Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ingeniería Eléctrica
IEE2713 - Sistemas Digitales 2021-2

Taller 13: Comunicación UART con Basys 3

23 de noviembre de 2021

Profesor: ÁNGEL ABUSLEME
Ayudantes: MIGUEL ANTONIO ESPINOZA (miguel.espinoza@uc.cl) FELIPE RUBIO (fnrubio@uc.cl) JAVIER ESTEBAN VILLEGAS (jvillegu@uc.cl)

Introducción

Este taller está enfocado en el uso del puerto USB de la Basys 3 como método para transmitir caracteres mediante el protocolo de comunicación UART (Universal Asynchronous Receiver and Transmitter) hacia o desde un computador. Utilizaremos los siguientes *inputs* y *outputs* de la FPGA: puerto USB y botones. Para identificar cuales son sus señales, recuerde revisar el Basys3-Master.xdc disponible en los archivos del curso y/o archivos de talleres anteriores.

Este enunciado se ve muy extenso, sin embargo, gran parte de él corresponde a una explicación detallada de lo que deberán realizar para que tengan una mejor comprensión del taller.

Objetivos

- Implementar una aplicación real para comprender lo importante del uso de FPGA y sus alcances.
- Implemenatar el protocolo de comunicación UART en la Basys 3 con una máquina de estados.
- Diseñar circuitos con *shift registers* para recibir/transmitir información de forma asíncrona.

1. Comunicación UART

A lo largo del curso hemos aprendido a usar varios elementos de la Basys 3. Sin embargo, nos falta uno fundamental: el puerto serial y la conexión USB. Esta conexión nos permite conectarnos con todo tipo de dispositivos, incluyendo un computador, por medio de un protocolo de comunicación llamado UART.

UART es un sistema de comunicación serial, lo que quiere decir que se transmite y recibe un bit a la vez. Esto, como alternativa más eficiente en términos de costo y número de cables que requiere la comunicación paralela. Como los computadores trabajan en *bytes* (conjuntos de 8 bits) o múltiplos de estos, cada transmisión se enfoca en transmitir y recibir paquetes de *bytes*. Por lo tanto, un puerto serial contiene un chip llamado UART enfocado en convertir estos bytes a un tren de *bits* para luego enviarlos de forma continua.

Para transmitir un *byte*, UART primero envía un START BIT, seguido de la información que queremos transmitir (5, 6, 7 u 8 *bits*), para luego terminar con un STOP BIT. Esta secuencia se repite para cada *byte* enviado (ver figura 1).

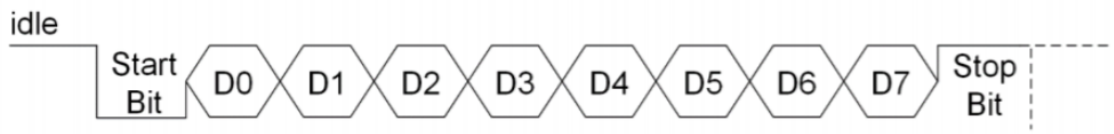


Figura 1: Diagrama de tiempo de 8 bits usando UART.

Un factor importante de esta comunicación es que no requiere de una señal de *clock* (de ahí a que sea asíncrono). En cambio, UART utiliza un *baud rate* (numeros de bit por segundo), número que debe ser compartido tanto por el transmisor como el receptor. Valores típicos son 2400, 9600, 19200 o incluso 115200 *bits* por segundo. Para esta tarea usaremos un *baud rate* de 9600. El unico requisito para el *clock*, es que debe ser bastante más rápido que el *baud rate* para alcanzar a detectar todos los *bits*. Generalmente se usan frecuencias suficientes para detectar (en cada flanco de subida) hasta 16 veces cada bit enviado.

