

SoC UART Peripheral

프로토콜 & TX & FIFO

You are free to fork or clone this material. See [LICENSE.md](<https://github.com/arm-university/Introduction-to-SoC-Design-Education-Kit/blob/main/License/LICENSE.md>) for the complete license.

Agenda

1. Serial and Parallel 통신의 원리
2. UART 통신의 개념 및 구조
 - UART 통신 개념 이해
 - UART 통신 시스템 이해
 - UART 프로토콜 이해
 - UART 모듈 내부 블록도 , baud rate 및 비트별 설정, tx/rx제어신호, 주요 레지스터맵
 - UART통신을 위한 HW 구조 및 시스템 이해
 - 주변 기기와 통신을 위한 연결 방법 및 구성방법
 - UART TX Module Testbench를 활용한 timing 시뮬레이션
3. Q&A

수업 목표

- 이 강의가 끝나면, 여러분은 다음을 할 수 있어야 합니다:
 - 직렬 통신과 병렬 통신의 개념을 설명하고, 각각의 응용 사례를 제시할 수 있다.
 - 동기식 직렬 통신과 비동기식 직렬 통신의 차이점을 설명할 수 있다.
 - 직렬 통신과 병렬 통신을 비교하고, 각각의 장단점을 설명할 수 있다.
 - UART와 그 통신 프로토콜에 대해 설명할 수 있다.
 - AHB UART 주변 장치의 구성 요소를 설명하고, 각 구성 요소의 기능을 기술할 수 있다

직렬 통신과 병렬 통신의 원리

- 직렬 통신 (Serial Communication)
 - 직렬 통신은 데이터를 한 번에 한 비트씩 순차적으로 전송하는 방식입니다. 이는 단일 채널 또는 라인을 사용하며, 송신 장치가 데이터를 하나씩 보내고 수신 장치가 이를 재조립하여 메시지를 완성
- 병렬 통신 (Parallel Communication)
 - 병렬 통신은 여러 비트를 동시에 전송하는 방식으로, 각 비트가 별도의 채널을 통해 전달

Serial Communication

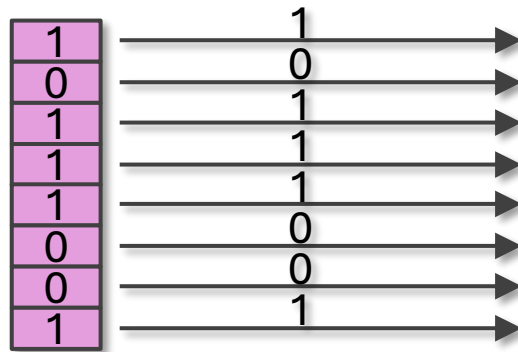
- Serial Communication
 - 데이터를 한 번에 한 비트씩 순차적으로 전송하는 방식
주로 장거리 통신, 모뎀, 그리고 네트워크가 아닌 장치 간 통신에 사용됨
예시로는 UART, SPI, I2C, USB, Ethernet, PCI Express 등이 있음



Serial communication

Parallel Communication

- 여러 비트가 동시에 전송
- Serial 방식은 데이터를 한 번에 한 비트씩 순차적으로 전송
- Parallel 전송은 일반적으로 동기식.
 - 예로는 Arm AHB와 같은 칩 내부 버스가 있습니다

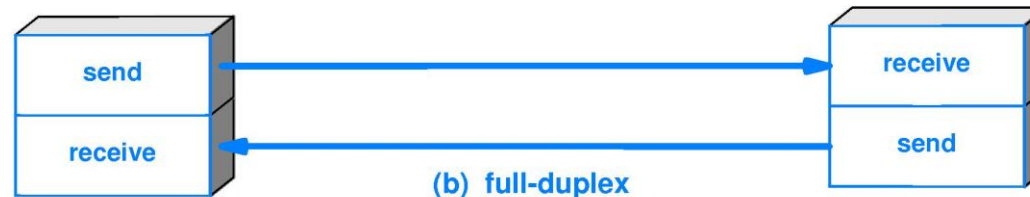


Parallel communication

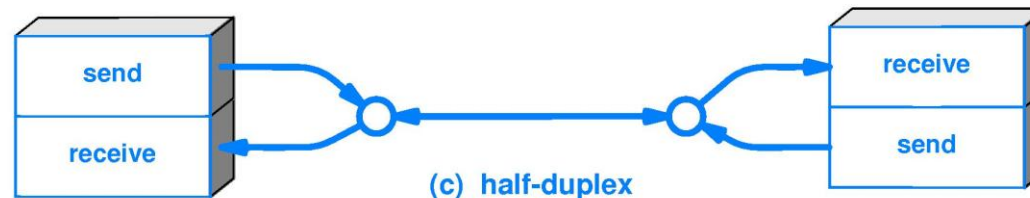
Mode of Serial Communication



(a) simplex



(b) full-duplex

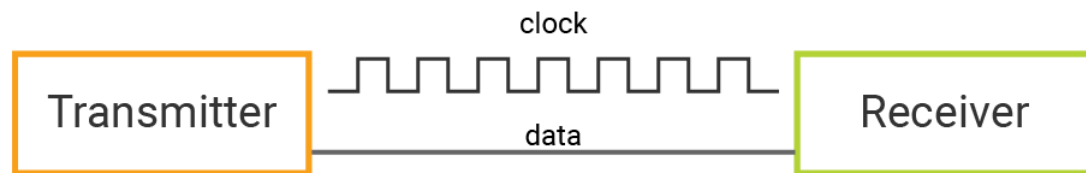


(c) half-duplex

Types of Serial Communication

동기식 serial transmission

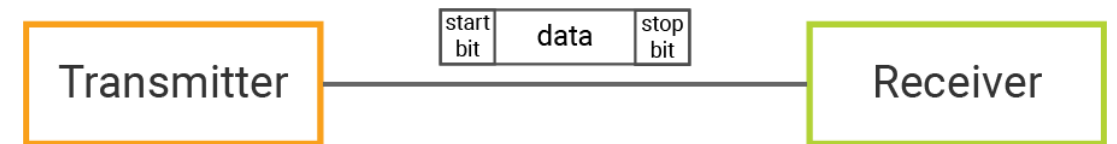
- 송신자와 수신자가 공통 클록을 공유함. 데이터 전송 전용으로 하나의 와이어가 사용되기 때문에 더 효율적인 전송이 가능함. 추가 클록 와이어가 필요하기 때문에 비용이 더 많이 듦.



Synchronous communication

비동기식 serial transmission

- 송신자는 클록 신호를 전송할 필요가 없음. 송신자와 수신자가 미리 타이밍 파라미터에 대해 합의함. 전송을 동기화하기 위해 특별한 비트가 추가됨.



Asynchronous communication

Serial vs Parallel Communication

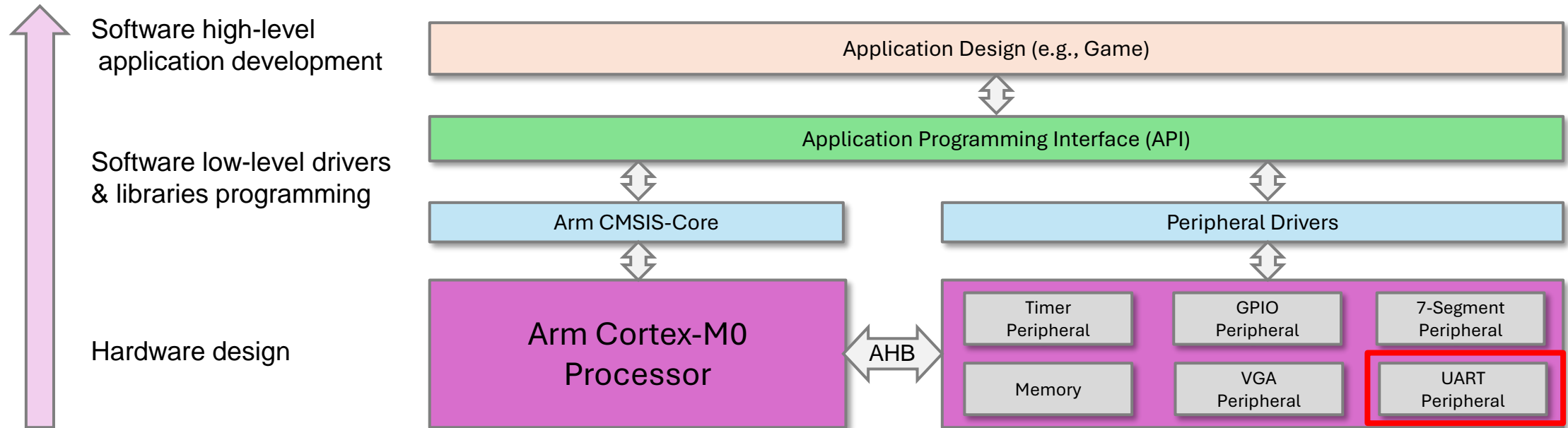
Serial

특성	직렬 통신
데이터 전송 방식	한 번에 한 비트씩 순차적으로
Wire Cost	적음 (1~4개)
거리	장거리 전송에 적합
간섭	간섭이 적음
동기화	프로토콜 등 다양한 방법이 필요함
전송속도	제한된 전송속도

