TAMAGOTCHI

Grupo C7

Jose Ángel Gumiel

Luis María Horvath

Christian Merino

Índice

Introducción

Un sistema digital es un conjunto de dispositivos destinados a la generación, transmisión, control, procesamiento o almacenamiento de señales digitales. A diferencia de un sistema analógico, un sistema digital es una combinación de dispositivos diseñados para manipular información representada en valores discretos.

Resumen

¿Qué hay que diseñar?

Se trata de un sistema digital que permite enviar y recibir comandos a través de la línea de serie RS-232, que interactúa con un software (tamagotchi).

La comunicación es bidireccional, pero los intercambios siempre empiezan desde la aplicación de escritorio. El tamagotchi debe responder al comando recibido y realizar una serie de cambios, según la orden que reciba.

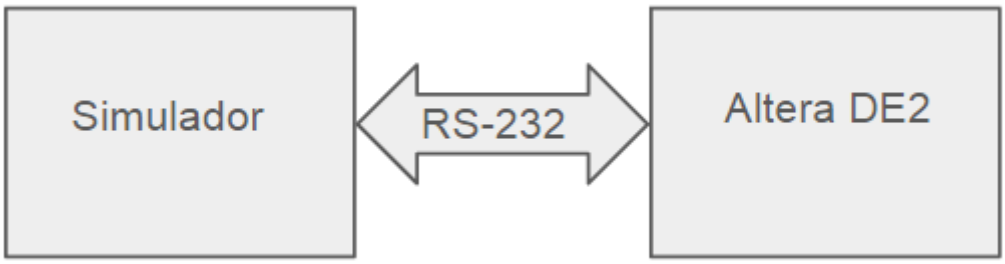
¿Cómo se trabajará?

Hardware	Software	RS-232
Placa Altera DE-2	Altera Quartus Modelsim Aplicación de Mirakonta	**Velocidad de transmisión:** 9600 bits/s **Bits de datos:** 8 **Bit de parada:** 1 Sin paridad Sin control de flujo

Diseño:

Se han propuesto los siguientes diseños para la implementación del sistema digital:

Esquema básico



Funcionamiento

1. Se envía una señal desde la aplicación. Ésta provoca el envío de un comando a través de RS-232 desde el PC hasta la placa Altera DE2. Los dispositivos deben estar sincronizados a través de un reloj.
2. La placa recibe esos comandos, para ello es necesario contar con dos módulos, un receptor de bits (rx_char) y un receptor de comandos (rx_comm). Ambos se juntan en el módulo rx_command.
3. Al haber muchos comandos, es necesario tener otro módulo que actúe como intérprete. (comm_gest).
4. El comm_gest, interactúa con el módulo tamagotchi, que es el encargado de realizar los cambios en el sistema, y los enviará al módulo tx_answer.
5. El tx_answer a su vez dará una respuesta legible por la aplicación de Mirakonta.

Objetivos del proyecto

Los objetivos del proyecto son los siguientes:

1. Aprender la metodología de trabajo para diseñar y construir un sistema digital. Saber compaginar una unidad de proceso, formada por bloques funcionales, con una unidad de control creada sobre el algoritmo ASM.

2. Comprender el funcionamiento y utilidad que nos ofrecen las siguientes herramientas:
    a. ModelSim: Crear y simular el funcionamiento de un sistema digital.
    b. SignalTap: Analizar las señales y debuggear el circuito.
    c. Quartus II: Compilar e implementar en placa el sistema diseñado.

3. Diseñar un sistema digital que pasándole por su entrada estándar una ristra de bits, los almacene en orden y comprenda de qué comando se trata dentro de una lista.

4. Entender lo vitales que son las señales de sincronización entre dispositivos y la importancia de saber gestionar los “handshakes” entre los diferentes módulos que forman nuestro sistema digital. En definitiva, conocer como intercomunicar diferentes módulos.

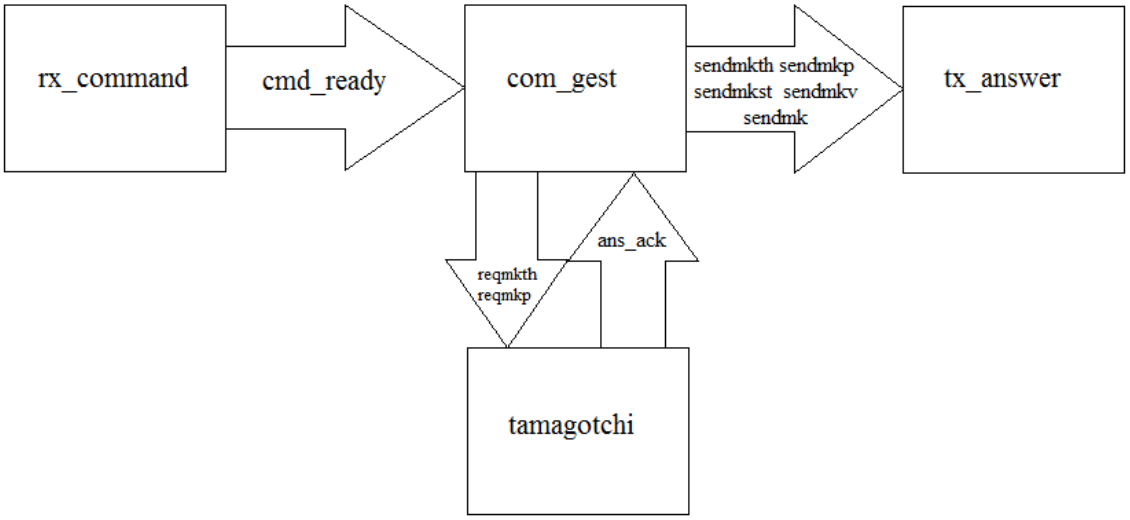
5. Por último, disponer de una implementación funcional en placa que responda a las peticiones del software de Mirakonta al finalizar el curso.

Desarrollo

En este apartado se mostrarán el diseño que se ha escogido para dar solución al problema planteado, y se explicará todo el proceso que se ha llevado durante estos meses, partiendo de la metodología hasta las herramientas utilizadas.

Diseño del prototipo

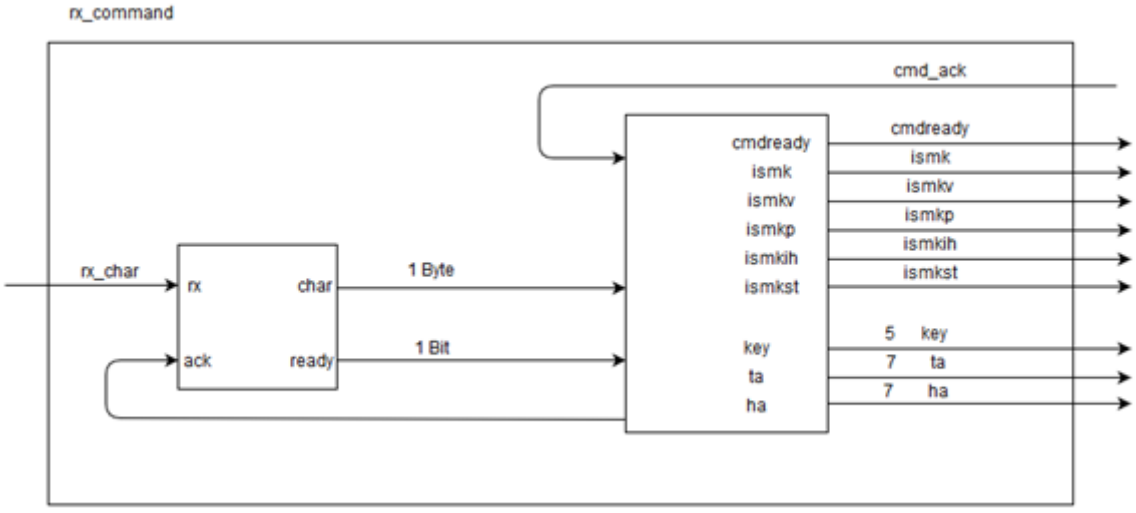
El sistema digital a diseñar tiene la siguiente estructura externa:



RX_COMMAND

Diseño del módulo

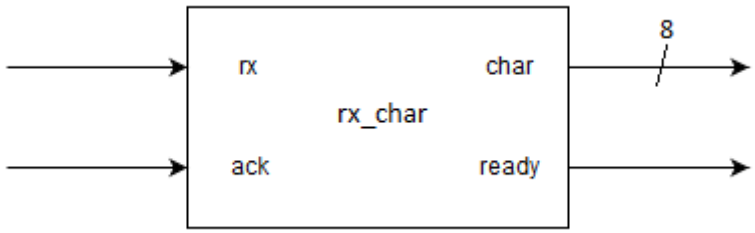
El sistema digital a diseñar tiene la siguiente estructura externa:



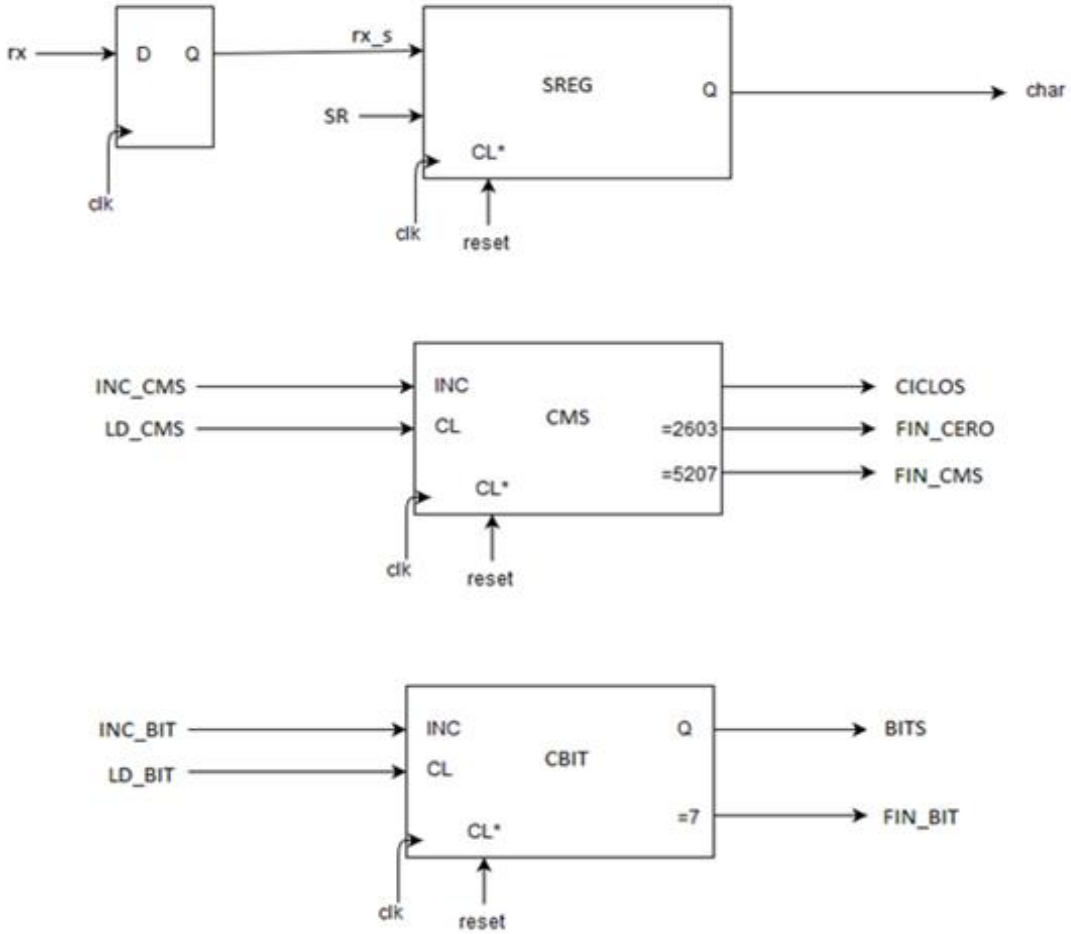
Para realizar dicho sistema digital se necesitará la ayuda de dos módulos internos:

RX_CHAR

El objetivo de este módulo es ir almacenando la secuencia de bits que va recibiendo por la entrada rx cuando la señal ack se lo pide. Las salidas son dos, en char se obtendrá un byte, representando la ristra de bits obtenidos por rx. La señal de salida ready avisará de que el sistema ha terminado de procesar los bits en rx.



