

# SoC Timer 와 PWM & Interrupt

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. Timer와 Counter의 개념 이해 및 구조
  - 타이머 주변 장치의 일반적인 아키텍처와 동작 모드
    - Timer 이해하기
    - Timer 제어 및 레지스터맵 이해 하기
    - PWM 이해하기
  - Timer Peripheral 시뮬레이션
- 2. Interrupt 이해하기 및 시뮬레이션
  - Interrupt 발생기 이해
    - Interrupt 제어구조 및 레지스터맵 이해
  - Interrupt 시뮬레이션
- Q&A

# SoC Timer & PWM

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.

# Learning Objectives

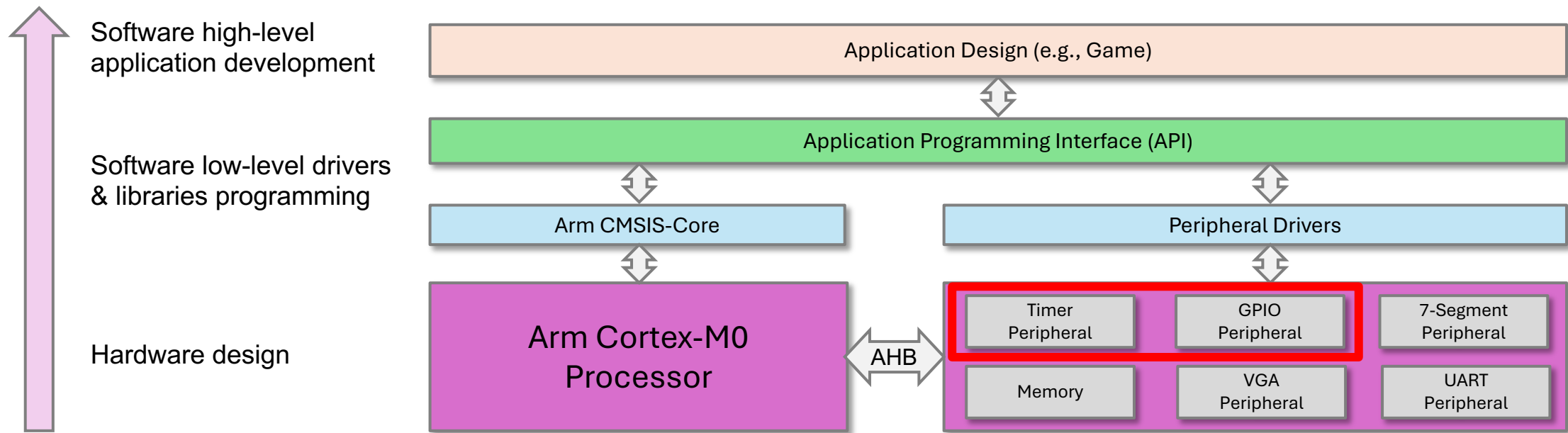
이 강의가 끝날 때, 여러분은 다음을 할 수 있어야 합니다:

- 타이머의 기능을 설명하고 하드웨어 타이머와 소프트웨어 타이머의 기능을 비교할 수 있다.
- 하드웨어 타이머의 표준 아키텍처에서 각 구성 요소의 기능을 설명할 수 있다.
- 표준 타이머의 세 가지 작동 모드('비교 모드', '캡처 모드', '펄스 폭 변조 모드')를 설명할 수 있다.
- AHB 타이머 주변 장치의 구성 요소를 개략적으로 설명하고 그들의 기능을 설명할 수 있다.

# Timer Overview

- 하드웨어 타이머 디지털 카운터로서 의 특징
  - 규칙적인 이벤트를 카운트하며, 일반적으로 비교적 높고 고정된 주파수를 가진 클록 소스를 참조.
  - 고정된 주파수로 증가하거나 감소.
  - 0 또는 미리 정의된 값에 도달하면 스스로 리셋 됨.
  - 리셋될 때 인터럽트를 생성.
- 반면, 소프트웨어 타이머는 유사한 기능 블록이지만 소프트웨어로 구현됨.  
소프트웨어 타이머의 특징
  - 하드웨어 타이머를 기반으로 합니다.
  - 하드웨어 타이머에 의해 인터럽트 될 때 증가하거나 감소.
  - 하드웨어 타이머에 비해 시간 정밀도가 낮음.
  - 실제 하드웨어 타이머보다 더 많은 인스턴스를 가짐.

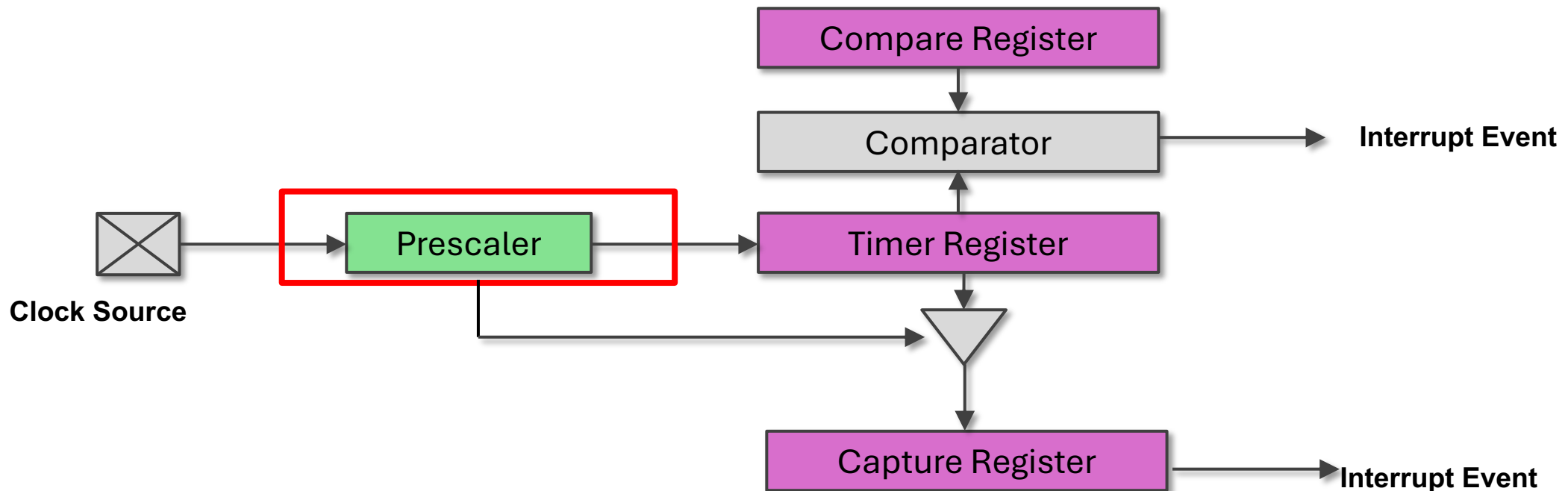
# Building a System on a Chip (SoC)



# Standard Architecture of Hardware Timers

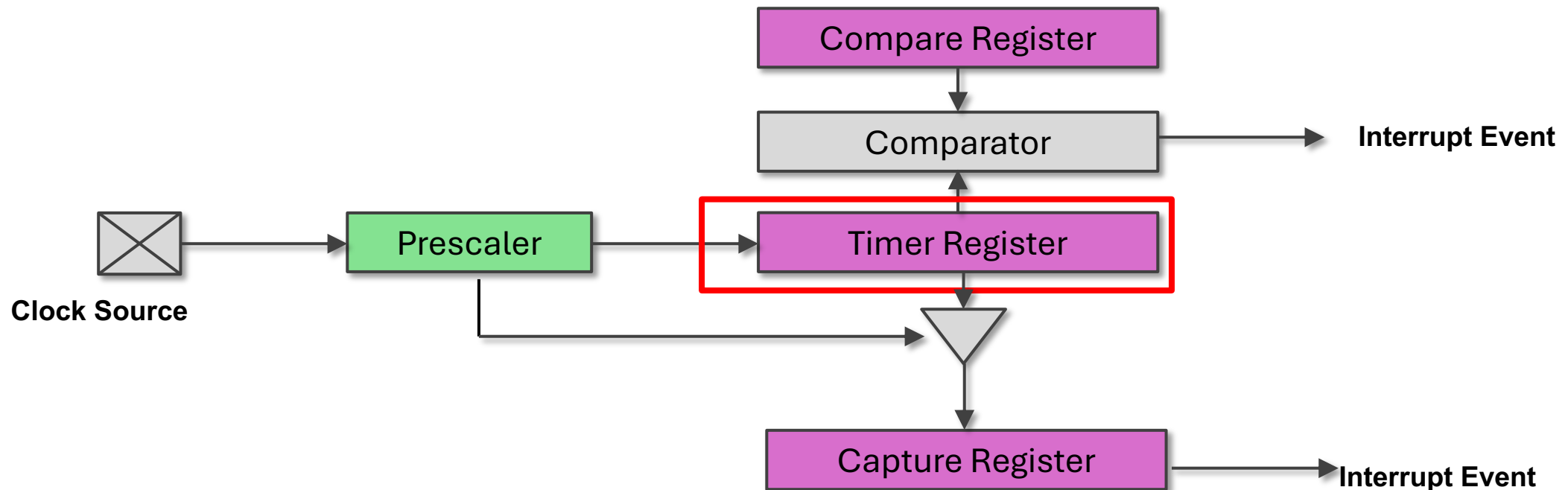
프리스케일러(Prescaler)는 다음과 같은 역할을 합니다:

- 클록 소스를 입력받음.
- 입력 주파수를 미리 정의된 값 (예: 4, 8, 16)으로 나눔.
- 나뉜 주파수를 다른 구성 요소에 출력



# Standard Architecture of Hardware Timers

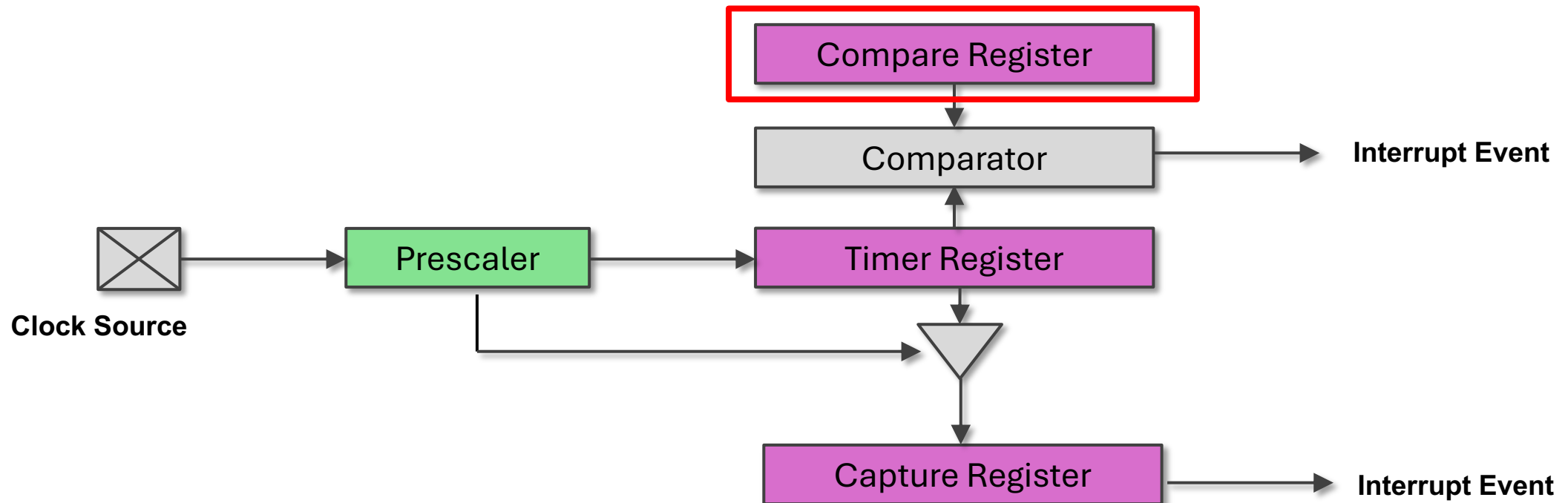
- 타이머 레지스터의 역할:
  - 고정된 주파수로 증가하거나 감소.
  - 프리스케일러의 출력(종종 "틱(ticks)"이라고 불림)에 의해 구동 됨.





