RISC-V-MULTICYCLE 발표

발표자 : 이승후

목차

1.개요

2.RISC-V-MULTICYCLE

3.AMBA-BUS

4.TROUBLESHOOTING

5.동작 영상

개요

프로젝트의 목표는 RISC-V-MULTICYCLE 코어, AMBA 버스 및 UART 주변 장치를 통합하여 연결하고, APB 프로토콜 숙달 및 SLAVE 인터페이스 구현을 통해 UART가 RISC-V의 제어 하에 기능을 정확히 수행하는지 검증하는 것이 목표

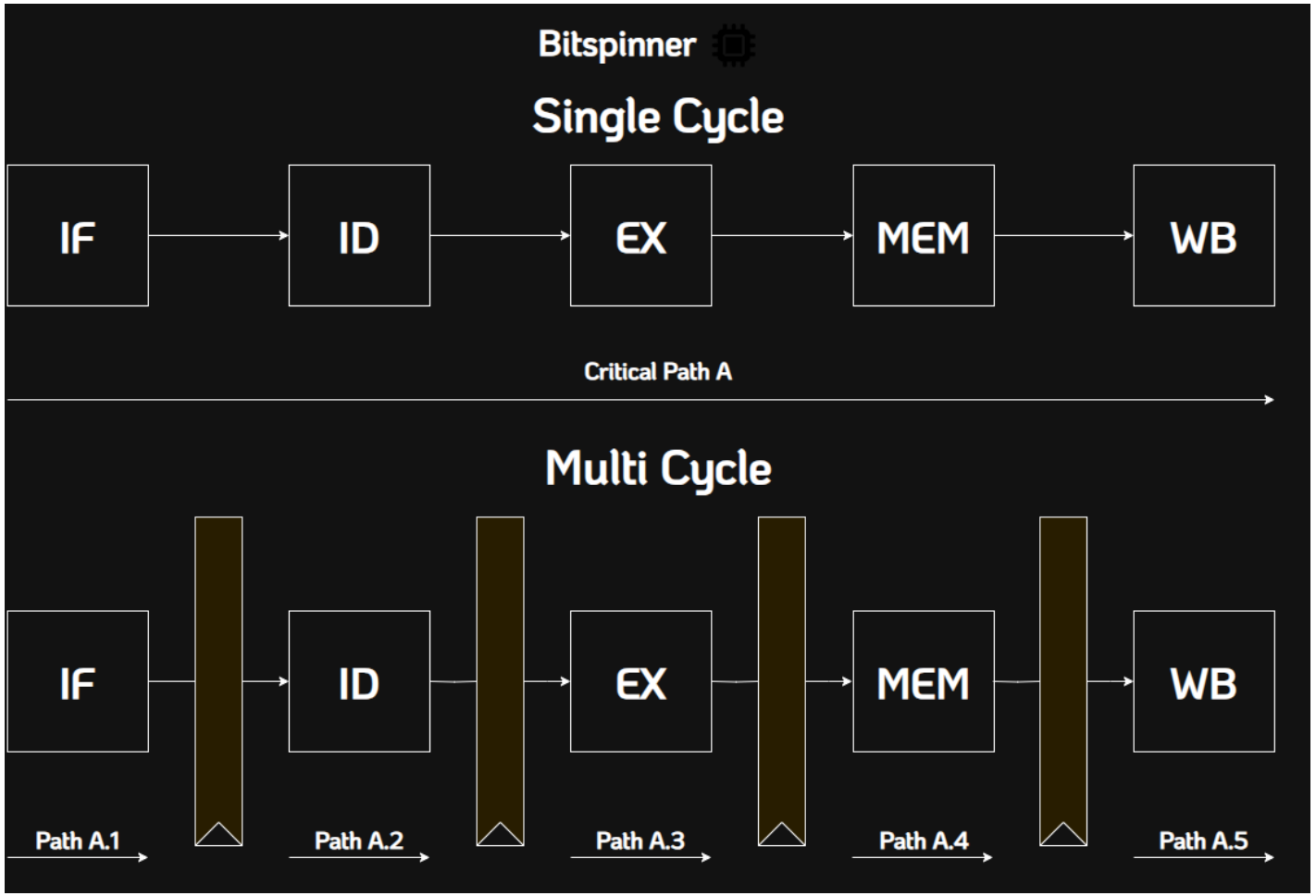
RISC-V-MULTICYCLE

RISC-V는 누구나 자유롭게 사용할 수 있는 개방형 표준 명령어 집합 구조(ISA)로, 단순하고 모듈화된 설계를 통해 맞춤형 프로세서 개발을 용이하게 만드는 아키텍처

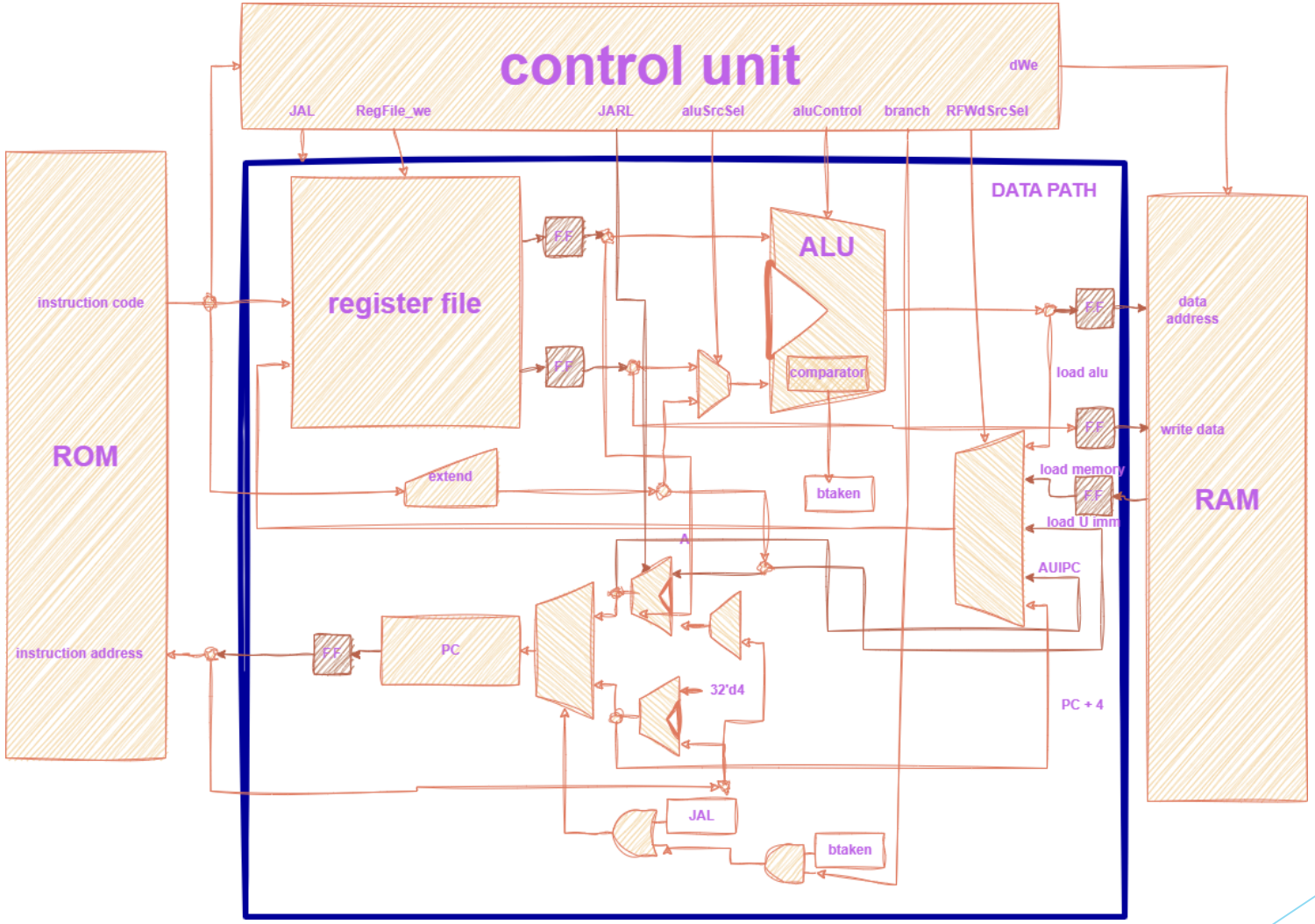
RISC-V-MULTICYCLE

MULTI CYCLE 구조는 하나의 명령어를 여러 독립적인 단계로 나누어 각 단계를 한 클럭 사이클에 실행함으로써, ALU와 같은 기능 블록을 재사용하고 최고 클럭 주파수를 높일 수 있도록 설계된 효율적인 프로세서 아키텍처