Parallel

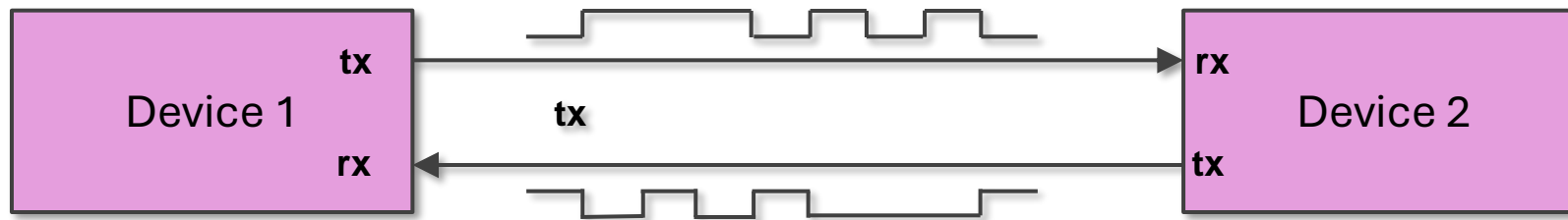
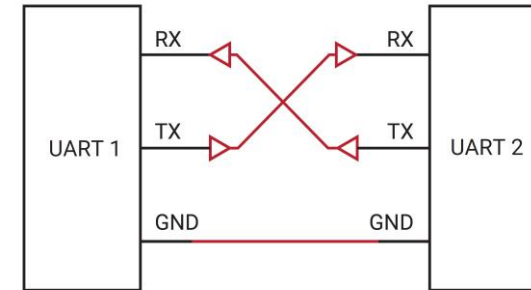
병렬 통신
여러 비트를 동시에
높음 (8개 이상)
짧은 거리에서만 효과적
간섭 가능성이 높음
복잡함
높은 전송속도

Building a System on a Chip (SoC)



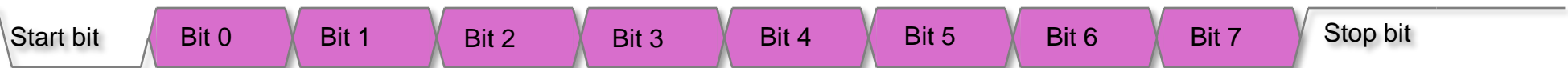
UART Overview

- UART
 - 비동기식 통신, clock 와이어가 필요 없음, 미리 합의된 baud rate 사용
 - 별도의 송신(TX) 및 수신(RX) 와이어 사용
- UART communication
 - 데이터를 병렬에서 직렬로 변환
 - 직렬 케이블을 통해 순차적으로 데이터 전송
 - 순차적으로 받은 데이터를 다시 병렬로 재조립



UART Protocol

- 데이터 전송은 시작 비트로 시작되며, 논리 신호를 한 클록 사이클 동안 낮게 유지.
- 다음 여덟 클록 사이클 동안 송신기로부터 8비트가 순차적으로 전송.
- 선택적으로, 전송 신뢰성을 향상시키기 위해 하나의 패리티 비트를 추가 가능.
- 전송이 완료되었음을 나타내기 위해 마지막에 데이터 와이어를 높은 상태로 유지.



Transfer one byte without parity bit



Transfer one byte with parity bit

UART Protocol - 패리티 비트 (Parity Bit)

- UART Protocol - 패리티 비트 (Parity Bit)

- 패리티 비트는 UART 통신에서 오류 검출을 위해 사용되는 선택적인 비트
 - 데이터 전송 중에 발생할 수 있는 단일 비트 오류를 감지하는 데 도움을 줍니다. 송신자는 패리티 비트를 데이터 프레임에 추가하고, 수신자는 이를 통해 데이터의 무결성을 확인

- 짝수 패리티 (Even Parity)

- 데이터 비트 내 '1'의 개수를 짝수로 만들기 위해 패리티 비트를 설정.
- 예: 데이터가 1011(1이 3개)라면, 패리티 비트를 1로 추가하여 '1'의 개수를 짝수로 만듦.

짝수 패리티 (Even Parity)의 예

START	D0	D1	D2	D3	D4	D5	D6	D7	PARITY (Even)	STOP
0	1	0	1	0	0	1	0	0	1	1

- 홀수 패리티 (Odd Parity)

- 데이터 비트 내 '1'의 개수를 홀수로 만들기 위해 패리티 비트를 설정.
- 예: 데이터가 1010(1이 2개)라면, 패리티 비트를 1로 추가하여 '1'의 개수를 홀수로 만듦.

홀수 패리티 (Odd Parity)의 예

START	D0	D1	D2	D3	D4	D5	D6	D7	PARITY (Odd)	STOP
0	1	0	1	0	0	1	0	0	0	1

- 없음 (No Parity)

- 패리티 비트를 사용하지 않습니다. 이 경우, 데이터 프레임은 START, DATA, STOP 비트만으로 구성.

UART Protocol 주요 설정 사항

- UART Protocol

- Baudrate

- Baud Rate는 초당 전송되는 신호 변화의 횟수를 나타내며, 데이터 전송 속도를 결정

- Data Bits

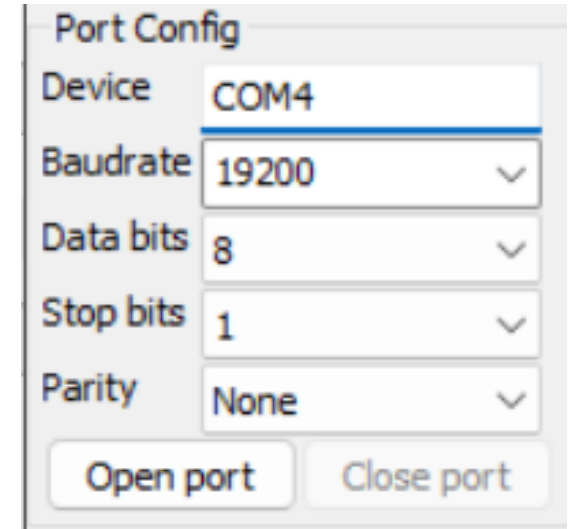
- 데이터 프레임 내에서 전송되는 데이터 비트의 개수를 나타냄.

- Stop Bits

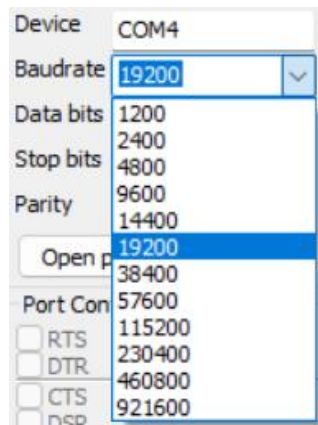
- 데이터 프레임의 끝을 나타내는 비트로, 수신기가 데이터 프레임의 종료를 인식

- Parity (None)

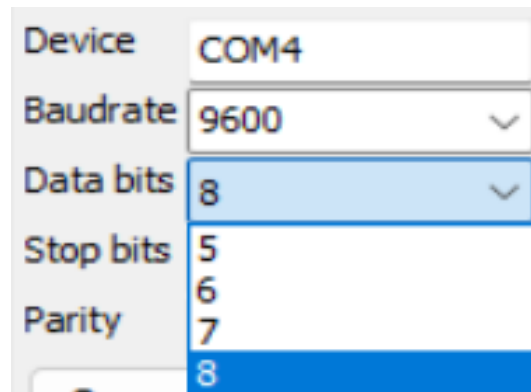
- 패리티 비트는 오류 검출을 위해 추가되는 선택적 비트



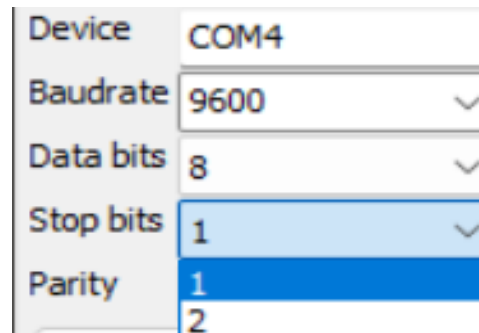
Port Config	
Device	COM4
Baudrate	19200
Data bits	8
Stop bits	1
Parity	None
<input type="button" value="Open port"/> <input type="button" value="Close port"/>	



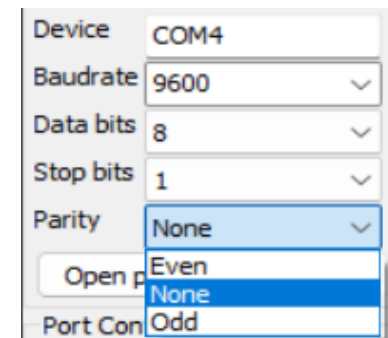
Device	COM4
Baudrate	19200
Data bits	1200
Stop bits	2400
Parity	4800
	9600
	14400
	19200
	38400
	57600
	115200
	230400
	460800
	921600