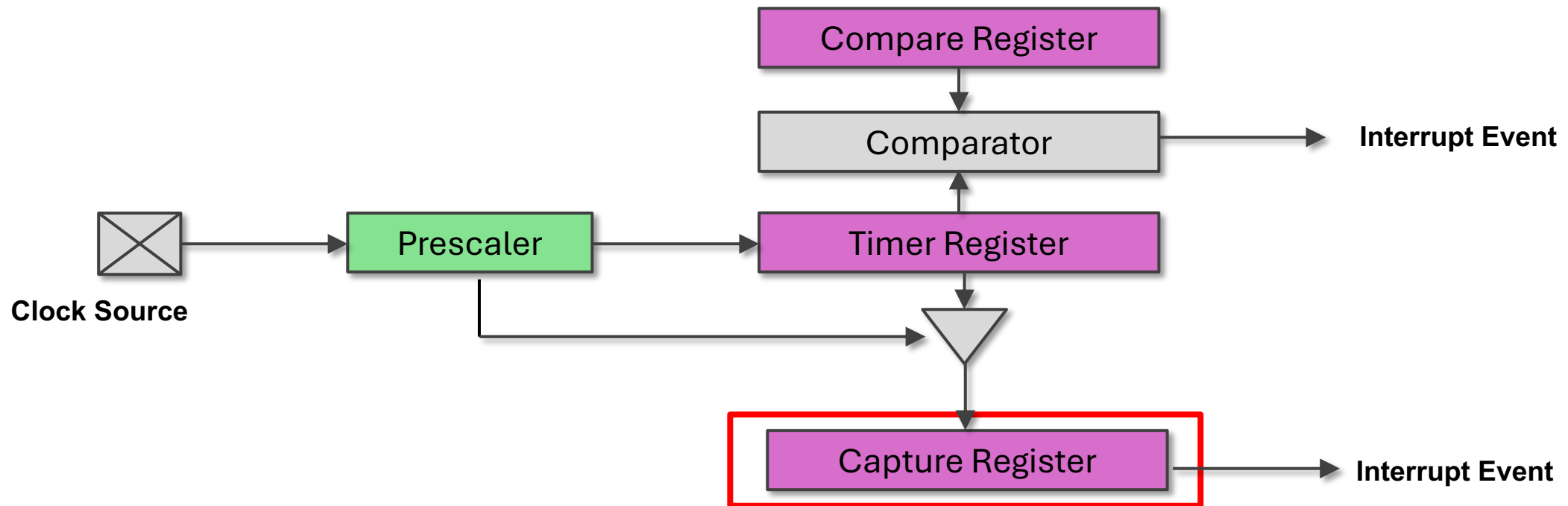
# Standard Architecture of Hardware Timers

- 비교 레지스터(Compare Register)는 다음과 같은 역할을 합니다:
  - 원하는 값으로 미리 로드되며, 이 값은 주기적으로 타이머 레지스터의 값과 비교됩니다.
  - 두 값이 동일할 경우 인터럽트를 생성할 수 있습니다.



# Standard Architecture of Hardware Timers

- 캡처 레지스터(Capture Register)의 역할:
  - 특정 이벤트가 발생하면 타이머 레지스터의 현재 값을 로드 함.
  - 특정 이벤트가 발생하면 인터럽트를 생성할 수도 있음.



# Timer Operation Modes

- Compare mode

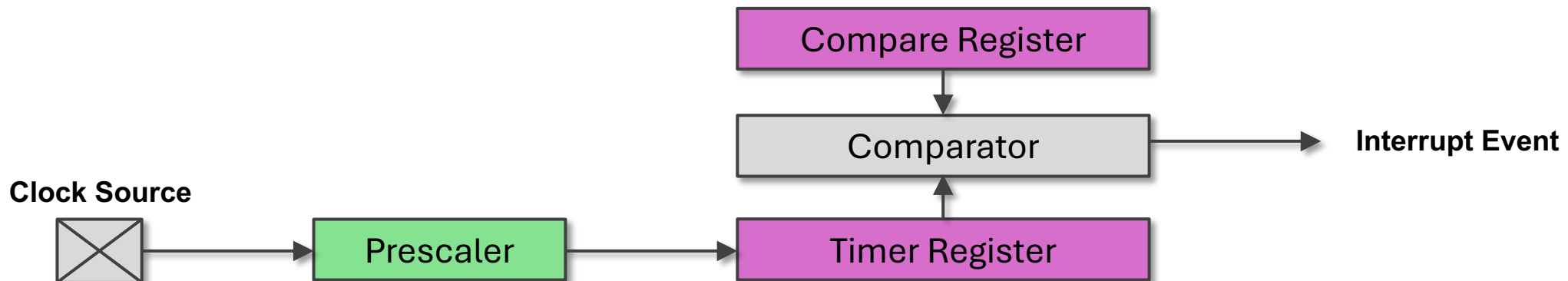
비교 레지스터에  
원하는 값으로  
미리 로드합니다.

타이머  
레지스터를  
증가시키거나  
감소시킵니다.

타이머  
레지스터의 값을  
비교합니다.

타이머  
레지스터와 비교  
레지스터의 값이  
동일해지면,  
인터럽트 신호를  
생성합니다.

타이머  
레지스터를 리셋  
합니다.



# Timer Operation Modes

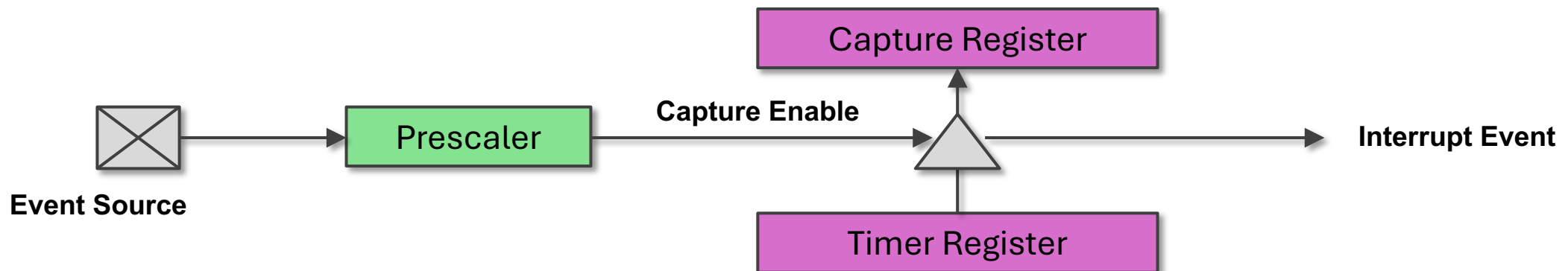
- Capture mode

**타이머 시작:**  
타이머 레지스터는 프리스케일러 출력 신호에 따라 증가하거나 감소하며 실행.

**캡처 설정:**  
캡처 모드를 활성화하고, 특정 엣지(상승 또는 하강)를 트리거 조건으로 설정

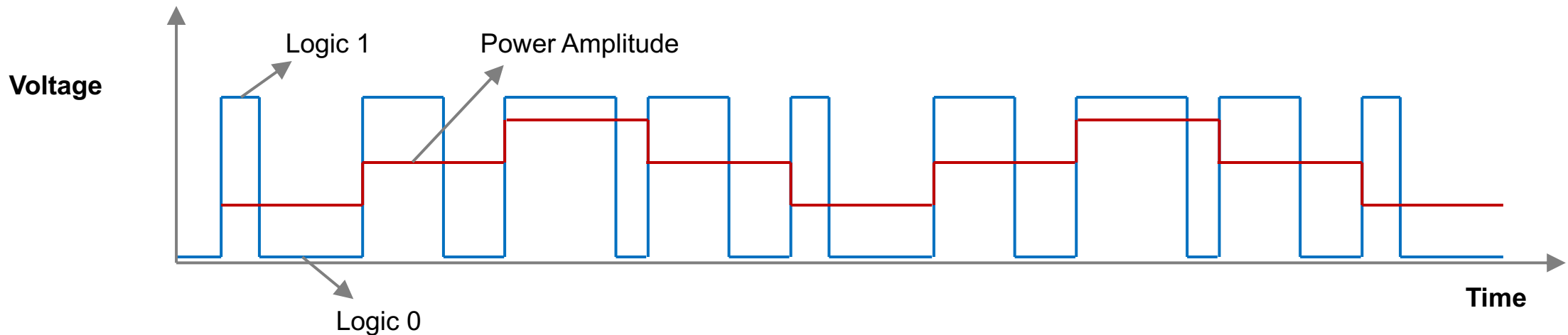
**캡처(SanpShot) 발생:**  
설정된 조건에 따라 이벤트가 발생하면 타이머 레지스터의 현재 값이 캡처 레지스터에 저장.

**인터럽트 처리:**  
캡처 완료 시 인터럽트가 생성되고, 프로세서는 이를 처리하여 필요한 작업을 수행.



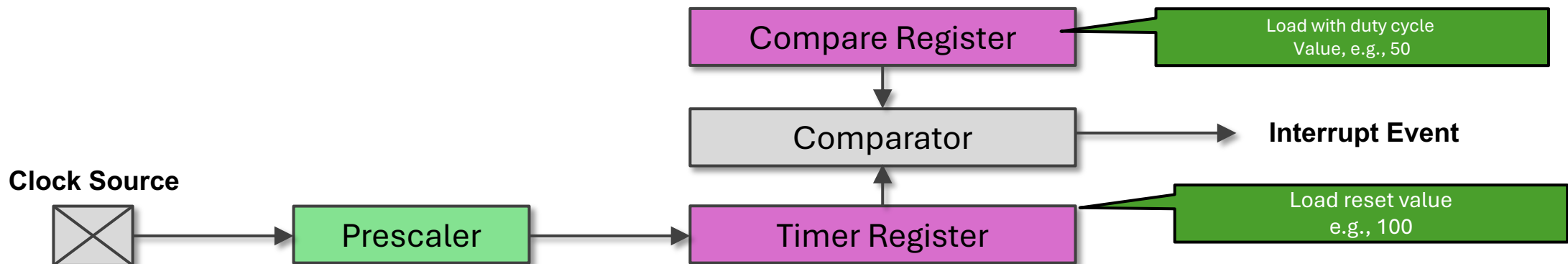
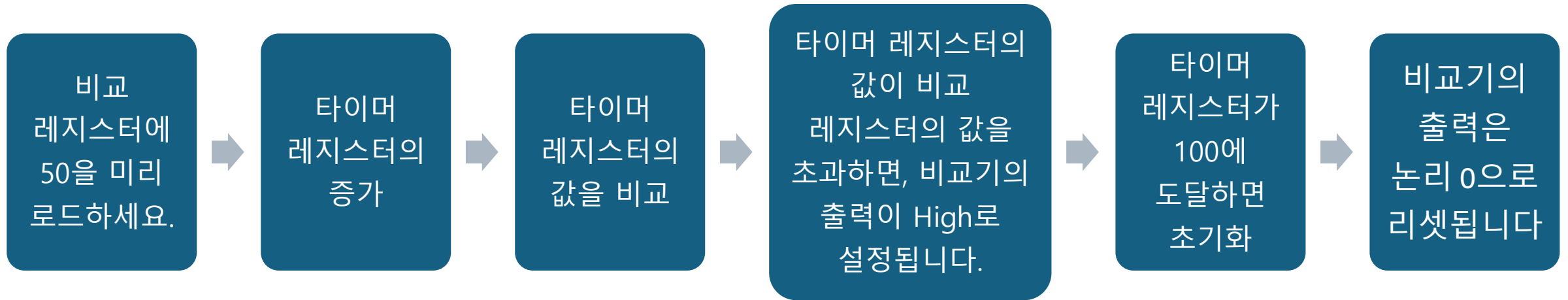
# Timer Operation Mode

- Pulse-width modulation (PWM) mode
  - PWM은 펄스의 폭을 사용하여 진폭을 변조함.
  - 주로 전력을 공급받는 전기 장치에 사용됨.
  - 펄스 주파수는 몇 kHz(예: 모터 구동)에서 수백 kHz (예: 오디오 증폭기, 컴퓨터 전원 공급 장치)까지 다양합니다.