El módulo rx_char sigue la siguiente arquitectura interna:

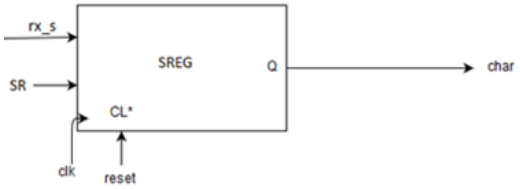


Módulos y señales implicadas

El componente **SREG** es un registro de desplazamiento cuya función es ir almacenando los bits sincronizados (generados por un biestable D) que vayan entrando al dispositivo hasta completar el Byte de la salida char. Esto permitirá su posterior lectura e interpretación.

Entradas:

- rx_s: Secuencia de bits de entrada.
- SR: Permite introducir un nuevo bit al registro.



Salidas:

- Q: Dato de 8 bits almacenado.

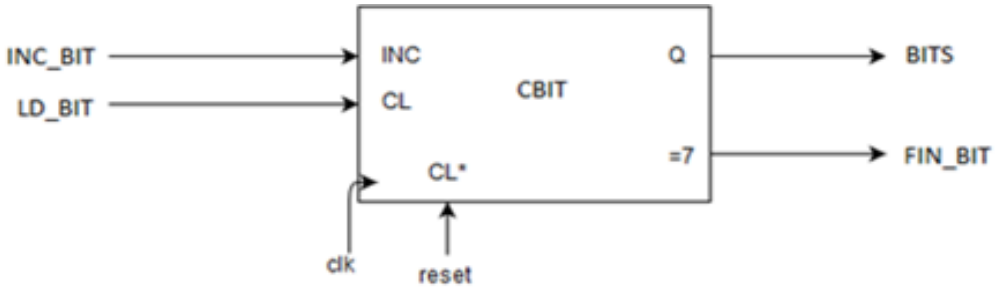
La función del módulo **CBIT** es ir controlando el número de bits que se han leído hasta el momento, para saber cuándo hay que dejar de leer. Dependiendo de la situación será necesario leer un número mayor o menor de bits, por lo que el "7" podría ser "n". No hay que olvidar que para la transmisión a través de RS232 hay un primer bit que determina el inicio de la transmisión y un último bit que determina el final, en este caso serían por lo tanto 10 bits (de 0 a 9).

Entradas:

- LD_BIT: Inicializa el contador de bits.
- INC_BIT: Incrementa el contador en 1.

Salidas:

- BITS: Número de bits contabilizados hasta el momento.
- FIN_BIT: Advierte cuando se han contabilizado 8 bits.



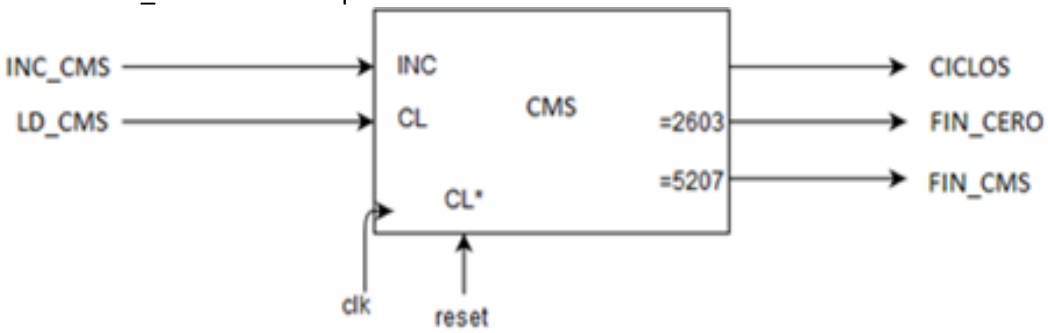
El módulo **CMS** es un contador de milisegundos. Advierte al sistema del número de milisegundos que han pasado, esto es necesario para poder situarse en la mitad de un bit. Ya que la mitad es la zona más significativa, hacerlo en la zona de transición sería un error. También es necesario éste módulo para contar la duración de los bits de entrada y poder almacenarlos correctamente. Se trata de algo de suma importancia, ya que se trata de un sistema síncrono.

Entradas:

- LD_CMS: Inicializa el contador de milisegundos.
- INC_CMS: Incrementa el contador en 1.

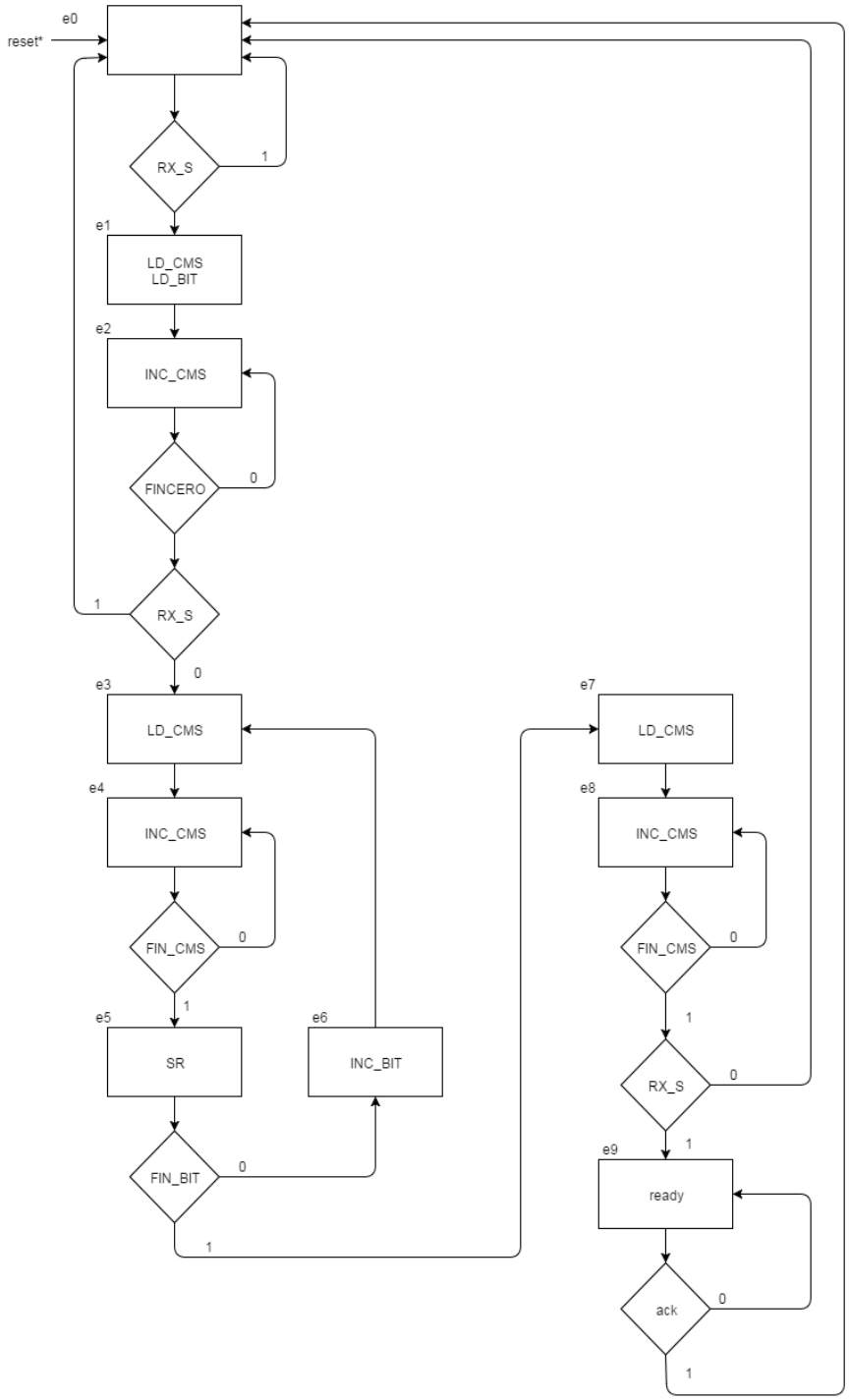
Salidas:

- CICLOS: Número de milisegundos transcurridos hasta la fecha.
- FIN_CERO: Advierte que se ha alcanzado la mitad de un bit.
- FIN_CMS: Advierte que se ha alcanzado la totalidad de un bit.



Todos estos bloques trabajan en paralelo siguiendo los pasos de unidad de control, que por razones de tamaño se encuentra en la siguiente página:

Unidad de Control y explicación



- La línea RS232 (RX) por defecto está a 1, en cuanto se recibe un 0, se resetean los contadores y se incrementan los ciclos para colocarse a la mitad del bit.

- No olvidemos que estamos en el mismo ciclo, puede parecer redundante el mismo start.

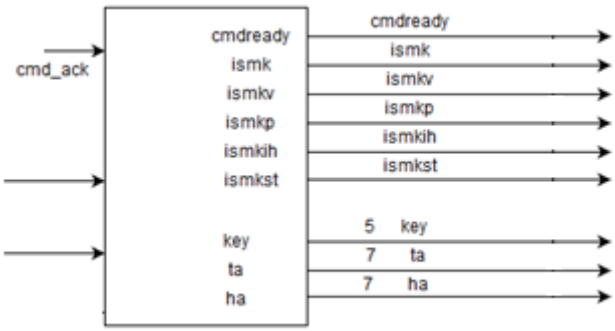
- Se realiza un clear de los contadores y empezamos a incrementarlos.
En cuanto observamos que hemos llegado al final del bit , lo cargamos con la señal SR al registro (shift register).

- Si no es el último, realizamos el mismo proceso.
Si llegamos al último bit hacemos un clear de ciclos hasta comprobar que recibimos el último.
Son 10 bits los que se usan en la transmisión:
- Bit Start (a 0 por defecto)
- 8 Bits de datos
- Bit Stop (a 1 por defecto)

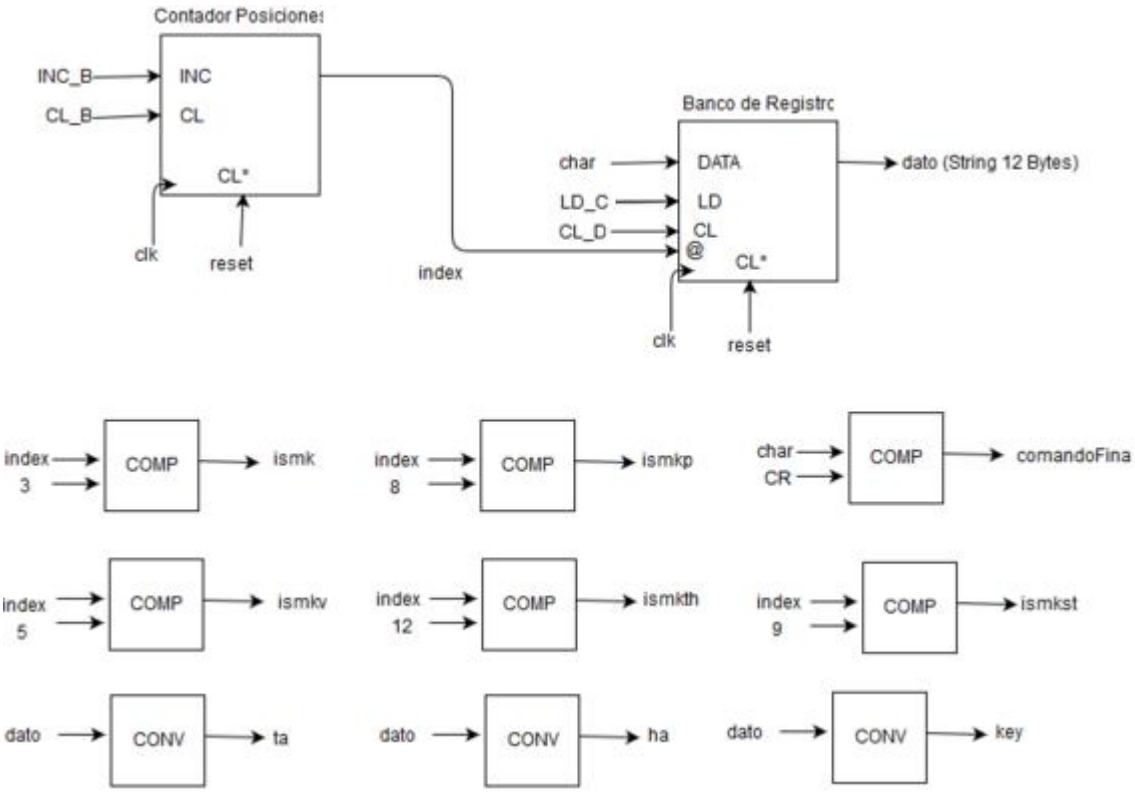
Una vez que se recibe el bit de Stop, volvemos al estado del principio

RX_COMM

Este módulo será el encargado de interpretar los mensajes que se recogen en rx_char. Transformará una secuencia de bits en código ASCII. Tendrá la capacidad de analizar qué tipo de comando se le ha enviado y actuar en consecuencia de una forma coherente, tal y como especifica el protocolo de comunicación establecido por Mirakonta y su simulador.



