گزارش فاز اول پروژه طراحی سیستم های دیجیتال

بنیامین جامی الاحمدی

94105282

محمد مهدی رضوی

94102982

علی ملکی

94102155

ماژول TX

ورودی ها: clock, send, tx_data, reset,

اگر send برابر 1 شود یعنی ماژول در حال فرستادن دیتا می باشد.

Tx_data یک bus 8بیتی است که حاوی data مورد نظر برای فرستادن می باشد.

اگر reset برابر 1 شود ماژول به حالت idle می رود.

خروجی ها: tx, tx_done

Tx یک خروجی تک بیتی است که دیتا مورد نظر را به صورت سریال از ماژول خارج می کند

Tx_done هنگامی که دیتا به صورت کامل از tx خارج شد به 1 تبدیل میشود

r_SM_Main: یک رجیستر 3 بیتی است که state فعلی در آن ذخیره می شود

r_Bit_Index: یک index است برای مشخص کردن اینکه بیت چندم از دیتا در حال خوانده شدن است

r_Tx_Data: یک رجیستر است که 8 بیت دیتا ورودی در آن ذخیره می شود تا در نهایت به صورت سریال خارج شود

در ابتدا برای حر حالت از ماژول یک عدد تعیین می کنیم .سپس چک می شود که اگر ریست برابر 1 باشد حالت idle در SM_Main ریخته می شود حال SM_Main چک می شود اگر مقدار آن برابر حالت idle باشد tx برابر 1 میشود(همانطور که در صورت پروژه گفته شده است در حالت idle مقدار tx برابر 1 می باشد) و tx_done را برابر 0 می کنیم زیرا دیتا مورد نظر فرستاده نشده است و Clock_Count و Bit_Index را 0 قرار میدهیم. حال در این حالت اگر send برابر 1 باشد به معنی این است که می خواهیم دیتا را ارسال کنیم پس مقدار 8 بیت دیتا را در رجیستر r_Tx_Data ذخیره می کنیم و حالت s_TX_START_BIT را در sm_Main میریزیم به این معنی که می خواهیم شروع به خواندن دیتا کنیم. اگر sendبرابر 1 نباشد پس همین حالت idle را در Sm_Main میریزیم.

حال اگر مقدار Sm_main برابر s_TX_START_BIT باشد ابتدا tx را برابر 0 میکنیم (مطابق با آنچه در صورت پروژه گفته شده است) سپس چک می شود که اگر تعداد کلاک های گذشته از کلاک tx کمتر باشد 1 واحد به تعداد کلاک های سپری شده اضافه می کنیم و دوباره همین حالت s_TX_START_BIT را در Sm_Main میریزیم پس آنقدر این عمل ادامه پیدا می کند تا آنجا که به انتهای کلاک tx برسیم. سپس r_Clock_Count را صفر می کنیم و حالت s_TX_DATA_BITS را در sm_Main میریزیم.(در واقع به اندازه ی 1 کلاک tx در اینجا صبر می کنیم)

حال در حالت s_TX_DATA_BITS شروع به فرستادن دیتا به صورت سریال می کنیم ابتدا بیت صفرام را در tx می ریزیم سپس باید به اندازه ی 1 کلاک tx در همین حالت بماند که if در خط 71 به همین دلیل است. بعد از اینکه مدت زمان مورد نیاز گذشت دوباره clock_count را صفر میکنیم و باید چک کنیم که هر 8بیت فرستاده شده است(خط 81) پس بعد از اینکه هر 8 بیت فرستاده شد (و برای هر بیت هم به مقدار کلاک tx زمان گذشت) index را صفر می کنیم و به حالت stop_bit می رویم .

سپس بعد از فرستادن کامل دیتا در حالت s_TX_STOP_BIT دوباره tx را به همان 1 برمی گردانیم سپس در این قسمت مطابق با صورت پروژه باید به اندازه 1واحد زمانی tx صبر کنیم در نهایت tx_done را 1 میکنیم و به حالت IDLE می رویم.

ماژول RX

ورودی ها : clock, rx, reset

خروجی ها : rx_data, rx_finish

ورودی rx یک ورودی تک بیتی است که به صورت سریال دیتا را وارد rx می کند.

خروجی rx_data یک باس 8بیتی است که مقادیر ورودی را به صورت موازی خارج می کند

rx_finish نیز زمانی که دریافت دیتا به صورت کامل انجام شد یک می شود

روند کلی این ماژول نیز مانند tx است و چند state در نظر می گیریم و یکی یکی آن ها را بررسی می کنیم.