El START BIT siempre es 0. Este es seguido por el LSB (D0) y el ultimo es el MSB (D7). El STOP BIT siempre es 1. La duración del STOP BIT puede variar entre 1, 1.5 o 2 duraciones de bit (esto también debe acordarse previamente entre ambos terminales). Por lo tanto, la sincronización se produce con el uso de estos bits y el *baud rate*, evitandonos así el uso de un cable adicional para la señal de *clock*. Opcionalmente, es posible anadir un *bit* de paridad para poder realizar un pequeño error de chequeo. Es posible elegir si este *bit* sera un*bit* de paridad impar, par u otras configuraciones posibles, o derechamente no utilizarlo.

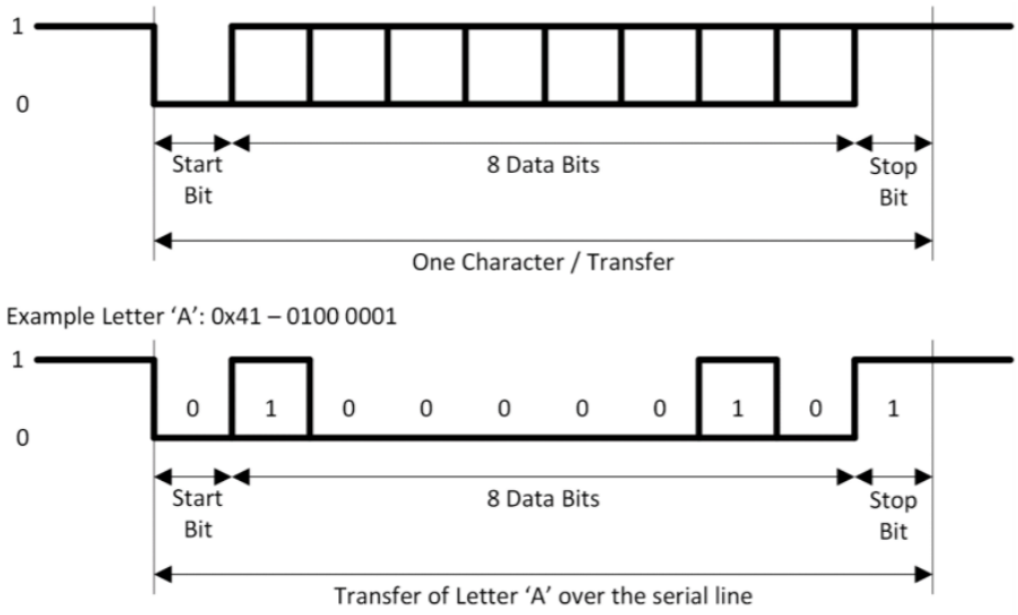


Figura 2: Transmisión de la letra 'A' por UART.

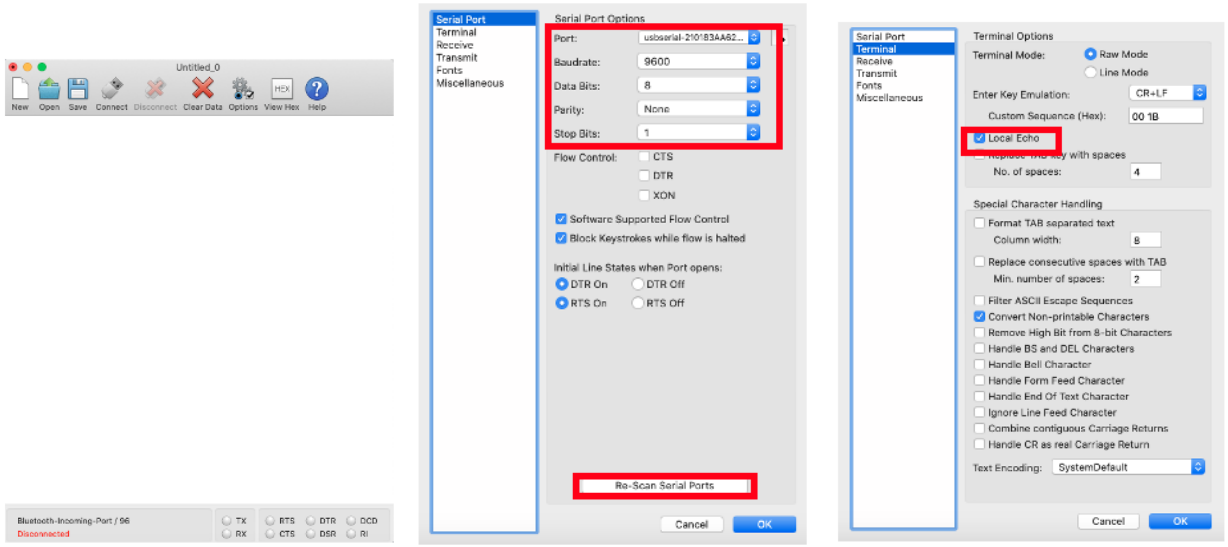
Para transmitir el *byte*, se realiza generalmente en codificación ASCII (8 bits). Por ejemplo, si queremos mandar una letra 'A', entonces el puerto serial transmitira un 01000001 ($0x41_{hex}$). Recordando que UART transmite desde LSB a MSB y usando un STOP BIT, la transmisión total sería **LSB (0 1 0 0 0 0 0 1 0 1) MSB**, en donde el primer 0 corresponde al START BIT y el último 1 al STOP BIT (ver figura 2). Si esta transmisión fue configurada a un *baud rate* de 9600, entonces como se necesitaron 10 *bits* para transmitir un carácter, logramos una tasa de **960 *bytes* por segundo**. Esta explicación esta basada en el siguiente link.

