Arquitectura de Computadoras

Maquinas de Estado Finitas
TP: UART

Temario

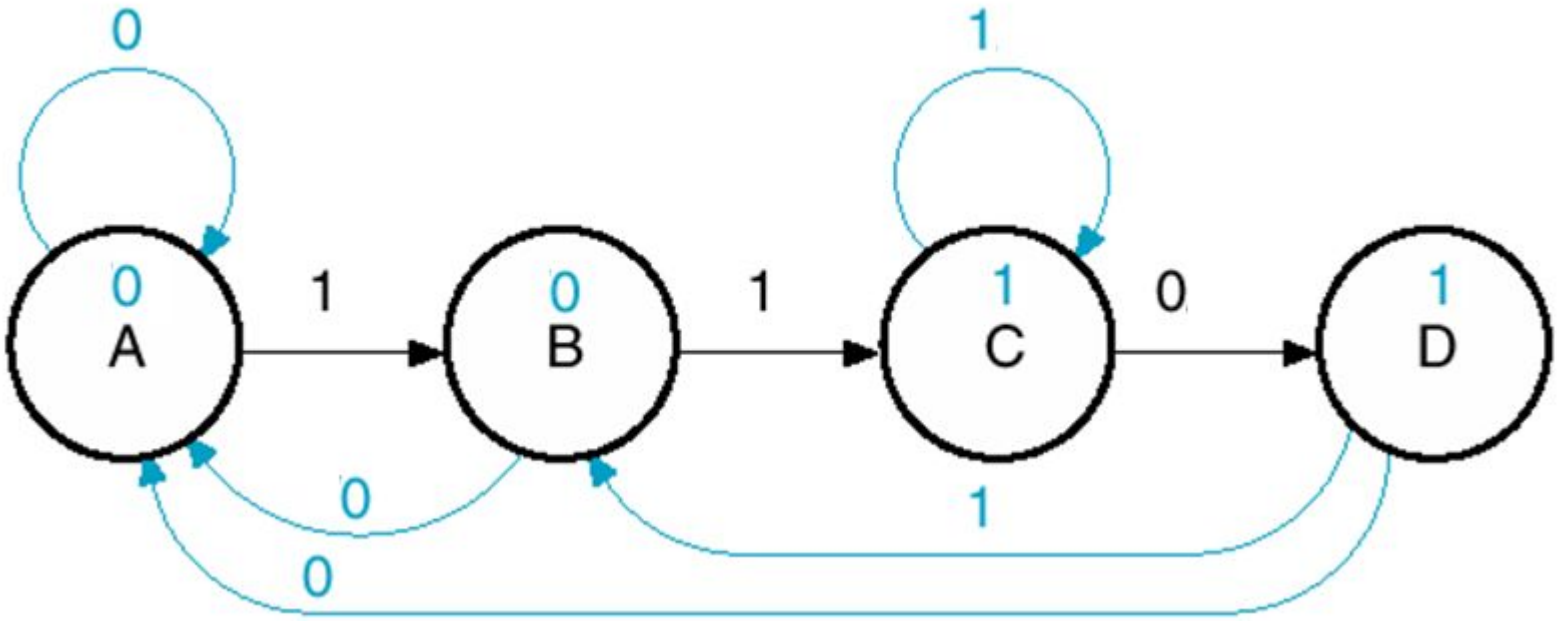
- Repaso de Máquinas de Estado
  - Definiciones
  - Representacion
  - Ejemplo
  - Consideraciones de Diseño
- Trabajo Práctico
  - UART

Maquinas de Estado Finitas (FSM)

- Las FSM se utilizan para ***modelar*** problemas que derivan en el diseño de ***lógica secuencial***.
- Modelan sistemas abstractos que pueden estar en un ***único*** estado de un conjunto de estados ***finitos***.
- Son usadas para modelar problemas en los que una ***secuencia de acciones*** depende de una ***secuencia de eventos***.

