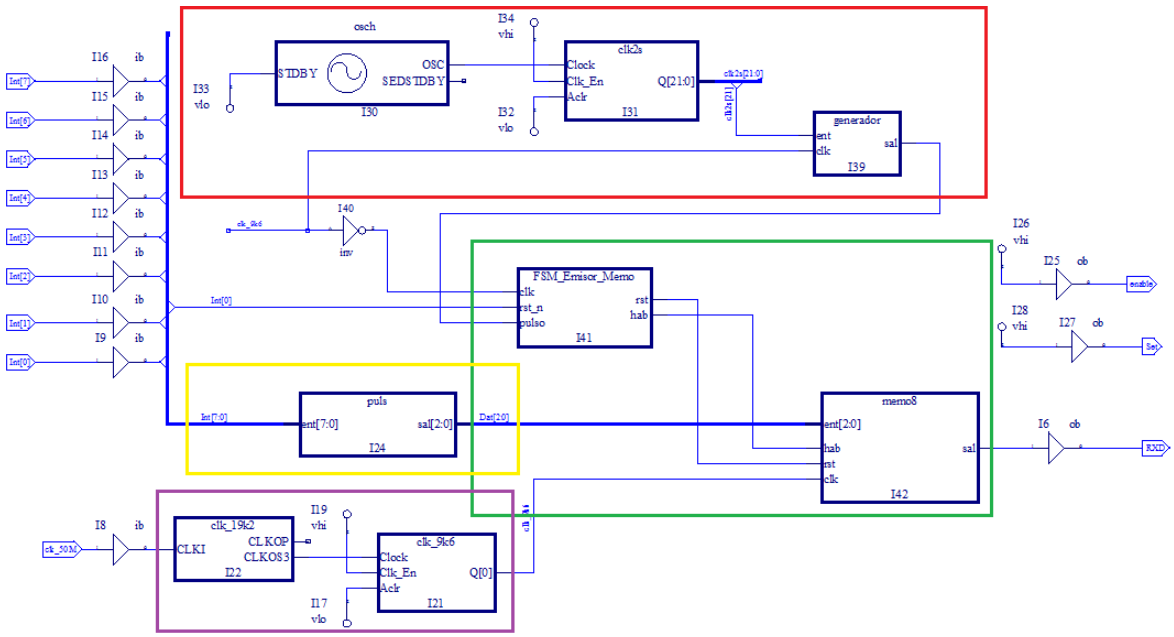


FIGURA 5.2.4. Circuito del Emisor de un sistema dirigido por tiempo

Podemos observar en dicha imagen el circuito completo con determinadas partes señalizadas, las cuales son las siguientes:

- Amarilla: Se produce la decodificación de los interruptores.
- Morada: Se genera una señal de reloj de 9600Hz.
- Roja: Generador del pulso de inicio cada 2 segundos enviado directamente a la FSM.
- Verde: Formada por el controlador de la memoria (FSM) y la propia memoria. Juntos generan un mensaje por el puerto serie. El controlador y la memoria utilizan el mismo reloj, a diferencia de que el controlador lo toma invertido con el objetivo de minimizar los posibles fallos. Esto es así debido a que se asegura que se produzca la salida correcta y determinada de un estado, es decir, en el flanco de bajada se actualiza el estado, mientras que en el flanco de subida se establece la salida, garantizando que no se produzca el cambio en el estado y en la salida simultáneamente..

Para comprobar su funcionamiento se ha decidido, al igual que en el sistema gobernado por eventos, cargar el circuito en el dispositivo y realizar un estudio del mensaje enviado mediante un analizador lógico, obteniendo el resultado de la Figura 5.2.5.

FIGURA 5.2.5. Señales enviadas hacia el módulo de Bluetooth de un sistema dirigido por tiempo

En la primera de ellas se envía el dato "0000 1100", es decir, el número 4 escogiendo los tres últimos bits según el protocolo (bit de paridad igual a "1"), mientras que en la segunda se muestra el número "0000 0110", es decir, el número 6 (bit de paridad igual a "0"). La señal "inicio" es el pulso emitido por el generador de pulsos cada 2 segundos, la misma que indica el inicio de la transmisión. El estado de la FSM desarrollada cambia en los flancos de bajada, como se ha indicado anteriormente.

Podemos concluir que la descripción realizada para el dispositivo origen es la correcta y, por lo tanto, lo es el circuito diseñado.

No es necesario desarrollar ningún circuito propio del receptor, puesto que ya ha sido diseñado en el sistema dirigido por eventos y el sistema cumple el funcionamiento deseado en ambos casos.

Obtenemos diferentes **conclusiones** para los sistemas dirigidos por tiempo:

- La pérdida de información no es crítica y por tanto no lo es la pérdida de cobertura entre los dispositivos emisor y receptor. Si se produce una pérdida en la cobertura, se perderán los datos que se encuentren en ese periodo de tiempo, sin embargo, ante una recuperación de la misma se recibirán los datos de una manera completamente correcta. Es decir, se garantiza que los datos son correctos mientras haya una cobertura aceptable.
- El sensor de temperatura y humedad utilizado nos convierte el sistema ineludiblemente en un sistema dirigido por tiempo. Esto se debe a que dicho sensor no detecta cambios en la temperatura y humedad ya que

sería una manera poco eficiente de realizar la toma de datos (debido a la gran variabilidad de estos dos parámetros), sino que envía con un periodo de dos segundos los valores de las variables de medida.

El circuito emisor del sistema dirigido por tiempo utiliza una metodología más eficiente y sofisticada que el circuito emisor del sistema dirigido por eventos. Esto se debe a que disponemos de una memoria (elemento más convencional que el multiplexor, generado únicamente para esta aplicación) y a una máquina de estados finitos para el control de la misma.

5.3. Alternativas de diseño

El funcionamiento de todos los circuitos desarrollados previamente son correctos, pero no son los mejores en todos los casos. Por ejemplo, encontramos algunas dificultades a la hora de realizar cambios en el diseño del multiplexor y del demultiplexor desarrollados, debido a que han sido desarrollados de manera muy personal y específica. Estos cambios pueden ser necesarios cuando el formato del mensaje varía, como es nuestro caso.

En el ejemplo realizado de los interruptores tenemos un formato de mensaje con 8 bits de datos, mientras que el formato del mensaje requerido para emitir los datos de temperatura y humedad es de 40 bits de datos (como indicamos en el apartado 3.1, en la Figura 3.2.

En el caso del circuito emisor existe una manera más sofisticada de desarrollarlo, y es mediante la máquina de estados finitos y mediante la memoria. Sin embargo, la forma de realizar el receptor se puede mejorar y hacerla más generalizable. Una alternativa es emplear una máquina de estados finitos que controle un registro de desplazamiento. Es este último el encargado de ir almacenando los valores procedentes del emisor.

El receptor sustituye el demultiplexor por estos dos bloques (FSM y registro). Por el contrario, no sufre ninguna modificación en el resto de componentes.

La **FSM controladora** necesita tener tantos estados como bits se reciban por el puerto serie, añadiéndole el estado de línea sin conexión. Un ejemplo de realización de esta máquina de estados finitos es la mostrada en la Figura 5.3.1, donde la entrada es la información recibida por el puerto serie, procedente del módulo de Bluetooth, y la salida es el habilitador del registro de desplazamiento.

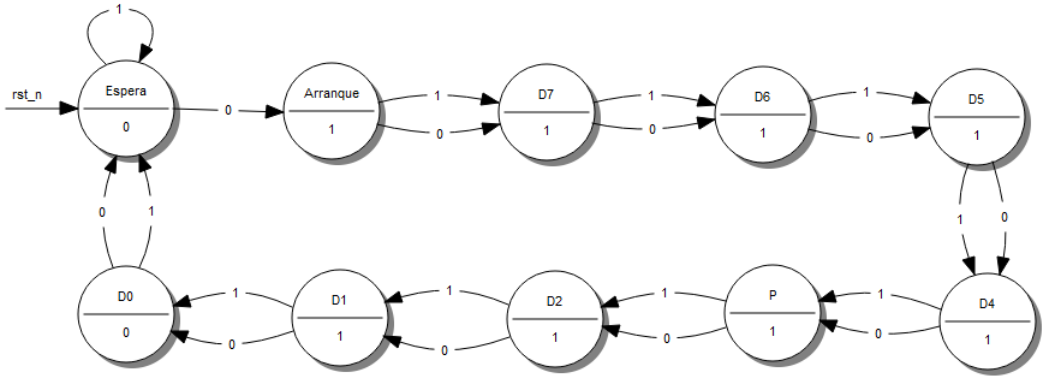


FIGURA 5.3.1. FSM controladora del registro del receptor

Mientras la entrada permanezca a nivel alto "1" se concluye que no se está recibiendo información, por lo que la FSM se mantiene en el estado de "Espera". El primer "0" recibido se interpreta como inicio de la comunicación y se pasa al estado "Arranque", el cual tiene la salida de habilitación del registro a nivel alto, con el objetivo de que en el siguiente flanco de subida del reloj se empiece a cargar el registro de desplazamiento con los datos correspondientes.

Después del bit de arranque llegan secuencialmente los bits de datos. Estos estados se van turnando en cada ciclo de reloj hasta el fin de datos, momento que pasaremos al estado de "Espera".

El **Registro de Desplazamiento** funciona de forma que cuando esté habilitado realiza un desplazamiento de sus datos y ,cuando se encuentre deshabilitado, mantiene dichos datos. El objetivo es ir desplazando el registro, hasta que se almacenen todos los datos necesarios en el mismo. Se ha decidido que el registro de desplazamiento debe cumplir también una función auxiliar, la de comprobación de errores. Esta función puede ser realizada por el registro de desplazamiento o por un bloque auxiliar, pero se considera un circuito más compacto si la función es realizada por el registro de desplazamiento.

Como es de esperar, la comprobación de errores es efectiva mientras los datos no varíen. Sin embargo, en el periodo de tiempo en el que los datos varían, la comprobación de errores no va a ser la adecuada debido a que comprueba la paridad de unos datos incompletos. Este periodo de tiempo está comprendido por 8 transiciones de reloj, es decir, desde que empieza a cargarse el dato en el registro hasta que termina de cargarse. Se concluye de esta manera que el periodo de tiempo es muy pequeño y por lo tanto no observable, ya que 8 transiciones

de reloj equivalen a menos de 1ms, tiempo demasiado pequeño como para que el usuario pueda observar el cambio.

Podemos observar las entradas y salidas que tiene el circuito del registro de desplazamiento en la Figura 5.3.2.

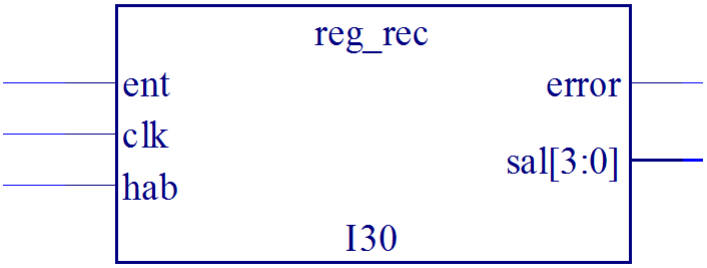


FIGURA 5.3.2. Registro de desplazamiento de 8 bits con cálculo de errores

La entrada a dicho bloque es la señal procedente del módulo de Bluetooth, la señal de habilitación es el control por parte de la FSM y el clk será el reloj del sistema de 9.6kHz. En el caso de las salidas, el error indica los posibles fallos en la transmisión gracias al bit de paridad y la salida es un vector de 4 bits, que indica el valor en BCD del código del interruptor más prioritario activado.

Para conseguir el objetivo propuesto se ha realizado el registro de desplazamiento basándose en la descripción funcional del mismo, a través del lenguaje VHDL. Podemos observar el código en la Figura A.1.7, localizada en el Anexo A, donde se puede observar el tratamiento de errores, el desplazamiento de los datos y el cálculo de la salida.

Si se realiza una simulación del registro de desplazamiento para comprobar su funcionamiento se observa las señales mostradas en la Figura 5.3.3.

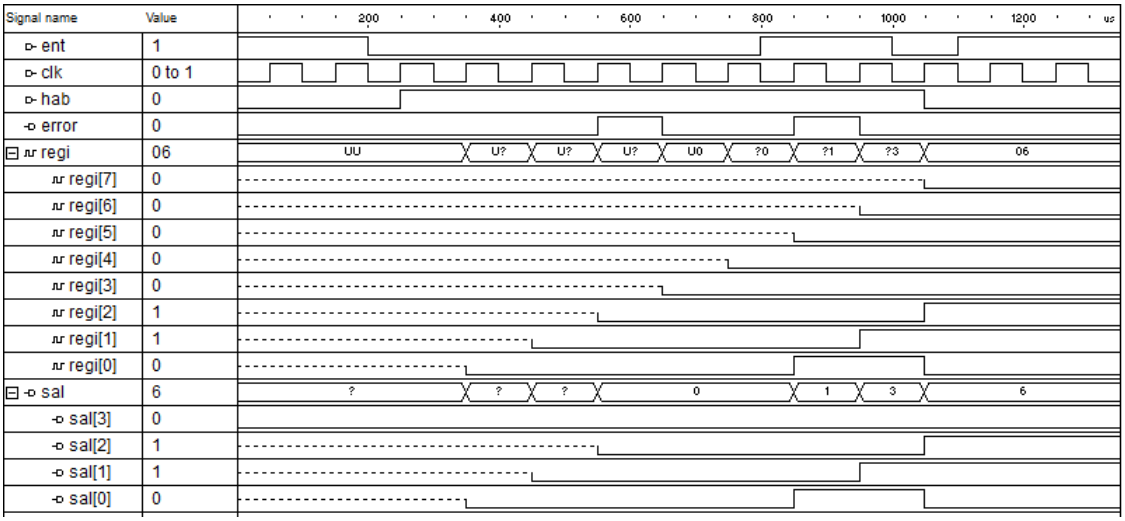


FIGURA 5.3.3. Simulación del Registro de Desplazamiento de 8 bits

Se observa la señal "ent", procedente del módulo de Bluetooth, que contiene el mensaje encapsulado. La señal de habilitación "hab" pasa a valer "1" cuando lo indique la máquina de estados finitos controladora. El registro se va rellenando poco a poco con los valores de entradas, mientras que la salida depende unicamente de 3 bits del registro, terminando con el valor deseado la salida.

Se puede observar también lo explicado anteriormente: mientras se carga el registro, la señal de error toma valores inapropiados (aunque no es importante). Se observa en cada señal como pasa a tomar el valor de la inmediatamente inferior, salvo la señal "regi[0]" que va tomando los valores de la entrada. La señal "hab" va a variar en los flancos de subida (gobernada por la FSM), al igual que la salida. Con la salvedad de que dicha salida se calcula tomando los valores anteriores al flanco de subida como entradas, por lo que se debe habilitar el registro un estado antes de que se empiece a cargar los datos por el puerto serie.

Es momento entonces de interconexionar los dos bloques anteriores con lo realizado en el circuito del receptor en apartados anteriores, obteniendo como resultado el circuito mostrado en la Figura 5.3.4.

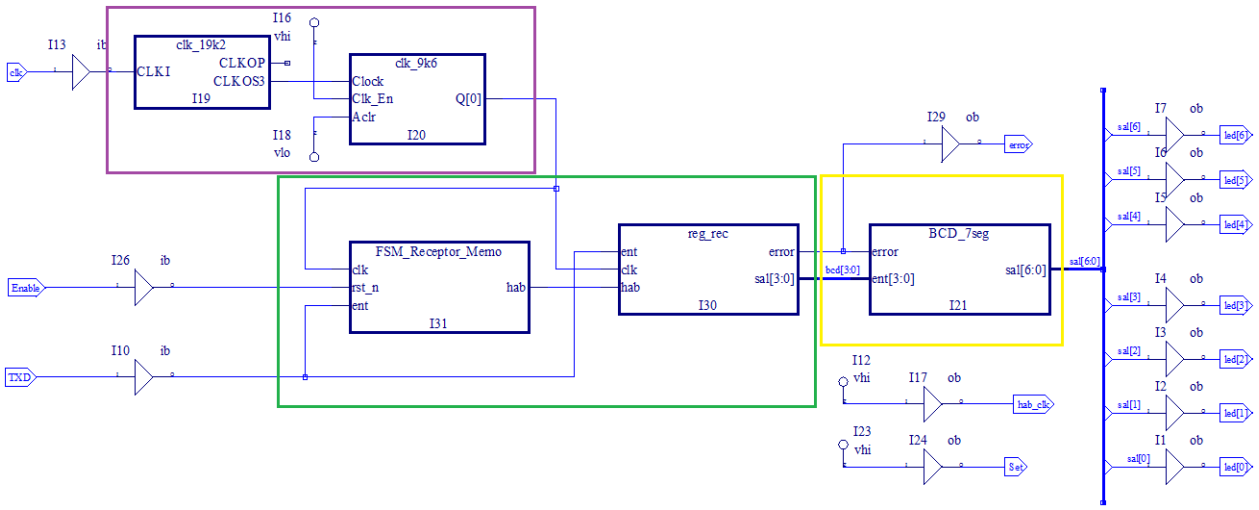


FIGURA 5.3.4. Circuito del Receptor formado por un Registro de Desplazamiento

Como en casos anteriores podemos distinguir los siguientes componentes:

- Parte morada: Generador de un reloj de 9.6kHz.
- Parte verde: Registro de desplazamiento donde se almacenarán los datos recibidos, con su correspondiente control por parte de una máquina de estados finitos.
- Parte amarilla: Decodificación de los datos recibidos y transformación para que sea mostrado por un display de 7 segmentos.

Posteriormente, se va a realizar las operaciones oportunas para cargar el circuito en nuestro dispositivo FPGA (p.e. generar el jedec, cargar ficheros, establecer los pines de salida y entrada correctos, etc.). Una vez cargada la descripción del circuito en nuestra FPGA se va a comprobar su funcionamiento. Para ello deberemos cargar cualquiera de los emisores generados anteriormente en otro dispositivo FPGA para que emita cierta información. Comprobamos de esta manera que nuestro diseño es correcto y el conjunto funciona correctamente para cualquier combinación de emisores y receptores.

Resumen

Se ha diseñado una **metodología de comunicación inalámbrica** mediante un **ejemplo básico**, a través de diferentes opciones. Durante la comunicación en este caso ha sido necesario el paso de información de una palabra (8 bits), que indica el interruptor más prioritario accionado.
Las alternativas que se han presentado para el **circuito responsable de la emisión** son, sistemas dirigidos por eventos y por tiempo. Debido a las mejores características de la segunda alternativa, y a un mayor generalidad de cara a futuras aplicaciones, se ha establecido dicha alternativa como la más funcional, reutilizable y de menor peso computacional.
Desde el punto de vista del **circuito responsable de la recepción** de la información, se planteaban dos alternativas viables. La primeras ha sido desarrollada a través de un demultiplexor especial. Con el objetivo de generalizar el circuito, se ha desarrollado la segunda alternativa, la cual utiliza una máquina de estados finitos y una memoria. Debido a las ventajas de esta última, se utilizará como base de futuras aplicaciones.

Capítulo 6

Estación meteorológica

Uno de los objetivos propuestos de proyecto es la comunicación entre dos FPGAs. Hasta este momento ya se ha cumplido ese objetivo, pero se propuso un objetivo algo mayor, el cual consistía en enviar la información de temperatura y humedad desde una FPGA origen que tomaba los datos, hacia una FPGA destino que realizaba un tratamiento de los mismos.

Recordemos también que la toma, interpretación y tratamiento de datos de temperatura y humedad formaba parte del Trabajo de Fin de Grado de un compañero, por lo que mi trabajo se centrará únicamente en la emisión y recepción de la información.

6.1. Toma y tratamiento de temperatura y humedad

Como se ha explicado en el apartado de componentes, el formato de la información que nos aporta el sensor de temperatura y humedad se basa en 40 bits (5 bytes) de información, organizados como muestra la Figura 3.2. Hasta ahora hemos enviado únicamente 8 bits (1 byte) por lo que se tendrá que realizar una serie de modificaciones en el diseño realizado anteriormente.

Se debe también "encajar" esos 40 bits a la metodología de emitir los datos por parte del módulo HC-12. Este módulo envía datos en paquetes de 8 bits, por lo que es necesaria la emisión de 5 paquetes de datos. Cada paquete de datos sigue el formado definido en apartados anteriores, es decir, un bit de arranque a nivel bajo y un byte de datos.

Si nos fijamos en la estructura del emisor más adecuado generado (del sistema dirigido por tiempo, apartado 5.2), nos damos cuenta de que existe una señal que genera un pulso para iniciar la emisión de los datos. En el caso anterior se generaba cada 2 segundos para simular el comportamiento del sensor de temperatura y humedad, sin embargo, en este caso se debe generar cuando se actualice el valor aportado por el sensor y sucesivas veces después hasta enviar el mensaje completo con una separación suficiente entre paquetes.

El circuito realizado en el TFG “Estación Meteorológica Basada en FPGA” (cuyo enlace con el presente proyecto se explica en el Anexo B) se puede observar en la Figura 6.1.1.

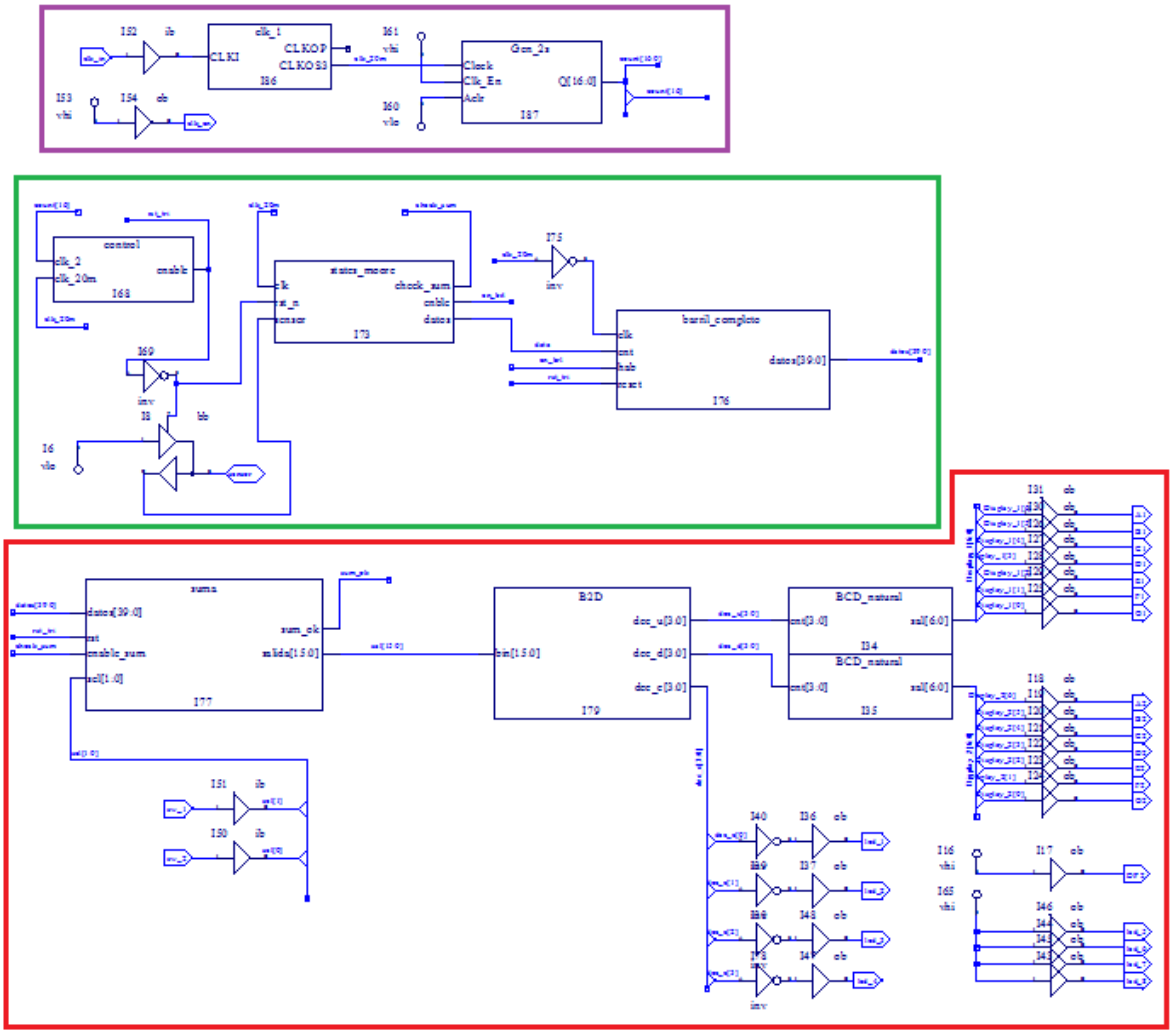


FIGURA 6.1.1. Circuito de obtención de Temperatura y Humedad

Donde podemos observar tres partes claramente diferenciadas:

- Parte morada: En esta parte se genera un reloj de 50kHz con el que trabaja todo el sistema (generado por el primer bloque llamado “clk_1”) y se genera un reloj de 0.5Hz, es decir, de periodo 2 segundos (generado por el segundo bloque llamado “Gen_2s”), el cual se utiliza para el inicio de la

obtención de datos. Se puede observar de una mejor forma en la Figura 6.1.2.

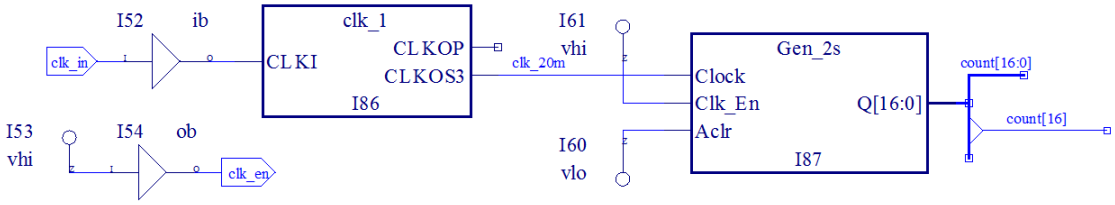


FIGURA 6.1.2. Generación de relojes

del circuito de obtención de temperatura y humedad

- Parte verde: En esta sección del circuito, se procede a emitir una petición al sensor para recibir información sobre la temperatura y humedad del ambiente. Posteriormente el sensor responde con dicha información, la cual se almacena en un registro de desplazamiento de 40 bits. Se puede observar esta parte del circuito completo en la Figura 6.1.3.

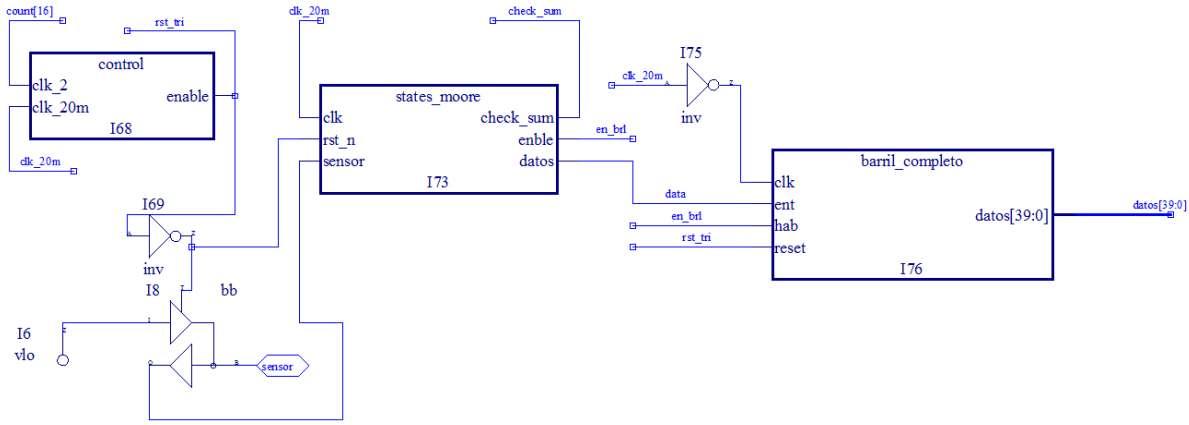


FIGURA 6.1.3. Petición de datos y recolección de los mismos

- Parte roja: Se procede a la comprobación de errores a partir del byte de suma (explicado en el apartado de los componentes utilizados), el cual es redundante de información. Si este byte coincide con la suma del resto de bytes se concluye la correcta recepción de los datos y se refresca la salida. Esta salida se muestra por los displays y los leds auxiliares de la placa MachXO2, dependiendo del interruptor accionado (para mostrar

temperatura o humedad). Se puede observar la Figura 6.1.4, donde se muestra esta parte del circuito.

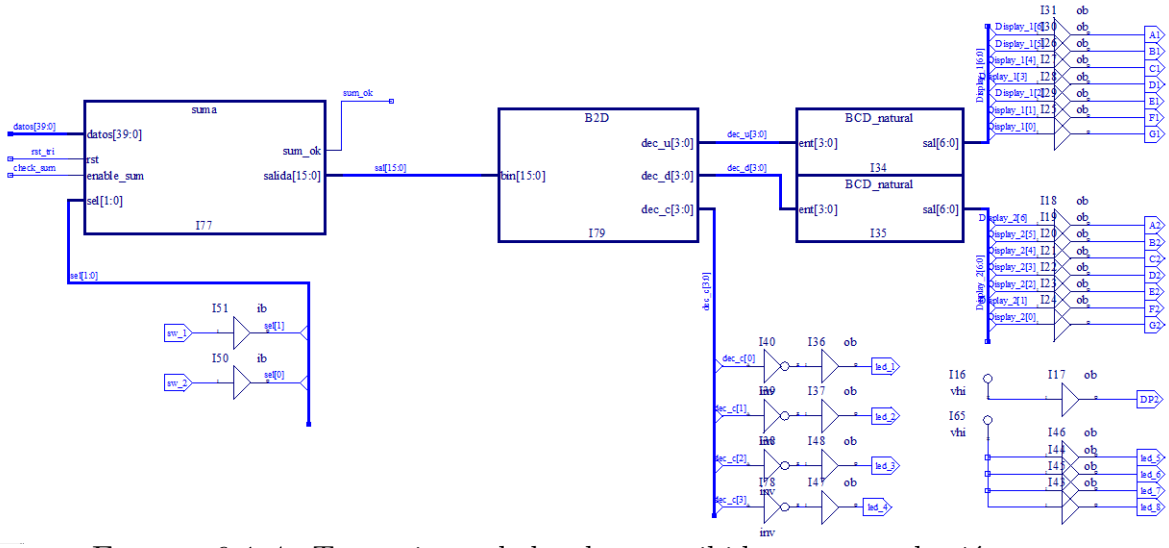


FIGURA 6.1.4. Tratamiento de los datos recibidos y comprobación de errores

En resumen, el funcionamiento del sistema es el siguiente: Cada dos segundos se envía una serie de valores binarios a través del pin de datos del sensor de temperatura y humedad DHT22. El sensor responde con una serie de valores correspondientes a la temperatura y a la humedad según el protocolo del mismo (con un byte de comprobación de errores). Siguiendo las pautas de velocidad de transmisión, se almacenan los valores recibidos en un registro de desplazamiento, el cual se trata de la manera más adecuada posible para mostrar los valores aportados por el sensor a través de los displays, no sin antes comprobar su correcta recepción a través del byte de comprobación de errores. Podemos observar tanto temperatura como humedad por los displays, la variable mostrada depende del estado de dos interruptores (uno activado y otro desactivado). Por los displays se mostrarán las unidades y las décimas, mientras que los leds mostrarán las decenas según el código binario.

6.2. Toma y emisión de datos

Una vez interpretado el circuito aportado en el TFG citado, es momento de realizar la parte correspondiente a la emisión y recepción de datos vía Bluetooth.

En primer lugar deberemos generar relojes con unas características especiales para cada parte del circuito, debido a que no todos los componentes van a utilizar el mismo reloj. Este hecho se ve reflejado en la utilización simultánea del sensor de temperatura y humedad, y del módulo de bluetooth, los cuales utilizan unas frecuencias de 50kHz y 9.6kHz respectivamente (9.6kHz es la velocidad de transmisión de datos por el puerto serie predeterminada).

Se debe realizar un diseño que genere estas frecuencias, a las que se añade un reloj de periodo 2 segundos, el cual se utiliza para dar comienzo a la toma de datos. Es decir, se necesitan relojes de frecuencias: 9.6kHz, 50kHz y 0.5Hz.

Prestando atención a las dos últimas, nos damos cuenta de que, puesto que disponemos de un reloj externo preciso de 50MHz, podemos conseguir dichas frecuencias con la división entre múltiplos de diez. Para ello tomaremos el reloj externo y, con contadores en cascada, realizaremos la división entre 1000 para conseguir un reloj de 50kHz (50MHz/1000 = 50kHz), con una posterior división entre 10000 para conseguir un reloj de 0.5Hz (50kHz/10000 = 0.5Hz).

Para la obtención de una frecuencia de 9.6kHz se debe realizar una rutina como en el apartado 5, en donde se conseguía una frecuencia de 19.2kHz mediante la utilización de un PLL, la cual se dividía entre dos mediante un contador para conseguir una frecuencia final de 9.6kHz.

El circuito aportado en el TFG nombrado se basaba en la utilización de un PLL para la obtención de 50kHz, pero nuestra FPGA únicamente cuenta con uno, por lo que se ha decidido emplear para la obtención de una frecuencia de 9.6kHz, la cual es más compleja y menos precisa de utilizar cuando la obtenemos mediante contadores.

La obtención de todas estas frecuencias se pueden observar en la Figura 6.2.1.

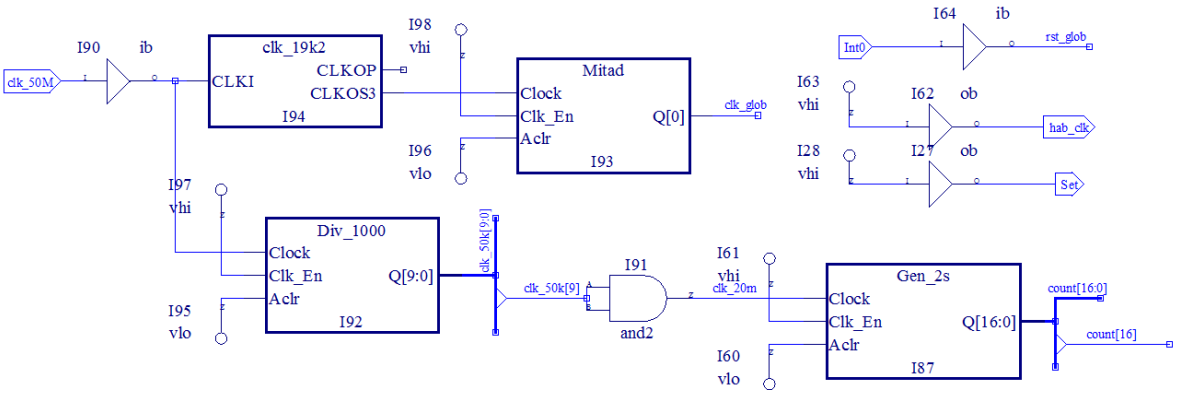


FIGURA 6.2.1. Generación de frecuencias para los diferentes módulos

Donde se observa en la parte superior la generación del reloj de 9.6kHz llamado “clk_glob” y en la parte inferior la generación, en primer lugar, del reloj de 50kHz llamado “clk_20m” o “clk_50k[9]” y, en segundo lugar, del reloj de 0.5Hz llamado “count[16]”.

Se ha añadido a dicha imagen la puesta a nivel alto del pin “SET” (para emisión vía bluetooth) y la entrada de reset a través de uno de los interruptores.

En el circuito de partida distinguíamos tres partes, entre las cuales se encuentra la toma de datos. A esta parte se deberá añadir determinados componentes para la emisión de los 40 bits de datos.

En esta parte del circuito (véase la Figura 6.1.3) podemos observar la señal "check_sum", la cual genera un pulso una vez que la información es aportada por el sensor. Esta información se vuelca sobre un vector de 40 bits llamado "datos". Por lo que es la señal “check_sum” la que indica el inicio de la comunicación.

Anteriormente se indicaba que necesitamos 5 pulsos que sirvan como inicio de la emisión de cada paquete, por lo que se necesita un bloque que los genere. Este bloque parte del pulso "check_sum" y genera cinco pulsos de una manera similar al generador de pulsos cada 2 segundos del apartado 5.2. Tiene las entradas y salidas mostradas en la Figura 6.2.2 y es un **Generador de 5 pulsos** en secuencia.

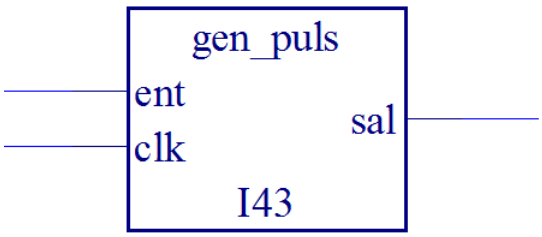


FIGURA 6.2.2. Generador de una secuencia de 5 pulsos

Este bloque ha sido generado a partir de la descripción funcional, mediante el lenguaje VHDL. Su código se muestra en la Figura A.2.1, en el Anexo A. Para comprobar su funcionamiento se ha realizado una simulación del mismo, en la cual se obtiene el resultado mostrado en la Figura 6.2.3.

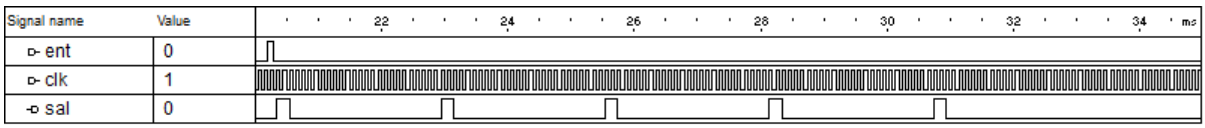


FIGURA 6.2.3. Simulación del Generador de una secuencia de 5 pulsos

Donde la señal "ent" corresponde a la señal "check_sum" (inicio de emisión).

Ante esta situación se presenta un problema, el "check_sum" es un pulso a nivel alto, pero trabaja con una frecuencia de 50kHz. Puesto que para la emisión a través del módulo HC-12 trabajamos con una frecuencia de 9.6kHZ, en la mayoría de las ocasiones no se podrá detectar dicho pulso. Por esta razón se deberá desarrollar un bloque que alargue el pulso durante un tiempo suficientemente largo para que, a una frecuencia de 9.6kHz, se pueda detectar su estado a nivel alto con seguridad.

Se ha desarrollado, por tanto, el bloque de la Figura 6.2.4, el cual es un **Alargador de pulsos**. Funciona de tal forma que, recibiendo un pulso de frecuencia 50kHz (o frecuencia del reloj que introduzcamos), se produce una ampliación del mismo a lo largo del tiempo. Cabe destacar que la frecuencia se refiere al tiempo entre pulsos, no a su tamaño.

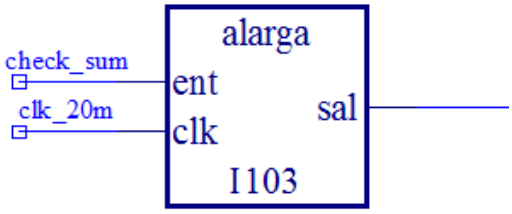


FIGURA 6.2.4. Alargador de pulso

Para el desarrollo de este bloque nos hemos basado en una descripción funcional mediante el lenguaje VHDL. Por lo que se ha desarrollado el código que se encuentra en la Figura A.2.2, en el Anexo A. Para la comprobación de su funcionamiento se ha realizado una simulación del mismo, obteniendo el resultado mostrado en la Figura 6.2.5.

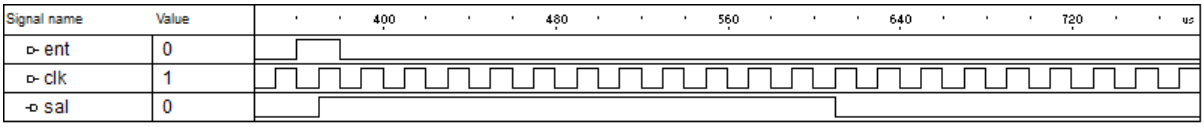


FIGURA 6.2.5. Simulación del Alargador de pulsos

Donde se puede observar que el pulso se prolonga durante un tiempo suficiente para que un reloj de 9.6kHz sea capaz de detectarlo en el caso más desfavorable (que se produzca justo después de su flanco de subida).

El resto de componentes se basan en la reutilización de componentes del Apartado 5. Esto se debe a que la emisión de datos se realiza de una manera idéntica, es decir, se dispone de una máquina de estados finitos que sigue el protocolo de emisión establecido (bit de arranque y byte de datos en paquetes) cuando reciba un pulso a nivel alto, y se dispone también de una memoria que contiene los valores de un vector (en este caso los 40 bits de datos de temperatura y humedad).

Puesto que en apartados anteriores ya se ha desarrollado la FSM, que sigue el protocolo establecido, es necesario el diseño de la memoria de 40 bits, la cual se basa en la memoria de 8 bits desarrollada en el Apartado 5. El bloque de dicha memoria tiene la apariencia de la Figura 6.2.6, donde como entradas, dispone de los 40 bits procedentes del sensor, el reloj y dos variables que controlan el puntero de lectura a dicha memoria (“rst” y “hab”). Se puede encontrar su código en la Figura A.2.3, localizada en el Anexo A..

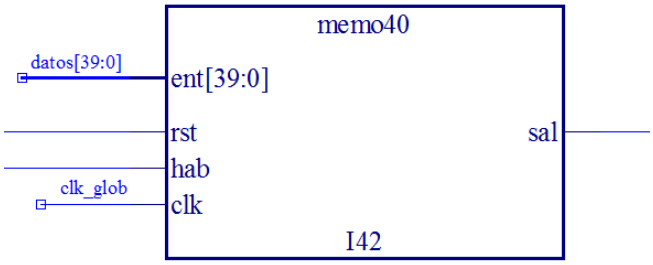


FIGURA 6.2.6. Memoria de 40 bits del emisor

El próximo paso es la interconexión de las partes del circuito propias de la comunicación. Para ello se ha desarrollado el circuito mostrado en la Figura 6.2.7.

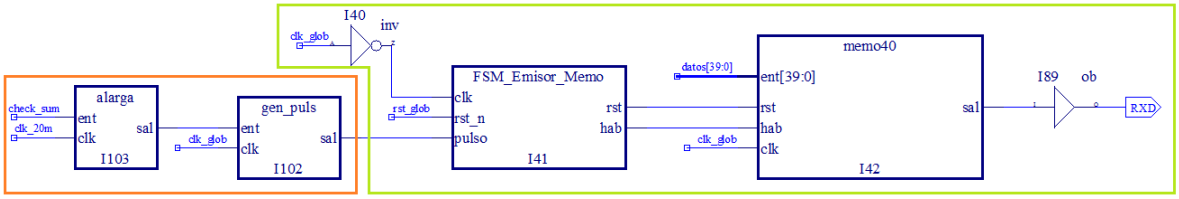


FIGURA 6.2.7. Emisión de temperatura y humedad (40 bits)

Se pueden observar en dicha imagen dos partes claramente diferenciadas. La primera de ellas (izquierda) está formada por el Alargador de pulso (aumenta el tiempo que "check_sum" se encuentra a nivel alto) y por el Generador de pulsos (el cual partiendo de un único pulso genera 5 pulsos para el inicio de la emisión de paquetes de datos). En el caso de la segunda parte, podremos diferenciar la FSM, que genera el protocolo, y la memoria de 40 bits, la cual contiene los valores de temperatura y humedad más recientes (los cuales se emiten por el puerto serie según el protocolo de emisión).

Para que se genere la señal "check_sum" y para que la memoria de 40 bits contenga los valores de temperatura y humedad adecuados, es necesaria la interconexión con la parte del circuito de petición y recolección de datos del sensor. Esta combinación da como resultado el circuito mostrado en la Figura 6.2.8.