El módulo rx_comm sigue la siguiente arquitectura interna:



Por un lado hay un contador de posiciones, éste tiene la función contar el número de caracteres que van llegando y se deposita en la variable index.

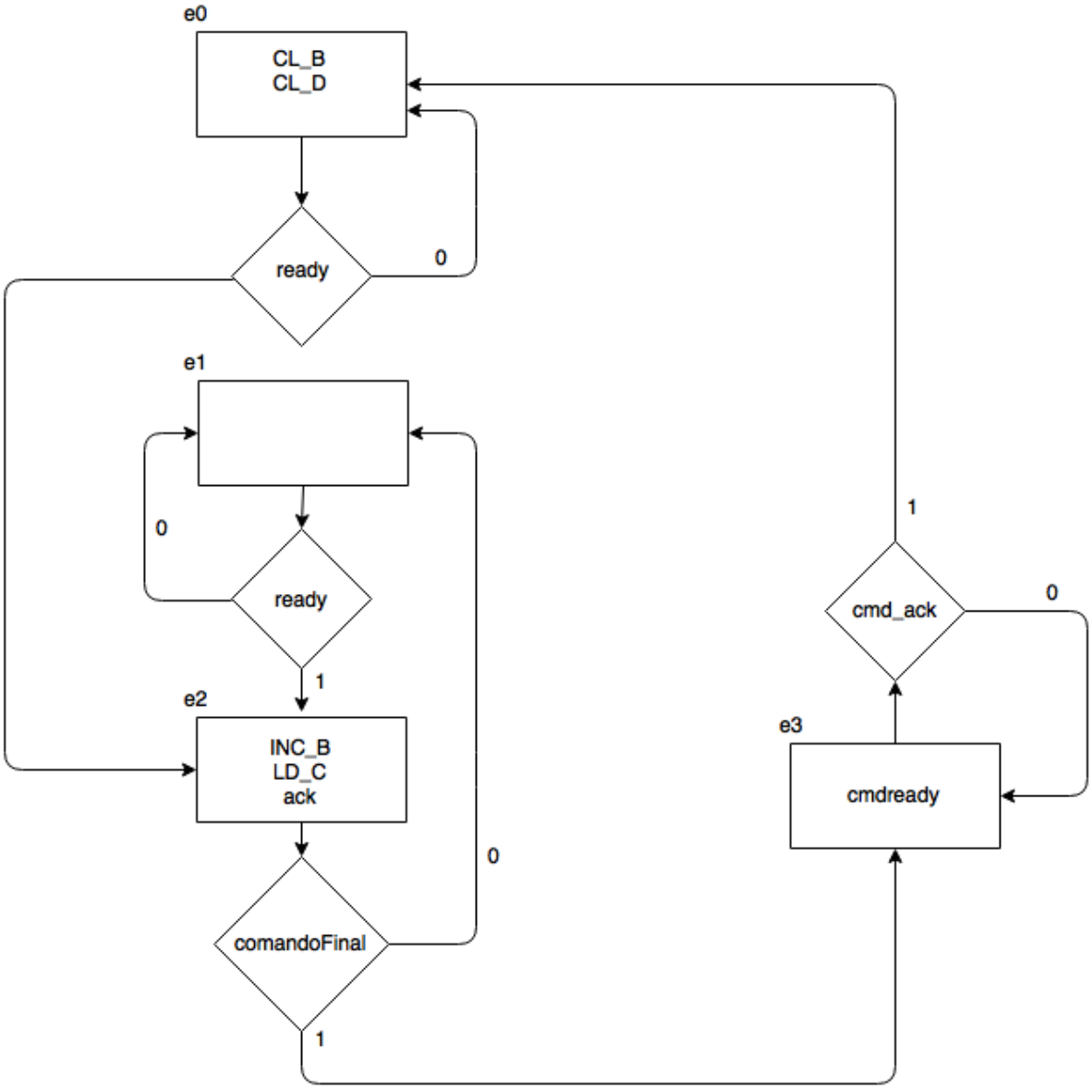
Cada carácter es un número binario de 8 bits que se almacenará en el banco de registro en la posición en la que se especifique.

Hay que tener en cuenta que se ha de saber cuando un comando ha sido completado, y qué comando se está tratando, para ello se usan algunos comparadores.

Por ejemplo, para saber si el comando está completo se compara el carácter recibido con CR. Todos los comandos que se envían acaban con éste valor especial de la tabla ASCII indicando así el final.

Todos los caracteres, a excepción del fin de comando: (CR) son almacenados en el Banco de registros. Al tener la longitud del comando entero que llega, se puede distinguir qué orden se ha enviado y qué respuesta se ha de tomar (activando las salidas de la señal correspondiente).

Unidad de control



Señales implicadas:

- **CL_B:** Realiza un clear a 1 de la variable index (que cuenta las posiciones de los datos que nos van llegando).
- **CL_D:** El dato del banco de registros se resetea a ceros.
- **Ready:** En rx_char, cuando envía un carácter, esta señal se activa. En este módulo, la recibimos y tenemos que tenerla en cuenta para ir cargando los caracteres.
- **INC_B:** Si recibimos un carácter, incrementamos el index.
- **LD_C:** Es la señal de carga en el banco de registros, cargamos el char que nos llega.
- **ack:** Es la señal que activamos y mandamos al rx_char para decirle que hemos procesado
  y almacenado su envío.
- **comandoFinal:** Si recibimos en algún momento un CR, avanzaremos a un nuevo estado.
- **cmdready:** Señal que activamos a 1 para avisar al módulo posterior que hemos procesado
  el comando y hemos activado las señales correspondientes
- **cmd_ack:** Respuesta del módulo anterior

Explicación de la Unidad de Control

El estado e0 borra el "index" y la memoria del registro, se quedará a la espera hasta que reciba una señal de ready, la cual representa que se recibirá un dato. A continuación pasa al estado e2, que incrementa el índice, activa la salida ack y si no es un CR vuelve al estado e1.

El e1 es un estado transitorio cuyo objetivo es diferenciar el primer Byte del resto, sirve para no hacer el reset que se hace en el e0.

Mientras no se reciba un CR, es decir, mientras "comandoFinal" sea 0, seguirá en el bucle.
Cuando ocurra lo contrario se pasará al estado e3, que activará la serial cmdready y el circuito se quedará a la espera de recibir un cmd_ack por parte del otro módulo, mientras esto no ocurra se seguirá enviando un cmdready.

Por último, cuando recibamos una confirmación por parte del otro módulo, volverá al estado e0, donde se quedará a la espera de recibir un mensaje nuevo.

Resultados de las simulaciones

RX_CHAR

Valores de entrada del TestBench:

```
begin
  --clk <= not (clk) after 10 ns;
 wait for 10 ns;
 cl <= '0';
 rx<='0';
 wait for 104.2 us;
  rx<='0';
 wait for 104.2 us;
  rx<='1';
 wait for 104.2 us;
  rx<='1';
 wait for 104.2 us;
  rx<='0';
 wait for 104.2 us;
  rx<='1';
 wait for 104.2 us;
  rx<='0';
  wait for 104.2 us;
  rx<='1';
  wait for 104.2 us;
  rx<='1';
  wait for 104.2 us;
  rx<='1';
  wait for 1 ms;
```

El funcionamiento de este banco de pruebas es el siguiente:

1.- Se inicializa la entrada al dispositivo y se niega el clear.

2.- Se van actualizando los valores de la entrada “rx” teniendo en cuenta el tiempo que transcurre en la lectura de un bit.

3.- La secuencia que forma los diferentes valores de rx es la siguiente: <11010110>

4.- El “1” del final advierte al dispositivo el final de la transmisión. El “0” inicial indica el inicio de la transmisión.

Resultados obtenidos al realizar la simulación:

Como se observa, el sistema se posiciona en la mitad del bit cuando la señal de entrada se pone a “0”. En ese momento comienza a contar bits completos, y a almacenarlos en su registro de desplazamiento hasta completar un byte. Una vez completado el byte, detecta el bit de stop y se pone de nuevo a la espera de órdenes.

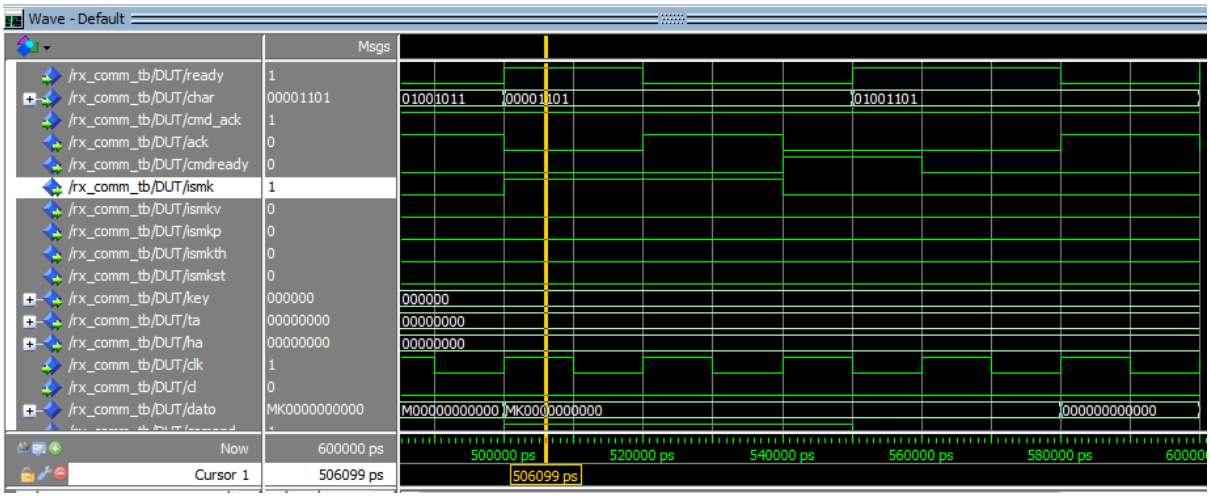
RX_COMM

Para probar éste módulo se han diseñado tres casos de prueba:

```
begin


 wait for 10 ns;
 cl <= '0';
 ready <='1';
 char<=std_logic_vector(to_unsigned(character'pos('M'),8));
 wait on ack; --en cuanto se produce un cambio de 1 a 0 o de 0 a 1 salta de este wait
 ready <='0';
 wait on ack;
 ready <='1';
 char<=std_logic_vector(to_unsigned(character'pos('K'),8));
 wait on ack;
 ready <='0';
 wait on ack;
 ready <='1';
  char<=std_logic_vector(to_unsigned(character'pos(CR),8));
 wait on ack;
 cmd_ack<='1';
 ready <='0';
```

En este caso se analiza el comando MK<CR>. Se pretende comprobar si el receptor de comandos va a recibir e interpretar el comando recibido.



En esta imagen se observa el resultado de la simulación.

La señal ismk se activa a 1 cuando recibe el comando MK. Esto ocurre porque está definido en la señal index que cuando la longitud sea de tamaño 3 se active a 1 la señal ismk.

El siguiente módulo recibirá ese 1 ya que cmdready también se activa.