Device	COM4
Baudrate	9600
Data bits	8
Stop bits	5
	6
	7
	8



Device	COM4
Baudrate	9600
Data bits	8
Stop bits	1
Parity	1
	2



Device	COM4
Baudrate	9600
Data bits	8
Stop bits	1
Parity	None
	Even
	None
	Odd

Character-Encoding Scheme

- 문자는 American Standard Code for Information Interchange로 인코딩 됩니다. (ASCII)
 - 128개의 문자를 인코딩
 - 95개의 출력 가능한 문자(예: "a", "b", "1", "2")
 - 33개의 출력 불가능한 제어 문자(예: 줄 바꿈, 백스페이스, 이스케이프)
 - 7비트로 표현 가능하며, 저장 편의를 위해 일반적으로 1바이트로 저장됨
- UTF-8 (UCS Transformation Format—8-bit)
 - 2007년부터 ASCII에서 파생됨
 - 가변 폭 인코딩 방식
 - 월드 와이드 웹에서 널리 사용됨
 - 원래의 ASCII와 호환됨

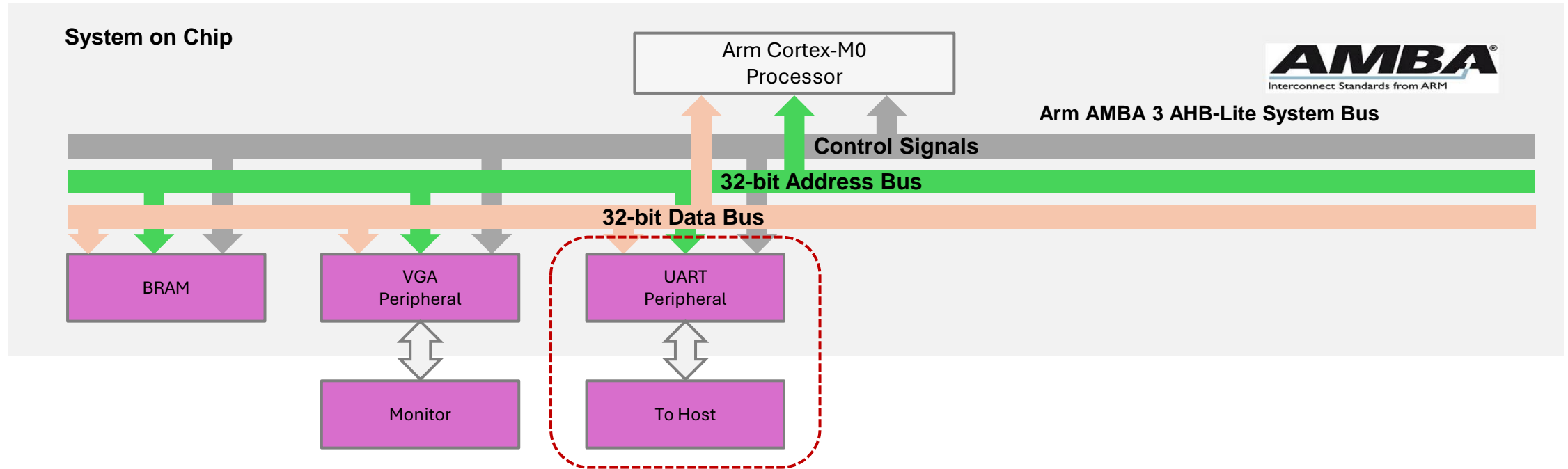
ASCII Encoded Characters

- 아래 표는 ASCII로 인코딩된 자주 사용되는 문자들을 나열한 것입니다.
- <https://en.wikipedia.org/wiki/ASCII>

Hex	Character		Hex	Character		Hex	Character
0x30	0		0x41	A		0x61	a
0x31	1		0x42	B		0x62	b
0x32	2		0x43	C		0x63	c
0x33	3		0x44	D		0x64	d
0x34	4		0x45	E		0x65	e
0x35	5		0x46	F		0x66	f
0x36	6		0x47	G		0x67	g
0x37	7		0x48	H		0x68	h
0x38	8		0x49	I		0x69	i
0x39	9		0x4A	J		0x6A	J
...			

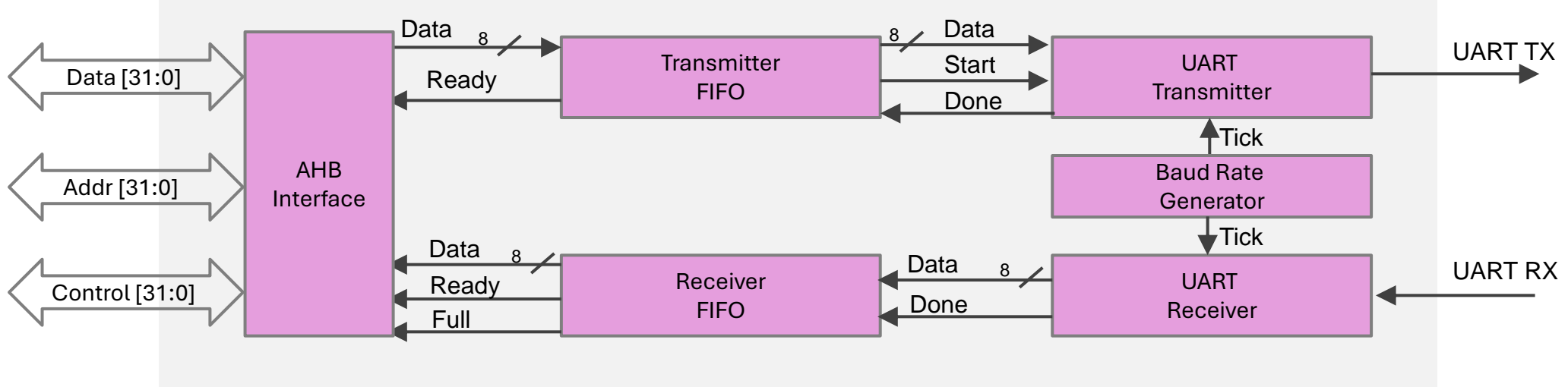
AHB UART Peripheral

Interfacing with UART using AHB3-Lite bus



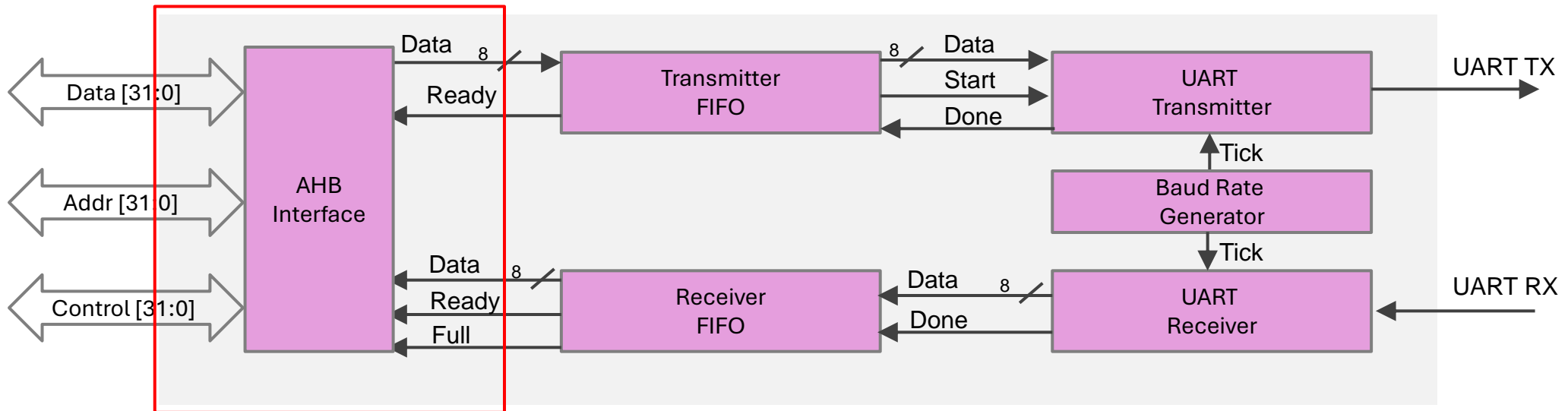
AHB UART Peripheral

- 우리 설계에서 UART 주변 장치는 다음으로 구성됩니다:
 - UART 송신기와 수신기
 - 송신 FIFO와 수신 FIFO
 - Baud rate 생성기
 - AHB 버스 인터페이스