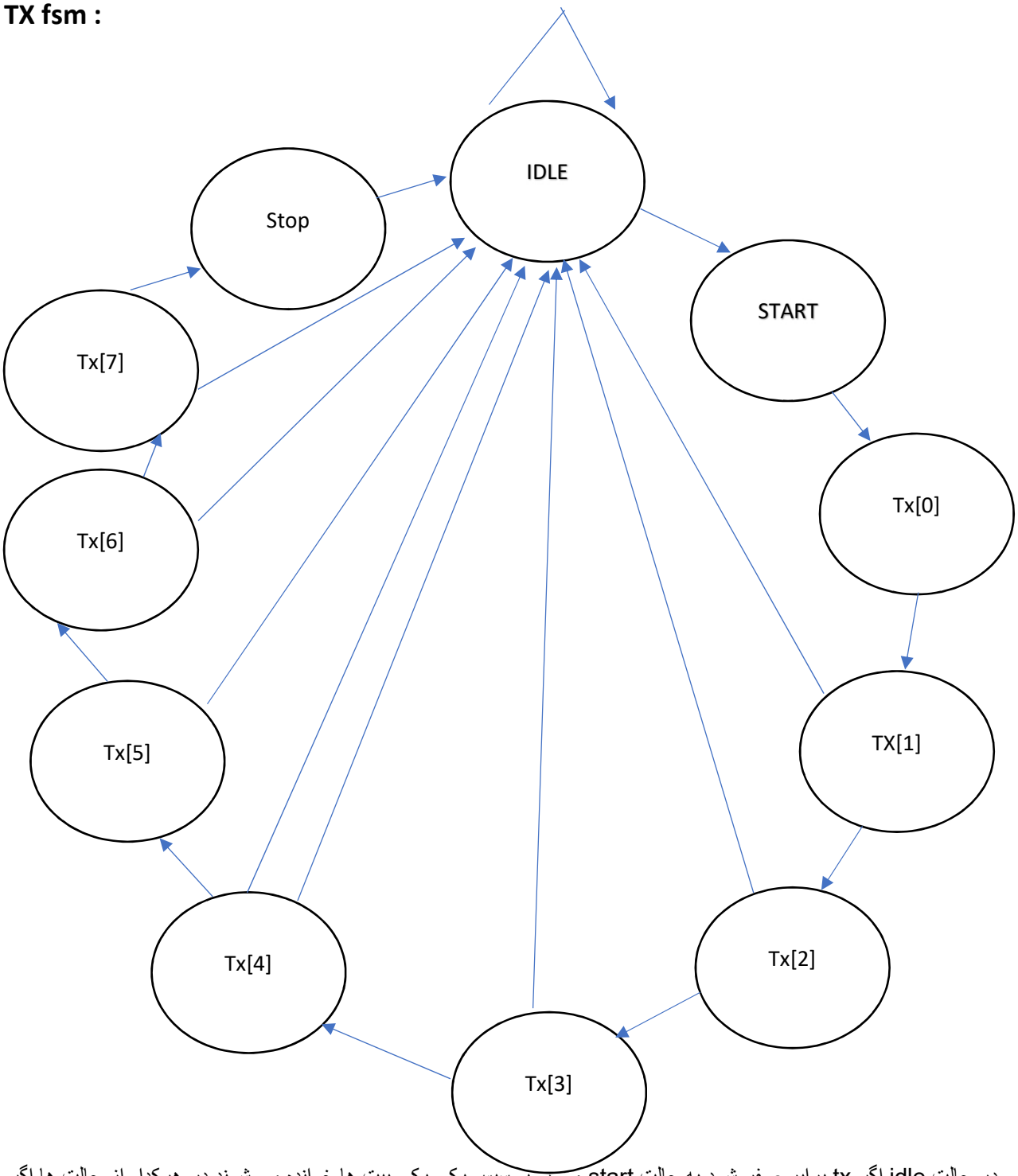
اگر sm_main برابر idle باشد clock_count و bit_index و rx-finish را برابر صفر قرار میدهیم و اگر rx_data برابر صفر شده بود گرفتن دیتا شروع می شود پس به حالت RX_START_BIT می رویم در غیر این صورت در همان حالت idle می مانیم.

سپس در حالت RX_START_BIT در ابتدا چک می شود که clock_count به زمان های sampling نرسیده باشد (clock_per_bit/2) و تا زمان سپری شدن و رسیدن به لحظه sampling در اینجا صبر می کنیم. و اگر Rx_Data برابر صفر بود یعنی می خواهیم شروع به دریافت کنیم پس clock_count را صفر کرده و به حالت start_bit می رویم و در غیر این صورت در همان idle می مانیم.

در حالت start_bit اگر clock_count به زمان sampling رسیده باشد وارد if می شویم در غیر اینصورت clock_count را 1واحد 1واحد اضافه می کنیم تا این اتفاق بیافتد سپس در if اگر rx_data برابر صفر شده بود (مطابق با صورت پروژه برای شروع دریافت دیتا باید ابتدا rx صفر شود) clock_count را ریست کرده و و به حالت RX_DATA_BITS می رویم.

درحالت RX_DATA_BITS اگر clock_count به sampling نرسیده باشد 1واحد 1 واحد به آن اضافه می کنیم تا به sampling برسد و در همین حالت می مانیم.سپس دوباره clock_count را ریست کرده و rx_data را در index مورد نظر می ریزیم سپس چک میشود که همه ی 8 بیت دریافت شده باشند و index را نیز در همین قسمت 1واحد 1واحد اضافه می کنیم.در نهایت به حالت stop_bit می رویم و در اینجا به اندازه 1 کلاک rx صبر می کنیم سپس rx_finish را برابر 1 و clock_count را ریست میکنیم و به حالت IDLE می رویم.