1.1. Terminal UART en computador

Antes de enfocarnos en la Basys, necesitamos un terminal para visualizar lo que transmitimos y recibimos por UART desde el computador. Para esto hay varias alternativas: RealTerm, CoolTerm, ZTerm, PuTTY. Pueden usar la que más les acomode pero es recomendable que utilicen CoolTerm ya que tiene una interfaz fácil de utilizar. Lo pueden descargar acá. (no se preocupen, es mucho más fácil y liviano que Vivado). Una vez instalado y abierto verán algo como en la figura 3a).

La configuración es bastante sencilla, solo deben seguir los siguientes pasos.

1) Abrir el menu **Options** y seleccionar un *baud rate* de 9600, 8 bits de datos, 1 bit de stop y 0 bits de paridad (ver figura 3b). Para detectar el puerto al que esta conectado la Basys deben hacer click en *Re-Scan Serial Ports* lo que detectará el puerto automáticamente (para que pueda detectar la Basys, esta debe estar encendida y conectada).



(a) Inicio. (b) Serial Port. (c) Terminal.

Figura 3: Interfaz de Coolterm

2) Diríjanse a **Terminal** y seleccione la casilla *LocalEcho*. Esto le permitira ver en pantalla lo que ha enviado (ver figura 3c).

2. Basys 3

En esta tarea describirán un circuito que actúe como transmisor (Tx) y receptor (Rx) UART. La idea central consiste en lo siguiente:

- En primer lugar, se recibirá una secuencia de caracteres ASCII enviados desde el computador, los cuales deben ser almacenados en RAM en direcciones consecutivas.
- Luego, deberán enviar esta misma secuencia pero en orden inverso. El envío se inicia al apretar algún botón de la Basys.

  Tip: Enviar el mensaje inverso es bastante simple si se piensa en términos de ir leyendo las direcciones de la RAM en sentido inverso al cuál se escribió.

2.1. RAM

Los caracteres ASCII son de 8 *bits*, por lo que su memoria RAM debe tener este ancho. Usen una memoria con 16 direcciones para que les sea más rápido sintetizar. Cuando su código funcione, pueden aumentar este valor para poder guardar mensajes más largos.

Tip: Puede ser que usar una *Distributed Dual-Port* RAM o una *Distributed Simple Dual-Port* RAM les facilite el trabajo.