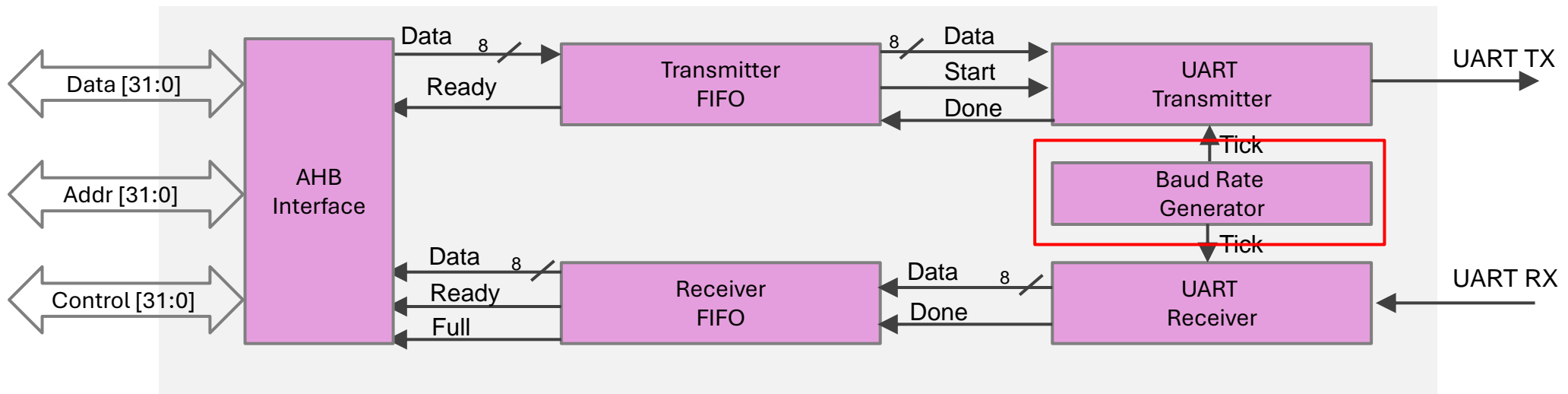
AHB UART Peripheral

- 우리 설계에서 UART 주변 장치는 다음으로 구성됩니다
 - UART 송신기와 수신기
 - 송신 FIFO와 수신 FIFO
 - Baud rate 생성기
 - AHB 버스 인터페이스



Baud Rate Generator

- Baud rate generator
 - 고정된 전송 baud rate(예: 19200bps)에 맞춰 시스템 틱을 생성하여 수신 FIFO로 바이트를 전달.



Baud Rate Generator

- UART에서 Baud 클록의 주파수를 Baud rate의 16배로 사용하는 이유
 - **정확한 샘플링 및 동기화**
 - UART는 비동기 통신 방식으로, 송신자와 수신자가 별도의 클록 신호를 공유하지 않음. 따라서, 데이터를 정확히 해석하려면 START 비트와 데이터 비트를 올바르게 샘플링해야 함.
 - Baud 클록이 Baud rate의 16배인 경우, 각 비트를 16번 샘플링. 이를 통해 START 비트의 중심을 정확히 감지하고, 이후 데이터 비트도 올바르게 해석할 수 있음.
 - **오류 감소**
 - START 비트의 정확한 감지를 위해 16× 오버 샘플링을 사용하면, 클록 드리프트나 잡음으로 인한 오류를 줄일 수 있음.
 - START 비트의 하강 에지를 감지한 후, 중심점을 기준으로 데이터 비트를 안정적으로 샘플링 할 수 있음.
 - **비트 중심점 계산**
 - UART 수신시 각 데이터 비트는 16개의 Baud 클록 사이클 동안 지속 됨. UART는 이 중간 지점(8번째 클록)을 기준으로 데이터를 샘플링 하여 수신함.
 - 이렇게 하면 신호 왜곡이나 타이밍 오류를 최소화할 수 있음.
 - **구현의 용이성**
 - 대부분의 마이크로 컨트롤러와 UART 하드웨어는 내부적으로 Baud rate 생성기를 사용하여 시스템 클록을 분주(divide)해 16× Baud 클록을 생성합니다. 이는 UART 설계에서 표준적인 방식으로 자리 잡았습니다

실습1 – Baud Rate Generate – baudgen.v

- 코드를 완성 하시오
- Baud Rate(bps) =
 - (system clock/ COUNT_N)
- COUNT_N =
 - System clock/19200
 - COUNT_N = 2604
- 1 → 16x, COUNT_N= 162.75
 - (system clock/ (Baud rate*16))

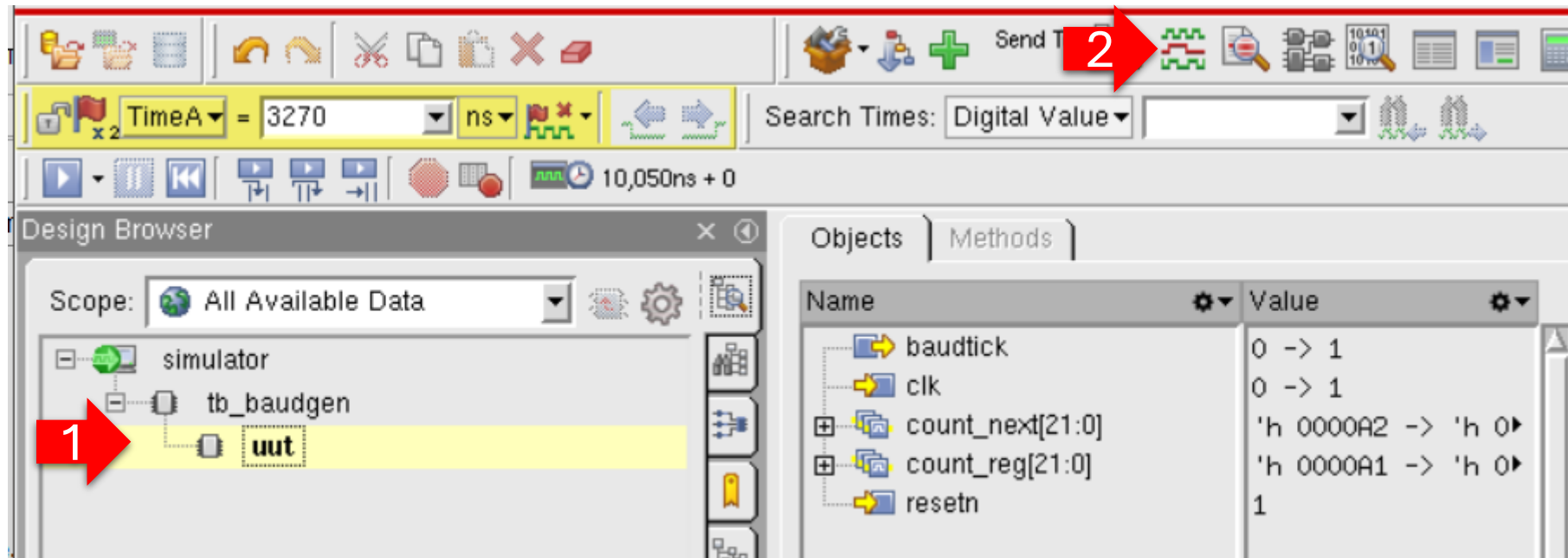
```
37 module BAUDGEN
38 (
39     input wire clk,
40     input wire resetn,
41     output wire baudtick
42 );
43
44
45 reg [21:0] count_reg;
46 wire [21:0] count_next;
47
48 //Counter
49 always @ (posedge clk, negedge resetn)
50 begin
51     if(!resetn)
52         count_reg <= 0;
53     else
54         count_reg <= count_next;
55 end
56
57
58 //Baudrate = 19200 = 50Mhz/(163*16)
59 assign count_next = ((count_reg == 162) ? 0 : count_reg + 1'b1);
60
61 assign baudtick = ((count_reg == 162) ? 1'b1 : 1'b0);
62
63 endmodule
```