RISC-V-MULTICYCLE



RISC-V-MULTICYCLE



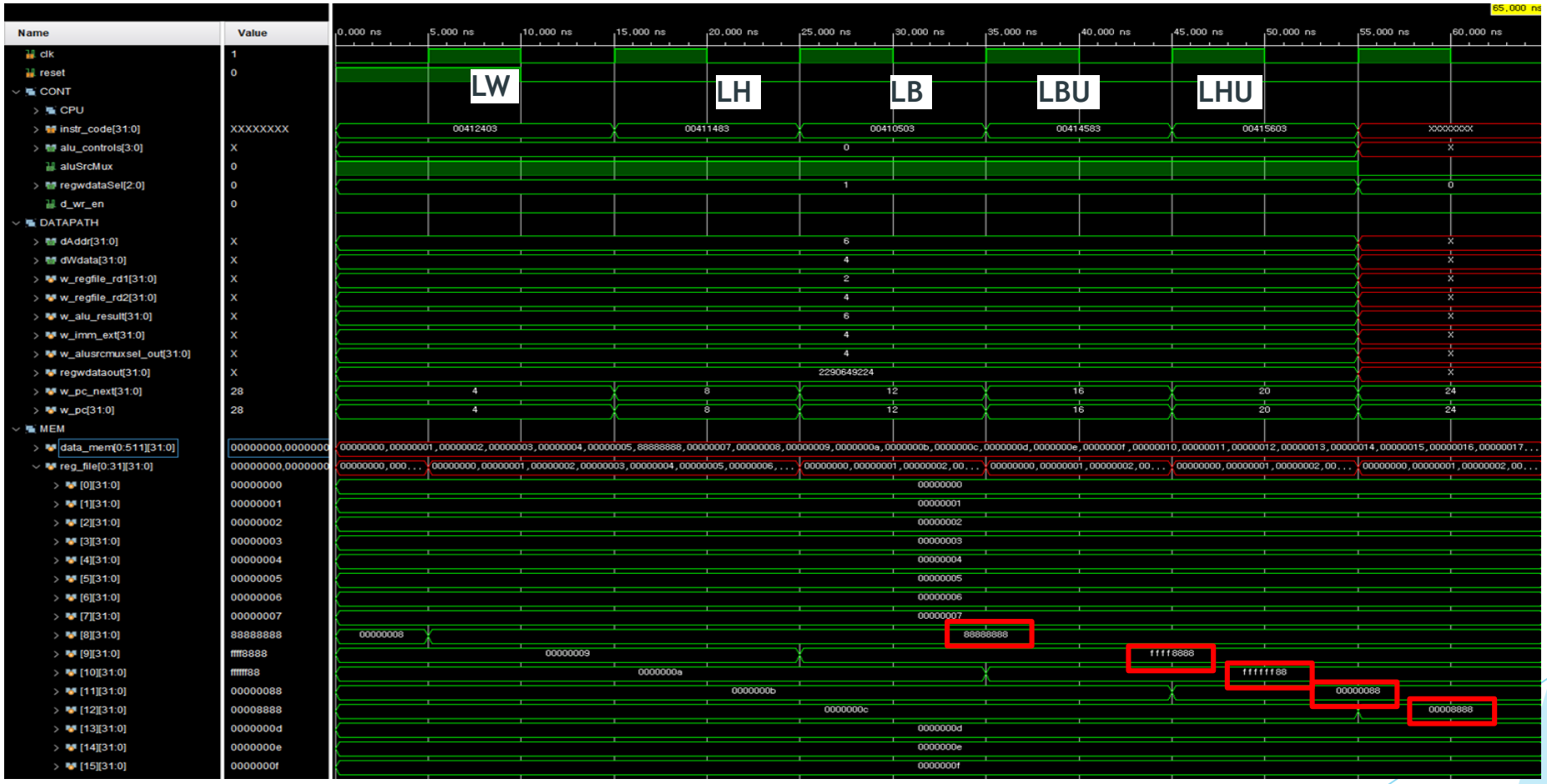
RISC-V-MULTICYCLE

0	0	0	0	0	0	0	rs2	rs1	0	0	0	rd	0	1	1	0	0	1	1	R	ADD	rd = rs1 + rs2	
0	1	0	0	0	0	0	rs2	rs1	0	0	0	rd	0	1	1	0	0	1	1	R	SUB	rd = rs1 - rs2	
0	0	0	0	0	0	0	rs2	rs1	0	0	1	rd	0	1	1	0	0	1	1	R	SLL	rd = rs1 << rs2	
0	0	0	0	0	0	0	rs2	rs1	0	1	0	rd	0	1	1	0	0	1	1	R	SLT	rd = (rs1 < rs2)?1:0	
0	0	0	0	0	0	0	rs2	rs1	0	1	1	rd	0	1	1	0	0	1	1	R	SLTU	rd = (rs1 < rs2)?1:0	zero-extends
0	0	0	0	0	0	0	rs2	rs1	1	0	0	rd	0	1	1	0	0	1	1	R	XOR	rd = rs1 ^ rs2	
0	0	0	0	0	0	0	rs2	rs1	1	0	1	rd	0	1	1	0	0	1	1	R	SRL	rd = rs1 >> rs2	
0	1	0	0	0	0	0	rs2	rs1	1	0	1	rd	0	1	1	0	0	1	1	R	SRA	rd = rs1 >> rs2	msb-extends
0	0	0	0	0	0	0	rs2	rs1	1	1	0	rd	0	1	1	0	0	1	1	R	OR	rd = rs1 | rs2	
0	0	0	0	0	0	0	rs2	rs1	1	1	1	rd	0	1	1	0	0	1	1	R	AND	rd = rs1 & rs2	

RISC-V-MULTICYCLE

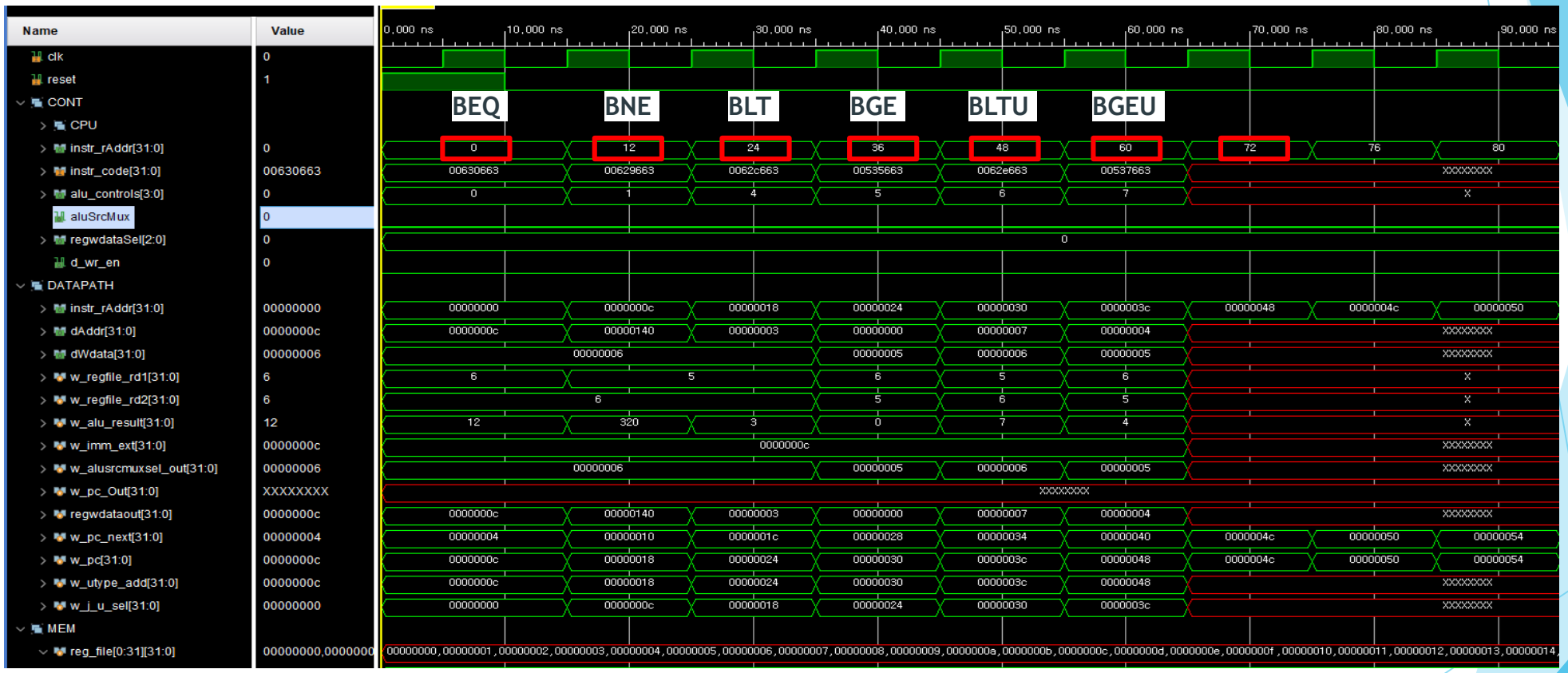
imm[11:5]	rs2	rs1	0	0	0	imm[4:0]	0	1	0	0	0	1	1	S	SB	M[rs1+imm][0:7] = rs2[0:7]
imm[11:5]	rs2	rs1	0	0	1	imm[4:0]	0	1	0	0	0	1	1	S	SH	M[rs1+imm][0:15] = rs2[0:15]
imm[11:5]	rs2	rs1	0	1	0	imm[4:0]	0	1	0	0	0	1	1	S	SW	M[rs1+imm][0:31] = rs2[0:31]