2.2. Transmisor

El primer paso que deben hacer es un generador de *baud rate*. Como estamos usando 9600 bits por segundo, deben crear un contador que genere una señal BAUD_EN en la cual se vuelve '1' durante un ciclo cada vez que se complete un intervalo de tiempo correspondiente a un *bit*. Concretamente, se vuelve '1' cuando se llega a la cuenta objetivo y luego vuelve inmediatamente a '0', estando activa solo durante un ciclo de *clock*. A este funcionamiento se le denomina *one-shot*, en donde a partir de cierta condición se produce como respuesta un pulso de tamaño igual a un ciclo.

- Para una frecuencia de 100MHz, el período debe ser dividido por 10416 (equivalente a contar hasta 10415).

Este contador nos permite que la transmisión esté sincronizada con el reloj (o en realidad con el contador). Luego, el módulo transmisor y su diagrama de estados los pueden ver en la figura 4 y su diagrama de estados en la figura 5.

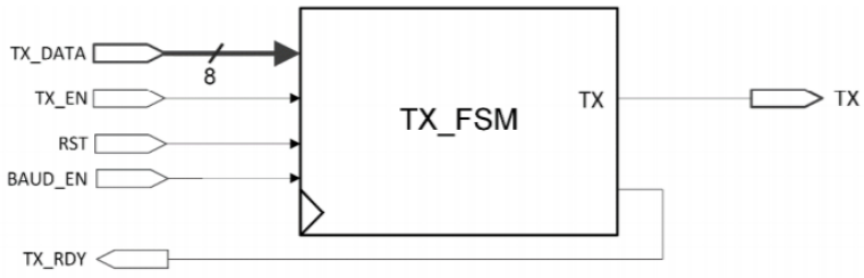


Figura 4: Bloque Tx UART.

Como podrán ver, el diagrama de estados no es muy complejo. En el estado START se genera el START BIT, en el estado BIT (en el cual se mantiene durante 8 períodos de *bit*) se generan los bits del mensaje y finalmente en el estado STOP se genera el STOP BIT.

IMPORTANTE: Los cambios de estado son generados solo en el ciclo de *clock* en el que BAUD_EN es '1'. Esto nos asegura que nos mantengamos en un mismo estado durante un tiempo de bit completo. Por esta razón es que BAUD_EN es '1' solo por un período de *clock*, y al volver a comenzar la cuenta vuelve a ser '0'.

La señal BIT_CNT tiene una funcionalidad interna dentro de la FSM (por lo que debe ser una variable que cambie de manera **síncrona**) y funciona como un indicador actual del *bit* que estamos enviando (equivalente a indexar un vector).

