RISC-V AMBA Peripheral Design

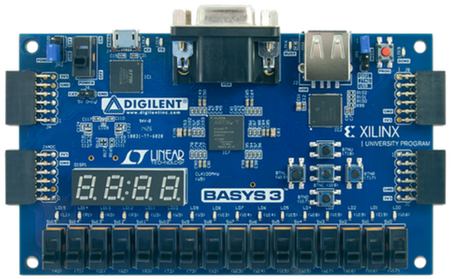
발표자 : 안유한

목차

01 프로젝트 개요

- **목표**
  - RV32I 명령어 세트를 지원하는 RISC-V 멀티사이클 CPU 코어 설계 및 검증
  - AMBA APB 버스 시스템을 구축하여 CPU와 주변장치 간의 UART 통신 인터페이스 구현
  - UART 설계하고 APB 버스에 연결하여 외부 PC(ComPortMaster)와의 시리얼 통신 기능 구현
  - 설계된 SoC(System on Chip) 시스템을 위한 제어 소프트웨어(C 코드) 개발 및 테스트

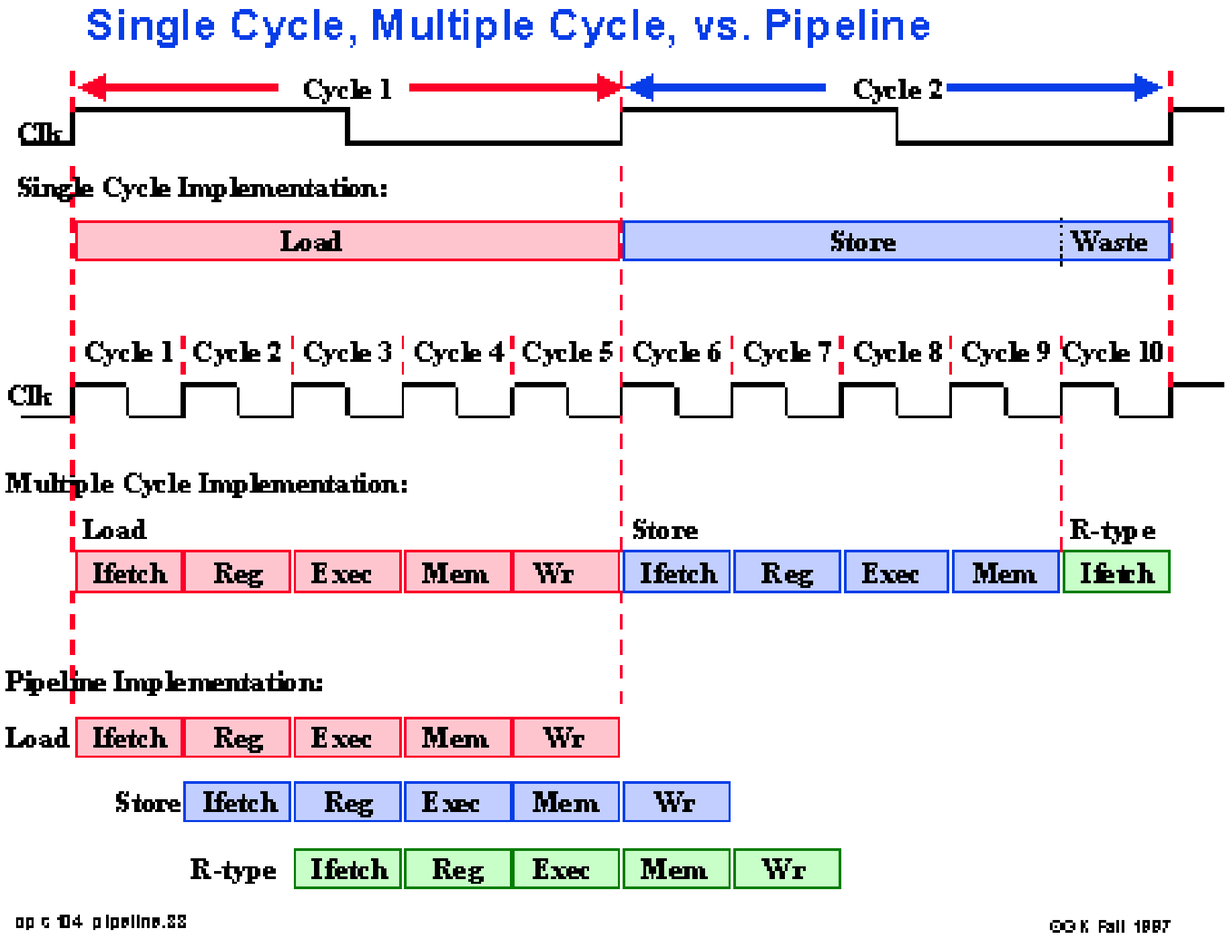
- **개발 환경**
  - 설계 언어: SystemVerilog
  - 설계 툴: Vivado
  - 합성/구현 툴: Basys 3
  - 소프트웨어 개발: C 언어
  - UART 테스트: ComPortMaster



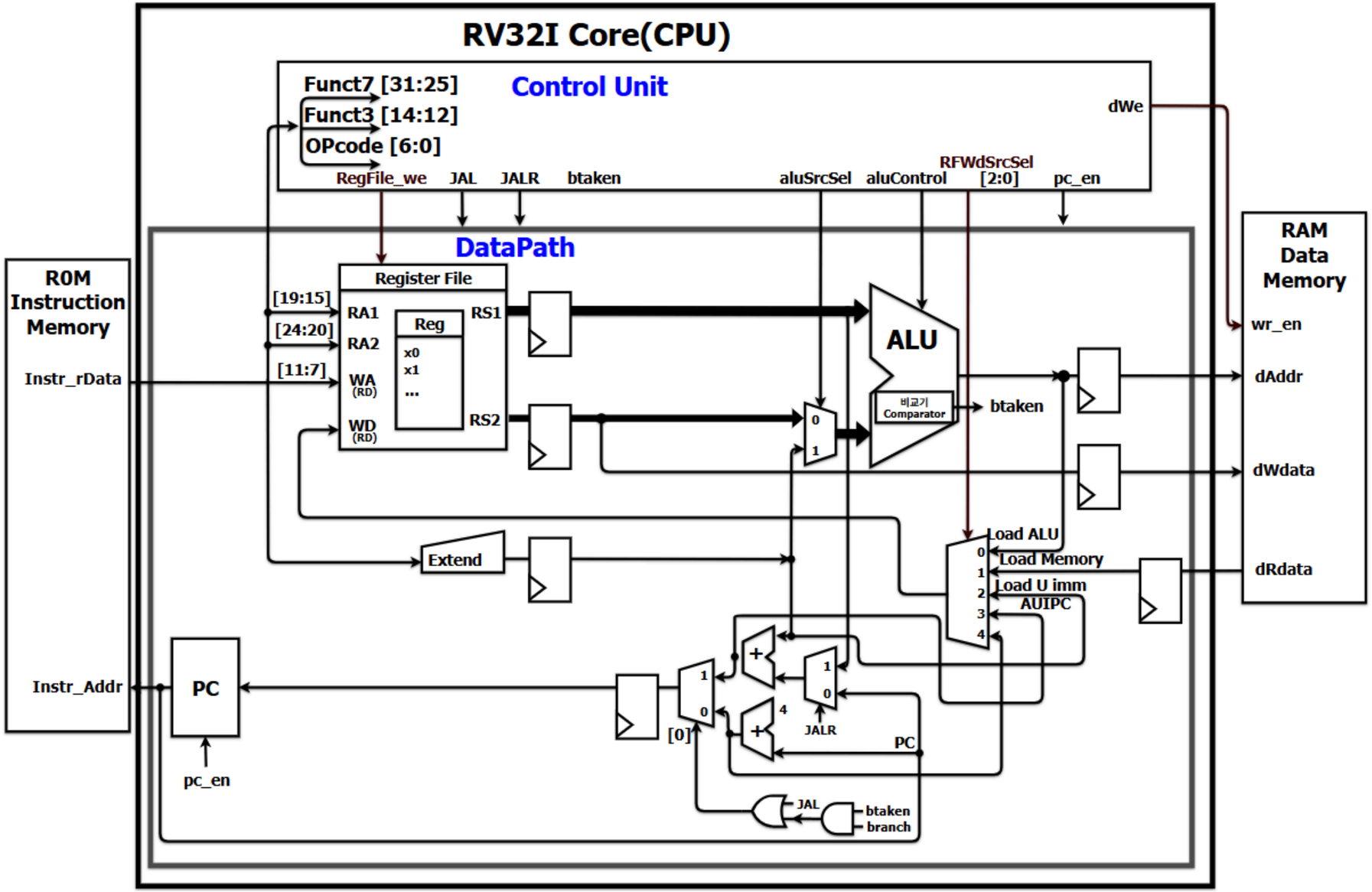
02 RISC-V Multicycle 설계

- **Single-cycle**
  - 한 사이클(clock) 동안 한 명령어를 Fetch → Decode → Execute → Memory → Write back 전부 처리
  - 이로 인해 전체 클럭이 느려짐