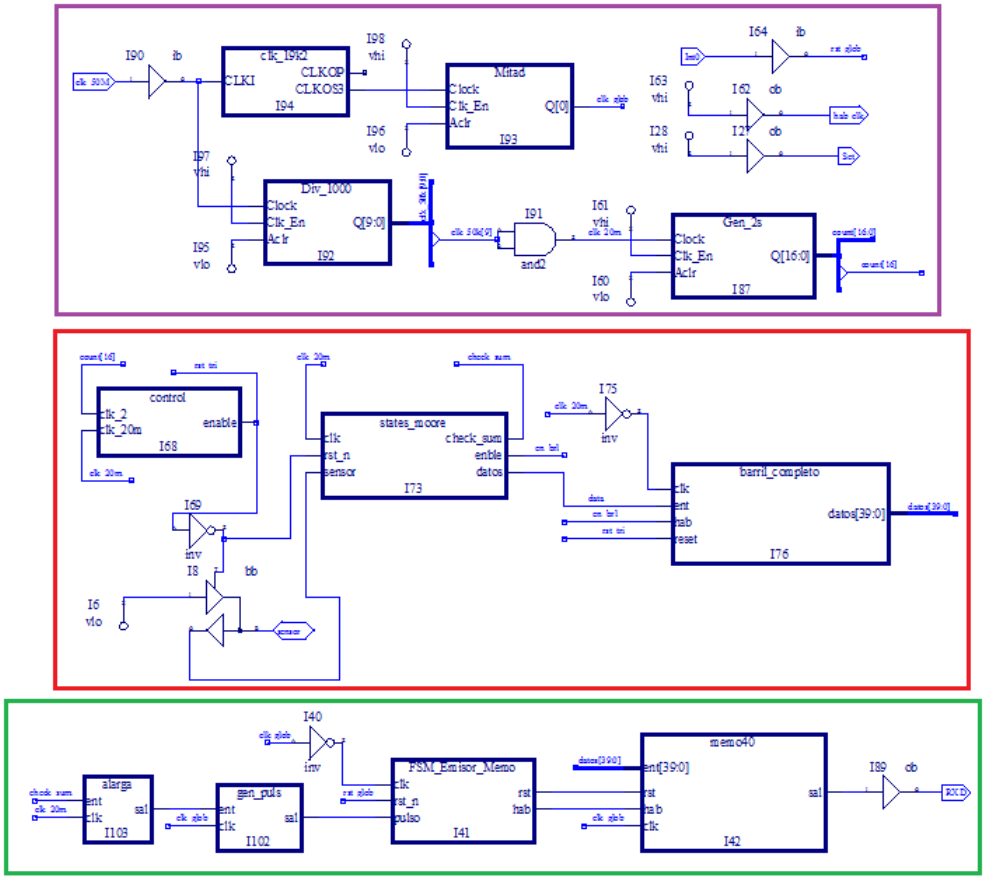


FIGURA 6.2.8. Circuito de toma y emisión de temperatura y humedad

Donde se distinguen tres partes funcionalmente distintas:

- Parte morada: Generación de los relojes de 9.6kHz, 50kHz y 0.5Hz.
- Parte roja: Petición y toma de datos de temperatura y humedad del sensor. Estos datos se almacenan en un registro de desplazamiento de 40 bits.
- Parte verde: Toma el contenido del registro de desplazamiento cuando se termina de actualizar y emite los datos por el puerto serie hacia el módulo de Bluetooth según el protocolo utilizado por el mismo.

Para comprobar su funcionamiento se ha decidido cargar dicho circuito en la FPGA y observar cual es la salida hacia el puerto serie "RXD" del módulo HC-12 mediante el analizador lógico, observando ciertas variables de interés. Esto se puede observar en la Figura 6.2.9, donde se muestra la señal enviada por el puerto

serie hacia el módulo Bluetooth, la señal recibida desde el sensor y las señales de "check_sum" y su correspondiente prolongación a lo largo del tiempo.

FIGURA 6.2.9. Señales generadas en la emisión

Como es de esperar, posterior a la señal "pulso inicio" se generarán 5 pulsos equiespaciados para indicar el inicio de cada paquete de una manera idéntica a la simulación realizada con anterioridad.

Una vez que se emitan todos los paquetes, el sistema queda a la espera de nuevos datos, para poder emitir información. Cada dos segundos se repite la secuencia con el objetivo de que un receptor capte la información.

6.3. Recepción y tratamiento de datos

Una vez emitida la información por parte del emisor, se debe desarrollar un receptor que capte dicha información y la trate de la manera más adecuada posible. Para ello, y puesto que el tratamiento de la información ya ha sido desarrollado, se debe diseñar una manera para recoger la información procedente del puerto serie del módulo de Bluetooth para que sea almacenada en un vector de 40 bits.

Basándose en la metodología utilizada en apartados anteriores para la recepción de información (FSM que interprete el protocolo y registro de desplazamiento), se debe hacer unas pequeñas modificaciones en las partes utilizadas por el receptor.

En primer lugar, la máquina de estados finitos que interpreta el protocolo utilizado por el módulo HC-12 no sufre ninguna modificación, puesto que el formato de los mensajes sigue siendo el mismo que en situaciones anteriores. Sin embargo, el registro de desplazamiento va a sufrir una serie de modificaciones de tal manera que, según las señales recibidas desde la FSM, desplace su contenido para albergar la información necesaria.

El registro de desplazamiento ha sido desarrollado mediante la descripción funcional mediante VHDL, por lo que su código se encuentra en el Anexo A, en la Figura A.2.4. Desde el punto de vista de entradas y salidas, la única modificación que sufre respecto al registro de desplazamiento desarrollado en apartados anteriores es que su salida será un vector de 40 bits (su propio contenido).

Observando la Figura 6.1.4, donde se muestra la parte del circuito de partida donde se tratan los datos de temperatura y humedad y donde se comprueba la posibilidad de errores, se puede observar que para el inicio de todo el proceso se requiere de la señal "check_sum". Esta señal forma parte de la toma de datos, por lo que estará en la FPGA origen de la información, es decir, del emisor.

Para que todo el proceso empiece, sin tener la señal "check_sum", es necesario el desarrollo de una señal sustitutiva a la anterior. Esta señal debe generar un pulso a nivel alto cuando toda la información haya sido recibida por parte del módulo de Bluetooth y haya sido almacenada correctamente en el registro de desplazamiento de 40 bits.

Con este objetivo se ha decidido realizar un estudio de la señal que comunica la máquina de estados finitos y el registro de desplazamiento, es decir, la señal que habilita el desplazamiento del contenido del registro con el ánimo de albergar la información. Esta señal llamada "hab", tanto en la salida de la FSM como en la entrada del registro, valdrá '1' cuando se reciban datos por el puerto serie (excluyendo los bits de parada y el bit de arranque). Por esta razón, si dicha señal vale '0' durante un largo periodo de tiempo, significa que no se está recibiendo datos y por tanto (como los 5 paquetes de datos vienen de una manera seguida) no hay transmisión de información.

En resumen, se debe diseñar un bloque que detecte si la señal de habilitación del registro "hab" se encuentra a nivel bajo durante un determinado periodo de tiempo y, si es así, se genere un pulso indicando que se deberá tratar los datos comprobando errores y mostrando el resultado por el display. Este bloque deberá tener las entradas y salidas mostradas en la Figura 6.3.1.

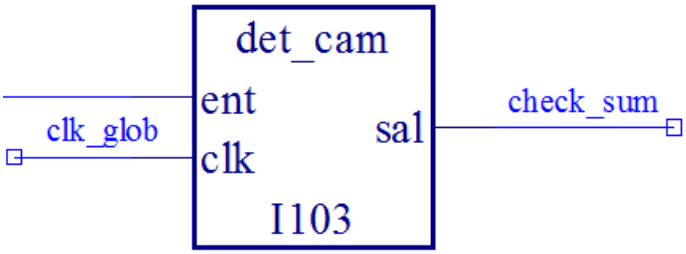


FIGURA 6.3.1. Detector del fin de transmisión

Su diseño ha sido realizado mediante la descripción funcional mediante VHDL, por lo que su código se muestra en la Figura A.2.5, en el Anexo A. Donde se puede observar que su funcionamiento consiste en lo siguiente: Si es la primera vez que se producen 10 transiciones de reloj con el habilitador a '0' desde que se recibió el último '1', se produce un pulso a nivel alto en la salida de una transición de reloj, en caso contrario la salida se mantiene a nivel bajo.

Para comprobar el funcionamiento del mismo se ha procedido a realizar una simulación, en la que se ha obtenido el resultado mostrado en la Figura 6.3.2.

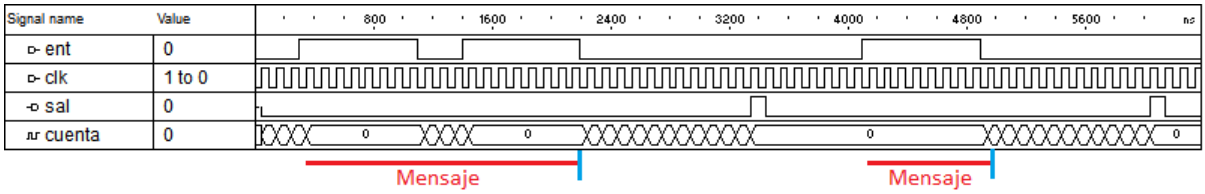


FIGURA 6.3.2. Simulación del Detector del fin de transmisión

Donde se puede observar con una línea azul el fin de cada transmisión (el habilitador "ent" se pone a '0' durante un largo periodo de tiempo). Después de 10 transiciones de reloj desde el fin de la transmisión se produce un pulso a nivel alto en la salida, mientras que después de dicho pulso el contaje de transiciones de reloj con el habilitador a '0' no varía, esperando a recibir que el habilitador valga '1' para volver a contar (recepción de un nuevo bloque de mensajes).

El siguiente paso será la interconexión de todos los bloques desarrollados anteriormente para la recepción de los 40 bits de información recibidos por el puerto

serie “TXD” del módulo de Bluetooth, es decir, la FSM que interpreta el protocolo, el registro de desplazamiento de 40 bits y el detector del final de la transmisión de un bloque de mensajes.

Esta combinación se puede observar en la Figura 6.3.3, donde se aprecian los tres componentes mencionados. Como entrada a esta parte del circuito se dispone el puerto serie “TXD” mientras que como salidas, los 40 bits de datos (contenido del registro) y la señal “check_sum” que se produce cuando se concluye la recepción de los datos.

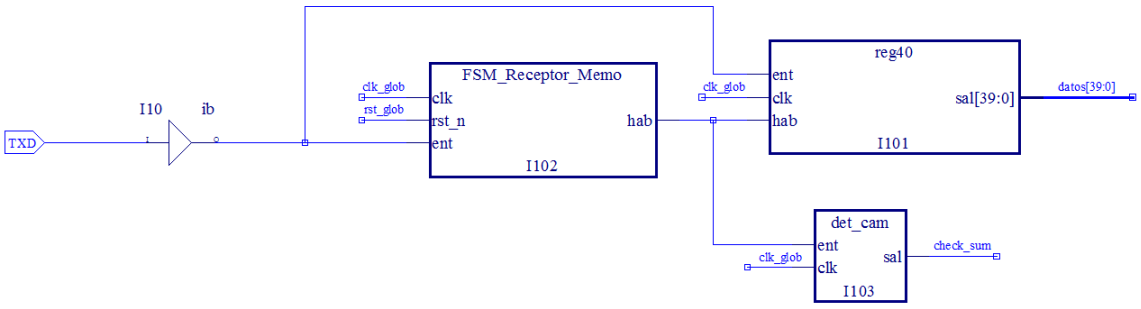


FIGURA 6.3.3. Recepción de temperatura y humedad (40 bits)

Se debe añadir al circuito anterior dos componentes. Uno de ellos consiste en la generación de relojes con diferentes frecuencias, mientras que el otro consiste en el tratamiento de los datos. Podemos observar la interconexión final en la Figura 6.3.4.

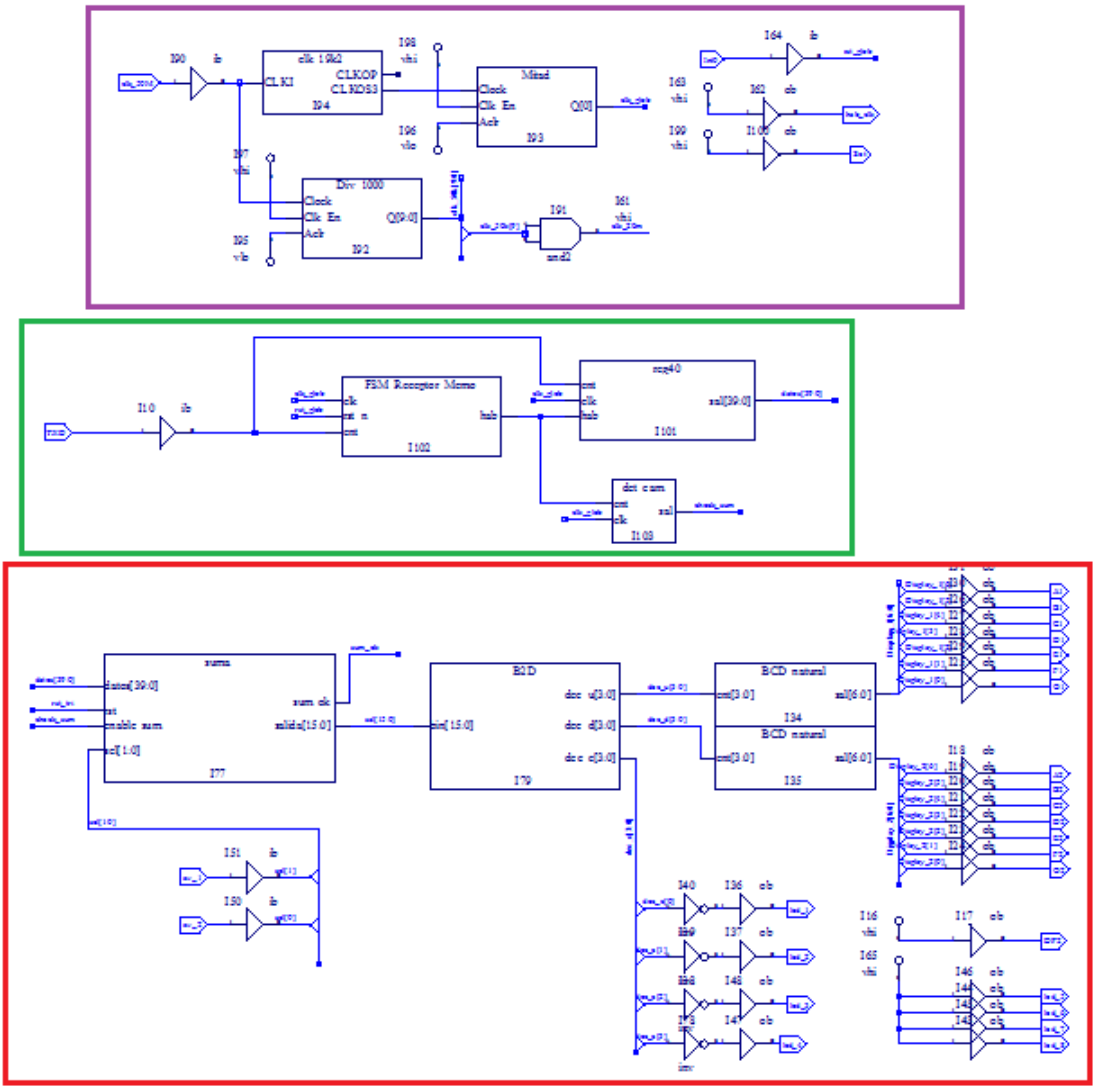


FIGURA 6.3.4. Circuito de recepción y representación de temperatura y humedad

Donde se pueden observar las siguientes partes:

- Parte morada: Generación de los diferentes relojes que utilizan el resto de partes del circuito (9.6kHz y 50kHz). Esta parte ha sido explicada con anterioridad por lo que ha sido realizada de la misma manera, con la única excepción de que en este circuito no es necesario un reloj de 0.5Hz, puesto que ésta se utilizaba para indicar el inicio de la toma de datos aportados por el sensor (véase Figura 6.2.1).

- Parte verde: Es la encargada de recibir la información de temperatura y humedad a través del módulo de Bluetooth. Almacena los datos en un vector de 40 bits que posteriormente se utiliza para decodificar ambas variables.
- Parte roja: Es la parte encargada de tratar la información recibida y almacenada en el registro de desplazamiento. Dependiendo del interruptor accionado, muestra la temperatura o la humedad relativa gracias a los displays y a los leds auxiliares (véase Figura 6.1.4).

El siguiente paso es la comprobación de la disponibilidad del funcionamiento del mismo, para ello se debe cargar una FPGA con el circuito descrito anteriormente. Por supuesto es necesario disponer de otra FPGA en la que se cargue el circuito que tome los valores de temperatura y humedad y envíe dichos valores gracias al módulo de Bluetooth HC-12.

Una vez hecho esto tendremos dos FPGAs, una origen de la información y otra destino, con sus correspondientes circuitos internos que describan un comportamiento adecuado para la correcta comunicación entre ambas. El comportamiento de ambos circuitos ha sido comprobado en todos los casos, garantizando su funcionamiento y garantizando que tanto el emisor como el receptor realizan sus funciones satisfactoriamente.

Resumen

Con el objetivo de la comunicación entre diferentes FPGA conseguido y optimizado, se ha diseñado una descripción del funcionamiento del circuito (basándonos en la **metodología de comunicación** anteriormente desarrollada) para la comunicación de **5 palabras (40 bits)**.
Para establecer la comunicación, y desarrollar el **circuito del emisor**, se ha partido de los datos de temperatura y humedad que han sido determinados en el TFG citado, desarrollado en paralelo con éste. Con esta información, se ha enviado secuencialmente palabras hasta emitir cinco de ellas.
En la otra parte, se dispone del **circuito del receptor**, en donde se han recibido las cinco palabras. Cuando finaliza el traspaso de información, se almacenan los datos en un registro de desplazamiento siguiendo el protocolo establecido y se lanza un tratamiento de los datos que comprueba posibles errores y muestra los parámetros, tomados por el emisor, a través de los displays y leds disponibles.

Capítulo 7

Estudio de los comandos AT

Cuando se describió el módulo HC-12 utilizado, se centraba la explicación en sus características, funcionamiento y peculiaridades. También se especifica que el dispositivo cuenta con una serie de comandos AT, los cuales nos permiten configurar los parámetros del mismo y elegir, de esta manera, su modo de funcionamiento, velocidad de transmisión por el puerto serie, etc.

Gracias a estos comandos podemos modificar los parámetros de tal forma que se ajusten de la mejor forma posible a cada aplicación.

En este apartado del proyecto, se va a realizar un ajuste de los parámetros de manera que se tenga un sistema más versátil, empleando los comandos AT. En primer lugar se va a diseñar un circuito que sea capaz de generar una serie de mensajes que, con un protocolo definido, sirvan para formar palabras. Para ello, y apoyándonos en la estructura realizada en apartados anteriores, donde se incluye la máquina de estados finitos y la memoria de emisión, es necesario el diseño de un circuito que genere pulsos, tantos como mensajes debamos enviar para generar el comando.

Puesto que el protocolo que sigue el módulo HC-12 define un formato de mensaje formado por un bit de arranque a nivel bajo y un byte de datos, es necesaria la emisión de tantos mensajes como caracteres tenga el comando AT. Los comandos que soporta este módulo son 12:

1. AT: Comando de test.
2. AT+Bxxxx: Cambia la velocidad de transmisión por el puerto serie.
3. AT+Cxxxx: Cambia el canal de comunicación inalámbrico, es decir, la frecuencia de emisión a través del aire (entre 001 y 127, trabajando el canal 001 con una frecuencia de 433.4MHz y aumentando en cada paso una frecuencia de 400KHz).
4. AT+FUx: Cambia el modo de transmisión transparente (FU1, FU2, FU3).
5. AT+Px: Configura la potencia transmitida por el módulo (entre 1 y 8). Nos permitirá realizar comunicaciones a distintas distancias.

6. AT+SLEEP: Apaga temporalmente la comunicación hasta recibir otro comando AT.
7. AT+DEFAULT: Establece los parámetros predeterminados.
8. AT+UPDATE: Inhibirá todos los comandos enviados hasta que se desconecte y conecte a la alimentación de nuevo.
9. AT+Ry: Obtiene la información de un parámetro en concreto del módulo (B, C, F y P).
10. AT+RX: Obtiene todos los parámetros del módulo.
11. AT+Uxxx: Configura el formato del mensaje. La modificación en el formato nos llevaría a una modificación de todo el circuito por lo que no utilizaremos este comando.
12. AT+V: Obtiene información de la versión del módulo.

Para utilizar estos comandos, a lo largo de este apartado se van a desarrollar los 8 primeros, de forma que cuando accionemos cada uno de los 8 botones, que se sitúan en una de las PCBs anidadas a la MachXO2, se genere un comando asociado. El resto de comandos no implementados tienen un procedimiento idéntico de generación y lectura.

7.1. Comando AT de test

Para comenzar, se va a enviar el comando "AT" (test) por el puerto serie a nuestro módulo. Para ello deberemos generar dos pulsos (uno por carácter) separados con una distancia suficiente para que no se superpongan los mensajes, es decir, se debe desarrollar un **Generador de Pulsos**.

Para ello, se va a dividir el desarrollo de este generador en dos partes, de tal forma que una parte del circuito genere un pulso cuando detecte que se ha pulsado uno de los botones, mientras que la otra parte genere tantos pulsos como deseemos a partir del primero de ellos (en este caso dos).

La primera parte es, por tanto, un bloque que genera un pulso cuando detecte un cambio en un botón determinado, siendo este cambio la pulsación del mismo, es decir, un flanco de subida. Este bloque realiza una función complementaria, que consiste en la activación del módulo para poder introducir comandos AT. En el funcionamiento normal del módulo, el pin SET debe estar a nivel alto, sin embargo si se quiere introducir comandos AT, el pin SET se debe establecer a nivel bajo, con una anterioridad y posterioridad suficiente (definidas en el datasheet del dispositivo).

Observando la hoja de características del módulo HC-12, se puede observar que es necesario establecer el pin SET a nivel bajo con una antelación de 40ms, mientras que se debe mantener dicho estado durante 80ms después de introducir el comando.

Si se calcula cuantas transiciones de reloj se deben generar para cumplir con el tiempo especificado, obtenemos un resultado de que, para un reloj de 9.6kHz (frecuencia predeterminada), se debe configurar el pin SET previamente durante 384 periodos de reloj y se debe mantener durante 768. Para asegurar su funcionamiento se va a añadir alguna transición más, consiguiendo una mayor fiabilidad.

Esta primera parte es un **Detector de la Pulsación,** el cual genera una señal que indica el inicio de la emisión (un pulso), y una señal enviada al pin SET del dispositivo para habilitar los comandos AT. El circuito a generar es un bloque con las entradas y salidas mostradas en la Figura 7.1.1.

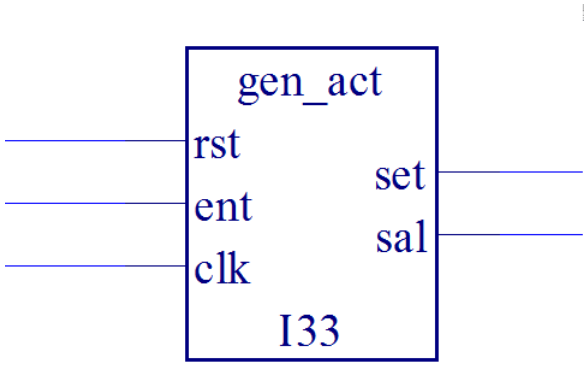


FIGURA 7.1.1. Detector de Pulsación

El circuito ha sido generado mediante código VHDL, teniendo la estructura que se muestra en la Figura A.3.1, localizada en el Anexo A..

La segunda parte del generador de pulsos es muy similar al generador de pulsos realizado en el apartado 6, en donde se generaban cinco pulsos. En este caso solo son necesarios dos pulsos, por lo que se requiere una modificación en el número de pulsos en el código realizado con anterioridad.

El resultado de la unión de las dos partes da como resultado un **Generador de pulsos,** el cual es fácilmente variable para generar diferente número de pulsos. Este generador se muestra en la Figura 7.1.2.

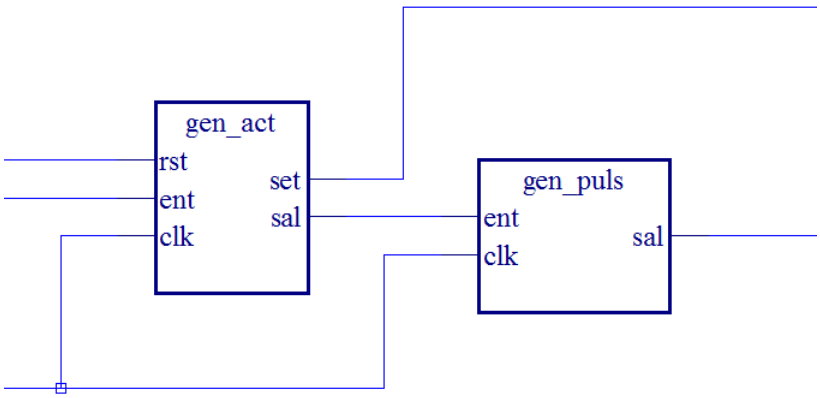


FIGURA 7.1.2. Generador de pulsos variable

Para la comprobación del funcionamiento hemos cargado nuestra FPGA con el circuito anterior, obteniendo un resultado tal y como se muestra en la Figura 7.1.3.

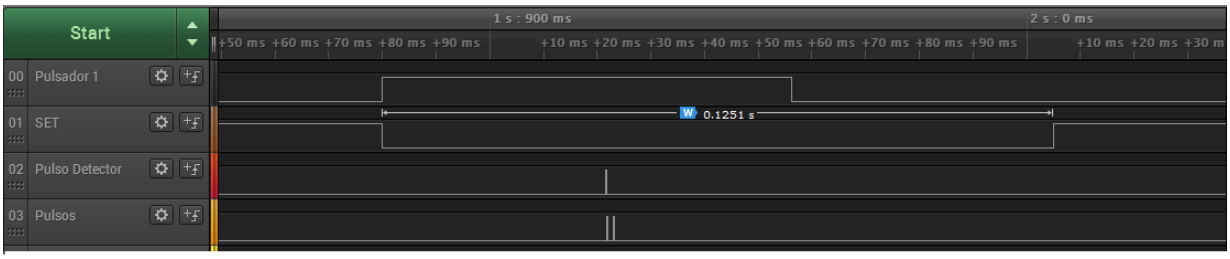


FIGURA 7.1.3. Resultado de la generación de pulsos

En dicha figura se pueden observar cuatro señales. En la primera de ellas, llamada Pulsador 1, se muestra el tiempo durante el cual ha estado accionado el pulsador (accionado por un usuario). En la segunda, llamada SET, se muestra la señal enviada hacia el módulo, la cual nos permite ajustar los parámetros mediante comandos AT. Esta señal tiene una duración de 12ms como mínimo (40ms antes de enviar el primer dato y 80ms después de enviar el último dato), aunque toma valores algo superiores para evitar errores. La tercera señal es el pulso generado por el primer bloque del Generador de Pulsos, es decir, el pulso de inicio. En el caso de la cuarta señal, llamada Pulsos, se muestra los pulsos generados para el inicio de cada mensaje (un carácter por mensaje). En este caso se generan dos pulsos (para 'A' y para 'T')..

Una vez generados dichos pulsos, se pasan como entrada a la máquina de estados finitos FSM, que se ha generado en apartados anteriores, la cual emite un mensaje (almacenado en una memoria) cuando recibe un pulso a nivel alto.

Una posible opción para emitir mensajes es poner la entrada de dicha máquina a nivel alto mientras se desee emitir, pero de la manera desarrollada se puede controlar la separación entre mensajes de una manera más sencilla (bits de parada), asegurando la correcta emisión.

Para poder emitir la información requerida en cada comando AT es necesaria una **Memoria de Test,** que almacene los valores de dicho comando. Partiendo de la memoria realizada en apartados anteriores se ha realizado una serie de modificaciones para que se emita el comando AT de test de manera adecuada. Se ha desarrollado (como en apartados anteriores) mediante la descripción funcional a través del lenguaje VHDL, mostrándose su código en la Figura A.3.2, del Anexo A..

Como se puede observar en dicho código, la variable "regis" toma el valor "10000010 00101010". Este valor ha sido creado partiendo de los códigos ASCII de los caracteres "A" y "T" ("0100 0001" y "01010100" respectivamente). Se puede observar que el orden de los caracteres es el correcto, es decir, primero enviamos el carácter "A" y luego el carácter "T". Sin embargo, el orden de los bits dentro del carácter esta invertido. El motivo de esta inversión se debe a que el módulo de Bluetooth trabaja con un formato Little Endian* (en vez de Big Endian*), es decir, se debe enviar primero el bit menos significativo, concluyendo la emisión con el bit más significativo.

Realizando la emisión de los comandos AT con el formato Little Endian podemos asegurar la correcta comunicación, en caso contrario, el módulo de Bluetooth no entiende el mensaje y devuelve una señal de error.

Como se ha indicado anteriormente, la unión entre FSM y memoria es suficiente para la emisión de mensajes. Esta unión se muestra en la Figura 7.1.4, donde se observa que al igual que en apartados anteriores se ha tomado como reloj de la máquina de estados finitos el inverso al reloj global, para asegurar que se produzca de una manera correcta la emisión del mensaje (actualización del estado en el flanco de bajada y establecimiento de la salida en el flanco de subida).

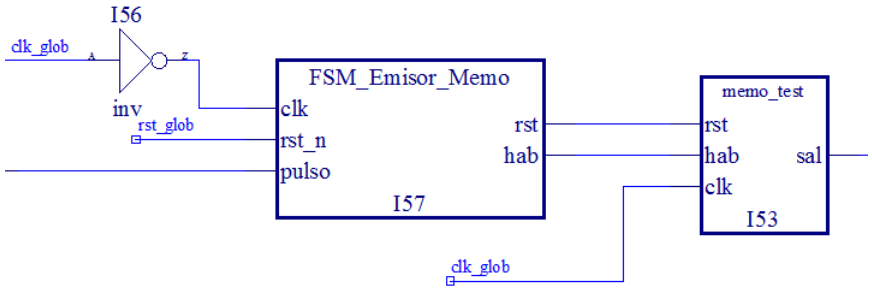


FIGURA 7.1.4. FSM y memoria de test

Una vez hecho todo esto, se procede a la interconexión de los diferentes elementos del circuito. Para ello es necesaria la generación de un reloj y reset común. El reloj será de 9600Hz (velocidad del puerto serie por defecto) y el reset se produce cuando el interruptor 1 se encuentra desactivado (como en apartados anteriores).

El reloj y la máquina de estados finitos ya han sido generados en apartados anteriores, mientras que el resto de componentes han sido realizados específicamente para esta aplicación.

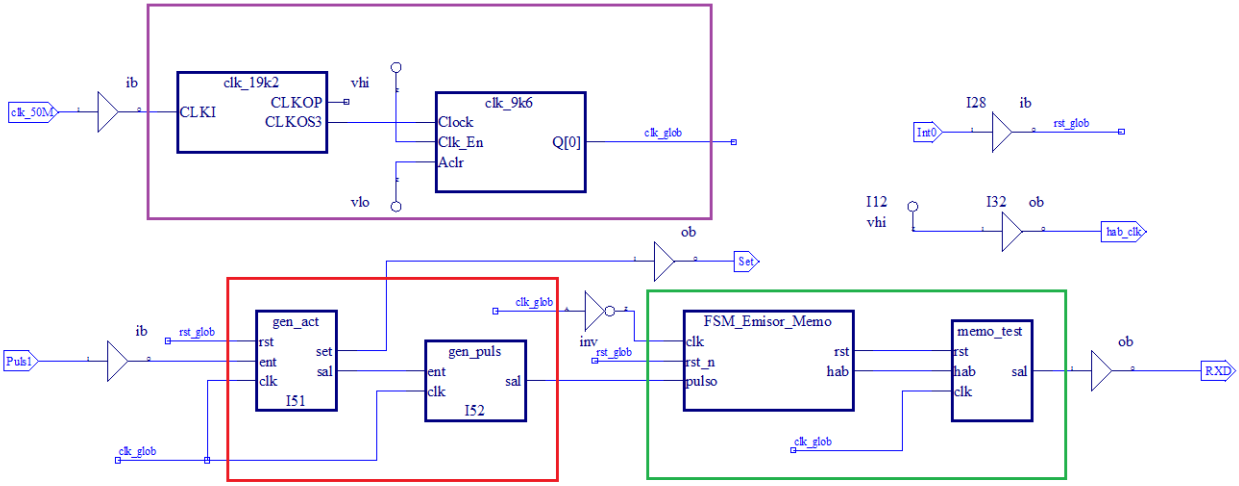


FIGURA 7.1.5. Circuito de emisión del comando "AT" (test)

El resultado se muestra en la Figura 7.1.5, donde se muestra el **Generador del Comando AT** de test. Se puede distinguir en la misma los siguientes componentes:

- Parte morada: Generación del reloj de 9.6kHz.
- Parte roja: Generación de dos pulsos con el objetivo de enviar dos caracteres.
- Parte verde: Generación y emisión de mensajes a partir de los pulsos generados en bloques anteriores. Se observa que la FSM y la memoria de comandos utilizan el reloj invertido (para asegurar la emisión correcta de los mensajes).

Para comprobar el funcionamiento del sistema se debe cargar el circuito en la FPGA, con su posterior estudio mediante un analizador lógico. Una vez realizado, se va a observar los tres pines de interés del módulo HC-12, es decir, el RXD, el TXD y el SET. Estas tres señales se muestran en la Figura 7.1.6.

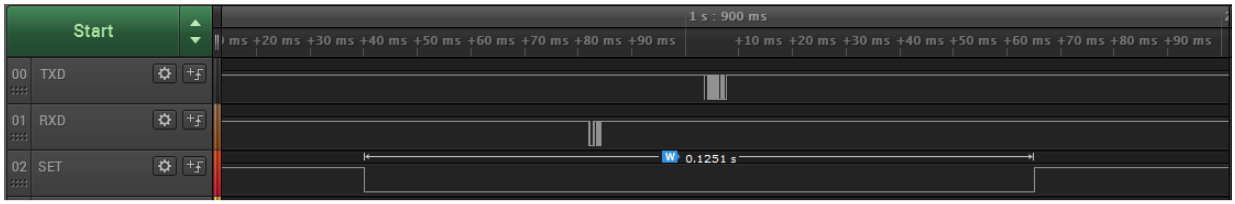


FIGURA 7.1.6. Señales en los pines del HC-12 para los comandos AT

Antes de enviar el comando AT por el puerto serie "RXD" se debe mantener la señal 40ms como habíamos indicado anteriormente. Transcurrido un determinado periodo de tiempo se recibe una señal por el puerto serie "TXD" la cual es la respuesta por parte del módulo, con su posterior interpretación.

La interpretación que se puede dar a las señales recibidas, va a consistir en la transformación de los mensajes recibidos, los cuales están en código binario, a código ASCII. Se puede recibir dos tipos de mensajes:

1. Mensaje de Error: Se produce cuando no se han introducido bien los parámetros, o en el caso de que la configuración no haya sido posible (por ejemplo cuando se intenta configurar al dispositivo con los parámetros ya existentes). En este caso se reciben una serie de mensajes por el puerto serie "TXD" con las características mostradas en la Figura 7.1.7. Si se descifra dicho código (teniendo en cuenta los bits de parada y los bits de arranque) obtenemos que el módulo responde con una serie de caracteres como los siguientes: "ERROR\r\n". Estos dos últimos caracteres son el retorno de

carro (\r) y el salto de linea (\n), juntos indican que se ha finalizado el mensaje y generan una nueva línea en un texto ficticio.

2. Mensaje de Confirmación: En este caso el módulo HC-12 devuelve una información de confirmación de configuración, es decir, confirma que los parámetros introducidos en el comando AT han sido establecidos correctamente. En este caso recibiremos por el puerto serie "TXD" los siguientes caracteres: "OK+...", donde a continuación se especifica los parámetros ajustados. En el caso de comandos AT cuyo objetivo no es ajustar determinados parámetros, sino que tiene como objetivo primordial obtener información sobre el estado del dispositivo, información del mismo o información de determinados parámetros, recibiremos por el puerto serie "TXD" la información codificada en binario de la misma manera que en casos anteriores. La recepción del mensaje "OK\r\n" se muestra en la Figura 7.1.8, siendo esta la respuesta producida al introducir el comando "AT" (test).

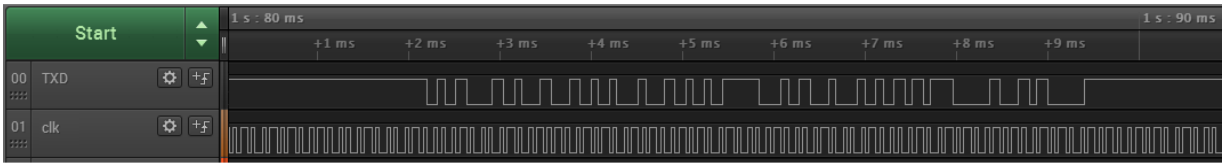


FIGURA 7.1.7. Código de error

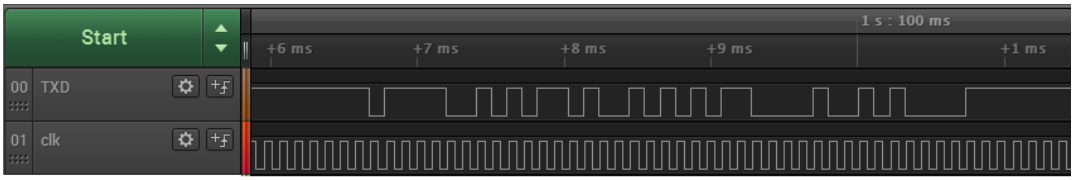


FIGURA 7.1.8. Código de confirmación de test

Descifrar estos códigos es una actividad tediosa, que requiere gran cantidad de tiempo. Únicamente han sido puestos a modo de ejemplo para poder observar lo que se recibe en cada caso. En el caso del código de error (también como ejemplo) se recibe un mensaje con el siguiente código: "10100010 (E) 01001010 (R) 01001010 (R) 11110010(O) 01001010 (R) 1011 0000 (\r) 01010000 (\n)". Recordemos que el módulo trabaja con un formato Little Endian, por lo que si se observa la conversión

entre los diferentes caracteres de ASCII a binario, se obtiene que el orden de los bits está invertido.

Resumen

Con el objetivo de realizar un estudio sobre los comandos AT, se ha emitido el comando de test “AT”, cuya respuesta de confirmación es “OK”. Para ello ha sido necesario realizar una serie de bloques que, partiendo del accionamiento de un pulsador, establezcan las salidas del sistema para emitir el comando en diferentes paquetes.

7.2. Emisión de los diferentes comandos

Una vez comprobado el funcionamiento del módulo en la emisión del comando AT de test y estudiado el funcionamiento de la emisión y recepción, es momento de realizar un circuito en donde los diferentes comandos AT coexistan. Cada uno de ellos se genera en uno de los pulsadores, según el orden en el que fueron explicados. Es decir, la activación de cada pulsador tiene como resultado el siguiente comando:

1. AT.
2. AT+Bxxxx.
3. AT+Cxxxx.
4. AT+FUx.
5. AT+Px.
6. AT+SLEEP.
7. AT+DEFAULT
8. AT+UPDATE

Para realizar una combinación entre todos los comandos se tiene dos opciones. En la primera de ellas se encuentra como solución al problema la modificación de los bloques anteriormente desarrollados parte por parte, es decir, se modifican diferentes partes del circuito para conseguir la sincronización completa entre los diferentes comandos AT.

En la segunda opción propuesta se plantea la realización del circuito como partes independientes, es decir, 8 secciones de circuito independientes que funcionen con un procedimiento similar al circuito del comando AT de test realizado en el apartado anterior. La única dependencia entre los ocho circuitos sería un pin "SET" común. Esta opción presenta diferentes ventajas e inconvenientes:

- Mayor facilidad de desarrollo partiendo del circuito previamente realizado.
- Comprensión sencilla del circuito.
- Circuito más visible y esquematizado.
- Gran despilfarro de los recursos de la FPGA (utilizaremos alrededor de 6 veces más recursos que en la primera opción).
- Necesidad de realizar comunicación entre diferentes bloques para no emitir diferentes comandos AT simultáneamente o durante un mismo periodo de tiempo (SET a nivel bajo).

Se ha decidido por tanto la realización del circuito a través de la primera opción ya que la sincronización que se debe hacer en la segunda opción resulta de una mayor complejidad. Se debe por tanto realizar modificaciones en dos partes del circuito:

- En el generador de pulsos variable, en el cual debe especificar cual de los pulsadores se ha accionado y debe generar un número específico de pulsos dependiendo del pulsador accionado en cada momento (ya que los diferentes comandos AT tienen diferente tamaño).
- En la memoria que contiene los comandos AT. En esta memoria se deben añadir los diferentes comandos para que se pueda emitir una información distinta dependiendo del pulsador accionado. Para ello es necesario también partir de una señal que indique cual de los botones ha sido pulsado (señal procedente del generador de pulsos variable).

Se puede observar que estas dos partes del circuito van a cambiar desde el punto de vista de entradas y salidas (puesto que se necesita una ampliación de las mismas). Estos cambios se ven plasmados tanto en la Figura 7.2.1, donde se muestran los bloques de ambas partes, como en los códigos en VHDL realizados para que cumplan con su funcionamiento, los cuales se muestran en las Figuras A.3.3 y A.3.4 (Generador de pulsos variable y Memoria de comandos respectivamente), situadas en el Anexo A..

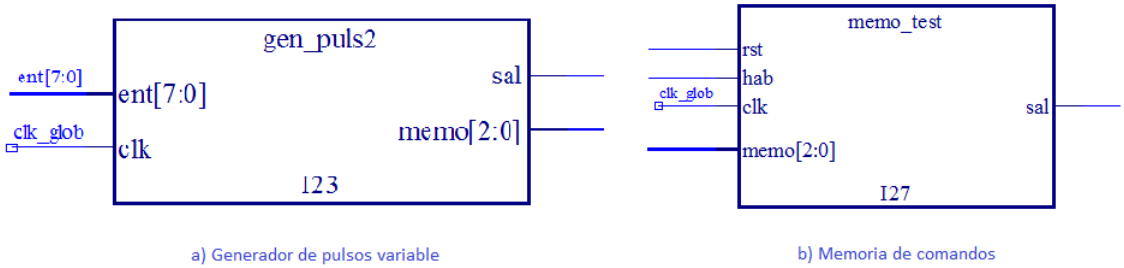


FIGURA 7.2.1. Bloques modificados para el generador de comandos

Es decir, el Generador de Comandos AT tiene:

- Tantos bloques de detección de pulsación como pulsadores tengamos (y por lo tanto comandos).
- Una sincronización de los "SET" de cada uno de los pulsadores para poder encaminarlo hacia el pin "SET" del módulo.
- Un generador de pulsos variable.
- Una máquina de estados finitos que nos permita enviar carácter por carácter siguiendo el protocolo establecido por el módulo.
- Una memoria que genere diferentes comandos dependiendo del pulsador accionado en cada caso.

Una vez desarrollado el circuito completo se obtiene un resultado como el que se muestra en la Figura 7.2.2.

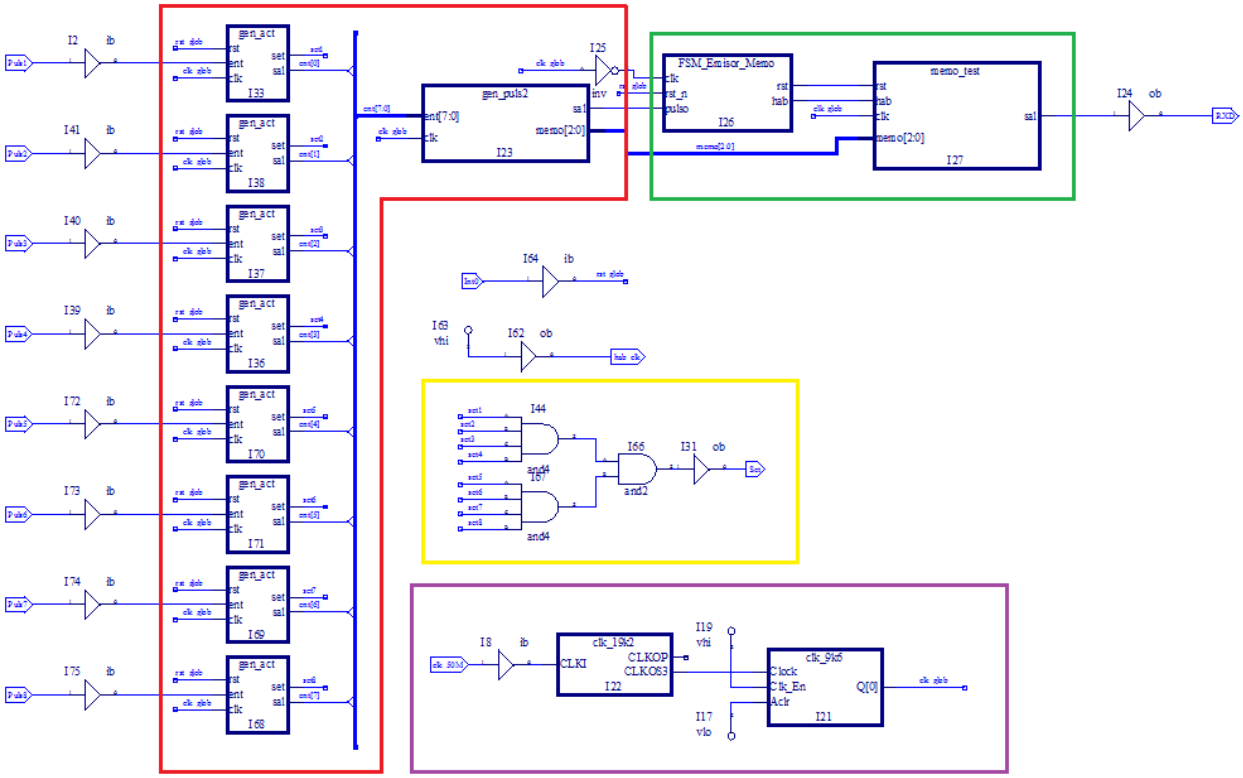


FIGURA 7.2.2. Primera aproximación al circuito generador de comandos

En dicho circuito se encuentran los siguientes componentes:

- Parte morada: Generación del reloj de 9.6kHz (reloj global para todos los componentes)
- Parte roja: Detecta el accionamiento de cada uno de los pulsadores, generando pulsos independientes unos de otros. Posteriormente se tratan dichos pulsos para que, dependiendo de que pulsador haya sido accionado, se genere tantos pulsos (para el arranque de la máquina de estados finitos) como caracteres tenga el comando asociado a dicho pulsador.
- Parte verde: Formada por la FSM (encargada de aplicar el protocolo) y por la memoria donde se guardan los diferentes comandos asociados a cada pulsador, es decir, dependiendo del pulsador accionado se genera un determinado conjunto de mensajes.
- Parte amarilla: Es la encargada de sincronizar las señales "SET" para un buen funcionamiento del módulo en estado de emisión de comandos.