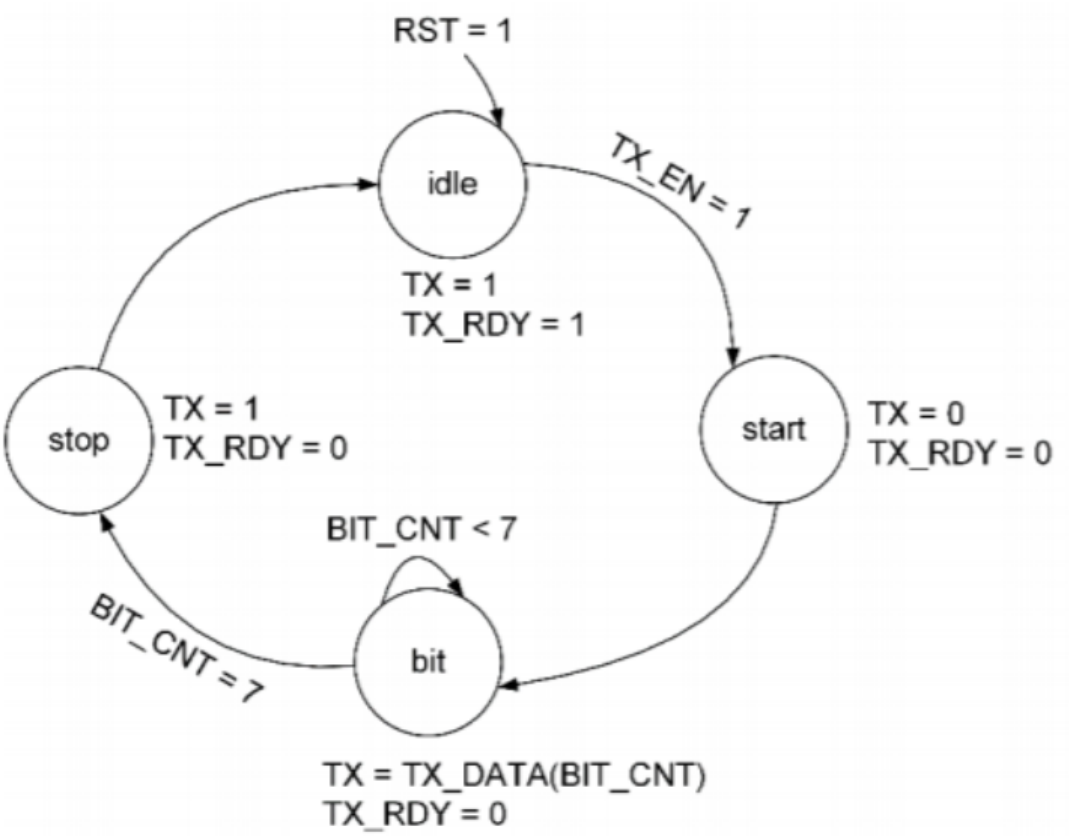


Figura 5: FSM de una Tx UART.

Debe ser incrementado cada vez para enviar el *bit* siguiente, además de reiniciar su cuenta cuando se salga del estado BIT. Las funciones de las demás señales se pueden deducir del diagrama de estados. En la figura 6 pueden ver cómo interactúan las señales para enviar una letra 'A'. Algunas consideraciones del diagrama de tiempo:

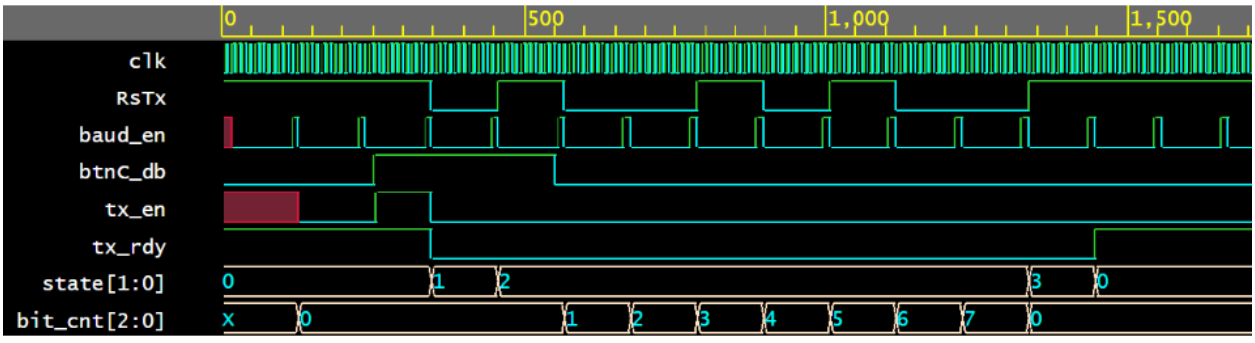


Figura 6: Diagrama de tiempo para una transmisión UART.

- El tiempo no está a escala. La señal BAUD_EN no refleja una transmisión a 9600 ya que no se está contando hasta 10415. Lo que indica el diagrama es cómo interactúan las señales.
- El estado '2' corresponde al estado BIT y se pueden dar cuenta cómo durante este estado el BIT_CNT aumenta.
- En los estados START y STOP ('1' y '3') pueden ver que se generan los *bits* '0' y '1' que se menciono en la sección 1.
- En este caso, el botón indico el inicio de la transmisión activando a TX_EN, sin embargo, la comunicación no inicia hasta que ocurra el pulso en BAUD_EN (para mantener la sincronización). Luego, TX_EN vuelve a '0' ya que queremos transmitir solo una vez y no el mismo caracter tantas veces como se mantenga apretado el botón. En este ejemplo si se aprieta el botón, se envía un caracter.
- La señal TX_RDY se explica por si sola y puede ser útil usarla para detectar cuándo se terminó una transmisión para seguir a la próxima y así transmitir varios caracteres seguidos.