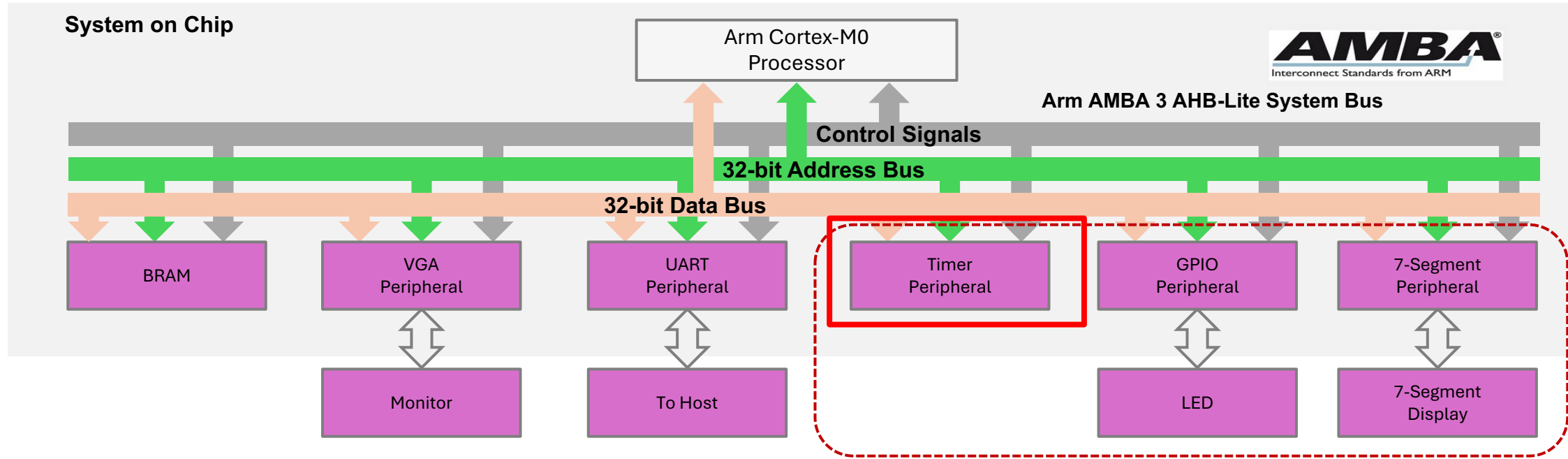


# Timer Operation Modes

- PWM 모드의 예: 50% 듀티 사이클을 가진 PWM을 생성하기

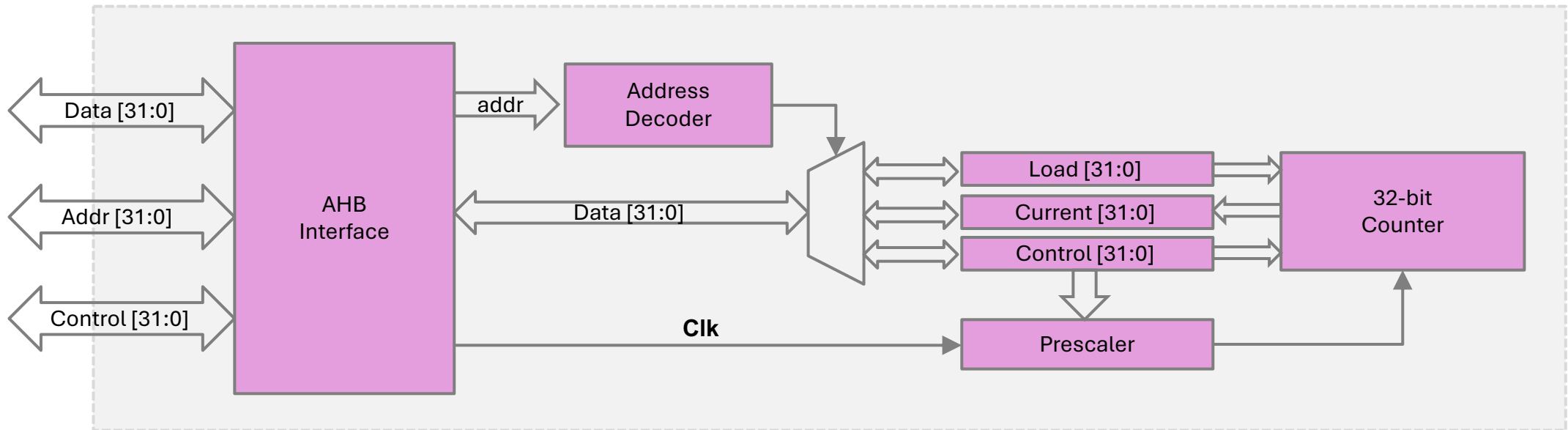


# Hardware Module Overview



# AHB Timer

- 작동 원리:
  - 활성화되면 자동으로 하향 카운트하는 32비트 카운터를 포함합니다.
  - 0에 도달하면 "로드 값" 레지스터에 저장된 값으로 초기화됩니다.
  - 동시에 인터럽트가 생성됩니다.






# Timer Registers

- 타이머 주변 장치의 최소한 네 가지 레지스터
  - 로드 값 레지스터 (Load Value Register)  
타이머가 0에 도달했을 때 리셋되는 값.
  - 현재 값 레지스터 (Current Value Register)  
32비트 카운터의 현재 값.
  - 제어 레지스터 (Control Register)  
카운터를 시작/중지하고 프리스케일러를 설정하는 데 사용됨.

Register	Address	Size
Base address	0x5200_0000	
Load value	0x5200_0000	4 Byte
Current value	0x5200_0004	4 Byte
Control	0x5200_0008	4 Byte

# 실습1 – timer\_pwm – timer\_pwm.v

- 코드를 완성 하시오
- 듀티 사이클 (%) =  
  $(\text{compare\_value} / \text{load\_value}) \times 100$

```
3  module Timer_PWM (  
4      input clk,                // Clock signal  
5      input reset,              // Reset signal  
6      input [31:0] load_value,  // Timer period  
7      input [31:0] compare_value, // Duty cycle value  
8      output reg pwm_out,       // PWM output signal  
9      output reg [31:0] timer_value // Current timer value  
10 );  
11  
12 always @(posedge clk or posedge reset) begin  
13     if (reset) begin  
14         timer_value <= 0;        // Reset timer value  
15         pwm_out <= 0;           // Reset PWM output  
16     end else begin  
17         if (timer_value < load_value) begin  
18             timer_value <= timer_value + 1; // Increment timer value  
19         end else begin  
20             timer_value <= 0;        // Reset timer when period is reached  
21         end  
22  
23         // Generate PWM signal based on compare value  
24         if (timer_value < compare_value) begin  
25             pwm_out <= 1;           // Logic High for duty cycle duration  
26         end else begin  
27             pwm_out <= 0;           // Logic Low for remaining period  
28         end  
29     end  
30 end  
31  
32 endmodule
```

# 실습1 – timer\_pwm 시뮬레이션, tb\_timer\_pwm.v

- 시뮬레이션 테스트 벤치 코드를 완성 하시오

```
// Instantiate the Timer_PWM module
Timer_PWM uut (
    .clk(clk),
    .reset(reset),
    .load_value(load_value),
    .compare_value(compare_value),
    .pwm_out(pwm_out),
    .timer_value(timer_value)
);

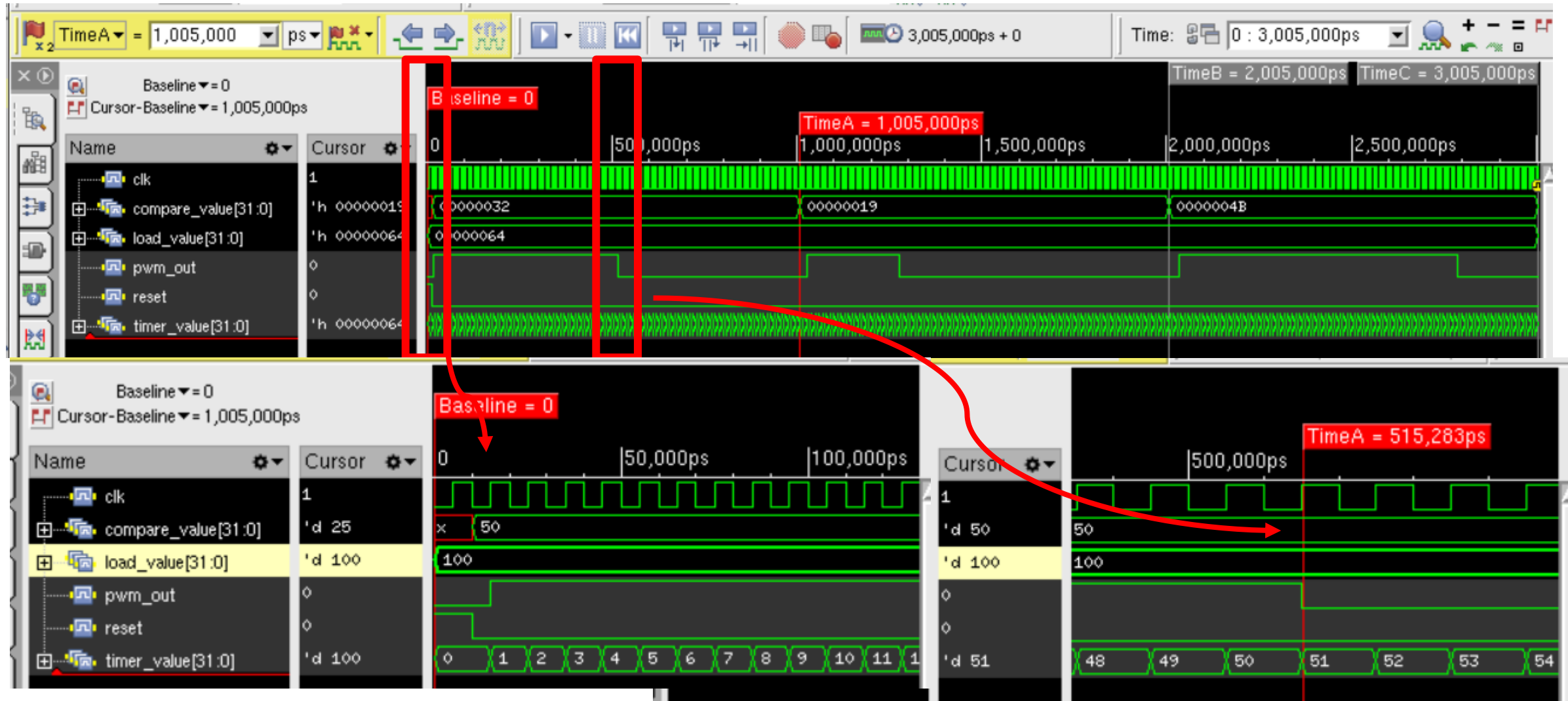
// Clock generation (10 ns clock period)
initial begin
    clk = 0;
    forever #5 clk = ~clk;    // Generate a clock with a period of 10 time units
end
```