RISC-V-MULTICYCLE



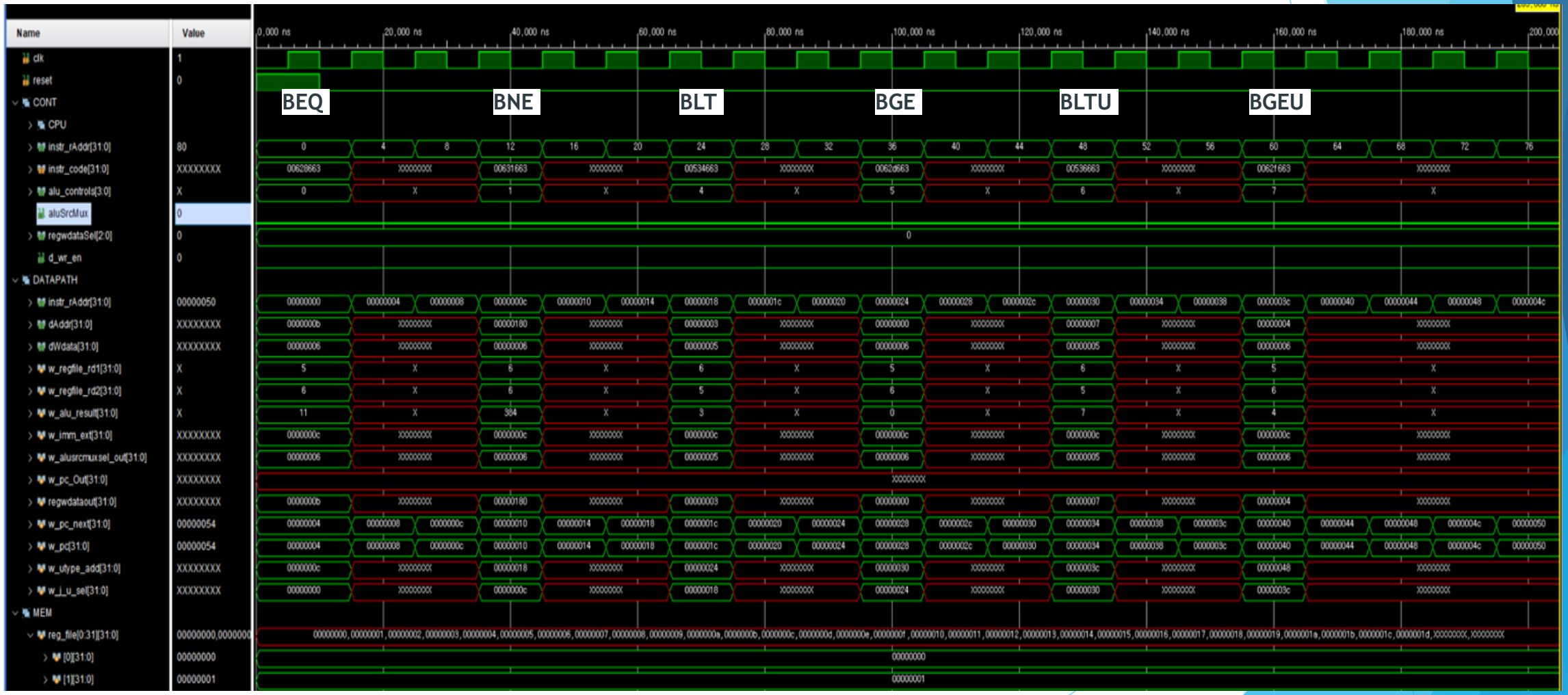
imm[11:0]	rs1	0	0	0	rd	0	0	0	0	0	1	1	I	LB	rd = M[rs1+imm][0:7]	
imm[11:0]	rs1	0	0	1	rd	0	0	0	0	0	1	1	I	LH	rd = M[rs1+imm][0:15]	
imm[11:0]	rs1	0	1	0	rd	0	0	0	0	0	1	1	I	LW	rd = M[rs1+imm][0:31]	
imm[11:0]	rs1	1	0	0	rd	0	0	0	0	0	1	1	I	LBU	rd = M[rs1+imm][0:7]	zero-extends
imm[11:0]	rs1	1	0	1	rd	0	0	0	0	0	1	1	I	LHU	rd = M[rs1+imm][0:15]	zero-extends