Por último, no olvide habilitar en su archivo *contraints* el transmisor UART que corresponde a la interfaz USB-RS232.

RECOMENDACIÓN: Para probar que su módulo funciona, conecte la entrada TX_DATA a los *switches* y por medio de un botón envíe alguna letra. Verifique que recibió la misma letra en el computador.

2.3. Receptor

Para crear un receptor UART el principio es el mismo que para la transmisión. Sin embargo, hay que tomar en consideración que como UART es un protocolo asíncrono, las señales BAUD_EN no están sincronizadas entre Tx y Rx (en realidad, ni siquiera los *clocks* están sincronizados).

El problema de sincronización se soluciona sobre muestreando la señal de entrada. En el peor de los casos, si el *clock* en el receptor es muy lento, puede ocurrir un cambio en la entrada (asíncronamente) entre dos flancos de *clock*, por lo que perderíamos información o incluso podríamos no detectar muy tarde el inicio de un *byte*. Por esta razón es que el clock debe ser como mínimo unas 16 veces más rápido que el *baud rate* (puede ser más pero siempre multiplos pares del *baud rate*), de esta forma nos podemos demorar como máximo 1/16 en detectar un cambio en la entrada.

Luego de sobre muestrear varias veces un mismo *bit*, se guarda solo un valor como el valor del *bit* (la muestra central). Para guardar los valores que vienen de manera serial y

después pasarlos a forma paralela, se debe utilizar un ***shift register*** para ir almacenando las muestras centrales de cada bit recibido.

TIP: En su módulo debe crear un *shift register* para ir almacenando los bits que le van a llegar serialmente (de a uno). En la práctica esto se puede hacer aprovechando las operaciones *shift right* y *shift left* para así crear una variable de varios *bits* y que cada vez que se tenga que guardar un valor nuevo, crear un espacio moviendo los *bits* previos a la izquierda/derecha.

Tomando en consideración esta necesidad de sobre muestrear, la señal BAUD_EN debe ser 16 veces más rápida que en el caso anterior. Su implementación es la misma que para el contador implementado en el transmisor, por lo que también tiene que ser de tipo *one-shot.*

- Para una frecuencia de 100MHz, el período debe ser dividido por 651 (o 10416/16).

El módulo receptor y su diagrama de estados se encuentran en las figuras 7 y 8 respectivamente.

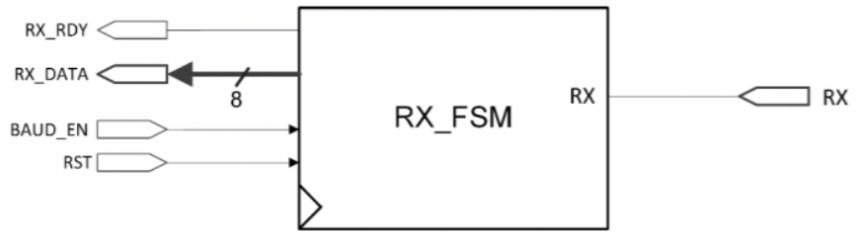


Figura 7: Bloque Rx UART.

Si bien el diagrama de estados se ve más complejo, el principio es el mismo. Se trata de una FSM común, por lo que la forma de enfrentar el diseño de ésta en Verilog es similar. Es decir, la estructura general del código se mantiene igual que para el transmisor (vea la parte de consejos al final).

Esta máquina de estados tiene dos contadores auxiliares (para Tx estaba solo BIT_CNT). Estos son BAUD_CNT y BIT_CNT. La funcionalidad de BIT_CNT es análoga su función en el transmisor y representa el número del *bit* actual que se esta recibiendo. No olvidar que debe ser reiniciado a '0' una vez terminada cada recepción.