La señal "memo" (formada por tres bits) es la encargada de indicar cual de los 8 pulsadores ha sido el accionado. Gracias a ello se elige en la memoria el comando a mandar.

Para comprobar su funcionamiento se debe cargar el circuito en nuestra FPGA y observar su comportamiento. Para ello se han puesto determinados ejemplos, como son la configuración de los parámetros siguientes: Velocidad por el puerto serie de 19200 baudios (B19200), canal de comunicación 30 (C030), modo de funcionamiento FU2 y potencia transmitida de 8dBm (P4, véase datasheet del módulo HC-12).

Se observa que todos los comandos tienen un resultado positivo, cuando accionamos los diferentes pulsadores, teniendo como respuesta por parte del módulo lo siguiente en cada caso:

1. "OK\r\n" como respuesta al comando de test "AT"
2. "OK+B19200\r\n" como respuesta al comando "AT+B19200"
3. "OK+C030\r\n" como respuesta al comando "AT+C030"
4. "OK+FU2,B4800\r\n" como respuesta al comando "AT+FU2"
5. "OK+P4\r\n" como respuesta al comando "AT+P4"
6. "OK+SLEEP\r\n" como respuesta al comando "AT+SLEEP"
7. "OK+DEFAULT\r\n" como respuesta al comando "AT+DEFAULT"
8. "OK\r\n" como respuesta al comando "AT+UPDATE"

También se puede recibir la respuesta "ERROR\r\n" cuando existe un fallo en la emisión del comando AT, o bien cuando se intenta establecer el valor de un parámetro que ya está ajustado (p.e. cuando enviamos dos veces seguidas el comando AT+DEFAULT)..

Todas las respuestas han sido comprobadas sistemática y cuidadosamente, obteniendo gracias al analizador lógico la respuesta emitida por el módulo y transformando carácter por carácter en binario a código ASCII, un trabajo laborioso pero que nos asegura que el sistema va a funcionar correctamente en un futuro.

Cuando se modifican determinados parámetros, como son la velocidad de comunicación por el puerto serie y el modo de funcionamiento, se modifica la frecuencia del reloj con la que va a trabajar el módulo. Esto supone un problema, ya que al emitir nuevos comandos con la frecuencia de 9600Hz, no existe una compatibilidad con el módulo.

Una solución es generar un reloj variable, que dependiendo del estado del módulo, genere una frecuencia acorde a su funcionamiento. Para ello, y utilizando los comandos utilizados anteriormente, vamos a requerir de la existencia de tres frecuencias distintas:

1. 9.6kHz: Cuando el módulo tenga la velocidad por el puerto serie predeterminada.
2. 19.2kHz: Cuando introduzcamos el comando "AT+B19200", el cual modifica la velocidad por el puerto serie.
3. 4.8kHz: Cuando introduzcamos el comando "AT+FU2", el cual establece como velocidad por el puerto serie de 4800 baudios.

Para realizar el reloj variable es necesaria una interpretación de las respuestas recibidas desde el puerto serie del módulo. Por lo que se deberá diseñar un circuito que sea capaz de tratar la información recibida.

Resumen

Una vez emitido el comando AT de test, se ha ampliado el repertorio para satisfacer todos los pulsadores disponibles de la PCB. Para ello se ha utilizado la misma metodología anterior, realizando una serie de ajustes para coordinar las diferentes señales.

7.3. Tratamiento de las respuestas recibidas

En los casos siguientes se necesita un bloque que se encargue de establecer el reloj correcto en cada momento, con el objetivo de trabajar a la misma frecuencia que el módulo de Bluetooth. Este bloque parte de tres relojes distintos (19.2kHz, 9.6 kHz y 4.8kHz) y de una entrada de selección "s_clk" procedente de otra parte del circuito que interprete las respuestas recibidas. El bloque es un **Selector de reloj** y tiene las entradas y salidas mostradas en la Figura 7.3.1.

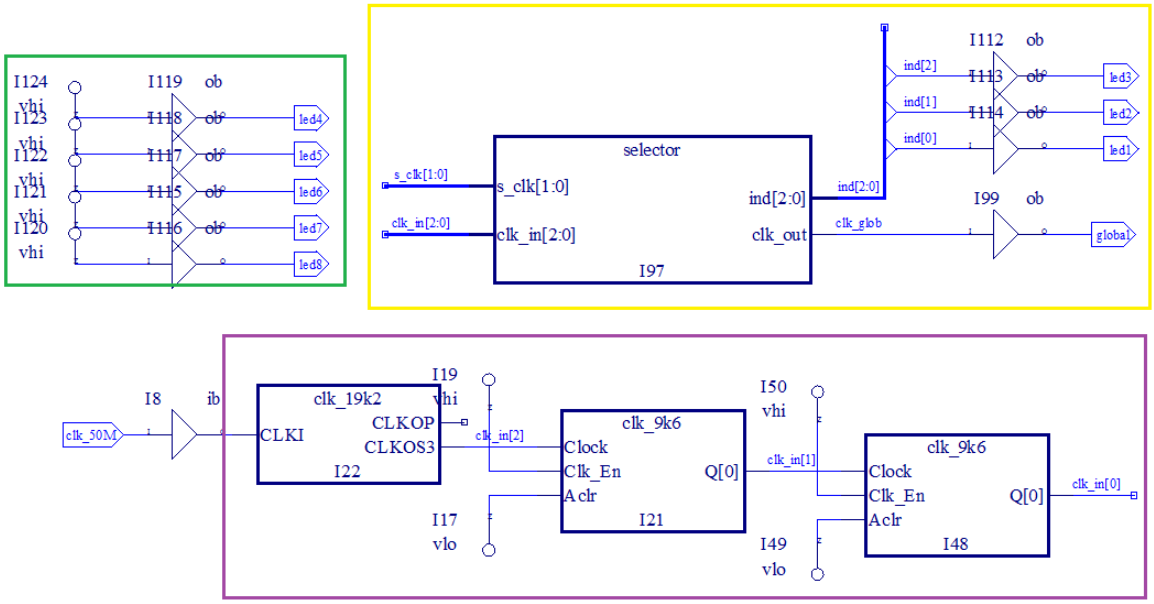


FIGURA 7.3.1. Selector de reloj

El selector tiene las siguientes partes:

- Parte morada: Es la generación de los tres relojes. Partiendo del reloj externo de 50MHz se obtiene un reloj de 19.2kHz tras pasar por el bloque PLL llamado "clk_19k2". El bloque llamado "clk_9k6" fue generado gracias a la herramienta IPexpress y es un contador, el cual cumple la función de dividir la frecuencia de su reloj de entrada a la mitad, por lo que la segunda señal tiene una frecuencia de 9.6kHz y la tercera una frecuencia de 4.8kHz.
- Parte verde: Dentro de la placa MachXO2 se dispone de ocho leds activos a nivel bajo, los cuales se han apagado en esta parte.
- Parte amarilla: Es un selector de frecuencia, funcionando de una manera muy similar a un multiplexor. Este bloque ha sido desarrollado mediante la descripción funcional a través del lenguaje VHDL, siendo su código el mostrado en la Figura A.3.6, localizada en el Anexo A. Dependiendo de la entrada de selección "s_clk" se establece como frecuencia de salida una frecuencia u otra. También se establece el valor de tres de los ocho leds que se encuentran en la placa, de tal forma que el usuario tiene conocimiento de cual es la frecuencia con la que está trabajando el sistema, estando el

led 3 encendido cuando utilicemos una frecuencia de 19.2kHz, el led 2 con una frecuencia de 9.6kHz y el led 1 con una frecuencia de 4.8kHz.

Para el tratamiento de la información recibida del módulo se plantean dos opciones que, a priori, van a ser válidas y van a desempeñar todas las funciones requeridas de una manera eficiente. Estas dos opciones son las siguientes:

1. Mediante la utilización de la máquina de estados finitos desarrollada para el receptor de la información en apartados anteriores, se pueden almacenar los datos recibidos por el puerto serie en un registro de desplazamiento. Posteriormente se procede a tratar la información del modo más eficiente posible, modificando el reloj dependiendo del estado del módulo.
2. Mediante el diseño y desarrollo de dos máquinas de estados finitos complementarias, que traten la información según se reciba por el puerto serie. Al igual que en la opción anterior, se debe modificar el reloj dependiendo del estado del módulo.

La información recibida en ambos casos consiste, dependiendo de la respuesta recibida, en un número de bytes que contienen la respuesta del módulo.

7.3.1. Primera opción (registro de desplazamiento).

Como se ha mencionado anteriormente, a priori las dos opciones son válidas y pueden ser desarrolladas. Por esta razón se comienza desarrollando la primera opción, observando las ventajas e inconvenientes.

Puesto que el tamaño de las respuestas depende del comando introducido, el registro debe tener el tamaño de la palabra con más caracteres, que en los ejemplos de comandos utilizados, es "OK+FU2,B4800\r\n", es decir, 14 bytes de información (112 bits).

Para la realización del circuito en este caso son necesarios los siguientes componentes:

1. Una máquina de estados finitos, FSM, que cumpla el protocolo de comunicación.
2. Un registro de desplazamiento de 112 bits (14 bytes) donde se almacene la información para su posterior tratamiento.
3. Un interpretador de los comandos, que lea la información almacenada en el registro de desplazamiento y realice las acciones adecuadas para su interpretación.

4. Salida de la información. La interpretación de los comandos se muestra por los displays localizados en una de las PCBs anidadas a la placa MachX02 para que el usuario conozca en todo momento que está ocurriendo.
5. Variador de reloj: El interpretador de los comandos indica a este bloque que reloj debe establecerse en cada momento.

Se debe realizar un pequeño ajuste en determinadas partes del circuito del generador de comandos, de tal forma que sea capaz de generar los comandos independientemente del reloj establecido.

No es necesaria la realización del diseño de una nueva FSM que cumpla el protocolo de comunicación, puesto que ya ha sido desarrollada en apartados anteriores y no sufre ninguna modificación. Es necesario realizar un nuevo registro de desplazamiento con el tamaño establecido anteriormente. Se toma como punto de partida el registro desarrollado para la recepción de la información de temperatura y humedad en apartados anteriores, en el cual se establece el tamaño de 112 bits. Se puede observar su código en la Figura A.3.7, localizada en el Anexo A..

Mediante estas dos partes (FSM y registro) se almacena la información recibida por el puerto serie en dicho registro de 112 bits. Es momento de desarrollar el **Intérprete de los Comandos** almacenados. Este bloque debe realizar las siguientes funciones:

- Reconocer diferentes letras para distinguir entre las diferentes respuestas a los comandos.
- Reconocer partes específicas del registro de desplazamiento, es decir, detectar donde está la última respuesta recibida.
- Realizar una distinción entre respuestas y mostrar el resultado por los displays.
- Modificar el reloj según se modifiquen los parámetros del módulo.

Se dispone de un registro de 112 bits, de tal manera que se plantean dos alternativas de reconocimiento de las respuestas. En la primera de ellas se trata de realizar un registro auxiliar en el cual se coloque la última respuesta recibida en los bits más significativos y se comience a leer desde el bit más significativo siempre, para esta alternativa se ha desarrollado el código mostrado en la Figura A.3.8, localizada en el Anexo A..

Como se puede observar, se declaran todos los caracteres que se van a utilizar para descifrar las respuestas del módulo, posteriormente, si hay un cambio en el registro, se actualiza el registro auxiliar para que ambos sean iguales. Una

vez hecho esto, se comprueba si ha llegado una respuesta completa (a través de los caracteres "\r\n" que se reciben cuando una respuesta se ha acabado). En el caso de que esto ocurra, se coloca la respuesta completa en la parte superior del registro auxiliar (bits más significativos). Para ello se buscan los caracteres "\r\n" de la respuesta anterior en el registro (para calcular el tamaño). Posteriormente se procede a comprobar cual ha sido la respuesta recibida, con una asignación de los leds del display para que el usuario sepa cual ha sido el resultado del comando emitido.

Es decir, se dispone del registro mostrado en la Figura 7.3.2.

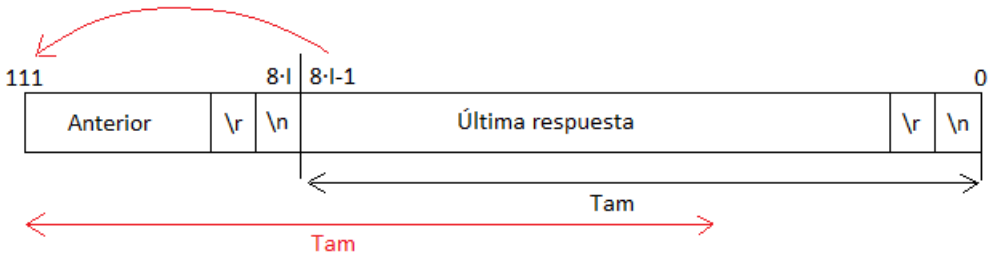


FIGURA 7.3.2. Registro de 112 bits. Primera manera

En esta alternativa de realizar el intérprete se plantea que en el bit más significativo se almacene la última respuesta, de tal manera que siempre comencemos a leer dicha respuesta por dicho bit.

Esta forma funciona correctamente, aunque tiene una desventaja, su alta utilización de recursos. Utilizará el 78 % de los Slices, que unido a la utilización de recursos del resto del circuito forma un 125 % de utilización, por lo que no puede ser utilizada para nuestra FPGA 1200-ZE, se necesita un dispositivo con más capacidad. Su funcionamiento ha sido comprobado con un registro de menor tamaño, pero en ese caso, no funciona correctamente para el cambio de relojes debido a que se desecha información útil en algunas ocasiones. Por esta razón se descarta esta alternativa de desarrollo por su baja eficiencia.

La segunda alternativa de diseño consiste en detectar en que posición se encuentra la última respuesta recibida. Para ello se ha desarrollado el código mostrado en la Figura A.3.10, situada en el Anexo A..

Como se puede observar, al igual que en el caso anterior, se declaran todos los caracteres a utilizar. Sin embargo, se realiza una búsqueda del carácter "\n" de la

palabra anterior. Cuando se encuentra dicho carácter se sitúa un puntero al bit más significativo del carácter anterior (a partir del cual se van a leer las diferentes respuestas). En el caso de no encontrar una respuesta anterior significaría que el registro tiene una única respuesta y por lo tanto el puntero debe situarse en el bit más significativo del mismo.

En este caso, se tiene la situación mostrada en la Figura 7.3.3.

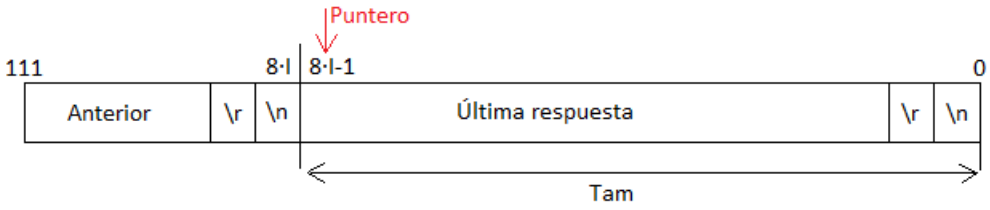


FIGURA 7.3.3. Registro de 112. Segunda manera

Únicamente se localiza el puntero y se recorre el registro para la detección de la respuesta.

Se destaca una ventaja, en este caso se trabaja con un único registro y no es necesario ningún registro auxiliar.

La utilización de recursos va a ser mucho menor que en el caso anterior, es decir, un 28 % de las Slices van a ser utilizadas, que sumado al resto de utilización del circuito forman un 75 % de Slices. Aún así, es un valor muy elevado que nos impide utilizar este método cuando se precisa de más aplicaciones funcionando alternativamente. Hasta el momento consideramos que es una opción válida que funciona correctamente en las situaciones tomadas.

El circuito finalmente obtenido en esta opción se muestra en la Figura 7.3.4.

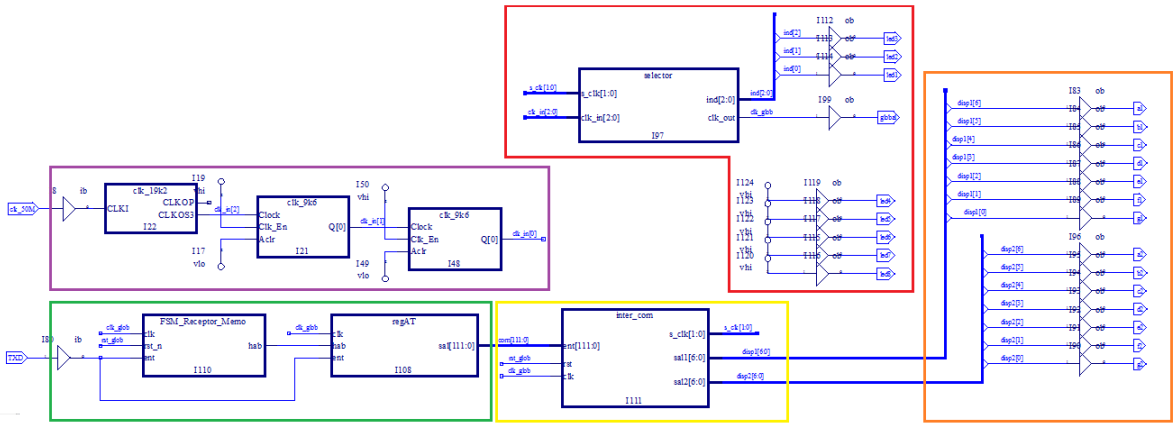


FIGURA 7.3.4. Circuito del Tratamiento de respuestas (Opción 1)

Donde se pueden observar las siguientes partes:

- Morada: Generación de los tres relojes (con distintas frecuencias).
- Roja: Selección del reloj utilizado para el sistema completo y selección del led encendido para la indicación el reloj utilizado.
- Verde: Será la parte encargada de generar un registro de desplazamiento de 112 bits siguiendo el protocolo establecido.
- Amarilla: Es la parte encargada de interpretar el contenido del registro de desplazamiento, generar las salidas correspondientes hacia los displays y establecer el control del reloj utilizado.
- Naranja: Establece la salida hacia los displays.

Dentro de esta opción se planteaban dos manera de realizar el intérprete, concluyendo que la única valida es la segunda (aunque se planteaban una serie de problemas). Ante ambas situaciones se emplea el mismo circuito completo, únicamente varía el contenido del bloque del intérprete.

7.3.2. Segunda opción (mediante dos FSM).

En el caso de la segunda opción planteada, se propone el diseño y el desarrollo de dos máquinas de estados finitos complementarias que, con ayuda de un registro de desplazamiento y un intérprete/controlador formen el circuito completo y cumplan las funciones requeridas. Es decir, el circuito de tratamiento de datos debe tener los siguientes componentes:

1. Una FSM que cumpla el protocolo establecido y sea capaz de controlar un registro de desplazamiento. Esta máquina de estados finitos ya ha sido realizada y utilizada en apartados anteriores por lo que no debe ser diseñada de nuevo.
2. Un registro de desplazamiento de 8 bits: Debe estar gobernada por la FSM anterior de tal manera que almacene los últimos 8 bits recibidos como datos por el puerto serie "TXD". Debe, también, detectar si su contenido coincide con determinados caracteres para su interpretación e indicarlo en caso afirmativo.
3. Una FSM que, dependiendo de las similitudes detectadas por el registro de desplazamiento, indique si se ha recibido alguna respuesta en concreto. Por ejemplo, si el registro detecta una "K" seguida de una "O" se deduce que la respuesta será "OK" o "OK+...".
4. Un controlador/interpretador: Dependiendo las respuestas recibidas indica por los displays los diferentes resultados. Es el encargado de establecer el reloj según la respuesta recibida y de distinguir entre las diferentes respuesta de confirmación.

La máquina de estados finitos que cumpla el protocolo establecido ya ha sido diseñada en apartados anteriores. Sin embargo, el registro de desplazamiento de 8 bits debe cumplir determinadas funciones a la vez que funcionar como un registro como tal, por lo que es necesario su desarrollo mediante el lenguaje VHDL, teniendo como entradas y salidas las mostradas en la Figura 7.3.5.

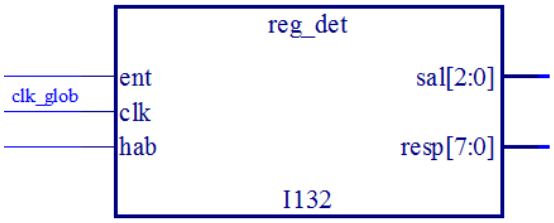


FIGURA 7.3.5. Registro de Desplazamiento de 8 bits y Decodificación de Caracteres

Donde la variable "sal" es un código identificativo del carácter que contiene el registro y la variable "resp" el contenido del mismo.

Este bloque es un **Registro de Desplazamiento y Decodificación de Caracteres**, el cual se va a rellenar con los datos recibidos por el puerto serie

según el protocolo, de tal manera que cuando se reciban bits de parada y arranque el contenido de dicho registro no se va a modificar. Será éste el momento en el cual se compruebe el contenido del registro y se envíe una determinada información a la FSM siguiente que indique cual es el contenido del registro.

Puesto que solo existen tres tipos diferentes de respuesta (OK, OK+..., ERROR), solo es necesario distinguir entre cinco caracteres ("O", "K", "E", "R" y "+"), utilizando un número distinto de identificación para cada uno de ellos. En el caso de no distinguir entre ningún carácter de la lista se envía el valor "000". Esta información es útil para la FSM siguiente para detectar que tipo de respuesta es y como interpretarla.

Los números de identificación de cada carácter se pueden observar en la Tabla 7.3.1. Estos números han sido establecidos de una manera arbitraria, por lo que se debe desarrollar las diferentes partes del circuito siguiendo las pautas establecidas, es decir, siguiendo el protocolo creado al establecer la serie de códigos asociados a cada carácter.

Carácter	Código
Sin coincidencias	0
O	1
K	10
+	11
E	100
R	101

TABLA 7.3.1. Tabla de equivalencias según un protocolo creado personalmente (Carácter/Código)

Cumpliendo con estas premisas se ha desarrollado el código mostrado en la Figura A.3.12, localizada en el Anexo A. Donde se puede observar que en el caso de que no exista cambio en el registro (carácter estable y registro sin cargar datos) se comprueba su contenido y se compara con la lista de caracteres, emitiendo como salida un código asociado al carácter existente.

Si simulamos dicho bloque podemos obtener las señales que se muestran en la Figura 7.3.6.

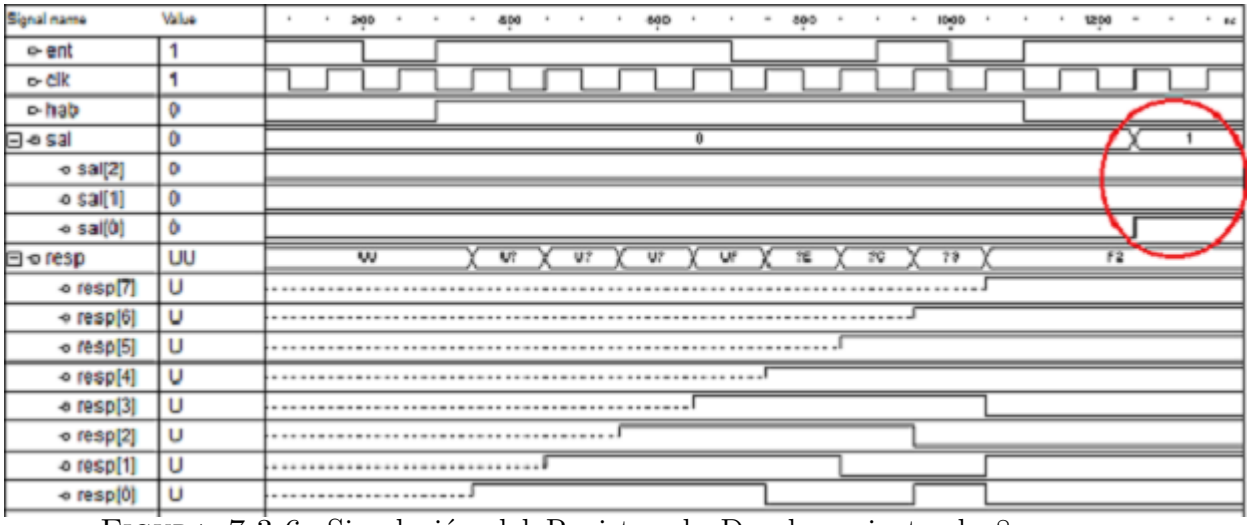


FIGURA 7.3.6. Simulación del Registro de Desplazamiento de 8 bits y Decodificación de Caracteres

Podemos observar en dicha simulación como va llegando el carácter "O" a través de la entrada (puerto serie), siguiendo el protocolo. El habilitador "hab" se activa según manda la FSM previa, la cual ayuda a cargar correctamente el mensaje recibido en el registro. La variable "resp" es el contenido del registro, en la cual podemos observar como se va cargando el valor recibido y como se van desplazando los valores a lo largo del mismo. Lo más llamativo se encuentra en la señal "sal", la cual es la señal que indica el código del carácter que se encuentra almacenado en el registro. Cuando dicho contenido no varía (estamos en bit de parada o de arranque), se actualiza el valor de dicho código, en este caso tomará el valor "001" (código del carácter "O").

En el caso de la **FSM que Detecta Respuestas**, se tiene como entrada el código del carácter detectado en el registro (además del reloj y el reset, como es habitual). Se distinguen dos caminos: El primero de ellos será recibir "OK", mientras que el segundo "ER" (error). En el primer camino tenemos otras dos rutas, en una de ellas se va a recibir el carácter "+" mientras que en la otra no. En caso afirmativo pasaremos a un estado que indique que vamos a recibir un comando específico (B, F, C...), mientras que en caso negativo se concluye que la respuesta por parte del módulo ha sido "OK".

Como salida de la máquina de estados finitos se tiene otro código de identificación, el cual depende de cual ha sido la respuesta recibida. Es decir, dependiendo de la respuesta recibida por el puerto serie se genera una salida u otra para que

el siguiente bloque la interprete y se genere un tratamiento correcto de la información. La salida generada es un código identificativo de cada respuesta, siendo la tabla de la Figura 7.3.2 la que define que código se produce en cada momento.

Tipo de respuesta	Código
Sin cambios	0
OK	1
Con parámetros asociados	10
Error	11

TABLA 7.3.2. Tabla de identificación de las respuestas

Esta FSM se va a desarrollar de tal manera que se prolongue la estancia en un estado mientras la entrada sea la misma (señal de identificación del registro de desplazamiento). Este hecho tiene su motivo en que durante el tiempo en el que el registro está variando, la entrada es constante. Se ha desarrollado por tanto la máquina de estados finitos mostrada en la Figura 7.3.7.

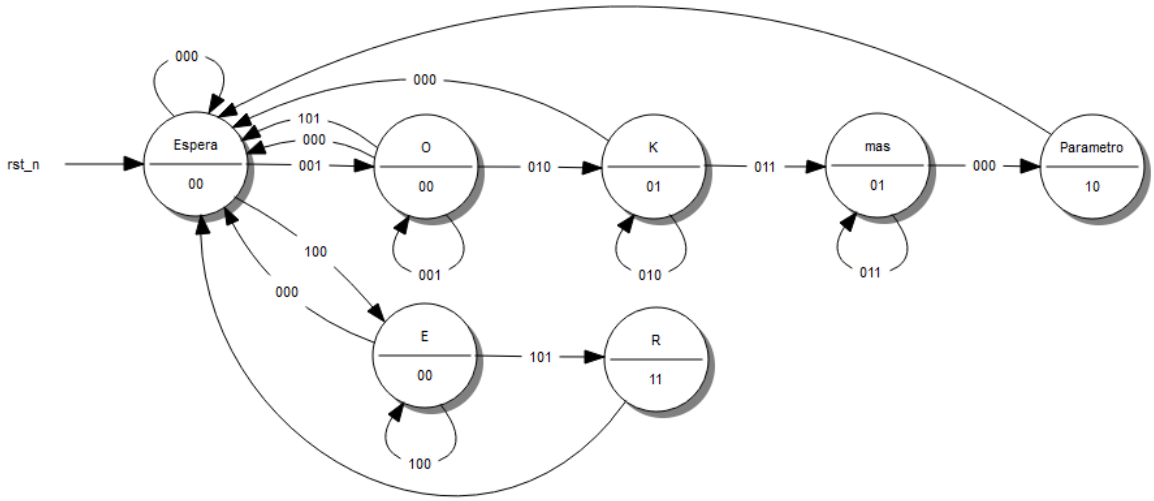


FIGURA 7.3.7. FSM que Detecta Respuestas

Tendremos la siguiente lista de estados:

- Espera: En este estado se encuentra a la espera de recibir uno de los caracteres esenciales y con los que comienza toda respuesta ("O" y "E"). Una vez que se reciba uno de ellos se pasará al siguiente estado, en caso contrario permanecerá en dicho estado. La salida de dicho estado es "00".
- O: La máquina de estados finitos pasa a este estado cuando se haya recibido una "O" por el puerto serie. Puesto que no se puede garantizar que la respuesta recibida sea "OK" la salida seguirá siendo "00".
- K: Cuando se llega a este estado se garantiza que se ha recibido la respuesta "OK" o "OK+..." por lo que la salida será "01".
- mas: Se recibe el carácter "+".
- Parámetro: Se ha recibido "OK+", por lo que se establece la salida como "10" para que bloques posteriores traten al siguiente carácter con el objetivo de conocer la respuesta recibida en su totalidad. La estancia en este estado es de una transición de reloj, por lo que se utiliza un reloj del sistema inverso para que el carácter almacenado en el registro de desplazamiento (el cual indica el parámetro modificado) sea debidamente tratado en el siguiente bloque, el cual funciona con el reloj sin invertir.
- E: No se puede garantizar que la respuesta recibida sea "ERROR", por lo que la salida sigue valiendo "00".
- R: Se garantiza que la respuesta recibida es "ERROR", por lo que la salida vale "11" y se considera al resto de respuesta como información no relevante.

Como ha ocurrido en situaciones anteriores, la máquina de estados finitos se ha desarrollado gracias al programa qFSM, el cual nos permite exportar dicha máquina como archivo en VHDL. De esta forma, se ha creado un bloque que realiza las funciones asociadas a la máquina, la cual tiene las entradas y salidas mostradas en la Figura 7.3.8.

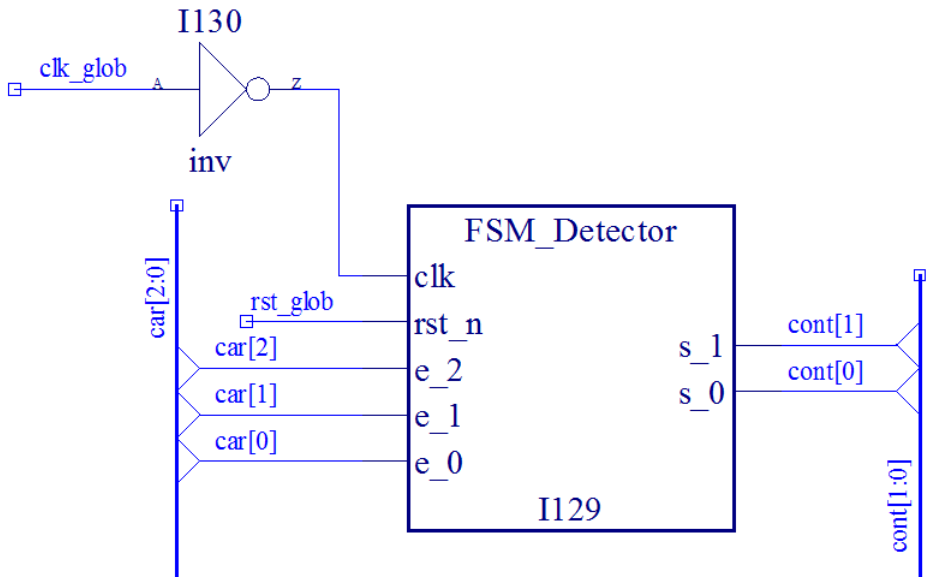


FIGURA 7.3.8. E/S de la FSM que detecta respuestas

Donde se puede observar la inversión del reloj, la entrada de 3 bits y la salida de 2 bits (códigos de entrada y salida). Para comprobar su funcionamiento de una manera sencilla nos hemos apoyado en una herramienta que nos ofrece el programa utilizado, en la cual introduces entradas y observas cual es el efecto sobre la FSM..

Será necesario el diseño de un bloque que, partiendo del tipo de respuesta indicado por la FSM anterior, muestre por los displays cual ha sido dicha respuesta. Por supuesto, tendrá como función prioritaria la modificación del reloj cuando las respuestas recibidas por el módulo lo precisen. Para conseguir todos estos objetivos se deberá desarrollar un bloque que tenga las características desde el punto de vista estructural mostradas en la Figura 7.3.9, formando un **Controlador e Intérprete de Respuestas**.

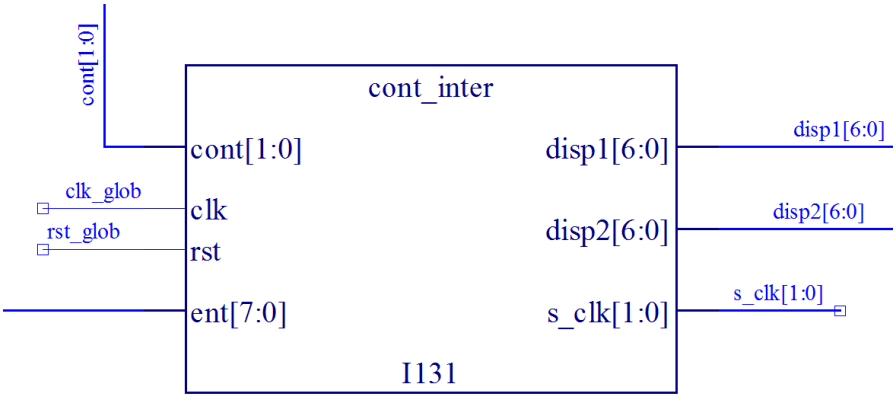


FIGURA 7.3.9. E/S del Controlador e Intérprete de Respuestas

Se puede observar, como entradas significativas, una señal de control llamada "cont" la cual indica el tipo de respuesta y una señal llamada "ent" cuyo contenido es el mismo que el del registro de desplazamiento. Como salidas tenemos, dos destinadas a los displays y una destinada a establecer la frecuencia de reloj llamada "s_clk".

Para cumplir con los objetivos propuestos se ha desarrollado una descripción funcional mediante el lenguaje VHDL. Dicha descripción queda plasmada en el código desarrollado y mostrado en la Figura A.3.13, situada en el Anexo A..

En dicho código se puede observar la declaración de una serie de constantes, las cuales toman valores con el objetivo de facilitar la utilización de los displays y la identificación de diferentes caracteres. En primer lugar se comprueba si se debe identificar un parámetro (estado "Parámetro" de la FSM anterior), en cuyo caso se comprueba cuál es, asignando un valor de salida a los displays y a la señal de control de relojes. Posteriormente se realiza una escritura en los displays para la visualización por parte del usuario.

Para terminar con esta segunda opción es necesaria la interconexión de todos los elementos descritos, obteniendo como resultado el circuito mostrado en la Figura 7.3.10.

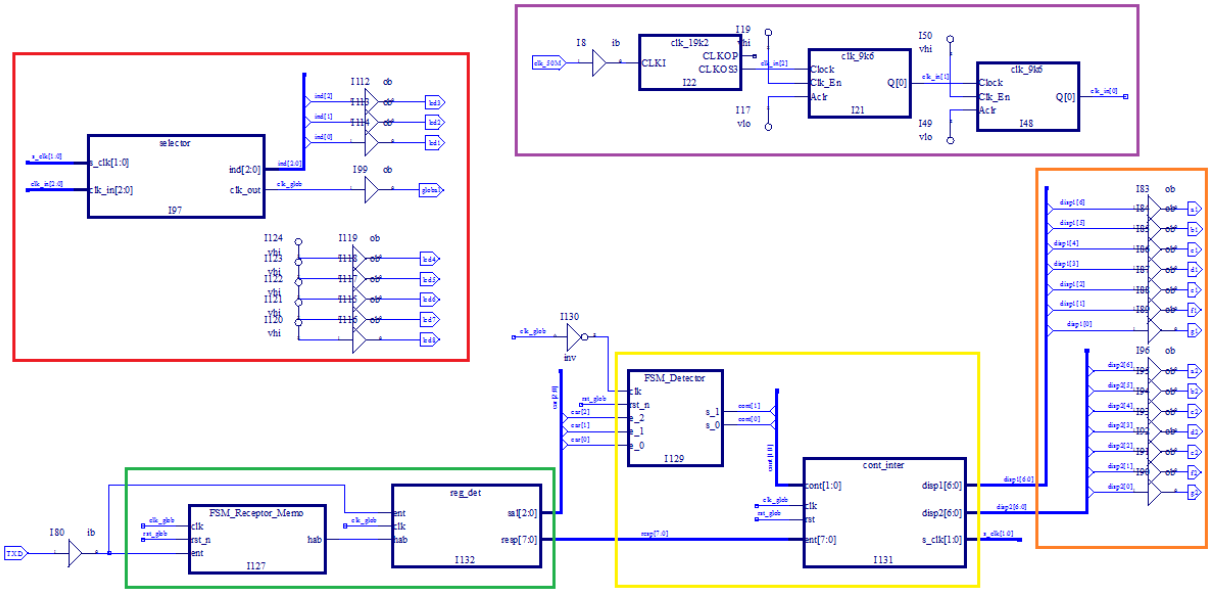


FIGURA 7.3.10. Circuito del Tratamiento de respuestas (Opción 2)

En dicha imagen se muestran los siguientes componentes del circuito:

- Parte morada: Generador de los diferentes relojes.
- Parte roja: Selector del reloj apropiado e indicación del mismo mediante leds.
- Parte verde: Es la parte encargada de almacenar en un registro el último carácter recibido según el protocolo. El bloque de la derecha generará un código de 3 bits dependiendo de su contenido.
- Parte amarilla: Dependiendo del código generado por la parte verde se producirán las salidas apropiadas dependiendo del contenido del registro, de tal manera que el sistema sea capaz de distinguir entre las diferentes respuestas que nos puede dar el módulo HC-12 por el puerto serie. Generará las señales para los displays y la señal de selección del reloj.
- Parte naranja: Establece la salida hacia los displays.

Para comprobar su funcionamiento se procederá a cargar la FPGA con la descripción del circuito desarrollado. Observando que el tratamiento de las respuestas recibidas es el correcto. En este caso el circuito completo va a utilizar un 58 % de las Slices disponibles, por lo que es la manera más adecuada de desarrollar el circuito puesto que su funcionamiento es correcto y es la opción que menos

recursos necesita, mejorando el rendimiento del sistema completo y permitiendo la adición de más elementos al sistema.

Evidentemente, los circuitos relacionados con el tratamiento de las respuestas recibidas no son útiles sin la emisión de comandos AT, por lo que los circuitos de emisión y tratamiento coexisten en un único circuito global.

Resumen

Una vez que se ha conseguido emitir correctamente los comandos AT deseados, y observar cual es la respuesta por parte del módulo a cada uno de ellos, es posible desarrollar un **tratamiento de las respuestas**.
En primer lugar se han desarrollado los **relojes** necesarios para todos los modos de funcionamiento, junto a un **selector del reloj** más apropiado. Después, se han planteado **dos opciones** para realizar el **intérprete de comandos**.
En la **primera opción**, se planteaba la posibilidad de utilizar un **registro de desplazamiento**. Para ello se dispone de **dos alternativas**, en la **primera** se utilizaba un registro de desplazamiento auxiliar cuyo objetivo es situar el último comando disponible en los bits más significativos para su posterior lectura. En la **segunda**, se sitúa un puntero sobre el inicio del último comando disponible para su posterior lectura. La primera alternativa ha sido completamente descartada por su baja eficiencia, mientras que la segunda supone una posible forma de funcionamiento.
En la **segunda opción**, se planteaba el problema a través de diferentes máquinas de estados finitos. En primer lugar, el registro de desplazamiento que contiene la información realiza un estudio sobre su contenido. Dependiendo de ello, se estudia la secuencia de contenidos para conocer posibles comandos. Por último, se produce un tratamiento del comando recibido para su representación a través de los displays.
Esta última opción es la mejor desde el punto de vista de recursos empleados y generalización, consiguiendo una estructura sofisticada y elegante.

7.4. Emisión y Tratamiento de los Comandos AT

Para que el sistema funcione correctamente, a todas las frecuencias, es necesario, además, ajustar el número de pulsos a nivel bajo en los que se encuentra el pin "SET" del módulo. La razón es muy simple, hablábamos en momentos anteriores

que dicho pin debía establecerse a nivel bajo 40ms antes de enviar el comando y 80ms de mantenimiento tras enviar el mismo. Sin embargo, el control de estos tiempos se producía mediante el contaje de transiciones de reloj (384 transiciones para generar 40ms con un reloj de 9600Hz), por lo que el numero de transiciones a contar dependerá de la frecuencia del reloj activo, es decir:

- 384/768 transiciones equivalen a 40/80ms con un reloj de 9.6kHz (Se tomarán 400/800).
- 768/1536 transiciones equivalen a 40/80ms con un reloj de 19.2kHzkHz (Se tomarán 800/1600).
- 192/384 transiciones equivalen a 40/80ms con un reloj de 9.6kHz (Se tomarán 200/400).

Por esta razón, se debe modificar el **Detector de Pulsación**, el cual generaba una señal que asociaba con la activación de los comandos AT ("SET"). Deberemos por tanto añadir como entrada a dicho bloque una señal que indique que reloj está siendo utilizado, es decir, la señal de selección de reloj. El nuevo bloque tendrá la apariencia de la Figura 7.4.1.

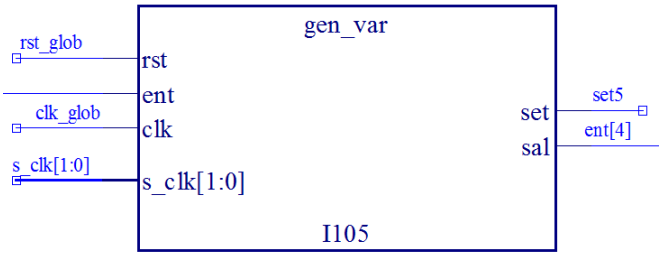


FIGURA 7.4.1. Bloque final del Detector de Pulsación

Donde se pueden observar las mismas entradas y salidas que el bloque inicial, añadiendo la señal de selector de reloj llamada "s_clk". La modificación desde el punto de vista del código se puede observar en la Figura A.3.15, localizada en el Anexo A. En dicho código podemos observar la modificación realizada, distinguiendo entre los diferentes casos de frecuencia de reloj para el cálculo del tiempo a nivel bajo de la señal "SET". El resto de funciones se realizarán de una manera idéntica al bloque anteriormente desarrollado.

Una vez hecho esto se procederá a la interconexión de todos los elementos para que el sistema funcione correctamente, emitiendo comandos AT y tratando

la respuesta del módulo de la manera más adecuada posible. El circuito completo se muestra en la Figura 7.4.2.