# 실습1 – timer\_pwm 시뮬레이션, tb\_timer\_pwm.v

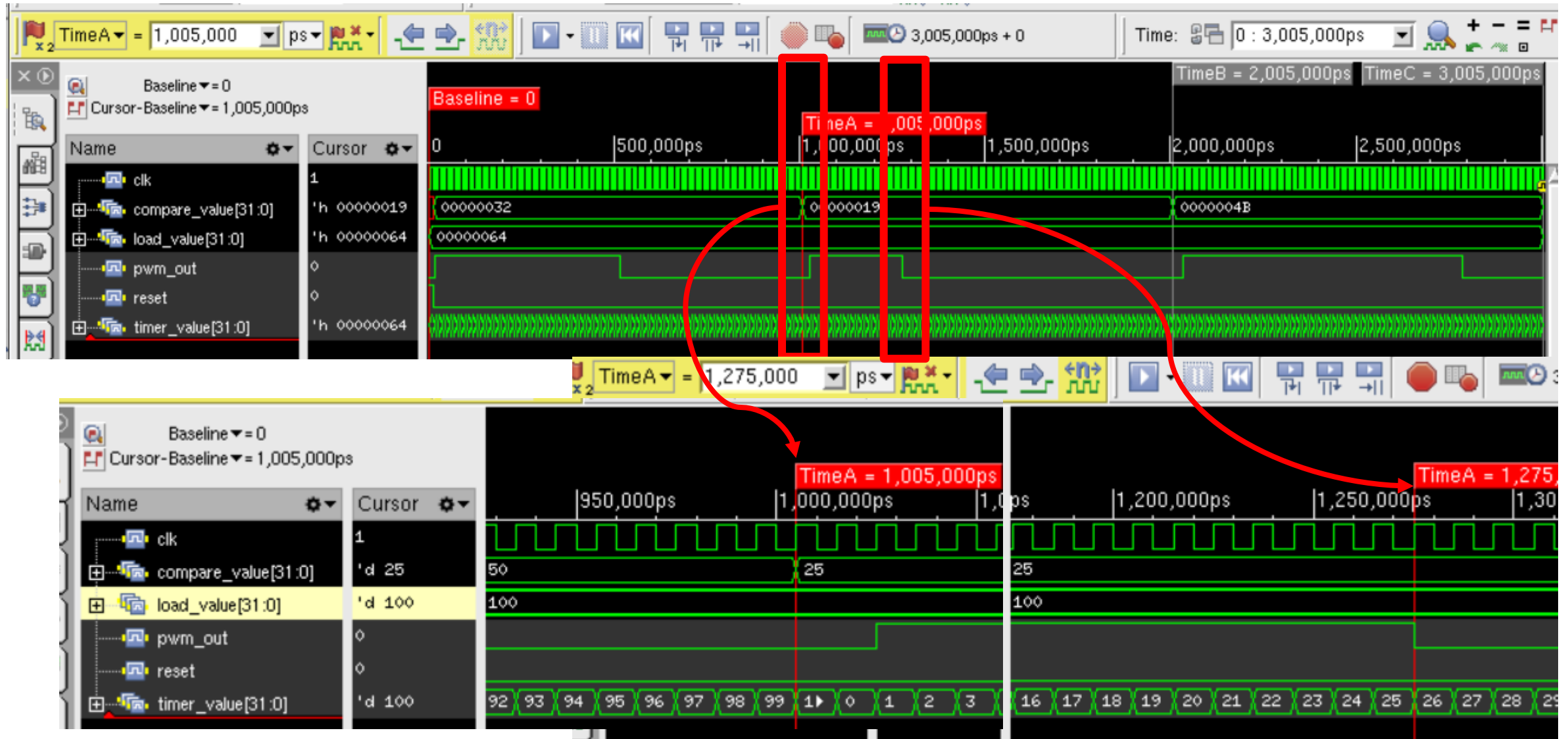
- 시뮬레이션 테스트 벤치 코드를 완성 하시오

```
28 // Test sequence for observing three PWM cycles
29 initial begin
30     $display("Starting simulation for observing three PWM cycles...");
31
32     reset = 1;
33     load_value = 32'd100; // Set timer period to 100 clock cycles
34
35     #10 reset = 0; // Release reset after some time
36
37     // Cycle 1: Duty cycle = 50%
38     compare_value = 32'd50; // Set duty cycle to 50%
39     $display("Cycle 1: Period = %d, Duty Cycle = %d%", load_value, (compare_value * 100) / load_value);
40
41     repeat (100) begin // Observe one full cycle (100 clock cycles)
42         @(posedge clk);
43         $display("Cycle %0t: Timer Value = %d, PWM Output = %b", $time, timer_value, pwm_out);
44     end
45
46     // Cycle 2: Duty cycle = 25%
47     compare_value = 32'd25; // Set duty cycle to 25%
48     $display("Cycle 2: Period = %d, Duty Cycle = %d%", load_value, (compare_value * 100) / load_value);
49
50     repeat (100) begin // Observe one full cycle (100 clock cycles)
51         @(posedge clk);
52         $display("Cycle %0t: Timer Value = %d, PWM Output = %b", $time, timer_value, pwm_out);
53     end
54
55     // Cycle 3: Duty cycle = 75%
56     compare_value = 32'd75; // Set duty cycle to 75%
57     $display("Cycle 3: Period = %d, Duty Cycle = %d%", load_value, (compare_value * 100) / load_value);
58
59     repeat (100) begin // Observe one full cycle (100 clock cycles)
60         @(posedge clk);
61         $display("Cycle %0t: Timer Value = %d, PWM Output = %b", $time, timer_value, pwm_out);
62     end
63
64     $stop; // End simulation
65 end
```

# 실습1 - timer\_pwm 시뮬레이션 결과1



# 실습1 - timer\_pwm 시뮬레이션 결과2





## 실습2 - AHB\_TIMER 설정

- SoC에서의 AHB\_TIMER와 prescaler를 설계 하십시오
- 다음장을 참조 하십시오
- Code를 완성 후 실행 방법은 다음과 같습니다.
- Prescaler.v
- AHB\_TIMER.v
- tb\_AHB\_TIMER.v

```
class5]$ xrun -gui -access +rw AHB_TIMER.v tb_AHB_TIMER.v prescaler.v  
4) 24.03-s005: Started on Mar 10, 2025 at 16:51:11 KST  
03-s005: (c) Copyright 1995-2024 Cadence Design Systems, Inc.
```



# 실습2 - AHB\_TIMER 설정 - AHBTIMER.v

- 코드를 완성 하시오

```
88 //Generate prescaled clk ticks
89 prescaler uprescaler16(
90     .inclk(HCLK),
91     .outclk(clk16)
92 );
93
94
95 assign HREADYOUT = 1'b1; //Always ready
96
97 always @(posedge HCLK)
98     if(HREADY)
99         begin
100             last_HWRITE <= HWRITE;
101             last_HSEL <= HSEL;
102             last_HADDR <= HADDR;
103             last_HTRANS <= HTRANS;
104         end
105
106 //Prescale clk based on control[2] 01 = 16 ; 00 = 1;
107 assign timerclk = ((control[2]) ? clk16 : 1'b1); //1'b1 signifi
108
109 assign enable = control[0];
110 assign mode = control[1];
```

```
113 //Control signal
114 always @(posedge HCLK, negedge HRESETn)
115     if(!HRESETn)
116         control <= 4'b0000;
117     else if(last_HWRITE & last_HSEL & last_HTRANS[1])
118         if(last_HADDR[3:0] == CTLADDR)
119             control <= HWDATA[3:0];
120
121
122 //Load signal
123 always @(posedge HCLK, negedge HRESETn)
124     if(!HRESETn)
125         load <= 32'h0000_0000;
126     else if(last_HWRITE & last_HSEL & last_HTRANS[1])
127         if(last_HADDR[3:0] == LDADDR)
128             load <= HWDATA;
129
130 //Clear signal
131 always @(posedge HCLK, negedge HRESETn)
132     if(!HRESETn)
133         clear <= 1'b0;
134     else if(last_HWRITE & last_HSEL & last_HTRANS[1])
135         if(last_HADDR[3:0] == CLRADDR)
136             clear <= HWDATA[0];
```

# 실습2 - AHB\_TIMER 설정 - prescaler.v

- 코드를 완성 하시오

```
37 module prescaler(  
38     input wire inclk,  
39     output wire outclk  
40 );  
41  
42 reg [3:0] counter;  
43  
44 always @(posedge inclk)  
45     counter <= counter + 1'b1;  
46  
47 assign outclk = counter == 4'b1111;  
48  
49 endmodule
```

# 실습2 - AHB\_TIMER 설정 - tb\_AHBTIMER.v

- 시뮬레이션 테스트 벤치 코드를 완성 하시오

```
34 // 클럭 생성 (10ns 주기)
35 initial begin
36     HCLK = 0;
37     forever #5 HCLK = ~HCLK; // 10ns 클럭 주기 (100MHz)
38 end
39
40 // 테스트 시나리오
41 initial begin
42     // 초기화 단계
43     HRESETn = 0; // 리셋 활성화
44     HADDR = 32'd0;
45     HWDATA = 32'd0;
46     HTRANS = 2'b00; // IDLE 상태
47     HWRITE = 0;
48     HSEL = 0;
49     HREADY = 1;
50
51     #20; // 리셋 유지 시간
52     HRESETn = 1; // 리셋 비활성화
53     // 타이머 로드값 설정
54     #10;
55     AHB_WRITE(32'h00000000, 32'd100); // LDADDR에 로드값 100 설정
56     // 타이머 제어 레지스터 설정 (Enable, Free-running mode)
57     #10;
58     AHB_WRITE(32'h00000008, 32'b0001); // CTLADDR에 제어 값 설정 (Enable)
59     // 타이머 동작 관찰
60     #500; // 타이머가 카운트다운하는 동안 대기
61     $display("타이머 인터럽트 상태: %b", timer_irq);
62     // 타이머 값 읽기
63     AHB_READ(32'h00000004); // VALADDR에서 현재 값 읽기
64
65     $finish; // 시뮬레이션 종료
66 end
```

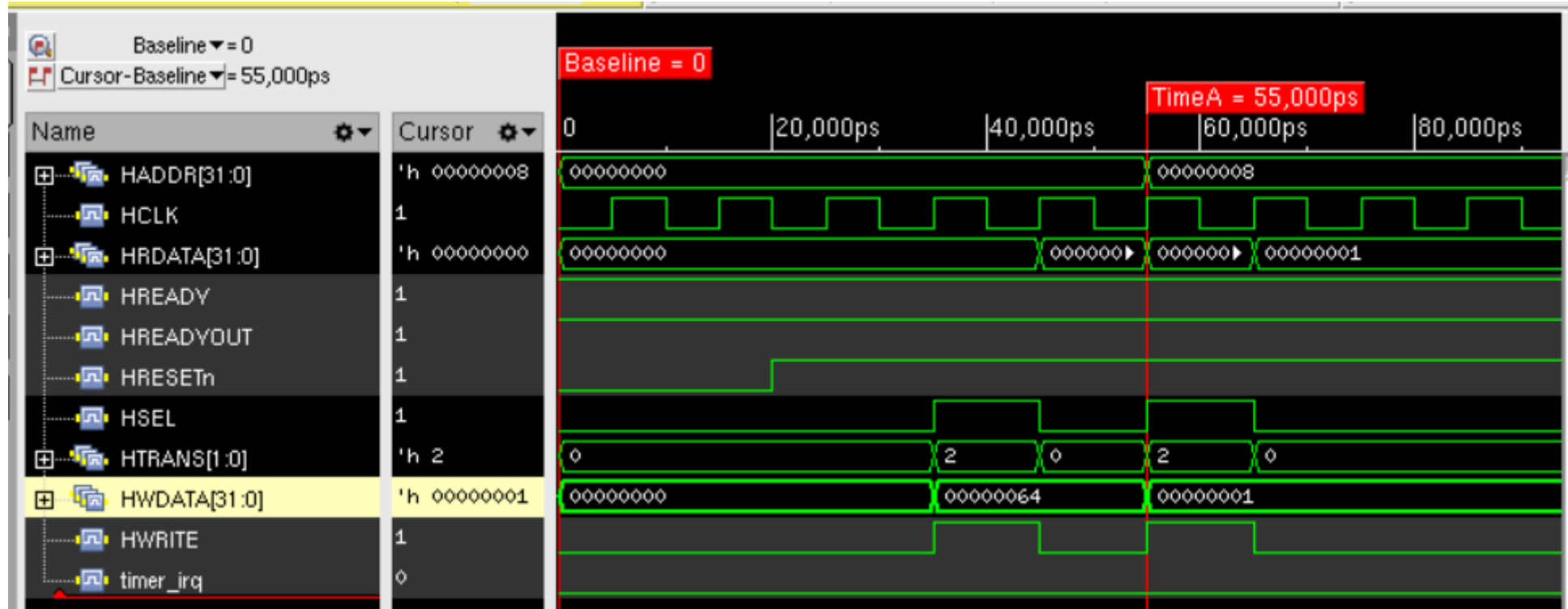
# 실습2 - AHB\_TIMER 설정 시뮬레이션 결과1



다음장  
실습결과2

다음다음장  
실습결과3

## 실습2 - AHB\_TIMER 설정 시뮬레이션 결과2



# 실습2 - AHB\_TIMER 설정 시뮬레이션 결과3

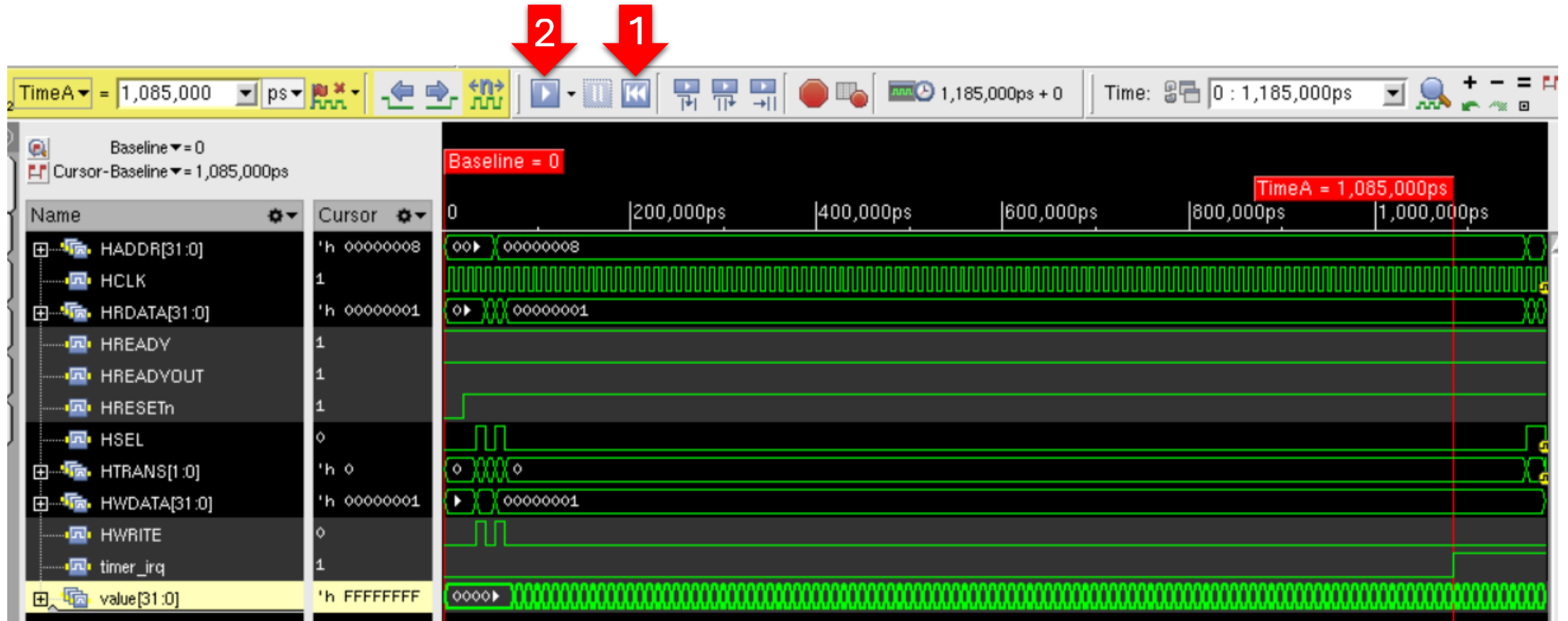
The image shows a simulation environment with three main panels:

- Design Browser:** Displays the project hierarchy. The 'uut' component is highlighted in yellow. A red arrow labeled '1' points from 'uut' to the 'value[31:0]' entry in the Objects list.
- Objects:** A list of variables and signals. The 'value[31:0]' entry is highlighted in yellow. A red arrow labeled '2' points from 'value[31:0]' to the 'value[31:0]' entry in the Waveform window.
- Waveform 1 - SimVision:** A window showing the simulation results. The 'value[31:0]' entry is highlighted in yellow. The waveform shows a signal that transitions from 0 to 1 at approximately 1,185,000 ps.

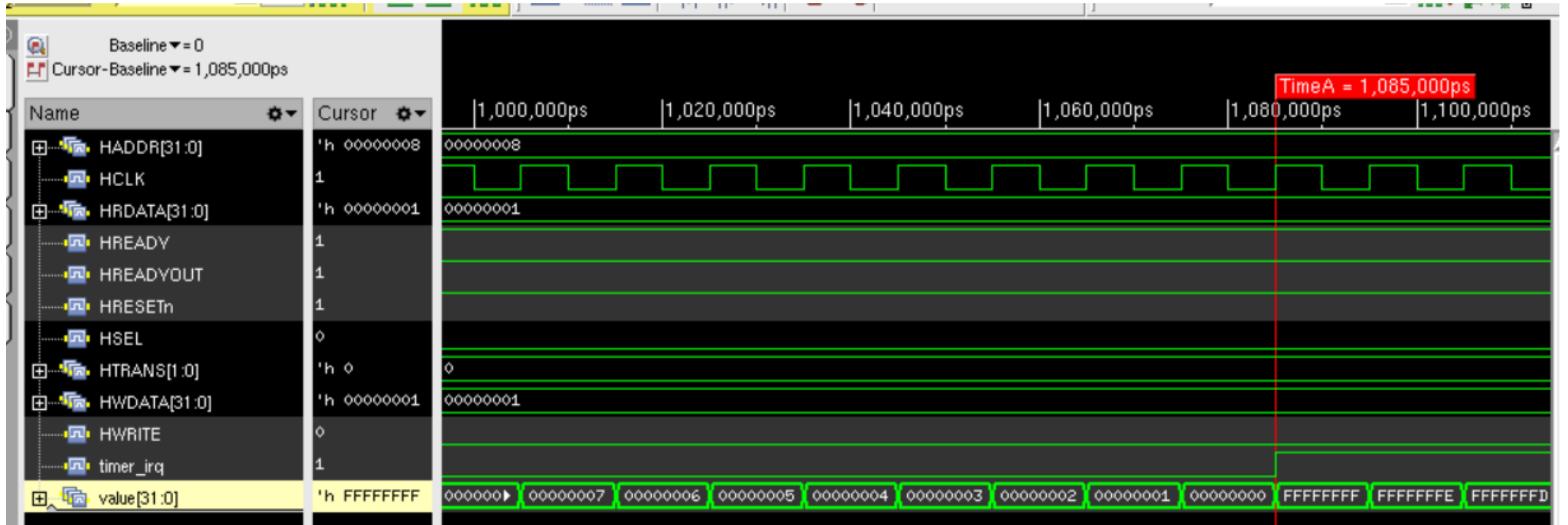
The Waveform window also displays a table of signal values:

Name	Cursor
HADDR[31:0]	'h 00000004
HCLK	1
HRDATA[31:0]	'h FFFFFFF6
HREADY	1
HREADYOUT	1
HRESETn	1
HSEL	0
HTRANS[1:0]	'h 0
HWDATA[31:0]	'h 00000001
HWRITE	0
timer_irq	1
value[31:0]	'h FFFFFFF6

## 실습2 - AHB\_TIMER 설정 시뮬레이션 결과4

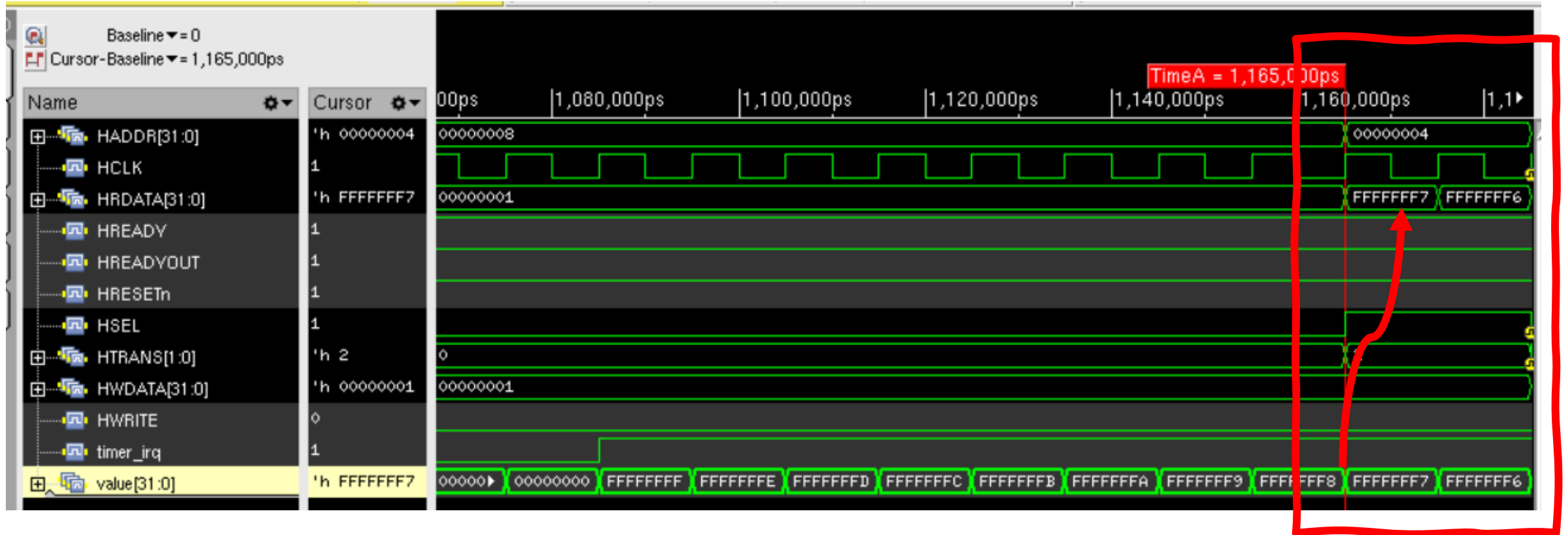


# 실습2 - AHB\_TIMER 설정 시뮬레이션 결과5





# 실습2 - AHB\_TIMER 설정 시뮬레이션 결과6



# SoC Interrupt

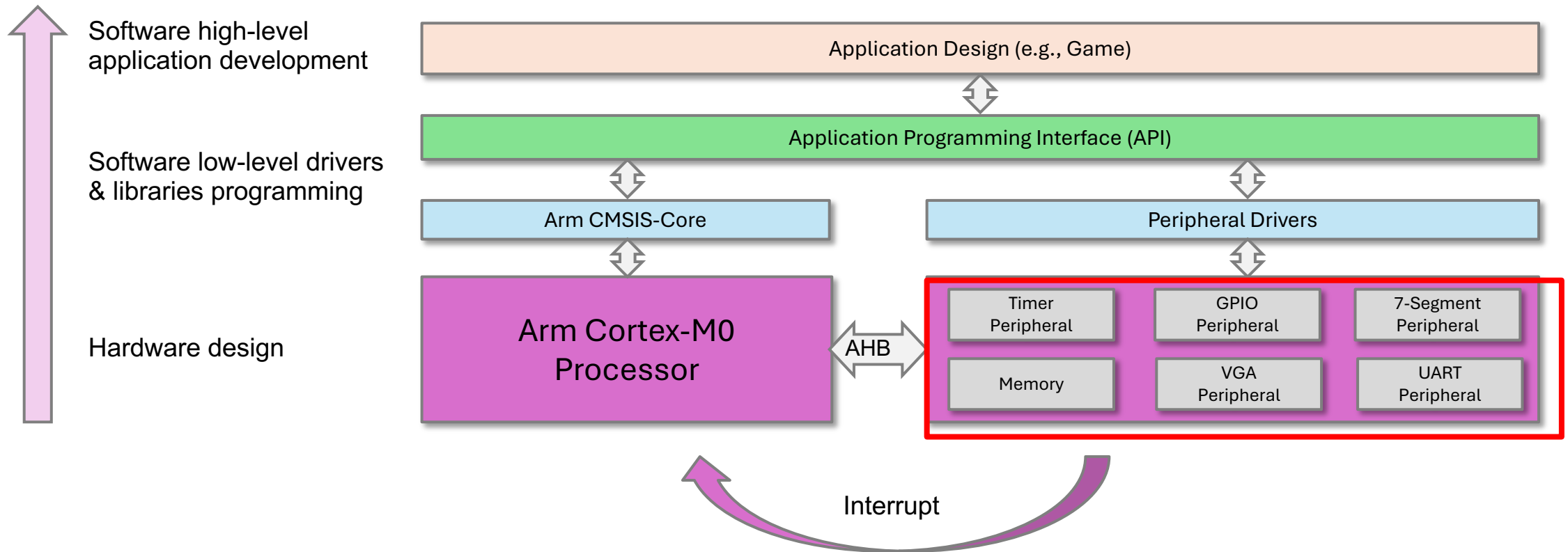
You are free to fork or clone this material. See [LICENSE.md](<https://github.com/arm-university/Introduction-to-SoC-Design-Education-Kit/blob/main/License/LICENSE.md>) for the complete license.

# 수업 목표

이 강의가 끝날 때, 여러분은 다음을 할 수 있어야 합니다:

- 다양한 인터럽트 유형과 그 기능을 설명할 수 있다.
- 프로세서가 인터럽트와 예외를 처리하는 단계들을 설명할 수 있다.
- 인터럽트와 예외의 차이점을 설명할 수 있다.
- Nested Vectored Interrupt Controller(NVIC)의 기능을 설명할 수 있다.
- 타이머, UART 등의 인터럽트 구현 방식을 설명할 수 있다.
- 인터럽트 메커니즘 프로세스를 개략적으로 설명하고 AHB 타이머 주변 장치의 인터럽트 구현 방식을 설명할 수 있다.