Maquinas de Estado Finitas (FSM)

- Representación



FSM: Representacion en HW

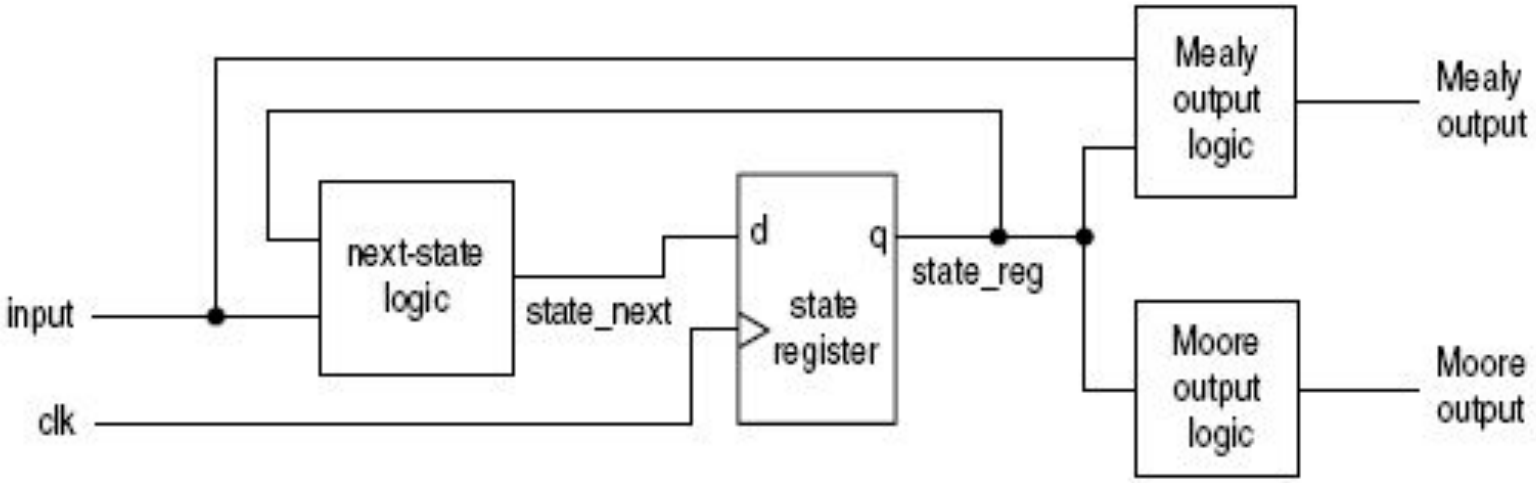
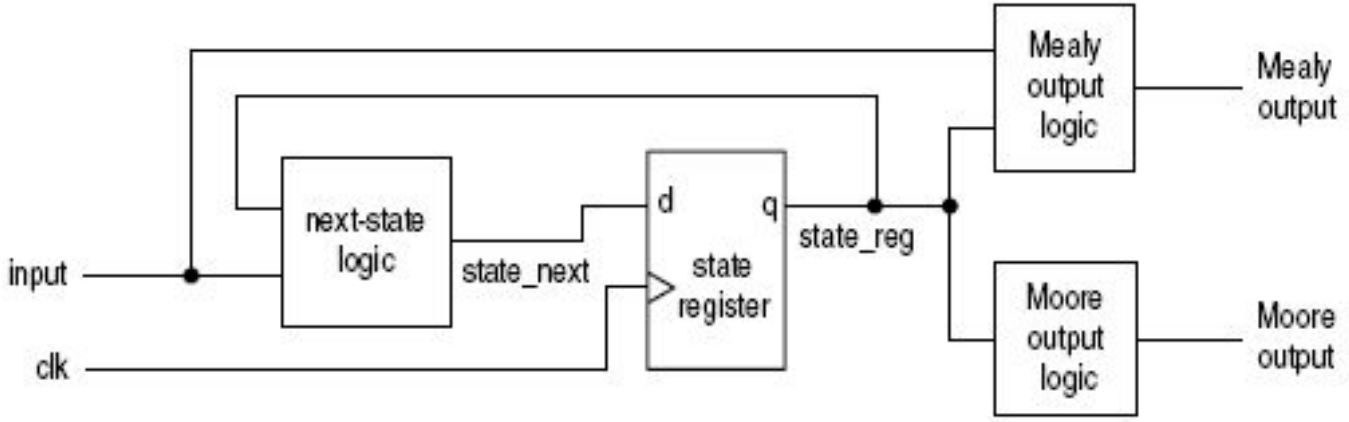


Figure 5.1 Block diagram of a synchronous FSM.