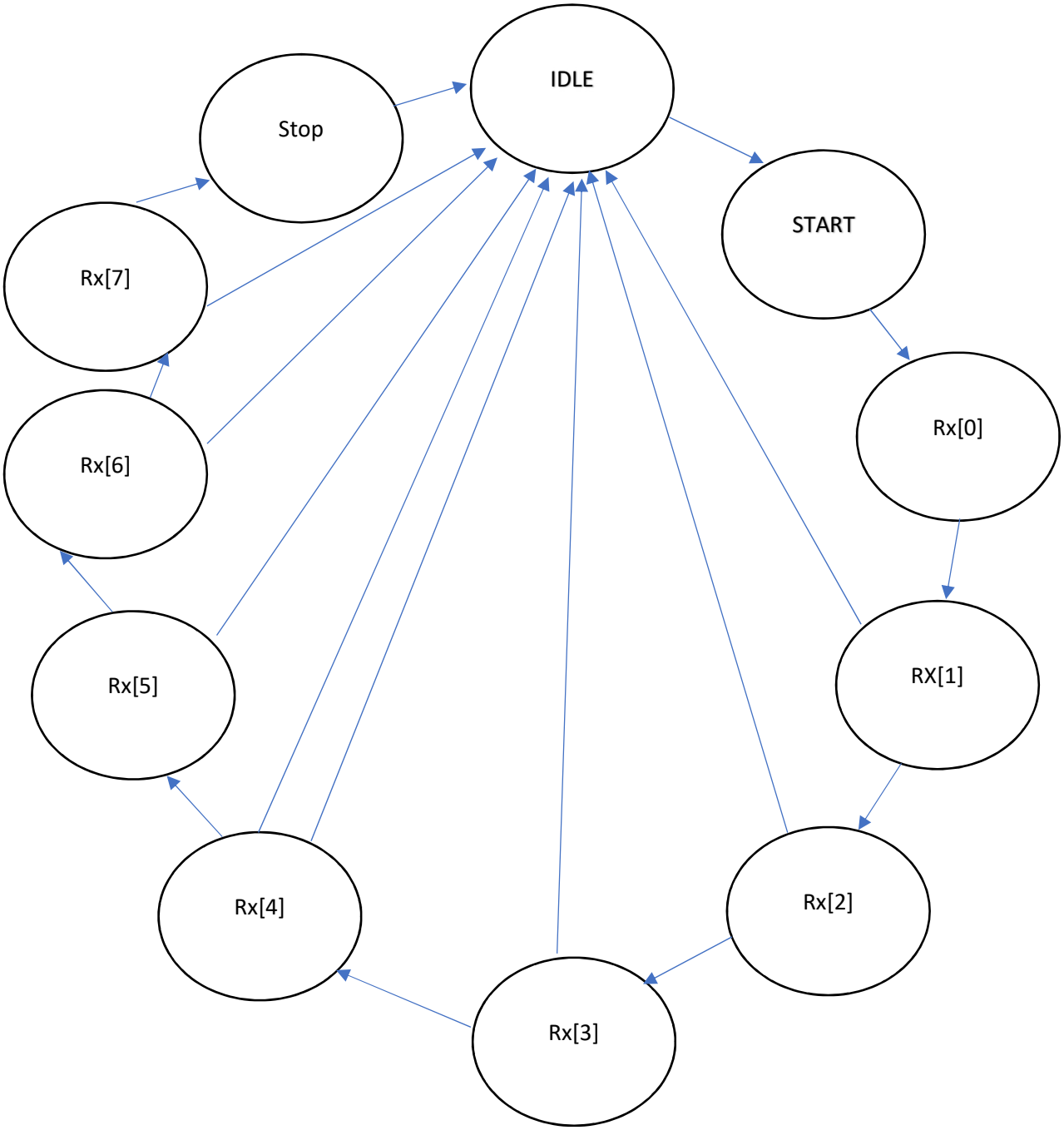
TX fsm :



در حالت idle اگر tx برابر صفر شود به حالت start می رویم سپس یکی یکی بیت ها خوانده می شوند در هرکدام از حالت ها اگر ریست برابر 1 شود به حالت idle برمی گردیم. و درهمان حالت idle نیز اگر tx 1 بماند در همان حالت idle می مانیم و در هر مرحله نیز به اندازه کلاک tx صبر میکنیم

.

RX fsm:



در حالت idle اگر rx برابر صفر شود به حالت start می رویم سپس یکی یکی بیت ها خوانده می شوند در هرکدام از حالت ها اگر ریست برابر 1 شود به حالت idle برمی گردیم. و درهمان حالت idle نیز اگر rx 1 بماند در همان حالت idle می مانیم و در هر مرحله نیز به اندازه کلاک rx صبر میکنیم

تابع tranceiver_tx

ورودی ها : clock, reset, rx_finish, rx_byte

خروجی ها : tx_byte, send

ابتدا توسط ماژول bit_String 22 بیت رابه 7 char تبدیل میکنیم. در این ماژول حالت های idle, get_string, send_data, wait وجود دارد.