# Building a System on a Chip (SoC)



# Polling vs Interrupts

매초 숫자가 증가하도록 표시하는 방법:

## Polling

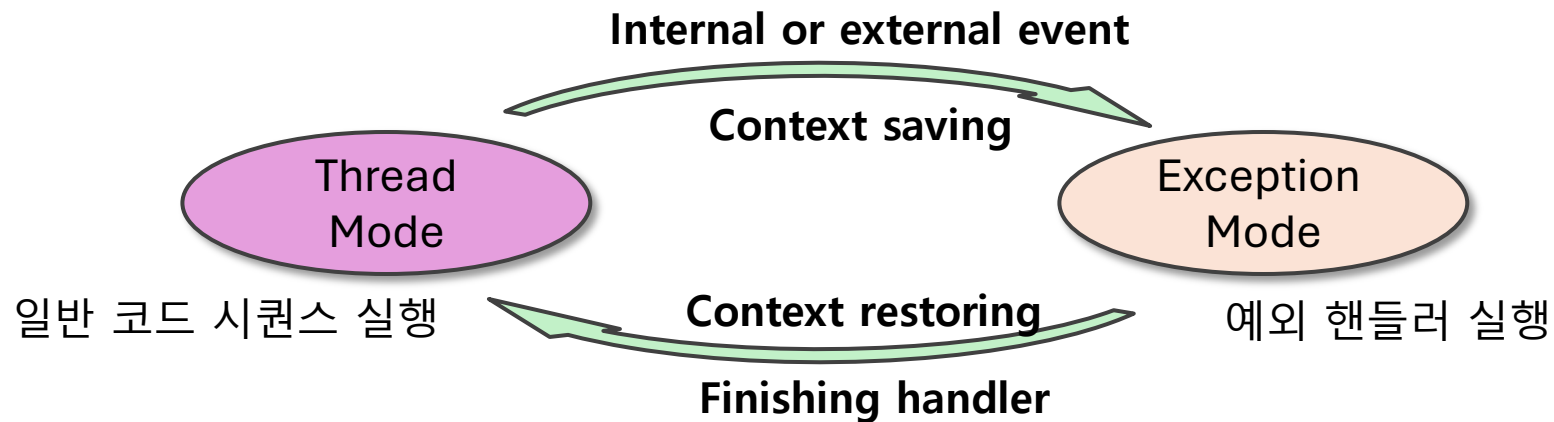
- 소프트웨어를 사용하여 하드웨어 타이머가 특정 값에 도달했는지 확인하기:
  - 느림: 타이머가 특정 값에 도달했는지 명시적으로 확인해야 함.
  - CPU 시간 낭비: 더 빠른 응답이 필요할수록 더 자주 확인해야 함.
  - 확장성이 낮음: 많은 활동을 빠르게 응답할 수 있는 시스템을 구축하기 어려움. 응답 시간은 다른 모든 처리에 따라 달라짐.

## Interrupt

- 타이머는 매초 인터럽트를 생성하며, 프로세서는 이에 응답하여 특정 코드(인터럽트 서비스 루틴: ISR)를 실행합니다.
- 빠름: 하드웨어 메커니즘 활용
- 효율적임: 필요한 경우에만 코드 실행
- 확장성이 높음
- ISR 응답 시간은 대부분의 다른 처리에 의존하지 않음
- 코드 모듈을 독립적으로 개발 가능

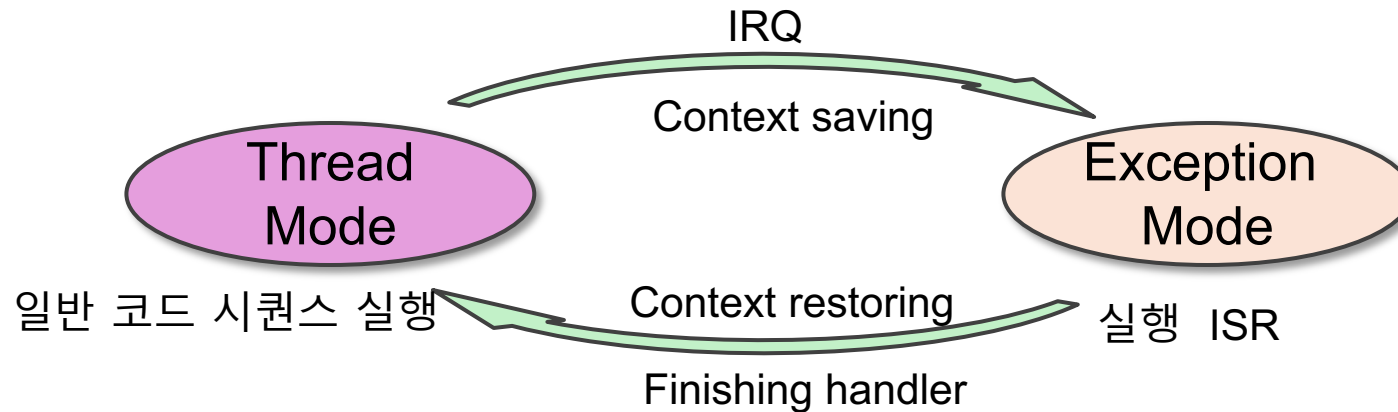
# Exception and Interruption

- 예외(Exception)
  - 예외는 프로그램 흐름이 현재 프로그램 스레드를 종료하고 해당 이벤트와 관련된 코드를 실행하게 만드는 이벤트(내부 또는 외부)입니다.
  - 이벤트는 내부 또는 외부일 수 있습니다.
  - 예외 핸들러(Exception Handler)는 예외 모드에서 실행되는 소프트웨어 코드입니다.



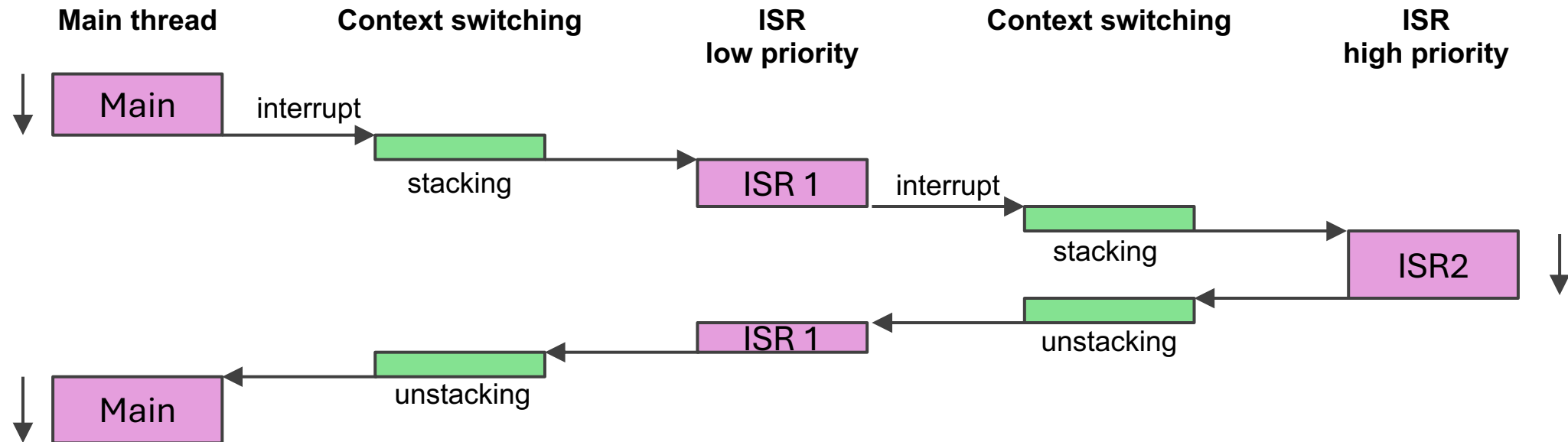
# Exception and Interruption

- 인터럽트(Interruption)
  - 외부 이벤트로 인해 발생하는 예외를 의미합니다.
  - 외부 이벤트는 인터럽트 요청(IRQ)이라고도 합니다.
  - 여기서 예외 핸들러는 인터럽트 서비스 루틴(ISR)이라고도 합니다.



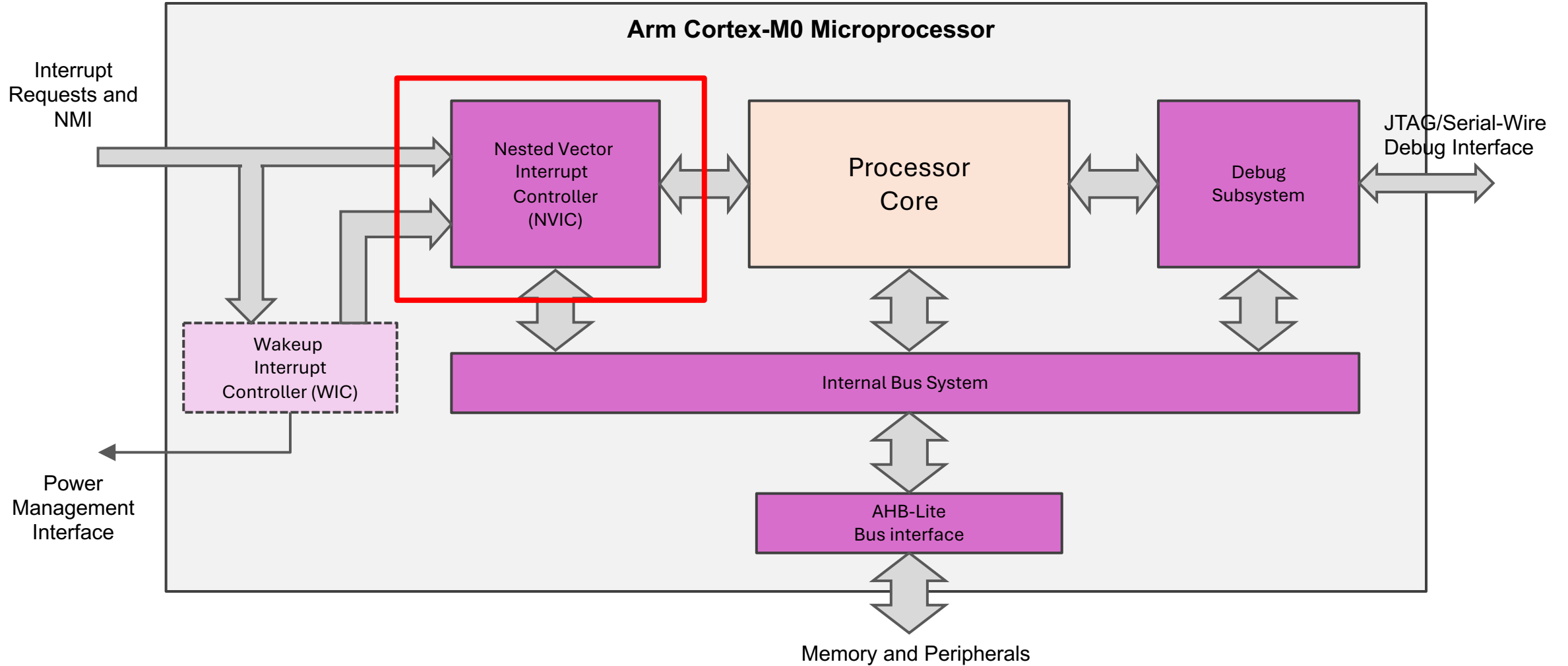
# Interrupt Preemption

- 예외(또는 인터럽트)는 일반적으로 여러 우선순위 수준으로 나뉩니다.
  - 더 높은 우선순위의 예외는 더 낮은 우선순위의 예외 동안에 발생하고 처리될 수 있습니다.
  - 이는 일반적으로 중첩 예외(nested exception) 또는 \*\*인터럽트 선점(interrupt preemption)\*\*이라고 알려져 있습니다.





# Cortex-M0 Block Diagram



# Armv6-M Exception Model

Exceptions with corresponding priority levels

Exception number	Exception type	Priority
1	Reset	-3 (highest)
2	NMI	-2
3	HardFault	-1
4-10	Reserved	
11	SVCall	Programmable
12-13	Reserved	
14	PendSV	Programmable
15	SysTick, optional	Programmable
16 + N	External interrupt 0-31	Programmable

# Armv6-M Exception Model

- 리셋(Reset)
  - Armv6-M 프로파일은 두 가지 수준의 리셋을 지원합니다.
    - 전원 온 리셋(Power-on reset): 프로세서, SCS(System Control Space), 디버그 로직을 리셋합니다.
    - 로컬 리셋(Local reset): 프로세서와 SCS를 리셋하지만 디버그 관련 리소스는 리셋하지 않습니다.
    - 리셋 예외는 고정된 우선순위 -3으로 영구적으로 활성화되어 있습니다.
- 비마스크 인터럽트(NMI: Non Maskable Interrupt)
  - 두 번째로 높은 우선순위를 가지는 예외로, 비활성화할 수 없습니다.
  - 산업 제어나 자동차와 같은 안전이 중요한 시스템에 유용합니다.
  - 전원 장애 처리 또는 워치독(Watchdog)으로 사용할 수 있습니다.