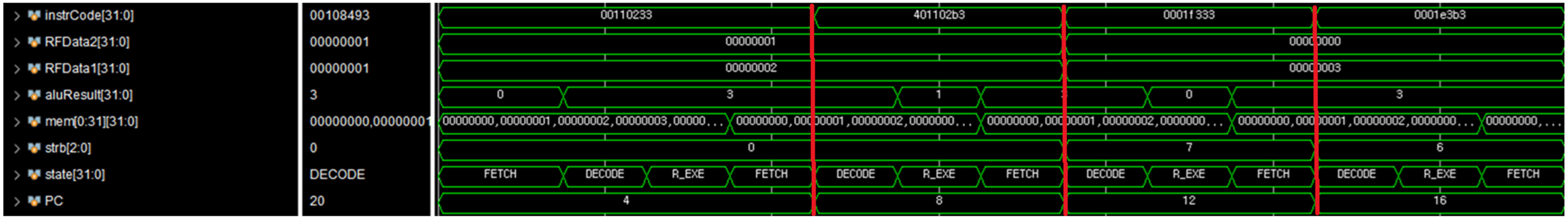
- **개선 : Multi-Cycle**
  - 하나의 명령어를 여러 개의 클럭 사이클로 분리하여 처리함
  - 각 사이클에서 수행할 동작을 FSM 상태로 구분함.



02 Block Diagram



02 시뮬레이션 - R Type




```
Assembly =   add x4, x2, x1
Binary =     0000 0000 0001 0001 0000 0010 0011 0011
Hexadecimal = 0x00110233
```

3

```
Assembly =   sub x5, x2, x1
Binary =     0100 0000 0001 0001 0000 0010 1011 0011
Hexadecimal = 0x401102b3
```

1

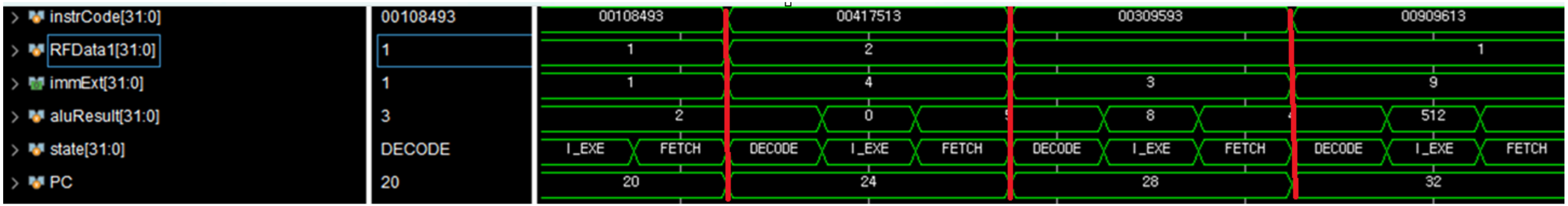
```
Assembly =   and x6, x3, x0
Binary =     0000 0000 0000 0001 1111 0011 0011 0011
Hexadecimal = 0x0001f333
```

0

```
Assembly =   or x7, x3, x0
Binary =     0000 0000 0000 0001 1110 0011 1011 0011
Hexadecimal = 0x0001e3b3
```

3

02 시뮬레이션 - I Type

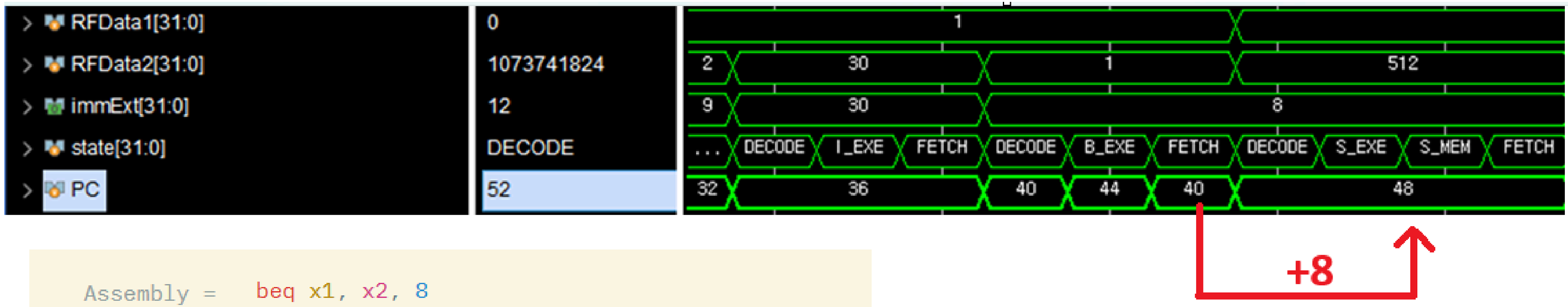



```
Assembly =    addi x9, x1, 1
Binary =      0000 0000 0001 0000 1000 0100 1001 0011
Hexadecimal = 0x00108493
```

```
Assembly =    andi x10, x2, 4
Binary =      0000 0000 0100 0001 0111 0101 0001 0011
Hexadecimal = 0x00417513
```

```
Assembly =    slli x12, x1, 9
Binary =      0000 0000 1001 0000 1001 0110 0001 0011
Hexadecimal = 0x00909613
```

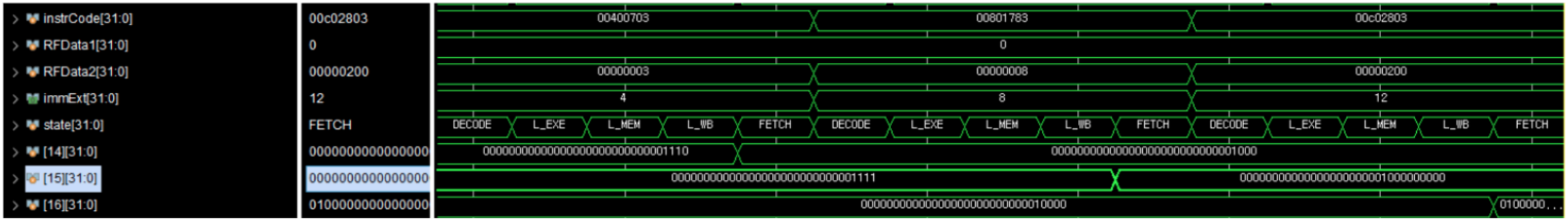
02 시뮬레이션 - B Type




```
   Assembly =   beq x1, x2, 8

     Binary =   0000 0000 0010 0000 1000 0100 0110 0011

Hexadecimal =   0x00208463
```

02 시뮬레이션 - S Type

02 시뮬레이션 - L Type




```
Assembly =   lb x14, 4(x0)
  Binary =   0000 0000 0100 0000 0000 0111 0000 0011
Hexadecimal =  0x00400703
```

```
Assembly =   lh x15, 8(x0)
  Binary =   0000 0000 1000 0000 0001 0111 1000 0011
Hexadecimal =  0x00801783
```