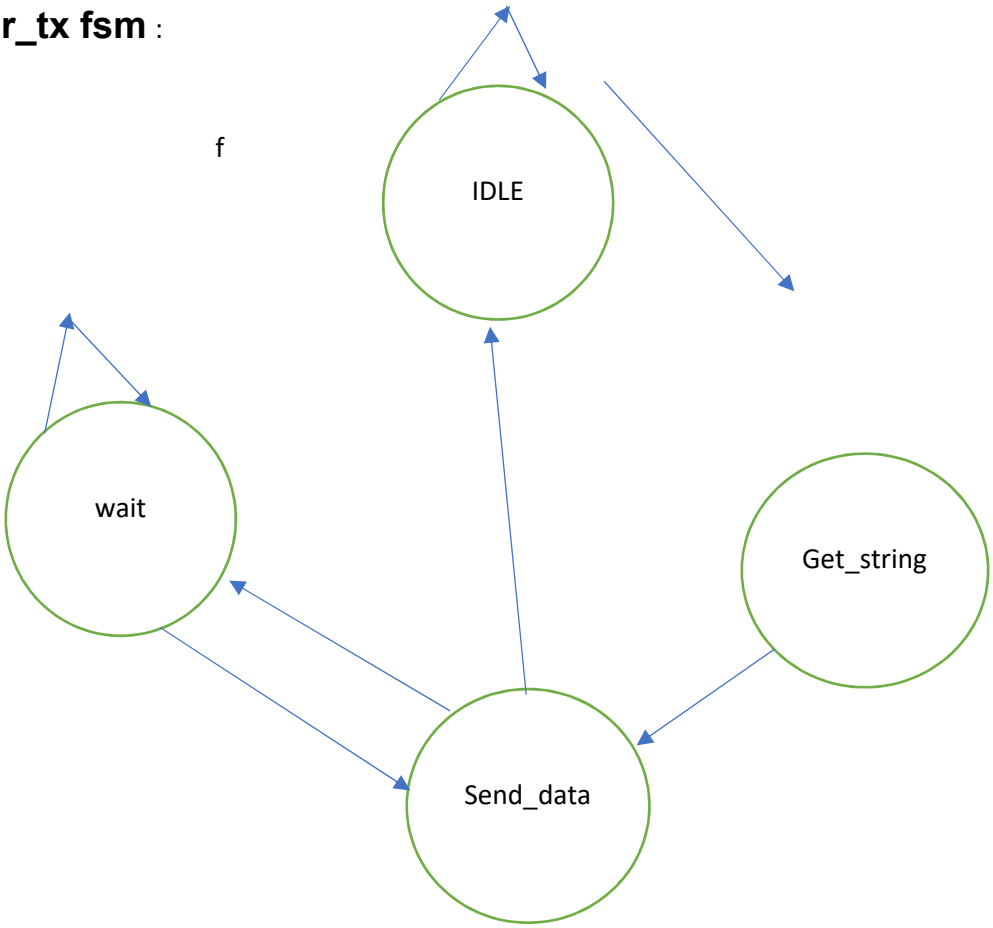
در حالت idle R_send را که به send در UART وصل است را صفر میکنیم و اگر start_transmit برابر 1 باشد move_in که 22بیت است را در r_move_in می ریزیم و acvtive که برای فعال کردن دو ماژول string است را 1 می کنیم و به حالت get_string می رویم. در غیر این صورت اگر start_transmit برابر 1 نباشد در همان idle می مانیم.

در حالت get_string string_out را که خروجی ماژول تبدیل کننده ی 22 بیت به char را در r_tx_out می ریزیم و سپس active را صفر کرده و به حالت send_data می رویم.

در حالت send_data در ابتدا چک می کنیم که که index کمتر از 8 باشد سپس اگر هر بایت مخالف صفر بود آن 8 بیت از tx_out را درtx_byte می ریزیم سپس sendرا برابر 1 می کنیم و به حالت wait می رویم و 1 واحد هم به index اضافه می کنیم.

در حالت wait send را صفر کرده و چک می کنیم که اگر سیگنال tx_done از ماژول tx برابر 1 شده است یا نه و در نهایت تا زمانی که 8 بیت را UART ارسال کند صبر می کند.

Tranceiver_tx fsm :



از حالت idle اگر start_transmit = 1 باشد به get_string می رویم از get_string به send_data می رویم از send_data اگر index بزرگتر از 8 باشد به idle بر می گردیم و اگر tx_out[index]!= 0 باشد به wait می رویم.در حالت wait اکر tx_done برابر 1 باشد به send_data برمی گردیم و در غیر این صورت در همان wait می مانیم

ماژول receiver_rx

ورودی ها : clock, reset, rx_finish, rx_byte

خروجی ها : move_out, color, end_receive

در این ماژول حالت های idle, receive_data, set_color, stop_receive, end وجود دارند.

ابتدا توسط ماژول string string_bit داده شده را به 22 بیت تبدیل میکنیم.سپس در حالت idle end_receive را برابر0 کرده و اگر rx_finish(که به UART rx_finish وصل است و هر وقت 1 بایت بگیرد 1 میشود) برابر 1 باشد به حالت receive_data می رویم در غیر این صورت در همان حالت idle می مانیم.