A continuación se realiza una prueba con un comando algo más complejo:

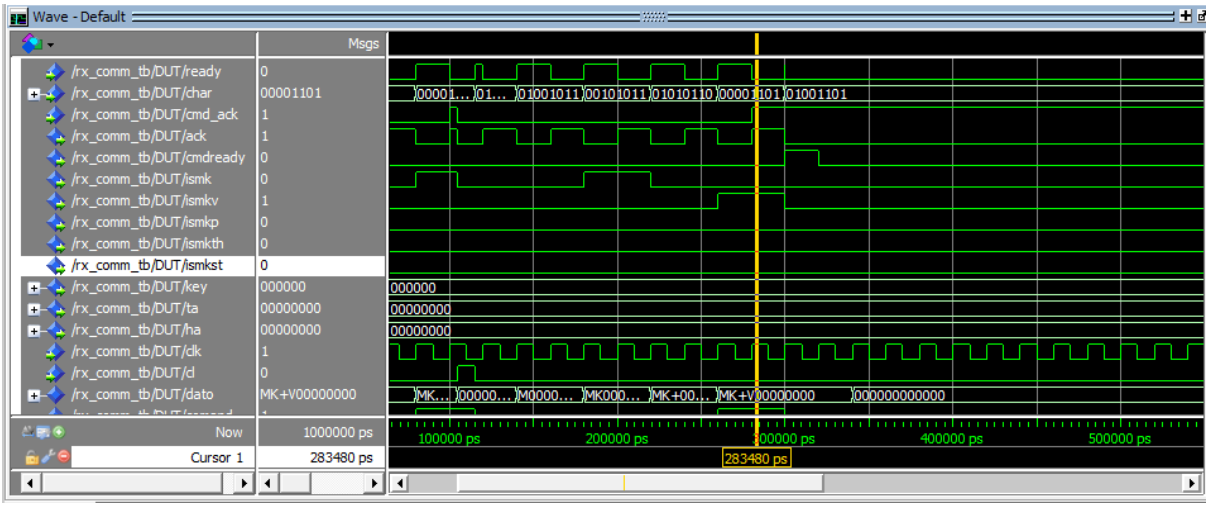
```
wait for 10 ns;
cl <= '0';
ready <='1';
char<=std_logic_vector(to_unsigned(character'pos('M'),8));
wait on ack;
ready <='0';
wait on ack;
ready <='1';
char<=std_logic_vector(to_unsigned(character'pos('K'),8));
wait on ack;
ready <='0';
wait on ack;
ready <='1';
char<=std_logic_vector(to_unsigned(character'pos('+'),8));
wait on ack;
ready <='0';
wait on ack;
ready <='1';
char<=std_logic_vector(to_unsigned(character'pos('V'),8));
wait on ack;
ready <='0';
wait on ack;
ready <='1';
char<=std_logic_vector(to_unsigned(character'pos(CR),8));
wait on ack;
cmd_ack<='1';
ready <='0';
```

En este caso lo que se hace es enviar el comando MK+V<CR>", que se utilizará para conocer la versión del programa.

A diferencia del anterior, éste comando lleva un parámetro.

En la siguiente imagen se muestra el resultado de la prueba, y se pueden observar las señales que se han activado.

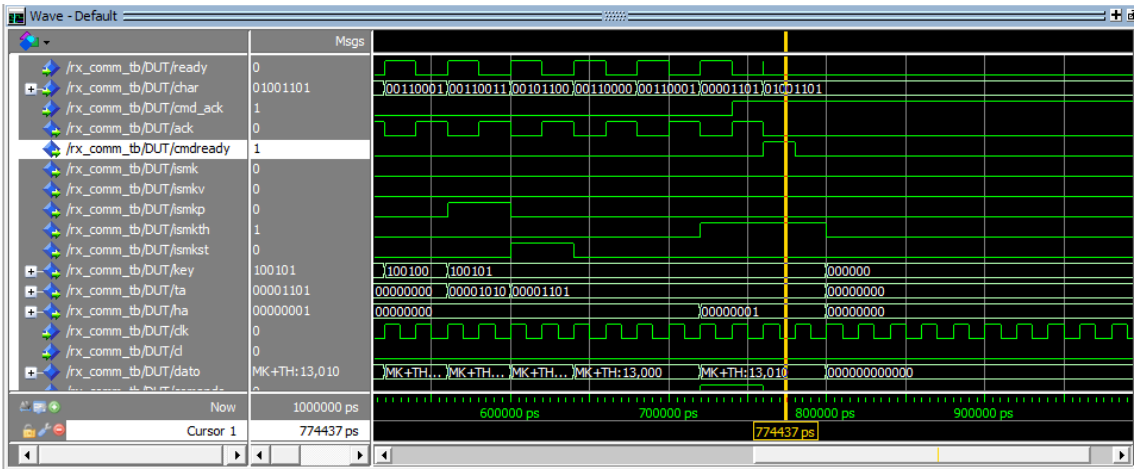
Este es el resultado de la simulación:



Se comprueba que la señal activada es la correcta. En este caso se ha probado MK+V y ha sido la señal ismkv la que ha cambiado del estado 0 al estado 1.
Observamos también que el dato almacenado se corresponde con la secuencia transmitida, y que se activan las señales de ack y cmd_ack.
Se concluye que el resultado de esta prueba también es satisfactorio.

Por último, para garantizar el correcto funcionamiento del RX_COMM, se hace una prueba con un comando aún más complejo que los anteriores: MK+TH 1301<CR>.

El código de la prueba es demasiado largo para incluirlo, y no añade ningún detalle más respecto a las imágenes anteriores, es por ello que se omite en esta prueba y se pasa directamente al análisis del resultado obtenido en ModelSim.



Se analiza el resultado. La señal que se activa es correcta, ismkth pasa de 0 a 1. En este comando hay que fijarse también en “key”, “ta” y “ha”, ya que estas tienen que cambiar con unos valores que se corresponden con los atributos del comando. Se ve como “key” cambia, y “ta” toma el valor de la temperatura ambiente en binario, el número 13 que se le pasa por parámetro se corresponde con el “1101”, y por último, la humedad ambiente es 1, lo que también se refleja en el cambio de la variable “ha”.
Con esta última prueba se da por concluido el test, se han obtenido unos resultados satisfactorios, por lo que se concluye que el diseño y la implementación son correctas.

Pruebas en Altera DE-2

¿Cómo programar la FPGA? Pasos a seguir

Una vez completada la simulación en ModelSim hay que comprobar que el diseño funcione también en una placa física. Para ello se han seguido los siguientes pasos:

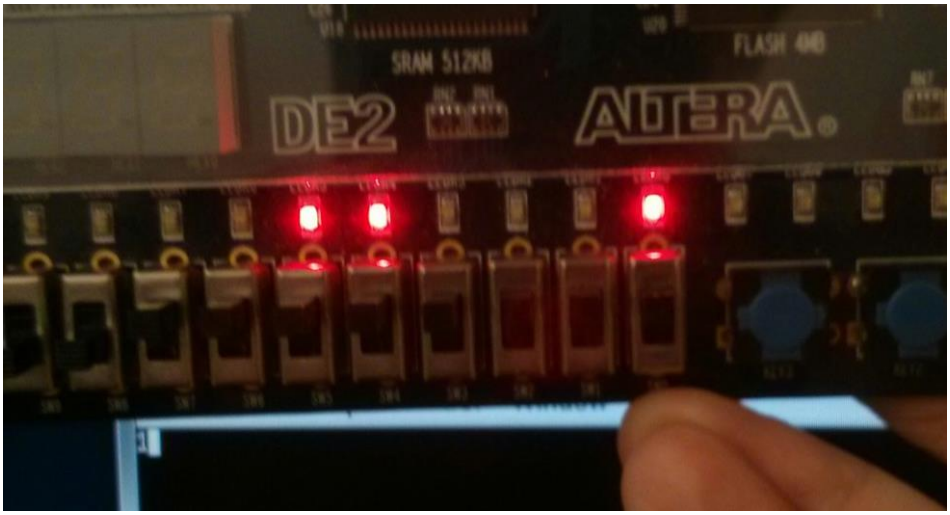
1. Se ha creado un nuevo proyecto en Quartus mediante el “wizard”, al que hemos llamado “rx_char_DE2”. Es importante de cara al correcto funcionamiento que los nombres sean los mismos que la entidad principal para todos los campos.
2. Los pines han sido configurados para que aquellos que no se usen estén como entrada triestado. También hemos tenido que utilizar un fichero llamado “DE2_pin_assignments”, que se nos ha sido facilitado para no recurrir a una asignación manual.
3. Se compila el código. Hay que tener especial cuidado con los nombres, ya que si hay diferencias entre los dos módulos utilizados para las comprobaciones, no funcionará.
4. Una vez que la compilación sea correcta y no haya errores se prueba en la placa. Para ello hay que ir en la barra de manú a Tools→Programmer. Hay que asignar el USB-Blaster y estar en modo, para eso hay que pinchar sobre Hardware Setup. Si es el primer proyecto es probable que no se pueda pasar el diseño a la placa, para eso hay que elegir un archivo *.sof, lo podemos encontrar en la carpeta “output_files” del proyecto.

Resultados

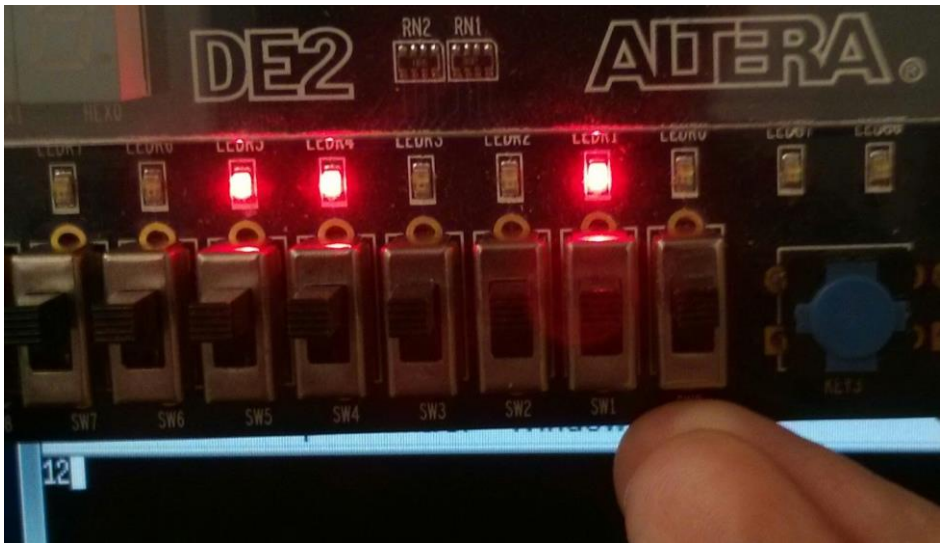
En las siguientes páginas se mostrarán los resultados obtenidos mediante capturas de pantalla o fotografías.

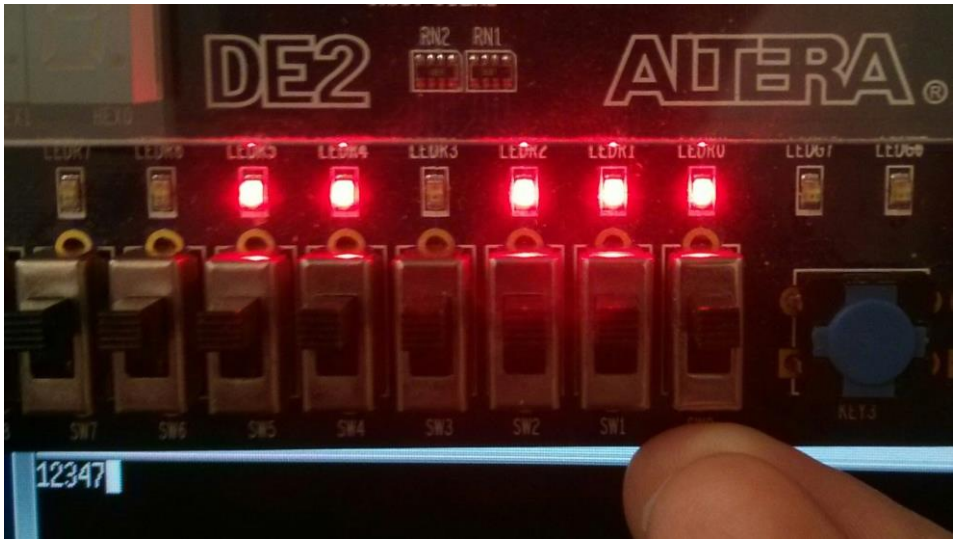
RX_CHAR

Se observa que la implementación en la FPGA ha sido satisfactoria, para ello se han realizado una serie de comprobaciones. Son las siguientes:



En esta imagen se ve cómo al introducir el carácter “1” por Teraterm se muestra su valor binario en la placa. En las siguientes capturas se observa lo mismo con los caracteres “2” y “7”. Durante la introducción de texto se aprecia que en la FPGA se enciende el led de RX, que se encuentra en la parte superior derecha, cerca del conector RS232, indicando que está recibiendo datos a través de la línea de serie.

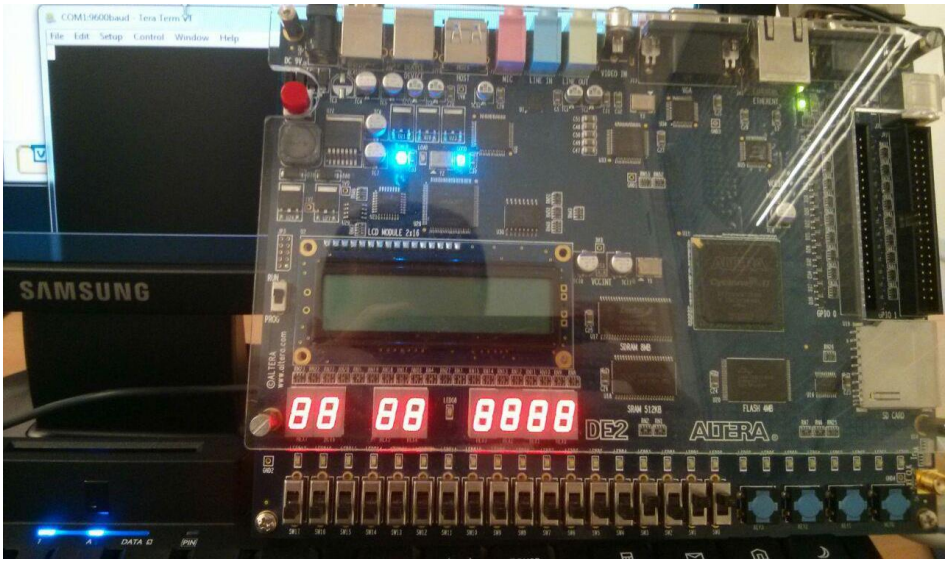




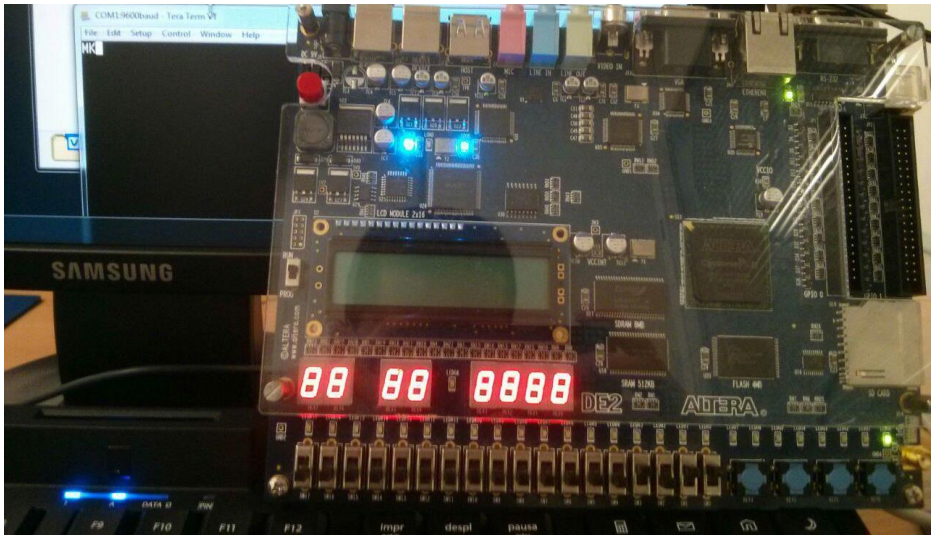
Lo mismo ocurre para otros caracteres no numéricos, se reconocen correctamente y se obtiene su representación binaria a través de los leds de la placa.

RX_COMM

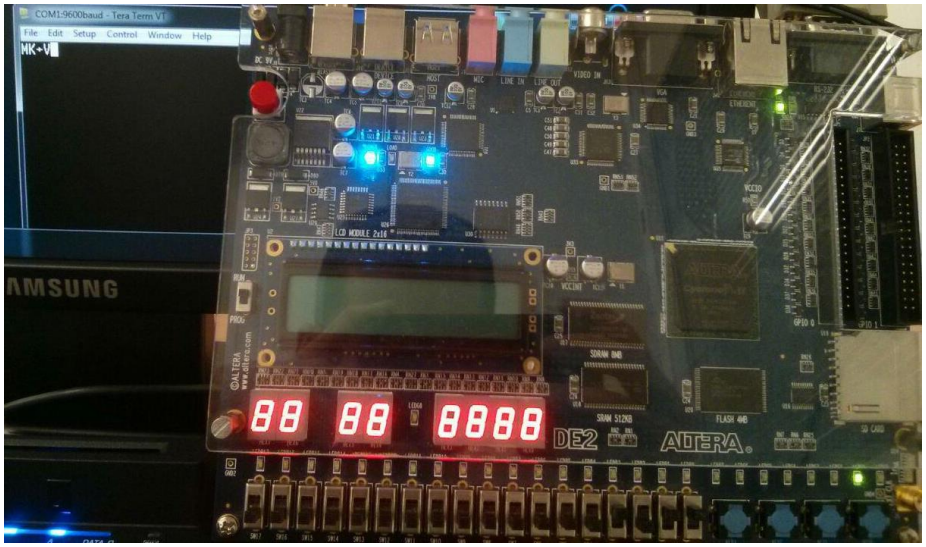
Para probar este módulo se introducirán los comandos que usa la aplicación a través de la consola de Teraterm. Se comprobará la respuesta de la placa a través los leds.

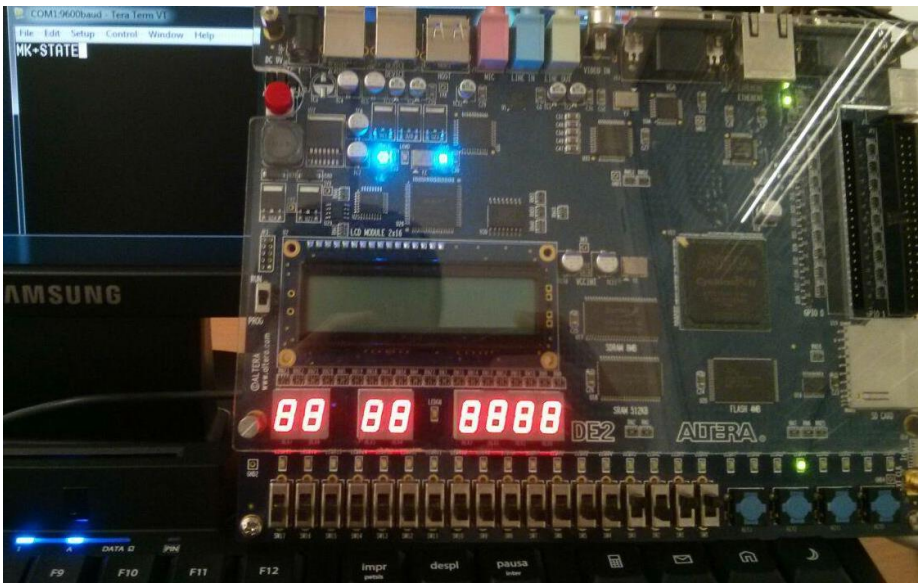


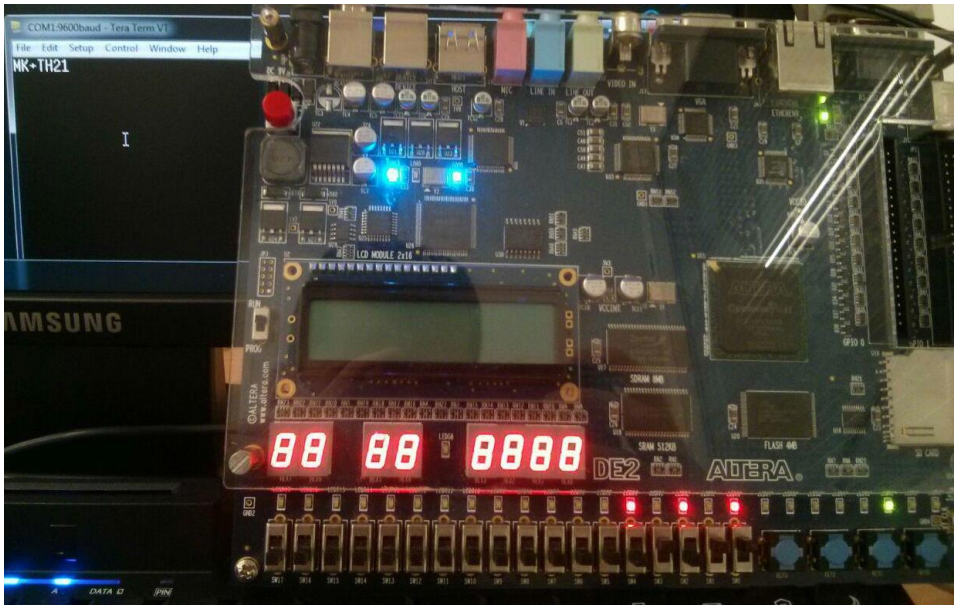
Éste es el estado inicial de la placa después de compilar el programa.



En esta segunda imagen se ha introducido el valor “MK”. La respuesta en placa correspondiente es encender el último led. Lo mismo ocurrirá para las siguientes imágenes que se muestran.





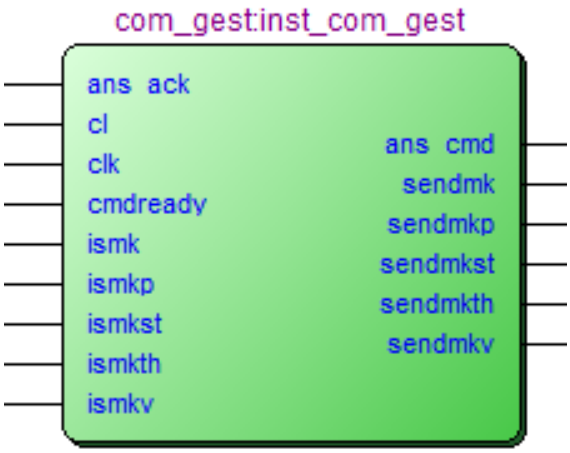


Esta última imagen muestra en los leds verdes el comando que se trata y en rojo muestra en binario el número 21, que es la orden que se le ha enviado por consola.

COMM_GEST

Éste módulo, com_gest, es el eje entorno al que gira todo el sistema digital, ya que su funcionamiento es imprescindible para que todos los módulos que forman la aplicación trabajen conjuntamente y de manera ordenada.