RISC-V-MULTICYCLE

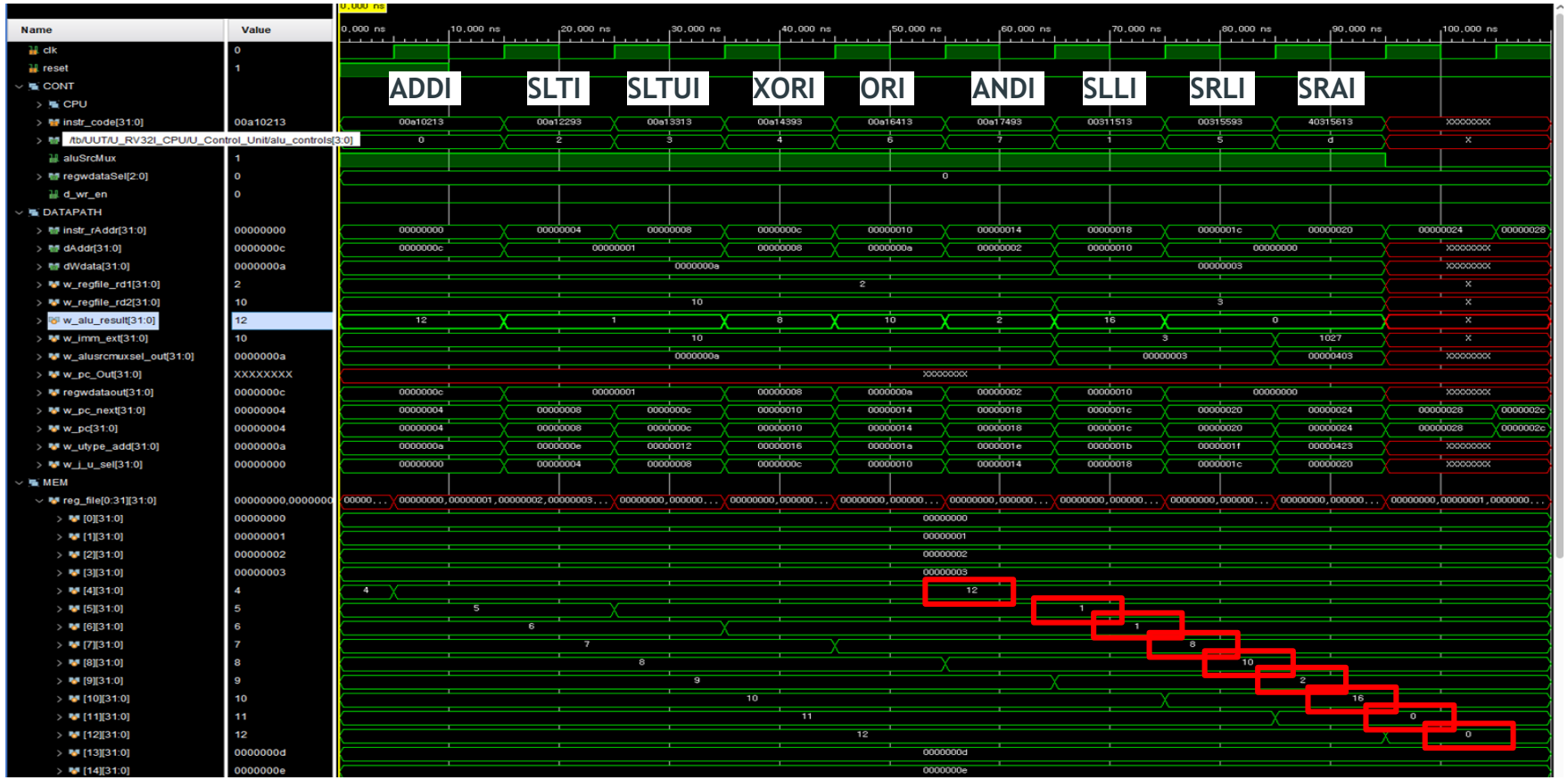


imm[12,10:5]	rs2	rs1	0	0	0	imm[4:1,11]	1	1	0	0	0	1	1	B	BEQ	if(rs1 == rs2) PC += imm	
imm[12,10:5]	rs2	rs1	0	0	1	imm[4:1,11]	1	1	0	0	0	1	1	B	BNE	if(rs1 != rs2) PC += imm	
imm[12,10:5]	rs2	rs1	1	0	0	imm[4:1,11]	1	1	0	0	0	1	1	B	BLT	if(rs1 < rs2) PC += imm	
imm[12,10:5]	rs2	rs1	1	0	1	imm[4:1,11]	1	1	0	0	0	1	1	B	BGE	if(rs1 >= rs2) PC += imm	
imm[12,10:5]	rs2	rs1	1	1	0	imm[4:1,11]	1	1	0	0	0	1	1	B	BLTU	if(rs1 < rs2) PC += imm	zero-extends
imm[12,10:5]	rs2	rs1	1	1	1	imm[4:1,11]	1	1	0	0	0	1	1	B	BGEU	if(rs1 >= rs2) PC += imm	zero-extends