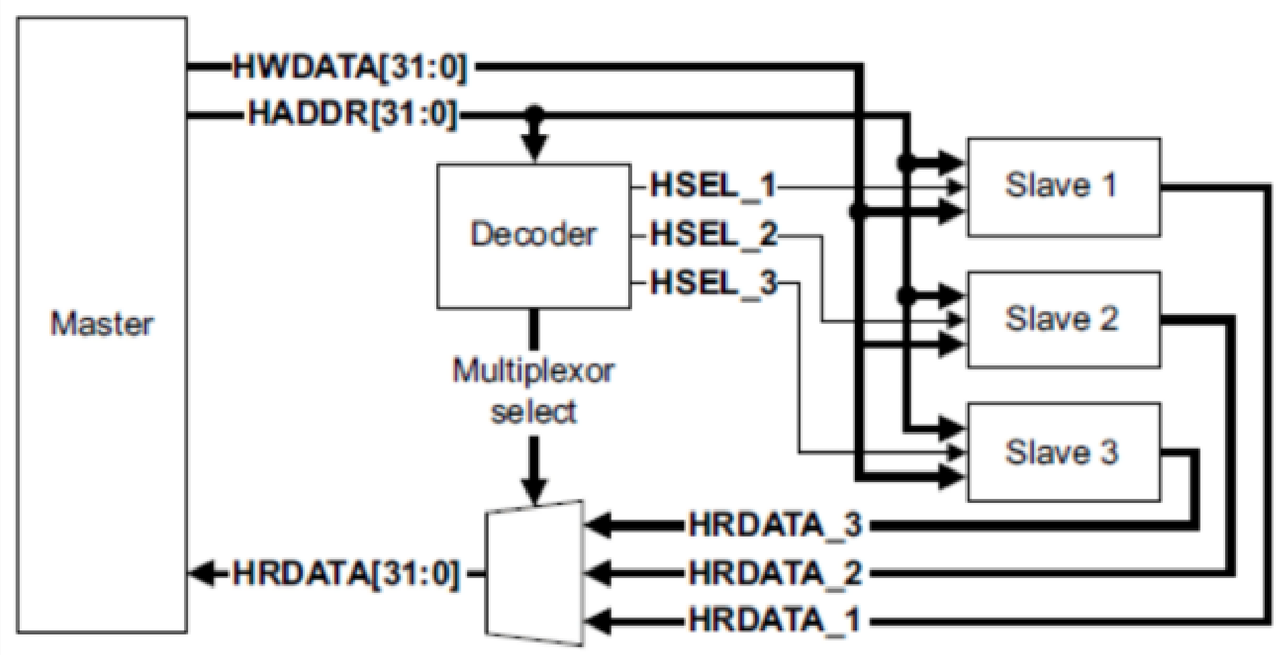
```
Assembly =   lw x16, 12(x0)
  Binary =   0000 0000 1100 0000 0010 1000 0000 0011
Hexadecimal =  0x00c02803
```

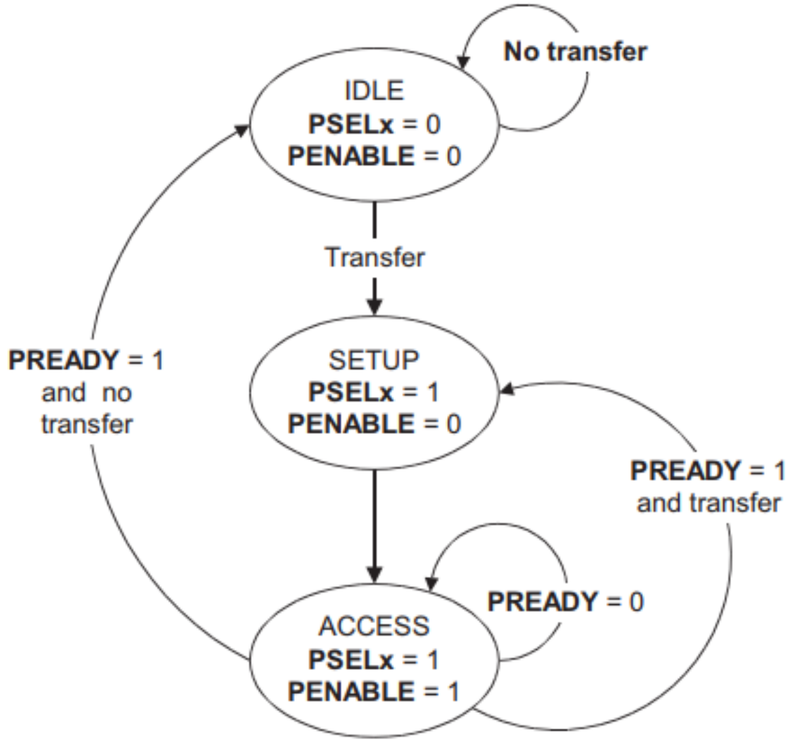
03 APB Bus 설계

- **목적**
  - MCU가 제어해야 하는 UART, FND, 센서 등 다양한 **저속 주변 장치(Peripheral)** 가 증가함.
  - AMBA APB 버스 시스템을 구축하여 CPU와 주변장치 간의 UART 통신 인터페이스 구현
  - UART 설계하고 APB 버스에 연결하여 외부 PC(ComPortMaster)와의 시리얼 통신 기능 구현
  - 설계된 SoC(System on Chip) 시스템을 위한 제어 소프트웨어(C 코드) 개발 및 테스트

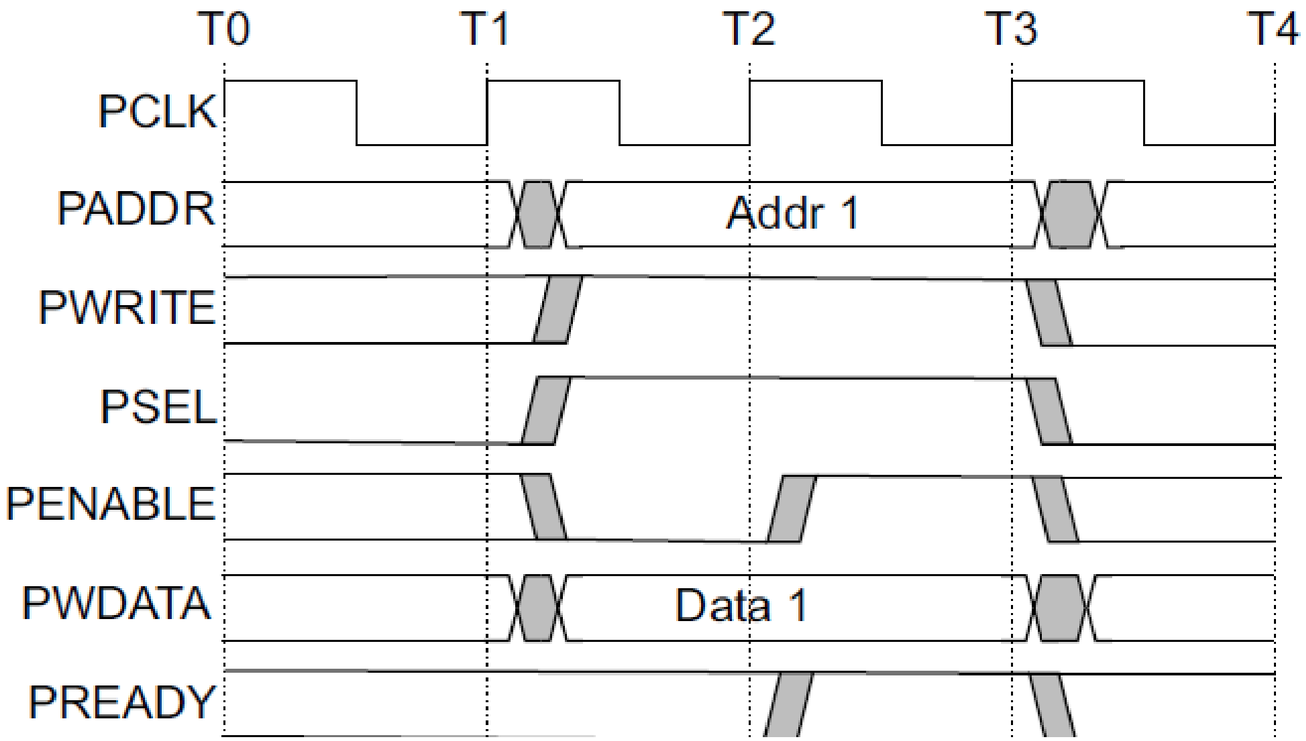
- **주변장치 제어를 단순화하고 재사용가능한 표준 인터페이스 구현 가능**