Tipos de FSM

- Se identifican dos tipos de máquinas de estados, dependiendo de la lógica de salida:
  - Máquina de Moore: La salida sólo depende del estado actual del circuito;
  - Máquina de Mealy: La salida es función del estado actual del circuito **Y** de la entrada.



Representación de Estados

- **Binaria**: Mínima cantidad de Flip-Flops
  - 000, 001, 010, 011…
- **One-Hot, One-Cold**: Mas usadas en FPGA
  - 0001, 0010, 0100, 1000
  - 1110, 1101, 1011, 0111.
- **Código Gray**: Imposible en FSM con transiciones complejas.
  - 000, 100, 110, 111…
- **Output-Equals-State**: Elimina lógica de salida

Consideraciones de Diseño

- Máquinas de Estado Seguras
  - Ante una entrada desconocida o ingreso a un estado inválido, en el siguiente ciclo se pasa a un estado de recuperación.
- Máquinas de Estado Rápidas
  - Utilizan menos lógica debido a la ausencia de un estado de recuperación.

Consideraciones de Diseño

- **Máxima Frecuencia:** One-Hot o One-Cold
- **Área:** Binaria
- **Min. Clock-To-Output delay:** Output-Equals-State o Output con registro
- **Glitch-Free Output:** Output-Equals-State o Output con registro
- **Mínimo Consumo:** Estados con código Gray

Maquinas de estado en Verilog

- Usar un módulo para definir la máquina de estado.
- Usar una variable para representar el estado de la FSM llamada **state**
- Usar una variable para representar el posible próximo estado de la máquina, llamada **next_state**.
- Usar **parameter** o **localparam** para definir los estados.

Maquinas de estado en Verilog

- Para codificar los estados se pueden usar parámetros:

```
parameter <state1> = 4'b0001;
parameter <state2> = 4'b0010;
parameter <state3> = 4'b0100;
parameter <state4> = 4'b1000;
```

- Para representar las variables de estado actual y siguiente:

```
reg [3:0] state = <state1>;
reg [3:0] next_state = <state2>;
```

Lógica de cambio de estado

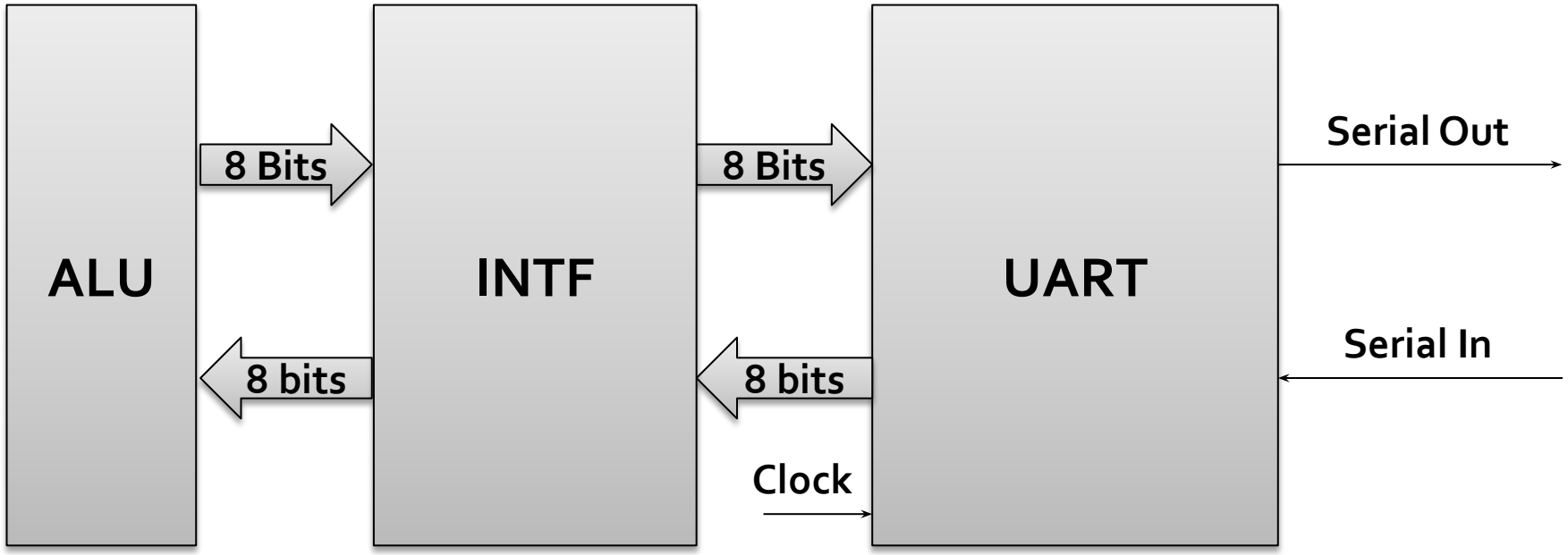
```
always @(posedge <clock>) //Memory
   if (reset)     state <= <state1>;
   else           state <= next_state;


always @* // Next-state logic
   case (state)
   <state1>: begin
           if (<condition>)
               next_state = <state1>;
               else
               next_state = <state2>;
            end
   <state2>:
           .
           .
           .
   default: next_state = <state1>; // Fault Recovery
   endcase
```