RISC-V-MULTICYCLE

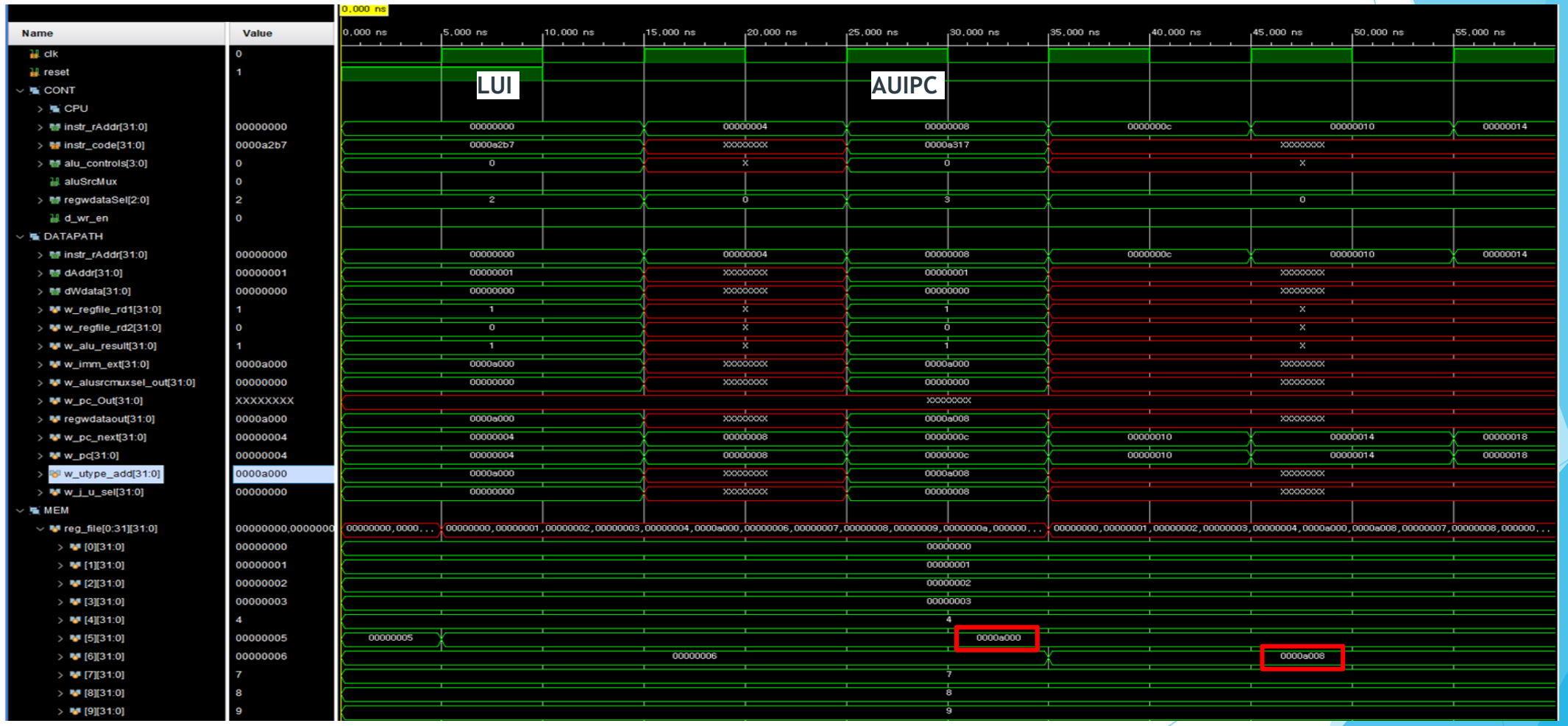


RISC-V-MULTICYCLE



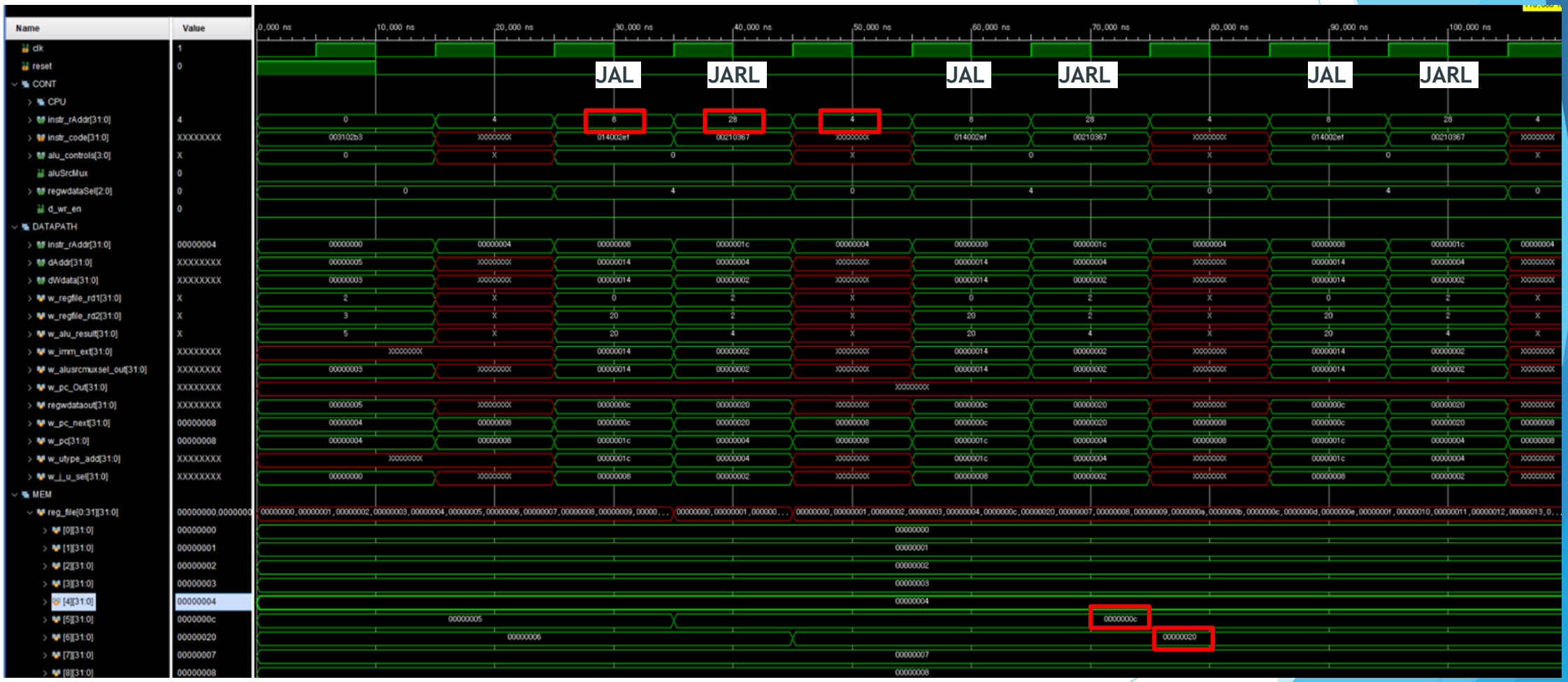
imm[11:0]							rs1	0	0	0	rd	0	0	1	0	0	1	1	I	ADDI	rd = rs1 + imm	
imm[11:0]							rs1	0	1	0	rd	0	0	1	0	0	1	1	I	SLTI	rd = (rs1 < imm)?1:0	
imm[11:0]							rs1	0	1	1	rd	0	0	1	0	0	1	1	I	SLTIU	rd = (rs1 < imm)?1:0	zero-extends
imm[11:0]							rs1	1	0	0	rd	0	0	1	0	0	1	1	I	XORI	rd = rs1 ^ imm	
imm[11:0]							rs1	1	1	0	rd	0	0	1	0	0	1	1	I	ORI	rd = rs1 | imm	
imm[11:0]							rs1	1	1	1	rd	0	0	1	0	0	1	1	I	ANDI	rd = rs1 & imm	
0	0	0	0	0	0	0	shamt	rs1	0	0	1	rd	0	0	1	0	0	1	1	I	SLLI	rd = rs1 << imm[0:4]	
0	0	0	0	0	0	0	shamt	rs1	1	0	1	rd	0	0	1	0	0	1	1	I	SRLI	rd = rs1 >> imm[0:4]	
0	1	0	0	0	0	0	shamt	rs1	1	0	1	rd	0	0	1	0	0	1	1	I	SRAI	rd = rs1 >> imm[0:4]	msb-extends

RISC-V-MULTICYCLE



imm[31:12]	rd	0	1	1	0	1	1	1	U	LUI	rd = imm
imm[31:12]	rd	0	0	1	0	1	1	1	U	AUIPC	rd = PC + imm

RISC-V-MULTICYCLE

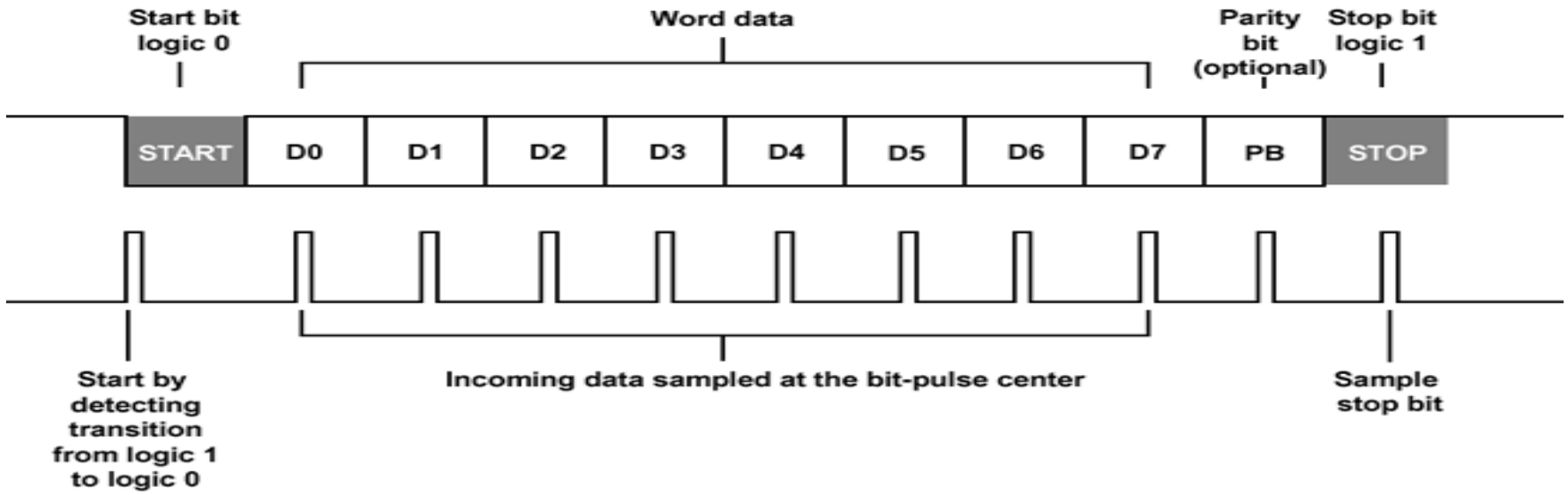


imm[11:0]	rs1	0	0	0	rd	1	1	0	0	1	1	1	I	JALR	rd = PC+4; PC = rs1 + imm
imm[20,10:1,11,19:12]					rd	1	1	0	1	1	1	1	J	JAL	rd = PC+4; PC += imm

AMBA-BUS

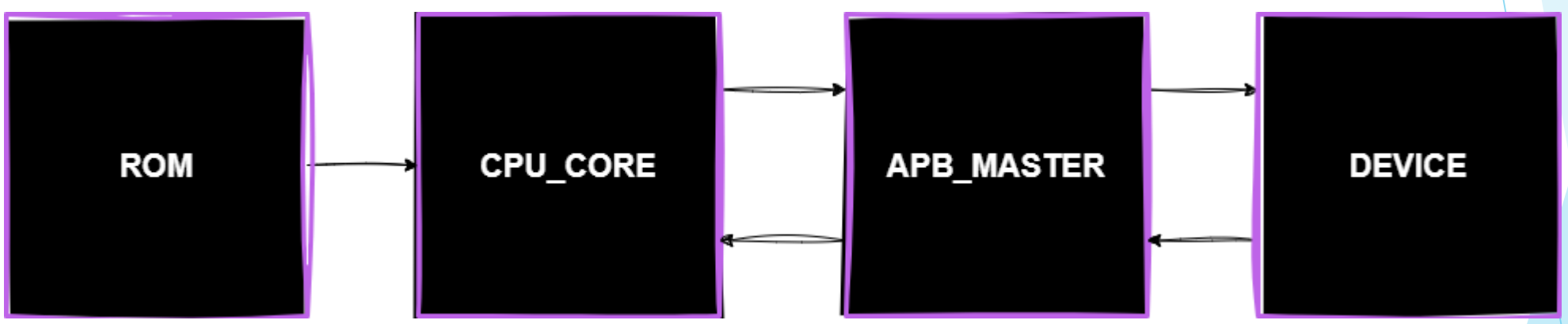
APB (Advanced Peripheral Bus)는 AMBA 프로토콜 제품군 중 하나로, 주로 저속 및 낮은 대역폭을 요구하는 주변 장치(예: UART, 타이머, GPIO 등)에 대한 접근을 위해 설계된 간단하고 지연 없는 인터페이스를 제공하는 버스입니다.