03 APB Diagram & FSM



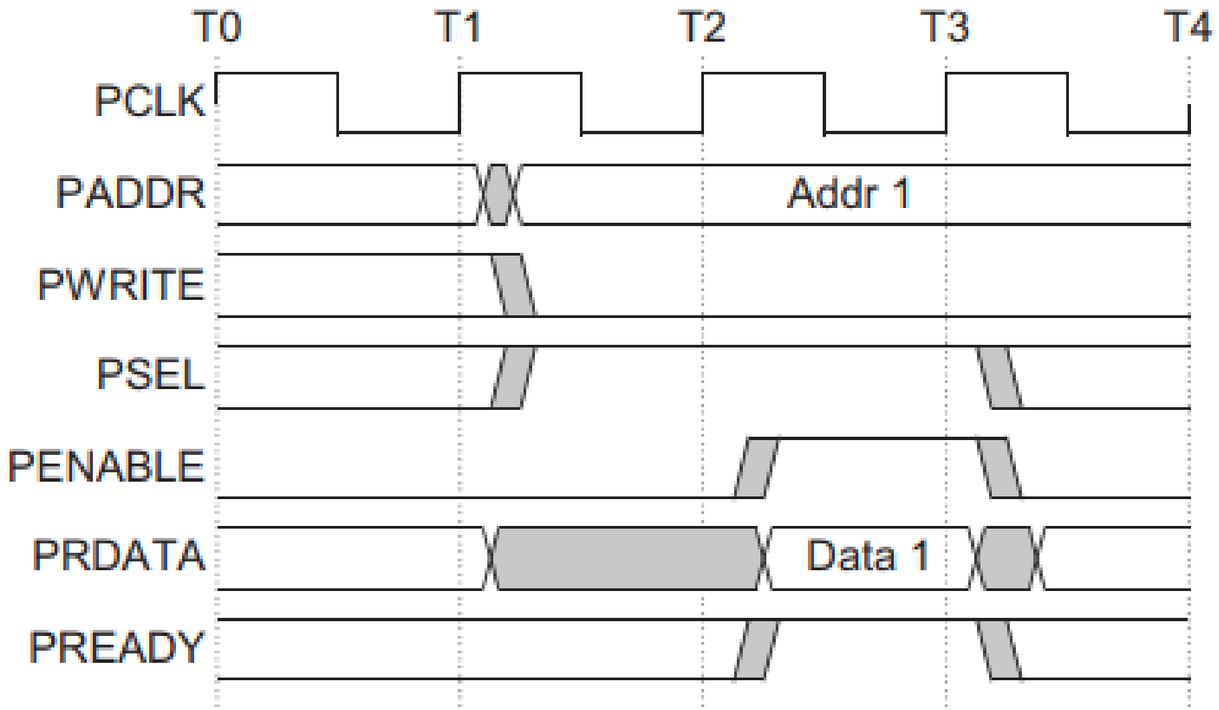


03 Write Transfer



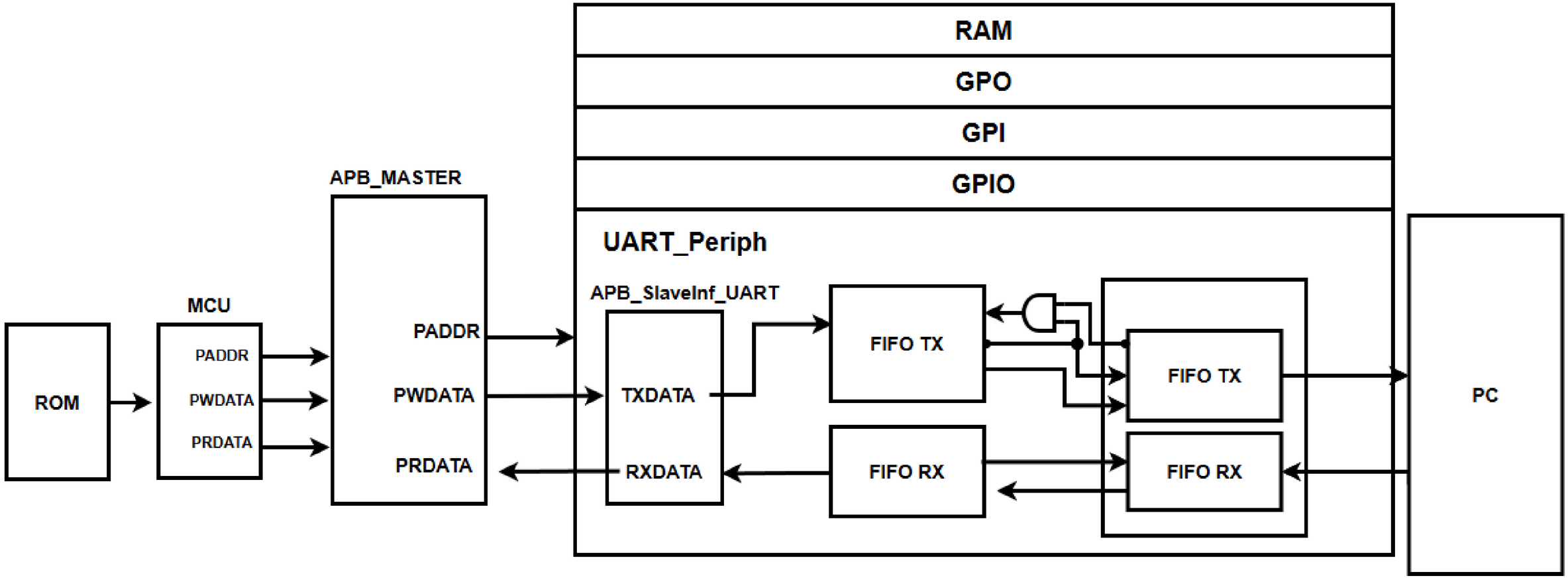
- 설정 (SETUP, T1~T2)
  - 마스터가 PSEL (슬레이브 선택), PADDR (주소), PWDATA (데이터), PWRITE (쓰기) 신호를 보냄
  - PENABLE은 아직 0 (LOW)
- 실행 (ACCESS, T2~T3)
  - 마스터가 PENABLE을 1 (HIGH)로 올려 전송을 확정.
  - 슬레이브가 준비되면 PREADY를 1 (HIGH)로 응답.
- 완료 (T3 클럭 엣지)
  - T3의 클럭 상승 엣지에서 PSEL, PENABLE, PREADY가 모두 1인 것을 확인.
  - 슬레이브는 이 순간 PWDATA를 PADDR에 실제로 저장.
- 종료 (T3~T4)
  - 마스터는 버스 사용을 끝내기 위해 PSEL과 PENABLE 신호를 0 (LOW).
  - 버스는 다음 전송을 기다리는 IDLE (대기) 상태로 돌아감.

03 Read Transfer



- 설정 (SETUP, T1~T2)
  - 마스터가 PSEL (선택), PADDR (주소), PWRITE=0 (읽기) 신호를 보냄.
  - PENABLE은 0.
- 실행 (ACCESS, T2~T3) *
  - 마스터가 PENABLE을 1 (HIGH)로 올림.
  - 슬레이브가 주소에 맞는 데이터를 PRDATA로 출력하고 PREADY를 1로 응답.
- 완료 (T3 클럭 엣지)
  - SEL, PENABLE, PREADY가 모두 1인 순간, 마스터가 PRDATA의 데이터를 가져감.
  - 마스터는 PSEL, PENABLE을 0으로 내려 버스를 해제함.