실습1 – baudrate 시뮬레이션, tb_baudrate.v

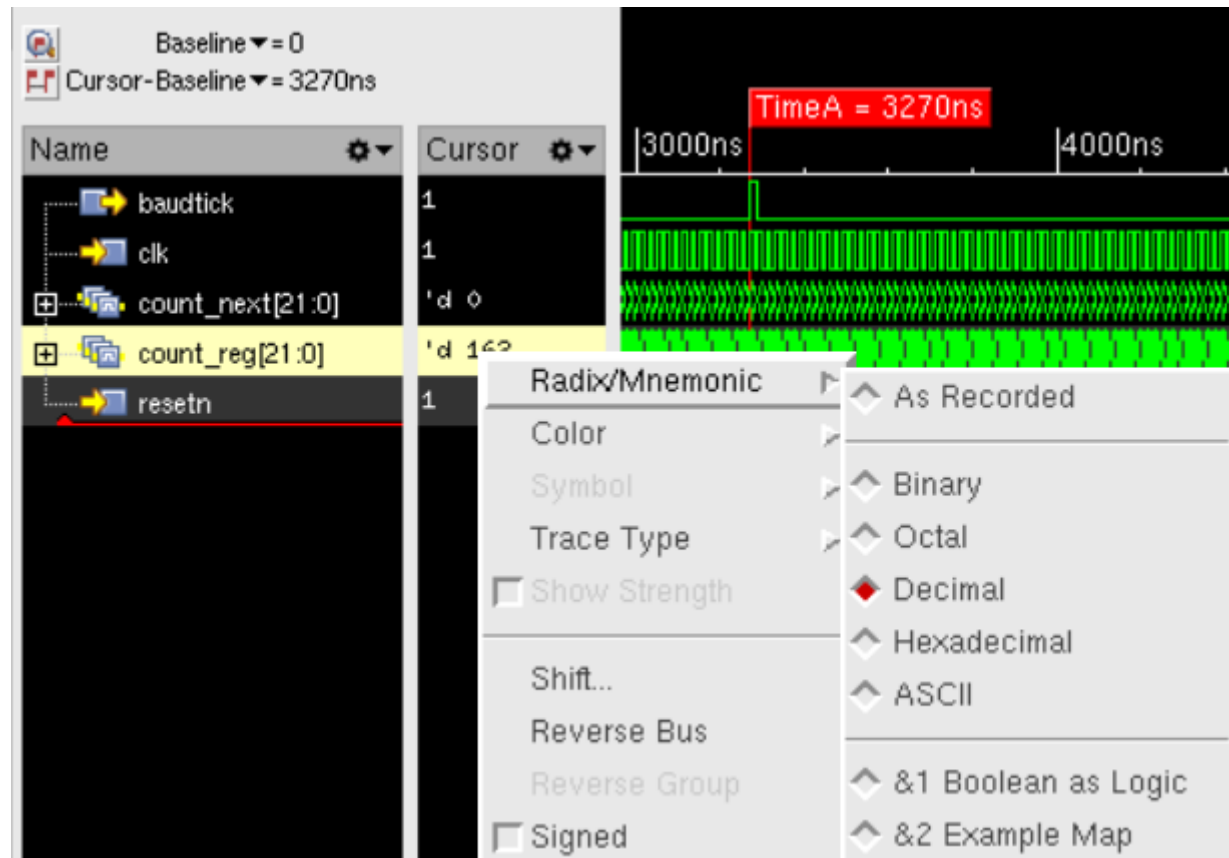
- 시뮬레이션 테스트 벤치 코드를 완성 하시오

```
1  module tb_baudgen();
2
3      // Testbench 신호 정의
4      reg clk;          // 시스템 클록
5      reg resetn;       // 비동기 리셋 (Active Low)
6      wire baudtick;    // Baud Rate Tick 출력
7
8      // DUT (Device Under Test) 인스턴스화
9      BAUDGEN uut (
10         .clk(clk),
11         .resetn(resetn),
12         .baudtick(baudtick)
13     );
14
15     // 클록 생성
16     initial begin
17         clk = 0;
18         forever #10 clk = ~clk; // 주기: 20ns (50MHz 클록)
19     end
20
21     // 리셋 및 테스트 시퀀스
22     initial begin
23         // 초기화
24         resetn = 0;
25         #50 resetn = 1; // 리셋 해제
26
27         // 시뮬레이션 실행
28         #1000 $finish; // 시뮬레이션 종료
29     end
30
31     // Baudtick 모니터링
32     initial begin
33         $monitor("Time: %t | baudtick: %b", $time, baudtick);
34     end
35
36 endmodule
```

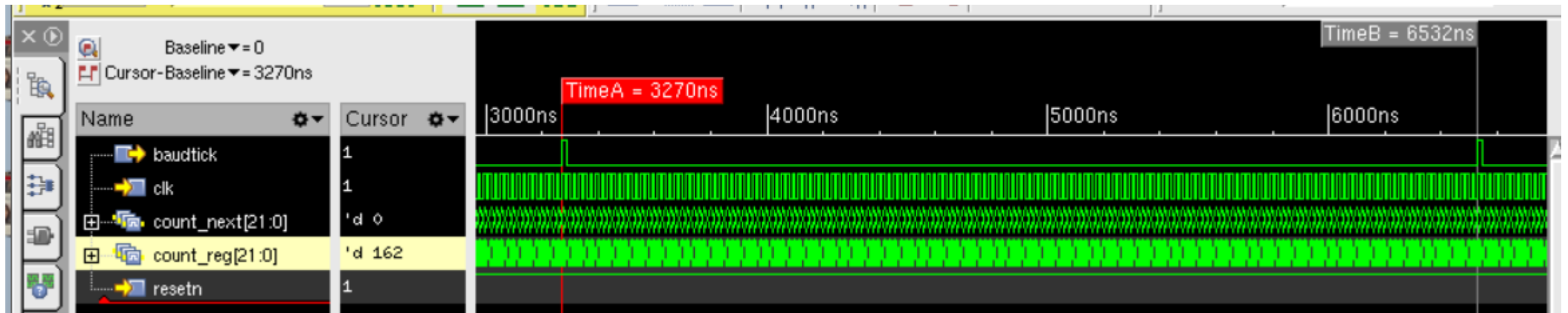
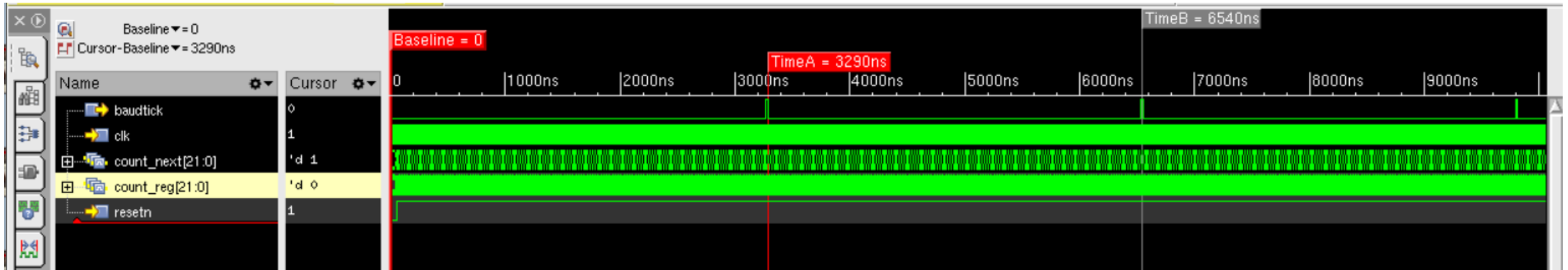
실습1 – baudrate 시뮬레이션 결과 setup



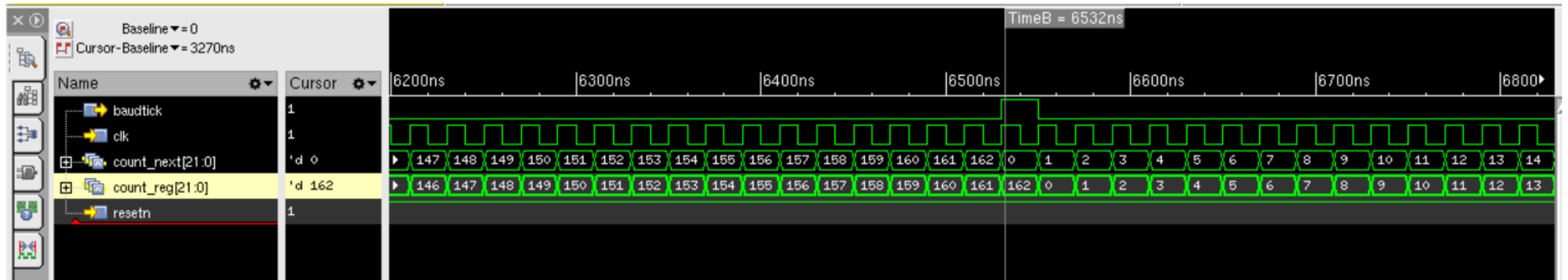
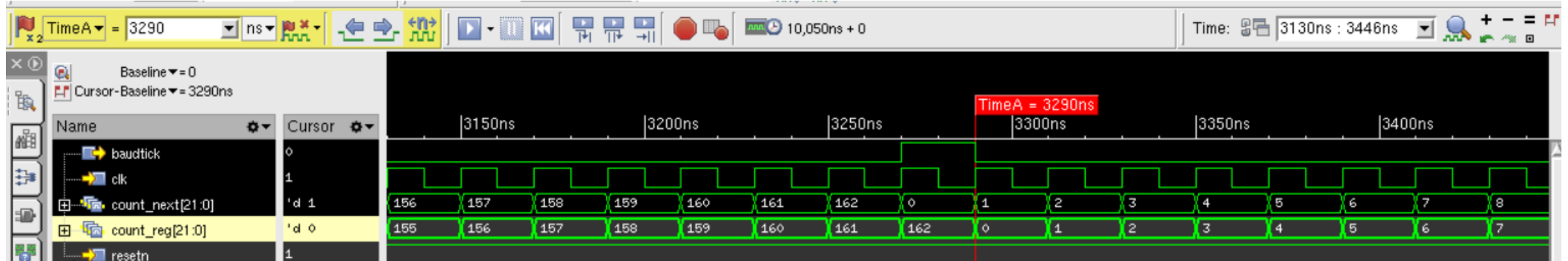
실습1 – baudrate 시뮬레이션 결과 setup



실습1 – baudrate시뮬레이션 결과1

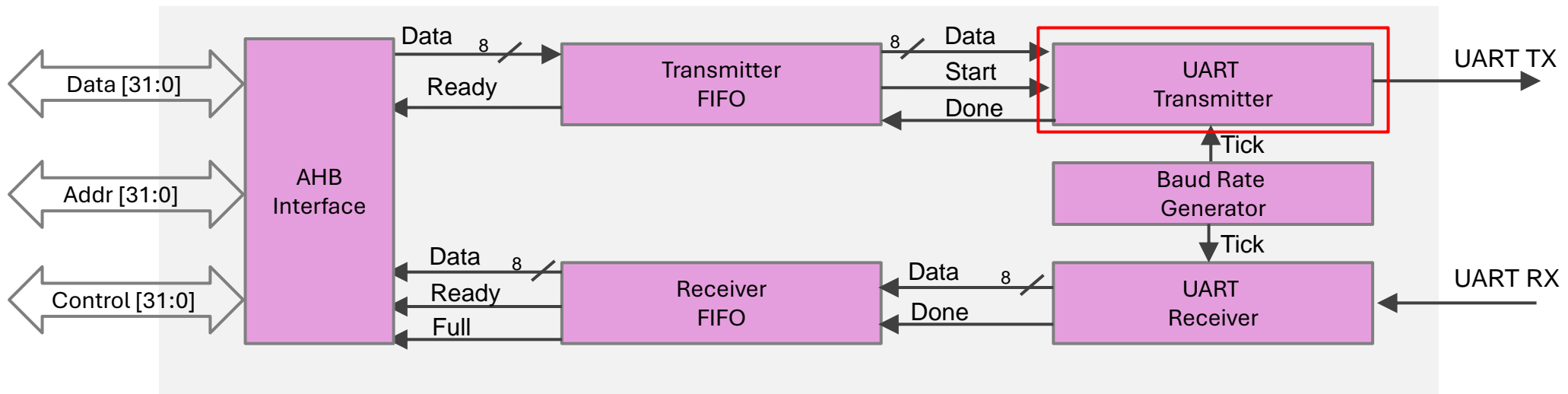


실습1 - baudrate시뮬레이션 결과2

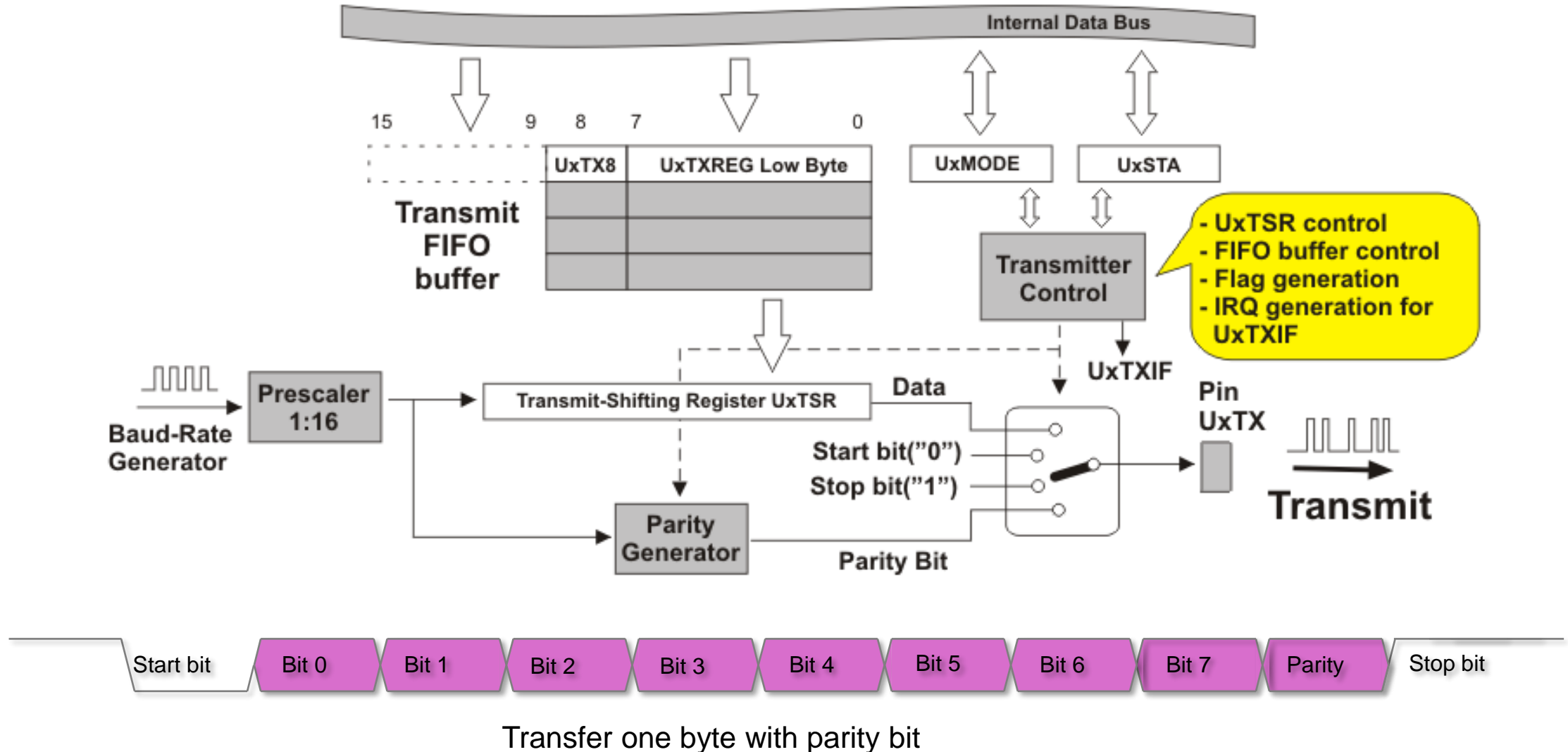


UART Transmitter

- 송신기 FIFO에서 데이터를 바이트 단위로 읽음.
- 단일 바이트 데이터를 순차적인 비트로 변환.
- 변환된 비트를 고정된 보드 레이트에서 제공되는 클럭에 따라 TX 핀으로 전송.



UART Transmitter Internal block



실습2 – UART TX – uart_tx.v

- 코드를 완성
하시오

1

```
36 `timescale 1ns / 1ps
37
38 module UART_TX(
39     input wire clk,
40     input wire resetn,
41     input wire tx_start,
42     input wire b_tick,           //baud rate tick
43     input wire [7:0] d_in,       //input data
44     output reg tx_done,          //transfer finished
45     output wire tx               //output data to RS-232
46 );
47
48
49 //STATE DEFINES
50 localparam [1:0] idle_st = 2'b00;
51 localparam [1:0] start_st = 2'b01;
52 localparam [1:0] data_st = 2'b11;
53 localparam [1:0] stop_st = 2'b10;
54
55 //Internal Signals
56 reg [1:0] current_state;
57 reg [1:0] next_state;
58 reg [3:0] b_reg;           //baud tick counter
59 reg [3:0] b_next;
60 reg [2:0] count_reg;       //data bit counter
61 reg [2:0] count_next;
62 reg [7:0] data_reg;        //data register
63 reg [7:0] data_next;
64 reg tx_reg;               //output data reg
65 reg tx_next;
```

```
67 //State Machine
68 always @(posedge clk, negedge resetn)
69 begin
70     if(!resetn)
71     begin
72         current_state <= idle_st;
73         b_reg <= 0;
74         count_reg <= 0;
75         data_reg <= 0;
76         tx_reg <= 1'b1;
77     end
78     else
79     begin
80         current_state <= next_state;
81         b_reg <= b_next;
82         count_reg <= count_next;
83         data_reg <= data_next;
84         tx_reg <= tx_next;
85     end
86 end
```

실습2 – UART TX – uart_tx.v

- 코드를 완성
하시오

1

```
89 //Next State Logic
90 always @*
91 begin
92     next_state = current_state;
93     tx_done = 1'b0;
94     b_next = b_reg;
95     count_next = count_reg;
96     data_next = data_reg;
97     tx_next = tx_reg;
98
99     case(current_state)
100     idle_st:
101     begin
102         tx_next = 1'b1;
103         if(tx_start)
104         begin
105             next_state = start_st;
106             b_next = 0;
107             data_next = d_in;
108         end
109     end
110
111     start_st: //send start bit
112     begin
113         tx_next = 1'b0;
114         if(b_tick)
115             if(b_reg==15)
116             begin
117                 next_state = data_st;
118                 b_next = 0;
119                 count_next = 0;
120             end
121             else
122                 b_next = b_reg + 1;
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
125 data_st: //send data serially
126 begin
127     tx_next = data_reg[0];
128
129     if(b_tick)
130         if(b_reg == 15)
131         begin
132             b_next = 0;
133             data_next = data_reg >> 1;
134             if(count_reg == 7) //8 data bits
135                 next_state = stop_st;
136             else
137                 count_next = count_reg + 1;
138         end
139     else
140         b_next = b_reg + 1;
141     end
142
143
144 stop_st: //send stop bit
145 begin
146     tx_next = 1'b1;
147     if(b_tick)
148         if(b_reg == 15) //one stop bit
149         begin
150             next_state = idle_st;
151             tx_done = 1'b1;
152         end
153     else
154         b_next = b_reg + 1;
155     end
156 endcase
end
```

실습2 – UART TX 시뮬레이션 – tb_uart_tx.v

- 시뮬레이션 테스트 벤치 코드를 완성

```
1 timescale 1ns / 1ps
2 module UART_TX_with_BAUDGEN_tb;
3
4 // 테스트벤치 신호 정의
5 reg clk;           // 클럭 신호
6 reg resetn;        // 리셋 신호 (Active Low)
7 reg tx_start;       // UART 송신 시작 신호
8 reg [7:0] d_in;     // UART 송신 데이터 입력
9 wire baudtick;      // BAUDGEN에서 생성된 보드레이트 틱 신호
10 wire tx_done;       // UART 송신 완료 신호
11 wire tx;            // UART 송신 출력 (TX 핀)
12
13 // BAUDGEN 모듈 인스턴스화
14 BAUDGEN baudgen_inst (
15     .clk(clk),        // 클럭 입력
16     .resetn(resetn),   // 리셋 입력
17     .baudtick(baudtick) // 보드레이트 틱 출력
18 );
19
20 // UART_TX 모듈 인스턴스화
21 UART_TX uart_tx_inst (
22     .clk(clk),        // 클럭 입력
23     .resetn(resetn),   // 리셋 입력
24     .tx_start(tx_start), // 송신 시작 신호 입력
25     .b_tick(baudtick), // 보드레이트 틱 입력
26     .d_in(d_in),       // 데이터 입력
27     .tx_done(tx_done), // 송신 완료 출력
28     .tx(tx)            // 직렬 데이터 출력 (TX 핀)
29 );
30
31 // 클럭 생성 (50MHz)
32 initial begin
33     clk = 0;
34     forever #10 clk = ~clk; // 주기가 20ns인 클럭 생성 (50MHz)
35 end
```

```
37 // 테스트 시퀀스 정의
38 initial begin
39     // 초기화 단계: 모든 신호를 초기값으로 설정
40     resetn = 0;           // 리셋 활성화 (Low)
41     tx_start = 0;         // 송신 시작 신호 비활성화
42     d_in = 8'b00000000;   // 기본 데이터 값 설정
43
44     #40 resetn = 1;       // 리셋 해제 (40ns 후)
45
46     // 테스트 케이스: 데이터 전송 테스트
47     #20 d_in = 8'b00110001; // 송신할 데이터 설정 (예: 이진 패턴)
48     tx_start = 1;         // 송신 시작 신호 활성화
49     #20 tx_start = 0;     // 송신 시작 신호 비활성화
50
51     wait(tx_done);        // tx_done이 활성화 될때까지 기다림
52     #10000;
53     $stop;               // 충분한 시간 후 시뮬레이션 종료 (약 2ms)
54 end
55
56 // 출력 모니터링 (디버깅용)
57 initial begin
58     $monitor("Time: %t | tx_done: %b | tx: %b | d_in: %b | baudtick: %b",
59         |           | $time, tx_done, tx, d_in, baudtick);
60     // 현재 시간, 송신 완료 신호, TX 출력, 입력 데이터, 보드레이트 틱 출력 표시
61 end
62
63 endmodule
```

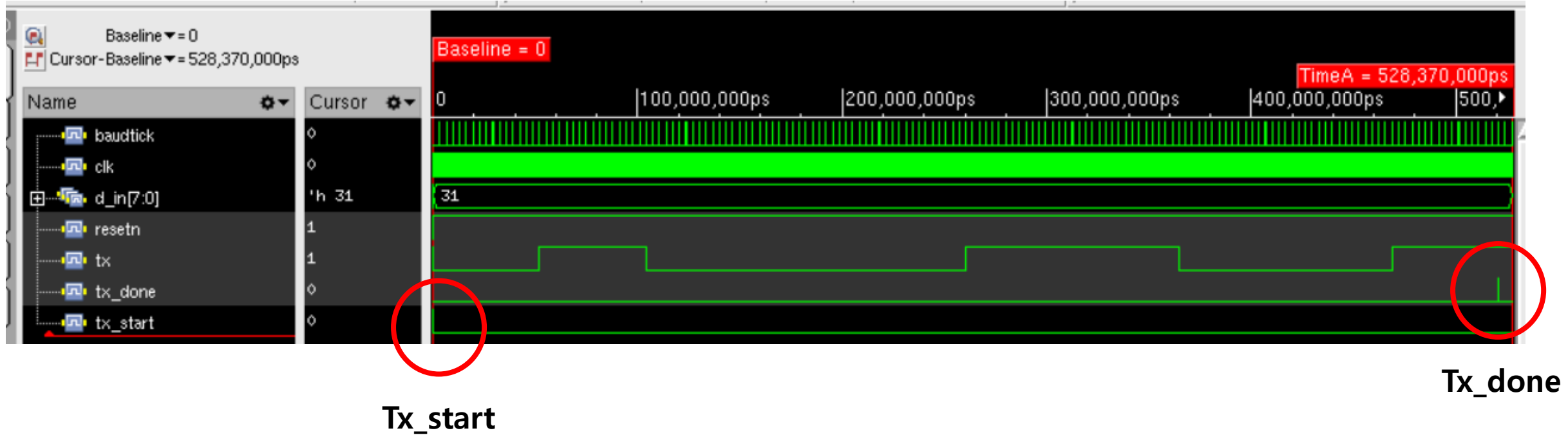

실습2 – UART TX 시뮬레이션 설정

```
class7]$ xrun -gui -access +rw baudgen.v uart_tx.v tb_uart_tx.v  
4) 24.03.2005: Started on Mar 23, 2025 at 14:20:06 KST  
B-s005: (c) Copyright 1995-2024 Cadence Design Systems, Inc.  
reason: file './baudgen.v' is newer than expected.
```

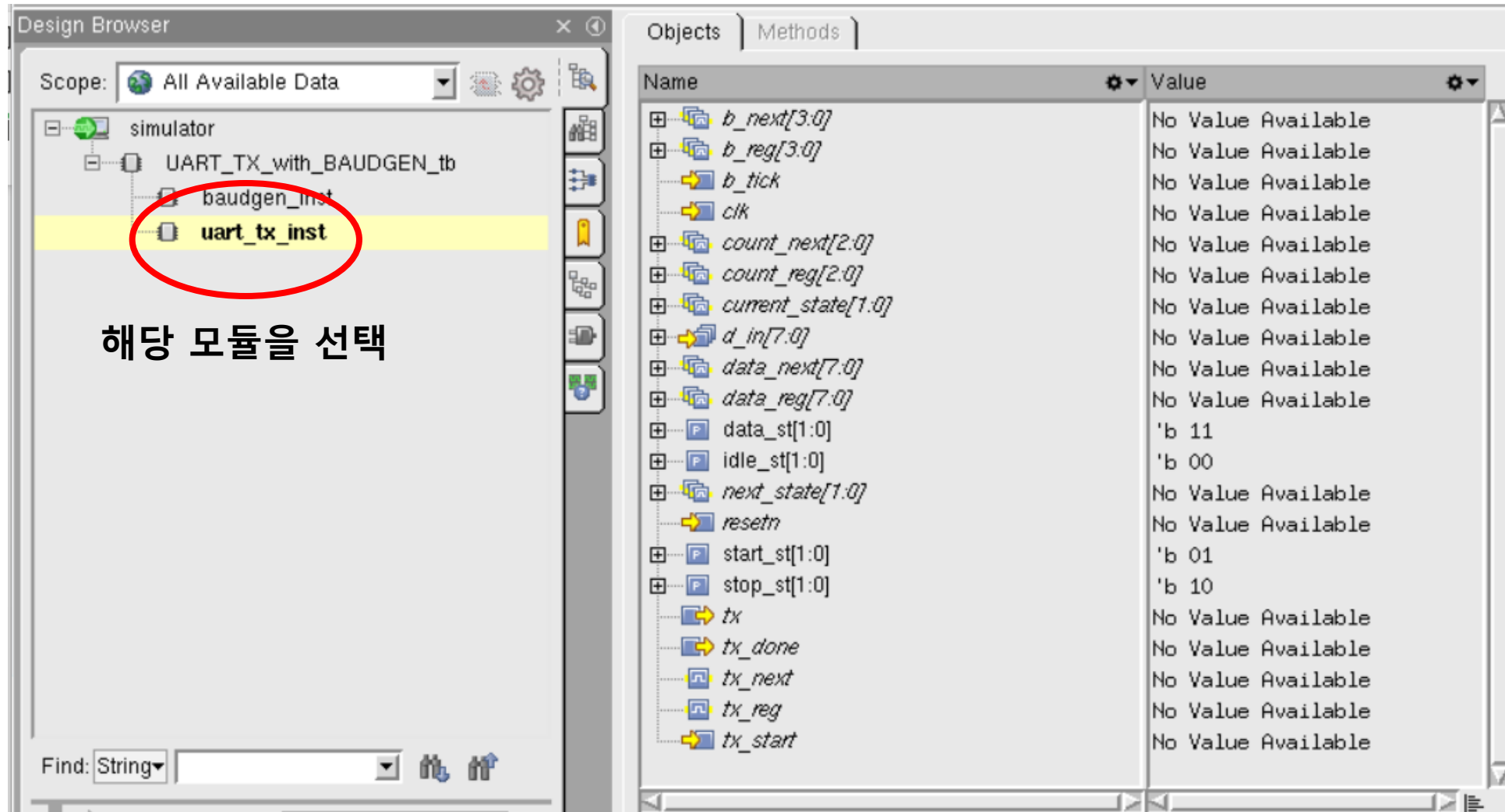
The screenshot displays the Cadence simulator interface. The top toolbar includes icons for file operations, simulation control, and data viewing. Below the toolbar, the 'Design Browser' panel on the left shows the project hierarchy with 'simulator' and 'UART_TX_with_BAUDGEN_tb' selected. The 'Objects' panel on the right lists the simulation objects and their current values.