El caso de BAUD_CNT es un poco distinto. También es un contador dentro de la FSM (por lo que debe funcionar **síncronamente** al igual que BIT_CNT en el Tx y Rx) y cuenta el numero de pulsos BAUD_EN para proporcionar un correcto funcionamiento del sobre muestreo. Recuerden que en este caso estamos sobre muestreando 16 veces cada *bit.*

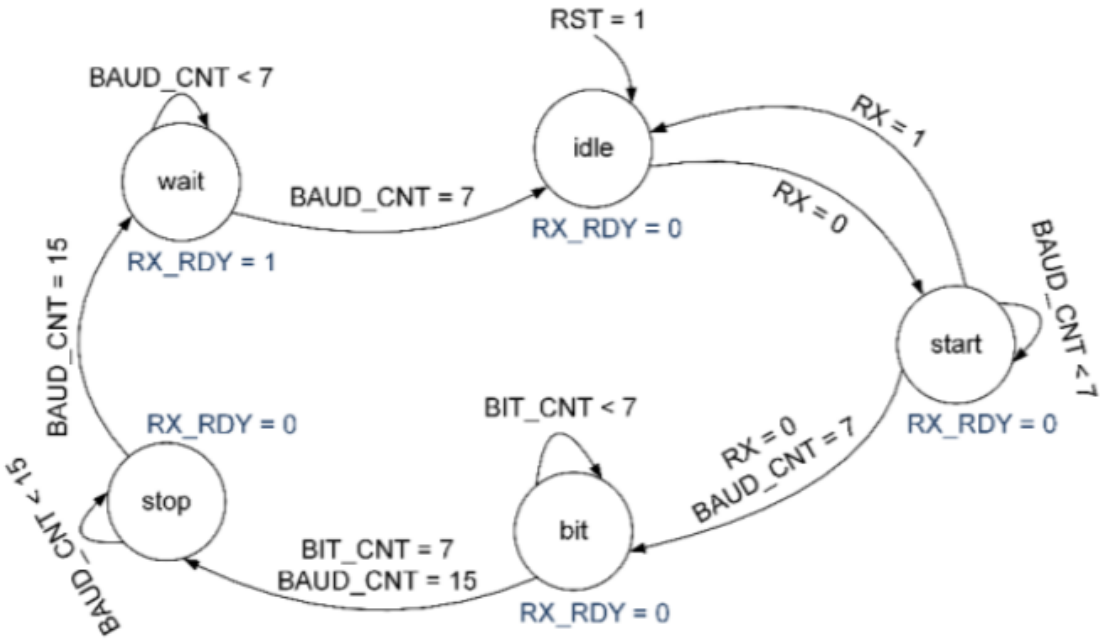


Figura 8: FSM de un Rx UART.

La importancia del estado START, es que al esperar a que BAUD_CNT sea '7' (y no '15') es para desfasar la toma de la muestra a la muestra central (y no exactamente en el momento cuando se produce el cambio de un *bit* a otro). El estado WAIT espera a que BAUD_CNT sea '7' nuevamente para volver al desfase cero original.

IMPORTANTE: Al igual que para el transmisor, los cambios de estado se producen únicamente en el pulso cuando BAUD_EN es '1'. Es importante mantener la sincronía para asegurarnos que cada *bit* tenga la misma cantidad de tiempo.

RECOMENDACIÓN: Para probar que su módulo funciona, conecten la salida RX_DATA al *display* de 7 segmentos y que muestre el último valor recibido. Una vez que este módulo funcione, pasen a la fase de integración con la RAM y el Tx.