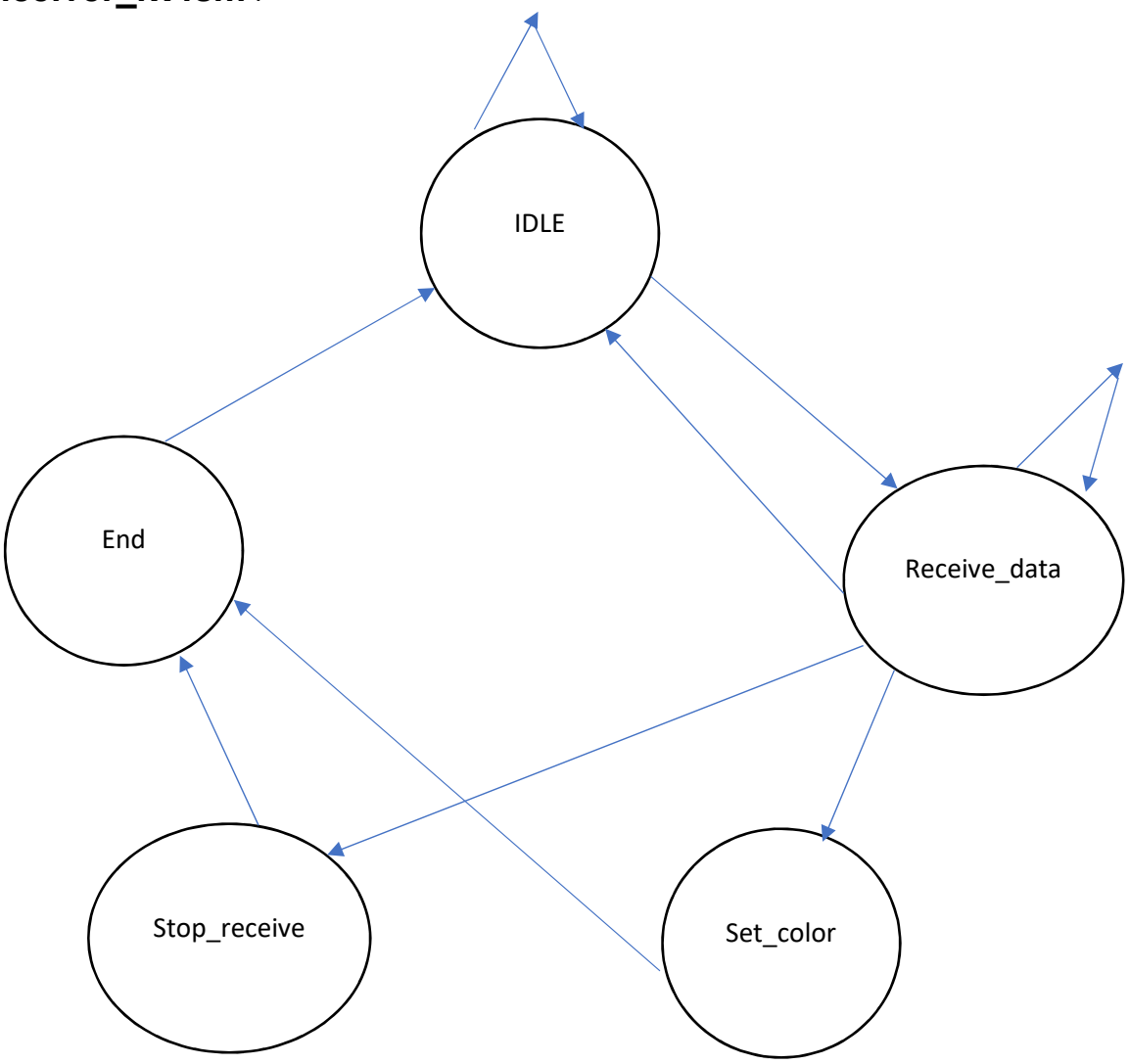
در حالت receive_data rx_byte که به UART rx_data وصل است 8 بیتی است و 8بیت 8بیت input_data را که یک رجیستر 56 بیتی است را پر می کند.متغیر number برای شمردن تعداد حروف استفاده می شود که در ماژول تبدیل string به 22 بیت استفاده می شود.سپس اگر بایت index از input_data برابر n\ باشد (به معنای زدن enter) و بیت 7تا 0 شامل -" " باشد به set_color می رویم و اگر برابر با "-" نباشد activeرا برابر 1 کرده و به stop_receiveمی رویم. در غیر این صورا به حالت idle باز می گردیم .

در حالت set_color اگر بیت 15 تا 8 برابر w باشد رنگ سیاه و عدد 0 را به آن نسبت می دهیم در غیر اینصورت رنگ سفید و عدد 1.سپس end_receive را 1 کرده و به حالت end می رویم.

در حالت stop ثدی قثزثهرث را 1 می کنیم و به endم یرویم.