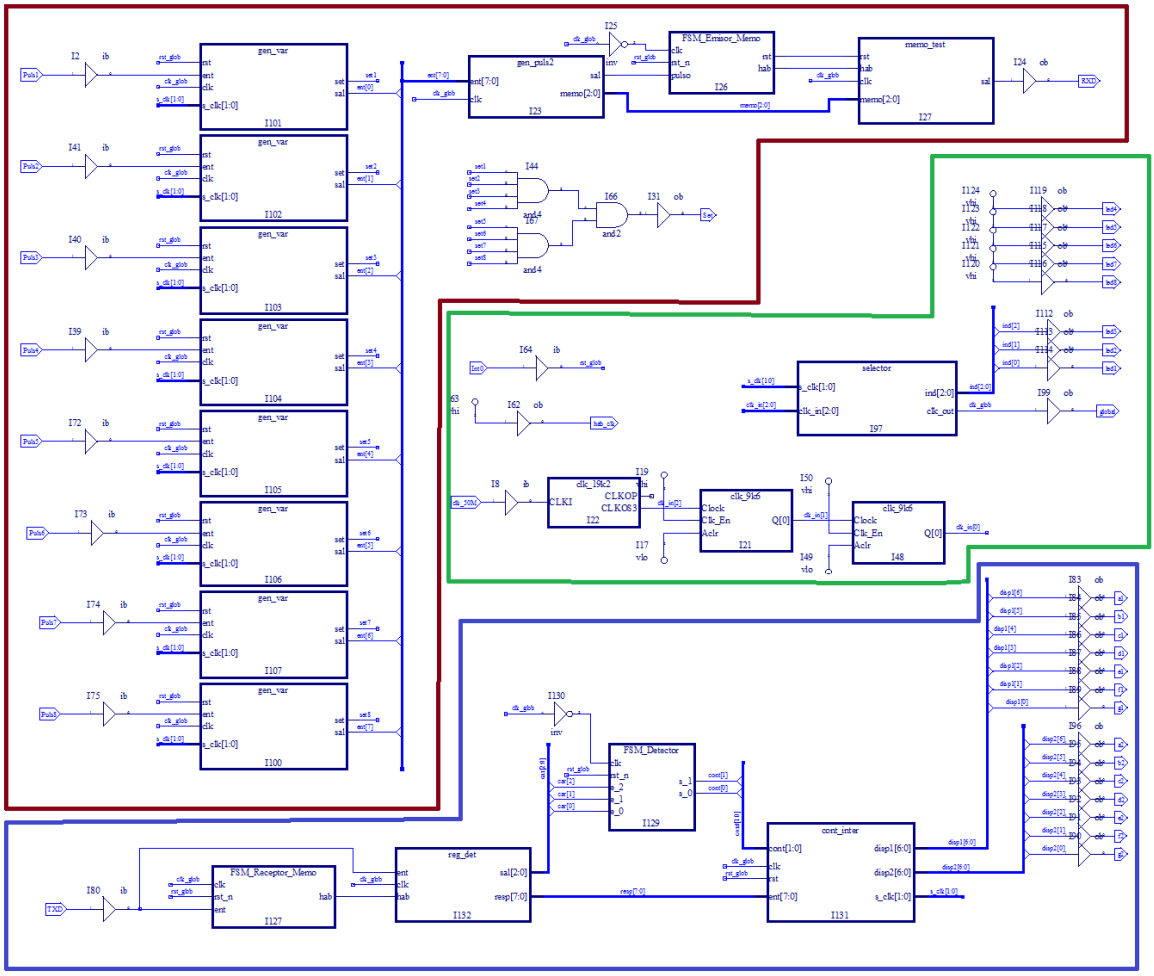


FIGURA 7.4.2. Emisión y Tratamiento de los Comandos AT

Donde se pueden observar tres partes claramente diferenciadas:

1. Parte granate: Es la encargada de la emisión de los comandos AT a través del puerto serie "RXD". La parte del circuito completo que forma esta sección se puede observar mejor en la Figura 7.2.2, donde se debe modificar el Detector de Pulsación. Todo el funcionamiento de esta parte se encuentra explicado en el Apartado 7.2 llamado "Emisión de los diferentes comandos".

2. Parte azul: Es la encargada del tratamiento de la información recibida por el puerto serie "TXD", la cual contiene a la respuesta por parte del módulo a un comando AT dado. Esta parte se puede observar mejor en la Figura 7.3.10, la cual queda explicada a lo largo del Apartado 7.3.2 llamado "Tratamiento de las repuestas recibidas".
3. Parte verde: Se encarga de la generación de los diferentes relojes necesarios, de la selección del reloj conveniente según el estado del sistema y de la generación del reset.

La descripción del circuito ha sido cargada debidamente en la FPGA. Con el fin de comprobar el funcionamiento del sistema se ha realizado un estudio de diferentes señales mediante un analizador lógico. En modo de ejemplo, se muestra en la Figura 7.4.3 la recepción de una respuesta hacia un cambio en la velocidad de transmisión del puerto serie, es decir, tras introducir el comando "AT+DEFAULT" (trabajando el sistema previamente con una velocidad de transmisión de 19.2kHz), se ha modificado dicha velocidad para que trabaje por defecto (9.6kHz).

FIGURA 7.4.3. Cambio en la velocidad de trasmisión

Se puede observar que la frecuencia del reloj cambia mucho antes de que la respuesta por parte del módulo concluya, esto se debe a que una vez recibida toda la información necesaria para saber cual es la respuesta, el resto es desechada (no importa su valor), generándose las acciones necesarias para tratar con eficiencia las siguientes respuestas.

Resumen

Para concluir el estudio sobre los comandos AT, se ha ajustado el tiempo previo y posterior en el que la señal “SET” se encuentra a nivel bajo. Además, se han combinado los circuitos de **emisión de comandos** y de **tratamiento**.

Capítulo 8

Sistema completo

A lo largo de este capítulo se procede a la interconexión de las descripciones funcionales diseñadas y desarrolladas en capítulos anteriores. Para ello se parte de los circuitos generados en los capítulos 6 y 7, donde determinadas señales son comunes. Para la comprensión del mismo se recomienda conocer el funcionamiento de las partes implicadas, es decir, del contenido de los capítulos 6 y 7. Tanto para el emisor como para el receptor se divide el circuito completo en cuatro secciones:

1. Esquemático del reloj.
2. Esquemático del tratamiento de comandos AT.
3. Esquemático de emisión y recepción de los datos de temperatura y humedad.
4. Lógica para el establecimiento de parámetros.

Se realiza por tanto dos circuitos diferentes, el circuito del emisor y el del receptor, los cuales tendrán características propias y comunes.

8.1. Emisor final

Para conseguir un circuito que realice la función deseada es necesario desarrollar cada una de las partes indicadas. Además es necesario combinar diferentes señales, en concreto la señal de salida "RXD", es decir, esta señal tiene dos propósitos diferentes:

1. Para emitir información sobre temperatura y humedad hacia otro módulo receptor.
2. Para emitir un comando AT para cambiar o testear algún parámetro del módulo.

En primer lugar, se realiza una combinación de la circuitería que genera las diferentes frecuencias. En adición a los relojes generados en el tratamiento de la temperatura y humedad, se debe incorporar el reloj de 4.8kHz mediante un

contador que divida la frecuencia a la mitad. Se dispone, por tanto, de frecuencias de 19.2kHz, 9.6kHz, 4.8kHz, 50kHz y 0.5Hz, empleadas con diferentes objetivos.

Se debe incluir el selector de reloj, el cual decide, dependiendo de su entrada, entre las tres primeras frecuencias mencionadas y muestra por los leds de la placa MachXO2 con cual de ellas se encuentra trabajando. De esta manera obtenemos un **Esquemático del reloj**, cuyo circuito se muestra en la Figura 8.1.1.

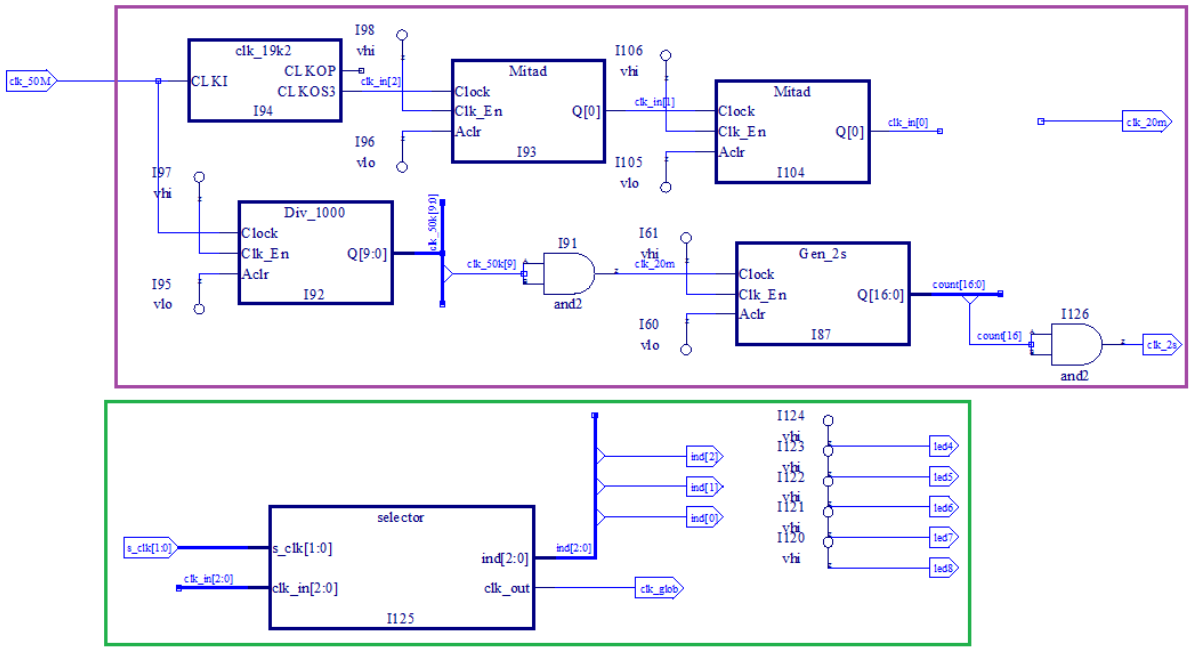


FIGURA 8.1.1. Esquemático del reloj del Emisor

Donde se observan las siguientes partes:

- Parte morada: Generación de los 5 relojes (uno por cada salida de cada bloque).
- Parte verde: Selector del reloj adecuado.

Se puede observar también que las entradas y salidas no disponen de buffer. Esto es debido a que el circuito está destinado para su inserción como bloque en un circuito final y superior en jerarquía. Por esta razón, se incluye el bloque "Esquemático_clk" con las conexiones pertinentes mostradas en la Figura 8.1.2.

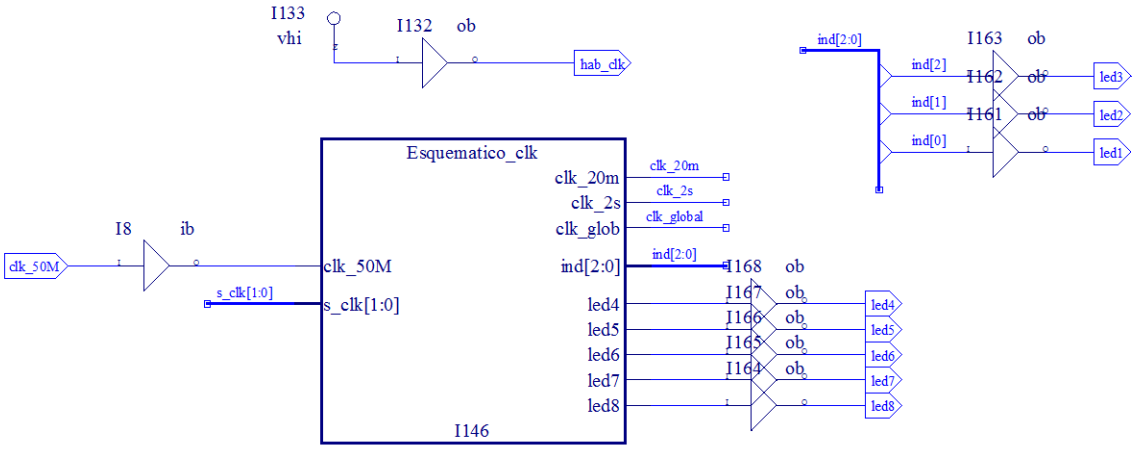


FIGURA 8.1.2. Bloque Esquemático del reloj incluido en el circuito del Emisor

Se puede observar como entrada el reloj de 50MHz y como salidas la habilitación del mismo y la visualización del led seleccionado. Cabe destacar que las frecuencias de 50kHz, 0.5Hz y global son utilizadas por otras secciones del circuito.

El siguiente paso es la realización de un **Esquemático de Comandos**, el cual se basa en el circuito realizado en apartados anteriores. Las modificaciones a realizar sobre el circuito realizado en el Capítulo 7, consisten en la supresión de la generación de relojes, quedando el resto de componentes intactos. El resultado se muestra en la Figura 8.1.3.

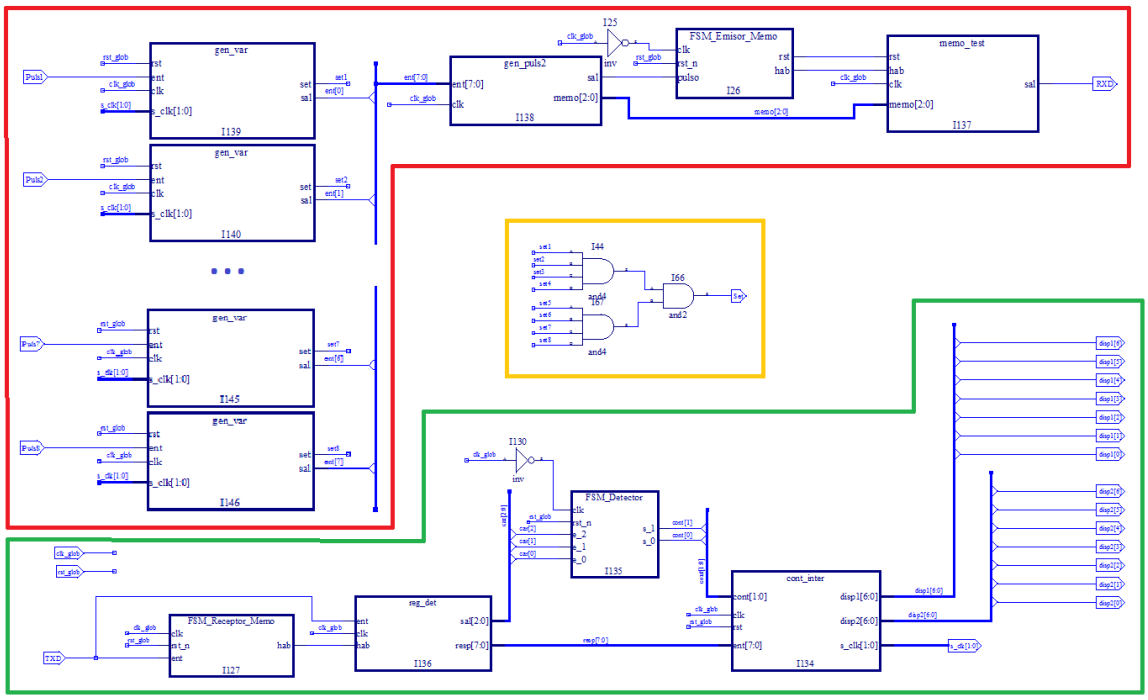


FIGURA 8.1.3. Esquemático de Comandos

En dicha Figura se pueden distinguir tres partes claramente diferenciadas:

- Parte roja: Encargada de la emisión de los comandos AT a través del puerto serie "RXD".
- Parte verde: Encargada del tratamiento de la información recibida por el puerto serie "TXD", la cual contiene la respuesta por parte del módulo de Bluetooth a un comando AT dado.
- Parte amarilla: Combina las señales "SET" de todos los pulsadores para obtener una señal común y poder ser redireccionada hacia el módulo de Bluetooth.

Para el circuito del Emisor se procede a añadir el esquemático de comandos, el cual tiene las conexiones mostradas en la Figura 8.1.4.

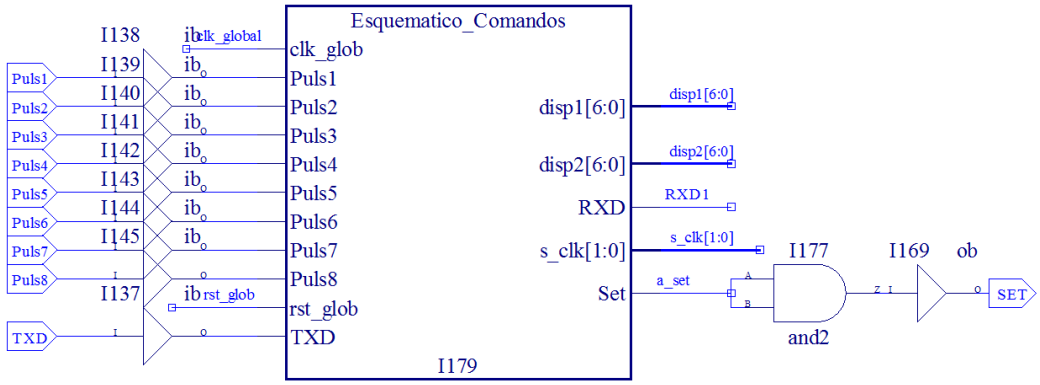


FIGURA 8.1.4. Bloque Esquemático de comandos incluido en el circuito del Emisor

Donde las señales llamadas “disp1” y “disp2” se conducen hacia los displays de salida para mostrar el resultado que ha tenido determinado comando. Otra de las salidas del sistema es la señal “SET”, algo que ocurre de una manera diferente a la señal “RXD1”, la cual no es la única destinada para el pin “RXD” del módulo, por lo que se debe realizar un tratamiento posterior de la misma.

El siguiente paso es desarrollar un **Esquemático del Sensor**, el cual debe leer los datos de temperatura y humedad procedentes del sensor, almacenarlos y enviarlos hacia el módulo de Bluetooth, de una manera idéntica a como se realizó en el Capítulo 6. El circuito tiene la apariencia mostrada en la Figura 8.1.5.

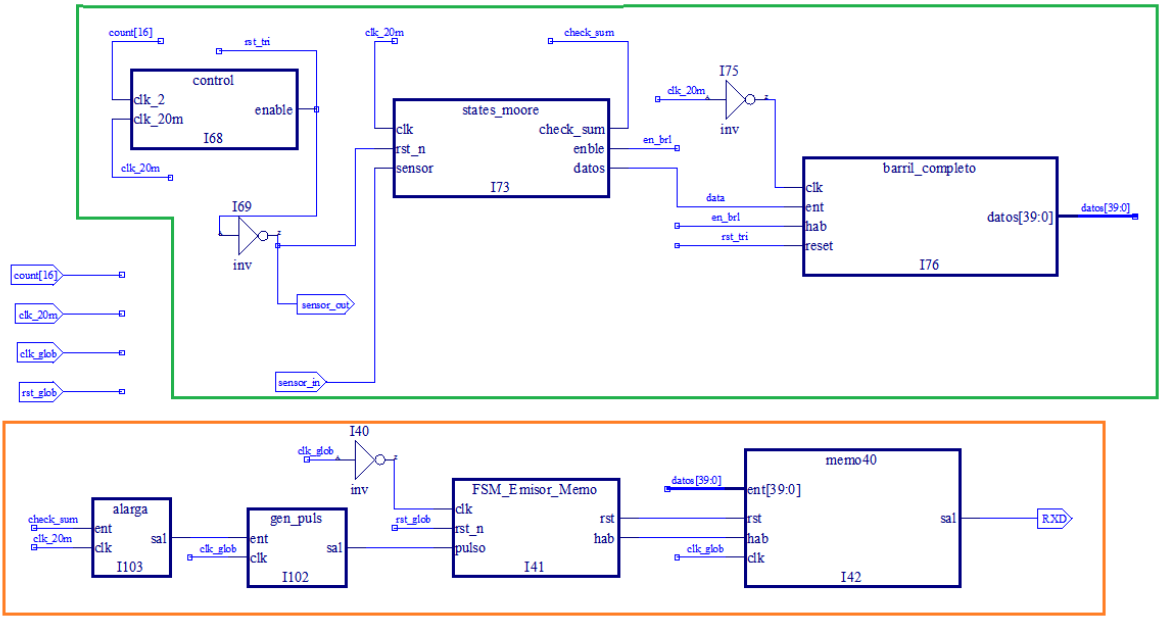


FIGURA 8.1.5. Esquemático del sensor

Puesto que el circuito mostrado forma parte de un esquemático (que será incluido en otro circuito de jerarquía superior), se deben establecer las entradas y salidas sin buffer. Además es necesario sustituir el buffer doble (entrada y salida) procedente del sensor, por dos señales independientes que van hacia el sensor.

El resto de componentes no sufren ninguna modificación, por lo que realizan las mismas funciones que en capítulos anteriores (donde ha sido explicado cada componente).

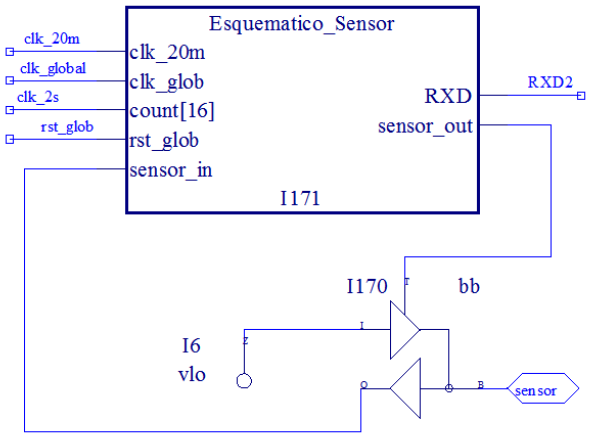


FIGURA 8.1.6. Bloque del esquemático del sensor

Se debe, por tanto, incorporar el circuito anterior como un conjunto, obteniendo un bloque como el mostrado en la Figura 8.1.6, con las señales de E/S, entre las que se destacan el buffer doble indicado anteriormente y dirigido hacia el sensor de temperatura y humedad y la señal "RXD2", la cual debe coexistir con la señal "RXD1" generada en bloques anteriores.

Para finalizar con el circuito del emisor se deben interconexionar todos los esquemáticos desarrollados anteriormente y diseñar la lógica necesaria para que las diferentes señales coexistan. Para ello se ha desarrollado la circuitería mostrada en la Figura 8.1.7.

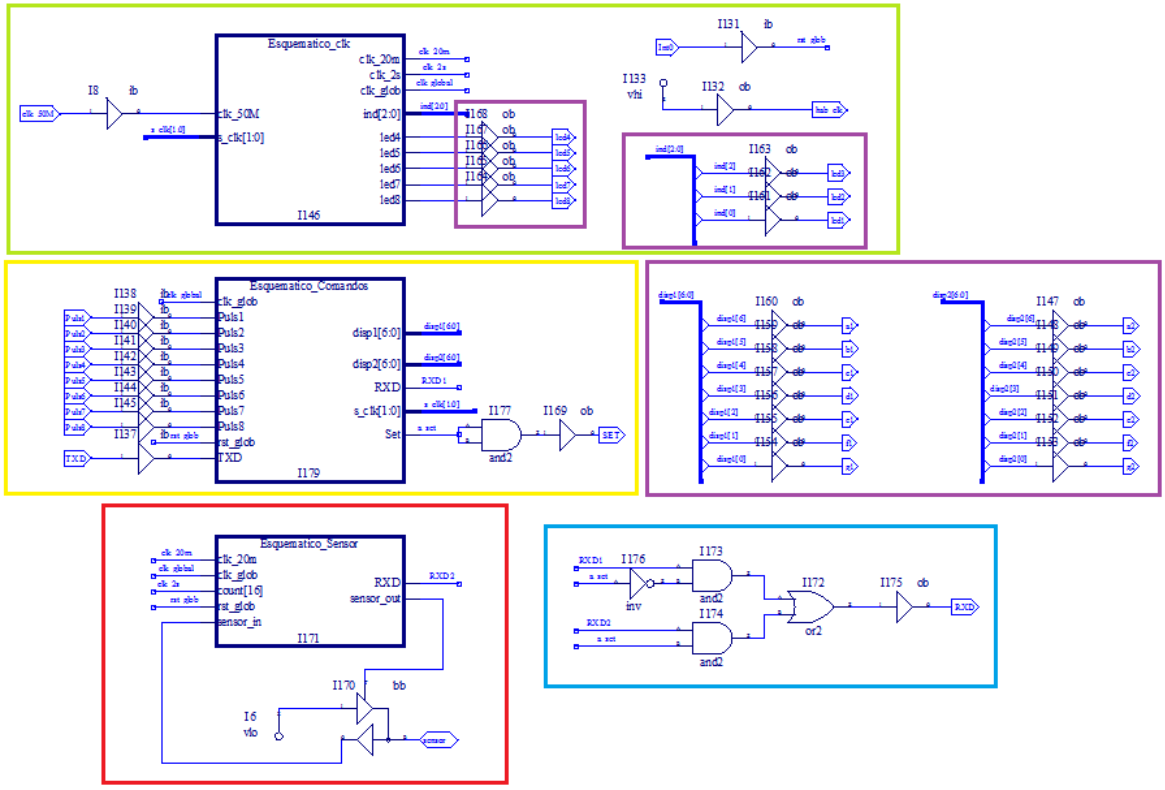


FIGURA 8.1.7. Emisor completo

En la cual se pueden observar las diferentes secciones:

- Partes verde, amarilla y roja: Son los esquemáticos explicados anteriormente.
- Parte azul: Es la lógica combinacional que genera la salida hacia el módulo (señal "RXD"). Gestiona la salida procedente del esquemático de comandos y el esquemático sensor dependiendo del valor de la señal "SET" (si está a nivel bajo la salida será un comando, en caso contrario una emisión de información).
- Partes moradas: Son las salidas relacionas con la representación de información, es decir, los leds y los displays, los cuales muestran el reloj con el que se encuentra trabajando el módulo de Bluetooth y el resultado de los comandos respectivamente.

Para terminar con el circuito perteneciente al emisor solo es necesario comprobar su funcionamiento a través del estudio de las señales del módulo. Para ello se

ha comprobado que el efecto del accionamiento de los pulsadores es el esperado (aplicación de los comandos) y que cada dos segundos se realiza una emisión de datos a través del puerto serie del módulo. El resultado es satisfactorio, asegurando el correcto funcionamiento del circuito diseñado.

8.2. Receptor final

Para el desarrollo del circuito del receptor es necesario el desarrollo de las cuatro partes definidas. Además se debe combinar la señal entrante "TXD", la cual puede tener dos orígenes:

1. El módulo emisor (recibiremos información sobre la temperatura y humedad).
2. El mismo sistema (recibiremos los resultados que ha producido un comando AT dado).

Por esta misma razón, se debe combinar la salida de los displays en la que se puede mostrar tanto información relacionada con la temperatura y humedad como información de los comandos AT..

Como en el caso del emisor es necesario realizar un **Esquemático del reloj**, que a diferencia del anterior, sufre determinadas modificaciones (por ejemplo la supresión del reloj de 0.5Hz, puesto que el mismo se utilizaba para solicitar información al sensor y el receptor no dispone de sensor). El resultado se puede observar en la Figura 8.2.1.

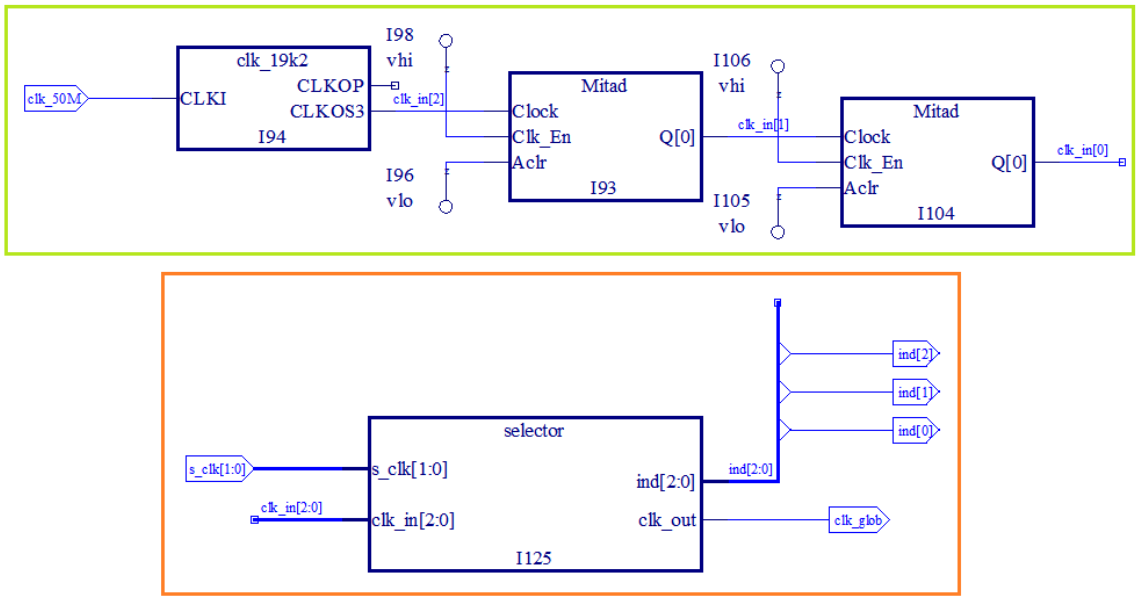


FIGURA 8.2.1. Esquemático del reloj del Receptor

En dicha imagen se puede apreciar la generación de los relojes necesarios en la parte verde (únicamente de frecuencias 19.2, 9.6 y 4.8kHz) y el selector del reloj en la parte naranja.

En este caso, solo se muestra la información sobre que reloj se encuentra activo por tres leds, puesto que se debe mostrar información sobre la temperatura y humedad por otros cuatro.

En el caso del esquemático de comandos utilizaremos el mismo circuito que para el emisor, simplemente se incluyen una serie de modificaciones sobre las señales de entrada y salida al bloque. Para entender estas variaciones, observaremos la Figura 8.2.2, en donde se observa que las señales “TXD” y “RXD” difieren del caso anterior.

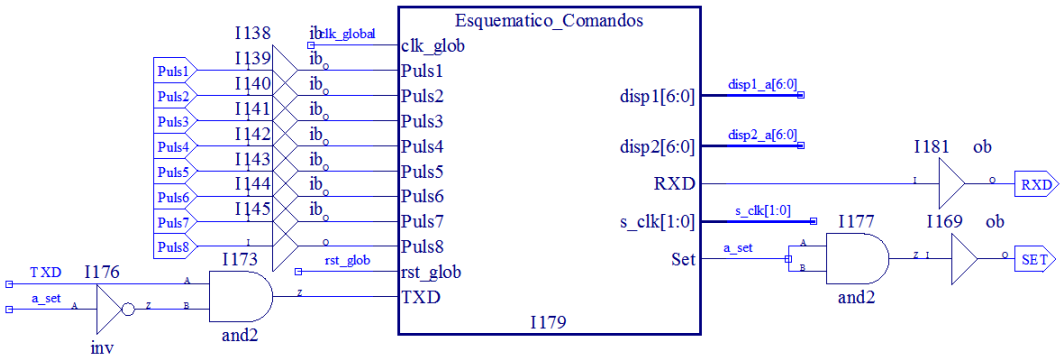


FIGURA 8.2.2. Bloque Esquemático de comandos incluido en el circuito del Receptor

La señal "TXD" en este caso interesa que llegue al esquemático de comandos cuando sea la respuesta de un comando AT, por lo que se debe combinar con la señal "SET" a nivel bajo. Sin embargo, la señal "RXD" será una salida directa hacía nuestro módulo y llevará consigo el contenido de un comando AT dado.

Se debe desarrollar un circuito compatible con el esquemático del sensor, es decir, que realice la función contraria y complementaria. Este circuito será un **Esquemático de tratamiento**, el cual recibe la información procedente de otro módulo de Bluetooth y realiza el tratamiento necesario para mostrar la información por los displays y leds según se desarrollo en capítulos anteriores. El circuito desarrollado se puede observar en la Figura 8.2.3.

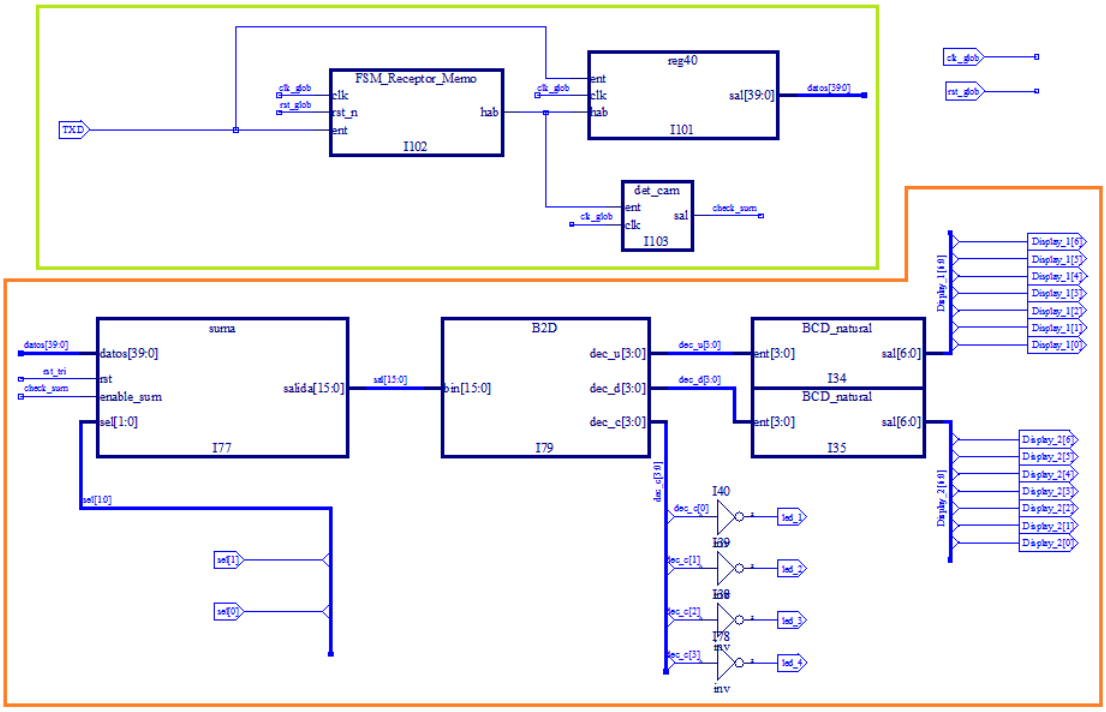


FIGURA 8.2.3. Esquemático de tratamiento

Como sucedía en el Capítulo 6, podemos distinguir dos partes:

1. Verde: Encargada de aplicar el protocolo para almacenar los datos recibidos en un registro.
2. Naranja: Encargada del tratamiento del contenido del registro para su representación por los displays y leds.

En este caso tenemos otra señal “TXD”, la cual debe ser útil cuando la información recibida proceda de una comunicación vía Bluetooth, es decir, con la señal “SET” a nivel alto. Este hecho se puede observar en la Figura 8.2.4.

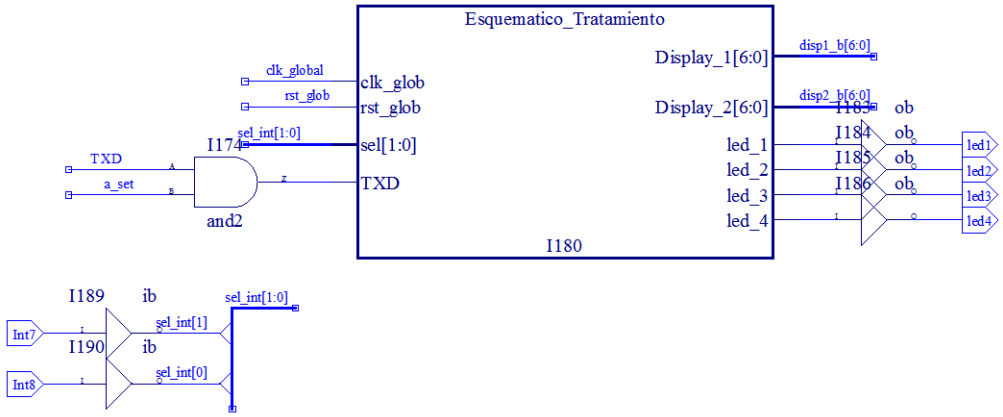


FIGURA 8.2.4. Bloque del esquemático de tratamiento

En el caso del receptor la lógica combinacional se ha desarrollado a partir de la descripción funcional del circuito a partir del lenguaje VHDL. Su objetivo consiste en la combinación de las señales dirigidas hacia los displays (una procedente del esquemático de comandos y otra del esquemático de tratamiento). Para ello se ha utilizado un criterio en el cual la señal procedente del esquemático de comandos tiene una mayor preferencia, es decir, siempre que haya un cambio en la señal procedente del mismo se mostrará su resultado en los displays, durante un tiempo de 3 segundos (en el caso de un reloj global de 9.6kHz).

El código desarrollado para que cumpla los requisitos definidos se muestra en la Figura A.4.1, localizada en el Anexo A. Cuando se detecta un cambio en la señal procedente del esquemático de comandos, esta pasa a mostrarse por los displays, en caso contrario se muestra la señal procedente del esquemático de tratamiento.

Por último, se realiza una interconexión de todos los esquemáticos anteriores (y lógica combinacional). El resultado obtenido se muestra en el circuito de la Figura 8.2.5.

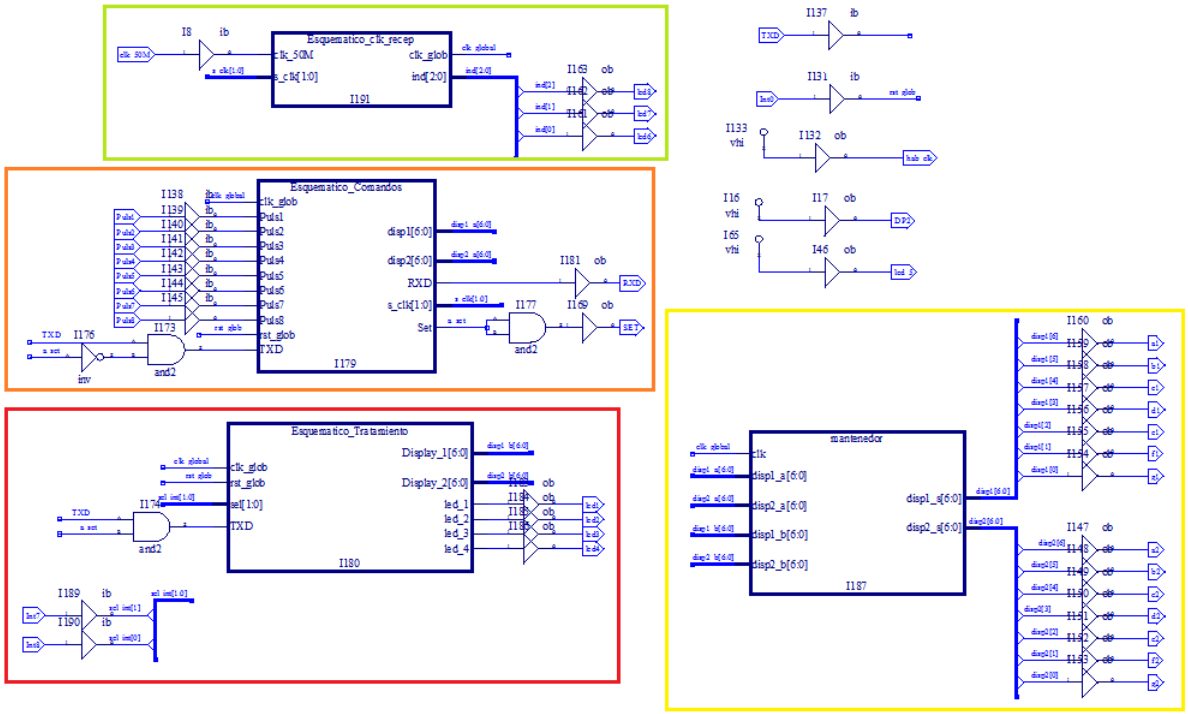


FIGURA 8.2.5. Receptor completo

Donde podemos observar 4 partes claramente diferenciadas:

1. Parte verde: Esquemático del reloj.
2. Parte naranja: Esquemático de comandos.
3. Parte roja: Esquemático de tratamiento.
4. Parte amarilla: Lógica combinacional. Como se puede observar como entrada tenemos las señales de los dos displays procedentes de diferentes esquemáticos y como salida tenemos las señales destinadas a los displays.

Para finalizar con el circuito del receptor es necesario comprobar su funcionamiento junto a otro sistema que funcione como emisor de la información. Para ello se ha de comprobar, en primer lugar, que el efecto del accionamiento de los pulsadores es el esperado (aplicación de los comandos) y que cada dos segundos se reciben determinados datos a través del puerto serie del módulo.

En este caso, cuando realizamos las acciones oportunas para cargar el circuito sobre la FPGA, hemos encontrado problemas relacionados con los recursos necesarios y disponibles. Hasta el momento, el único dispositivo FPGA utilizado

ha sido la placa MachXO2-1200ZE sin ningún tipo de problema de recursos. Sin embargo, para el circuito diseñado para el receptor, el uso de los recursos ha sido del 111 % de las Slices.

Por esta razón se ha decidido sustituir el dispositivo utilizado por una placa MachXO2-7000HE, la cual tiene sobre 5 veces más recursos disponibles. Para ello se han realizado los ajustes oportunos, obteniendo un resultado satisfactorio, asegurando el correcto funcionamiento del circuito diseñado.

Se puede observar la disposición de los elementos emisor y receptor en la Figura 8.2.6, donde el emisor (a la izquierda de la imagen) dispone del sensor de temperatura y humedad, y del módulo de Bluetooth, mientras que el receptor (a la derecha) únicamente dispone de este último.

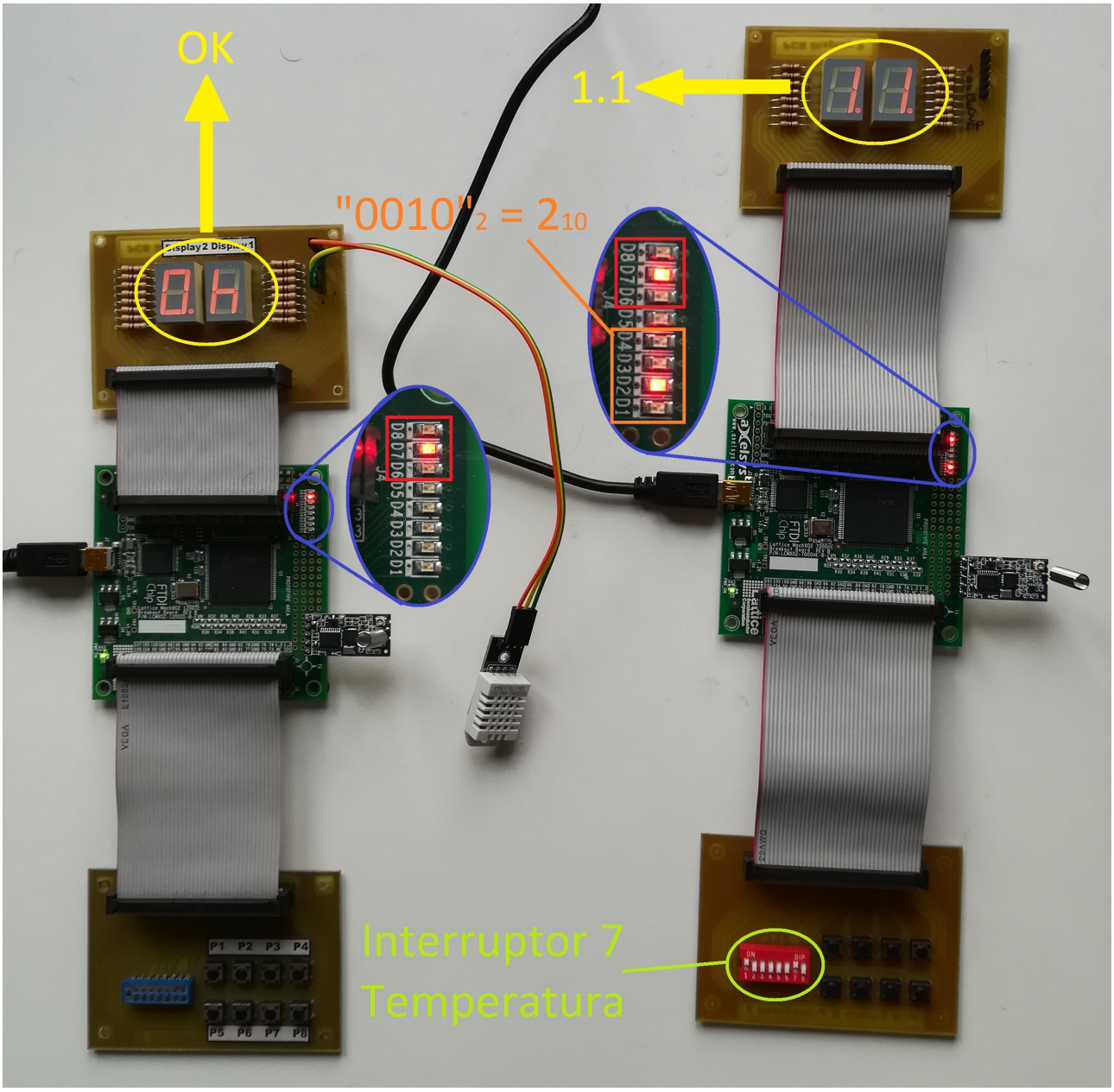


FIGURA 8.2.6. Emisor y Receptor comunicándose

En el emisor se puede apreciar que la respuesta a un comando ha sido "OK" y que el reloj activo se muestra en los leds (activo el reloj de 9.6kHz). En el receptor apreciamos que se muestra la temperatura del ambiente (accionado el interruptor 7), con un reloj activo de 9.6kHz y con una temperatura de 21.1ºC (decenas mostrado gracias a los leds, unidades y décimas mostradas por los displays).

En el caso de que utilicen una frecuencia de trabajo diferente, la comunicación no se realizará de manera satisfactoria. Al igual ocurre con el canal de comunicación, ambos módulos deben trabajar en una misma banda de frecuencia.

Resumen

Con el objetivo de **combinar** la **comunicación entre diferentes dispositivos FPGA** (para la estación meteorológica), y la utilización de los **comandos AT**, se han realizado una serie de esquemáticos para desarrollar dos circuitos compatibles.
Estos son el **emisor** y el **receptor** de datos, los cuales siguen el mismo protocolo y componen la estación completa. En cada uno de ellos se han interconexionado los esquemáticos necesarios y señales comunes a diferentes secciones del circuito.