Lógica de las salidas

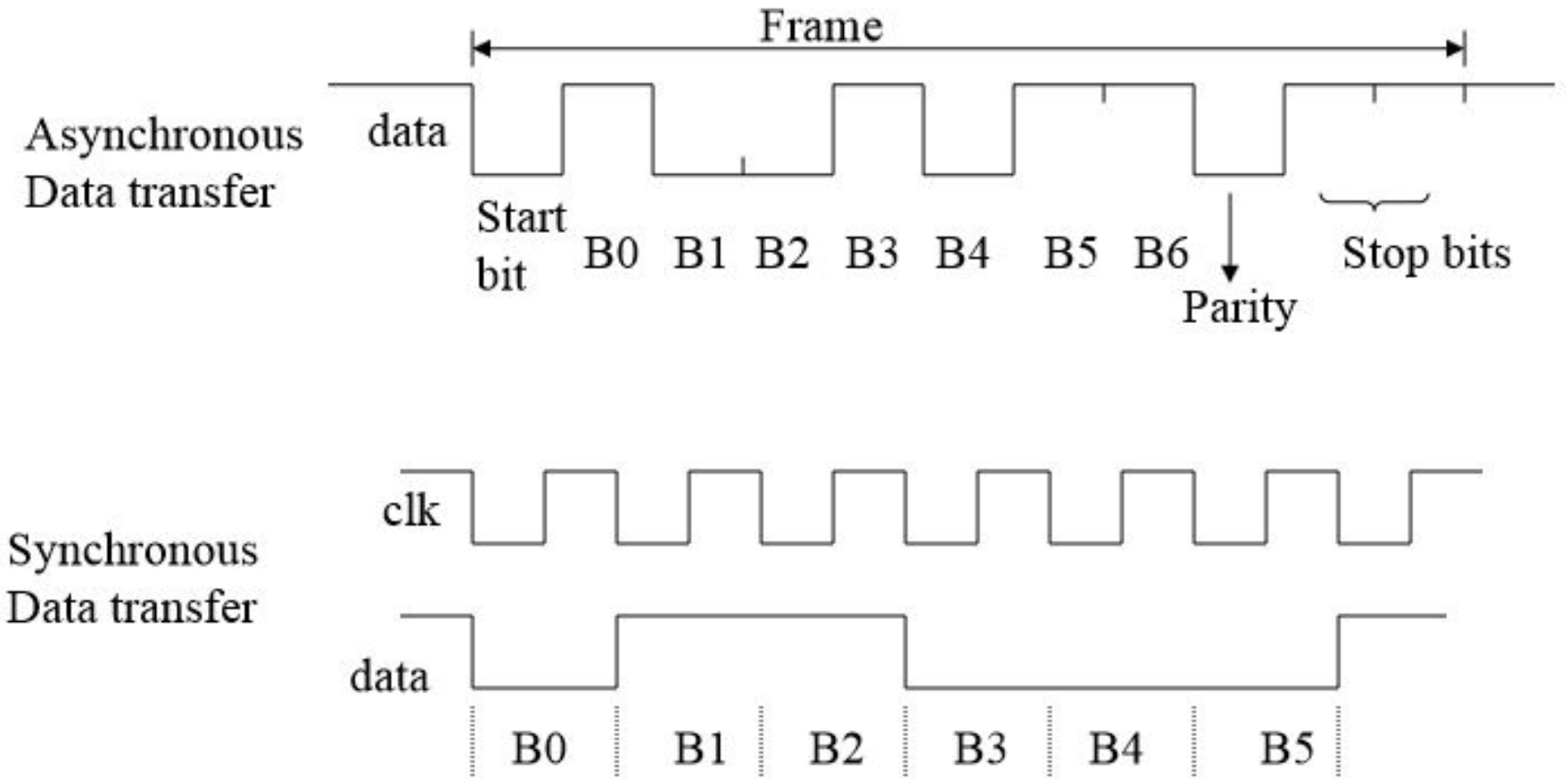
```
always @* // Output logic
   case (state)
   <state1>:
      <outputs> = <values>;
   <state2>:
         .
         .
         .
   default:
      <outputs> = <values>; // Fault Recovery
   endcase
```