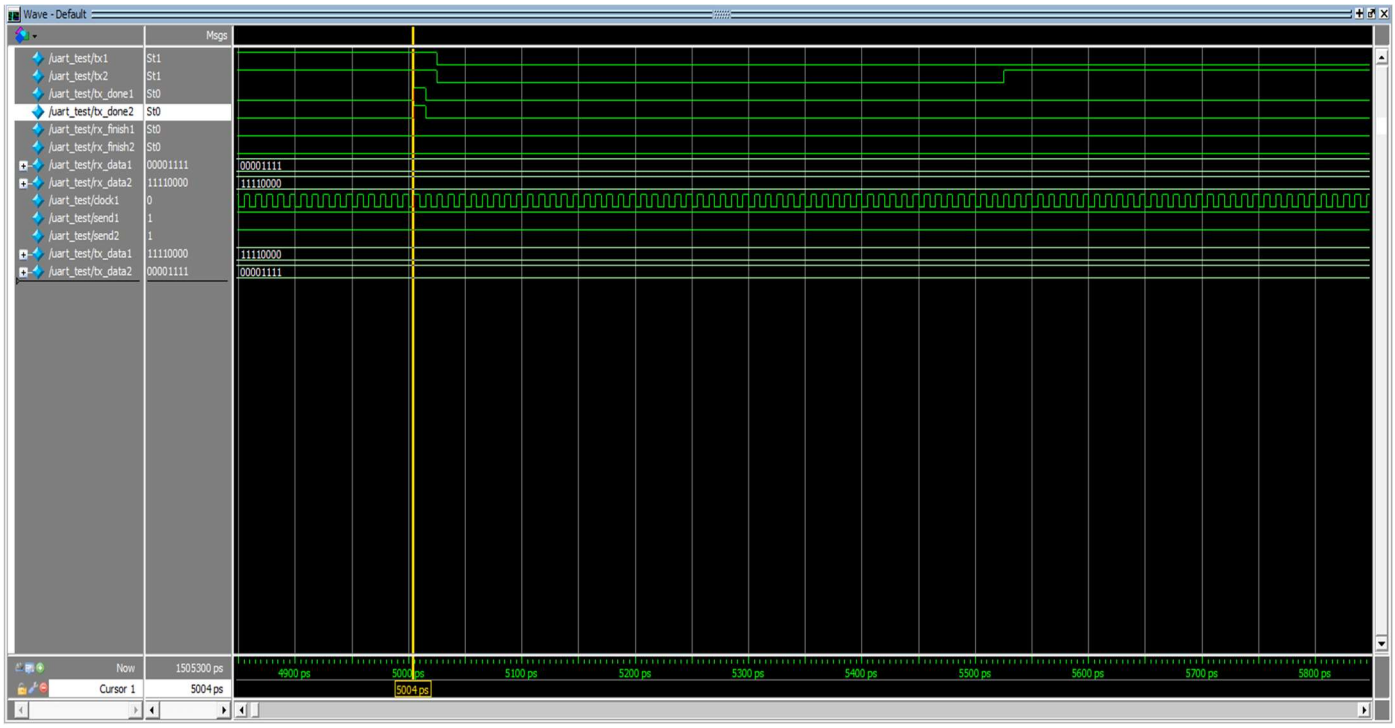
در حالت end number و active را صفر می کنیم و به حالت idle بر می گردیم.

Tranceiver_rx fsm :



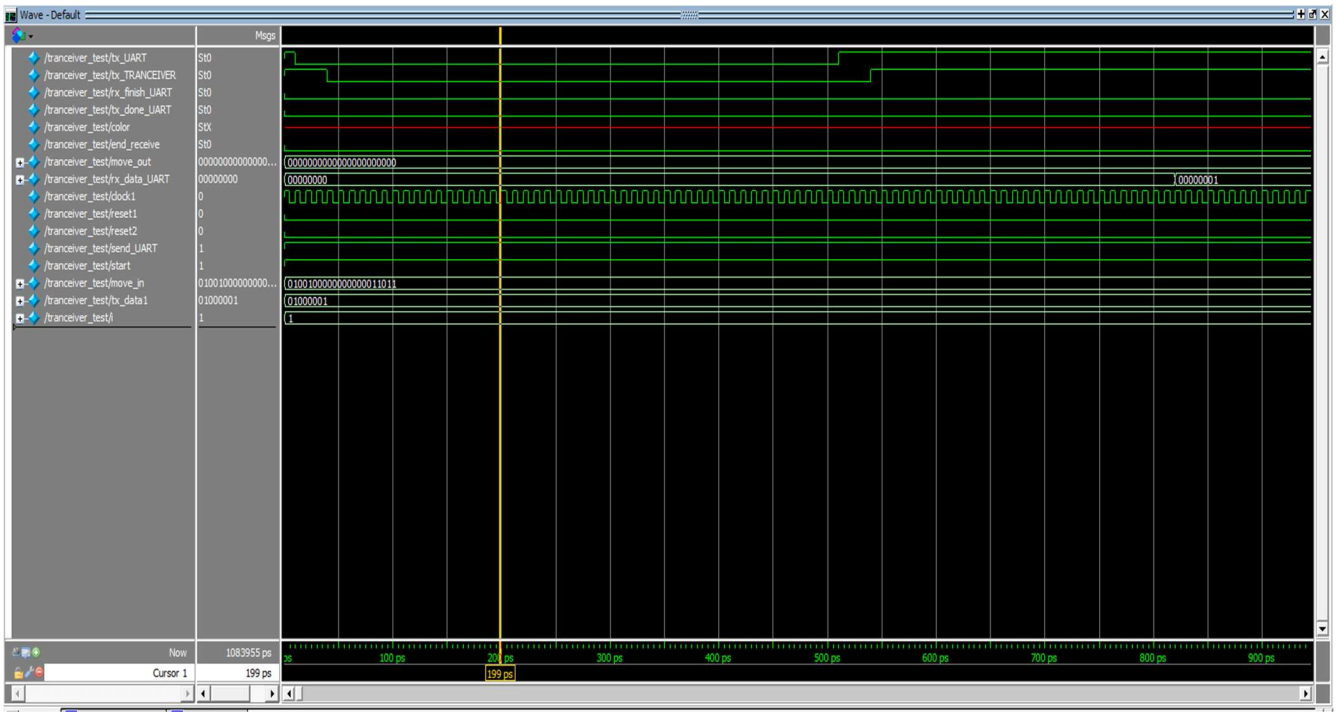
از حالت IDLE شروع می کنیم اگر rx_finish برابر 1 شود به حالت receive_data می رویم در غیر اینصورت در همان idle می مانیم. در حالت receive_data اگر خانه ی مربوط به index برابر با \n و بیت 7تا 0 برابر – باشد به set color می رود اگر بیت 7تا 0 برابر – نباشد به stop_receive می رود و اگر بیت index برابر \n نباشد در همان receive_data می ماند.در حالت set_colorهمیشه به حالت end می رویم و در حالت stop_receive نیز همیشه به حالت end می رویم. و از حالت end همیشه به idle می رویم.