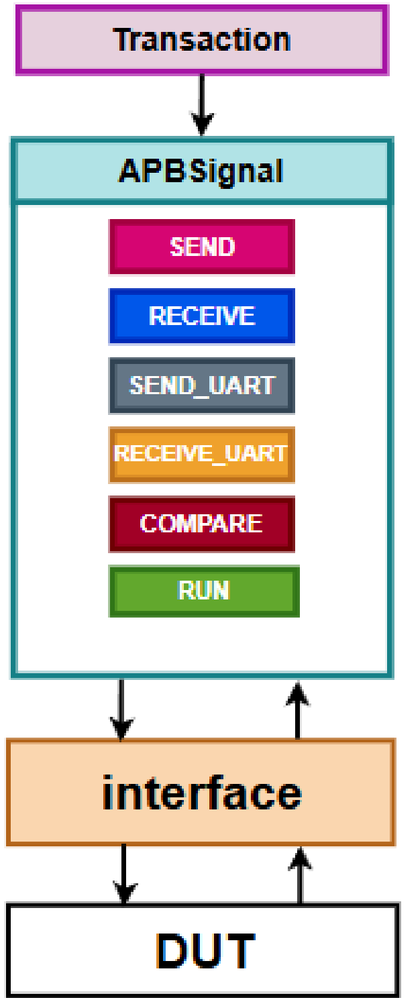
04 UART+FIFO Peripheral



04 UART+FIFO Peripheral

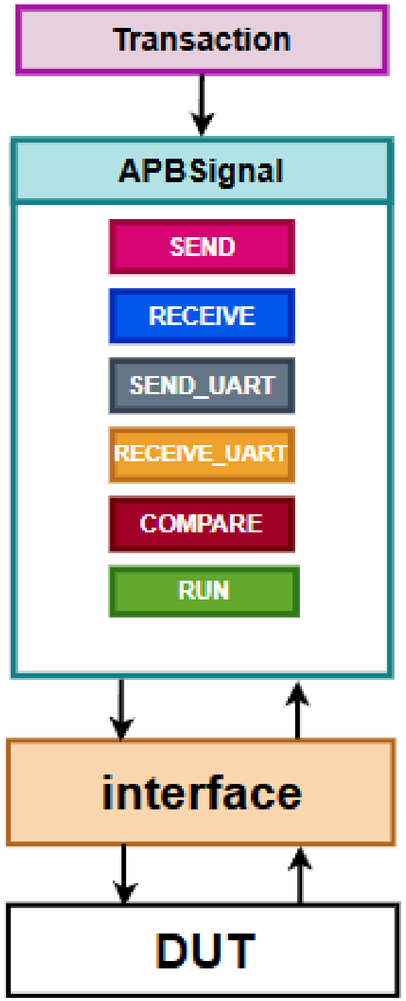
Address (PADDR)	Register Name	Access Type	Bit Field 설명	기능 요약
0x0	TxData (Transmit Data)	Write	[7:0] 전송할 데이터	CPU가 UART로 데이터를 보낼 때 작성하는 레지스터.
0x4	RxData (Receive Data)	Read Only	[7:0] 수신된 데이터	UART가 수신한 데이터가 저장됨. CPU가 읽을 때 o_rx_pop 신호 발생
0x8	Status Register	Read Only	[0] RxValid (1: 수신 데이터 있음) [1] TxFull (1: 전송 불가, FIFO Full)	UART 동작 상태 확인용 레지스터

04 UART Verification



- Transaction (트랜잭션): 테스트에 필요한 데이터를 담아둠
- APBSignal (테스트 실행기): 실제 테스트 시나리오.
  - SEND / RECEIVE: APB 버스를 통해 DUT의 레지스터에 값을 쓰거나 읽음.
  - SEND_UART / RECEIVE_UART: DUT의 UART 핀(rx/tx)에 데이터를 보내거나 받음
  - COMPARE: 보낸 데이터와 받은 데이터가 일치하는지 비교하여 PASS/FAIL을 판정
  - RUN: 위의 기능들을 순서대로 조합하여 전체 테스트를 실행
- interface (연결 통로): APBSignal 클래스(소프트웨어)와 DUT(하드웨어) 사이를 연결
- DUT (검증 대상): 테스트하려는 실제 하드웨어 모듈, 즉 APB-UART임

04 UART Verification

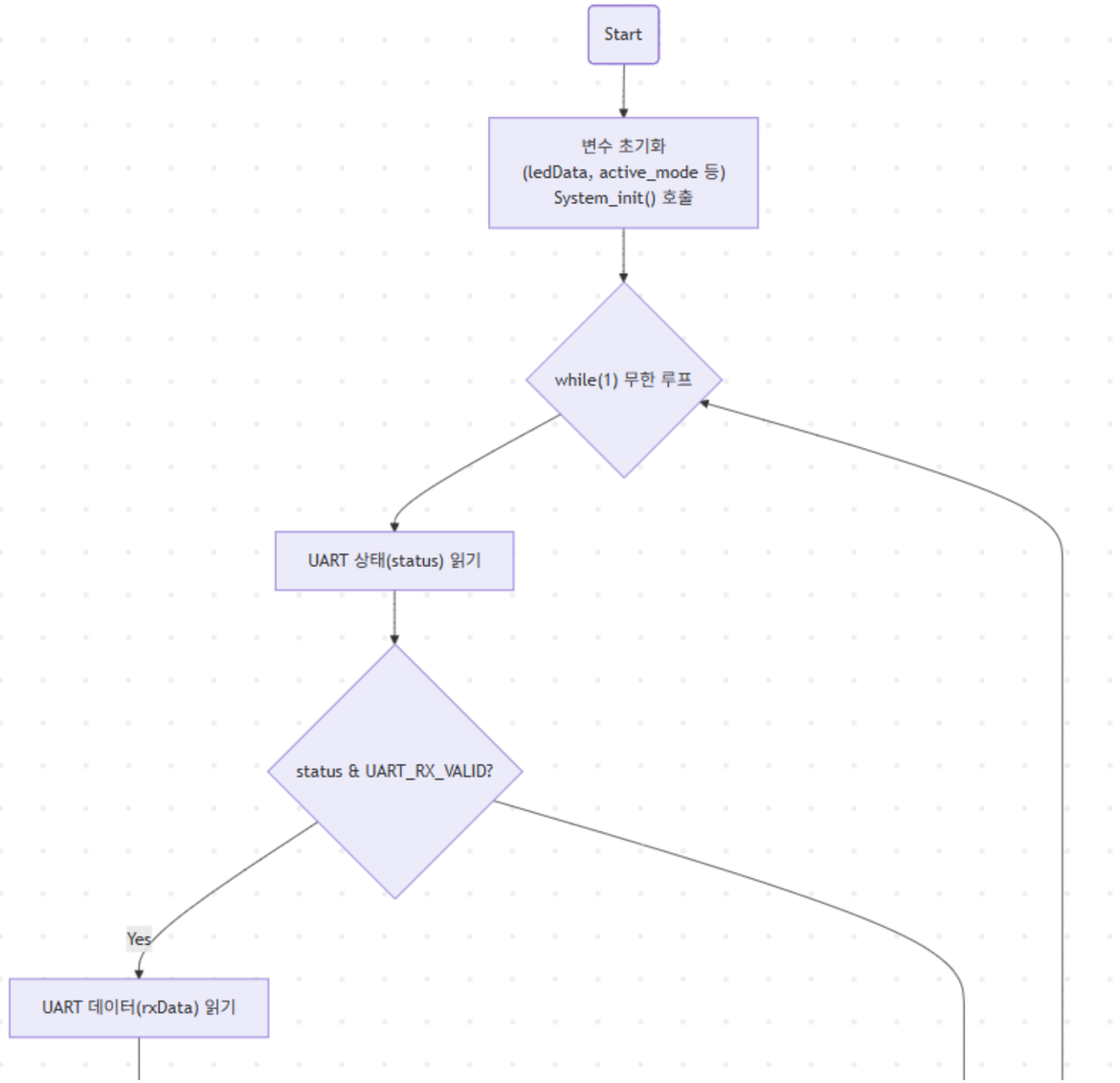



```
========== Test #96 ==========
[192016575000][APB_WRITE] addr=10004000 wdata=000000a1 rdata=bbcd4008 send=90 recv=a0
[192118435000][UART_TX_RECV] addr=10004000 wdata=000000a1 rdata=bbcd4008 send=90 recv=a0
[193016575000][UART_RX_SEND] addr=10004000 wdata=000000a1 rdata=bbcd4008 send=90 recv=a1
[194016625000][APB_READ] addr=10004004 wdata=000000a1 rdata=bbcd4004 send=90 recv=a1
[194016685000][APB_READ] addr=10004008 wdata=000000a1 rdata=bbcd4008 send=90 recv=a1
[PASS] TX=a1 == RX=a1

========== Test #97 ==========
[194016745000][APB_WRITE] addr=10004000 wdata=000000a2 rdata=bbcd4008 send=91 recv=a1
[194224395000][UART_TX_RECV] addr=10004000 wdata=000000a2 rdata=bbcd4008 send=91 recv=a1
[195016745000][UART_RX_SEND] addr=10004000 wdata=000000a2 rdata=bbcd4008 send=91 recv=a2
[196016795000][APB_READ] addr=10004004 wdata=000000a2 rdata=bbcd4004 send=91 recv=a2
[196016855000][APB_READ] addr=10004008 wdata=000000a2 rdata=bbcd4008 send=91 recv=a2
[PASS] TX=a2 == RX=a2

========== Test #98 ==========
[196016915000][APB_WRITE] addr=10004000 wdata=000000a3 rdata=bbcd4008 send=92 recv=a2
[196115195000][UART_TX_RECV] addr=10004000 wdata=000000a3 rdata=bbcd4008 send=92 recv=a2
[197016915000][UART_RX_SEND] addr=10004000 wdata=000000a3 rdata=bbcd4008 send=92 recv=a3
[198016965000][APB_READ] addr=10004004 wdata=000000a3 rdata=bbcd4004 send=92 recv=a3
[198017025000][APB_READ] addr=10004008 wdata=000000a3 rdata=bbcd4008 send=92 recv=a3
[PASS] TX=a3 == RX=a3
```

```
========== TEST SUMMARY ==========
PASS COUNT = 100
FAIL COUNT = 0
===  ALL TEST PASSED ===
==================================
```