Capítulo 9

Aplicación móvil

Con el objetivo de observar la información sobre la temperatura y la humedad de una manera más visual, se ha decidido desarrollar una serie de aplicaciones móviles. Se plantean dos aplicaciones similares en las que se pueden realizar diferentes operaciones:

1. **App de Selección**: El objetivo de esta aplicación consiste en seleccionar, o bien temperatura, o bien humedad, para su representación por la pantalla de nuestro Smarphone.
2. **App de Datos**: En esta aplicación se muestra tanto la temperatura como la humedad de diferentes maneras para su visualización.

Ambas aplicaciones han sido desarrolladas gracias a una plataforma online llamada MIT App Inventor*, la cual permite diseñar el interfaz del usuario y desarrollar la lógica interna de la aplicación.

Son necesarias determinadas modificaciones en los circuitos desarrollados en capítulos anteriores, con el objetivo de que el mensaje transmitido tenga el formato requerido por la aplicación.

Es necesario, también, programar las aplicaciones, de tal forma que soporte la comunicación Bluetooth, accediendo al soporte de Bluetooth del dispositivo móvil y enlazándose con el módulo HC-05 utilizado y situado en la FPGA.

9.1. App de Selección

En esta aplicación se proponen una serie de funciones, para que el usuario pueda elegir mostrar temperatura o humedad. Para ello la app debe disponer de dos botones (uno por variable), que cuando sean pulsados transmitan un mensaje hacia el módulo HC-05, indicando qué parámetro se desea conocer. Los datos recibidos por el dispositivo móvil se añaden a una lista que se vacía cada cierto tiempo.

Para poder establecer la comunicación, se debe incluir un elemento de conectividad que funcione como cliente de Bluetooth. Es necesario, también, añadir dos temporizadores, uno de frecuencia relativamente alta para el testeo de mensajes disponibles para recibir y otro con una frecuencia tal, que se active cuando la pantalla del dispositivo se encuentre saturada de datos.

A modo de ampliación, se pueden añadir componentes que hacen a nuestra aplicación más vistosa. Los elementos elegidos han sido, un soporte de reconocimiento de voz y otro de transformación de texto a voz. Con ello se concluye con los componentes "no visibles".

En relación con los componentes "visibles", nuestra interfaz gráfica tiene los siguientes elementos:

- Botones de temperatura y humedad.
- Una lista donde se vayan escribiendo los datos recibidos (con su título).
- Un botón de reconocimiento de voz.
- Botones de Conectar y Desconectar a dispositivos Bluetooth (este último permanece oculto cuando se arranca la aplicación y aparece cuando se conecta un dispositivo).
- Botón de Cerrar la aplicación.

Todos los componentes se pueden observar en la Figura 9.1.1. Para la colocación correcta en los dispositivos móviles es necesaria la inserción de los botones en cuadros especiales.

FIGURA 9.1.1. Interfaz de la App de Selección

Cada uno de los elementos ha sido modificados desde el punto de vista de su tamaño, forma, color, nombre, etc. de tal forma que se mejore su apariencia de cara al usuario.

Para que la circuitería realizada en la FPGA y el módulo de Bluetooth HC-05 envíen los datos deseados por el usuario, se deben realizar determinados cambios en uno de los circuitos diseñados anteriormente. El circuito de partida es el que se muestra en la Figura 6.2.8, en el cual se realiza una toma de temperatura y humedad y se procede a la emisión de los datos.

No se van a enviar los 40 bits de datos (temperatura y humedad), sino que se va a seleccionar la variable solicitada (16 bits) y se va a generar a partir de ella un mensaje (de 56 bits, es decir, 7 caracteres) que es el que se muestra por la pantalla del dispositivo móvil.

Además, se debe desarrollar la lógica necesaria para que se reciba, desde el Smarphone, a través del módulo HC-05, una indicación de la variable que se interesa mostrar. Para ello se recibe un mensaje (de un carácter) que indica temperatura o humedad ("T" o "H"). Cuando la FPGA reciba dicho mensaje debe enviar periódicamente el valor de dicha variable.

Se han desarrollado diferentes partes del circuito final, entre las que se encuentra un bloque **Generador de mensajes**. Este bloque, partiendo del valor de una variable (de 16 bits), genera el mensaje destinado a su envío. Este bloque tiene la apariencia mostrada en la Figura 9.1.2, mientras que su descripción funcional (mediante un código desarrollado en VHDL) se muestra en la Figura A.5.1, localizada en el Anexo A. Este código ha tomado como base el realizado en el TFG "Estación Meteorológica Basada en FPGA", que desarrolla la toma y tratamiento de los datos de temperatura y humedad. Por esta razón se ha señalizado en el código modificado la parte añadida al mismo.

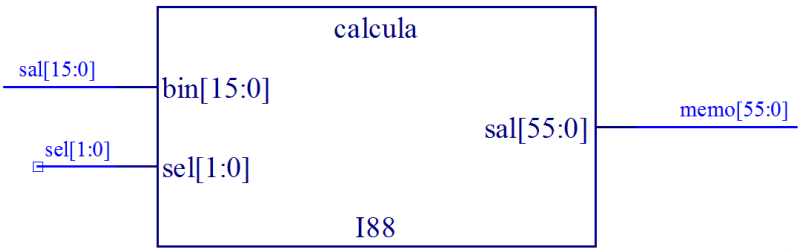


FIGURA 9.1.2. Generador de mensajes

La salida del bloque va destinado hacia una **Memoria de almacenamiento de 56 bits**. Además (y como en casos anteriores) debe generar una serie de mensajes, siguiendo el protocolo utilizado por los módulos de Bluetooth. Por esta razón se ha desarrollado el bloque mostrado en la Figura 9.1.3.

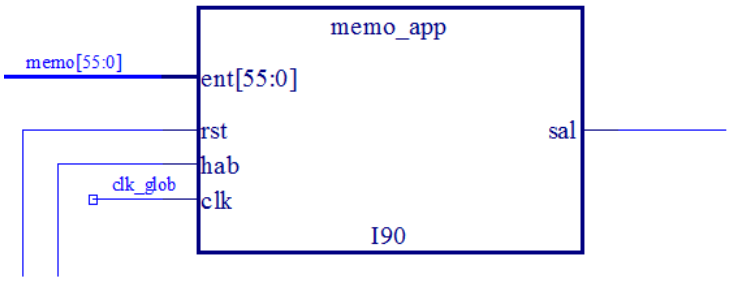


FIGURA 9.1.3. Memoria de la App de Selección

Al igual que en casos anteriores su funcionamiento se ha establecido mediante la descripción funcional mediante el lenguaje VHDL, para ello se ha desarrollado un código como el que se muestra en la Figura A.5.3, localizada en el Anexo A..

Con todas estas partes es posible generar los mensajes y enviarlos por el puerto serie. Sin embargo, no es posible distinguir qué variable quiere mostrar por la pantalla del teléfono móvil el usuario, por lo que se debe realizar un tratamiento de los datos recibidos por parte del módulo de Bluetooth.

Para ello, en primer lugar, se debe almacenar los datos recibidos, según el protocolo establecido, y seguidamente una distinción de los mismos. El único elemento que no ha sido desarrollado con anterioridad es el bloque que distingue entre los datos recibidos y genera una salida que sirva de control para el resto de elementos (dependiendo de su valor se enviará temperatura o humedad).

Este bloque es un **Selector de temperatura o humedad**. Como entradas dispone de 8 bits de datos (será un carácter "T" o "H") y como salida genera una señal de control. Su apariencia se muestra en la Figura 9.1.4, mientras que el código que describe su funcionamiento se puede observar en la Figura A.5.4, localizada en el Anexo A..

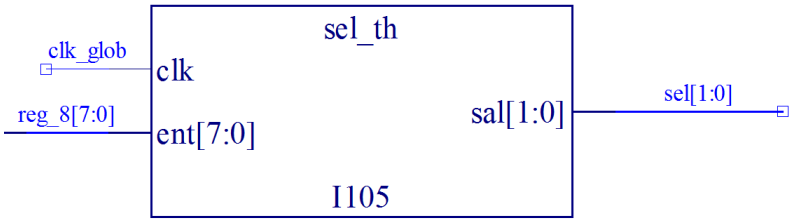


FIGURA 9.1.4. Selector de temperatura o humedad

El último paso para el desarrollo del circuito es la correcta combinación y coordinación de todos las partes realizadas a lo largo del proyecto. El circuito resultante es compatible con la App de Selección, mostrándose en la Figura 9.1.3.

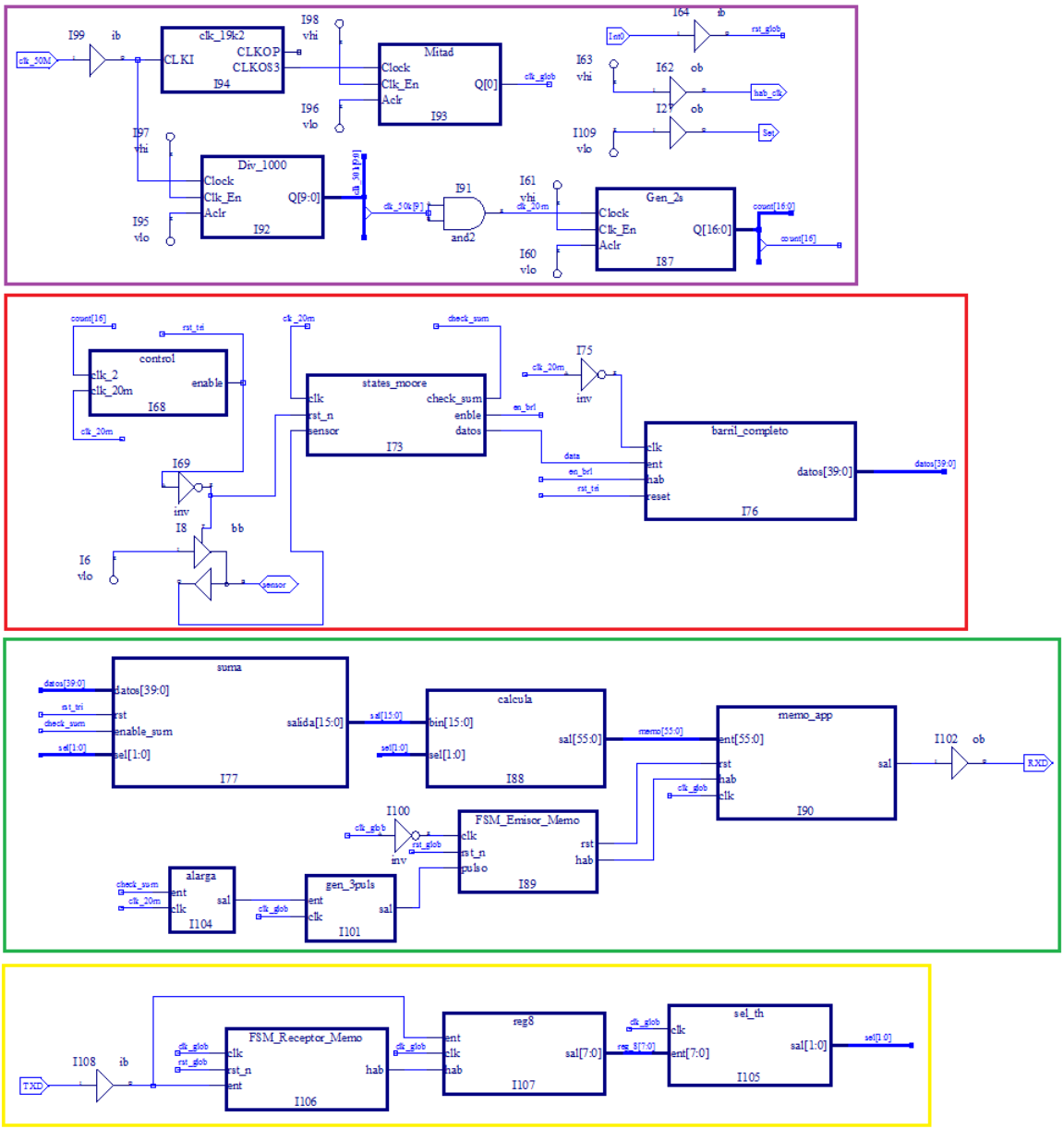


FIGURA 9.1.5. Circuito compatible con la App de Selección

Donde se pueden observar las siguientes secciones:

- Sección morada: Consiste en la generación de las frecuencias necesarias (9.6kHz, 50kHz y 0.5Hz).
- Sección roja: Realiza la toma de datos procedentes del sensor de temperatura y humedad y los almacenará en un registro de 40 bits.
- Sección verde: Encargada de la comprobación de errores, generación del mensaje a enviar y emisión de mensajes por el puerto serie "RXD" cuando se tomen nuevos datos de las variables.
- Sección amarilla: Es la encargada de recibir un carácter por el puerto serie "TXD" e interpretarlo según el protocolo y su contenido para el control de la información enviada.

La lógica de la aplicación debe cumplir unos determinados requisitos para ser compatible con el circuito diseñado. Por esta razón, tiene las siguientes características:

- Cuando arranca la aplicación se solicita mediante voz la activación del Bluetooth en el caso de que no se encuentre.
- Se deberá pulsar el botón "Conectar", el cual, con el Bluetooth activado, nos mostrará una lista de dispositivos posibles para enlazar. Se deberá seleccionar el módulo HC-05. La correcta o incorrecta vinculación con el dispositivo seleccionado se indicará mediante voz.
- Inicialmente el circuito propio de la FPGA no envía ni temperatura ni humedad, sino una pregunta. Se deberá enviar un mensaje hacia el módulo HC-05 indicando que queremos visualizar.
- Deberemos por tanto pulsar uno de los botones llamados "Temperatura" y "Humedad", o bien pulsar el botón "Micrófono" y decir verbalmente las palabras "Temperatura" o "Humedad". La aplicación generará un mensaje ("T" o "H") que será enviado vía Bluetooth hacía el módulo localizado en la FPGA.
- La aplicación no distingue el tipo de mensaje recibido, simplemente se dedica a mostrar en la lista los datos.
- Todas las acciones serán señalizadas mediante voz, indicando lo que se muestra, posibles fallos, conexiones, etc.

En la Figura 9.1.6 se muestra un ejemplo de la utilización de la aplicación desarrollada cuando el dispositivo móvil se encuentra enlazado con el módulo HC-05 localizado en la placa de la FPGA..

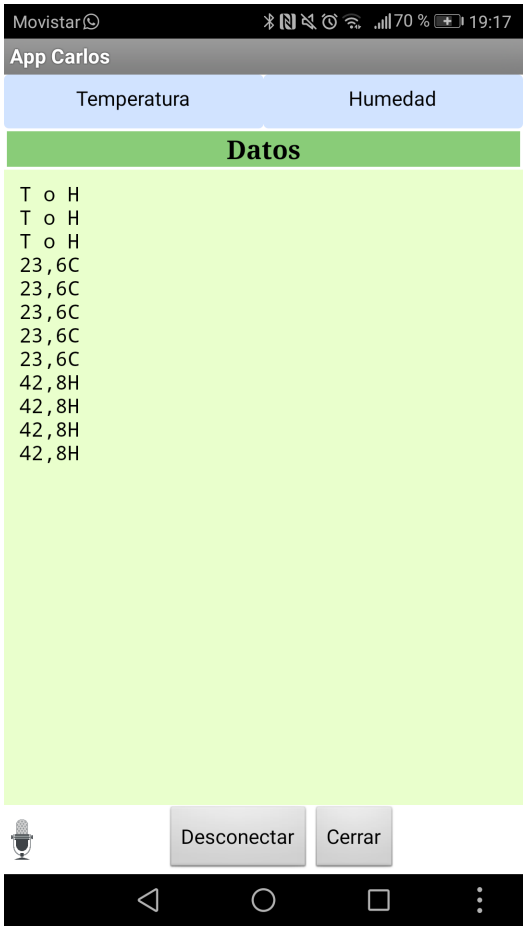


FIGURA 9.1.6. Ejemplo de la utilización de la App de selección

En dicha imagen se muestran los tres casos posibles de información a recibir: Sin petición de información ("T o H"), temperatura y humedad.

9.2. App de Datos

En esta aplicación se propone algo similar al caso anterior, con la diferencia de que se va a realizar una versión más sofisticada, mejorando tanto la interfaz como la lógica interna. Se plantea la disponibilidad de dos pantallas para la representación de información. En la primera de ellas se muestran los valores de la temperatura y humedad actuales y medios con su tendencia actual (desde que se ha iniciado la aplicación), mientras que en la segunda pantalla se observa una gráfica de las variables durante los últimos 200 segundos.

Ambas pantallas son variables, es decir, podremos observar la gráfica de temperatura mientras observamos los valores de humedad actual y media, y viceversa.

Recordemos que para desarrollar la aplicación (utilizando la plataforma MIT App Inventor) se deben incluir elementos "visibles" y "no visibles". Al igual que en la App de selección, se utilizan los siguientes elementos "no visibles":

- Cliente de Bluetooth: Para poder enlazar dispositivos.
- 2 temporizadores: Uno para comprobar la existencia de datos nuevos y otro para lanzar un valor hacia la gráfica.
- Reconocimiento de Voz y Transformador de Texto a Voz: Con el objetivo de reconocer la voz del usuario para navegar por la aplicación e indicar las pantallas, acciones realizadas, posibles errores... a través del habla del dispositivo móvil.

Como elementos "visibles" se han añadido los botones, pantallas, textos, etc., mostrados en la Figura 9.2.1. Donde además se añaden gráficas, botones, textos y demás elementos auxiliares que se encuentran ocultos.

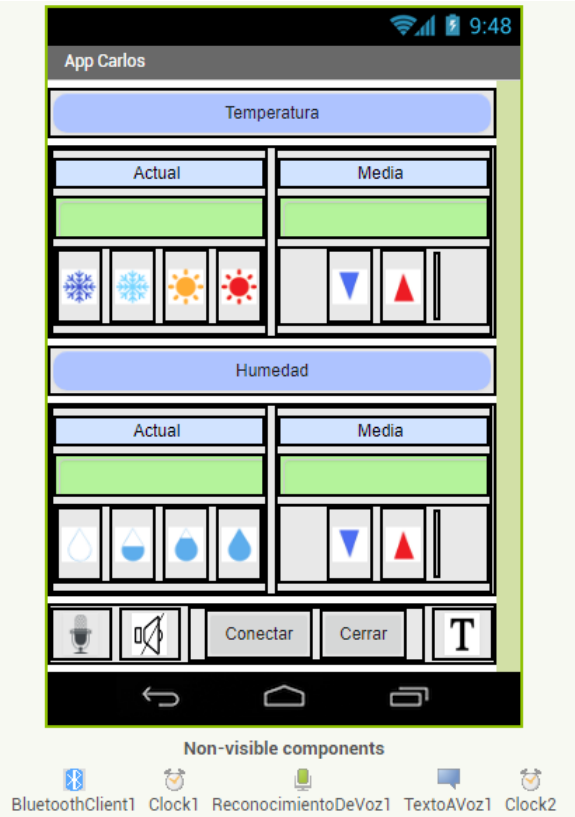


FIGURA 9.2.1. Interfaz de la App de Datos

Es decir, los elementos serán:

- Botones “Conectar” y “Desconectar” a dispositivos Bluetooth.
- Botones de navegación entre gráficas y datos de temperatura y humedad.
- 4 pantallas de representación de datos (con su título).
- Imágenes de señalización de tendencia y rango de variables.
- Botones de “Silenciar” y “Activar Sonido”.
- Botón de “Tutorial” donde se explica mediante voz la utilización de la aplicación.
- Botón de “Micrófono” para la navegación entre pantallas.
- Botón de “Cerrar” aplicación.

Al igual que en la App de Selección, se necesita una modificación en la circuitería incluida en la FPGA origen de la información. La mejor solución encontrada para realizar nuestro circuito, ha sido mediante la alternación entre la emisión

de las variables (temperatura y humedad). Para ello se ha partido del circuito realizado para la App de selección, mostrado en la Figura 9.1.5.

Una forma de alternar ambas variables es mediante el desarrollo de un bloque que funcione como **Alternador**. Este parte de la señal que marca el inicio de la comunicación ("check_sum") y genera dos señales "check_sum" (una para temperatura y otra para humedad), variando en cada una de ellas el valor de la variable de selección (que indica que variable enviar). El bloque tiene la apariencia mostrada en la Figura 9.2.2, y el código que describe su funcionamiento se encuentra en la Figura A.5.5del Anexo A.. Se deben realizar dos modificaciones más, en el Generador de mensajes y en la Memoria de la aplicación. Desde el punto de vista del Generador de mensajes, modificaremos el mensaje enviado, el cual depende de si se envía temperatura o humedad. La modificación se refleja en el código realizado, el cual se muestra en la Figura A.5.6, del Anexo A.

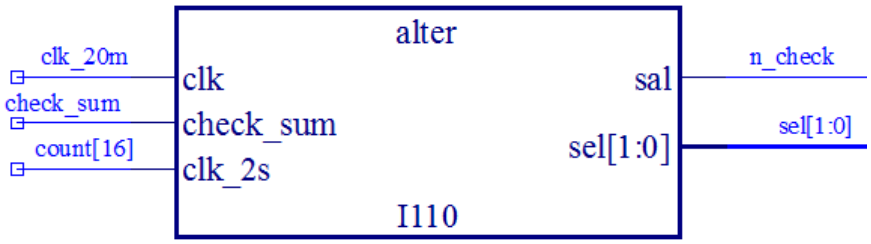


FIGURA 9.2.2. Bloque Alternador

La modificación realizada refleja un cambio en el tamaño del mensaje enviado y en su contenido, es decir, pasará a emitir 6 bytes de información en vez de 7 bytes por bloque de mensajes. Desde el punto de vista de la Memoria de la Aplicación, su estructura y contenido son idénticos a los casos anteriores, con la salvedad de que el tamaño del mensaje varía, por lo que el tamaño de dicha memoria también varia. Su código se puede observar en la Figura A.5.8, localizada en el Anexo A, donde la única modificación se basa en el cambio de tamaño.

El siguiente paso consiste en la combinación de todos los bloques desarrollados para generar correctamente los mensajes, obteniendo un circuito como el mostrado en la Figura 9.2.3, el cual debe ser compatible con la App de Datos desarrollada.

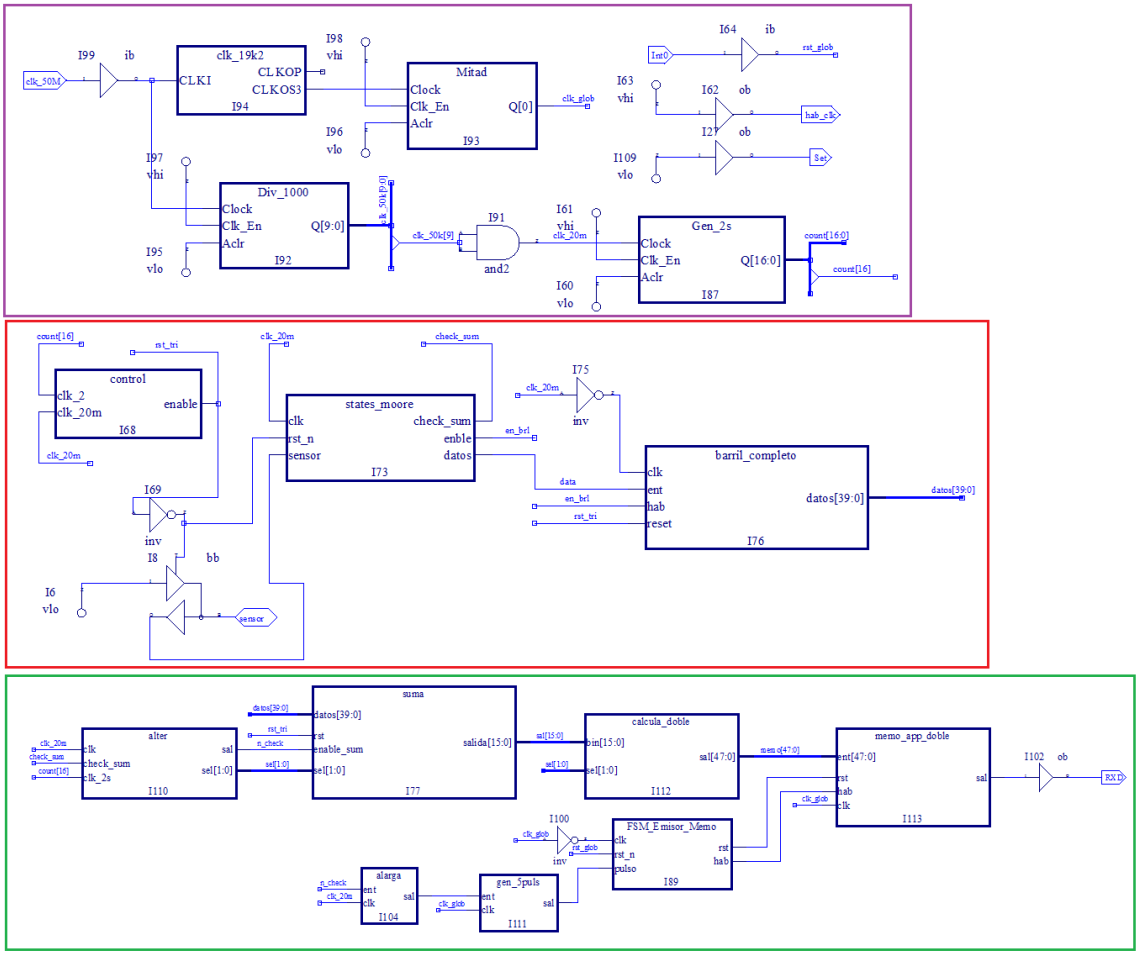


FIGURA 9.2.3. Circuito compatible con la App de Datos

Donde podemos diferenciar entre tres secciones claramente diferenciadas:

- Sección morada: Consiste en la generación de los relojes necesarios (9.6kHz, 50kHz y 0.5Hz).
- Sección roja: Realiza la toma de datos procedentes del sensor de temperatura y humedad y los almacena en un registro de 40 bits (esta parte no ha sufrido ninguna modificación).
- Sección verde: Alterna entre la emisión de dos mensajes diferentes, para lo cual emite un mensaje de 6 bytes (6 paquetes de datos independientes enviados según el protocolo del módulo de Bluetooth).

Para garantizar la compatibilidad aplicación/circuito y para que la App sea lo más sofisticada posible, la lógica de la misma tendrá las siguientes características:

- Cuando se inicia la aplicación se solicita, mediante voz, la activación del Bluetooth en el caso de que no se encuentre ya.
- Para enlazar nuestra App a cualquier dispositivo, se debe pulsar el botón "Conectar", el cual, con el Bluetooth activado, nos muestra una lista de dispositivos (normalmente se debe enlazar al módulo HC-05). La correcta o incorrecta vinculación con el dispositivo seleccionado se indica mediante voz.
- Una vez conecta a un dispositivo se muestra el botón "Desconectar", ocultando el botón "Conectar". Mientras haya un dispositivo emisor funcionando, los datos se reciben, refrescándose la información mostrada en las pantallas constantemente.
- En la parte inferior de la aplicación se muestran diferentes botones auxiliares.
- Cuando se recibe un dato, la aplicación distingue entre temperatura o humedad según las características del mensaje. En el caso de que el mensaje tenga una estructura correcta, se procede a refrescar la información cada periodo de tiempo establecido por los temporizadores. Uno de los cuales establece el momento de volcar la temperatura y humedad actual sobre las gráficas correspondientes.
- La temperatura y humedad actual se muestra en uno de los displays. Al igual ocurre con la media, la cual es calculada desde el inicio de la aplicación.
- Las gráficas disponen de 10 puntos de datos, es decir, 10 elementos que se van desplazando por dichas gráficas según transcurre el tiempo. Se representan los datos de los últimos 200 segundos. Además disponen de una línea horizontal que atraviesa ambas gráficas, la cual indica la temperatura o humedad media respectivamente.

En la Figura 9.2.4 se muestra el interfaz gráfica de la aplicación desarrollada cuando la misma es iniciada, en donde se muestran diferentes elementos que posteriormente serán utilizados.

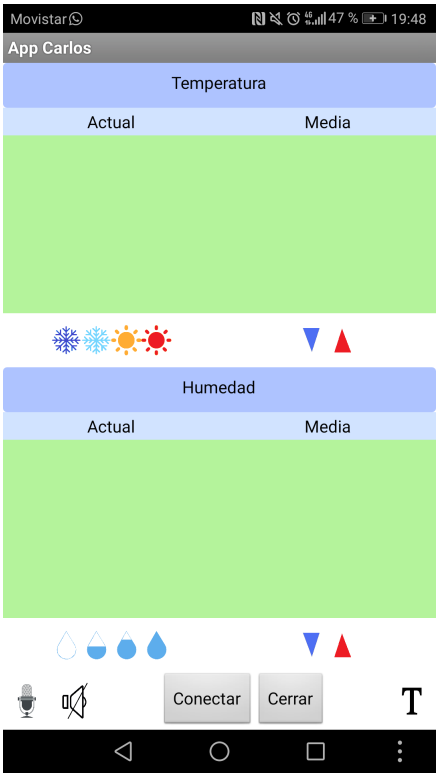


FIGURA 9.2.4. Interfaz inicial de la App de Datos

Por el contrario, en la Figura 9.2.5 se muestra un par de ejemplos del funcionamiento de la aplicación. En la pantalla izquierda se puede observar cómo se muestran los datos actuales y medios, mientras que en la pantalla derecha se observan ambas gráficas con datos temporales y valores medios.

FIGURA 9.2.5. Interfaz durante el funcionamiento de la App de Datos

Cabe destacar que ambas aplicaciones se pueden descargar de Internet para su uso de forma gratuita. Tanto la App de Selección como la App de Datos han sido desarrolladas de manera específica para este proyecto y para el sistema operativo Android, por lo que no se garantiza su correcta funcionalidad para aplicaciones externas.

Resumen

Se han desarrollado dos aplicaciones móviles, a través de **MIT App Inventor**, en las que se pueda observar el estado de la estación meteorológica. La primera de ellas, **App de Selección**, permite una comunicación **bidireccional** (del móvil a la FPGA y viceversa), donde es posible seleccionar la variable a recibir. La segunda, **App de Datos**, es algo más sofisticada, permitiendo una comunicación **unidireccional** (de la FPGA al dispositivo móvil), muestra ambas variables y datos de interés de la estación.

Evaluación del proyecto

Capítulo 10

Estudio de los recursos empleados en la FPGA

A lo largo de este apartado del proyecto se va a proceder a realizar un estudio de los recursos empleados por la FPGA cuando las descripciones estructurales y funcionales han sido desarrolladas y debidamente cargadas sobre el dispositivo.

Las descripciones (circuitería desarrollada) sometidas a estudio van a ser los circuitos principales, aunque en algunos casos se realizará una observación sobre determinadas partes que por diferentes motivos son de interés desde el punto de vista de recursos empleados.

Las FPGA disponen de gran cantidad de componentes, sin embargo, a la hora del diseño hay una serie de elementos que van a ser críticos desde el punto de vista de utilización de recursos. Vamos a basarnos en los siguientes recursos para el estudio realizado: número de registros, número de Slices, número de Luts4 y disponibilidad de PLL..

En primer lugar cabe destacar que la FPGA utilizada en la mayoría de los circuitos (salvo en el receptor del sistema completo) ha sido la MachXO2-1200ZE, es decir, vamos a tener a nuestra completa disposición 1604 registros, 640 Slices, 1280 Luts4 y un PLL. El cambio de dispositivo en el receptor del sistema completo se debe al resultado del estudio realizado sobre los recursos empleados, el cual se muestra en la Tabla 10.0.1.

	MachXO2 1200-ZE		Registros		Slices		Luts4		PLL	
	Disponibles (u.)		1604		640		1280		1	
			u.	(%)	u.	(%)	u.	(%)	u.	(%)
Interruptor más prioritario	Sistema por eventos	Emisor	1	0,1	3	0,5	6	0,5	1	100,0
		Receptor	43	2,7	45	7,0	89	7,0	1	100,0
	Sistema por tiempo	Emisor	43	2,7	45	7,0	89	7,0	1	100,0
	Circuito alternativo	Receptor	16	1,0	18	2,8	35	2,7	1	100,0
Temperatura y humedad	Completo		92	5,7	348	54,4	693	54,1	1	100,0
	Toma y emisión		154	9,6	98	15,3	170	13,3	1	100,0
	Recepción y tratamiento		67	4,2	324	50,6	648	50,6	1	100,0
Comandos AT	Emisión		34	2,1	301	47,0	599	46,8	1	100,0
	Recepción y tratamiento	Registro (op. 1)	182	11,3	501	78,3	1001	78,2	1	100,0
		Registro (op. 2)	23	1,4	181	28,3	156	12,2	1	100,0
		Dos FSMs	10	0,6	71	11,1	138	10,8	1	100,0
	Completo		239	14,9	367	57,3	725	56,6	1	100,0
Descripción funcional final	Emisor		391	24,4	455	71,1	899	70,2	1	100,0
	Receptor		347	21,6	711	111,1	1414	110,5	1	100,0
Aplicaciones	App de Selección		160	10,0	398	62,2	792	61,9	1	100,0
	App de Datos		150	9,4	396	61,9	787	61,5	1	100,0

Tabla 10.0.1. Recursos empleados

En dicha tabla podemos observar tanto los recursos disponibles, como los empleados en cada una de las etapas desarrolladas a lo largo del proyecto. Se aprecia en color verde los valores de los recursos que no superan el 100 % de los disponibles mientras que en color rojo cuando se sobrepasa el uso de recursos disponibles en la MachXO2-1200ZE..

Es esta la razón por la cual en el circuito donde se rebasa el uso de recursos será necesario emplear la MachXO2-7000HE, cuyo número de recursos es altamente superior.

Cabe destacar que a lo largo del presente proyecto se han diseñado la circuitería de tal manera que se utilice únicamente un PLL. Recordemos que este recurso se empleaba para obtener frecuencias muy exactas a partir de una frecuencia base, por lo que resulta muy interesante para el desarrollo de frecuencias utilizadas para sincronizar todo el sistema, como son los relojes.

Capítulo 11

Estudio económico

Después de realizar el estudio técnico, es decir, el estudio de los recursos empleados por la FPGA, se determinará el conjunto de recursos económicos necesarios para la realización de este proyecto.

En primer lugar se determinarán los costes de inversión en equipo, es decir, los costes de adquisición. Estos costes están asociados a la inversión del **prototipo inicial**. Durante el desarrollo del proyecto se han utilizado una serie de materiales, que con su correspondiente coste se muestran en la Tabla 11.0.1.

Componente	Coste unitario (€)	Cantidad	Coste total (€)
MachXO2 1200-ZE	30	1	30
MachXO2 7000-HE	30	1	30
HC-12	1,65	2	3,3
HC-05	1,42	1	1,42
DHT22	2,05	1	2,05
Analizador lógico y cableado	8,25	1	8,25
			75,02

Tabla 11.0.1. Coste de materiales del prototipo

Al coste total de 94.42€ se añaden los costes de trasporte (sobre un 5 % del coste total) y los de fabricación de las PCB anidadas a las placas MachXO2 (cerca de 3€ por cada una de ellas), las cuales fueron desarrolladas con anterioridad. Tanto la placa 1200-ZE como la 7000-HE tienen un precio de 30€, esto se debe a que la primera de ellas es una versión anterior que no se comercializa actualmente.

Los costes de depreciación y amortización de materiales suponen alrededor de un 2 % del coste total. Además no ha habido costes relacionados los programas de ordenador instalados, puesto que son de software libre. El coste total es el que se muestra en la tabla 11.0.2.

Costes	Valor
Materiales	75,02
Transporte	3,75
PCB (x4)	12,00
Depreciación	1,50
	92,27

Tabla 11.0.2. Coste total del prototipo

Resultando un coste total de **92.27€**. Por último, se debe determinar el coste de mano de obra por parte del ingeniero responsable del proyecto. Se estima que el desarrollo del mismo se ha establecido en cerca de 425 horas de trabajo a lo largo del presente curso académico.

De cara a la comercialización del producto, donde se estima una producción mínima de 1000 ejemplares, se obtienen los costes máximos relacionados con los materiales que se muestran en la tabla 11.0.3.

Componente	Coste unitario (€)	Cantidad	Coste total (€)
LCMXO2-1200ZE-4TG144C	3,45	1	3,45
LCMXO2-7000HE-4TG144C	3,45	1	3,45
HC-12	0,59	2	1,18
HC-05	0,49	1	0,49
DHT22	0,85	1	0,85
			9,42

Tabla 11.0.3. Coste por producto

Consiguiendo un producto competitivo y versátil de **9.42€** por elemento.

Capítulo 12

Conclusiones y Líneas futuras de desarrollo

En este apartado se exponen las conclusiones que se han sacado como consecuencia de la realización del proyecto.

La conclusión fundamental es que se ha alcanzado el objetivo principal planteado, consistente en la implantación de comunicaciones inalámbrica, entre los elementos que componen una estación meteorológica.

Para alcanzar los objetivos, se han planteado técnicas modernas de diseño, estructurado y jerárquico, describiendo los bloques fundamentales en lenguaje VHDL, e interconectando los diferentes bloques mediante esquemas.

Para facilitar la puesta a punto, el proyecto se ha realizado de forma **incremental**, partiendo de diseños sencillos y aumentando paulatinamente la complejidad, hasta llegar al sistema completo, de una gran complejidad.

Para conseguir el objetivo principal, ha sido necesario alcanzar una serie de objetivos parciales:

- Estudio profundo de herramientas de diseño electrónico modernas:
  - Software de diseño Lattice Diamond.
  - Simulador digital Active-HDL.
  - Herramienta de síntesis lógica Synopsys.
  - Herramienta de generación de bloques lógicos parametrizables IPexpress.
  - Editor de diagramas de estados para máquinas de estados finitos, Qfsm, la cual incluye una herramienta de generación automática de código VHDL.
  - Herramienta de análisis de señales "Logic" para la utilización del analizador lógico.
- Estudio detallado, y selección, de dispositivos:
  - Dispositivos FPGA, soportes del proyecto.
  - Sensores de temperatura y humedad.
  - Módulos de comunicación inalámbrica, vía Bluetooth.

    - Analizador lógico y componentes auxiliares.
- Estudio y análisis de protocolos de comunicación, tanto estándar, como específicos de determinados componentes, como son:
    - Comunicación I2C y SPI.
    - Protocolo específico de los sensores.
    - Comunicación inalámbrica, bluetooth.
- Estudio de la comunicación entre diferentes estaciones para su implantación real.
- Estudio de los comandos AT, para incorporar una gran flexibilidad en las comunicaciones entre los diferentes elementos del sistema.
- Estudio de las técnicas de desarrollo de aplicaciones Android para el diseño de diferentes aplicaciones que permitan visualizar el estado de la estación meteorológica.
- Se ha realizado un análisis de los recursos empleados en los dispositivos FPGA que soportan el sistema y una estimación de los costes de producción, si se decidiera su comercialización.

Es importante resaltar la generalidad del diseño realizado, que al haberse hecho de forma modular, permitiría incorporar determinados bloques a otros sistemas diferentes. Es decir, se han realizado **módulos reutilizable**s.

Líneas futuras de desarrollo.

Una vez cumplidos los objetivos de comunicación, como líneas futuras de desarrollo se presentan:

- Búsqueda de aplicaciones útiles para la comunicación entre diferentes dispositivos FPGA y entre estos dispositivos y el móvil.
- Diseño y desarrollo de una PCB de pequeño tamaño que incluya únicamente la FPGA, el sensor, el módulo y algún elemento auxiliar.
- Desarrollo de aplicaciones móviles de carácter comercial.
- Implantación de estaciones meteorológicas a nivel industrial en un amplio sentido.
- Utilización de la estación meteorológica en domótica, para la visualización de la temperatura y humedad del hogar. Un control por parte del usuario de las diferente variables desde el dispositivo móvil, sería una buena complementación de la estación desarrollada.
- Selección de componentes, a mayores, que aporten valores de presión, para ampliar la capacidad de la estación meteorológica.

Hechos.

A lo largo del proyecto se han concluido una serie de hechos relacionados con temas propios de presente documento.

En el caso del problema propuesto sobre el interruptor más prioritario activado (Capítulo 5 sobre la comunicación básica), podremos utilizar cualquier combinación de circuitos emisores con circuitos receptores. Este hecho se debe a que utilizan el mismo protocolo, el cual ha sido definido gracias al formato del mensaje que utilizará el módulo de Bluetooth a su entrada. Gracias a esta comprobación podemos concluir un aspecto muy importante:

Hecho 1. *Para garantizar la compatibilidad absoluta entre dos sistemas, las mismas funciones han de ser realizadas de la misma forma en todos los equipos. Las reglas que determinan estas funciones vienen dadas por el protocolo del nivel correspondiente, por lo que si los diferentes sistemas trabajan con el mismo protocolo se garantiza que son compatibles. Para ello se debe establecer la información intercambiada, el formato del mensaje y las pautas para establecer el diálogo.*

Puesto que en nuestro sistema utilizamos el mismo protocolo, aseguramos que se va a establecer una correcta comunicación entre emisor y receptor.

Concluimos que, después de comprobar que todo funciona correctamente, el módulo de Bluetooth aportará un servicio a nuestro dispositivo FPGA, concluyendo el siguiente hecho:

Hecho 2. *El objetivo final de cualquier sistema de comunicación es proporcionar una serie de servicios de comunicación a sus usuarios. Para hacer transparentes a estos la complejidad de las funciones necesarias, cada nivel engloba unas funciones concretas, cuya realización proporciona un conjunto preciso de servicios al nivel superior.*

En nuestro sistema el módulo HC-12 o HC-05 están aportando un servicio muy valioso para establecer la comunicación desde el punto de vista de la FPGA. No sería posible establecer una comunicación inalámbrica entre dos dispositivos FPGA si no utilizamos un nivel inferior que nos aporte ese servicio. En este caso, ese nivel inferior será el módulo de Bluetooth que nos facilitará dicha acción.

Bibliografía

[1] Peter J. Ashenden (1990). The VHDL Cookbook, First Edition. Universidad de Adelaide (South Australia). Peter J. Ashenden.

[2] K.C. Chang (1997). Digital Design and Modeling with VHDL and Synthesis. Los Vaqueros Circle: IEEE Computer Society Press.

[3] S.A. Pérez, E. Soto, S. Fernández (2002). Diseño de sistemas digitales con VHDL. Fuenlabrada (Madrid). Thomson.

[4] MachXO2 Breakout Board Evaluation Kit User's Guide (Anexo D.1).

[5] Manual de utilización del programa Qfsm (http://qfsm.sourceforge.net/qfsm_doc/user/qfsm.html).

[6] Manual de utilización de MIT App Inventor (http://codeweek.eu/resources/spain/guia -iniciacion-app-inventor.pdf)

[7] Guía de usuario de LyX (http://portal.uned.es/pls/portal/docs/PAGE/UNED_MAIN/LAUNIVERSIDAD/UBICACIONES/01/OFERTAESTUDIOS/GRADOS_CIENCIAS/SOFT)

[8] MIT App Inventor: http://ai2.appinventor.mit.edu/.