TP: UART

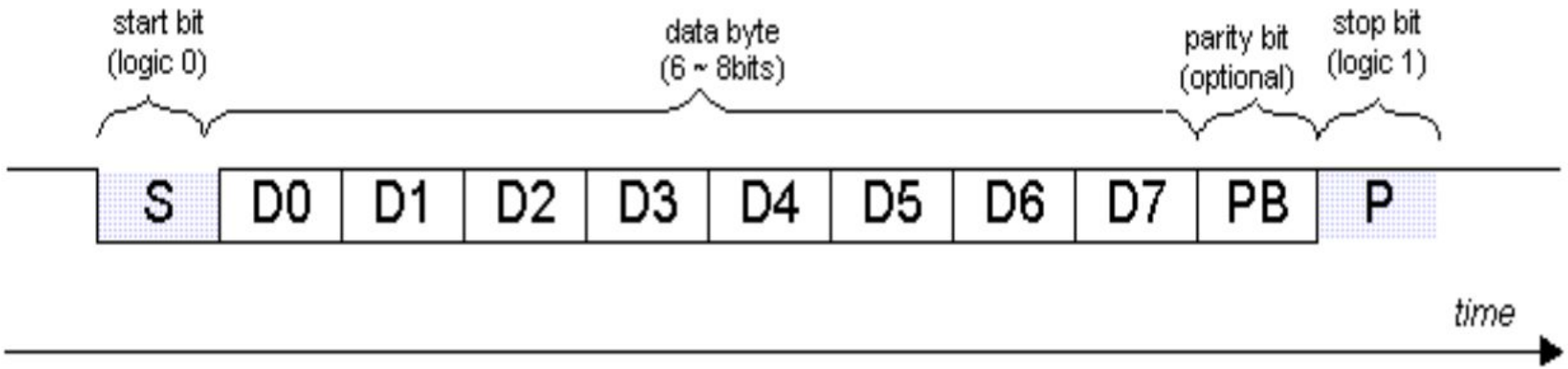
- Universal Asynchronous Receiver and Transmitter

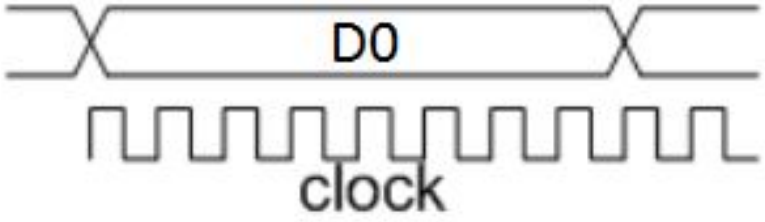


Comunicacion Síncrona vs Asíncrona



Trama UART


start bit
(logic 0)
data byte
(6 ~ 8bits)
parity bit
(optional)
stop bit
(logic 1)
S
D0
D1
D2
D3
D4
D5
D6
D7
PB
P
time


D0
clock

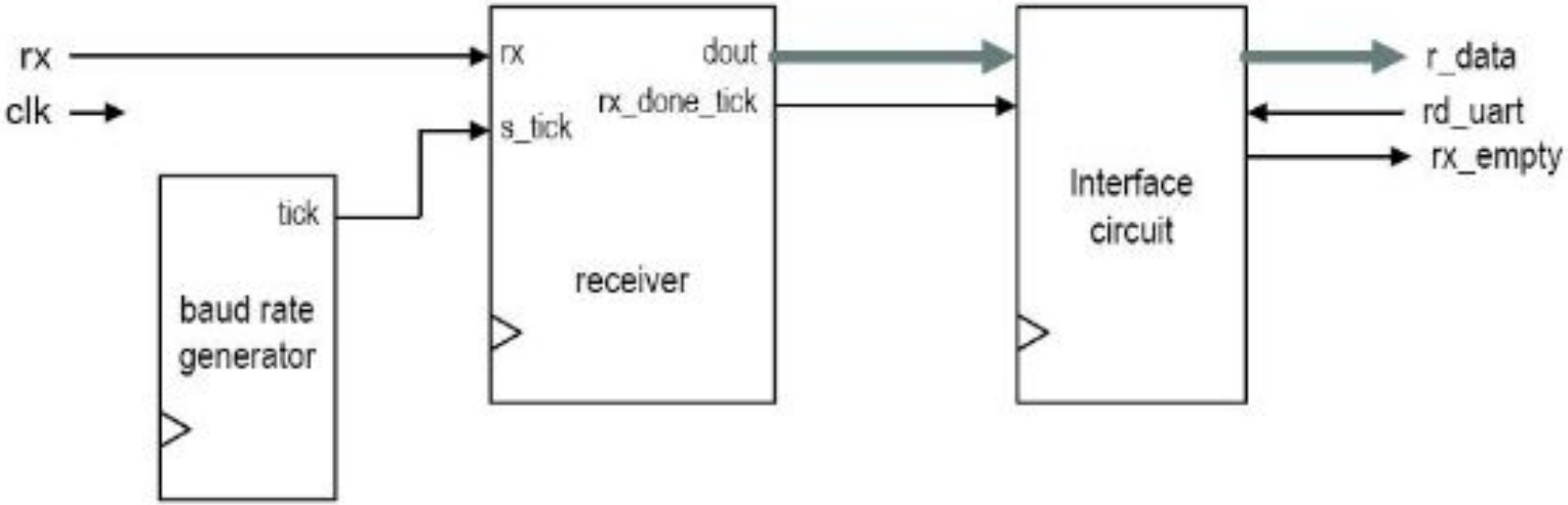
Baud Rate Generator

- Genera un Tick 16 veces por Baud Rate
- Si baud rate es 19.200 ciclos por segundo, la frecuencia de muestreo debe ser 19.200 *16= 307.200 ticks por segundo. Si el clock de la placa es 50 Mhz, hay que generar un tick cada 163 ciclos de reloj.

$$\frac{Clock}{BaudRate * 16} \cong 163$$

- El Baud Rate Generator es un contador módulo 163.

Receptor UART



Secuencia de estados Rx

- Asumiendo N bits de datos, M bits de Stop.
  - 1) Esperar a que la señal de entrada sea 0, momento en el que inicia el bit de Start. Iniciar el Tick Counter.
  - 2) Cuando el contador llega a 7, la señal de entrada está en el punto medio del bit de Start. Reinicar el contador.
  - 3) Cuando el contador llega a 15, la señal de entrada avanza 1 bit, y alcanza la mitad del primer bit de datos. Tomar este valor e ingresarlo en un shift register. Reinicar el contador.
  - 4) Repetir el paso 3 N-1 veces para tomar los bits restantes.
  - 5) Si se usa bit de paridad, repetir el paso 3 una vez mas.
  - 6) Reperir el paso 3 M veces, para obtener los bits de Stop.

TP: UART