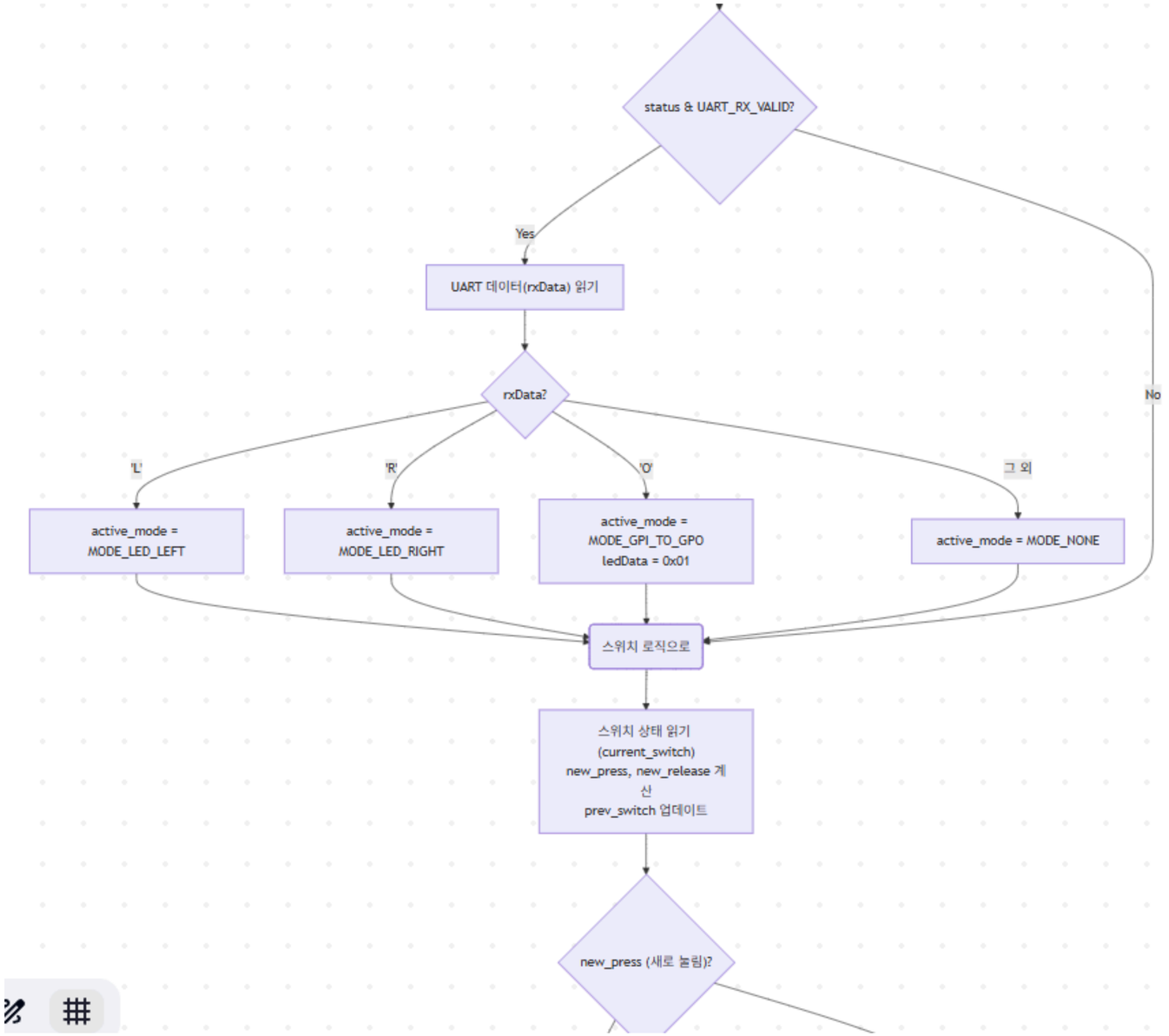
05 C Code

```
// uart 상시 동작
status = UART_STATUS;
if(status & UART_RX_VALID){
    rxData = UART_RXDATA & 0xff;
    // [수정] 수신된 데이터(rxData)에 따라 모드 변경
    if (rxData == 'L') {
        active_mode = MODE_LED_LEFT;
        LED_write(ledData); // 모드 변경 시 즉시 반영

    } else if (rxData == 'R') {
        active_mode = MODE_LED_RIGHT;
        LED_write(ledData); // 모드 변경 시 즉시 반영
    } else if (rxData == 'O') {
        active_mode = MODE_GPI_TO_GPO;
        ledData = 0x01;
    } else {
        // 'L', 'R', 'O' 외 다른 문자를 받으면 끔
        active_mode = MODE_NONE;
    }

}
```



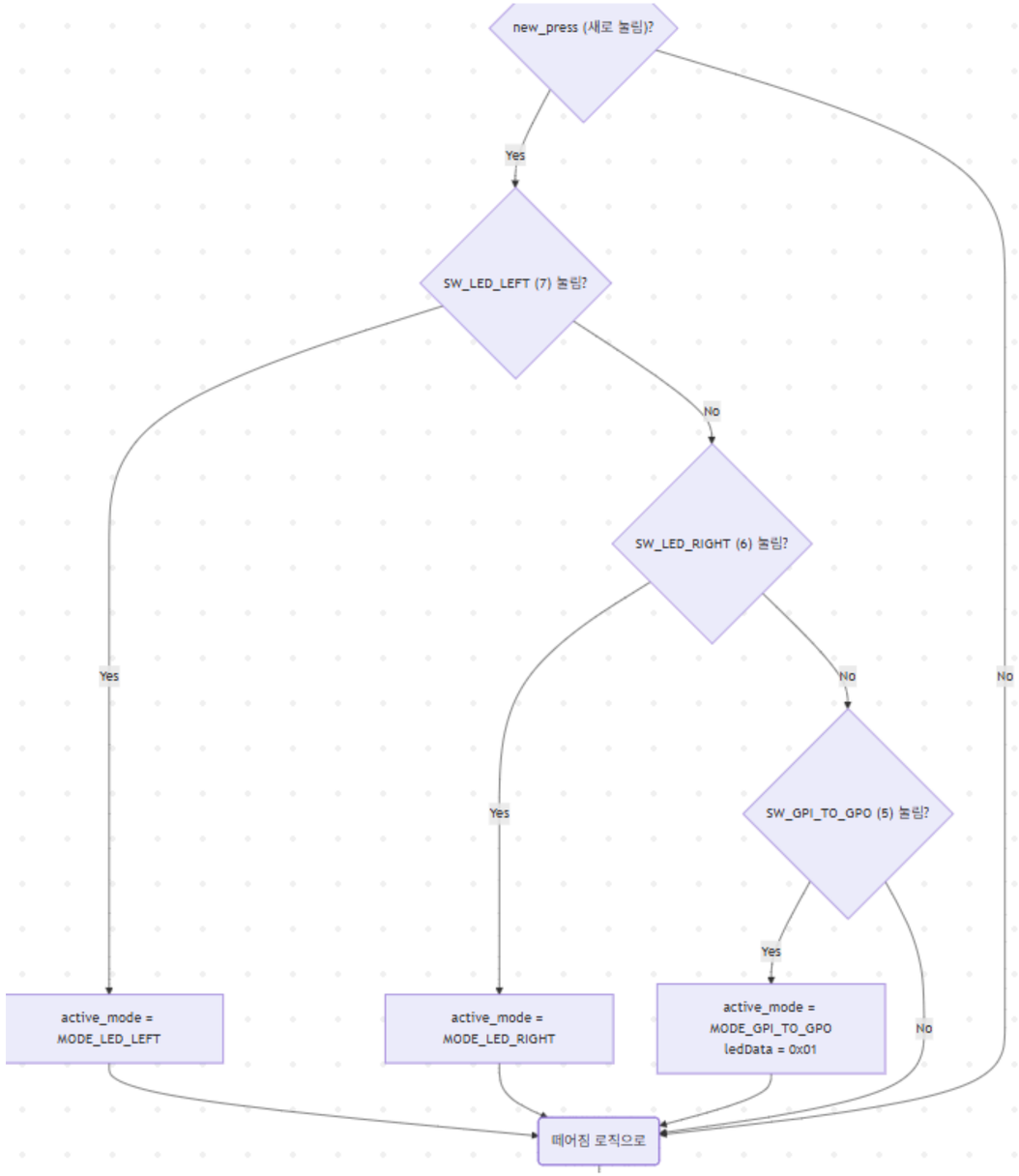
05 C Code

```
// uart 상시 동작
status = UART_STATUS;
if(status & UART_RX_VALID){
    rxData = UART_RXDATA & 0xff;
    // [수정] 수신된 데이터(rxData)에 따라 모드 변경
    if (rxData == 'L') {
        active_mode = MODE_LED_LEFT;
        LED_write(ledData); // 모드 변경 시 즉시 반영

    } else if (rxData == 'R') {
        active_mode = MODE_LED_RIGHT;
        LED_write(ledData); // 모드 변경 시 즉시 반영
    } else if (rxData == 'O') {
        active_mode = MODE_GPI_TO_GPO;
        ledData = 0x01;
    } else {
        // 'L', 'R', 'O' 외 다른 문자를 받으면 끔
        active_mode = MODE_NONE;
    }

}
```



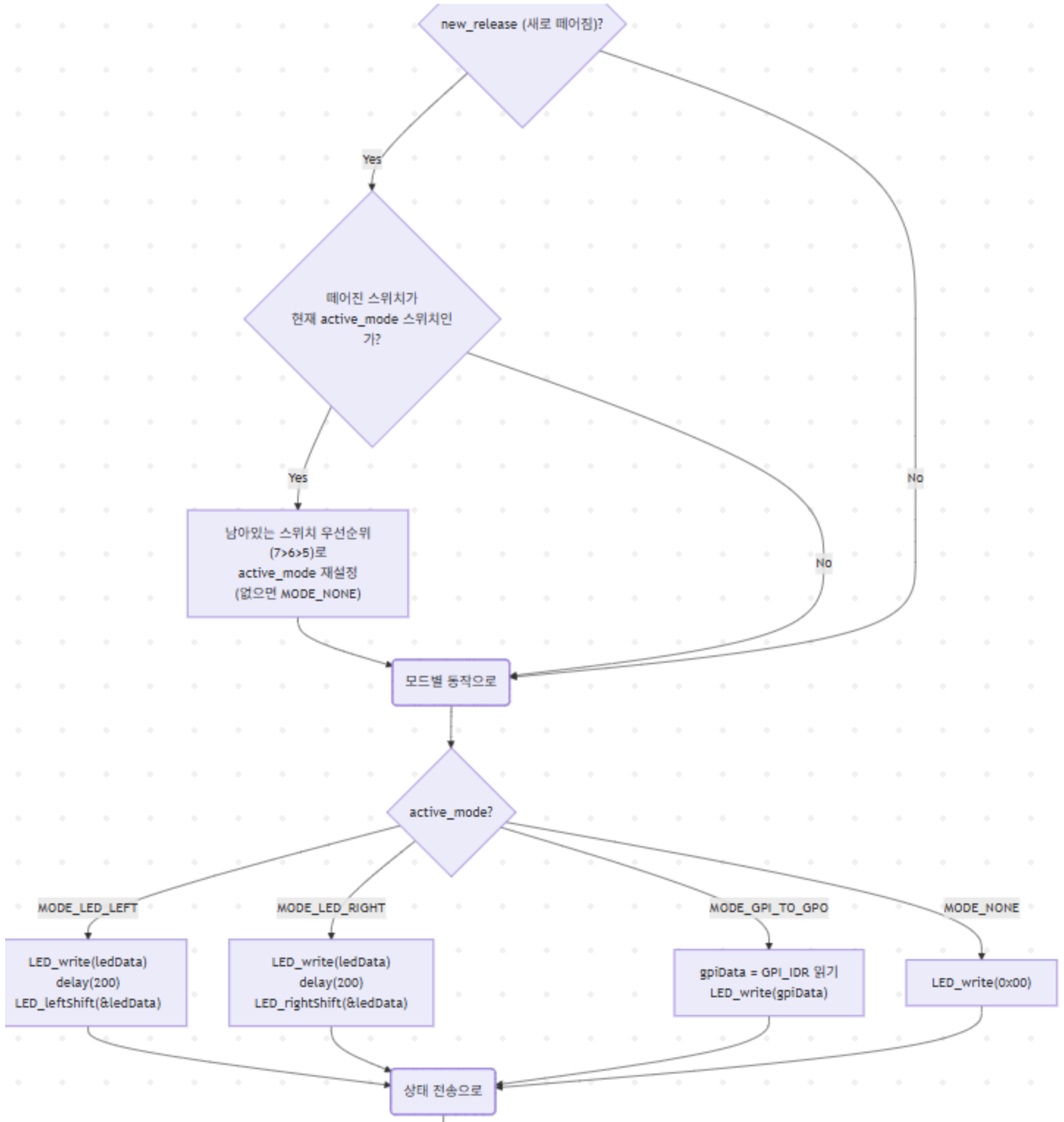
05 C Code

```c
// 모드 변경 감지
uint32_t current_switch = MODE_SWITCH_PORT;
uint32_t new_press = current_switch & ~prev_switch; // 새로 눌린 스위치
uint32_t new_release = ~current_switch & prev_switch; // 새로 떼어진 스위치
prev_switch = current_switch; // 이전 스위치 업데이트

// 새로 눌린 스위치가 있으면, 그 스위치로 모드 변경 (우선순위: 7 > 6 > 5)
if (new_press & SW_LED_LEFT) {
    active_mode = MODE_LED_LEFT;
} else if (new_press & SW_LED_RIGHT) {
    active_mode = MODE_LED_RIGHT;
} else if (new_press & SW_GPI_TO_GPO) {
    active_mode = MODE_GPI_TO_GPO;
    ledData = 0x01;
}
```