[9] Visitado por última vez 15/02/2018: https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_VHDL/Elementos_b%C3%A1sicos_del_lenguaje

[10] Visitado por última vez 02/03/2018: http://mixteco.utm.mx/~merg/AC/vhdl/5_Elementos_de_la_Arquitectura.pdf

[11] Visitada por última vez 22/03/2018: https://www.prometec.net/bt-hc05/

[12] Visitada por última vez 28/03/2018: https://www.ibm.com/support/knowledgecenter/es/ssw_aix_61/com.ibm.aix.networkcomm/conversion_table.htm

Anexos

ANEXO A

Códigos VHDL desarrollados

A.1. Comunicación básica

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity puls is
port(
    ent:    in std_logic_vector(7 downto 0);
    sal:    out std_logic_vector(2 downto 0)
    );
end puls;

architecture arch_puls of puls is
begin

    sal <= "000" when ent="00000001" else
           "001" when (ent(7)='0' and ent(6)='0' and ent(5)='0' and ent(4)='0' and ent(3)='0' and ent(2)='0' and ent(1)='1') else
           "010" when (ent(7)='0' and ent(6)='0' and ent(5)='0' and ent(4)='0' and ent(3)='0' and ent(2)='1') else
           "011" when (ent(7)='0' and ent(6)='0' and ent(5)='0' and ent(4)='0' and ent(3)='1') else
           "100" when (ent(7)='0' and ent(6)='0' and ent(5)='0' and ent(4)='1') else
           "101" when (ent(7)='0' and ent(6)='0' and ent(5)='1') else
           "110" when (ent(7)='0' and ent(6)='1') else
           "111" when ent(7)='1';

end arch_puls;
```

FIGURA A.1.1. Código del Decodificador de Interruptores

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity BCD_7seg is
port (
    error:  in std_logic;
    ent: in std_logic_vector (3 downto 0);
    sal: out std_logic_vector (6 downto 0)
    );
end BCD_7seg;

architecture arch_BCD of BCD_7seg is
begin

    process(error,ent)
    begin
        if error='0' then
            case ent is
                when "0001" => sal <= "0110000";
                when "0010" => sal <= "1101101";
                when "0011" => sal <= "1111001";
                when "0100" => sal <= "0110011";
                when "0101" => sal <= "1011011";
                when "0110" => sal <= "1011111";
                when "0111" => sal <= "1110000";
                when "1000" => sal <= "1111111";
                when "1001" => sal <= "1111011";
                when "0000" => sal <= "1111110";
                when others => sal <= "1001111";
            end case;
        end if;
    end process;

end arch_BCD;
```

FIGURA A.1.2. Código del Convertidor BCD-7segmentos

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity multi is
port(
    ent:    in std_logic_vector(2 downto 0);
    clk, enable:    in std_logic;
    sal:    out std_logic
    );
end multi;

architecture arch_multi of multi is
signal anterior:    std_logic_vector(2 downto 0) := "000"; -- Detecta el cambio
constant num_ent:   integer range 0 to 32 := 4; -- Número de bits de datos
begin

    process(clk)
    variable aux:   bit; -- Variable introductora al campo de datos
    variable cuenta:    integer range 0 to 8 := 8; -- Bits de datos restantes
    variable inicio:    bit; -- Variables que indican arranque y paridad
    variable paridad:   std_logic := '0'; -- Bit de paridad
    begin

        if (clk'event and clk='1' and enable='1') then

            if (anterior /= ent) then -- Detección del cambio
                inicio := '1';
            end if;

            if inicio='1' then -- Se ha producido un cambio en la entrada

                if aux='0' then -- Inicio del mensaje (Bit de arranque)
                    sal <= '0';
                    aux := '1';
                    cuenta := 8; -- Se reinician las variables

                elsif (aux='1') then -- Inicio de datos

                    if cuenta>num_ent then
                        sal <= '0';
                        cuenta := cuenta - 1;
                    elsif cuenta=num_ent then
                        paridad := '0';
                        for I in 0 to 2 loop
                            paridad := paridad xor ent(I); -- Calculamos paridad
                        end loop;
                        cuenta := cuenta - 1;
                        sal <= paridad; -- Enviamos paridad
                    else
                        sal <= ent(cuenta - 1); -- Datos enviados secuencialmente
                        cuenta := cuenta - 1;
                        if cuenta=0 then
                            aux := '0'; -- Fin de datos
                            inicio := '0'; -- Indicamos fin de mensaje
                            anterior <= ent; -- Actualizamos señal anterior
                        end if;
                    end if;

                end if;

            else
                sal <= '1'; -- Normalmente linea a nivel alto

            end if;

        end if;

    end process;

end arch_multi;
```

FIGURA A.1.3. Código del Multiplexor especial

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity demux is
port(
    ent, enable, clk:   in std_logic;
    error:  out std_logic;
    sal:    out std_logic_vector(2 downto 0)
    );
end demux;

architecture arch_demux of demux is
constant num_datos: integer range 0 to 32 := 4; -- Número de datos
begin

    process(clk)
    variable aux: bit; -- Bit auxiliar para inicializar el campo de datos
    variable paridad, par_lleg: std_logic := '0';
    variable cuenta: integer range 0 to 8 := 8; -- Número de bits del campo de datos
    begin
        if(enable='1' and clk'event and clk='1') then

            if (aux='0' and ent='0') then -- No hay mensaje transmitiendose y llega un '0' (arranque)
                aux := '1'; -- Indicamos inicio de datos
                paridad := '0';
                cuenta := 8;
                error <= '0'; -- Inicializamos variables

            elsif aux='1' then -- Campo de datos

                if cuenta>num_datos then -- Desechamos bits sobrantes
                    cuenta := cuenta - 1;
                elsif cuenta=num_datos then -- Almacenamos bit de paridad
                    par_lleg := ent;
                    cuenta := cuenta - 1;
                elsif cuenta>0 then
                    sal(cuenta-1) <= ent; -- Asignamos cada dato a una linea
                    paridad := paridad xor ent; -- Calculamos paridad
                    cuenta := cuenta - 1;
                else
                    if paridad/=par_lleg then -- Comprobamos posibles errores
                        error <= '1';
                    end if;
                    aux := '0'; -- Fin del campo de datos y del mensaje

                end if;

            end if;

        end if;

    end process;

end arch_demux;
```

FIGURA A.1.4. Código del Demultiplexor especial

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity generador is
5  port(
6       ent:    in std_logic;
7       clk:    in std_logic;
8       sal:    out std_logic);
9  end generador;
10
11 architecture arch_gene of generador is
12 signal anterior:    std_logic;
13 signal aux: bit;
14 signal cuenta:  integer range 0 to 2 := 0;
15 begin
16
17     process(clk)
18     begin
19
20         if (clk'event and clk='1') then
21
22             if (ent/=anterior and aux='0') then
23                 if ent='1' then
24                     aux <= '1';
25                 end if;
26                 anterior <= ent;
27
28             elsif aux='1' then
29                 if cuenta<2 then
30                     cuenta <= cuenta + 1;
31                     sal <= '1';
32                 else
33                     aux <= '0';
34                     cuenta <= 0;
35                     sal <= '0';
36                 end if;
37
38             end if;
39
40         end if;
41
42     end process;
43
44 end arch_gene;
```

FIGURA A.1.5. Código del Generador de pulsos

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity memo8 is
port(
    ent:    in std_logic_vector(2 downto 0);
    hab, rst, clk:  in std_logic;
    sal:    out std_logic
    );
end memo8;

architecture arch_memo8 of memo8 is
signal regis:   std_logic_vector(7 downto 0);
begin

    process(clk)
    variable paridad:   std_logic;
    variable cuenta:    integer range 0 to 8;
    begin

        if (clk'event and clk='1') then

            if rst='1' then -- Actualizo la memoria

                regis <= "00000000";
                paridad := '0';
                cuenta := 8;
                for I in 0 to 2 loop
                    regis(I) <= ent(I); -- Escribo datos
                    paridad := paridad xor ent(I); -- Calculo paridad
                end loop;
                regis(3) <= paridad; -- Escribo paridad
                sal <= '1'; -- Salida normalmente a '1'

            elsif hab='0' then
                sal <= '0'; -- Bit de arranque

            else -- Recorro la memoria
                cuenta := cuenta - 1; -- El puntero de lectura se desplaza
                sal <= regis(cuenta); -- Leo la memoria y plasmo en la salida

            end if;

        end if;

    end process;

end arch_memo8;
```

FIGURA A.1.6. Código de la Memoria del Emisor

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity reg_rec is
port(
    ent, clk, hab:  in std_logic;
    error:  out std_logic;
    sal:    out std_logic_vector(3 downto 0)
    );
end reg_rec;

architecture arch_reg of reg_rec is
signal regi:    std_logic_vector(7 downto 0); -- Contenido del registro
signal paridad: std_logic; -- Paridad calculada para la comprobación de errores
begin

    process(clk)
    begin

        if (hab='1' and clk'event and clk='1') then -- Flanco de subida y habilitado el registro

            for I in 1 to 7 loop
                regi(I) <= regi(I-1); -- Desplazo el registro desechando el último valor
            end loop;

            regi(0) <= ent; -- Cargo en el inicio del registro la entrada

        end if;

    end process;

    sal <= '0' & regi(2) & regi(1) & regi(0); -- Determino la salida
    paridad <= regi(0) xor regi(1) xor regi(2); -- Calculo paridad
    error <= '1' when paridad/=regi(3) else -- Compruebo errores
             '0';

end arch_reg;
```

FIGURA A.1.7. Código del Registro de desplazamiento del Receptor

A.2. Estación meteorológica

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity gen_puls is
5  port(
6      ent:    in std_logic;
7      clk:    in std_logic;
8      sal:    out std_logic);
9  end gen_puls;
10
11 architecture arch_gen_puls of gen_puls is
12 signal anterior:    std_logic;
13 signal aux: bit;
14 signal cuenta:  integer range 0 to 25 := 0;
15 signal veces:   integer range 0 to 6 := 0;
16 begin
17
18     process(clk)
19     begin
20
21         if (clk'event and clk='1') then
22
23             if (ent/=anterior and aux='0') then -- Cambio en la entrada
24                 if ent='1' then                 -- con valor '1'
25                     aux <= '1';
26                     veces <= 0;
27                 end if;
28                 anterior <= ent; -- Se actualiza el valor anterior
29
30             elsif aux='1' then
31
32                 if veces>=5 then -- Se han generado 5 pulsos
33                     aux <= '0';
34                     cuenta <= 0;
35                 elsif cuenta<2 then -- Pulso a nivel alto
36                     sal <= '1';
37                     cuenta <= cuenta + 1;
38                 elsif cuenta<12 then -- Separación entre pulsos
39                     sal <= '0';
40                     cuenta <= cuenta + 1;
41                 else -- Se ha generado un pulso
42                     cuenta <= 0;
43                     veces <= veces + 1;
44                 end if;
45
46             end if;
47
48         end if;
49
50     end process;
51
52 end arch_gen_puls;
```

FIGURA A.2.1. Código del Generador de 5 pulsos

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity alarga is
port(
    ent:    in std_logic;
    clk:    in std_logic;
    sal:    out std_logic);
end alarga;

architecture arch_alarga of alarga is
signal anterior:    std_logic;
signal aux: bit;
signal cuenta:  integer range 0 to 12 := 0;
begin

    process(clk)
    begin

        if (clk'event and clk='1') then

            if (ent/=anterior and ent='1') then -- Activación de la entrada
                aux <= '1';
                sal <= '1';
            end if;

            if aux='1' then -- Se ha recibido un pulso
                cuenta <= cuenta + 1;
                if cuenta>10 then -- Alargo el pulso hasta 10 transiciones
                    aux <= '0';
                    cuenta <= 0;
                    sal <= '0';
                end if;
            end if;

            anterior <= ent; -- Se actualiza el valor anterior

        end if;

    end process;

end arch_alarga;
```

FIGURA A.2.2. Código del Alargador de pulsos

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity memo40 is
port(
    ent:    in std_logic_vector(39 downto 0);
    rst, hab, clk:  in std_logic;
    sal:    out std_logic
    );
end memo40;

architecture arch_memo40 of memo40 is
signal regis:   std_logic_vector(39 downto 0);
begin

    process(clk)
    variable cuenta:    integer range 0 to 40 := 40;
    begin

        if (clk'event and clk='1') then

            if rst='1' then -- Actualizo la memoria

                if cuenta=0 then
                    cuenta := 40;
                end if;

                for I in 0 to 39 loop
                    regis(I) <= ent(I); -- Escribo datos en memoria
                end loop;

                sal <= '1'; -- Salida normalmente a '1'

            elsif hab='0' then
                sal <= '0'; -- Bit de arranque

            else -- Recorro la memoria
                cuenta := cuenta - 1; -- El puntero de lectura se desplaza
                sal <= regis(cuenta); -- Leo la memoria y plasmo en la salida

            end if;

        end if;

    end process;

end arch_memo40;
```

FIGURA A.2.3. Código de la Memoria de 40 bits

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity reg40 is
5  port(
6      ent, clk, hab:  in std_logic;
7      sal:    out std_logic_vector(39 downto 0)
8      );
9  end reg40;
10
11 architecture arch_reg40 of reg40 is
12 signal regi:    std_logic_vector(39 downto 0); -- Contenido del registro (auxiliar)
13 begin
14
15     sal <= regi;
16
17     process(clk)
18     begin
19
20         if (hab='1' and clk'event and clk='1') then -- Flanco de subida y habilitado el registro
21
22             for I in 1 to 39 loop
23                 regi(I) <= regi(I-1); -- Desplazo el registro desechando el último valor
24             end loop;
25
26             regi(0) <= ent; -- Cargo en el inicio del registro la entrada
27
28         end if;
29
30     end process;
31
32 end arch_reg40;
```

FIGURA A.2.4. Código del Registro de desplazamiento de 40 bits

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity det_cam is
port(
    ent, clk:   in std_logic;
    sal:    out std_logic
    );
end det_cam;

architecture arch_det_cam of det_cam is
signal primero: bit;
signal cuenta:  integer range 0 to 12 := 0;
begin

    process(clk)
    begin

        if (clk'event and clk='1') then

            if (primero='0' and ent='0') then -- Estamos en bit de parada (1ª vez)
                cuenta <= cuenta + 1;
                if cuenta > 10 then -- 10 bits de parada seguidos
                    sal <= '1'; -- Pulso de fin de datos
                    cuenta <= 0;
                    primero <= '1'; -- Sin pulso hasta que se vuelvan a cargar datos
                                    -- en el registro
                else
                    sal <= '0';
                end if;

            elsif ent='1' then -- Cargando datos en el registro
                primero <= '0'; -- Cuando se termine de cargar datos -> Posible pulso
                cuenta <= 0;
                sal <= '0';
            else
                sal <= '0';
            end if;

        end if;

    end process;

end arch_det_cam;
```

FIGURA A.2.5. Código del Detector de fin de transmisión

A.3. Comandos AT

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity gen_act is
port(
    rst, ent, clk:  in std_logic; -- Entradas
    set, sal:   out std_logic); -- Salidas
end gen_act;

architecture arch_gen_act of gen_act is
signal anterior:    std_logic; -- Deteccion de cambio del pulsador
signal aux: bit; -- Separación de zonas
signal cuenta:  integer range 0 to 4001 := 0;
constant previa:    integer range 0 to 800 := 400;
constant posterior: integer range 0 to 2400 := 1200;
begin

    process(clk)
    begin

        if (rst ='1' and clk'event and clk='1') then -- Sin reset/Flanco de subida

            if (ent/=anterior and aux='0') then -- Detección del pulsador
                if ent='1' then -- Pulso a nivel alto
                    aux <= '1'; -- Se activa generación del pulso
                    set <= '0'; -- Permito comando AT
                end if;
                anterior <= ent; -- Actualizo valor anterior

            elsif aux='1' then

                if cuenta < (previa-2) then -- Espera para generar el pulso (40ms)
                    cuenta <= cuenta + 1;
                    sal <= '0'; -- Sin pulso
                elsif cuenta < previa then -- Generación del pulso
                    cuenta <= cuenta + 1;
                    sal <= '1'; -- Pulso de dos transiciones de reloj
                elsif cuenta < posterior then  -- Periodo de SET a '0' (40+80)ms
                    cuenta <= cuenta + 1;
                    sal <= '0'; -- Fin de pulso
                else -- Inicialización de variables
                    aux <= '0';
                    set <= '1';
                    cuenta <= 0;

                end if;

            else
                set <= '1'; -- Normalmente SET a '1'

            end if;

        end if;

    end process;

end arch_gen_act;
```

FIGURA A.3.1. Código del Detector de Pulsación

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity borrar is
port(
    rst, hab, clk:  in std_logic;
    sal:    out std_logic
    );
end borrar;

architecture arch_borrar of borrar is
signal regis:   std_logic_vector(15 downto 0); -- AT
signal cuenta:  integer range 0 to 15 := 15;
begin

    regis <= "1000001000101010"; -- AT

    process(clk)
    begin

        if (clk'event and clk='1') then

            if rst='1' then -- Inicializo variables

                if cuenta=0 then -- Inicialización de variable 'cuenta'
                    cuenta<=15;
                end if;

                sal <= '1'; -- Salida normalmente a '1'

            elsif hab='0' then
                sal <= '0'; -- Bit de arranque

            else -- Recorro la memoria
                sal <= regis(cuenta); -- Leo la memoria y plasmo en la salida
                cuenta <= cuenta - 1; -- El puntero de lectura se desplaza

            end if;

        end if;

    end process;

end arch_borrar;
```

FIGURA A.3.2. Código de la Memoria del comando de test

```
library IEEE;
use IEEE.std_logic_1164.all;

entity gen_puls2 is
port(
    ent:    in std_logic_vector(7 downto 0); -- Pulsadores
    clk:    in std_logic;
    sal:    out std_logic; -- Pulsos
    memo:   out std_logic_vector(2 downto 0)); -- Indicador mensaje
end gen_puls2;

architecture arch_gen_puls2 of gen_puls2 is
signal anterior:    std_logic_vector(7 downto 0); -- Detección de cambio
signal aux: bit; -- Divisor de etapas (pulsos y espera)
signal cuenta:  integer range 0 to 25 := 0; -- Contador de transiciones de reloj
signal veces:   integer range 0 to 10 := 0; -- Contador de pulsos generados
signal repet:   integer range 0 to 10; -- Pulsos que debemos dar
begin

    process(clk)
    begin

        if (clk'event and clk='1') then

            -- ETAPA ESPERA
            if (ent/=anterior and aux='0') then -- Cambio en la entrada con flanco de subida
                if (ent(0)='1' or ent(1)='1' or ent(2)='1' or ent(3)='1' or ent(4)='1' or ent(5)='1' or ent(6)='1' or ent(7)='1')then
                    aux <= '1'; -- Paso a la etapa de pulsos
                    veces <= 0; -- Inicializo

                    if ent(0)='1' then -- Asignación de pulsos e indicador de mensajes
                        memo <= "000";
                        repet <= 2; -- AT
                    elsif ent(1)='1' then
                        memo <= "001";
                        repet <= 9; -- AT+Bxxxxx
                    elsif ent(2)='1' then
                        memo <= "010";
                        repet <= 7; -- AT+Cxxx
                    elsif ent(3)='1' then
                        memo <= "011";
                        repet <= 6; -- AT+FUx
                    elsif ent(4)='1' then
                        memo <= "100";
                        repet <= 5; -- AT+Px
                    elsif ent(5)='1' then
                        memo <= "101";
                        repet <= 8; -- AT+SLEEP
                    elsif ent(6)='1' then
                        memo <= "110";
                        repet <= 10; -- AT+DEFAULT
                    elsif ent(7)='1' then
                        memo <= "111";
                        repet <= 9; -- AT+UPDATE
                    end if;

                end if;
                anterior <= ent; -- Se actualiza el valor anterior

            -- ETAPA PULSOS
            elsif aux='1' then

                if veces>=repet then -- Se han generado 'repet' pulsos (fin de generación)
                    aux <= '0';
                    cuenta <= 0;
                elsif cuenta<2 then -- Pulso a nivel alto
                    sal <= '1';
                    cuenta <= cuenta + 1;
                elsif cuenta<12 then -- Separación entre pulsos
                    sal <= '0';
                    cuenta <= cuenta + 1;
                else -- Se ha generado un pulso
                    cuenta <= 0;
                    veces <= veces + 1;
                end if;

            end if;
```

FIGURA A.3.3. Código modificado del Generador de Pulsos con detección del pulsador accionado

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity memo_test is
port(
    rst, hab, clk:  in std_logic; -- Entradas
    memo:   in std_logic_vector(2 downto 0); -- Selector de comando
    sal:    out std_logic -- Salida al puerto serie
    );
end memo_test;

architecture arch_memo_test of memo_test is
signal regis1:  std_logic_vector(15 downto 0); -- AT
signal regis2:  std_logic_vector(71 downto 0); -- AT+Bxxxxx
signal regis3:  std_logic_vector(55 downto 0); -- AT+Cxxx
signal regis4:  std_logic_vector(47 downto 0); -- AT+FUx
signal regis5:  std_logic_vector(39 downto 0); -- AT+Px
signal regis6:  std_logic_vector(63 downto 0); -- AT+SLEEP
signal regis7:  std_logic_vector(79 downto 0); -- AT+DEFAULT
signal regis8:  std_logic_vector(71 downto 0); -- AT+UPDATE
signal cuenta:  integer range 0 to 79 := 0; -- Puntero de lectura a memoria
signal memo_ant:    std_logic_vector(2 downto 0); -- Detección de cambio
begin

    -- CONTENIDO DE LA MEMORIA
    regis1 <= "1000001000101010"; -- AT
    regis2 <= "100000100010101011010100010000101000110010011100010011000000110000001100"; -- AT+B19200
    regis3 <= "10000010001010101101010011000010000011001100110000001100"; -- AT+C030
    regis4 <= "100000100010101011010100011000101010101001001100"; --AT+FU2
    regis5 <= "10000010001010101101010000001010001011100"; -- AT+P4
    regis6 <= "1000001000101010110101001100101000110010101000101010001000001010"; -- AT+SEEEP
    regis7 <= "100000100010101011010100001000101010001001100010100000101010101000110010001010100"; -- AT+DEFAULT
    regis8 <= "10000010001010101101010010101010000010100010001010000010001010101010100010"; -- AT+UPDATE
```

FIGURA A.3.4. a) Código modificado de la Memoria de Comandos

(continúa)

```
36      process(clk)
37      begin
38
39          if (clk'event and clk='1') then
40
41              if rst='1' then -- Inicializo el puntero a memoria
42
43                  if (cuenta=0 or memo/=memo_ant) then -- Inicialización de variable 'cuenta'
44                      case memo is -- 'cuenta' cambia si llega a 0 o si se cambia el modo (memo)
45                          when "000" => cuenta<=15;
46                          when "001" => cuenta<=71;
47                          when "010" => cuenta<=55;
48                          when "011" => cuenta<=47;
49                          when "100" => cuenta<=39;
50                          when "101" => cuenta<=63;
51                          when "110" => cuenta<=79;
52                          when others => cuenta<=71;
53                      end case;
54                  end if;
55
56                  memo_ant <= memo; -- Actualizo valor
57                  sal <= '1'; -- Salida normalmente a '1'
58
59              elsif hab='0' then
60                  sal <= '0'; -- Bit de arranque
61
62              else -- Recorro la memoria
63                  case memo is -- Leo la memoria y plasmo en la salida
64                      when "000" => sal <= regis1(cuenta);
65                      when "001" => sal <= regis2(cuenta);
66                      when "010" => sal <= regis3(cuenta);
67                      when "011" => sal <= regis4(cuenta);
68                      when "100" => sal <= regis5(cuenta);
69                      when "101" => sal <= regis6(cuenta);
70                      when "110" => sal <= regis7(cuenta);
71                      when others => sal <= regis8(cuenta);
72                  end case;
73
74                  cuenta <= cuenta - 1; -- El puntero de lectura se desplaza
75
76              end if;
77
78          end if;
79
80      end process;
81
82  end arch_memo_test;
```

FIGURA A.3.5. b) Código modificado de la Memoria de Comandos

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity selector is
port(
    s_clk:  in std_logic_vector(1 downto 0);
    clk_in: in std_logic_vector(2 downto 0);
    ind:    out std_logic_vector(2 downto 0);
    clk_out:    out std_logic);
end selector;

architecture arch_selector of selector is
begin
    with s_clk select
        clk_out <= clk_in(1) when "00",
                   clk_in(2) when "01",
                   clk_in(0) when others;

    ind <= "011" when s_clk="01" else
           "101" when s_clk="00" else
           "110";

end arch_selector;
```

FIGURA A.3.6. Código del Selector de reloj

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity regAT is
port(
    clk, hab, ent:  in std_logic;
    sal:    out std_logic_vector(111 downto 0)
    );
end regAT;

architecture arch_regAT of regAT is
signal regi:    std_logic_vector(111 downto 0); -- Contenido del registro (auxiliar)
begin

    sal <= regi;

    process(clk)
    begin

        if (hab='1' and clk'event and clk='1') then -- Flanco de subida y habilitado el registro

            for I in 1 to 111 loop -- 112 es el caso de comando más largo (OK+FU2,B4800\r\n)
                regi(I) <= regi(I-1); -- Desplazo el registro desechando el último valor
            end loop;

            regi(0) <= ent; -- Cargo en el inicio del registro la entrada

        end if;

    end process;

end arch_regAT;
```

FIGURA A.3.7. Código del Registro de desplazamiento de 112 bits

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity interprete is
port(
    ent:    in std_logic_vector(111 downto 0);
    rst, clk:   in std_logic;
    s_clk:  out std_logic_vector(1 downto 0);
    sal1:   out std_logic_vector(6 downto 0);
    sal2:   out std_logic_vector(6 downto 0));
end interprete;

architecture arch_int of interprete is
constant O: std_logic_vector(7 downto 0) := "11110010";
constant K: std_logic_vector(7 downto 0) := "11010010";
constant B: std_logic_vector(7 downto 0) := "01000010";
constant F: std_logic_vector(7 downto 0) := "01100010";
constant U: std_logic_vector(7 downto 0) := "10101010";
constant P: std_logic_vector(7 downto 0) := "00001010";
constant S: std_logic_vector(7 downto 0) := "11001010";
constant C: std_logic_vector(7 downto 0) := "11000010";
constant D: std_logic_vector(7 downto 0) := "00100010";
constant E: std_logic_vector(7 downto 0) := "10100010";
constant R: std_logic_vector(7 downto 0) := "01001010";
constant L: std_logic_vector(7 downto 0) := "00110010";
constant c_mas: std_logic_vector(7 downto 0) := "11010100";
constant retorno:   std_logic_vector(7 downto 0) := "10110000";
constant avance:    std_logic_vector(7 downto 0) := "01010000";
signal aux: std_logic_vector(111 downto 0);
signal ant: std_logic_vector(15 downto 0);
begin

    process(clk,rst)
    begin
        if (rst='1' and clk'event and clk='1') then

            if ant/=ent(15 downto 0) then -- Actualizo 'ant' y 'aux' si hay cambio
                ant <= ent(15 downto 0);
                aux <= ent;
            end if;

            -- Caracteres menos significativos son "\r\n" => Fin de comando
            if (aux(15 downto 8)=retorno and aux(7 downto 0)=avance) then
                for I in 1 to 13 loop -- Busco (si hay) otro fin de comando "\n"

                    if (aux(7+8*I downto 8*I)=avance) then
                    -- El comando debe empezar en el carácter más significativo para su lectura
                        aux(111 downto 112-8*I) <= aux(8*I-1 downto 0);
                    end if;

                end loop;
            end if;
```

FIGURA A.3.8. a) Código Opción 1 del Intérprete de Comandos

(continúa)

```
54              -- CONFIRMACIÓN
55              if (aux(111 downto 104)=O and aux(103 downto 96)=K) then -- OK
56
57                  -- Muestra por el display (OK)
58                  sal1 <= "1111110"; -- O
59                  sal2 <= "0010111"; -- k
60
61                  if (aux(95 downto 88)=c_mas) then -- OK+
62
63                      -- Modificación del reloj
64                      if (aux(87 downto 80)=B) then -- OK+B
65                          s_clk <= "01"; -- Cambio reloj a 19200Hz
66                          sal1 <= "0011111"; -- b
67                          sal2 <= "0001101"; -- c (cambiado)
68                      elsif (aux(87 downto 80)=C) then -- OK+C
69                          sal1 <= "1001110"; -- C
70                          sal2 <= "0001101"; -- c (cambiado)
71                      elsif (aux(87 downto 80)=F and ent(79 downto 72)=U) then -- OK+FU
72                          s_clk <= "10"; -- Cambio reloj a 4800Hz
73                          sal1 <= "1000111"; -- F
74                          sal2 <= "1101101"; -- 2
75                      elsif (aux(87 downto 80)=P) then -- OK+P
76                          sal1 <= "1100111"; -- P
77                          sal2 <= "0110011"; -- 4
78                      elsif (aux(87 downto 80)=S) then -- OK+SLEEP
79                          sal1 <= "1011011"; -- S
80                          sal2 <= "0001110"; -- L
81                      elsif (aux(87 downto 80)=D) then -- OK+DEFAULT
82                          s_clk <= "00"; -- Cambio reloj a 9600Hz
83                          sal1 <= "0111101"; -- d
84                          sal2 <= "1101111"; -- e
85                      end if;
86                  end if;
87
88              -- ERROR
89              elsif (aux(111 downto 104)=E and aux(103 downto 96)=R) then
90
91                  -- Muestra por el display (ERROR)
92                  sal1 <= "1001111"; -- E
93                  sal2 <= "0000101"; -- r
94
95              end if;
96
97          end if;
98      end process;
99
00  end arch_int;
```

FIGURA A.3.9. b) Código Opción 1 del Intérprete de Comandos

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity inter_com is
port(
    ent:    in std_logic_vector(111 downto 0); -- Registro
    rst, clk:   in std_logic;
    s_clk:  out std_logic_vector(1 downto 0); -- Selector clk
    sal1:   out std_logic_vector(6 downto 0); -- Display 1
    sal2:   out std_logic_vector(6 downto 0)); -- Display 1
end inter_com;

architecture arch_int_com of inter_com is
constant O: std_logic_vector(7 downto 0) := "11110010";
constant K: std_logic_vector(7 downto 0) := "11010010";
constant B: std_logic_vector(7 downto 0) := "01000010";
constant F: std_logic_vector(7 downto 0) := "01100010";
constant P: std_logic_vector(7 downto 0) := "00001010";
constant S: std_logic_vector(7 downto 0) := "11001010";
constant C: std_logic_vector(7 downto 0) := "11000010";
constant D: std_logic_vector(7 downto 0) := "00100010";
constant E: std_logic_vector(7 downto 0) := "10100010";
constant R: std_logic_vector(7 downto 0) := "01001010";
constant c_mas: std_logic_vector(7 downto 0) := "11010100";
constant retorno:   std_logic_vector(7 downto 0) := "10110000";
constant avance:    std_logic_vector(7 downto 0) := "01010000";
begin

    process(clk,rst)
    variable puntero:   integer range 0 to 111; -- Puntero al inicio de la respuesta
    variable aux:   integer range 0 to 11; -- Indica si se ha detectado un comando anterior
    variable salir: bit; -- Auxiliar para leer la última respuesta
    begin
        if (rst='1' and clk'event and clk='1') then

            -- Caracteres menos significativos son "\r\n" => COMANDO RECIBIDO
            if (ent(7 downto 0)=avance) then

                aux := 0; -- Inicializo
                puntero := 0;
                salir:='0';

                for I in 3 to 13 loop -- Busco (si hay) otro fin de comando "\n"
                    if (ent(7+8*I downto 8*I)=avance and salir='0') then
                    -- Coloco el puntero al principio de la ultima respuesta recibida
                        puntero := 8*I-1;
                        salir := '1'; -- Ya no busco más
                    else
                        aux := aux + 1;
                    end if;
                end loop;

                if aux=11 then
                    puntero := 111; -- Si no se encuentra ninguna respuesta anterior
                end if;
```

FIGURA A.3.10. a) Código Opción 2 del Intérprete de Comandos

(continúa)

```
57                  -- CONFIRMACIÓN
58                  if (ent(puntero downto puntero-7)=O) then -- OK
59
60                      -- Muestra por el display (OK)
61                      sal1 <= "1111110"; -- O
62                      sal2 <= "0010111"; -- k
63
64                      if (ent(puntero-16 downto puntero-23)=c_mas) then -- OK+
65                      -- Si no se cumple se recibe 'OK\r\n' (Aseguramos posibles fallos)
66
67                          -- Modificación del reloj
68                          if (ent(puntero-24 downto puntero-31)=B) then -- OK+B
69                              s_clk <= "01"; -- Cambio reloj a 19200Hz
70                              sal1 <= "0011111"; -- b
71                              sal2 <= "0001101"; -- c (cambiado)
72                          elsif (ent(puntero-24 downto puntero-31)=C) then -- OK+C
73                              sal1 <= "1001110"; -- C
74                              sal2 <= "0001101"; -- c (cambiado)
75                          elsif (ent(puntero-24 downto puntero-31)=F) then -- OK+FU
76                              s_clk <= "10"; -- Cambio reloj a 4800Hz
77                              sal1 <= "1000111"; -- F
78                              sal2 <= "1101101"; -- 2
79                          elsif (ent(puntero-24 downto puntero-31)=P) then -- OK+P
80                              sal1 <= "1100111"; -- P
81                              sal2 <= "0110011"; -- 4
82                          elsif (ent(puntero-24 downto puntero-31)=S) then -- OK+SLEEP
83                              sal1 <= "1011011"; -- S
84                              sal2 <= "0001110"; -- L
85                          elsif (ent(puntero-24 downto puntero-31)=D) then -- OK+DEFAULT
86                              s_clk <= "00"; -- Cambio reloj a 9600Hz
87                              sal1 <= "0111101"; -- d
88                              sal2 <= "1101111"; -- e
89                          end if;
90                      end if;
91
92                  -- ERROR
93                  elsif (ent(puntero downto puntero-7)=E) then
94
95                      -- Muestra por el display (ERROR)
96                      sal1 <= "1001111"; -- E
97                      sal2 <= "0000101"; -- r
98
99                  end if;
100
101             end if;
102
103         end if;
104     end process;
105
106 end arch_int_com;
```

FIGURA A.3.11. b) Código Opción 2 del Intérprete de Comandos

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity reg_det is
port(
    ent, clk, hab:  in std_logic;
    sal:    out std_logic_vector(2 downto 0);
    resp:   out std_logic_vector(7 downto 0)
    );
end reg_det;

architecture arch_reg_det of reg_det is
signal regi:    std_logic_vector(7 downto 0); -- Contenido del registro auxiliar
signal ant: std_logic_vector(7 downto 0); -- Contenido del registro
constant O: std_logic_vector(7 downto 0) := "11110010"; -- O
constant K: std_logic_vector(7 downto 0) := "11010010"; -- K
constant mas:   std_logic_vector(7 downto 0) := "11010100"; -- +
constant E: std_logic_vector(7 downto 0) := "10100010"; -- E
constant R: std_logic_vector(7 downto 0) := "01001010";    R
begin

    resp <= regi;

    process(clk)
    begin

        if (clk'event and clk='1') then

            ant <= regi; -- Actualizo anterior

            if hab='1' then -- Actualizo el registro

                for I in 1 to 7 loop
                    regi(I) <= regi(I-1); -- Desplazo el registro desechando el último valor
                end loop;

                regi(0) <= ent; -- Cargo en el inicio del registro la entrada

            end if;

            if ant=regi then -- Actualización de la salida del registro en el bit de parada
                case regi is
                    when O => sal <= "001"; -- O
                    when K => sal <= "010"; -- K
                    when mas => sal <= "011"; -- +
                    when E => sal <= "100"; -- E
                    when R => sal <= "101"; -- R
                    when others => sal <= "000"; -- Otro carácter
                end case;
            end if;

        end if;

    end process;

end arch_reg_det;
```

FIGURA A.3.12. Código del Registro de Desplazamiento de 8 bits y Decodificación de su contenido

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity cont_inter is
5  port(
6      cont:   in std_logic_vector(1 downto 0);
7      clk, rst:   in std_logic;
8      ent:    in std_logic_vector(7 downto 0);
9      disp1:  out std_logic_vector(6 downto 0);
10     disp2:  out std_logic_vector(6 downto 0);
11     s_clk:  out std_logic_vector (1 downto 0)
12     );
13 end cont_inter;
14
15 architecture arch_cont_inter of cont_inter is
16 constant d_O:   std_logic_vector(6 downto 0) := "1111110"; -- O por el display
17 constant d_K:   std_logic_vector(6 downto 0) := "0010111"; -- k por el display
18 constant d_B:   std_logic_vector(6 downto 0) := "0011111"; -- b por el display
19 constant d_c:   std_logic_vector(6 downto 0) := "0001101"; -- c por el display
20 constant d_Cm:  std_logic_vector(6 downto 0) := "1001110"; -- C por el display
21 constant d_F:   std_logic_vector(6 downto 0) := "1000111"; -- F por el display
22 constant d_P:   std_logic_vector(6 downto 0) := "1100111"; -- P por el display
23 constant d_D:   std_logic_vector(6 downto 0) := "0111101"; -- d por el display
24 constant d_E:   std_logic_vector(6 downto 0) := "1101111"; -- e por el display
25 constant d_R:   std_logic_vector(6 downto 0) := "0000101"; -- r por el display
26 constant d_S:   std_logic_vector(6 downto 0) := "1011011"; -- S por el display
27 constant d_4:   std_logic_vector(6 downto 0) := "0110011"; -- 4 por el display
28 constant d_2:   std_logic_vector(6 downto 0) := "1101101"; -- 2 por el display
29 constant d_L:   std_logic_vector(6 downto 0) := "0001110"; -- L por el display
30 constant B: std_logic_vector(7 downto 0) := "01000010"; -- B
31 constant C: std_logic_vector(7 downto 0) := "11000010"; -- C
32 constant F: std_logic_vector(7 downto 0) := "01100010"; -- F
33 constant P: std_logic_vector(7 downto 0) := "00001010"; -- P
34 constant S: std_logic_vector(7 downto 0) := "11001010"; -- S
35 constant D: std_logic_vector(7 downto 0) := "00100010"; -- D
```

FIGURA A.3.13. a) Código el Controlador e Intérprete de Respuestas

(continúa)

```
36  begin
37
38      process(clk)
39      variable d1:    std_logic_vector(6 downto 0); -- auxiliar para el display 1
40      variable d2:    std_logic_vector(6 downto 0); -- auxiliar para el display 1
41      begin
42          if (rst='1' and clk'event and clk='1') then
43
44              if cont="10" then -- Si recibimos el contenido de un parámetro
45
46                  -- Se establecen d1 y d2 para mostrar por los displays
47                  case ent is -- Se distingue entre los diferentes parámetros
48                      when B => d1 := d_B; -- OK+B19200
49                                d2 := d_c;
50                                s_clk <= "01"; -- Cambia reloj a 19.2kHz
51                      when C => d1 := d_Cm; -- OK+C030
52                                d2 := d_c;
53                      when F => d1 := d_F; -- OK+FU2,B4800
54                                d2 := d_2;
55                                s_clk <= "10"; -- Cambia reloj a 4.8kHz
56                      when P => d1 := d_P; -- OK+P4
57                                d2 := d_4;
58                      when S => d1 := d_S; -- OK+SLEEP
59                                d2 := d_L;
60                      when D => d1 := d_D; -- OK+DEFAULT
61                                d2 := d_E;
62                                s_clk <= "00"; -- Cambia reloj a 9.6kHZ (por defecto)
63                      when others => d1 := d1; -- Sentencia auxiliar (evita fallos)
64
65                  end case;
66              end if;
67
68              case cont is -- Muestra por el display
69                  when "01" => disp1 <= d_O;
70                               disp2 <= d_K;
71                  when "10" => disp1 <= d1;
72                               disp2 <= d2;
73                  when "11" => disp1 <= d_E;
74                               disp2 <= d_R;
75                  when others => d1 := d1; -- Sentencia auxiliar (evita fallos)
76              end case;
77
78          end if;
79
80      end process;
81
82  end arch_cont_inter;
```

FIGURA A.3.14. b) Código el Controlador e Intérprete de Respuestas

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity gen_var is
port(
    rst, ent, clk:  in std_logic; -- Entradas
    s_clk:  in std_logic_vector(1 downto 0);
    set, sal:   out std_logic); -- Salidas
end gen_var;

architecture arch_gen_var of gen_var is
signal anterior:    std_logic; -- Deteccion de cambio del pulsador
signal aux: bit; -- Separación de zonas
signal cuenta:  integer range 0 to 4001 := 0;
signal previa:  integer range 0 to 800 := 400;
signal posterior:   integer range 0 to 2400 := 1200;
begin

    with s_clk select
        previa <= 400 when "00",
                  800 when "01",
                  200 when others;

    with s_clk select
        posterior <= 1200 when "00",
                     2400 when "01",
                     600 when others;

    process(clk)
    begin

        if (rst ='1' and clk'event and clk='1') then -- Sin reset/Flanco de subida

            if (ent/=anterior and aux='0') then -- Detección del pulsador
                if ent='1' then -- Pulso a nivel alto
                    aux <= '1'; -- Se activa generación del pulso
                    set <= '0'; -- Permito comando AT
                end if;
                anterior <= ent; -- Actualizo valor anterior

            elsif aux='1' then

                if cuenta < (previa-2) then -- Espera para generar el pulso (40ms)
                    cuenta <= cuenta + 1;
                    sal <= '0'; -- Sin pulso
                elsif cuenta < previa then -- Generación del pulso
                    cuenta <= cuenta + 1;
                    sal <= '1'; -- Pulso de dos transiciones de reloj
                elsif cuenta < posterior then  -- Periodo de SET a '0' (40+80)ms
                    cuenta <= cuenta + 1;
                    sal <= '0'; -- Fin de pulso
                else -- Inicialización de variables
                    aux <= '0';
                    set <= '1';
                    cuenta <= 0;

                end if;

            else
                set <= '1'; -- Normalmente SET a '1'

            end if;

        end if;

    end process;

end arch_gen_var;
```

FIGURA A.3.15. Código del Detector de Pulsación definitivo

A.4. Descripción estructural final

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity mantenedor is
port(
    clk:    in std_logic;
    disp1_a, disp2_a:   in std_logic_vector(6 downto 0); -- Procedente de comandos
    disp1_b, disp2_b:   in std_logic_vector(6 downto 0); -- Procedente de tratamiento
    disp1_s, disp2_s:   out std_logic_vector(6 downto 0) -- Salida hacia los displays
    );
end mantenedor;

architecture arch_mante of mantenedor is
signal ant1, ant2:  std_logic_vector(6 downto 0); -- Reconocer el cambio en la salida de comandos
begin

    process(clk)
    variable activo:    bit;
    variable cuenta:    integer range 0 to 28801;
    begin

        if (clk'event and clk='1') then

            if (disp1_a/=ant1 or disp2_a/=ant2) then -- Cambio en la salida de comandos
                activo := '1'; -- Inicia cuenta
                cuenta := 0;
            end if;

            if activo='1' then

                if cuenta<28800 then
                    cuenta := cuenta + 1;
                else -- Han transcurrido 3 segundos (para 9.6kHz)
                    cuenta := 0;
                    activo := '0'; -- Fin de cuenta
                end if;

                disp1_s <= disp1_a; -- Si hay cambio => se muestra los comandos
                disp2_s <= disp2_a;

            else
                disp1_s <= disp1_b; -- No hay cambio => se muestran los datos
                disp2_s <= disp2_b;

            end if;

            ant1 <= disp1_a; -- Actualizo valores
            ant2 <= disp2_a;

        end if;

    end process;

end arch_mante;
```

FIGURA A.4.1. Código de la lógica combinacional del receptor

A.5. Aplicación móvil

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_arith.all;
4  use IEEE.std_logic_unsigned.all;
5
6  entity calcula is
7  port(
8      bin :   in  std_logic_vector (15 downto 0);
9      sel:    in std_logic_vector(1 downto 0);
10     sal:    out std_logic_vector(55 downto 0)
11     );
12 end calcula;
13
14 architecture arch_calcula of calcula is
15 signal dec_u:   std_logic_vector (7 downto 0);
16 signal dec_d:   std_logic_vector (7 downto 0);
17 signal dec_c:   std_logic_vector (7 downto 0);
18 constant coma:  std_logic_vector (7 downto 0) := "00110100";
19 constant retorno:   std_logic_vector (7 downto 0) := "10110000";
20 constant avance:    std_logic_vector (7 downto 0) := "01010000";
21 constant Cm:    std_logic_vector(7 downto 0) := "11000010";
22 constant H: std_logic_vector(7 downto 0) := "00010010";
23 constant T: std_logic_vector(7 downto 0) := "00101010";
24 constant esp:   std_logic_vector(7 downto 0) := "00000100";
25 constant o: std_logic_vector(7 downto 0) := "11110110";
26
27 begin
28
29     process (bin) is
30     variable res, den_1,den_2, d_1,d_2, c: integer;
31     variable aux_c, aux_d, aux_u:   std_logic_vector (7 downto 0);
32
33     begin
34
35         c := conv_Integer(bin);
36         den_1 := 10;
37         d_1 := 0;
38         den_2 := 100;
39         d_2 := 0;
40
```

FIGURA A.5.1. a) Código del Generador de mensajes

```
41          for I in 1 to 10 loop
42
43              if (c>=den_2) then
44                  c := c - den_2;
45                  d_2 := d_2 +1;
46              end if;
47
48          end loop;
49
50          c := (conv_Integer(bin)-d_2*100);
51
52
53          for I in 1 to 10 loop
54
55              if (c>=den_1) then
56                  c := c - den_1;
57                  d_1 := d_1 +1;
58              end if;
59
60          end loop;
61
62          -- PARTE AÑADIDA
63
64          -- Partimos de decenas, unidades y decimas
65          -- Añadimos el prefijo "0011" (cógido ASCII que indica valor númerico)
66          aux_c := "0011" & conv_std_logic_vector(d_2,4);
67          aux_d := "0011" & conv_std_logic_vector((d_1),4);
68          aux_u := "0011" & conv_std_logic_vector((conv_Integer(bin)-d_1*10-d_2*100),4);
69
70          -- Cambio a little endian
71          for I in 0 to 7 loop
72              dec_c(I) <= aux_c(7-I);
73              dec d(I) <= aux d(7-I);
74              dec_u(I) <= aux_u(7-I);
75          end loop;
76
77          case sel is
78              when "10" => sal <= dec_c & dec_d & coma & dec_u & Cm & retorno & avance; -- °C
79              when "01" => sal <= dec_c & dec_d & coma & dec_u & H & retorno & avance; -- %H
80              when others => sal <= T & esp & o & esp & H & retorno & avance; -- T o H?
81          end case;
82
83          -- FIN DE PARTE AÑADIDA
84
85      end process;
86
87  end arch_calcula;
```