Diseño del módulo

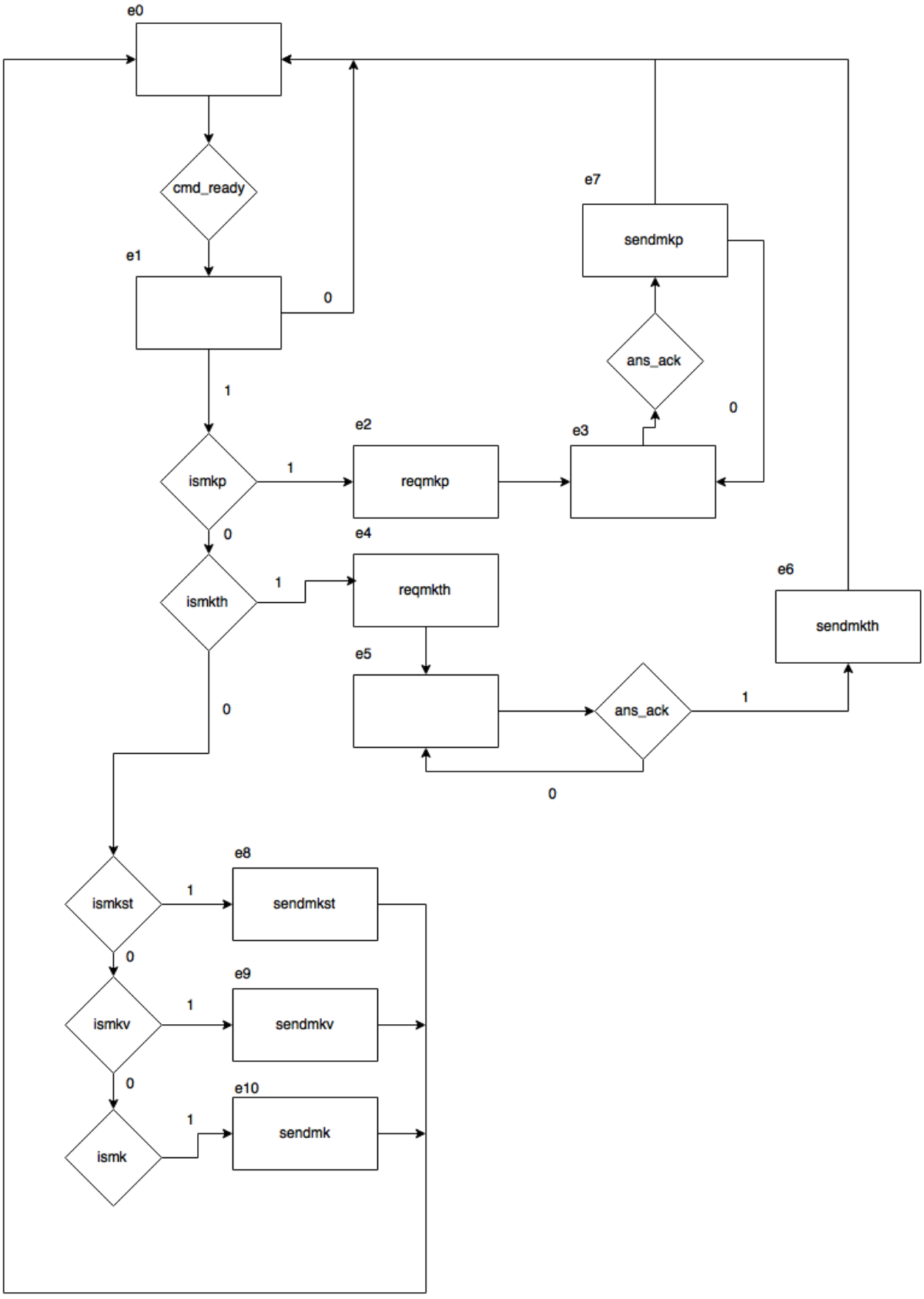


Cómo se ve en la descripción gráfica del módulo, com_gest recibe unas señales, que las leerá cuando ans_ack esté activo. Tiene que reconocer cuál de ellas le ha llegado, y en consonancia tendrá una salida que servirá para solicitar un dato u acción a otro módulo.

Va a enviar y recibir mensajes con el módulo, y enviará mensajes a tx_answer (módulo que ya se nos ha facilitado).

En la siguiente página se muestra la Unidad de Control diseñada:

Unidad de Control y explicación



A continuación se describe el funcionamiento de la Unidad de Control:

El sistema comienza en el estado **e0.** Se queda a la espera de que el rx_command termine de analizar la instrucción que se le pasa bit a bit. Una vez termine su función, enviará la señal de sincronización cmd_ready, que permitirá el paso al siguiente estado.

En el estado **e1** se decidirá el camino a seguir según el tipo de instrucción recibida por rx_command. Son dos las posibles opciones a seguir:

a) Recibir una señal que **no** interactúe con la base de datos (ismkst, ismkv y ismk). En ese caso el com_gest advertirá al emisor del comando a enviar.
b) Recibir una señal que interactúe con la base de datos (ismkp o ismkth). En este caso, el módulo com_gest tendrá que contactar con la base de datos y advertirle del cambio a realizar en sus datos. Una vez actualice los datos el módulo tamagotchi se lo hará saber a com_gest y éste a su vez avisará al emisor de que ya puede enviar el comando.

Los estados **e8**, **e9** y **e10** son los que cumplen con el primer caso. Los valores a enviar son el estado, la versión y la tecla pulsada. Para ello no se necesitará acceder a la base de datos, por lo que el emisor puede realizar su función sin tener que esperar a nadie.

En los estados **e2** y **e4** com_gest activa la señal de handshake con la base de datos, advirtiéndole del tipo de actualización a realizar (valores de estado del tamagotchi o de la temperatura y humedad).

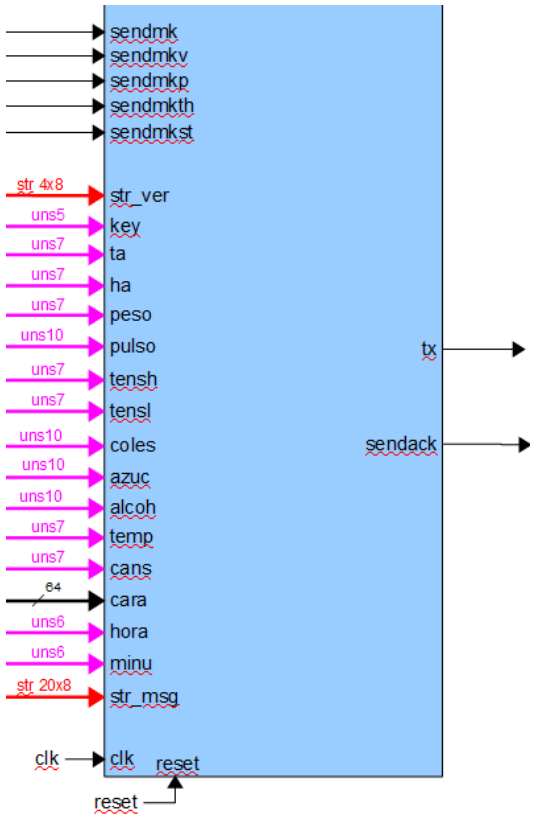
En los estados **e3** y **e5** com_gest se quedará a la espera de que la base de datos actualice los datos y se los entregue al emisor. Cuando realice lo anteriormente mencionado, devolverá al gestor la señal de sincronización ans_akh.

Cuando ya se ha actualizado la base de datos y los datos han sido enviados al emisor, en los estados **e6** y **e7** activamos las señales sendmkp o sendmkth para advertir al emisor que ya dispone de todo lo necesario para emitir el comando completo.

TX_ANSWER

El módulo tx_answer genera la respuesta y la envía por la línea serie. (Por ese orden)

Hay que indicarle cual es la respuesta que se quiere enviar, y hay que proporcionarle la información para que pueda enviar todo correctamente.



Señales implicadas:

Entradas:

sendmk: Petición de transmitir respuesta MK.
sendmkv: Petición de transmitir respuesta V.
sendmkp: Petición de transmitir respuesta P.
sendmkth: Petición de transmitir respuesta TH
sendmkst: Petición de transmitir respuesta STATE
str_ver: Versión del sistema
key: Código de la tecla pulsada
ta: Temperatura ambiente
ha: Humedad ambiente
peso: Peso del Tamagochi
pulso: Pulso del Tamagochi
tensh: Tensión alta del Tamagochi
tensl: Tensión baja del Tamagochi
coles: Nivel de colesterol del Tamagochi
azuc: Nivel de azúcar del Tamagochi
alcoh: Nivel de alcohol del Tamagochi
temp: Temperatura del Tamagochi
cans: Nivel de cansancio del Tamagochi
cara: Cara del Tamagochi
hora: Horas de la edad del Tamagochi
minu: Minutos de la edad del Tamagochi
str_msg: Mensaje en pantalla

Salidas:

tx: Línea serie de salida
sendack: Indica que se ha terminado de transmitir el comando. Se desactiva cuando las señales sendmkxx están desactivadas.

Tx_Answer está formado a su vez por dos submódulos:

tx_char

Es el encargado de enviar un carácter por la línea serie tx de acuerdo a las especificaciones del puerto RS232, es decir, enviar un bit de start y un bit de stop por cada carácter. La velocidad de envío ha de ser 9600 baudios.

Entradas:

char: Dato de 8 bits a transmitir.
send: Indica que debe comenzar la transmisión.

Salidas:

tx: Línea serie de salida (ver figura inferior).
ack: Indica que ha terminado la transmisión. Se desactiva cuando send está a '0'.

Parámetros:

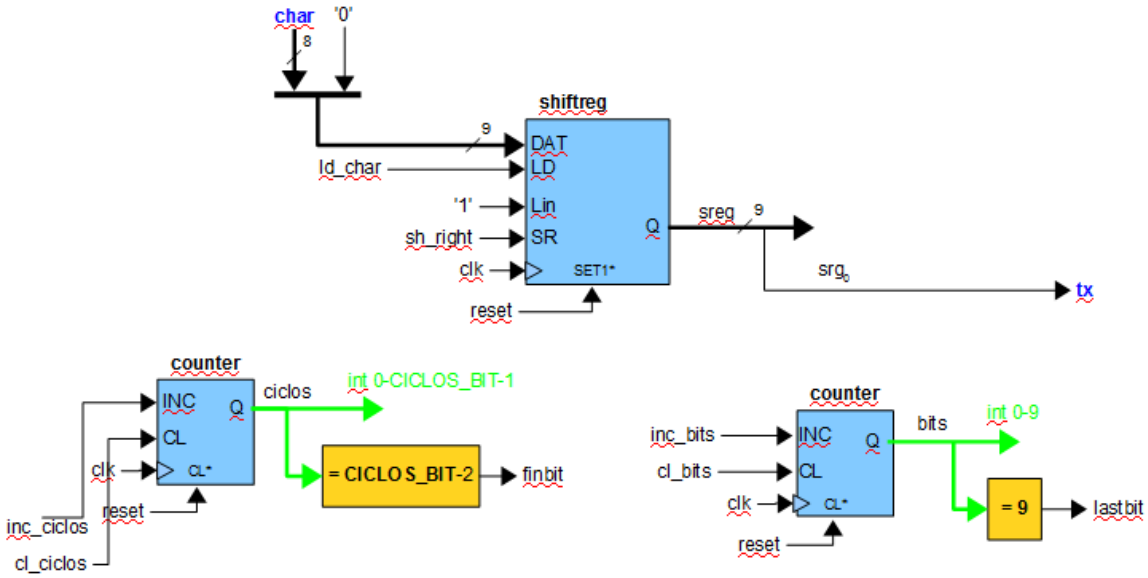
BAUD_RATE: Velocidad de transmisión en baudios
CLK_FREQ: Frecuencia de reloj en ciclos por segundo

Diseño del módulo

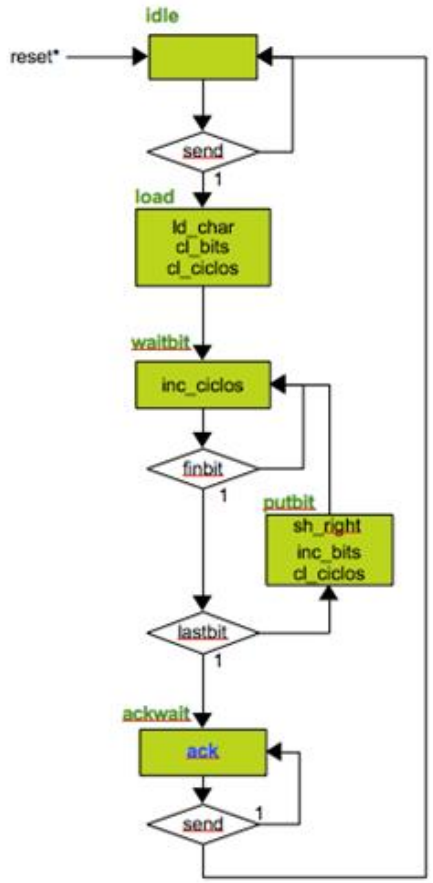
Un registro de desplazamiento de 9 bits (shiftreg) en el que se guarda el carácter a enviar junto con el bit start. La línea serie tx está conectada al bit de menos peso del registro (srg_0)

Un contador para controlar el número de ciclos durante los que hay que mantener cada bit en la línea tx

Un segundo contador para controlar el número de bits que se envían



Unidad de control



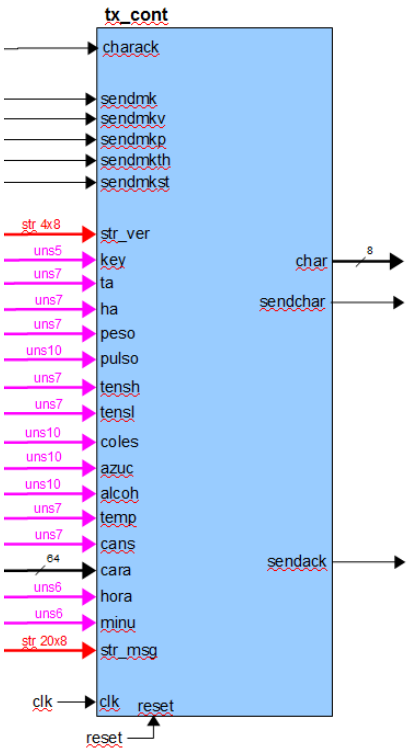
En el estado inicial, idle, permanece a la espera de recibir una orden de envío.

Tras recibir la orden, realiza las inicializaciones pertinentes en los elementos de la UP (estado load) y ejecuta un bucle en el que por cada bit, permanece 5208 ciclos en el estado waitbit y luego pasa al bit siguiente mediante un desplazamiento (estado putbit). La condición de salida del bucle la proporciona el contador que indica que ya se ha enviado el último bit (bit stop).

Por último, activa la señal ack correspondiente al protocolo en el estado ackwait y se queda en él mientras la señal de comienzo send esté activada.

tx_cont

Es el encargado de controlar el envío de la respuesta completa con todos sus caracteres. Para ello, se valdrá del módulo tx_char, al que proporcionará la información necesaria para enviar cada uno de los caracteres de la respuesta por la línea serie tx.



Salidas:

char: Dato de 8 bits para ser enviado por tx_char
sendchar: Indica a tx_char que comience a transmitir el carácter que está en char.
sendack: Indica que se ha terminado de transmitir el comando. Se desactiva cuando las señales sendmkxx están desactivadas.

Diseño del módulo

Tabla de traducción: el módulo tx_cont tiene que formar la respuesta a enviar de acuerdo a la petición que recibe (sendmkxx) y para ello cuenta con una tabla de traducción. Dicha tabla tiene como entradas tanto la petición que se hace como el resto de información que se proporciona en las entradas, y de ese modo devuelve un string que, como máximo, tendrá 84 bytes. Se pueden ver los detalles de las respuestas en la siguiente página.

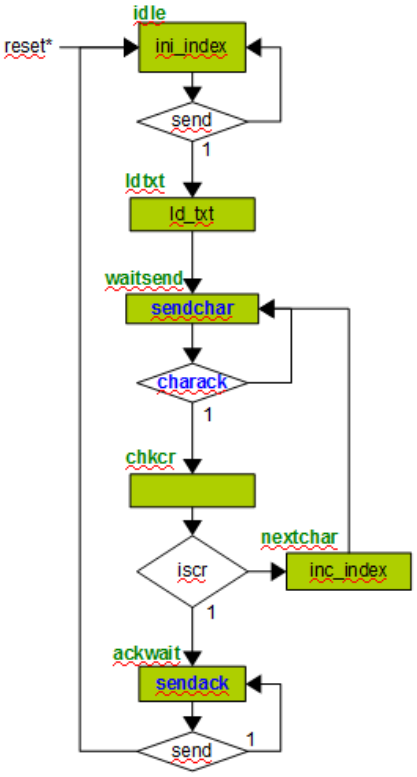
La respuesta formada mediante la tabla anterior se registra en un conjunto de registros para que permanezca estable durante el envío.

Son necesarios un contador y un multiplexor para ir seleccionando uno a uno los caracteres de la respuesta que hay que ir pasando al módulo tx_char.

Por último, utiliza un comparador para detectar cuándo finaliza la respuesta: carácter CR.

Unidad de control

Se trata de un algoritmo muy simple.



Permanece en el estado inicial, idle, hasta recibir alguna petición de envío: alguna de las señales sendmkxx activada. En dicho estado inicializa el contador de caracteres.

Cuando se activa alguna de las peticiones, la tabla de traducción proporciona la respuesta a enviar, y se guarda dicha cadena en el conjunto de registros (estado *ldtxt*).

A continuación, el algoritmo se mete en un bucle en el que en cada iteración proporciona un carácter distinto de la respuesta en la salida char, se solicita al módulo tx_char que la envíe y se queda a la espera de su respuesta (estado *waitsend*).

Tras el envío del carácter (*charack*) se chequea si se trata del último carácter (estado *chkcr*) para decidir si se continúa en el bucle o no. Si se permanece en el bucle, hay que incrementar el contador (estado *nextchar*) para que la selección sea correcta.

Finalmente, al salir del bucle, activa la salida *sendack* en el estado *waitack* y permanece en el mismo hasta que todas las señales sendmkxx estén desactivadas.

Resultado de las simulaciones

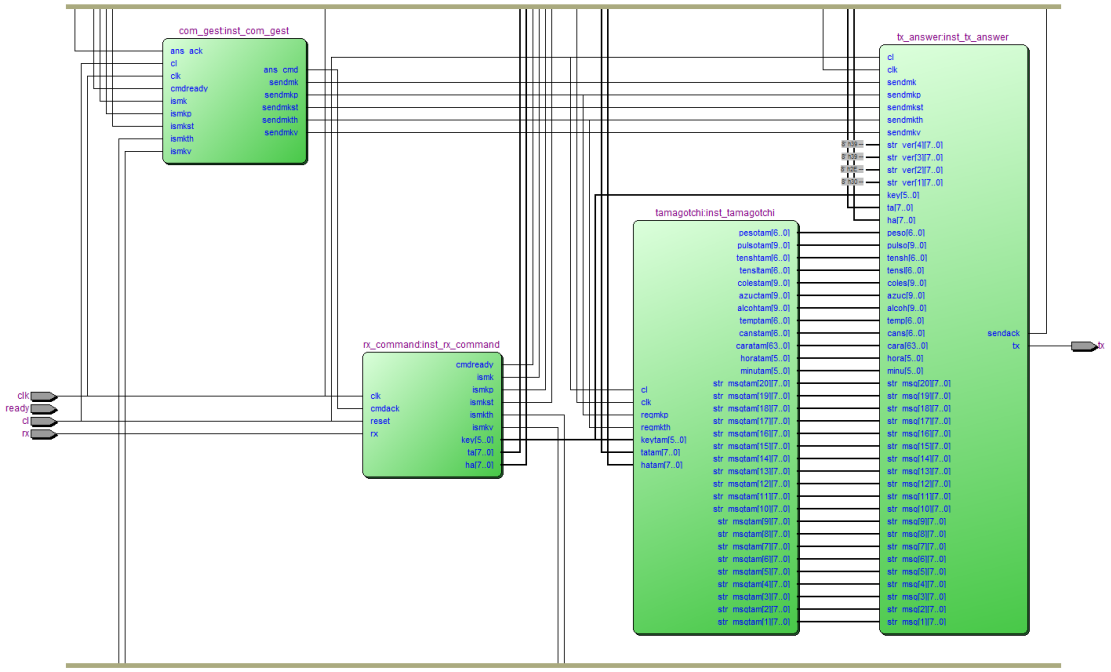
Se ha probado el módulo con el testbench, y al asignar a los switches unos comandos en respuesta, la placa escribe los mensajes en la consola.

No se ha recogido material gráfico, pero se ha comprobado que el módulo dado funciona correctamente.

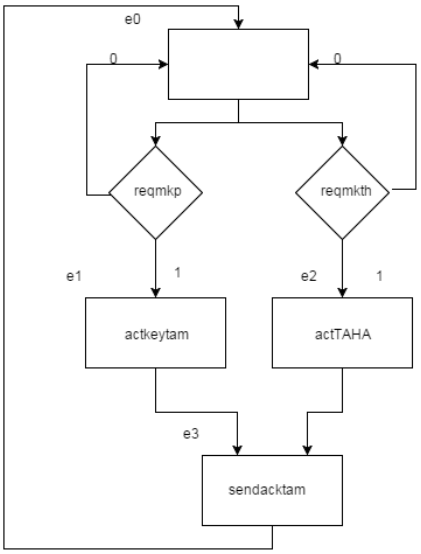
Tamagotchi

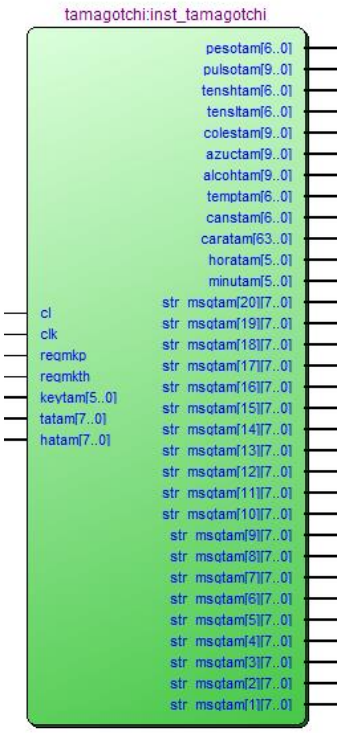
Tamagotchi es el módulo encargado de realizar los cambios necesarios en el sistema. Se trata de una base de datos inicializados a unos valores concretos y que deberán modificarse en función de las teclas que pulse el usuario.

Para ello interactuará con el módulo com_gest, del que recibirá las señales reqmkth y reqmkp. También recibirá de rx_command la tecla pulsada, la temperatura y la humedad.



Éste módulo utiliza una una unidad de control muy sencilla, la complejidad viene en la unidad de proceso para realizar todas las operaciones.

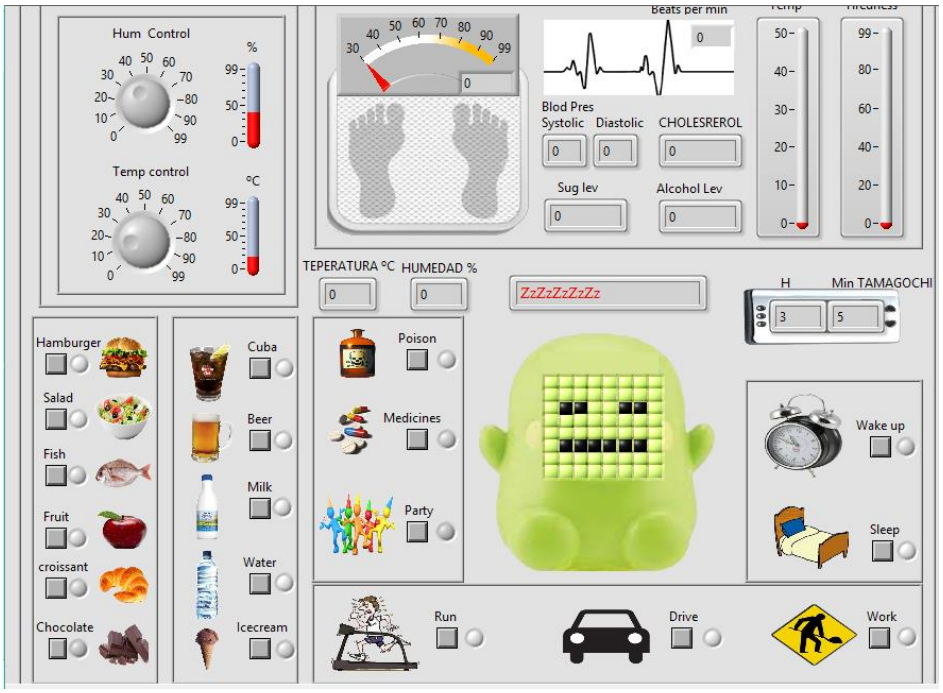




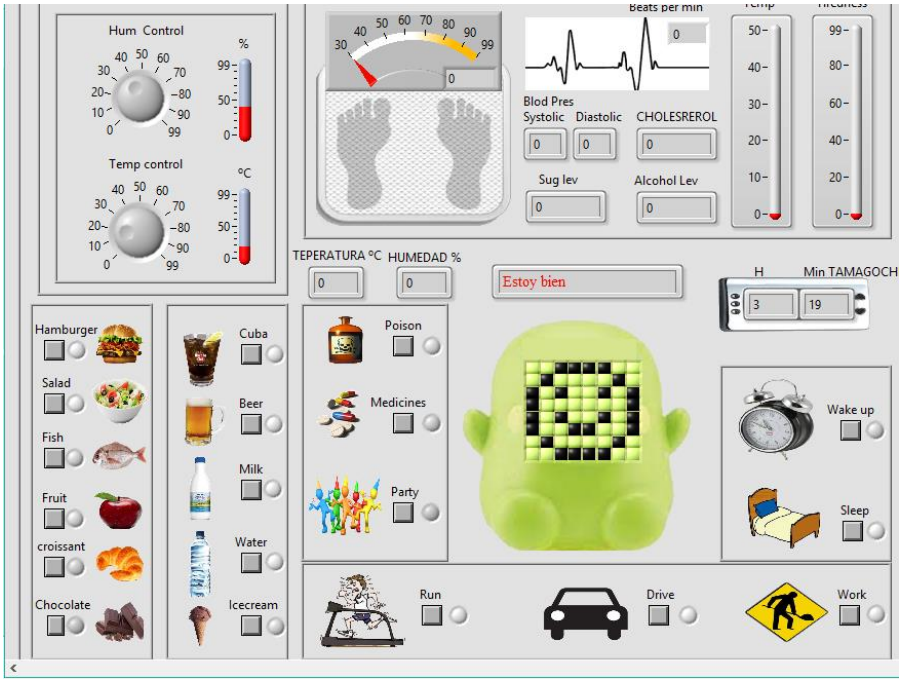
Una vez de implementarlo ya se puede probar todo con el software de Mirakonta, tan sólo hay que compilarlo y grabarlo en la placa como se ha explicado anteriormente.