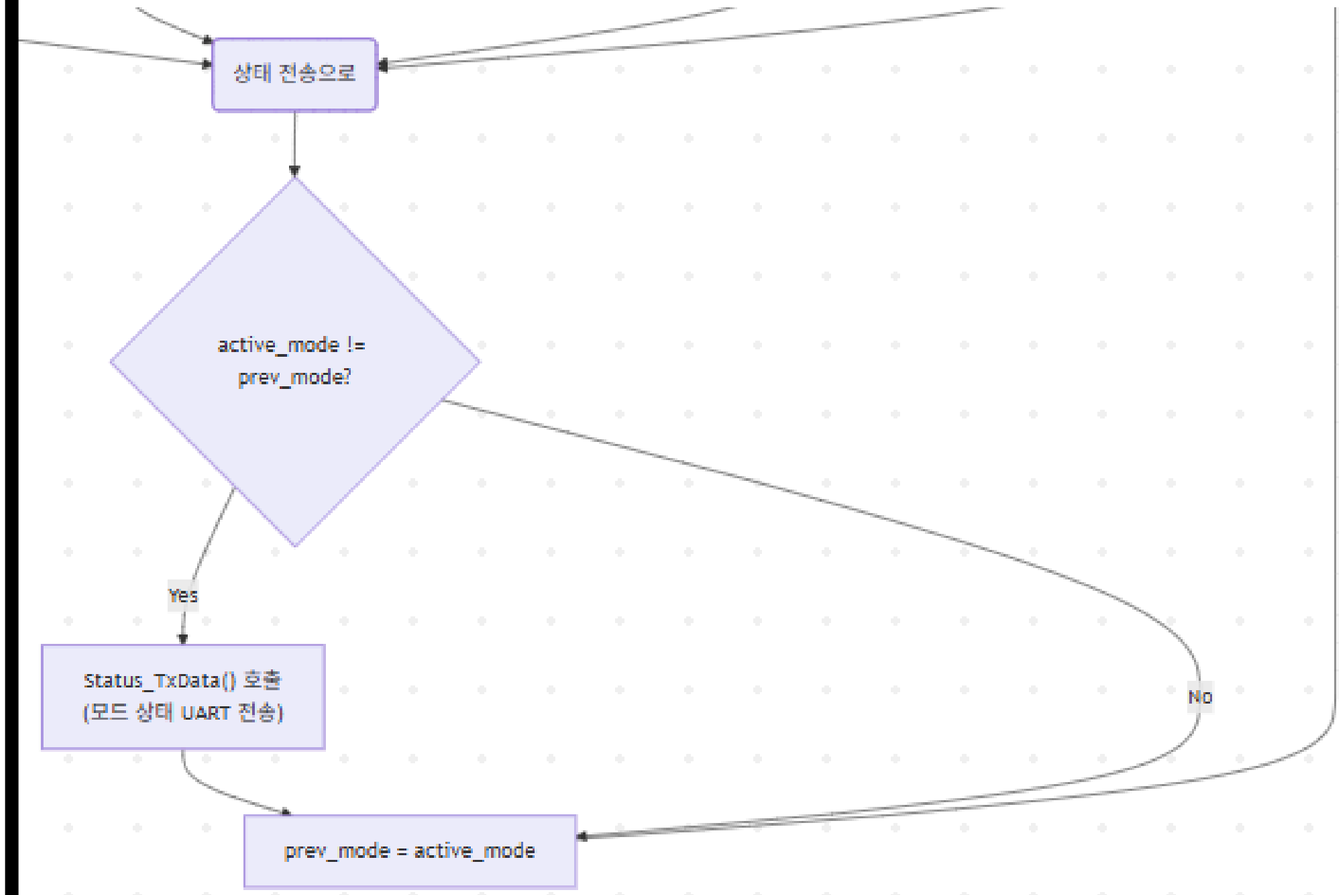
05 C Code

```c
// 스위치가 떼어졌을 때, 떼어진 스위치가 현재 활성 모드였다면?
if (new_release) {
    uint32_t released_mode_sw = 0;
    if (active_mode == MODE_LED_LEFT) released_mode_sw = SW_LED_LEFT;
    else if (active_mode == MODE_LED_RIGHT) released_mode_sw = SW_LED_RIGHT;
    else if (active_mode == MODE_GPI_TO_GPO) released_mode_sw = SW_GPI_TO_GPO;
    if(new_release & released_mode_sw) {
        // 떼어진 스위치가 현재 모드의 스위치라면, 다른 켜진 스위치로 모드 변경
        // 우선순위: 7 > 6 > 5
        if (current_switch & SW_LED_LEFT) {
            active_mode = MODE_LED_LEFT;
        } else if (current_switch & SW_LED_RIGHT) {
            active_mode = MODE_LED_RIGHT;
        } else if (current_switch & SW_GPI_TO_GPO) {
            active_mode = MODE_GPI_TO_GPO;
            ledData = 0x01;
        } else {
            active_mode = MODE_NONE; // 켜진 스위치가 아무것도 없으면 기본 모드
            ledData = 0x01;
        }
    }
}
```

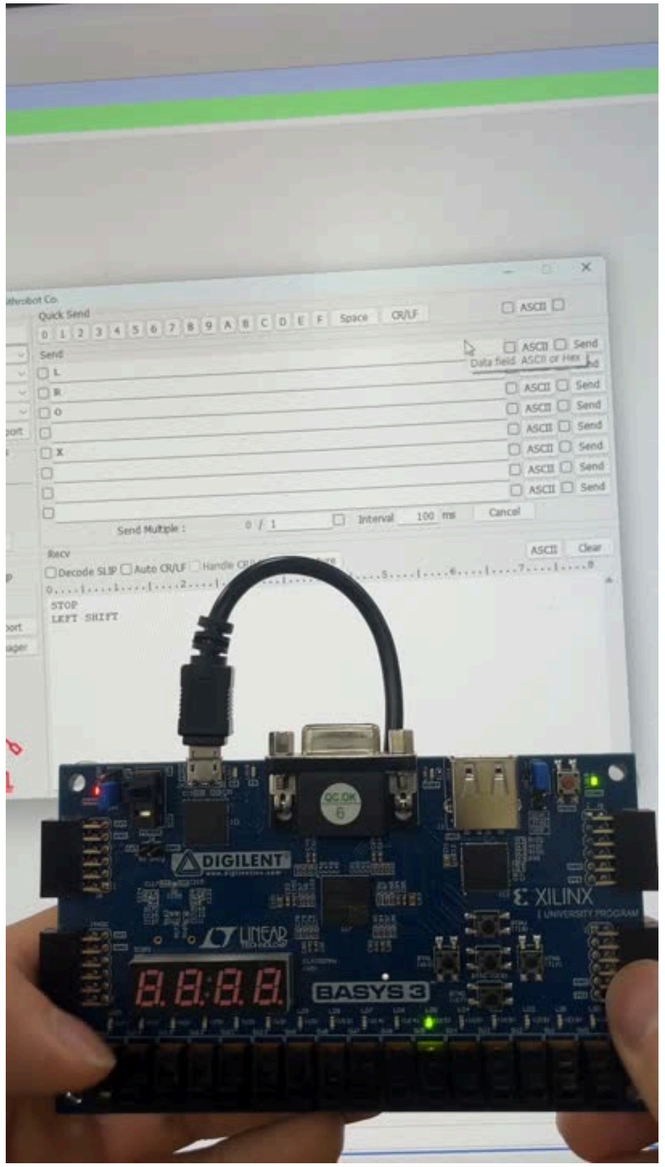


05 C Code

```c
// active_mode에 따른 동작 수행
if(active_mode == MODE_LED_LEFT){
    LED_write(ledData);
    delay(200);
    LED_leftShift(&ledData);

} else if(active_mode == MODE_LED_RIGHT){
    LED_write(ledData);
    delay(200);
    LED_rightShift(&ledData);

} else if(active_mode == MODE_GPI_TO_GPO){
    uint32_t gpiData = (*(uint32_t *)&GPI_IDR) & 0xff;
    LED_write(gpiData);

} else {
    // 모든 모드 비활성화 시 LED 끄기
    LED_write(0x00);
}

if(active_mode != prev_mode){
    if(active_mode == MODE_LED_LEFT) Status_TxData('L');
    else if(active_mode == MODE_LED_RIGHT) Status_TxData('R');
    else if(active_mode == MODE_GPI_TO_GPO) Status_TxData('O');
    else Status_TxData('S');


}
prev_mode = active_mode; // 이전 모드 업데이트
```



05 동작 영상


Quick Send
Space
CR/LF
ASCII
Send
Send Multiple :
Interval
100 ms
Cancel
Recv
Decode SLIP
Auto CR/LF
Clear
STOP
LEFT SHIFT
DIGILENT
XILINX
BASYS 3

06 고찰

- 이번 프로젝트는 RV32I CPU 코어에 APB 버스와 UART, GPIO 등 직접 설계한 주변 장치를 연결하여 MCU 수준의 통합 시스템을 구현하는 경험이었습니다.

- APB 프로토콜 학습과 FIFO를 이용한 하드웨어 설계, 그리고 C 코드를 통한 소프트웨어 제어를 함께 경험하며 시스템 동작 원리를 체감했습니다.

- 특히 APB 타이밍과 UART FIFO 버그 등 실제 설계 이슈를 테스트벤치와 C 코드 디버깅으로 해결하면서, 검증의 중요성과 하드웨어/소프트웨어 통합 설계 능력의 필요성을 느꼈습니다.