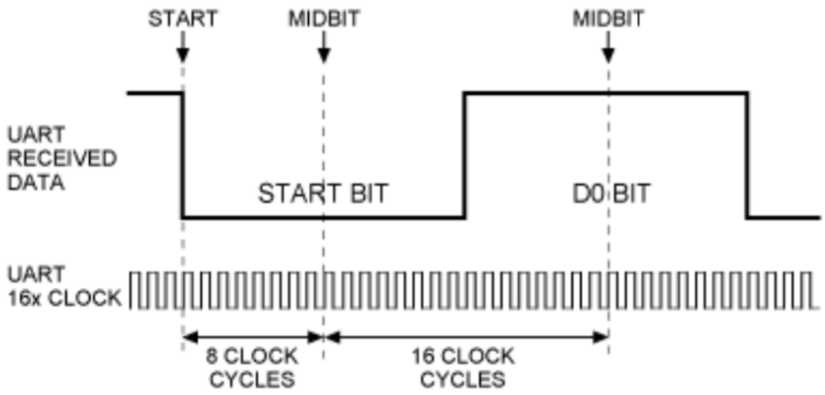


Figura 9: Ejemplo de sobre muestreo a 16x para el Rx.

3. Consejos

- Parta con un diagrama de bloques de su circuito completo. Incluyendo la RAM, el *input* del botón, el bloque transmisor y el bloque receptor.
- Pueden usar los LEDs o los *display* de 7 segmentos para mostrar información útil que les sirva para ver cómo está funcionando su código y hacer mas fácil el trabajo de *debug* (a tasas lentas ya que con 9600 los cambios son irreconocibles).
- Otra excelente opción es utilizar Digital o EDA Playground para verificar que sus señales se comporten de manera correcta. Si llevan mucho tiempo estancados, esta es la opción más rápida para ver cómo cambian sus estados y variables en el tiempo.
- Dividir y conquistar: Trabaje enfocándose en un módulo antes de pasar al siguiente. Vayan probando de manera continua para asegurarse que sus módulos funcionan bien. Ejemplo: cuando termine su módulo Tx, pruebe enviando caracteres al PC. Los debería poder ver en su computador y no es necesario tener previamente la RAM ni el módulo Rx (podrían mandar un caracter indicado por *switches* en vez de valores guardados en RAM).
- Los modulos Tx y Rx son FSM, por lo que sus módulos en Verilog siguen un esqueleto común para todas las FSM. Repase las diapositivas de clases y busque en internet cómo hacer máquinas de estado finitas en Verilog de manera genérica antes de comenzar a escribir en Vivado.
- Si CoolTerm muestra el mismo caracter independiente de cuál se envíe, probablemente haya un error en el contador o en la duración de cada estado. El envío debe ser totalmente constante y cada *bit* debe durar exactamente la misma cantidad de flancos de reloj.

- Los contadores internos de las FSM funcionan de manera distinta a las entradas y salidas de una FSM. Si bien las entradas y salidas se generan de manera asíncrona y solo el cambio de estados es síncrono, como los contadores son inherentemente síncronos no deben ser cambiados de forma asíncrona. Con esto se evitaran bastantes problemas y tiempo de *debug* y corrección.

4. Entrega

- Deben entregar los archivos .v de cada módulo que utilizo en su tarea, además del archivo .bit. No entregue el proyecto de Vivado completo. Los códigos deben estar debidamente comentados explicando la funcionalidad de cada grupo de líneas.
- Deben entregar un .pdf (NO .doc/.docx) explicando el funcionamiento de sus módulos y cómo se relacionan con los otros. No se contengan en su explicación, siempre es mejor dejar bien descrito el funcionamiento de su código para mostrar que entendieron bien.
- Pueden usar diagramas de bloques, dibujos, e incluso adjuntar el *link* a un video en el que explican su código (y así se ahorran tener que escribirlo todo).
- NO se aceptan implementaciones con FSM y entradas distintas a las que se indican en el enunciado. Si creen que necesitan una entrada o salida extra la pueden agregar, pero no pueden eliminar ningún puerto a los Tx y Rx. Hay otras soluciones en internet, pero lo importante es que aprendan a diseñar FSM por lo que deben seguir los diagramas de estados en esta tarea.

5. Referencias

Esta tarea fue basada en las guías de laboratorio del curso Arquitectura de Computadores del profesor Mihai Negru. Pueden encontrarlas acá.