AMBA-BUS

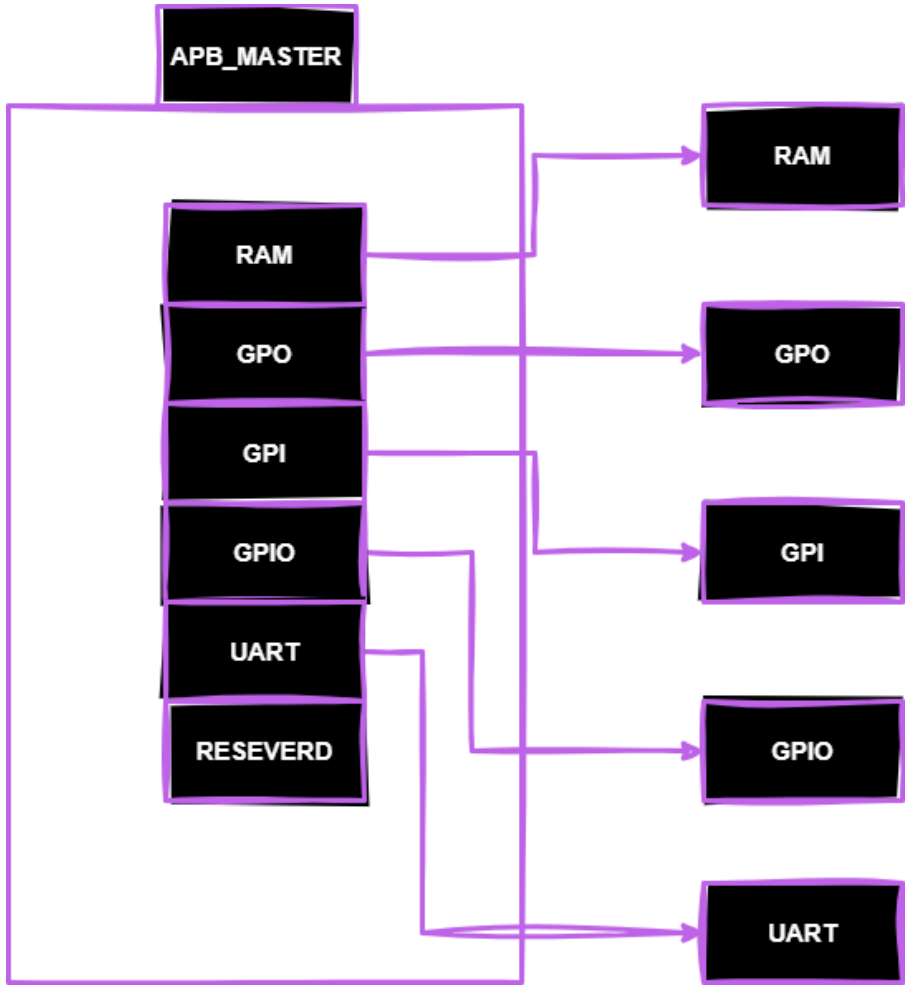


UART (Universal Asynchronous Receiver/Transmitter)는 데이터 프레임(Start bit, Data bits, Stop bit)을 사용하여 병렬 데이터를 직렬 형태로 변환하여 전송하거나 직렬 데이터를 병렬 데이터로 변환하여 수신하는 비동기식 직렬 통신을 위한 하드웨어 장치

AMBA-BUS



AMBA-BUS



RAM = 0x10000000 ~ 0x10000FFF

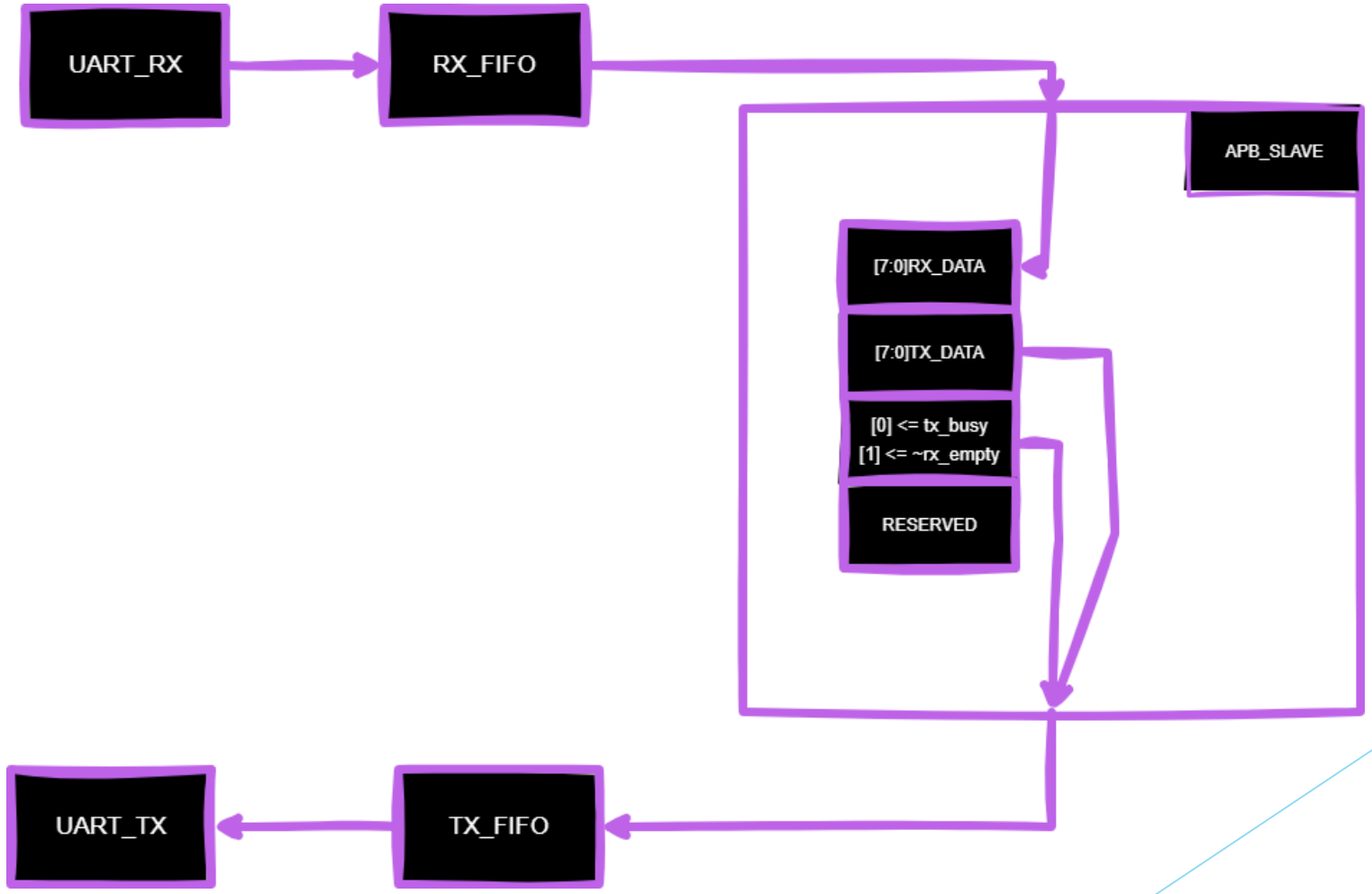
GPO = 0x10001000 ~ 0x10001FFF

GPI = 0x10002000 ~ 0x10002FFF

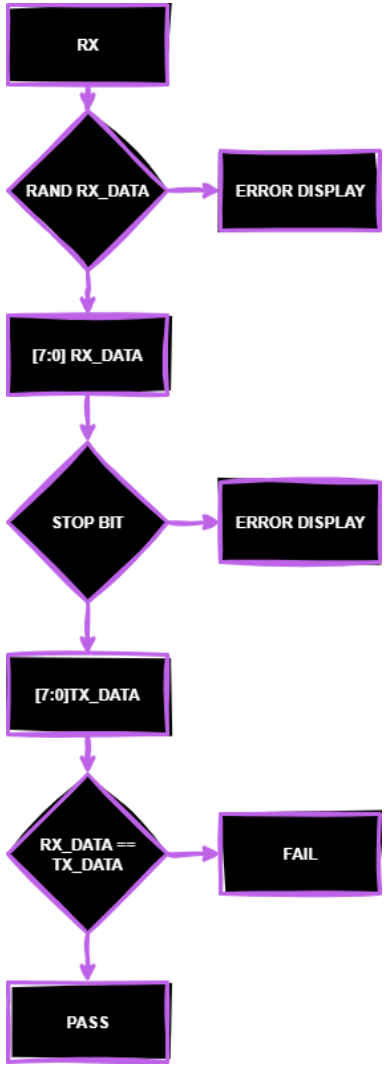
GPIO = 0x10003000 ~ 0x10003FFF

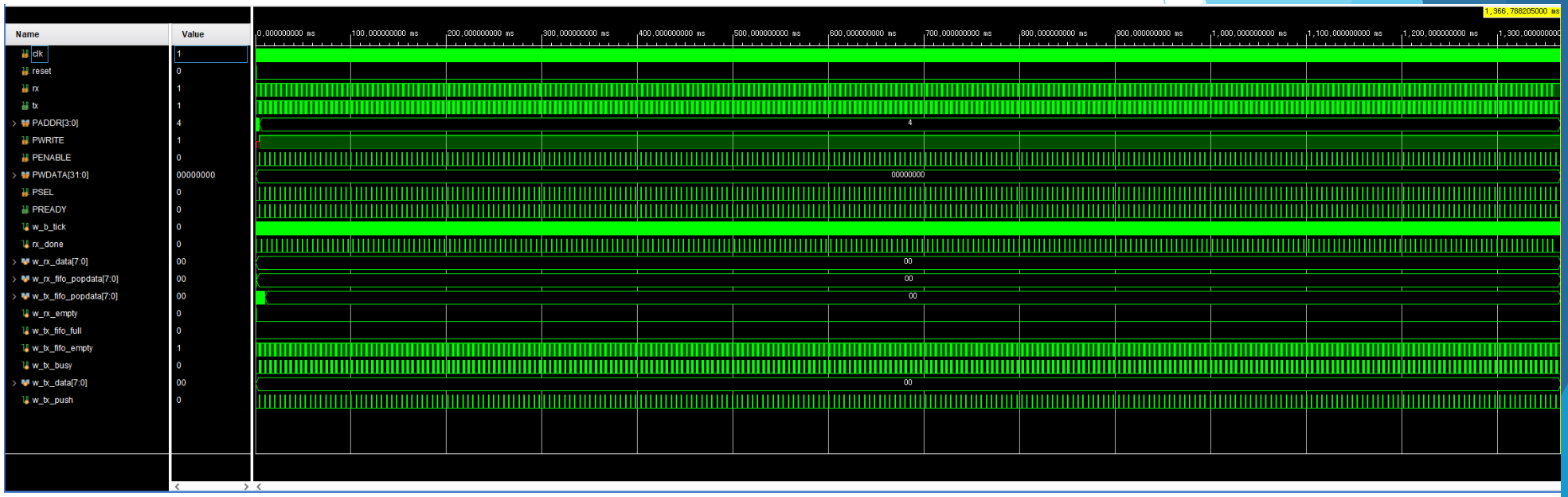
UART = 0x10004000 ~ 0x10004FFF

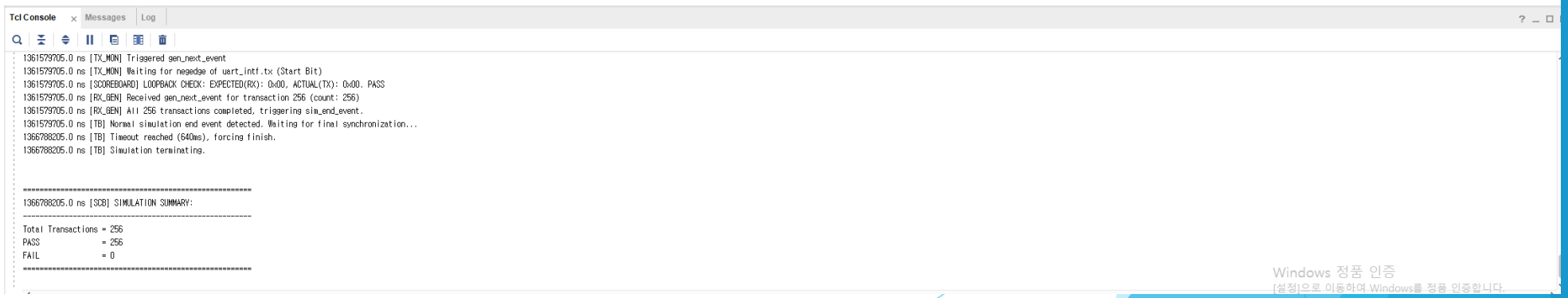
AMBA-BUS


UART_RX
RX_FIFO
APB_SLAVE
[7:0]RX_DATA
[7:0]TX_DATA
[0] <= tx_busy
[1] <= ~rx_empty
RESERVED
UART_TX
TX_FIFO

