Facultad de
Ciencias Exactas
Físicas y Naturales

Arquitectura de Computadoras 2023
Trabajo Práctico N°2
Módulo UART

Alumna: Gina Commisso
Materia: Arquitectura de Computadoras
Profesores: Martin M. Pereyra, Santiago A. Rodriguez

Índice

rx_uart **3**
Descripción de Entradas y Salidas 3
Máquina de Estados 3
Almacenamiento de Bits 4
Reseteo y Sincronización 4
tx_uart **5**
Descripción de Entradas y Salidas 5
Descripción Interna 5
Máquina de Estados 5
Control de los Contadores 6
Almacenamiento y Desplazamiento de Bits 6
Reseteo y Sincronización 6
uart_alu_interface **7**
Descripción de Entradas y Salidas 7
Descripción Interna 7
Máquina de Estados 7
Almacenamiento de Datos 8
Sincronización y Reset 8
Funcionamiento General 8
mod_m_counter **9**
Descripción de Entradas y Salidas 9
Parámetros 9
Descripción Interna 9
Contador 9
Generación de la señal s_tick 9
Funcionamiento General 10
Aplicaciones 10
UART **11**
Descripción de Entradas y Salidas 11
Entradas: 11
Salidas: 11
Parámetros 11
top **12**

rx_uart

El módulo `rx_uart` es un receptor UART (Universal Asynchronous Receiver/Transmitter) diseñado para recibir datos seriales y convertirlos en formato paralelo. Funciona mediante la detección de ticks de reloj para sincronizar la recepción de bits individuales. Este módulo forma parte de un sistema de comunicación UART, que es ampliamente utilizado para la transmisión y recepción de datos de manera asíncrona.

Descripción de Entradas y Salidas

- **Entradas:**
  - `clk`: Señal de reloj para sincronizar el funcionamiento del módulo.
  - `reset`: Señal de reset que reinicia el estado del módulo a su estado inicial.
  - `rx`: Entrada de datos seriales, es decir, los bits que se están recibiendo de manera asíncrona.
  - `s_tick`: Señal que indica la ocurrencia de un tick de reloj válido para la transmisión. Esto se utiliza para asegurarse de que se recibe un bit completo.
- **Salidas:**
  - `rx_done_tick`: Señal que indica que se ha completado la recepción de un byte (8 bits).
  - `data_out`: Salida paralela de los datos recibidos, lista para ser enviada a la interfaz o procesador.

Máquina de Estados

El funcionamiento del módulo se organiza en cuatro estados principales mediante una máquina de estados finita (FSM). Estos estados se definen como constantes utilizando `localparam` y se describen a continuación:

1. **Idle (2'b00):**
   - El módulo está en estado de reposo esperando la llegada de un bit de inicio (start bit).
   - Si el bit de inicio es detectado (`~rx`), el estado cambia a `start`.
2. **Start (2'b01):**
   - En este estado, el módulo espera que se estabilice el bit de inicio. Este proceso está controlado por un contador (`s_reg`), que incrementa con cada tick válido (`s_tick`).
   - Cuando el contador llega a 7, se asume que el bit de inicio ha sido correctamente detectado y el estado cambia a `data`.
3. **Data (2'b10):**
   - Aquí se empieza a recibir los bits de datos, uno por uno. El registro de bits (`b_reg`) se utiliza para almacenar los bits recibidos.
   - Se utiliza un contador (`n_reg`) para llevar la cuenta de cuántos bits se han recibido. El proceso de recepción de bits continúa hasta que se reciben los 8 bits.

    - El módulo también reinicia el contador `s_reg` en cada bit recibido y cambia el estado a `stop` una vez que se han recibido todos los bits.
4. **Stop (2'b11):**
    - En este estado, el módulo espera la recepción del bit de parada (stop bit), que indica el fin de la transmisión.
    - Si se recibe correctamente el bit de parada, el estado vuelve a `idle` y se activa la señal `rx_done_tick` para indicar que los datos están listos para ser procesados.

Almacenamiento de Bits

- El registro `b_reg` se utiliza para almacenar los bits que se van recibiendo. Durante el estado `data`, los bits se desplazan dentro de este registro con cada tick de reloj válido. Al completar la recepción de los 8 bits, el contenido de `b_reg` se pasa a la salida `data_out`.

Reseteo y Sincronización

- El módulo se reinicia completamente cuando se activa la señal `reset`, volviendo todos los registros y contadores a su estado inicial. Esto es útil para mantener la sincronización en caso de errores o pérdida de comunicación.
- Los contadores `s_reg` y `n_reg` son utilizados para controlar el timing de la recepción de los bits y para garantizar que los bits sean capturados con precisión.

tx_uart

El módulo `tx_uart` es un transmisor UART (Universal Asynchronous Receiver/Transmitter) encargado de convertir datos paralelos en una señal serial para ser transmitida bit a bit. Funciona mediante una máquina de estados finita (FSM) que controla el flujo de los datos y asegura que se transmiten en el formato correcto, incluyendo los bits de inicio y parada, esenciales en la comunicación asíncrona.

Descripción de Entradas y Salidas

- **Entradas:**
  - `clk`: Señal de reloj que sincroniza el funcionamiento del módulo.
  - `reset`: Señal de reset que reinicia el transmisor a su estado inicial.
  - `tx_start`: Señal que indica el inicio de la transmisión de un nuevo dato.
  - `s_tick`: Señal que actúa como un tick de temporización para sincronizar la transmisión de los bits.
  - `data_in`: Dato paralelo de 8 bits (configurable por parámetro) que se va a transmitir serialmente.
- **Salidas:**
  - `data_sent`: Señal que indica que el dato ha sido completamente transmitido.
  - `tx`: Salida de datos seriales, que transmite un bit a la vez hacia el receptor.

Descripción Interna

Máquina de Estados

El módulo está organizado en cuatro estados principales utilizando una máquina de estados finita, similar a la estructura del receptor. Los estados son los siguientes:

1. **Idle (2'b00):**
   - Este es el estado de reposo del transmisor, donde se espera la señal `tx_start` para comenzar la transmisión.
   - En este estado, la línea de transmisión (`tx`) se mantiene en un valor alto (`1`), como es estándar en UART cuando no hay datos en curso.
   - Cuando `tx_start` se activa, el módulo se prepara para transmitir el bit de inicio (start bit) y pasa al estado `start`.
2. **Start (2'b01):**
   - En este estado, se transmite el bit de inicio, que siempre es un valor bajo (`0`). Esto le indica al receptor que un nuevo byte está por ser transmitido.
   - El contador `s_reg` controla la duración del bit de inicio, y cuando alcanza el valor requerido (7 ticks), el estado cambia a `data`.
3. **Data (2'b10):**

    - Aquí se transmiten los bits del dato que se encuentra en `data_in`. Cada bit se envía uno a uno, comenzando desde el bit menos significativo.
    - El registro `send_byte` contiene los bits del dato a transmitir y se desplaza a la derecha después de enviar cada bit.
    - El contador `n_reg` lleva el conteo de cuántos bits se han transmitido. Cuando se completa la transmisión de todos los bits, el estado cambia a `stop`.
4. **Stop (2'b11):**
    - En este estado, se transmite el bit de parada (stop bit), que es un valor alto (1). El bit de parada señala el final de la transmisión del byte.
    - Una vez que se ha transmitido el bit de parada durante el número de ticks especificado por el parámetro SR, el estado vuelve a `idle` y se activa la señal `data_sent` para indicar que el proceso ha finalizado.

Control de los Contadores

- **Contador de ticks (`s_reg`):** Controla el tiempo que el transmisor permanece en cada estado, asegurándose de que cada bit se transmita con la duración adecuada. Se reinicia en cada estado y se incrementa con cada tick (`s_tick`).
- **Contador de bits (`n_reg`):** Lleva la cuenta de cuántos bits del dato han sido transmitidos. Cuando todos los bits han sido enviados, el transmisor pasa al estado de parada.

Almacenamiento y Desplazamiento de Bits

- El dato a transmitir (`data_in`) se almacena en el registro `send_byte`. Durante el estado `data`, el bit menos significativo de `send_byte` se asigna a la línea de transmisión (`tx`) y, luego de transmitirlo, el registro se desplaza a la derecha para colocar el siguiente bit en la posición menos significativa.

Reseteo y Sincronización

- Si se activa la señal `reset`, todos los registros y contadores se reinician, lo que permite que el transmisor vuelva a su estado inicial. Esto garantiza que no haya fallos en la transmisión debido a errores previos o desincronización.

uart_alu_interface

El módulo `uart_alu_interface` actúa como una interfaz entre los datos recibidos a través de una UART y una unidad aritmético-lógica (ALU). Su propósito es recibir datos seriales que representan los operandos y el código de operación de la ALU, almacenarlos, y finalmente señalizar a la ALU que los datos están listos para ser procesados.

Descripción de Entradas y Salidas

- **Entradas:**
  - `clk`: Señal de reloj, necesaria para sincronizar las operaciones del módulo.
  - `reset`: Señal que reinicia el estado de la máquina y todos los registros a sus valores iniciales.
  - `uart_data_in`: Dato recibido desde el receptor UART, representando los operandos o el código de operación de la ALU.
  - `uart_data_ready`: Señal que indica cuando el dato recibido por UART está listo para ser leído por la interfaz.
- **Salidas:**
  - `alu_data_ready`: Señal que indica a la ALU que los datos han sido recibidos y están listos para su procesamiento.
  - `alu_opA`: Primer operando que será enviado a la ALU.
  - `alu_opB`: Segundo operando que será enviado a la ALU.
  - `alu_opCode`: Código de operación que la ALU debe ejecutar.

Descripción Interna

Máquina de Estados

La lógica del módulo se organiza mediante una máquina de estados finita (FSM), con los siguientes estados definidos por un registro de 3 bits (`uart_state`):

1. **Estado 3'b000 (Esperar opA):**
   - El módulo se encuentra en espera de que la UART indique que el primer operando (`opA`) está listo.
   - Cuando `uart_data_ready` se activa, se almacena el valor de `uart_data_in` en el registro `opA_reg`, y el estado cambia a `3'b001`.
2. **Estado 3'b001 (Esperar opB):**
   - En este estado, el módulo espera que la UART indique que el segundo operando (`opB`) está listo.
   - Al activarse `uart_data_ready`, se almacena el valor de `uart_data_in` en el registro `opB_reg`, y el estado cambia a `3'b010`.

3. **Estado 3'b010 (Esperar opCode):**
    - En este punto, el módulo espera recibir el código de operación (`opCode`), que determinará la operación que ejecutará la ALU.
    - Al activarse `uart_data_ready`, el valor de `uart_data_in` se almacena en el registro `opCode_reg`, y el estado cambia a `3'b011`.
4. **Estado 3'b011 (Datos Listos para la ALU):**
    - Este estado indica que los datos (`opA`, `opB`, y `opCode`) están listos para ser procesados por la ALU.
    - La señal `alu_data_ready` se activa para avisar a la ALU que puede comenzar a operar.
    - Después de un ciclo de reloj, el estado vuelve a `3'b000`, para permitir la recepción de una nueva operación.

Almacenamiento de Datos

- **Registros de operandos y código de operación:**
    - El valor de `uart_data_in` se almacena en los registros `opA_reg`, `opB_reg` y `opCode_reg` según el estado de la máquina de estados.
    - Estos registros son asignados a las salidas `alu_opA`, `alu_opB` y `alu_opCode`, respectivamente, para que la ALU pueda utilizarlos.

Sincronización y Reset

- **Reseteo:**
    - Cuando se activa la señal `reset`, todos los registros (`opA_reg`, `opB_reg`, `opCode_reg`) y la máquina de estados (`uart_state`) se reinician a sus valores iniciales.
- **Sincronización:**
    - Las operaciones de este módulo están sincronizadas con el reloj (`clk`), lo que garantiza una operación estable y ordenada.

Funcionamiento General

Este módulo actúa como una interfaz de control que recibe los datos provenientes de un sistema UART y los prepara para ser enviados a una ALU. Los operandos (`alu_opA` y `alu_opB`) y el código de operación (`alu_opCode`) se reciben secuencialmente y, una vez almacenados, el módulo activa la señal `alu_data_ready` para indicar que la ALU puede proceder con el cálculo.

mod_m_counter

El módulo `mod_m_counter` es un contador parametrizable que cuenta hasta un valor `M`, definido por un parámetro, y luego se reinicia a cero. Al llegar a este valor, genera una señal de salida llamada `s_tick`, que indica que el contador ha completado un ciclo. Este tipo de contador es útil en diversas aplicaciones de temporización, generación de pulsos o división de frecuencias.

Descripción de Entradas y Salidas

- **Entradas:**
  - `clk`: Señal de reloj, utilizada para sincronizar el incremento del contador.
  - `reset`: Señal de reinicio que resetea el contador a su estado inicial.
- **Salidas:**
  - `s_tick`: Señal de salida que se activa cuando el contador alcanza el valor `M`. Permanece activa por un ciclo de reloj.

Parámetros

- **`M`**: Es el valor máximo que alcanzará el contador antes de reiniciarse. En este caso, su valor por defecto es 326.
- **`N`**: Es la cantidad de bits necesarios para representar el valor `M`. En este caso, se usa un valor predeterminado de 9 bits para representar 326.

Descripción Interna

El módulo usa un registro de tamaño `N` (en este caso, 9 bits) para contar los ciclos del reloj. La lógica interna se basa en un bloque secuencial controlado por la señal de reloj (`clk`) y reinicio (`reset`).

Contador

- El registro `counter` almacena el valor actual del contador, y en cada ciclo de reloj, su valor se incrementa en 1, siempre y cuando no se haya alcanzado el valor de `M`.
- Cuando el contador alcanza el valor de `M`, este se reinicia a 0 en el siguiente ciclo de reloj.

El funcionamiento puede describirse paso a paso:

1. **Reinicio**: Si la señal `reset` está activa, el contador se reinicia a cero.
2. **Incremento**: Mientras el valor del contador sea menor que `M`, este se incrementa en cada ciclo de reloj.

3. **Reinicio del ciclo**: Si el valor del contador llega a `M`, se reinicia a cero, comenzando un nuevo ciclo de conteo.

Generación de la señal `s_tick`

- La señal `s_tick` es una señal de un bit que se activa cuando el contador llega al valor de `M`. Se genera comparando el valor actual del contador con `M`.
- Cuando `counter` es igual a `M`, la salida `s_tick` se pone en alto (1), indicando que se ha completado el ciclo de conteo.

Funcionamiento General

El contador comienza desde 0 y se incrementa en cada ciclo de reloj hasta alcanzar el valor de `M` (en este caso, 326). Una vez que se alcanza el valor de `M`, el contador se reinicia, y la señal de salida `s_tick` se activa durante un ciclo de reloj. Este proceso se repite indefinidamente mientras el sistema esté habilitado y no se active la señal de `reset`.

Aplicaciones

El `mod_m_counter` puede utilizarse en:

1. **División de Frecuencia**: El contador puede ser utilizado para dividir la frecuencia de una señal de reloj. Por ejemplo, si el contador está parametrizado para contar hasta 326, la salida `s_tick` tendrá una frecuencia que es 1/327 de la frecuencia de entrada del reloj.
2. **Generación de Pulsos Temporales**: La señal `s_tick` puede usarse para generar un pulso de temporización una vez cada `M + 1` ciclos de reloj, lo que es útil en sistemas de control o temporizadores.
3. **Control de Eventos**: En sistemas donde se requiere que un evento ocurra después de un número fijo de ciclos de reloj, este tipo de contador puede ser utilizado para activar ese evento cuando el contador alcanza el valor deseado.

UART

El módulo UART es un sistema de comunicación asíncrona utilizado para la transmisión y recepción de datos entre la PC y un sistema digital, como una ALU u otros dispositivos. El UART se compone de tres submódulos principales: un contador de baud rate (`mod_m_counter`), un receptor (`rx_uart`) y un transmisor (`tx_uart`). Estos submódulos están interconectados para permitir una comunicación bidireccional mediante las señales `rx` y `tx`, que corresponden a la recepción y transmisión de datos.

Descripción de Entradas y Salidas

Entradas:

- **`clk`**: Señal de reloj del sistema, que sincroniza todas las operaciones internas del módulo UART.
- **`reset`**: Señal de reinicio que restablece el estado de los submódulos y registros internos.
- **`rx`**: Señal de entrada que recibe datos desde la PC.
- **`data_in`**: Dato que se envía desde la interfaz ALU hacia el transmisor UART para ser enviado a la PC.
- **`alu_result_ready`**: Señal que indica si el resultado de la ALU está listo para ser transmitido por el transmisor UART.

Salidas:

- **`tx`**: Señal de salida que transmite datos hacia la PC.
- **`rx_done_tick`**: Señal que indica que el receptor ha recibido un byte completo de datos, útil para sincronizar la interfaz con la llegada de datos.
- **`data_sent`**: Señal que indica que el transmisor ha enviado un byte de datos.
- **`data_out_rx`**: Datos recibidos completos por el receptor UART, listos para ser procesados por la interfaz.

Parámetros

- **`NB_OP`**: Número de bits del dato de entrada/salida, por defecto 8.
- **`NB_OPCode`**: Número de bits del código de operación, por defecto 6 (no se utiliza en este módulo específico).
- **`SR`**: Sample rate, determina cuántos ciclos de reloj se necesitan para muestrear la señal `rx`.

- **`M`**: Valor máximo que el contador debe alcanzar antes de generar un "tick" que sincroniza la velocidad de transmisión y recepción de datos.
- **`N`**: Número de bits para representar el valor máximo del contador `M`.

top

El módulo `top` combina una ALU y una interfaz UART para permitir la interacción entre una PC y una ALU a través de señales de transmisión y recepción serie (`tx` y `rx`). La ALU realiza operaciones aritméticas o lógicas sobre los operandos que se reciben a través del puerto `rx`, y los resultados se transmiten a través del puerto `tx`. El módulo incluye tres bloques principales: la ALU, la interfaz UART-ALU y el módulo UART.