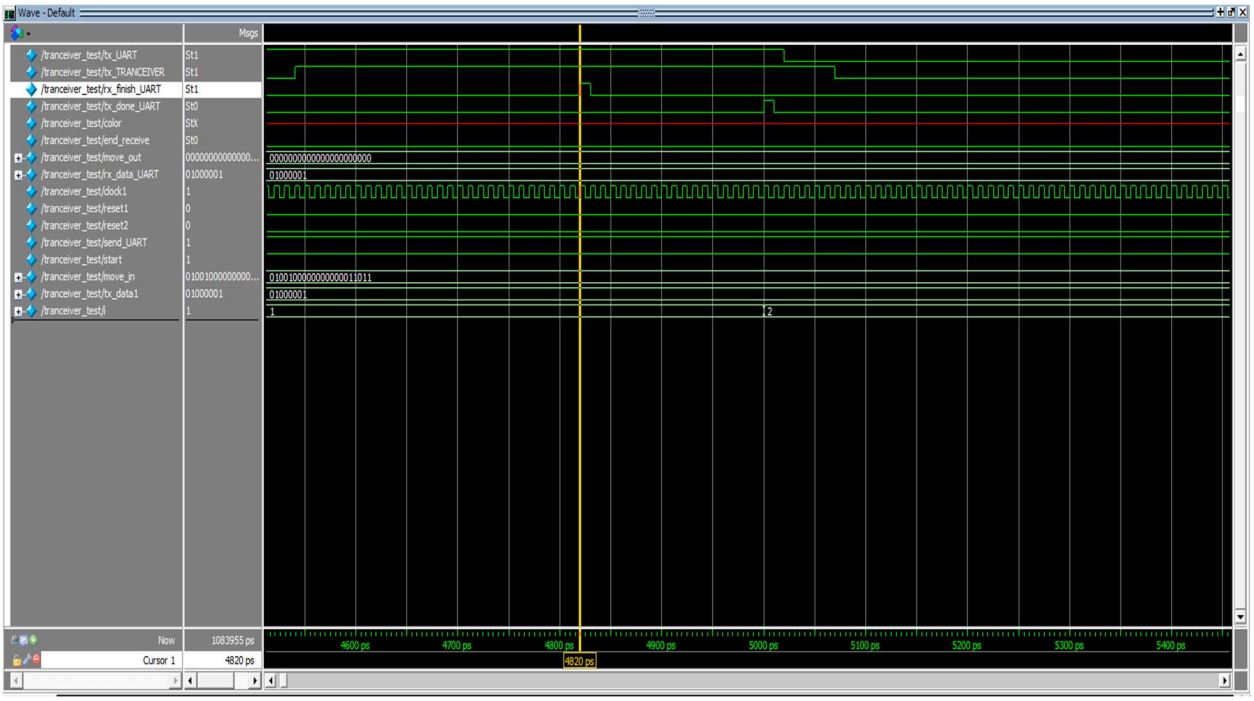
در ادامه شکل موج های مربوطه را می توان مشاهده کرد:

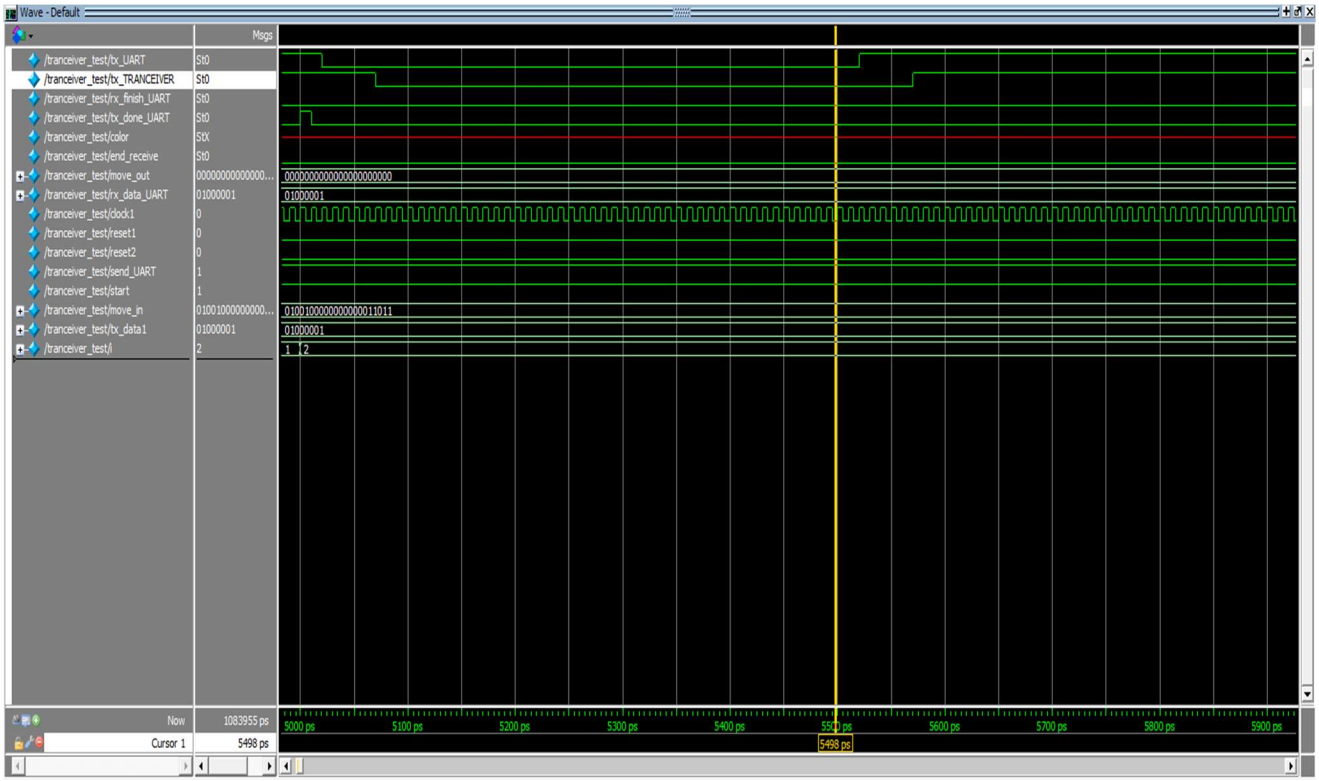


تصویر اول مربوط UART است

تصاویر بعدی مربوط به tranceiver است که زمان های مختلف موج را نشان می دهد


Wave - Default
Msgs
/tranceiver_test/tx_UART
/tranceiver_test/tx_TRANCEIVER
/tranceiver_test/rx_finish_UART
/tranceiver_test/tx_done_UART
/tranceiver_test/color
/tranceiver_test/end_receive
/tranceiver_test/move_out
/tranceiver_test/rx_data_UART
/tranceiver_test/clock1
/tranceiver_test/reset1
/tranceiver_test/reset2
/tranceiver_test/send_UART
/tranceiver_test/start
/tranceiver_test/move_in
/tranceiver_test/tx_data1
/tranceiver_test/i
St0
St0
St0
St0
StX
St0
00000000000000000000
00000000
0
0
0
1
1
01001000000000011011
01000001
1
00000000
00000001
Now
1083955 ps
Cursor 1
199 ps
100 ps
200 ps
300 ps
400 ps
500 ps
600 ps
700 ps
800 ps
900 ps
199 ps


Wave - Default
Msgs
/tranceiver_test/tx_UART
/tranceiver_test/tx_TRANCEIVER
/tranceiver_test/rx_finish_UART
/tranceiver_test/tx_done_UART
/tranceiver_test/color
/tranceiver_test/end_receive
/tranceiver_test/move_out
/tranceiver_test/rx_data_UART
/tranceiver_test/clock1
/tranceiver_test/reset1
/tranceiver_test/reset2
/tranceiver_test/send_UART
/tranceiver_test/start
/tranceiver_test/move_in
/tranceiver_test/tx_data1
/tranceiver_test/i
St1
St1
St1
St0
StX
St0
00000000000000000000
01000001
1
0
0
1
1
01001000000000011011
01000001
1
2
Now
1083955 ps
Cursor 1
4820 ps
4600 ps
4700 ps
4800 ps
4900 ps
5000 ps
5100 ps
5200 ps
5300 ps
5400 ps
4820 ps


Wave - Default
Msgs
/tranceiver_test/tx_UART
/tranceiver_test/tx_TRANCEIVER
/tranceiver_test/rx_finish_UART
/tranceiver_test/tx_done_UART
/tranceiver_test/color
/tranceiver_test/end_receive
/tranceiver_test/move_out
/tranceiver_test/rx_data_UART
/tranceiver_test/clock1
/tranceiver_test/reset1
/tranceiver_test/reset2
/tranceiver_test/send_UART
/tranceiver_test/start
/tranceiver_test/move_in
/tranceiver_test/tx_data1
/tranceiver_test/i
St0
St0
St0
St0
StX
St0
00000000000000000000
01000001
0
0
0
1
1
01001000000000011011
01000001
2
01000001
1 2
Now
1083955 ps
Cursor 1
5498 ps
5000 ps
5100 ps
5200 ps
5300 ps
5400 ps
5500 ps
5600 ps
5700 ps
5800 ps
5900 ps
5498 ps