AMBA-BUS

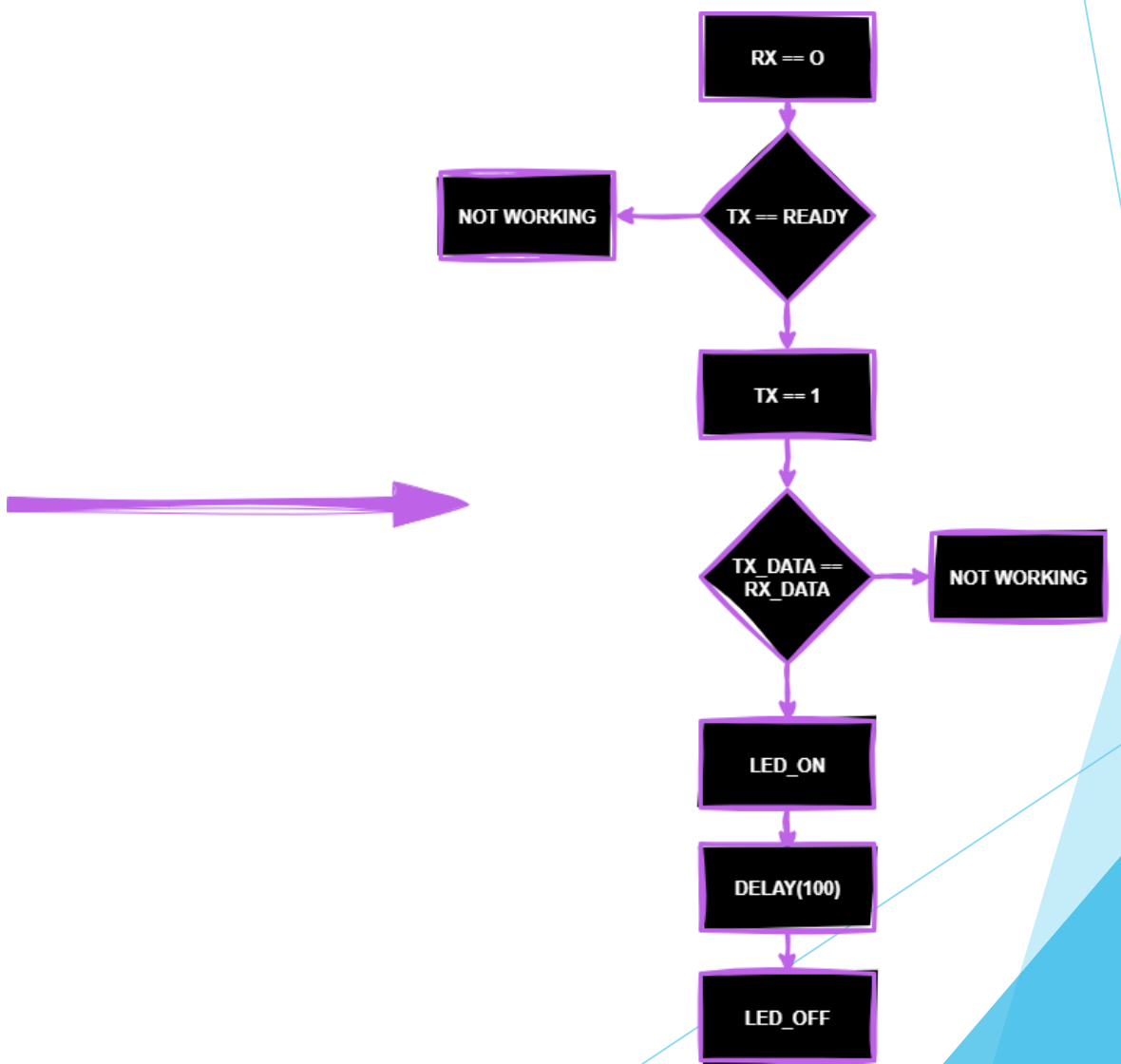


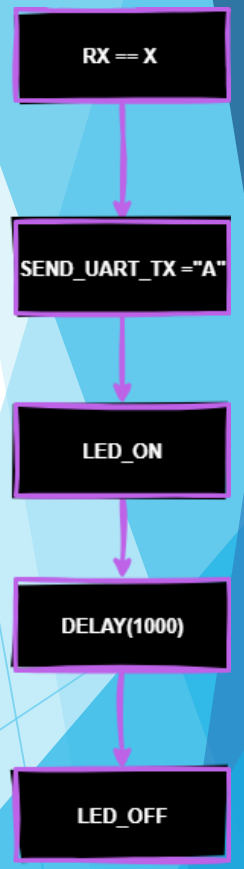




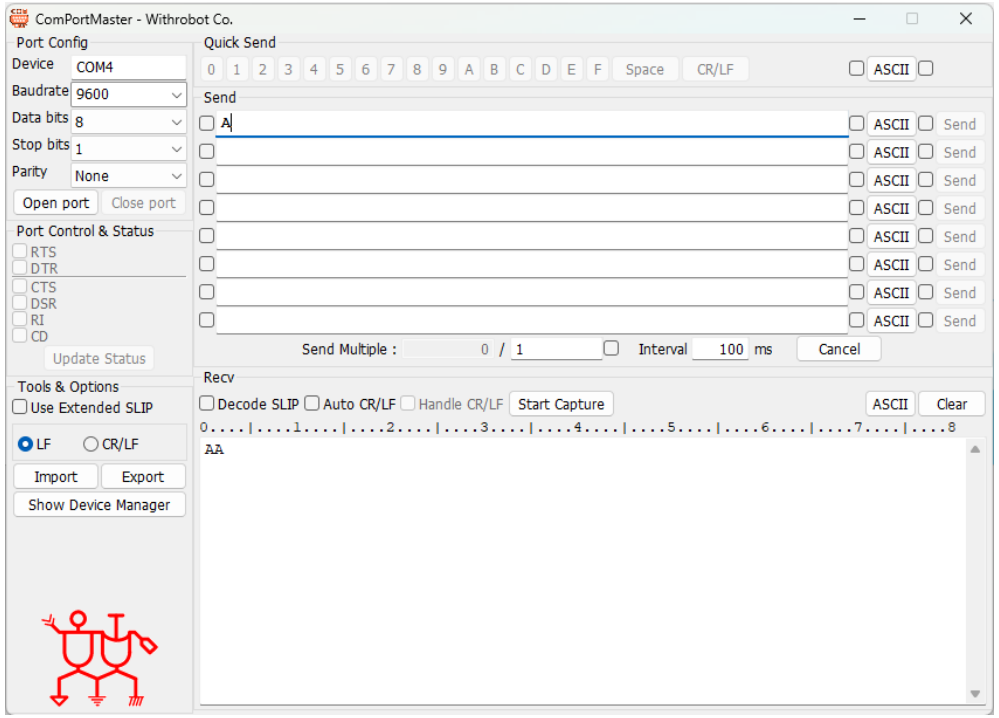
AMBA-BUS

```c
#include <stdint.h>

#define APB_BASE        0x10000000
#define GPO_OFFSET      0x1000
#define UART_OFFSET     0x4000

#define GPO_BASE        (APB_BASE + GPO_OFFSET)
#define UART_BASE       (APB_BASE + UART_OFFSET)

#define GPO_CR          (*(volatile uint32_t *)(GPO_BASE + 0x00))
#define GPO_ODR         (*(volatile uint32_t *)(GPO_BASE + 0x04))
#define UART_RDATA      (*(volatile uint32_t *)(UART_BASE + 0x00))
#define UART_TXDATA     (*(volatile uint32_t *)(UART_BASE + 0x04))
#define UART_STATUS     (*(volatile uint32_t *)(UART_BASE + 0x08))

void System_init(void);
void delay(uint32_t t);
void set_led_pattern(uint8_t led_value);

int main() {
    System_init();

    uint8_t led_index = 0;
    uint8_t pattern = 0x01;

    uint32_t timer_ms = 0;
    const uint32_t PERIOD_MS = 1000;

    while (1) {
        if (UART_STATUS & 0x02) {

            uint8_t rx_byte = (uint8_t)UART_RDATA;

            for (volatile int i = 0; i < 50; i++);

            while (UART_STATUS & 0x01);
            UART_TXDATA = rx_byte;

            set_led_pattern(pattern);
            led_index = (led_index + 1) % 4;

            delay(100);

            set_led_pattern(0x0);

            timer_ms = 0;

        } else {

            delay(1);
            timer_ms++;

            if (timer_ms >= PERIOD_MS) {

                uint8_t tx_char = 'A';

                while (UART_STATUS & 0x01);
                UART_TXDATA = tx_char;

                // 5번 LED (0x10) 켜짐
                set_led_pattern(0x10);

                // LED 켜짐 시간 측정
                delay(1000);

                // LED 소등
                set_led_pattern(0x0);

                timer_ms = 0;
            }
        }
    }
}

void System_init(void) {
    // GPO_CR을 0x0F이면 0x1F로 변경하여 5번 LED (Bit 4)를 출력으로 설정
    GPO_CR = 0x1F;
    GPO_ODR = 0x00;
}

void delay(uint32_t t) {
    volatile uint32_t temp = 0;
    for (uint32_t i = 0; i < t; i++) {
        for (uint32_t j = 0; j < 1000; j++) {
            temp++;
        }
    }
}

void set_led_pattern(uint8_t led_value) {
    // 0x0F이면 0x1F로 변경하여 GPO Bit 0~4 (5개 LED)를 제어
    GPO_ODR = led_value & 0x1F;
}
```





TROUBLESHOOTING



옆에 사진 처럼 값이 두번 나오는 현상이 있었음

APB_MASTER에서 ACCESS가 두클락이 유지되면서 PENABLE과 DECODER_EN이 둘다 유지되면서 생기는 현상임을 알아냄

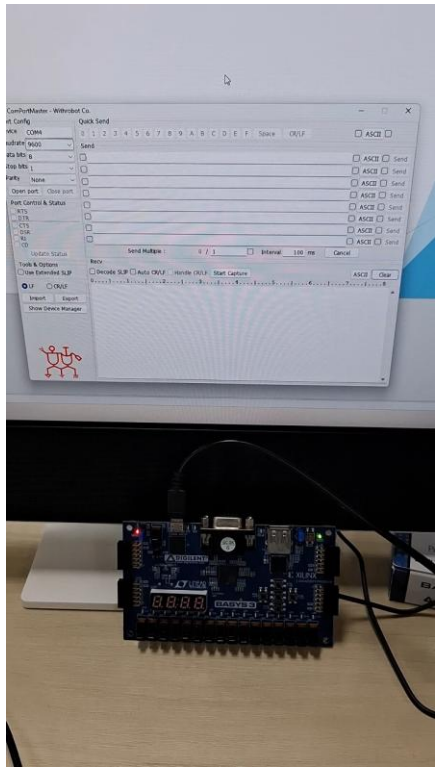
TROUBLESHOOTING

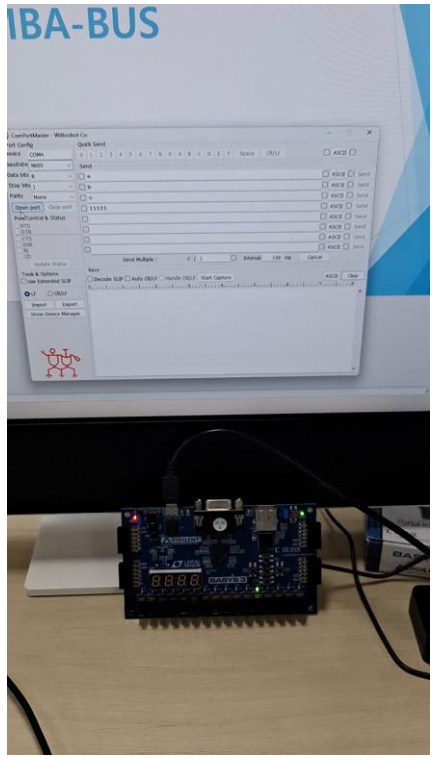
```
            ACCESS: begin
                decoder_en = 1'b1;
                PENABLE    = 1'b1;
                if (ready) begin
                    state_next = IDLE;
                end
            end
        endcase
    end
```

```
            ACCESS: begin
                decoder_en = 1'b1;
                PENABLE    = 1'b1;
                if (ready) begin
                    decoder_en = 1'b0;
                    PENABLE    = 1'b0;
                    state_next = IDLE;
                end
            end
        endcase
    end
```

코드를 수정함으로 써 TX로 나오는 값이 2개 나오는 현상을 해결함

동작 영상





THE END