FIGURA A.5.2. b) Código del Generador de mensajes

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity memo_app is
port(
    ent:    in std_logic_vector(55 downto 0);
    rst, hab, clk:  in std_logic;
    sal:    out std_logic
    );
end memo_app;

architecture arch_memo_app of memo_app is
signal regis:   std_logic_vector(55 downto 0);
begin

    process(clk)
    variable cuenta:    integer range 0 to 56 := 56;
    begin

        if (clk'event and clk='1') then

            if rst='1' then -- Actualizo la memoria

                if cuenta=0 then
                    cuenta := 56;
                end if;

                for I in 0 to 55 loop
                    regis(I) <= ent(I); -- Escribo datos en memoria
                end loop;

                sal <= '1'; -- Salida normalmente a '1'

            elsif hab='0' then
                sal <= '0'; -- Bit de arranque

            else -- Recorro la memoria
                cuenta := cuenta - 1; -- El puntero de lectura se desplaza
                sal <= regis(cuenta); -- Leo la memoria y plasmo en la salida

            end if;

        end if;

    end process;

end arch_memo_app;
```

FIGURA A.5.3. Código de la memoria de la App de selección

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity sel_th is
5  port(
6      clk:    in std_logic;
7      ent:    in std_logic_vector(7 downto 0);
8      sal:    out std_logic_vector(1 downto 0)
9      );
10 end sel_th;
11
12 architecture arch_sel_th of sel_th is
13 constant T: std_logic_vector(7 downto 0) := "00101010";
14 constant H: std_logic_vector(7 downto 0) := "00010010";
15 begin
16
17     process(clk)
18     begin
19
20         if (clk'event and clk='1') then
21
22             case ent is
23                 when T => sal <= "10";
24                 when H => sal <= "01";
25                 when others => sal <= "00";
26             end case;
27
28         end if;
29
30     end process;
31
32 end arch_sel_th;
```

FIGURA A.5.4. Código del selector de temperatura o humedad

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity alter is
5  port(
6      clk:    in std_logic;
7      check_sum:  in std_logic;
8      clk_2s:     in std_logic;
9      sal:    out std_logic;
10     sel:    out std_logic_vector(1 downto 0) -- T o H
11     );
12 end alter;
13
14 architecture arch_alter of alter is
15 signal ant_c, ant_2:    std_logic; -- Señales auxiliares (anteriores)
16 signal aux: std_logic;
17 begin
18
19     process(clk)
20     begin
21
22         if (clk'event and clk='1') then
23
24             if (check_sum/=ant_c and check_sum='1') then
25                 sel <= "10"; -- Check_sum normal -> Temperatura
26             end if;
27
28             if (clk_2s/=ant_2 and clk_2s='0') then
29                 aux <= '1';
30                 sel <= "01"; -- Check_sum forzado -> Humedad
31             else
32                 aux <= '0'; -- Sin check_sum
33             end if;
34
35             ant_c <= check_sum;
36             ant_2 <= clk_2s;
37
38         end if;
39     end process;
40
41     sal <= aux or check_sum; -- Check_sum normal y forzado
42
43 end arch_alter;
```

FIGURA A.5.5. Código del Alternador

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity calcula_doble is
port(
    bin :   in  std_logic_vector (15 downto 0);
    sel:    in std_logic_vector(1 downto 0);
    sal:    out std_logic_vector(47 downto 0)
    );
end calcula_doble;

architecture arch_calcula_doble of calcula_doble is
signal dec_u:   std_logic_vector (7 downto 0);
signal dec_d:   std_logic_vector (7 downto 0);
signal dec_c:   std_logic_vector (7 downto 0);
constant coma:  std_logic_vector (7 downto 0) := "00110100";
constant punto: std_logic_vector (7 downto 0) := "01110100";
constant retorno:   std_logic_vector (7 downto 0) := "10110000";
constant avance:    std_logic_vector (7 downto 0) := "01010000";
constant Cm:    std_logic_vector(7 downto 0) := "11000010";
constant porc:  std_logic_vector(7 downto 0) := "10100100";
constant H: std_logic_vector(7 downto 0) := "00010010";
constant T: std_logic_vector(7 downto 0) := "00101010";
constant esp:   std_logic_vector(7 downto 0) := "00000100";
constant o: std_logic_vector(7 downto 0) := "11110110";
begin

    process (bin) is
    variable res, den_1,den_2, d_1,d_2, c: integer;
    variable aux_c, aux_d, aux_u:   std_logic_vector (7 downto 0);

    begin

        c := conv_Integer(bin);
        den_1 := 10;
        d_1 := 0;
        den_2 := 100;
        d_2 := 0;
```

FIGURA A.5.6. a) Código del Generador de mensajes (App de Datos)

```
41
42          for I in 1 to 10 loop
43
44              if (c>=den_2) then
45                  c := c - den_2;
46                  d_2 := d_2 +1;
47              end if;
48
49          end loop;
50
51          c := (conv_Integer(bin)-d_2*100);
52
53
54          for I in 1 to 10 loop
55
56              if (c>=den_1) then
57                  c := c - den_1;
58                  d_1 := d_1 +1;
59              end if;
60
61          end loop;
62
63          -- PARTE AÑADIDA
64
65          -- Partimos de decenas, unidades y decimas
66          -- Añadimos el prefijo "0011" (cógido ASCII que indica valor númerico)
67          aux_c := "0011" & conv_std_logic_vector(d_2,4);
68          aux_d := "0011" & conv_std_logic_vector((d_1),4);
69          aux_u := "0011" & conv_std_logic_vector((conv_Integer(bin)-d_1*10-d_2*100),4);
70
71          -- Cambio a little endian
72          for I in 0 to 7 loop
73              dec_c(I) <= aux_c(7-I);
74              dec_d(I) <= aux_d(7-I);
75              dec_u(I) <= aux_u(7-I);
76          end loop;
77
78          case sel is
79              when "10" => sal <= dec_c & dec_d & punto & dec_u & Cm & esp; -- ºC
80              when "01" => sal <= dec_c & dec_d & punto & dec_u & porc & esp; -- %H
81              when others => sal <= "00000000000000000000000000000000000000000000000000";
82          end case;
83
84          -- FIN DE PARTE AÑADIDA
85
86      end process;
87
88  end arch_calcula_doble;
```

FIGURA A.5.7. b) Código del Generador de mensajes (App de Datos)

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity memo_app_doble is
port(
    ent:    in std_logic_vector(47 downto 0);
    rst, hab, clk:  in std_logic;
    sal:    out std_logic
    );
end memo_app_doble;

architecture arch_memo_app_doble of memo_app_doble is
signal regis:   std_logic_vector(47 downto 0);
begin

    process(clk)
    variable cuenta:    integer range 0 to 48 := 48;
    begin

        if (clk'event and clk='1') then

            if rst='1' then -- Actualizo la memoria

                if cuenta=0 then
                    cuenta := 48;
                end if;

                for I in 0 to 47 loop
                    regis(I) <= ent(I); -- Escribo datos en memoria
                end loop;

                sal <= '1'; -- Salida normalmente a '1'

            elsif hab='0' then
                sal <= '0'; -- Bit de arranque

            else -- Recorro la memoria
                cuenta := cuenta - 1; -- El puntero de lectura se desplaza
                sal <= regis(cuenta); -- Leo la memoria y plasmo en la salida

            end if;

        end if;

    end process;