Resultados

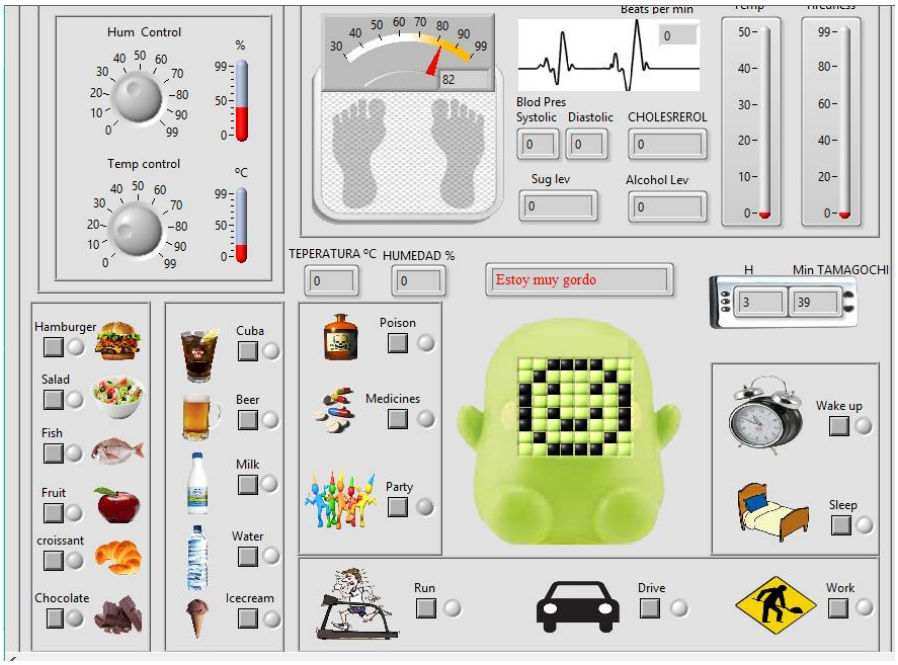
Tras la implementación de todos los módulos anteriores se ha probado el funcionamiento final de la aplicación. Aún quedan mejoras que se podrían haber implementado, de momento el resultado es el que exponemos a continuación:



En esta imagen, tras iniciar la aplicación y conectar se le ha ordenado dormir.



En esta segunda captura se le ha mandado despertar. Se ve que ha cambiado el tiempo y que también tiene una cara diferente. También ha cambiado el mensaje que muestra.



Por último en esta tercera fotografía se le ha dado una hamburguesa. Se observa un cambio en el peso y un nuevo mensaje, esta vez sobre su estado físico.

Manual de usuario

Aplicación Tamagotchi. Se trata de un juego en el cual el usuario tendrá que cuidar de una mascota virtual. Cada acción del usuario repercutirá en el estado de esta mascota. Podrá alimentarla, mandarla a hacer actividades, controlar sus constantes vitales…

Entre la mascota y el usuario habrá una comunicación, es decir, un intercambio de mensajes bidireccional. Para esto se emplea un ordenador con un software prediseñado y una placa FPGA.

Requisitos previos

- Ordenador con un puerto de serie RS232 o un adaptador USB-RS232.
- SO: Windows (La aplicación de Mirakonta es un ejecutable .exe)
- Labview Runtime
- Altera DE-2 FPGA
- Quartus II (En el caso de que el programa de la placa sea volátil, será necesario Quartus para programarlo)

Instalación y configuración

Hay que instalar los runtimes de National Instruments para poder utilizar el programa de Mirakonta.

- NI-VISA Run-Time Engine 5.4
- LabVIEW Run-Time Engine 2012

La instalación es muy sencilla, basta con descargar ambos archivos y seguir los pasos que aparecen por pantalla.

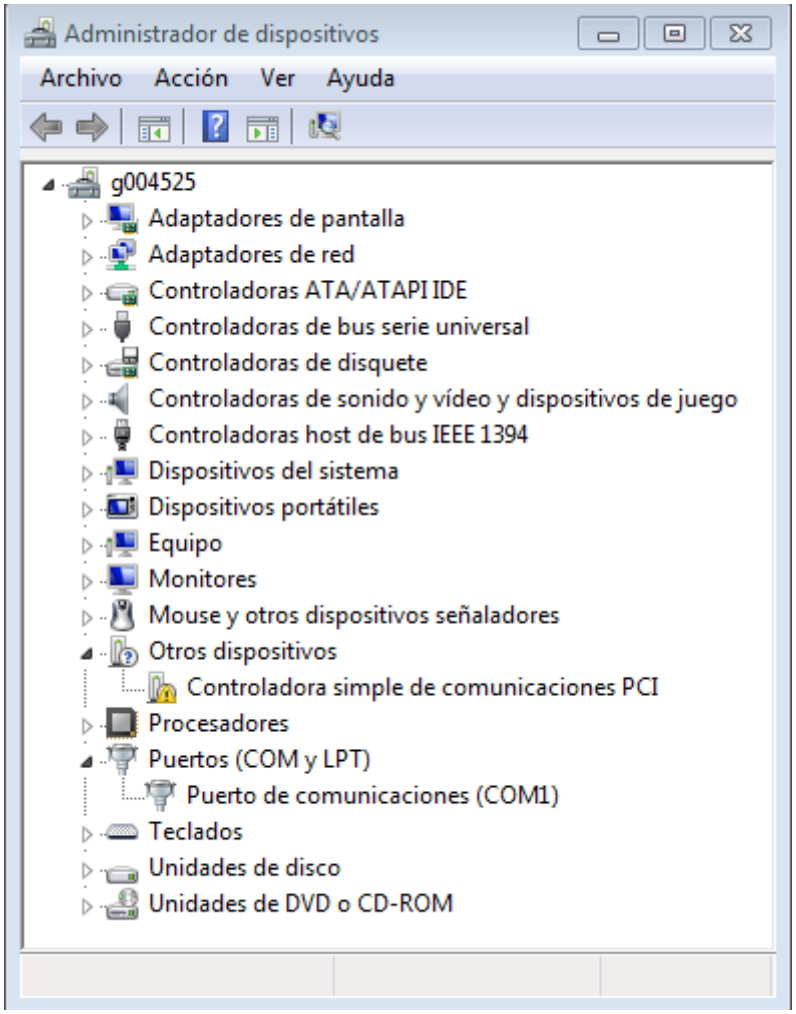
Debe de estar instalado también el software Quartus II, para poder programar la placa en caso de que sea necesario. Para programar la placa se siguen los pasos, revise la referencia:

¿CÓMO PROGRAMAR LA FPGA? PASOS A SEGUIR

Primeros pasos

Para utilizar la aplicación, una vez cumplimentados los pasos anteriores, basta con lanzar el ejecutable Tamagotchi.exe. La FPGA va a necesitar una comunicación a través del puerto serial RS-232, para trabajar con la aplicación de Mirakonta es necesario saber qué puerto se está utilizando. Se hace de la siguiente forma:

1. Entrar en el Administrador de dispositivos.
2. Buscar el apartado “Puertos (COM y LPT)



En este caso el puerto correcto es COM1.

3. En la aplicación Tamagotchi hay que seleccionar el puerto correspondiente al que está conectada la placa.
4. Por último hay que pulsar el botón de start y empezará la ejecución de la aplicación. A partir de ahora se podrá interactuar con el tamagotchi a través del resto de botones.

Funciones básicas

Existen 23 teclas a través de las que puede interactuar el usuario repartidas en tres grandes grupos

Actividades		Control Ambiente	Administración
Hamburger	Icecream		
Salad	Poison		
Fish	Medicines		
Fruit	Party		
Croissant	Run	Temp Control	MK
Chocolate	Drive	Hum Control	MK+V
Cuba	Work		
Beer	Sleep		
Milk	Wake up		
Water			

Actividades: Este primer grupo modifica los valores de la mascota. Por ejemplo, la ingesta de hamburguesas hará que tanto el colesterol como el peso del personaje aumente.

Control Ambiente: En esta versión del software se permite cambiar los valores de la temperatura y humedad, sin embargo no afecta al tamagotchi.

Administración: El MK permite saber que existe una comunicación, y el MK+V hace que el tamagotchi implementado en la FPGA devuelva la versión actual del software.

Solución de problemas

Algunos de los problemas más frecuentes son los que se exponen a continuación.

No se encuentra el dispositivo

Asegúrese de que el driver de la placa Altera DE-2 está instalado. En caso de no estarlo se puede encontrar en la carpeta de Altera: "C:\altera\13.0\quartus\drivers". Actualice el controlador a través del Administrador de dispositivos.

La aplicación no establece conexión

A veces al darle al botón de start en el software de Mirakonta no funciona a la primera. Hay que darle al botón de stop y volver a darle de nuevo a start hasta que funcione. Si el problema persiste, revisar que los puertos de comunicación sean los correctos.

Conclusiones

Desde el grupo hay una satisfacción con el trabajo realizado. Se ha conseguido diseñar una aplicación desde cero capaz de cumplir con el protocolo de comunicación que establece la aplicación que ha sido dada.

Esta experiencia ha mostrado cómo es posible diseñar y aplicar la metodología del aprendizaje basado en proyectos. Durante estos meses ha existido un apoyo mutuo para llevar adelante el proyecto con éxito.

Además las fechas límites de entrega han favorecido al cumplimiento de los objetivos durante el curso, ya que ha logrado que haya un esfuerzo para seguir en la evaluación continua.

Errores

No se ha logrado actualizar correctamente los valores de algunas variables. Existe algún tipo de problema con las operaciones, se ha probado a introducir los valores en binario, hacer casting a decimal y el problema persiste. En cambio, si introducimos un valor fijo y constante, este se muestra correctamente. No se ha conseguido solventar este problema tras muchos intentos.

Posibles mejoras

- **Arreglar los problemas.** Solucionar los fallos de forma que la modificación y visualización de los valores se hagan de forma correcta.
- **Más interactividad.** Que las elecciones del usuario tengan más efectos en el tamagotchi.
- **Tiempo determinante.** Que el tiempo que transcurre durante el juego pueda afectar al estado de la mascota. Por ejemplo, que según pasan las horas tenga hambre o sueño.
- **Fin de partida.** Que cuando algunas de las constantes lleguen al máximo se acabe la partida. Por ejemplo en el caso de que no tenga comida o en el que no duerma lo suficiente.

- **Tiempo simulado.** El tiempo podría avanzar de una forma más rápida para que sea más realista.
- **Caras nuevas.** Añadir nuevas caras acordes con los estados de la mascota.