Name	Value
<i>baudtick</i>	x
<i>clk</i>	x
<i>d_in[7:0]</i>	'h xx
<i>resetn</i>	x
<i>tx</i>	x
<i>tx_done</i>	x
<i>tx_start</i>	x

실습2 – UART TX 시뮬레이션 결과1



실습2 – UART TX 시뮬레이션 설정



Design Browser

Scope: All Available Data

- simulator
 - UART_TX_with_BAUDGEN_tb
 - baudgen_inst
 - uart_tx_inst**

해당 모듈을 선택

Find: String

Objects | Methods

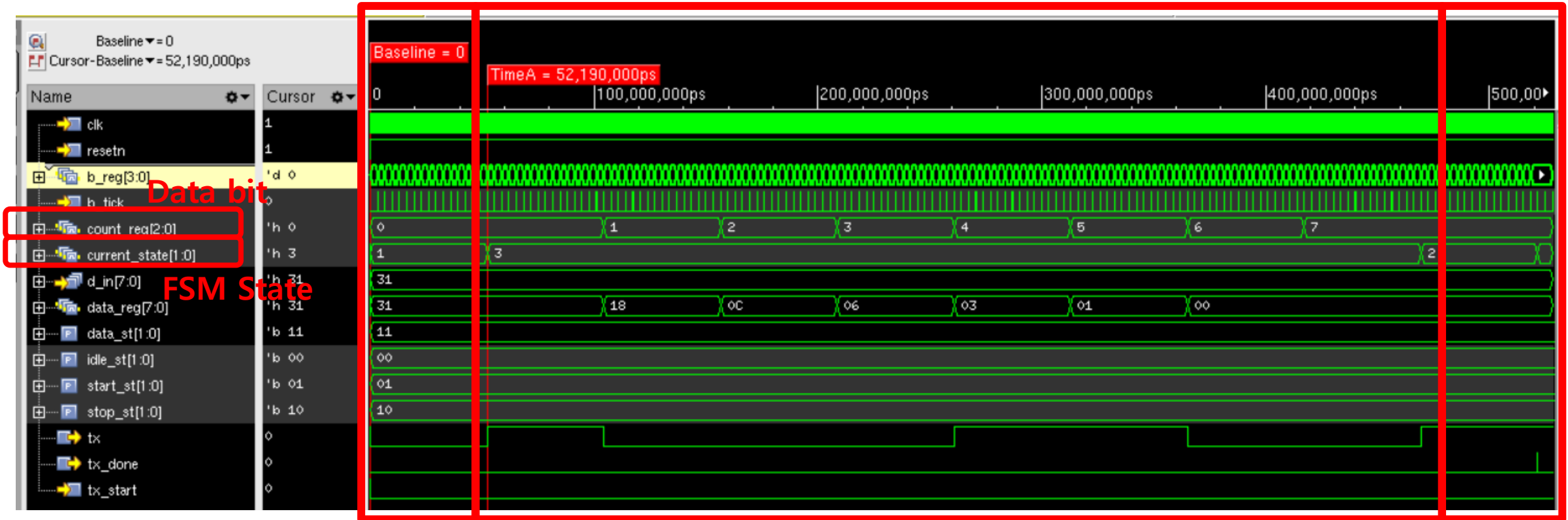
Name	Value
b_next[3:0]	No Value Available
b_reg[3:0]	No Value Available
b_tick	No Value Available
clk	No Value Available
count_next[2:0]	No Value Available
count_reg[2:0]	No Value Available
current_state[1:0]	No Value Available
d_in[7:0]	No Value Available
data_next[7:0]	No Value Available
data_reg[7:0]	No Value Available
data_st[1:0]	'b 11
idle_st[1:0]	'b 00
next_state[1:0]	No Value Available
resetn	No Value Available
start_st[1:0]	'b 01
stop_st[1:0]	'b 10
tx	No Value Available
tx_done	No Value Available
tx_next	No Value Available
tx_reg	No Value Available
tx_start	No Value Available

실습2 – UART TX 시뮬레이션 결과2

START

Data 전송

STOP

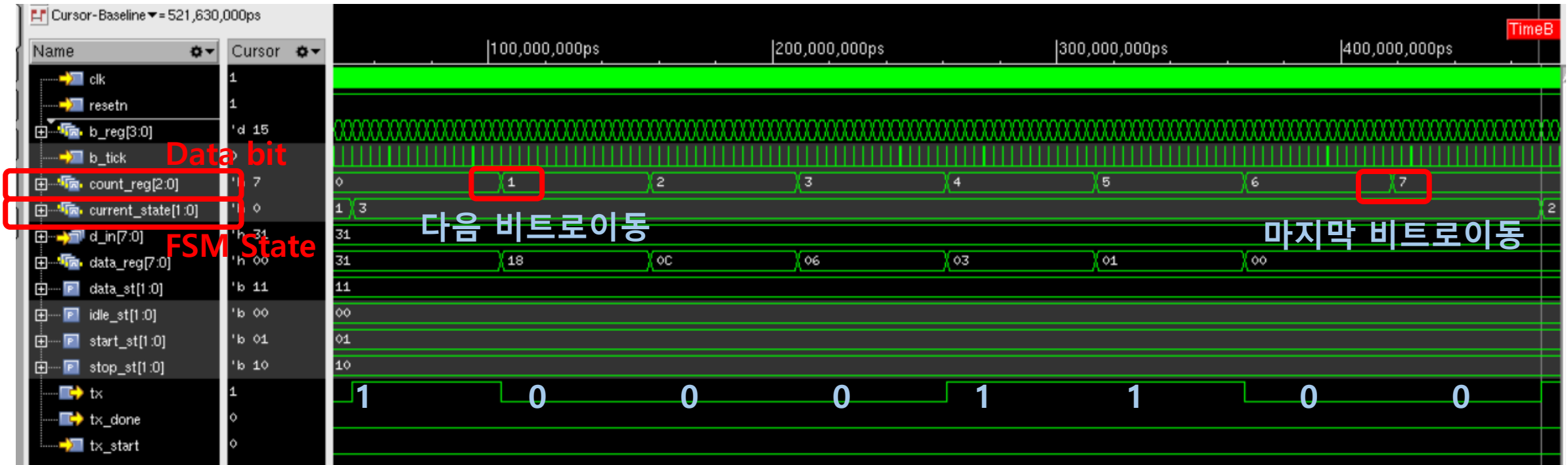


실습2 – UART TX 시뮬레이션 결과3 – START

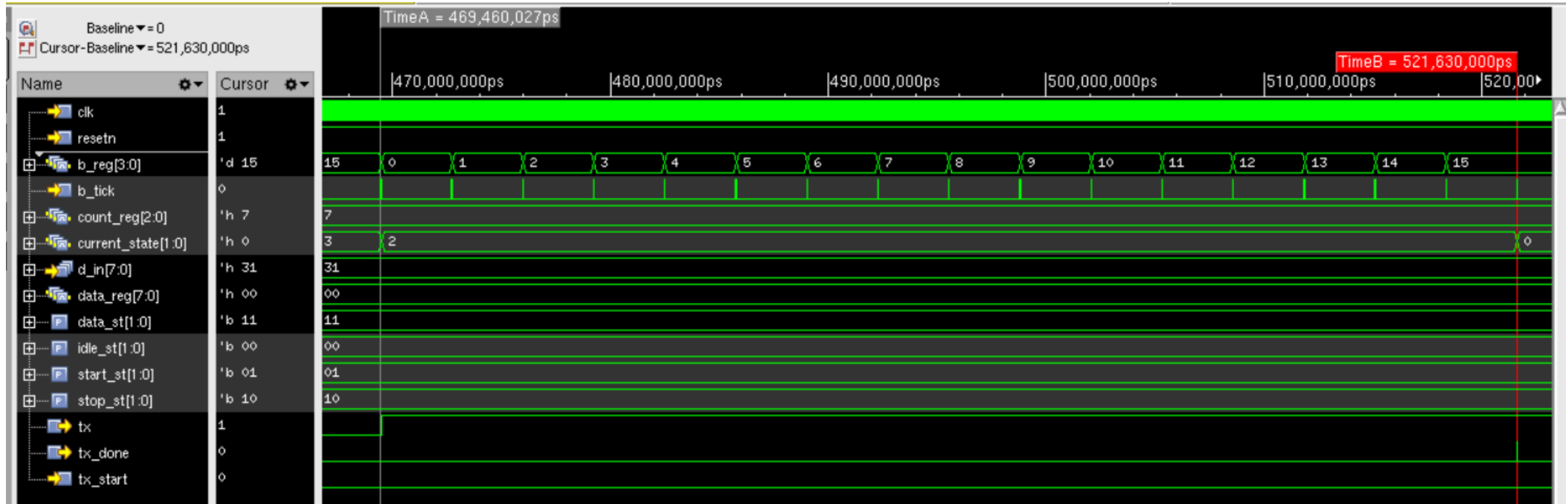


Tx_start trigger

실습2 – UART TX 시뮬레이션 결과4



실습2 – UART TX 시뮬레이션 결과5



Thanks