end arch_memo_app_doble;
```

FIGURA A.5.8. Código de la memoria de la App de datos

ANEXO B

Estación Meteorológica basada en FPGA

El objetivo fundamental de este TFG es la implantación de comunicaciones inalámbricas, entre los elementos de una estación meteorológica. Esta estación se ha desarrollado en otro TFG, que ha sido desarrollado en paralelo con éste. Evidentemente, aunque se trata de dos trabajos independientes, están relacionados entre sí.

Esto quiere decir que para el desarrollo del presente proyecto es necesario partir de la implementación de la estación meteorológica. En su esencia, la comunicación no requiere de ninguna implementación auxiliar, sin embargo, el objetivo de este trabajo se centra en la comunicación de diferentes dispositivos, fijando como una de las aplicaciones posibles la comunicación de información generada por la estación meteorológica.

Puesto que ambos se han realizado en paralelo, los circuitos desarrollados en el otro TFG incluidos en este proyecto no son los más actuales. Esto se debe a que en el momento de implementar la comunicación entre las diferentes estaciones, estas estaban correctamente desarrolladas pero se encontraban en un continuo desarrollo por parte de el responsable de dicho trabajo.

Durante el desarrollo del proyecto se realiza una pequeña introducción al funcionamiento de la estación meteorológica con el objetivo de comprender mejor el circuito de partida. El comportamiento del mismo sería el siguiente:

Cada dos segundos se envía una serie de valores binarios a través del pin de datos del sensor DHT22. El sensor responde con una serie de valores correspondientes a la temperatura y a la humedad según el protocolo del mismo (con un byte de comprobación de errores). Siguiendo las pautas de velocidad de transmisión, se almacenan los valores recibidos en un registro de desplazamiento, el cual se trata de la manera más adecuada posible para mostrar los valores aportados por el sensor a través de los displays, no sin antes comprobar su correcta recepción a través del byte de comprobación de errores. Podemos observar tanto temperatura como humedad por los displays, la variable mostrada depende del estado de dos

interruptores (uno activado y otro desactivado), que han sido establecidos en el '7' y en el '8' (Temperatura y humedad respectivamente). Por los displays se mostrarán las unidades y las décimas, mientras que los leds mostrarán las decenas según el código binario.

ANEXO C

Manuales de Usuario

C.1. Manual TFG

UVa

Departamento de Electrónica
Carlos Prados Sesmero
(+34) 625 75 34 43

Junio de 2018

©Manual de Usuario: Control de dispositivos externos, desde una FPGA, vía Bluetooth

Visión general

El presente documento incluye una serie de pautas para la puesta en marcha y utilización de los diferentes elementos y dispositivos utilizados en el TFG "Control de dispositivos externos, desde una FPGA, vía Bluetooth". Se describe la manera de configurar la comunicación entre los distintos elementos que constituyen la estación meteorológica, al igual que para ciertas aplicaciones previas a su desarrollo.

Materiales necesarios

Para obtener información útil sobre la estación meteorológica van a ser necesarios los siguientes elementos: Una FPGA MachXO2 1200-ZE, una FPGA MachXO2 7000-HE, dos módulos de Bluetooth HC-12, un módulo de Bluetooth HC-05 y un sensor de temperatura y humedad DHT22.

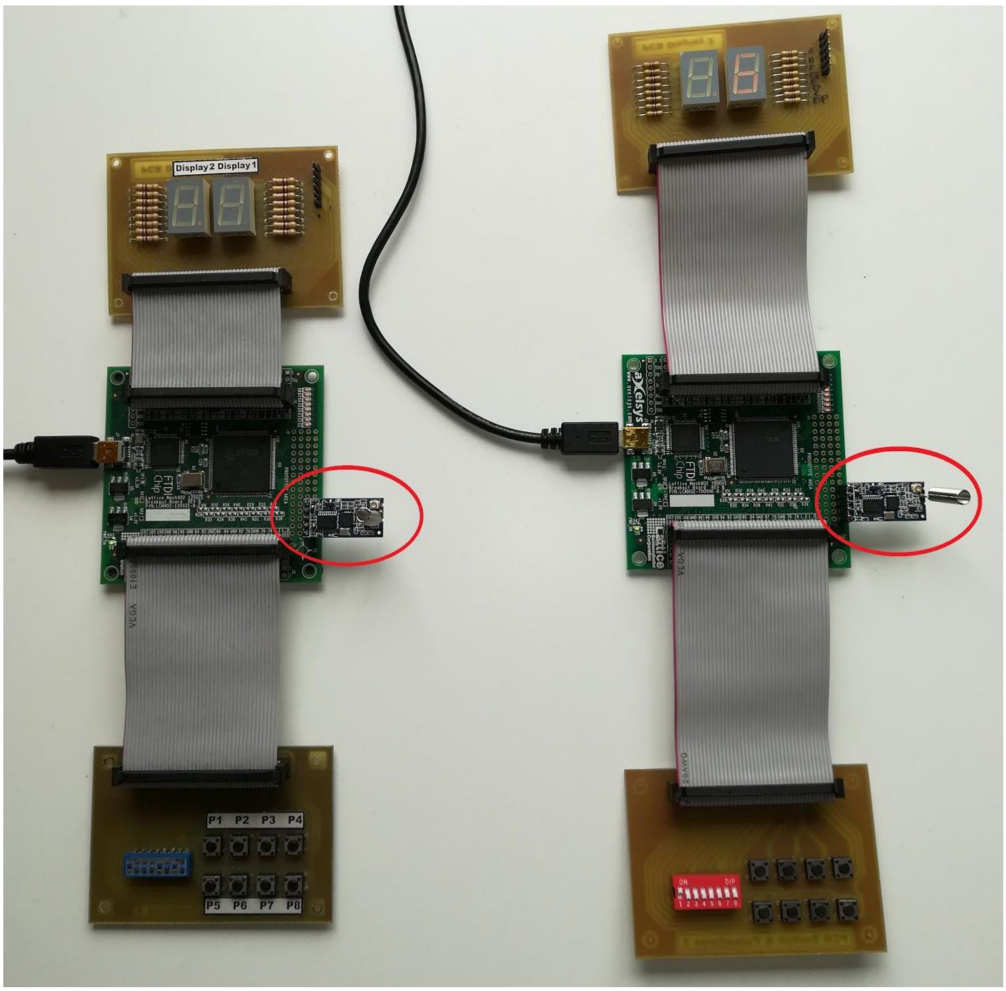
Aplicaciones

A continuación se presentan una serie de aplicaciones, las cuales tienen diferentes características y funcionamiento.

2

Comunicación básica

En este caso, se ha implantado una metodología de comunicación inalámbrica mediante un ejemplo básico de paso de información. Se utilizan, una MachXO2 1200-ZE, una MachXO2 7000-HE y dos módulos HC-12 (rodeados en rojo), conectados de la siguiente manera:



3

En la placa de la izquierda (1200-ZE, emisora) se pueden cargar uno de los siguientes archivos:

- /Interruptor sin FSM/Emisor: Sistema basado en eventos.
- /Interruptor sin FSM/Emisor_Memo: Sistema basado en tiempo.

En la placa de la derecha (7000-HE, receptora) se pueden cargar uno de los archivos siguientes:

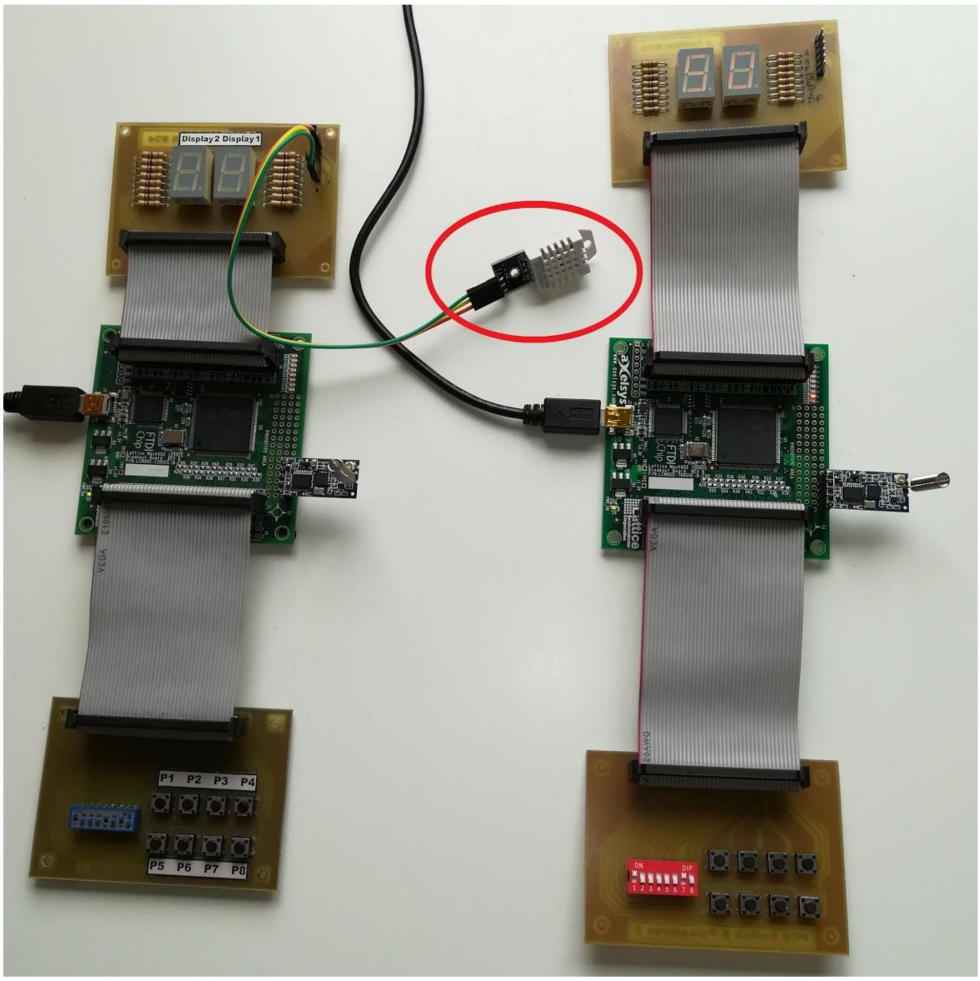
- /Interruptor sin FSM/Receptor: Sistema basado en eventos.
- /Interruptor sin FSM/Receptor_Memo: Sistema basado en tiempo, alternativa de diseño.

Ambos sistemas (emisor y receptor) tienen como habilitador el interruptor "1". En el dispositivo de la izquierda, se puede jugar con los interruptores "2-8", mientras se observa, en el dispositivo de la derecha, el número que aparece en el display de la derecha, que indica el interruptor más prioritario activado, es decir, el mayor. En dicho display hay un punto en una de sus esquinas, el cual se enciende cuando la información recibida es errónea.

Temperatura y humedad

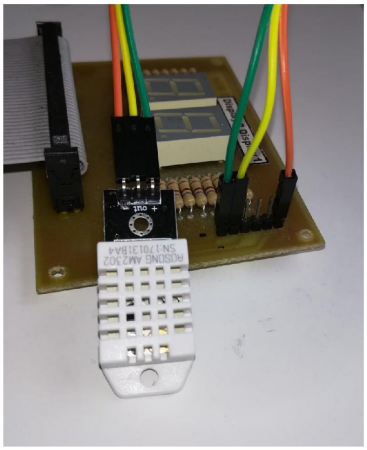
Como continuación de la aplicación anterior, se ha diseñado una comunicación, entre dos dispositivos FPGA, de datos relacionados con la estación meteorológica. Se utilizan, una MachXO2 1200-ZE, una MachXO2 7000-HE, dos módulos HC-12 y un sensor DHT22 (rodeado en rojo), conectados de la siguiente manera:

4



Estando el sensor conectado de la siguiente manera:

5

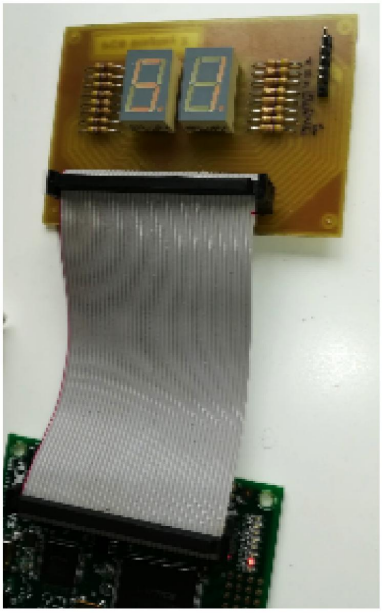


En la placa de la izquierda (emisora) se cargará el archivo llamado “/Temp_Hum/Emisor_Temp”, mientras que en el de la derecha el archivo “/Temp_Hum/Receptor_Temp”.

Ambos sistemas (emisor y receptor) tienen como habilitador el interruptor “1”. El dispositivo de la izquierda toma información del sensor y la envía cada dos segundos, mientras que el dispositivo de la derecha recibe dichos datos y los muestra, en caso de ser correctos. En los leds de la placa se observan las decenas según el código binario, mientras que en los displays se muestran las unidades y las décimas. Para mostrar la temperatura activaremos el interruptor “7”, por el contrario, activaremos el interruptor “8” para mostrar la humedad relativa.

Por ejemplo, con el interruptor “7” activado, se muestra la siguiente información:

6



Los leds muestran “0010”, es decir, un “2” como decenas. Los displays muestran “5” y “1”. Como resultado se obtiene una temperatura ambiente de 25.1ºC.

Estudio de los comandos AT

Se ha diseñado un comportamiento sobre la aplicación de comandos AT a nuestro módulo, de tal manera que se puedan modificar los parámetros de la comunicación vía Bluetooth. Se utilizan, una MachXO2 1200-ZE y un módulo HC-12.

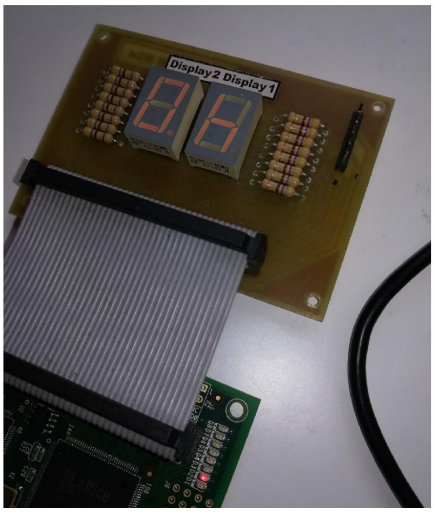
En la placa se cargará el archivo llamado “/Interruptor sin FSM/Comandos_AT_2”. El sistema tiene como habilitador el interruptor “1”. Cada uno de los pulsadores tiene asociado un comando AT, con la siguiente relación:

1. AT: Testeo del módulo.
2. AT+B19200: Establece la velocidad del puerto serie a 19200 baudios.
3. AT+C030: Establece el canal de comunicación al 30.

7

4. AT+FU2: Establece el modo de funcionamiento 2.
5. AT+P4: Establece la potencia transmitida a 8dBm.
6. AT+SLEEP: Inhabilita la transmisión por el puerto serie para el ahorro de energía.
7. AT+DEFAULT: Establece los parámetros por defecto.
8. AT+UPDATE: Inhabilita los comandos AT hasta que se suprima la alimentación del dispositivo.

Para más información, es recomendable consultar el datasheet del módulo HC-12. En los displays de la placa se observan las respuestas recibidas por parte del módulo, mientras que en los leds, se observa el reloj activo. Por ejemplo, si accionamos el pulsador “1”, recibimos el estado del dispositivo (OK):



Los leds muestran “010”, es decir, una frecuencia del puerto serie de 9.6kHz, mientras que si fuera “001” sería de 19.2kHz, y si fuera “100” sería de 4.8kHz.

Estación meteorológica completa

Esta aplicación es una combinación de las dos últimas, acogiendo las características y competencias de ambas. Se utilizan, una MachXO2 1200-ZE, una MachXO2 7000-HE, dos

8

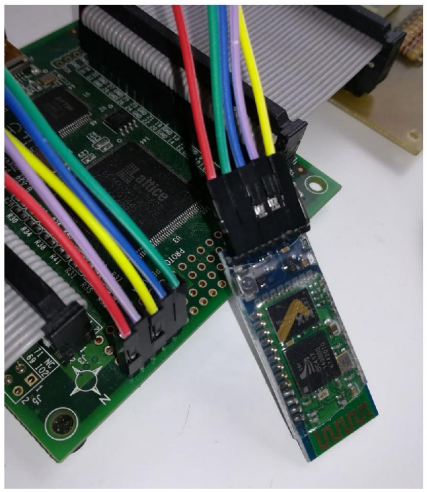
módulos HC-12 y un sensor DHT22, conectados de la misma manera que en el caso de Temperatura y Humedad.

En la placa de la izquierda se cargará el archivo llamado "/Temp_Hum/Emisor_Completo", mientras que en el de la derecha el archivo "/Temp_Hum/Receptor_Completo".

Ambos sistemas (emisor y receptor) tienen como habilitador el interruptor "1". El dispositivo de la izquierda toma información del sensor y la envía cada dos segundos, mientras que el dispositivo de la derecha recibe dichos datos y los muestra, en caso de ser correctos. Tiene una funcionalidad idéntica a la aplicación "Temperatura y humedad", sin embargo, ambos bloques pueden modificar los parámetros de comunicación mediante los comandos AT. Para garantizar la correcta comunicación, ambos deben trabajar con los mismos parámetros.

Aplicaciones móviles

Esta aplicación se utiliza para poder comunicar la estación meteorológica con las diferentes aplicaciones móviles desarrolladas. Se utilizan, una MachXO2 1200-ZE, un módulo HC-05 y un sensor DHT22. El módulo debe conectarse de la siguiente manera:



9

En la placa se cargará el archivo llamado “/Temp_Hum/App_elegir” en el caso de querer utilizar la aplicación móvil “App de Selección”, o el archivo “/Temp_Hum/App_Doble” en el caso de querer utilizar la aplicación “App de Datos”.

Contacto

Para ponerse en contacto con nosotros, hágalo a través del siguiente correo electrónico: carlos.prados@hotmail.com

C.2. App de Selección

UVa

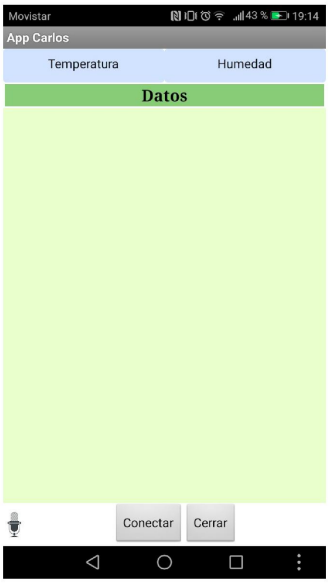
Departamento de Electrónica
Carlos Prados Sesmero
(+34) 625 75 34 43

Junio de 2018

©App de Selección

Visión general de la aplicación

La aplicación App de Selección es una aplicación para smartphones Android versión 2.3 o superiores, que permite hacer un seguimiento rápido del estado de la estación meteorológica manejada por una FPGA. La aplicación es nativa, por lo que solamente funcionará con SO Android y nos aportará información sobre la temperatura o humedad actual en la estación.



Descarga de la aplicación

Esta aplicación se puede descargar accediendo al Link de descarga. Donde se podrá descargar desde cualquier dispositivo electrónico. Una vez descargada ejecutaremos el archivo y se instalará automáticamente.

Compatibilidades de la aplicación

Esta aplicación mostrará por pantalla los datos que recibe desde cualquier dispositivo Bluetooth, aunque resulta conveniente utilizarla para la aplicación específica a la que se encuentra asignada.

2

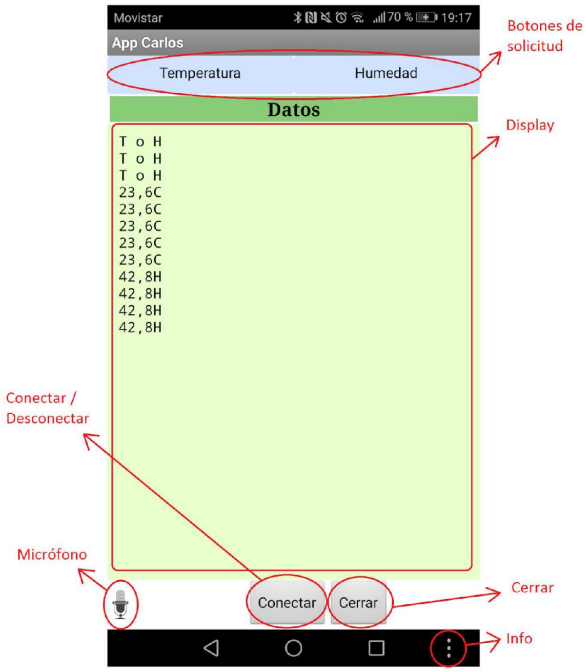
Navegación

Para utilizar la aplicación el usuario debe tener en cuenta que es necesario el uso del Bluetooth, de lo contrario la comunicación será inviable. Cuando arrancamos la aplicación se solicitará (en caso de no estar activo) la activación del Bluetooth.

La conexión al dispositivo HC-05 (utilizado para la comunicación con la estación meteorológica), solicitará el uso de contraseña. Normalmente esta clave será “0000” o “1234”.

Todas las acciones realizadas en la navegación por la aplicación serán señalizadas de una manera sonora, a través de la voz que da soporte Google.

Funcionamiento



- **Botones de solicitud**: Emitirán información vía Bluetooth para que la estación meteorológica transmita una variable en concreto (Temperatura o Humedad).
- **Display:** Muestra la información recibida. Cada 40 segundos se vaciará de datos.
- **Micrófono:** Permite solicitar una variable mediante el habla (a través de reconocimiento de voz).
- **Cerrar:** Cierra la aplicación.
- **Conectar/Desconectar:** Establece o suprime en el enlace Bluetooth con dispositivos.
- **Info:** Aporta información del desarrollo de la aplicación.

3

Reconocimiento de voz

Cuando pulsamos el botón “Micrófono”, tenemos dos opciones (las mismas que botones de selección). Podremos decir:

1. Temperatura.
2. Humedad.

En caso de que el reconocimiento de voz sea exitoso, la aplicación emitirá la información necesaria hacía la estación meteorológica para que emita la variable solicitada. En caso contrario, la aplicación indicará por audio que el reconocimiento no ha sido posible.

C.3. App de Datos

UVa

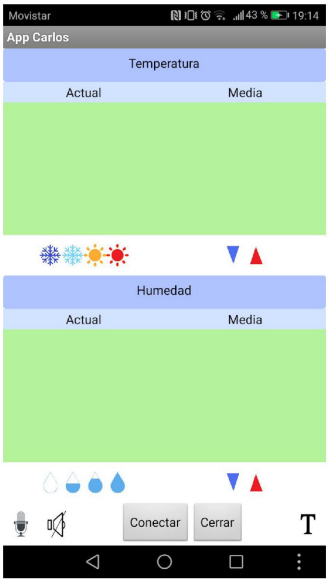
Departamento de Electrónica
Carlos Prados Sesmero
(+34) 625 75 34 43

Junio de 2018

©App de Datos

Visión general de la aplicación

La aplicación App de Datos es una aplicación para smartphones Android versión 2.3 o superiores, que permite hacer un seguimiento rápido del estado de la estación meteorológica manejada por una FPGA. La aplicación es nativa, por lo que solamente funcionará con SO Android y nos aportará información sobre las temperaturas y humedades actuales y medias en la estación, además de una gráfica temporal de las mismas.



Descarga de la aplicación

Esta aplicación se puede descargar accediendo al Link de descarga. Donde se podrá descargar desde cualquier dispositivo electrónico. Una vez descargada ejecutaremos el archivo y se instalará automáticamente.

Compatibilidades de la aplicación

Esta aplicación mostrará por pantalla los datos que recibe desde la estación meteorológica vía Bluetooth. En caso de ser utilizada en otro ámbito, la compatibilidad se ve enormemente reducida por falta de seguimiento del protocolo utilizado.

2

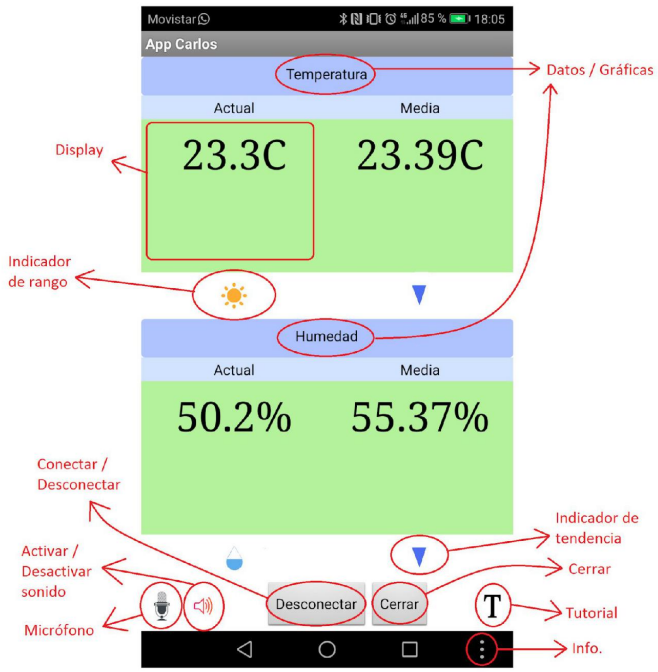
Navegación

Para utilizar la aplicación el usuario debe tener en cuenta que es necesario el uso del Bluetooth, de lo contrario la comunicación será inviable. Cuando arrancamos la aplicación se solicitará (en caso de no estar activo) la activación del Bluetooth.

La conexión al dispositivo HC-05 (utilizado para la comunicación con la estación meteorológica), solicitará el uso de contraseña. Normalmente esta clave será “0000” o “1234”.

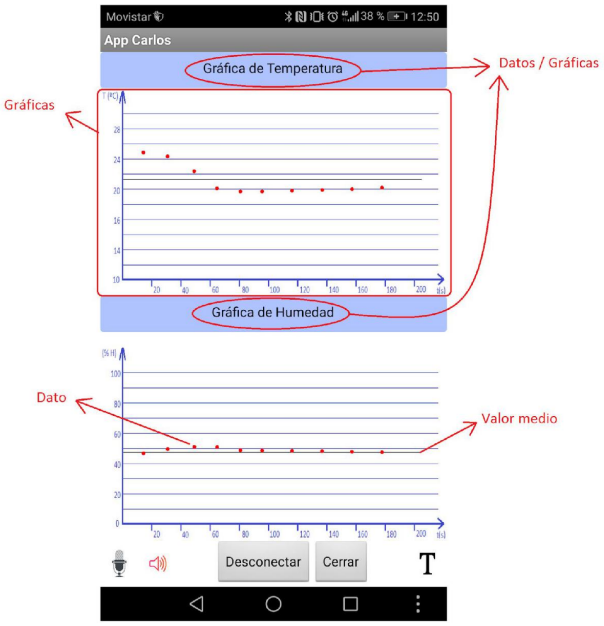
Todas las acciones realizadas en la navegación por la aplicación serán señalizadas de una manera sonora por defecto, a través de la voz que da soporte Google, aunque esta función puede ser desactivada.

Funcionamiento



- **Datos/Gráficas**: Cambio entre pantallas. En una de ellas se mostrarán los datos y en la otra las gráficas.
- **Display:** Muestran los datos actuales (a la izquierda) y medios (a la derecha), tanto de temperatura (arriba) como de humedad (abajo).
- **Indicador de rango:** Muestra una imagen u otra dependiendo del rango en el que se encuentren las variables.
- **Indicador de tendencia**: Indica la tendencia (aumentando o disminuyendo) actual de la temperatura y de la humedad.
- **Info:** Aporta información del desarrollo de la aplicación.

3



- **Conectar/Desconectar:** Establece o suprime en el enlace Bluetooth con dispositivos.
- **Activar/Desactivar:** Permite activar o desactivar el habla de la aplicación. En caso de estar desactivado inhibirá cualquier sonido de la aplicación.
- **Micrófono:** Permite solicitar una variable mediante el habla (a través de reconocimiento de voz).
- **Tutorial:** Recitará un tutorial de utilización de la aplicación donde se explican todos los aspectos de la misma.
- **Cerrar:** Cierra la aplicación.
- **Gráficas:** Se muestra, de una manera temporal, la temperatura y la humedad existentes en la estación meteorológica. Las gráficas disponen de **Datos** (representados mediante puntos rojos) y una línea horizontal que representa el **Valor medio**.

Reconocimiento de voz

Cuando pulsamos el botón “Micrófono”, tenemos varias opciones. Podremos decir:

1. Gráfica de temperatura.
2. Gráfica de humedad.
3. Datos de temperatura.
4. Datos de humedad.
5. Tutorial.

4

En caso de que el reconocimiento de voz sea exitoso, la aplicación navegará entre las diferentes pantallas o bien recitará el tutorial de la aplicación (en el caso de que el audio se encuentre activado). En caso contrario, la aplicación indicará por audio que el reconocimiento no ha sido posible.

Datos

Por motivos de configuración de la estación meteorológica y en caso de disponer de buena cobertura con la misma, se recibirán datos cada segundo (una vez temperatura y otra humedad). Por esta razón, los datos actuales mostrados por los displays tendrán una antigüedad máxima de 2 segundos.

Al igual pasaría con los valores medios, los cuales se actualizan cada vez que se recibe un nuevo dato.

Gráficas

Las gráficas están compuestas por una serie de puntos que van apareciendo con un periodo de 20 segundos. Estos puntos se van a ir desplazando a una velocidad programada de tal manera que cuando llegan al final de la gráfica (200 segundos de antigüedad), reaparecen al principio de la misma, actualizando el valor al actual.

Las gráficas disponen de una línea horizontal que indica el valor medio de las variables. Este valor se va a actualizar cada 20 segundos, momento en el que aparecerá un nuevo dato.

Contacto

Gracias a una continua utilización de la aplicación es posible notar ciertos aspectos, que a la hora del desarrollo no son visibles, necesitan ser mejorados o actualizados. Si nota alguna incidencia, deficiencia o punto de mejora en la aplicación, le agradecemos que se ponga en contacto con nosotros a través del siguiente correo electrónico: carlos.prados@hotmail.com

ANEXO D

Datasheet y documentos externos

D.1. MachXO2-1200ZE Breakout Board

MachXO2-1200ZE Breakout Board Evaluation Kit

User's Guide

January 2012
Revision: EB68_01.1

Introduction

Thank you for choosing the Lattice Semiconductor MachXO2™-1200ZE Breakout Board Evaluation Kit!

This user's guide describes how to start using the MachXO2-1200ZE Breakout Board, an easy-to-use platform for evaluating and designing with the MachXO2-1200ZE PLD. Along with the board and accessories, this kit includes a pre-loaded demonstration design. You may also reprogram the on-board MachXO2-1200ZE device to review your own custom designs.

Note: Static electricity can severely shorten the lifespan of electronic components. See the Storage and Handling section of this document for handling and storage tips.

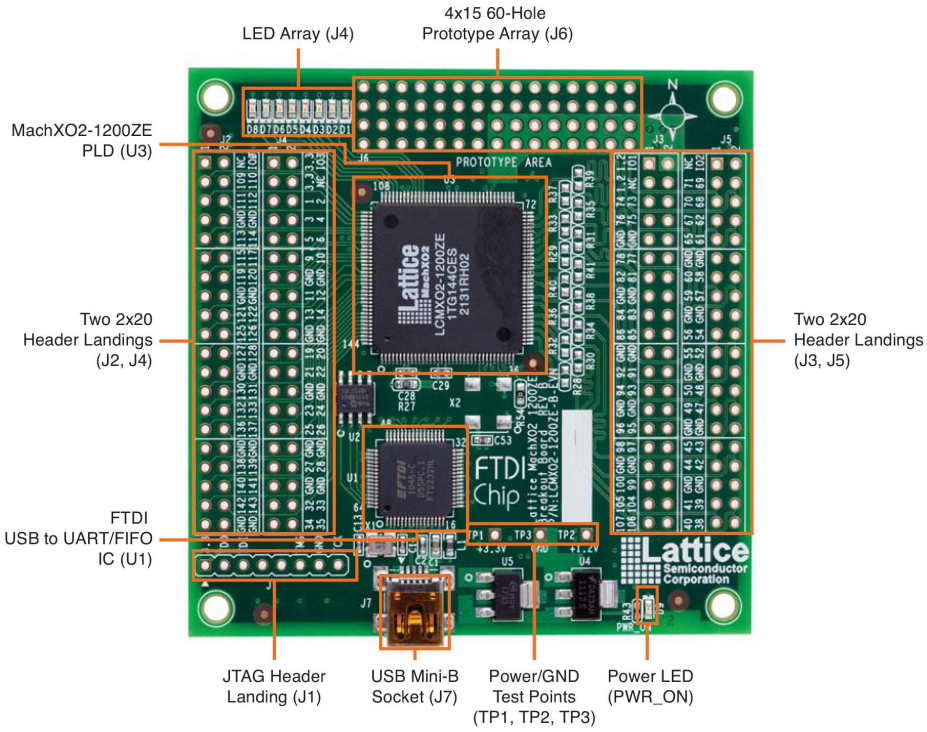
Features

The MachXO2-1200ZE Breakout Board Evaluation Kit includes:

- **MachXO2-1200ZE Breakout Board** – The board is a 3" x 3" form factor that features the following on-board components and circuits:
  - MachXO2-1200ZE PLD (LCMXO2-1200ZE-1TG144C)
  - USB mini-B connector for power and programming
  - Eight LEDs
  - 60-hole prototype area
  - Four 2x20 expansion header landings for general I/O, JTAG, and external power
  - 1x8 expansion header landing for JTAG
  - 3.3V and 1.2V supply rails
- **Pre-loaded Demo** – The kit includes a pre-loaded counter design that highlights use of the embedded MachXO2-1200ZE oscillator and programmable I/Os configured for LED drive.
- **USB Connector Cable** – The board is powered from the USB mini-B socket when connected to a host PC. The USB channel also provides a programming interface to the LCMXO2-1200ZE JTAG port.
- **Lattice Breakout Board Evaluation Kits Web Page** – Visit www.latticesemi.com/breakoutboards for the latest documentation (including this guide) and drivers for the kit.

The content of this user's guide includes demo operation, programming instructions, top-level functional descriptions of the Breakout Board, descriptions of the on-board connectors, and a complete set of schematics.

Figure 1. MachXO2-1200ZE Breakout Board, Top Side



Storage and Handling

Static electricity can shorten the lifespan of electronic components. Please observe these tips to prevent damage that could occur from electro-static discharge:

- Use anti-static precautions such as operating on an anti-static mat and wearing an anti-static wrist-band.
- Store the evaluation board in the packaging provided.
- Touch a metal USB housing to equalize voltage potential between you and the board.

Software Requirements

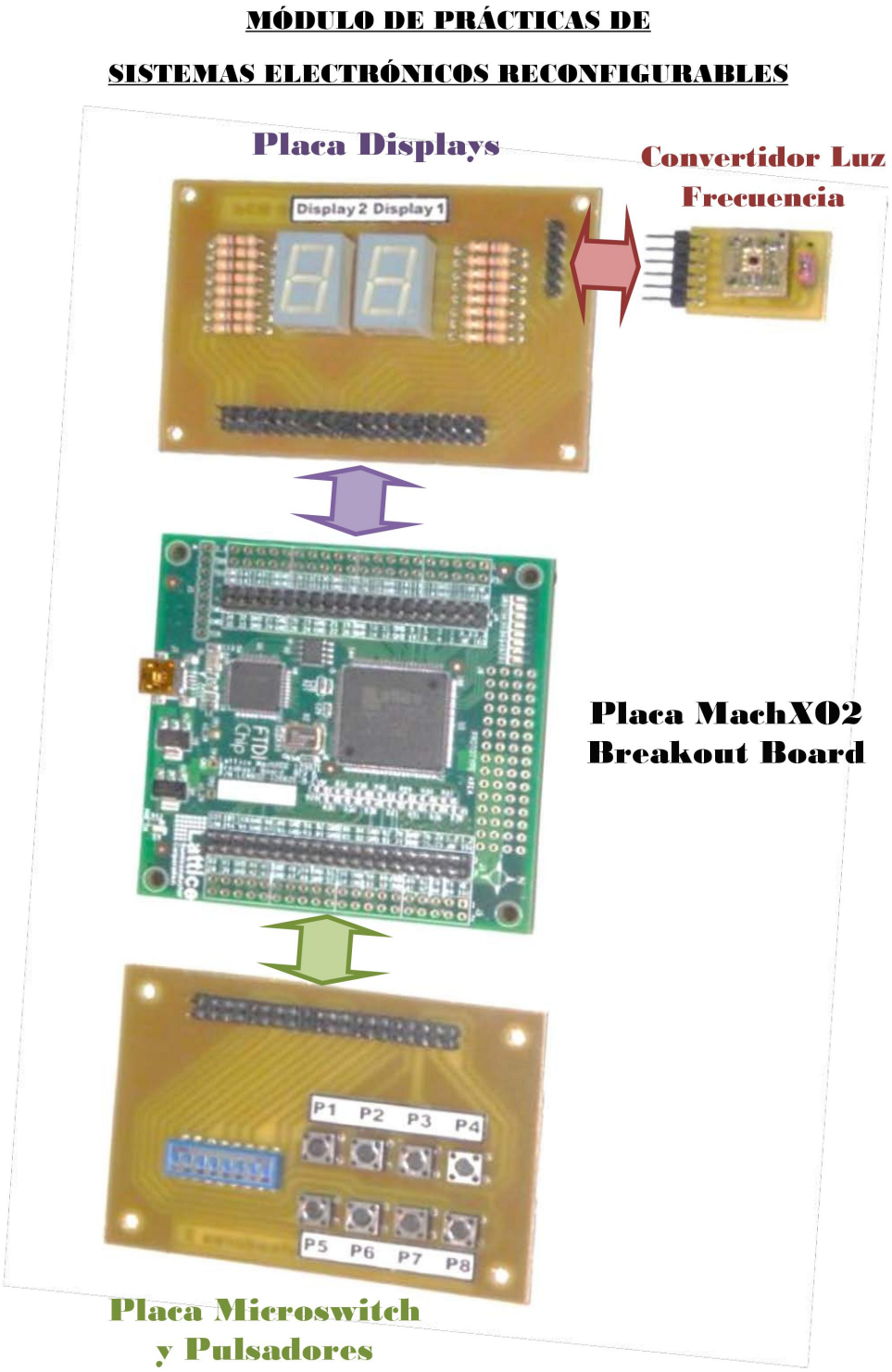
You should install the following software before you begin developing new designs for the Breakout board:

- Lattice Diamond® design software
- FTDI Chip USB hardware drivers (installed as an option within the Diamond installation program)

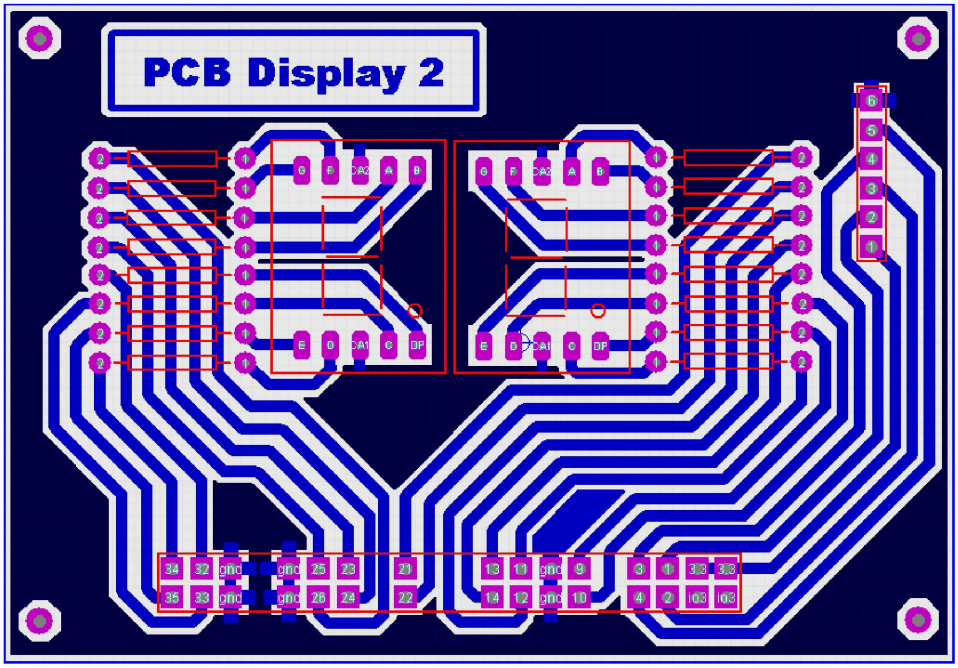
MachXO2-1200ZE Device

This board features the MachXO2-1200ZE PLD which offers embedded Flash technology for instant-on, non-volatile operation in a single chip. Numerous system functions are included, such as a PLL and 64Kbits of embedded RAM plus hardened implementations of I^2C, SPI, timer/counter, and user Flash memory. Flexible, high performance I/Os support numerous single-ended and differential standards including LVDS, and also source synchro-

D.2. PCBs de la FPGA (I/O)



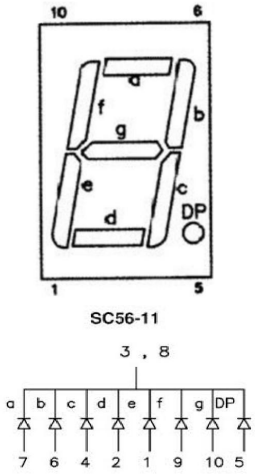
PLACA PCB DISPLAY 2



PINOUT DE LA FPGA ASIGNADO EN LA PLACA PCB DISPLAY 2:

DISPLAY 2 (A LA IZQDA)

A2	PIN7	23
B2	PIN6	25
C2	PIN4	33
D2	PIN2	34
E2	PIN1	35
F2	PIN9	26
G2	PIN10	24
DP2	PIN5	32



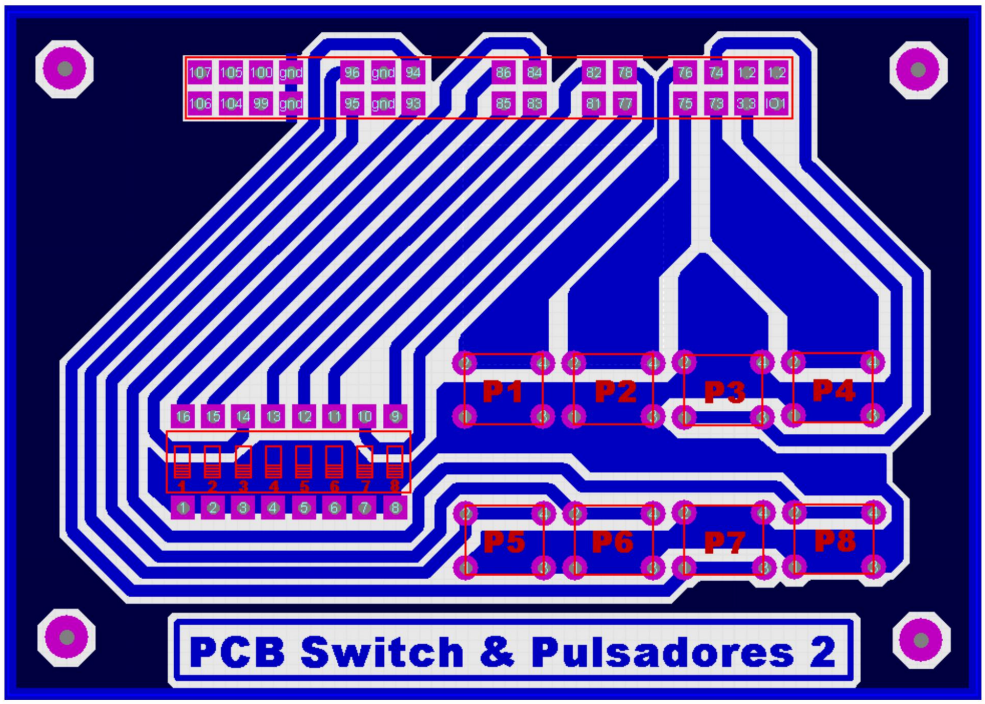
DISPLAY1 (A LA DRCHA)

A1	PIN7	21
B1	PIN6	22
C1	PIN4	11
D1	PIN2	10
E1	PIN1	13
F1	PIN9	12
G1	PIN10	14
DP1	PIN5	9

PINES AUXILIARES PARA OTRA PLACA DE AMPLIACIÓN (DE ABAJO A ARRIBA)

PIN1	VCC (3.3v)	PIN4	3
PIN2	1	PIN5	4
PIN3	2	PIN6	GND

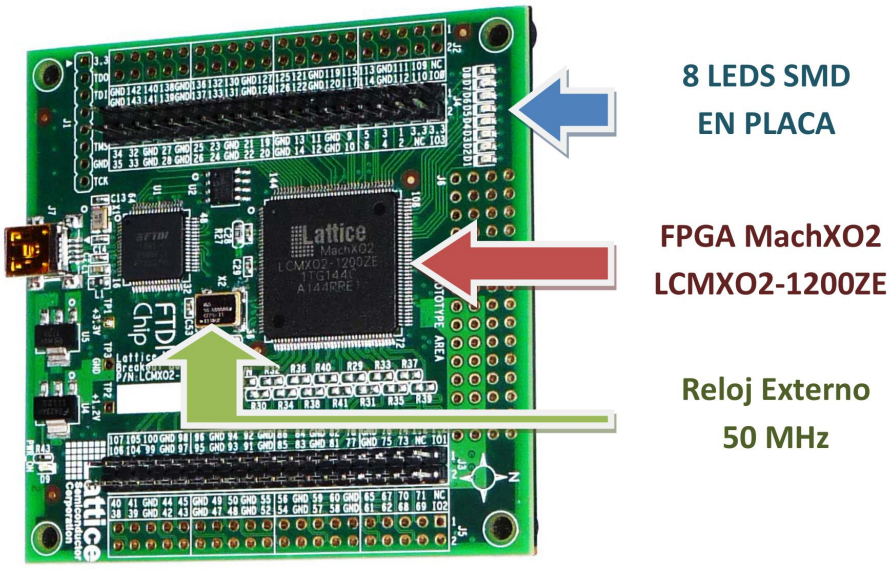
PLACA PCB SWITCH & PULSADORES 2



PINES MICROSWITCH		PINES PULSADORES	
SW1	86	PULS1	76
SW2	85	PULS2	75
SW3	84	PULS3	74
SW4	83	PULS4	73
SW5	82	PULS5	96
SW6	81	PULS6	95
SW7	78	PULS7	94
SW8	77	PULS8	93

NOTA: Al subir a 'ON' un microswitch o presionar un pulsador el estado de la entrada correspondiente en la FPGA se pone a nivel alto (3.3 V).

PLACA MachXO2 BREAKOUT BOARD



- **RELOJ EXTERNO**

Se ha agregado a la placa un reloj externo de 50 MHz. El reloj dispone de una patilla de habilitación CLK_EN que va al PIN 32 de la FPGA. A nivel alto el Cristal externo esta habilitado y a nivel bajo deshabilitado. La salida del reloj CLK_OUT va al PIN 27 de la FPGA.

CLK_OUT PIN 27 **CLK EN PIN 32**

- **LEDS ON BOARD**

D1 -> PIN 97	**D5 -> PIN 104**
D2 -> PIN 98	**D6 -> PIN 105**
D3 -> PIN 99	**D7 -> PIN 106**
D4 -> PIN 100	**D8 -> PIN 107**

D.3. Módulo HC-05

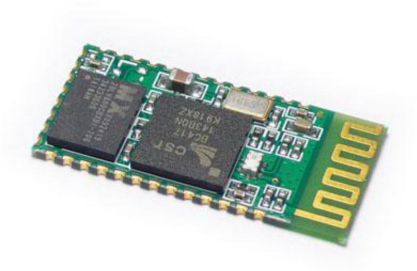
ITead Studio
Make Innovation easier

Tech Support: info@iteadstudio.com

HC-05

-Bluetooth to Serial Port Module

Overview



HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup.
Serial port Bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband. It uses CSR Bluecore 04-External single chip Bluetooth system with CMOS technology and with AFH(Adaptive Frequency Hopping Feature). It has the footprint as small as 12.7mmx27mm. Hope it will simplify your overall design/development cycle.

Specifications

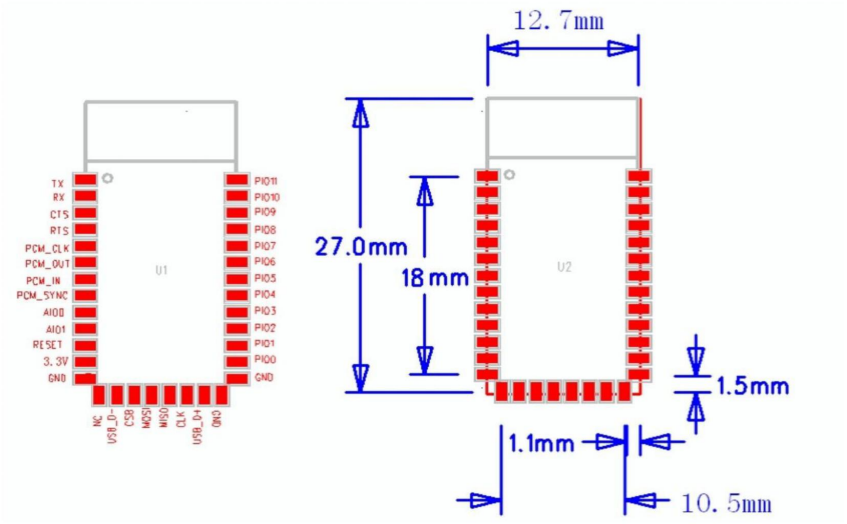
Hardware features

- Typical -80dBm sensitivity
- Up to +4dBm RF transmit power
- Low Power 1.8V Operation ,1.8 to 3.6V I/O
- PIO control
- UART interface with programmable baud rate
- With integrated antenna
- With edge connector

Software features

- Default Baud rate: 38400, Data bits:8, Stop bit:1,Parity:No parity, Data control: has. Supported baud rate: 9600,19200,38400,57600,115200,230400,460800.
- Given a rising pulse in PIO0, device will be disconnected.
- Status instruction port PIO1: low-disconnected, high-connected;
- PIO10 and PIO11 can be connected to red and blue led separately. When master and slave are paired, red and blue led blinks 1time/2s in interval, while disconnected only blue led blinks 2times/s.
- Auto-connect to the last device on power as default.
- Permit pairing device to connect as default.
- Auto-pairing PINCODE:"0000" as default
- Auto-reconnect in 30 min when disconnected as a result of beyond the range of connection.

Hardware



USB_+	20	Bi-Directional		
NC	14			
PCM_CLK	5	Bi-Directional	Synchronous PCM data clock	
PCM_OUT	6	CMOS output	Synchronous PCM data output	
PCM_IN	7	CMOS Input	Synchronous PCM data input	
PCM_SYNC	8	Bi-Directional	Synchronous PCM data strobe	

AT command Default:

How to set the mode to server (master):
1. Connect PIO11 to high level.
2. Power on, module into command state.
3. Using baud rate 38400, sent the "AT+ROLE=1\r\n" to module, with "OK\r\n" means setting successes.
4. Connect the PIO11 to low level, repower the module, the module work as server (master).

AT commands: (all end with \r\n)

1. Test command:

Command	Respond	Parameter
AT	OK	-

2. Reset

Command	Respond	Parameter
AT+RESET	OK	-

3. Get firmware version

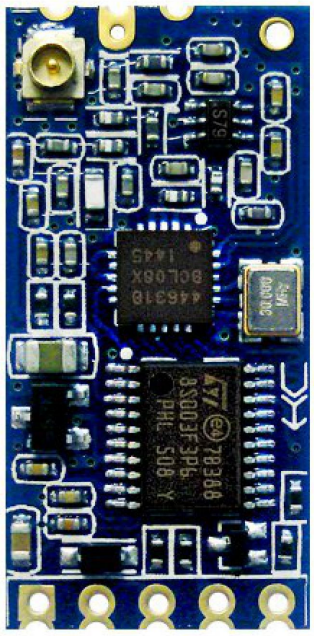
Command	Respond	Parameter
AT+VERSION?	+VERSION:<Param> OK	Param : firmware version

Example:
AT+VERSION?\r\n
+VERSION:2.0-20100601
OK

D.4. Módulo HC-12

HC-12 Wireless Serial Port Communication Module

User Manual version 2.3B
(updated from v1.1 English and v2.3 Chinese)



Product Applications

- Wireless sensor
- Community building security
- Robot wireless control
- Industrial remote control and telemetry
- Automatic data acquisition
- Container information management
- POS system
- Wireless acquisition of gas meter data
- Vehicle keyless entry system
- PC wireless networking

Document History:

2012/10	version 1.1	English, www.seeedstudio.com
2014/09	version 2.3	Chinese, www.wavesen.com
2016/01	version 2.3B	Translated v2.3 online and merged with v1.1 by RR

Product Features

- Long-distance wireless transmission (FU3: 1000m in open space, baud rate 5000bps in the air. FU4: 1800m in open space, baud rate 500bps in the air)
- Working frequency range (433.4-473.0MHz, with 100 communication channels)
- Maximum 100mW (20dBm) transmitting power (8 levels of power can be set)
- Four working modes, adapted to different application situations
- Built-in MCU performs communication with external device through serial port, no programming or configuration required for basic use
- The number of bytes transmitted continuously is unlimited (FU1 and FU3 modes only)
- Update software version through the serial port

Product Introduction

The HC-12 wireless serial port communication module is a new generation of multi-channel embedded wireless data transmission module. Its wireless working frequency band is 433.4-473.0MHz. Multiple channels can be set, with a channel stepping of 400kHz and a total of 100 channels. The maximum transmitting power of the module is 100mW (20dBm), the receiving sensitivity is -117dBm at a baud rate of 5000bps in the air. Communication distance is 1000m (FU3 mode at 4800bps serial speed) in open space, 1800m in FU4 mode at reduced baud rate and volume of data.

The module uses stamp hole packaging to allow for patch soldering, with dimensions of 27.8mm ×14.4mm ×4mm (including antenna cap, excluding spring antenna), making it is very convenient for incorporate into user specific applications. There is a PCB antenna socket ANT1 on the module, so an external 433MHz frequency band antenna can be attached via a coaxial cable; there is also an antenna solder eyelet ANT2 on the module, convenient to solder a spring antenna to. Select one of these antenna options according to usage requirements.

The module has an onboard MCU, eliminating the need for user to program the radio section separately, with transparent half-duplex serial transmission provided for receiving and sending serial port data. This making the HC-12 very easy to interface with. The module adopts multiple serial port transparent transmission modes that are user selected by AT commands according to usage requirements. The average working current of the four modes FU1, FU2, FU3, and FU4 in idle state are: 3.6mA, 80uA, 16mA, and 16mA respectively, while the maximum working current in any mode is 100mA (in the transmitting state).

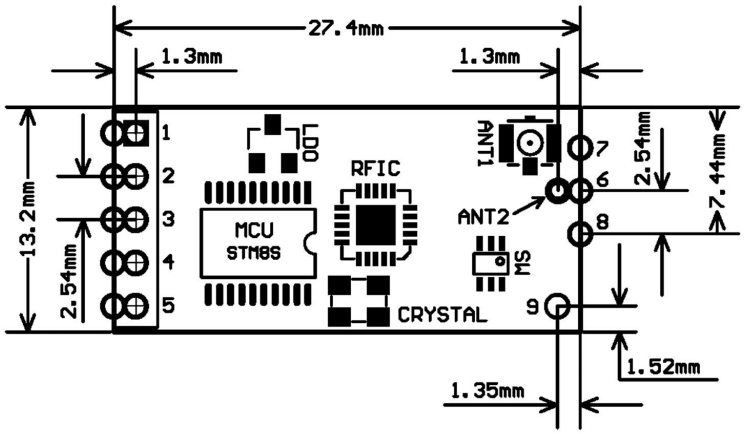
Product Configuration

Standard configuration of the HC-12 module only contains one 433MHz-frequency-band wireless communication module with IPEX20279-001E-03 standard RF socket. Optional accessories are 433MHz frequency band spring antenna, or IPEX-to-BNC coaxial cable and matching 433MHz frequency band omni-directional rubber antenna with BNC connector base. The user can purchase these according to their application requirements.

Technical Details

The HC-12 module uses a Silicon Labs Si4463 to provide the RF communications link. This is a high performance, low current, single-chip “EZRadioPRO” family transceiver with up to 20dBm (100mW) transmitting output power. The Si4463 communicates through an SPI bus with an STMicroelectronics STM8S003F3 8-bit MCU that runs the HC-12 firmware. The STM8S provides a transparent serial data interface for interfacing to the module, allowing two HC-12 modules to act like a wired TTL level serial cable without any attached hardware devices needing to be aware of the RF link. Serial port and transceiver configurations are held in onboard non-volatile flash memory.

Product Dimensions



Definition of Pins

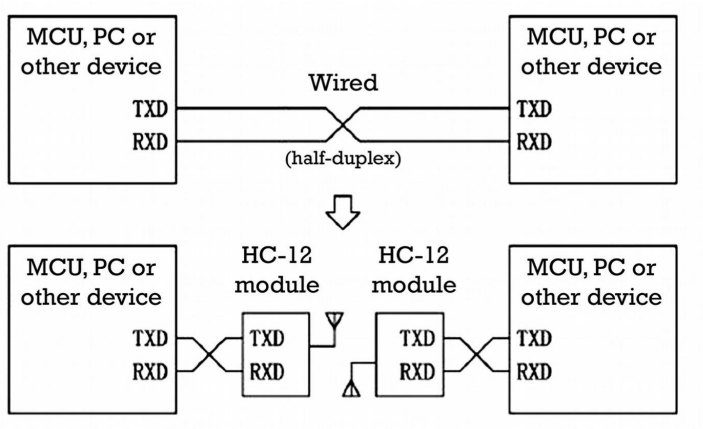
The HC-12 module can be patch soldered, or have a 2.54mm-spacing pin header attached and directly inserted onto the user's PCB. The module has nine pins in total, and one RF antenna socket (ANT1), with definitions as shown in the table below:

Pin	Definition	I/O direction	Notes
1	Vcc		Power supply input, DC3.2V-5.5V, with load capacity not less than 200mA. Note: if the module is working in the transmitting state for an extended time, it is suggested that a 1N4007 diode be connected in series if the supply voltage is greater than 4.5V, so as to avoid overheating the onboard LDO regulator
2	GND		Common ground
3	RxD	Input (weak pullup)	UART data input, TTL level. 1k resistor connected in series inside the module
4	TxD	Output	UART data output, TTL level. 1k resistor connected in series inside the module
5	SET	Input (10k pullup)	Parameter setting control pin, active low level. 1k resistor connected in series inside the module
6	ANT	Input/Output	433MHz antenna pin
7	GND		Common ground
8	GND		Common ground
9	NC		No connection, used in mechanical fixing, compatible with HC-11 module pin position
ANT1	ANT	Input/Output	IPEX20279-001E-03 antenna socket
ANT2	ANT	Input/Output	433MHz spring antenna solder eyelet

Pins 1-6 each have two bonding pads, with the outer half-hole bonding pads intended for patch soldering. When the inner bonding pad ANT2 of Pin 6 is used for connection, the spring antenna can be soldered here by hand. The inner round-hole bonding pads of Pins 1-5 may then be used to solder a 2.54mm-spacing pin header that can be plugged into a PCB socket.

Wireless Serial Port Transparent Transmission

(1) Simple introduction of working principle



As shown in the above diagram, two HC-12 modules can be used in place of physical wiring to replace a wired half-duplex serial communications link carrying TTL level signals. The left device sends serial port data to the module, and after the RxD port of the left module receives the serial port data, it will automatically send the data over the air via radio wave. The right module receives the data, and restores the serial port data originally sent by the left device and sends it out TxD. It is the same from right to left. Only a half-duplex link is available between modules, as they can not receive and send data over the air at the same time.

(2) Serial port transparent transmission

The HC-12 module has four serial port transparent transmission modes, expressed as FU1, FU2, FU3, and FU4. In operation, these modes hide all the details of wireless communications from attached devices. The factory default working mode of the system is FU3 full-speed mode, and in this mode the baud rate in this air is automatically adjusted according to baud rate that the serial port has been set to. The usable communication distance will be the farthest at the lowest baud rate. Different modes can not transmit data to each other, and the user should select the optimal mode according to practical circumstances.

The modules are usually operated in pairs, with data transmitted by means of a half-duplex link. For successful wireless transmission, the transparent transmission mode, serial port baud rate, and wireless communication channel of the two paired modules must be set the same. The factory default module setting are: FU3, 9,600bps (8N1: 8 data bits, no parity, 1 stop bit), CH001 (433.4MHz), 20dBm power (100mW).

The number of bytes that can be continuously sent to the serial port of the module is unlimited in modes FU1 and FU3. However, considering ambient interference and other factors, if thousands of data bytes are sent continuously, some number of bytes may be lost. Therefore, the attached devices at each end of the link should have some sort of response and resending mechanism to avoid information loss.

(3) The four serial port transparent transmission modes

When the HC-12 module leaves the factory, its default serial port transparent transmission mode is FU3. In this mode the module remains in full-speed state, with an idle current of about 16mA. The module automatically adjusts the baud rate of wireless transmission in the air according to the serial port baud rate,

with the corresponding relationship as shown in the table below:

Serial port baud rate	1200 bps	2400 bps	4800 bps	9600 bps	19,200 bps	38,400 bps	57,600 bps	115,200 bps
Baud rate in the air	5000bps		15,000bps		58,000bps		236,000bps	

To get the maximum communication distance, the serial port baud rate should be set to be low (1200bps or 2400bps). For short-time transmission of mass data, the serial port baud rate may be set high, but be aware that the communication distance will be reduced accordingly.

The receiving sensitivity of the module at different baud rates in the air is as shown in the table below:

Baud rate in the air	5000bps	15,000bps	58,000bps	236,000bps
Wireless receiving sensitivity	-117dBm	-112dBm	-107dBm	-100dBm

Generally, every time the receiving sensitivity is reduced by 6dB, the communication distance will be reduced by half.

When the "SET" pin of the module is pulled low, the serial port transparent transmission mode and other parameters can be set through AT commands (see the introduction in the following chapter for details).

FU1 mode is a moderate power saving mode, with an idle working current of about 3.6mA. In this mode, the module can also be set to any of the eight serial port baud rates shown in the above table, but the baud rate in the air is a uniform 250,000bps.

FU2 mode is an extreme power saving mode, with an idle working current of about 80uA. In this mode, the module only supports baud rates of 1200bps, 2400bps, and 4800bps, with the baud rate in the air uniform at 250,000bps. If the module is subsequently set to any other serial port baud rate, the module will not be able to conduct wireless communication normally.

When the module is set to FU2 mode, if the currently set baud rate exceeds 4800bps it will be automatically reduced to 4800bps. In FU2 mode, the sending time interval of data packets can not be too short, otherwise data will be lost. It is suggested that the sending time interval between data packets should be no less than 1 second.

FU4 mode is useful for maximum range, up to 1.8km. Only a single baud rate of 1200bps is supported, with the in the air baud rate reduced to 500bps for improved communication distance. This mode can only be used for small amounts of data (each packet should be 60 bytes or less), and the time interval between sending packets must not be too short (preferably no less than 2 seconds) in order to prevent loss of data.

The following table gives typical reference values for the various modes:

Mode	FU1	FU2	FU3	FU4	Remarks
Idle current	3.6mA	80uA	16mA	16mA	Average value
Transmission time delay	15-25mS	500mS	4-80mS	1000mS	Sending one byte
Loopback test time delay 1	31mS				Serial port baud rate 9600, sending one byte

Loopback test time delay 2	31mS				Serial port baud rate 9600, sending 10 bytes
Operating range at full power (20dBm)	100m	100m	600m at 9600bps 1000m at 2400bps	1800m at 1200bps	Clear line of sight between modules under ideal conditions

Note: Loopback test time delay means the round trip time taken for data that is sent to the input (RxD pin) of one module, to begin to emerge from the output (TxD pin) of the same module, where a second (remote) module has been configured with the TxD and RxD pins connected together.

Module Parameter Setting AT Commands

AT commands are used to set module parameters and switch between module functions when the module is in command mode. After being set, these changes will become valid only after exiting from command mode. Parameters are stored in onboard non-volatile flash memory, so will not be lost when power is removed.

(1) Entering command mode

There are two ways to enter command mode:

1. while energized, pull Pin 5 ("SET") low,
   wait 40ms for command mode to engage

2. disconnect the power supply,
   connect Pin 5 ("SET") to GND,
   re-energize the module

Note: pin 5 has a 10k pullup resistor connected internally, allowing the pin to be driven by an open-collector output from an attached device.

Either of the above two methods will place the module in command mode ready to accept AT commands; releasing pin 5 ("SET") in either case exits from command mode. If the module settings have changed after exiting from command mode, it will be switched to the new settings within 80ms.

When the second method (pin 5 "SET" tied to ground before power is applied), the module always enters command mode with the serial port configured for 9600bps, 8 data bits, no parity, 1 stop bit, irrespective of any previously configured settings.

(2) Command instructions

- AT

Test command. Send command "AT" to the module, and the module returns "OK".

- AT+Bxxxx

Change the serial port baud rate. The baud rate can be set to 1200bps, 2400bps, 4800bps, 9600bps, 19,200bps, 38,400bps, 57,600bps, or 115,200bps. The default value is 9600bps.

e.g: To set the serial port baud rate of the module to 19,200bps, send command "AT+B19200" to the module, and the module will return "OK+B19200". After exiting from command mode, the module will begin to communicate at 19,200bps.

- AT+Cxxx

Change wireless communication channel, selectable from 001 to 127 (for wireless channels exceeding 100, the communication distance cannot be guaranteed). The default value for the wireless channel is 001, with a working frequency of 433.4MHz. The channel stepping is 400KHz, and the working frequency of channel

100 is 473.0MHz.

e.g: To set the module to work on channel 21, send command “AT+C021” to the module, and the module will return “OK+C021”. After exiting from command mode, the module will work on channel 21, with a working frequency of 441.4MHz.
Note: As the wireless receiving sensitivity of the HC-12 module is relatively high, when the serial port baud rate is greater than 9600bps five adjacent channels should be staggered for use. Even when the serial port baud rate is not greater than 9600bps, over short distances (less than 10m) also five adjacent channels should be staggered for use.

- AT+FUx

Change the serial port transparent transmission mode of the module. Four modes are available, namely FU1, FU2, FU3, and FU4. Only when the serial port speed, channel, and transparent transmission mode of two modules is set to be the same, can normal wireless communications occur. For more details, please see the above section “Wireless Serial Port Transparent Transmission”.

e.g: Send command “AT+FU1” to the module, and the module returns “OK+FU1”.

- AT+Px

Set the transmitting power of the module, with x selectable from 1 to 8. The corresponding transmitting power of the module is as shown below:

x value	1	2	3	4	5	6	7	8
Transmitting power of module	-1 dBm (0.8mW)	2 dBm (1.6mW)	5 dBm (3.2mW)	8 dBm (6.3mW)	11 dBm (12mW)	14 dBm (25mW)	17 dBm (50mW)	20 dBm (100mW)

The default value is 8, and the higher the transmitting power, the farther the possible wireless communication distance. When the transmitting power level is set to 1, the transmitting power is at the minimum. Generally speaking, every time the transmitting power is reduced by 6dB, the communication distance will be reduced by half.

e.g: Send command “AT+P5” to the module, and the module returns “OK+P5”. After exiting from command mode, the transmitting power of the module will be set to 11dBm.

- AT+Ry

Obtain a single parameter from the module, where y is any letter among B, C, F, and P, respectively representing: baud rate, communication channel, serial port transparent transmission mode, and transmitting power.

Example 1:
Send command “AT+RB” to the module, and if the module returns “OK+B9600” it is confirmed that the serial port baud rate of the module is 9600bps.

Example 2:
Send command “AT+RC” to the module, and if the module returns “OK+RC001” it is confirmed that the communication channel of the module is 001.

Example 3:
Send command “AT+RF” to the module, and if the module returns “OK+FU3” it is confirmed that the module is working in serial port transparent transmission mode FU3.

Example 4:
Send command “AT+RP” to the module, and if the module returns “OK+RP:+20dBm” it is confirmed that the transmitting power of module is set to 20dBm (100mW).

- AT+RX

Obtain all parameters from the module. Returns serial port transparent

transmission mode, serial port baud rate, communication channel, and transmitting power in that order.

e.g: Send command “AT+RX” to the module, and the module returns “OK+FU3\r\n OK+B9600\r\n OK+C001\r\n OK+RP:+20dBm\r\n”. (“\r\n” means return\newline)

- AT+Udps

Set data bits (d), parity (p), and stop bits (s) for serial port communication. For parity, N means none, O means odd check, and E means even check. For stop bits, 1 means one stop bit, 2 means two stop bits, and 3 means 1.5 stop bits.

e.g: To set the serial port format to eight data bits, odd parity, and one stop bit, send command “AT+U8O1” to the module. The module will return “OK+U8O1”.

- AT+V

Request firmware version information from the module.

e.g: Send command “AT+V” to the module, and the module returns “HC-12_V2.3”.

- AT+SLEEP

After receiving this command, the module will enter sleep mode upon exiting from command mode, with a working current of about 22uA. This mode doesn’t allow serial port data transmission. Upon entering command mode again the module will exit from sleep mode automatically.

e.g: When wireless data transmission is not needed, to save power send command “AT+SLEEP” to the module, and the module will return “OK+SLEEP”. Upon exit from command mode the working current will drop to about 22uA.

- AT+DEFAULT

Set serial port baud rate and configuration, communication channel, power, and serial port transparent transmission mode back to the factory default values.

e.g: Send command “AT+DEFAULT” to the module, and the module returns “OK+DEFAULT”, with the factory default values restored. The factory default serial port baud rate is 9600bps, 8 data bits, no parity, 1 stop bit, communication channel is 001, transmitting power is 20dBm, and serial port transparent transmission mode is FU3.

- AT+UPDATE

Puts the module in the state of waiting for a software update. After receiving this command the module will not respond to any further AT commands until power has been cycled.

Design Considerations

- Do not connect a light-emitting diode and resistor directly to the module's TxD output as this may affect serial port communication.

- If using a PC or MCU to dynamically modify the module parameters, after pulling pin 5 (“SET”) low wait at least 40ms before sending any AT commands to the module. After releasing pin 5 (“SET”), wait at least 80ms for the module to return to serial port pass-through mode.

- The HC-12 may require up to 100mA of current when transmitting. Ensure sufficient current is available - a USB bridge device may not be able to supply sufficient current. It is recommended that a reservoir capacitor be provided across the power supply of at least 22uF, preferably 1000uF.

edited by Robert Rozee, 15 January 2016

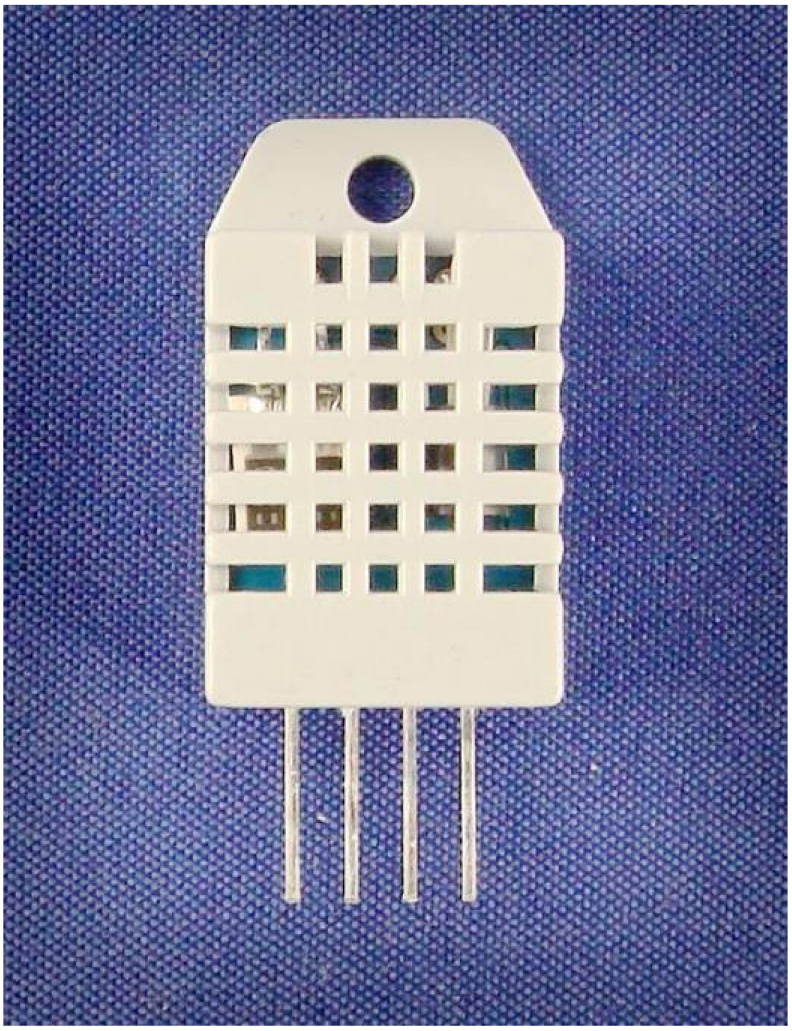
D.5. Sensor DHT22

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors

Digital-output relative humidity & temperature sensor/module

DHT22 (DHT22 also named as AM2302)



Capacitive-type humidity and temperature module/sensor

1

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors

1. Feature & Application:

* Full range temperature compensated * Relative humidity and temperature measurement
* Calibrated digital signal *Outstanding long-term stability *Extra components not needed
* Long transmission distance * Low power consumption *4 pins packaged and fully interchangeable

2. Description:

DHT22 output calibrated digital signal. It utilizes exclusive digital-signal-collecting-technique and humidity sensing technology, assuring its reliability and stability.Its sensing elements is connected with 8-bit single-chip computer.

Every sensor of this model is temperature compensated and calibrated in accurate calibration chamber and the calibration-coefficient is saved in type of programme in OTP memory, when the sensor is detecting, it will cite coefficient from memory.

Small size & low consumption & long transmission distance(20m) enable DHT22 to be suited in all kinds of harsh application occasions.

Single-row packaged with four pins, making the connection very convenient.

3. Technical Specification:

Model	DHT22
Power supply	3.3-6V DC
Output signal	digital signal via single-bus
Sensing element	Polymer capacitor
Operating range	humidity 0-100%RH; temperature -40~80Celsius
Accuracy	humidity +-2%RH(Max +-5%RH); temperature <+-0.5Celsius
Resolution or sensitivity	humidity 0.1%RH; temperature 0.1Celsius
Repeatability	humidity +-1%RH; temperature +-0.2Celsius
Humidity hysteresis	+-0.3%RH
Long-term Stability	+-0.5%RH/year
Sensing period	Average: 2s
Interchangeability	fully interchangeable
Dimensions	small size 14*18*5.5mm; big size 22*28*5mm

4. Dimensions: (unit----mm)

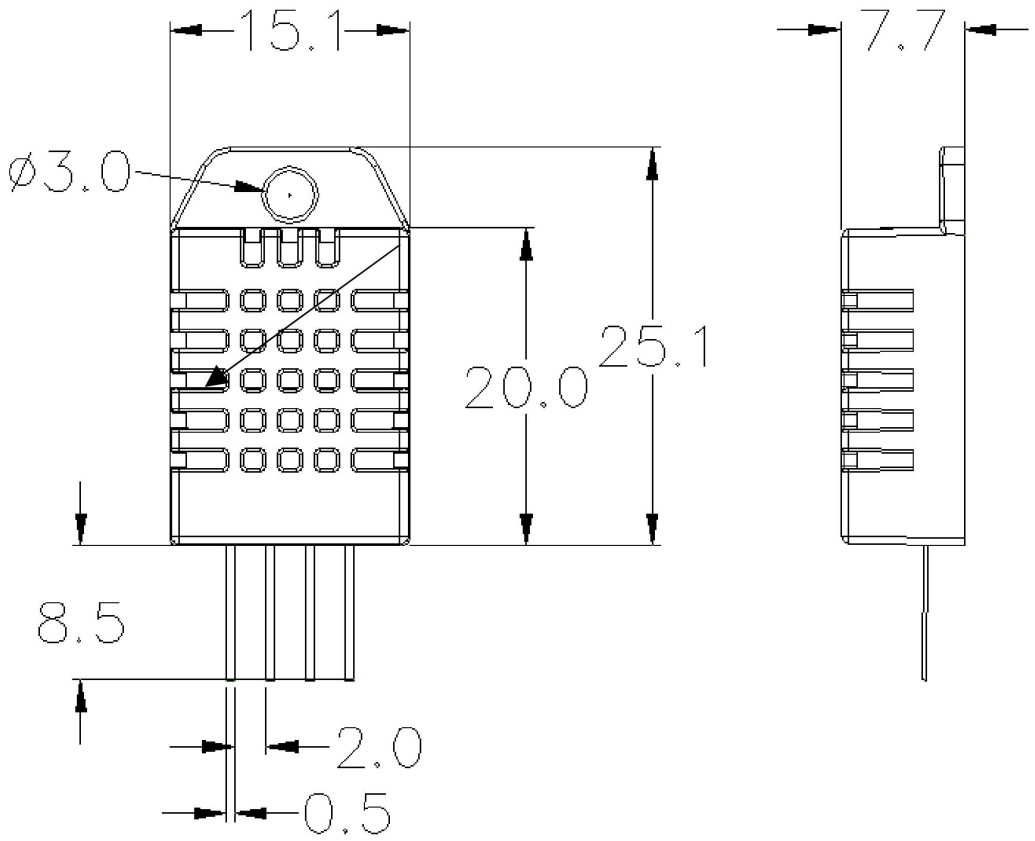
1) Small size dimensions: (unit----mm)

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors



Pin sequence number: 1 2 3 4 (from left to right direction).

Pin	Function
1	VDD----power supply
2	DATA--signal
3	NULL
4	GND

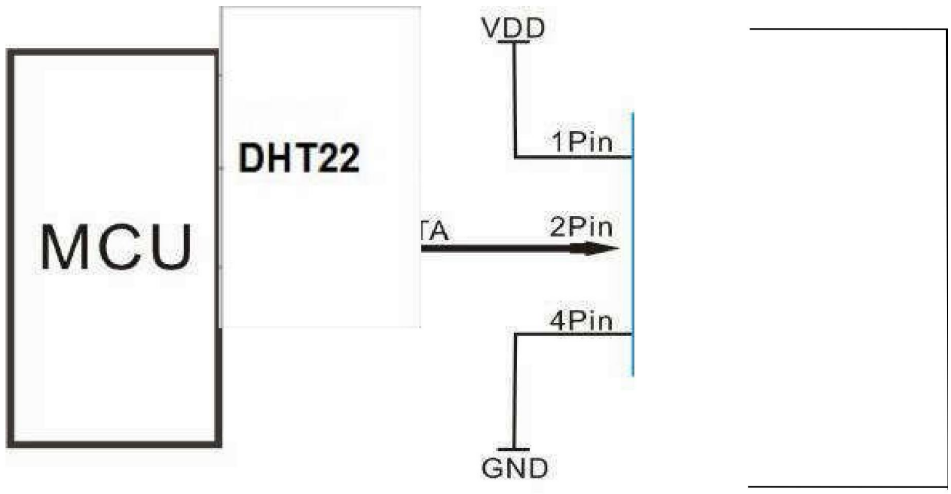
Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors

5. Electrical connection diagram:



3Pin---NC, AM2302 is another name for DHT22

6. Operating specifications:

(1) Power and Pins

Power's voltage should be 3.3-6V DC. When power is supplied to sensor, don't send any instruction to the sensor within one second to pass unstable status. One capacitor valued 100nF can be added between VDD and GND for wave filtering.

(2) Communication and signal

Single-bus data is used for communication between MCU and DHT22, it costs 5mS for single time communication.

Data is comprised of integral and decimal part, the following is the formula for data.

DHT22 send out higher data bit firstly!

DATA=8 bit integral RH data+8 bit decimal RH data+8 bit integral T data+8 bit decimal T data+8 bit check-sum

If the data transmission is right, check-sum should be the last 8 bit of "8 bit integral RH data+8 bit decimal RH data+8 bit integral T data+8 bit decimal T data".

When MCU send start signal, DHT22 change from low-power-consumption-mode to running-mode. When MCU finishs sending the start signal, DHT22 will send response signal of 40-bit data that reflect the relative humidity

Thomas Liu (Business Manager)

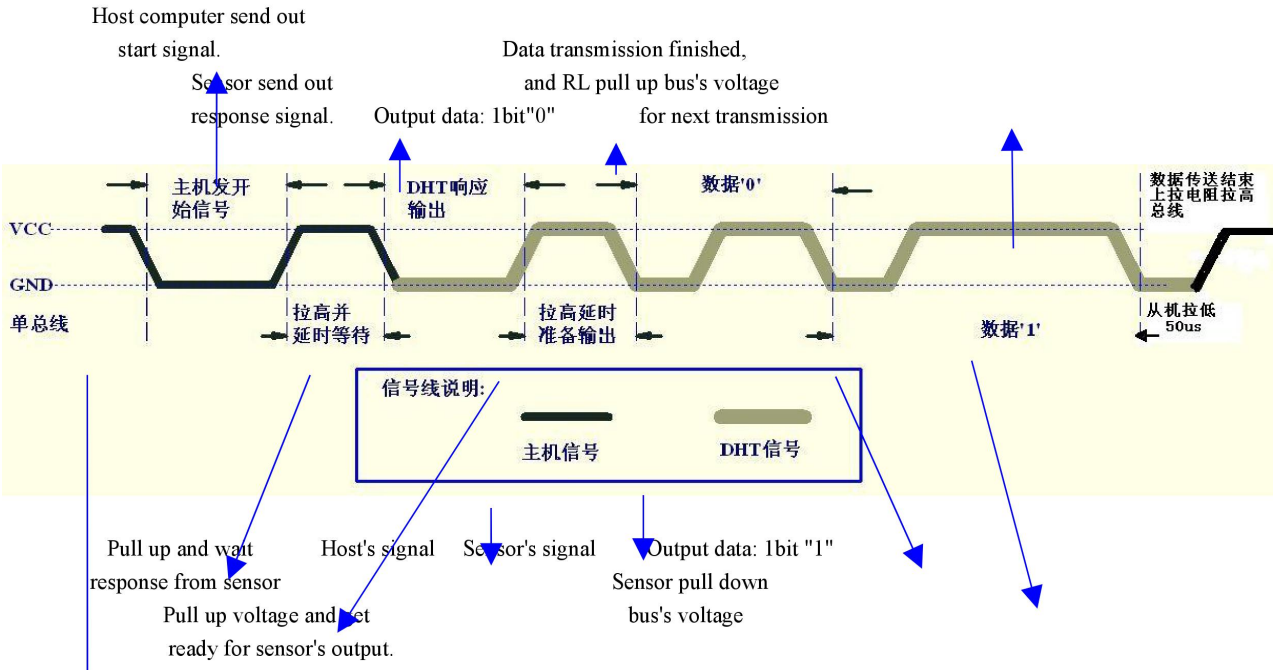
Email: thomasliu198518@yahoo.com.cn

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors

and temperature information to MCU. Without start signal from MCU, DHT22 will not give response signal to MCU. One start signal for one time's response data that reflect the relative humidity and temperature information from DHT22. DHT22 will change to low-power-consumption-mode when data collecting finish if it don't receive start signal from MCU again.

1) Check bellow picture for overall communication process:



---

2) Step 1: MCU send out start signal to DHT22

Data-bus's free status is high voltage level. When communication between MCU and DHT22 begin, program of MCU will transform data-bus's voltage level from high to low level and this process must beyond at least 1ms to ensure DHT22 could detect MCU's signal, then MCU will wait 20-40us for DHT22's response.

Check bellow picture for step 1:

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn