پروژه درس UART receiver – VHDL

استاد درس: دکتر سیاوش اسحقی

مرضیه غیور نجف آبادی – بهار معدلی

98242138 – 98242112

UART

فرستنده سریال غیرهمزمان جهانی(UART)، روش انتقال سریال است که در هر لحظه فقط یک بیت را ارسال میکند، سرعت کمتری نسبت به روش انتقال سری دارد اما هزینه کمتری دارد و برای انتقال در فواصل طولانی بهتر است. همچنین فقط در هرلحظه باید پردازش روی یک بیت صورت بگیرد و این موضوع میتواند یکی از فواید این ارتباط شمرده شود.

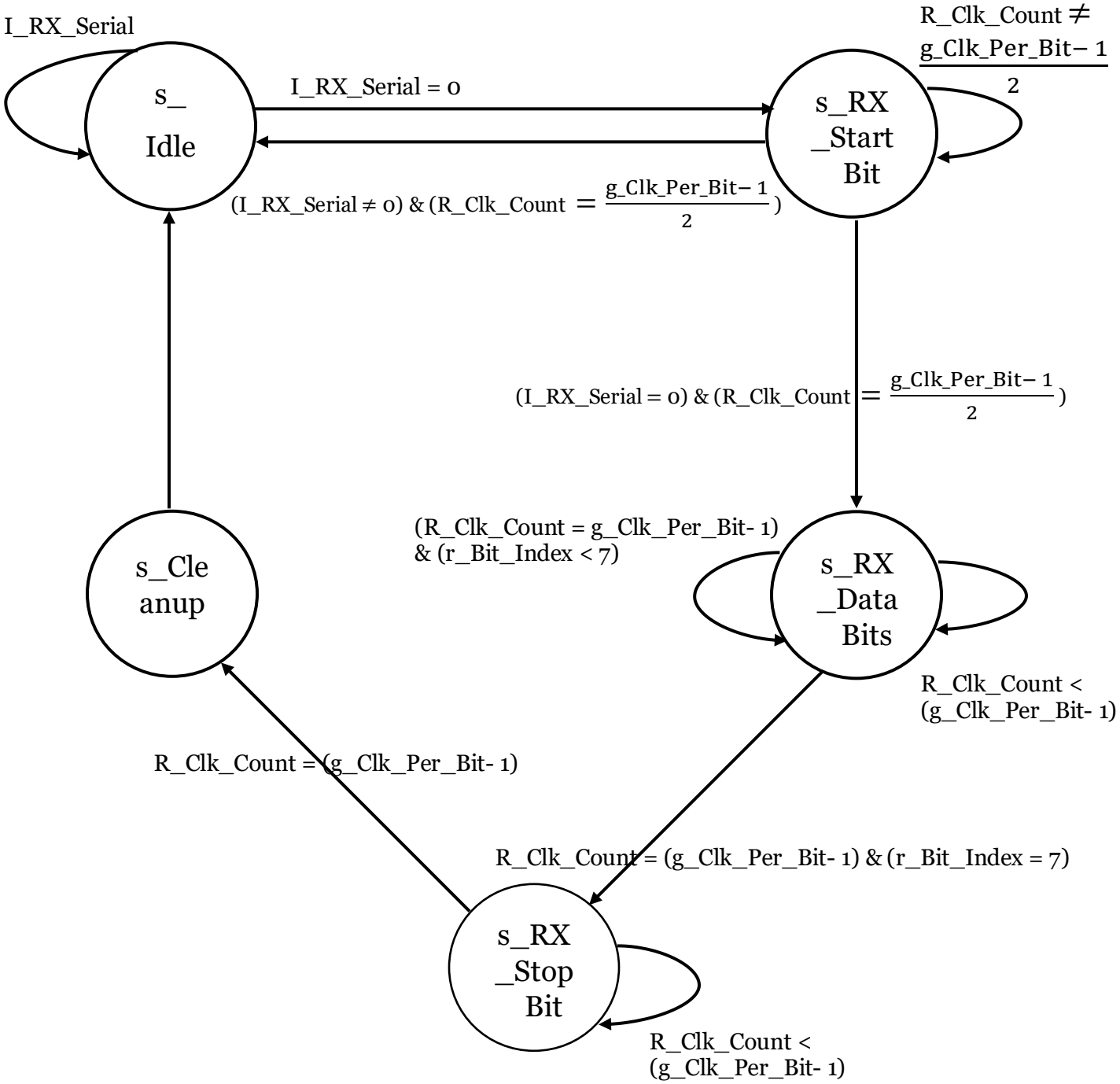
این ارتباط می تواند 8 بیت را به صورت سریال دریافت کند که 8 بیت سریال، یک start بیت و یک stop بیت دارد. در این پروژه بیت parity درنظر گرفته نشده و وقتی دیتا به طور کامل دریافت شد، بیت valid به مدت یک clock cycle مقدار 1 را تحویل می‌دهد.

UART receiver: در حالتی که هیچ ارسالی صورت نگرفته باشد، خط انتقال در حالت بیکار قرار دارد. (idle) و سطح ولتاژ 1 منطقی بر روی خط قرار میگیرد. تغییر وضعیت از 1 به صفر به مدت یک clock cycle به معنای شروع ارسال است و گیرنده آماده دریافت اطلاعات می‌شود. سپس 8 بیت اطلاعات را دریافت کرده و منتظر بیت valid می‌ماند.

Parallelization: پس از دریافت 8 بیت، داده سریالی به موازی تبدیل می‌شود.

FSM

چون مدار ما به صورت ترتیبی(sequential) می باشد و نیاز به یک مدار کنترلی دارد، بنابراین می توانیم آن را به صورت FSM توصیف کنیم. برای انتقال دیتا ا=توسط سریال به 5 state احتیاج داریم:


I_RX_Serial
s_
Idle
I_RX_Serial = 0
R_Clk_Count ≠ g_Clk_Per_Bit− 1 / 2
s_RX
_Start
Bit
(I_RX_Serial ≠ 0) & (R_Clk_Count = g_Clk_Per_Bit− 1 / 2)
(I_RX_Serial = 0) & (R_Clk_Count = g_Clk_Per_Bit− 1 / 2)
s_Cle
anup
(R_Clk_Count = g_Clk_Per_Bit- 1)
& (r_Bit_Index < 7)
s_RX
_Data
Bits
R_Clk_Count <
(g_Clk_Per_Bit- 1)
R_Clk_Count = (g_Clk_Per_Bit- 1)
R_Clk_Count = (g_Clk_Per_Bit- 1) & (r_Bit_Index = 7)
s_RX
_Stop
Bit
R_Clk_Count <
(g_Clk_Per_Bit- 1)

1. **S_Idle**

در این state هیچ عملیاتی صورت نمیگیرد و منتظر می‌ماند تا بیت 0 را از فرستنده دریافت کند و در این صورت به s_RX_Start_Bit میرود، در غیر اینصورت در همین state منتظر می‌ماند.

2. **S_RX_Start_Bit**

در این state صبر میکند تا به وسط شمارش clock در این مرحله برسد، در این صورت، ورودی سریال را مجددا چک میکند برای اطمینان از درست بودن بیت 0 فرستاده شده، اگر ورودی 0 اشتباه فرستاده شده باشد به s_Idle بر میگردد و اگر شمارش clock به وسط آن نرسیده بود به s_RX_Start_Bit برمیگردد و منتظر رسیدن به وسط clock می‌ماند. اگر به وسط clock برسد و 0 به درستی فرستاده شده باشد، به s_RX_Data_Bits می‌رود.

3. **S_RX_Data_Bits**

در این مرحله دریافت دیتا شروع می‌شود. برای دریافت هر بیت از دیتا اطمینان حاصل می‌کند که یک clk_Per_Bit به طور کامل طی شده باشد، سپس هر بیت از دیتا را به آخر یک متغیر 8 بیتی اضافه میکند. و درنهایت این متغیر 8 بیتی را به سیگنال خروجی می‌دهد. به فرآیند تبدیل بیت های سریالی به یک دیتا چند بیتی parallelization می‌گویند. پس از دریافت هر 8 بیت به s_Stop_Bit میرویم.

4. **S_RX_Stop_Bit**

پس از دریافت 8 بیت در این state به اندازه یک clk_Per_Bit منتظر می‌ماند و سپس خروجی r_RX_Valid را به معنای دریافت کامل 8 بیت، 1 کرده و به s_Cleanup می‌رود.

5. **S_Cleanup**

در این state، متغیر r_RX_Valid را 0 میکند تا بتواند برای دریافت دیتای جدید آماده باشد و به s_Idle میرود و مجددا برای شروع فرآیند دریافت دیتای بعدی منتظر می‌ماند.

References

1. https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html
2. https://nandland.com
3. https://www.quantil.com/content-delivery-insights/content-acceleration/data-transmission/#:~:text=There%20are%20two%20methods%20used,same%20time%20over%20multiple%20channels