# Armv6-M Exception Model

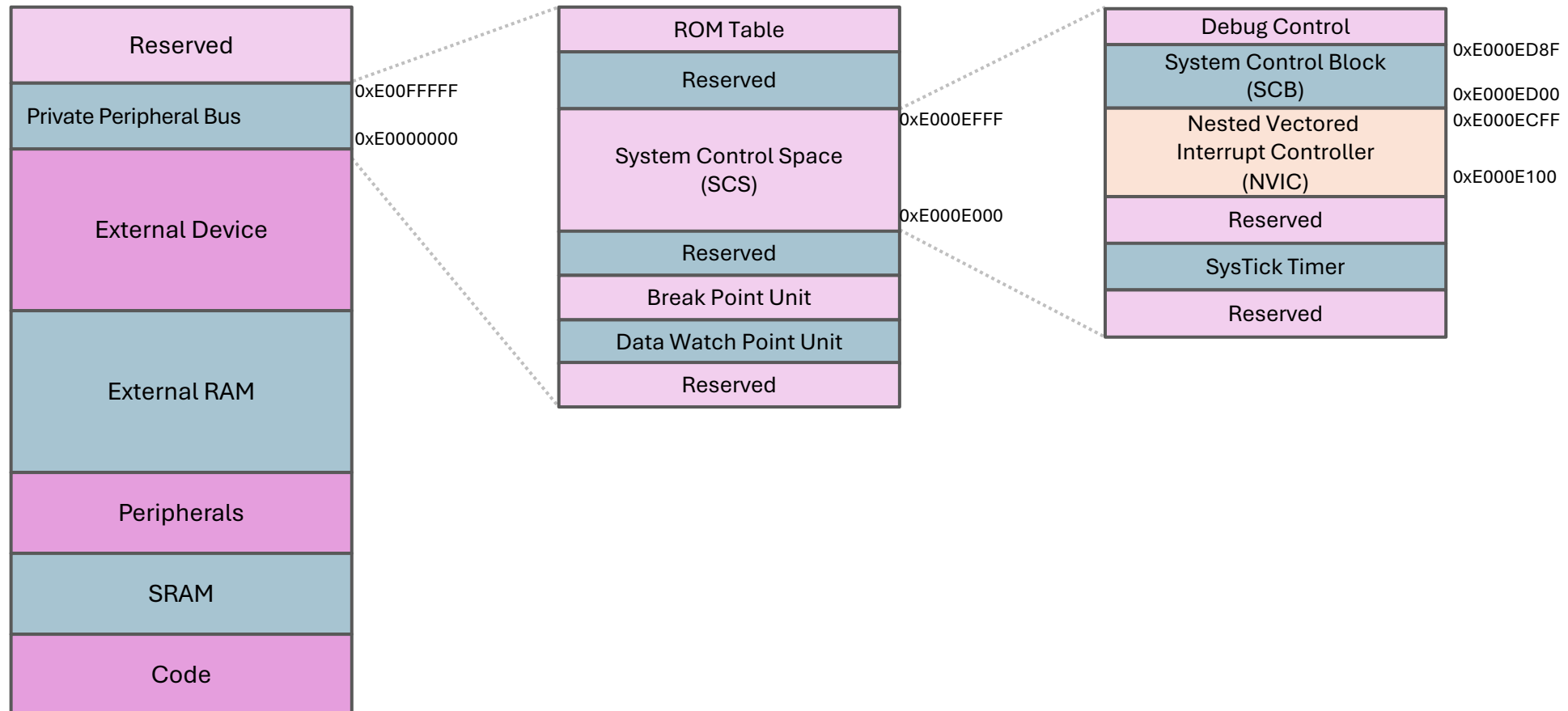
- HardFault
  - HardFault는 다른 예외 메커니즘으로 처리할 수 없는 모든 종류의 오류에 대해 존재하는 일반적인 오류입니다.
  - 예: 알 수 없는 명령어(opcode) 실행 시도, 버스 인터페이스 또는 메모리 시스템에서의 오류
- SVCall (SuperVisor Call)
  - SVCall 예외는 SVC 명령어가 실행될 때 발생합니다.
  - 주로 운영 체제(OS) 제어에서 사용됩니다.
- PendSV (Pendable Service Call)
  - SVCall과 유사하지만 즉시 실행되지 않습니다.
  - 우선순위가 높은 작업이 완료된 후에만 시작됩니다.

# Armv6-M Exception Model

- SysTick
  - SysTick은 운영 체제(OS) 애플리케이션의 또 다른 기능입니다.
  - 예외 시작: 타이머에서 정기적인 인터럽트가 생성될 때 시작됩니다.
  - 외부 인터럽트: 최대 32개의 외부 인터럽트를 지원하며, 온칩 주변 장치에서 연결될 수 있습니다. 사용 전에 활성화가 필요합니다.
  - 프로그래밍 가능한 우선순위: 우선순위 값을 설정할 수 있습니다.

# Cortex-M0 Interrupt Controller

- Memory map of nested vectored interrupt controller (NVIC)



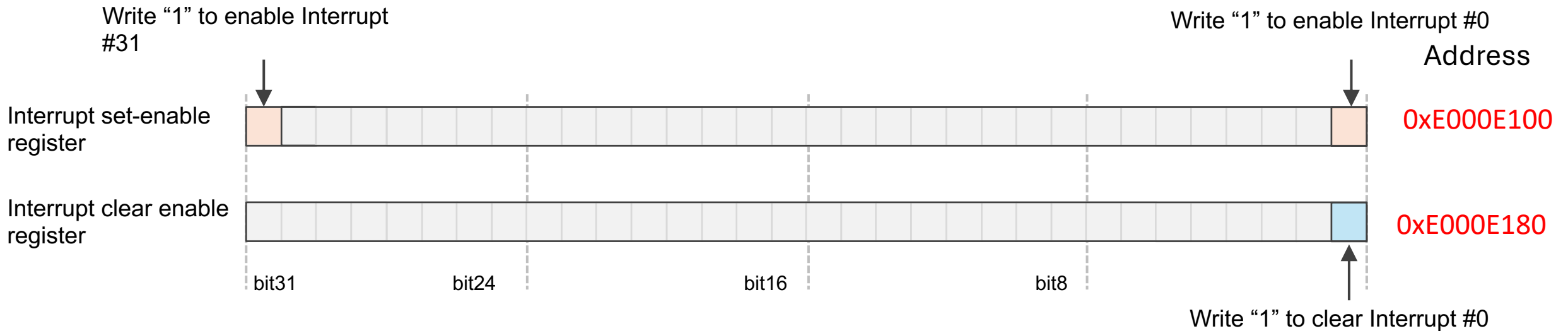
# NVIC Registers

## NVIC register addresses

Address	Register
<b>0xE000E100</b>	<b>Interrupt set-enable register</b>
0xE000E104 — 0xE000E17F	Reserved
<b>0xE000E180</b>	<b>Interrupt clear enable register</b>
0xE000E184 — 0xE000E1FF	Reserved
<b>0xE000E200</b>	<b>Interrupt set-pending register</b>
0xE000E204 — 0xE000E27F	Reserved
<b>0xE000E280</b>	<b>Interrupt clear-pending register</b>
0xE000E300 — 0xE000E3FC	Reserved
<b>0xE000E400 — 0xE000E41C</b>	<b>Interrupt priority registers</b>
0xE000E420 — 0xE000E43C	Reserved

# NVIC Registers

- Interrupt set-enable register
  - Write “1” to enable one or more interrupts
  - Write “0” has no effect
- Interrupt clear enable register
  - Write “1” to one or more interrupts
  - Write “0” has no effect





# NVIC Registers

- 왜 별도의 레지스터 주소를 사용하는가?
  - 인터럽트를 활성화/비활성화하는 데 필요한 단계를 줄여 코드 크기와 실행 시간을 감소시킵니다.
  - **\*\*경쟁 상태(race condition)\*\***를 방지합니다. 예를 들어, 메인 스레드가 레지스터를 "읽기-수정-쓰기(read-modify-write)" 과정으로 접근하는 동안 인터럽트가 발생하여 "읽기"와 "쓰기" 작업 사이에 중단될 수 있습니다. 만약 ISR이 메인 스레드가 현재 접근 중인 동일한 레지스터를 다시 수정하면 충돌이 발생할 수 있습니다.

# NVIC Registers

- 인터럽트 대기 상태와 대기 해제
  - 인터럽트는 즉시 처리될 수 없는 경우 대기 상태(pending)로 들어갑니다. 예를 들어, 더 높은 우선순위의 인터럽트가 처리 중인 경우, 낮은 우선순위의 인터럽트는 대기 상태로 유지됩니다.

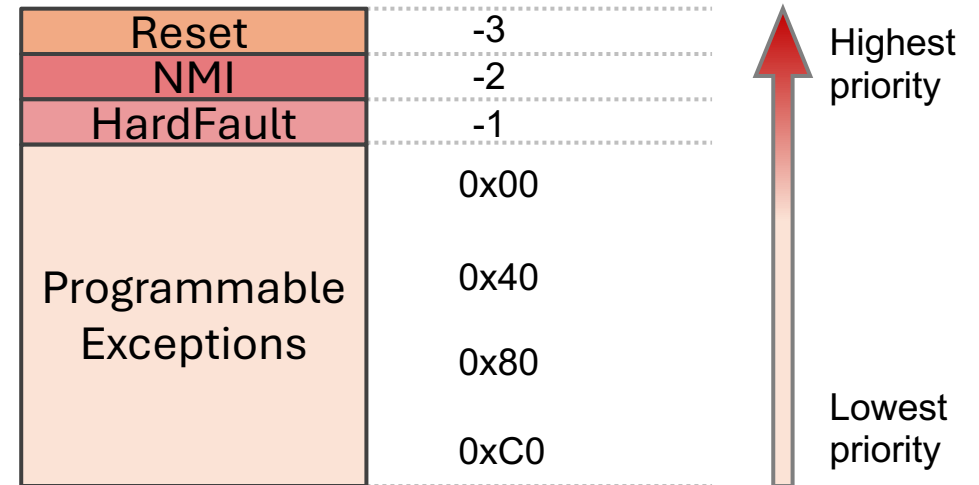
# NVIC Registers

- 인터럽트 우선순위 레지스터
  - 우선순위 수준을 설정하는 레지스터는 8비트 폭을 가지고 있지만, Cortex-M0에서는 오직 2비트만 구현되어 있습니다.
  - 2개의 최상위 비트(MSB)만 사용되므로, 4단계의 우선순위를 나타낼 수 있습니다: 0x00, 0x40, 0x80, 그리고 0xC0.



↓

Possible priorities:  
0x00, 0x40, 0x80, 0xC0

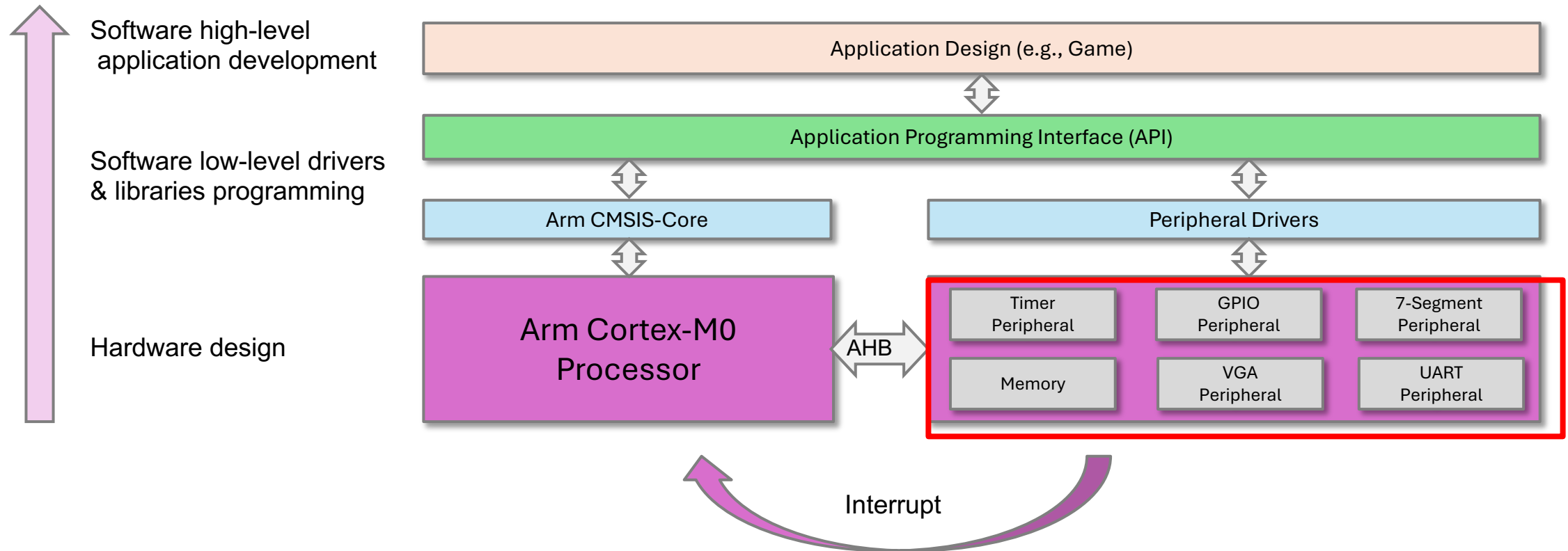


# NVIC Registers

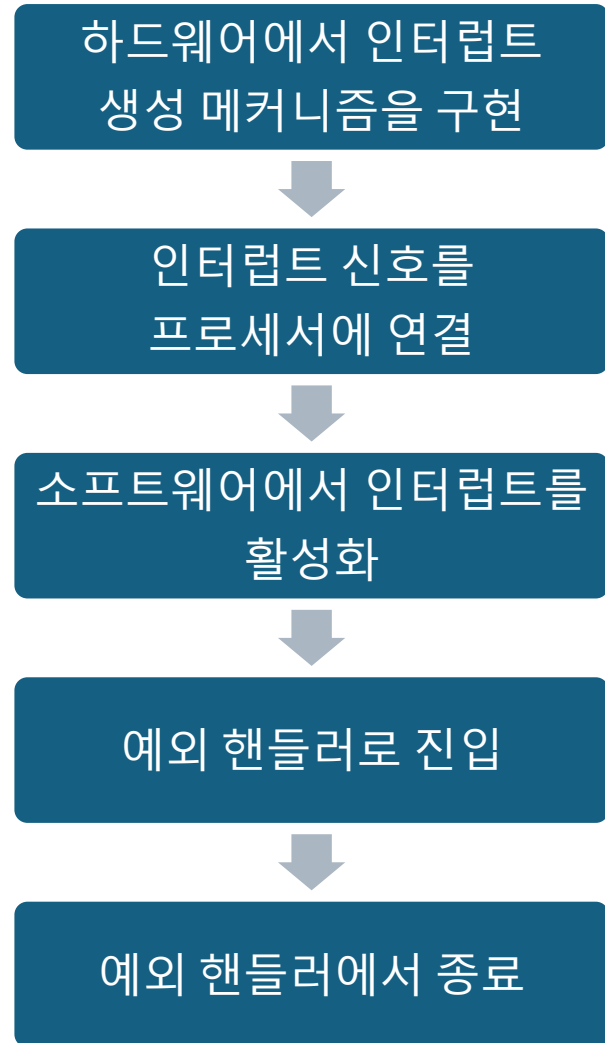
- 인터럽트 우선순위 레지스터
  - 32개의 인터럽트에 대한 우선순위를 설정하기 위해 8개의 32비트 레지스터를 사용합니다.
  - 각 레지스터는 4개의 인터럽트에 대한 우선순위를 포함하며, 각 인터럽트 우선순위는 2비트를 사용하여 구현됩니다.

0xE000E41C	31							30									29										28						
0xE000E418	27							26									25										24						
0xE000E414	23							22									21										20						
0xE000E410	19							18									17										16						
0xE000E40C	15							14									13										12						
0xE000E408	11							10									9										8						
0xE000E404	7							6									5										4						
0xE000E400	3							2									1										0						
	bit32							bit24									bit16									bit8							bit1

# Building a System on a Chip (SoC)

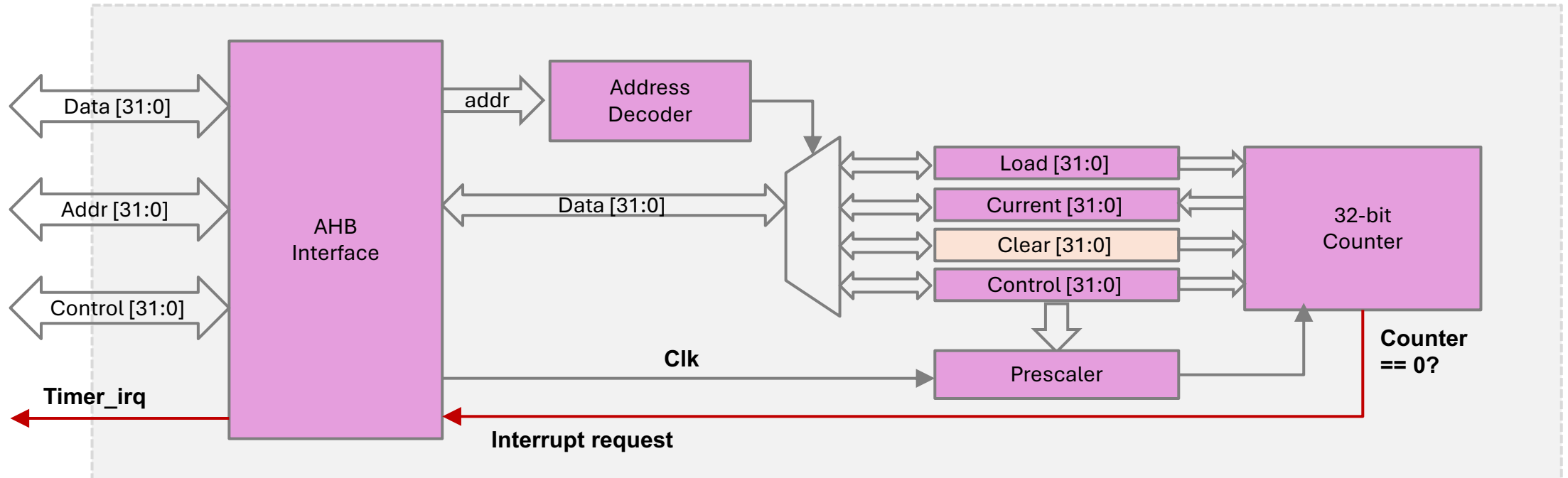


# The Interrupt Mechanism Process



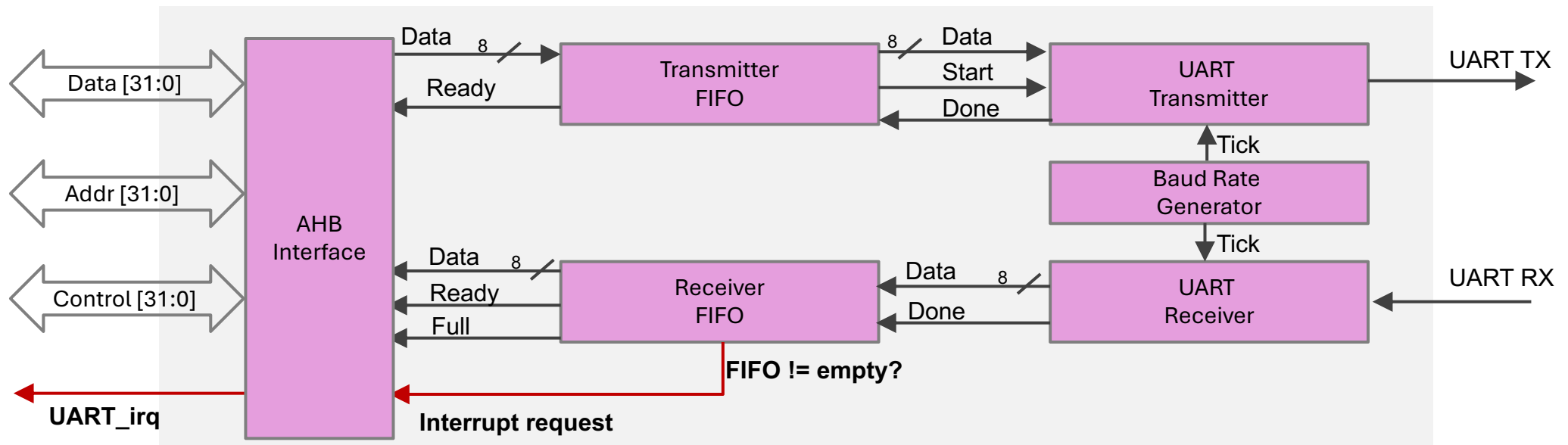
# Interrupt Implementation for Timer

- AHB 타이머 주변 장치에 대한 인터럽트 메커니즘을 구현하려면 다음과 같은 작업을 수행해야 합니다:
  - 카운터가 0에 도달할 때마다 인터럽트가 생성됩니다.
  - 프로세서가 ISR(Interrupt Service Routine)을 완료한 후 인터럽트 요청을 해제하기 위해 클리어 레지스터를 추가해야 합니다



# Interrupt Implementation for UART

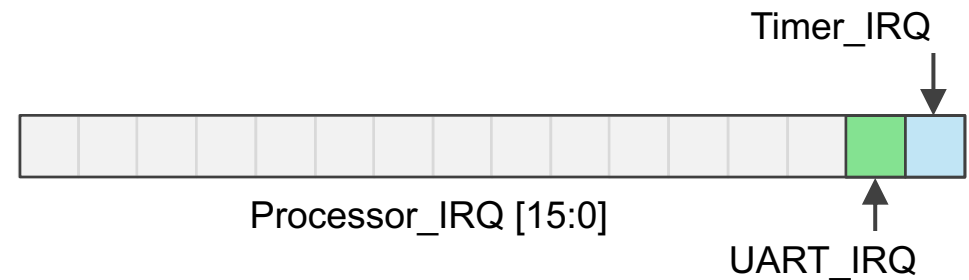
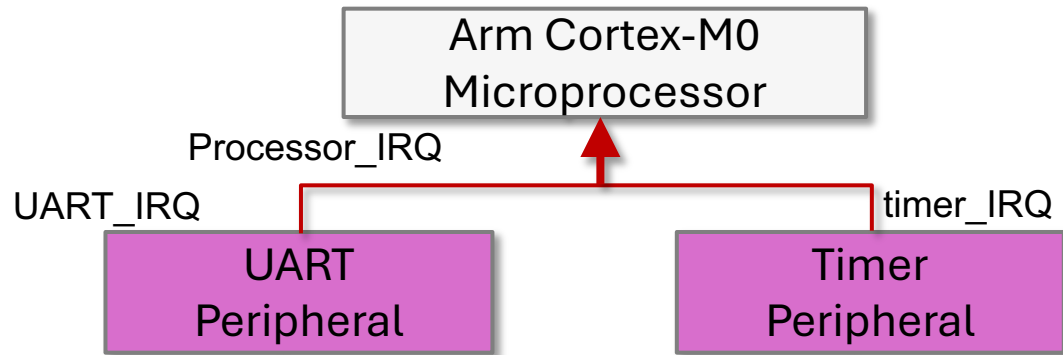
- AHB UART 주변 장치에 대한 인터럽트 메커니즘을 구현하십시오.
  - 예를 들어, 수신 FIFO가 비어 있지 않을 때 인터럽트를 생성하도록 설정할 수 있습니다.





# Connect Interrupts to Processor

- 주변 장치의 인터럽트를 Cortex-M0 마이크로프로세서에 연결.
  - 이 교육 자료에서는 간소화된 버전의 Cortex-M0 (Cortex-M0 DesignStart)를 사용하며, 이는 16개의 외부 인터럽트만 지원



# Enable Interrupts in Software

- Configure NVIC:
  - Set interrupt priority registers, for example:

LDR	R0, =0xE000E400	: Address of priority0 register
LDR	R1, [R0]	: Get priority0 register
MOVS	R2, #0xFF	: Byte mask
BICS	R1, R1, R2	: R1= R1 AND (Not (0x000000FF))
MOVS	R2, #0x40	: Priority level
ORRS	R1, R1, R2	: Update the value of priority register
STR	R1, [R0]	: Write back the priority register

- Set interrupt enable register, for example:

LDR	R0, =0xE000E100	: NVIC Enable register
MOVS	R1, #0x1	: Interrupt #0
STR	R1, [R0]	: Enable interrupt #0

- Make sure PRIMASK register is zero:
  - Set to one will block all the interrupts, other than non-maskable interrupt (NMI) and the hard fault exception, for example:

MOVS	R0, #0x0	:
MSR	PRIMASK, R0	: Clear PRIMASK register

# Entering an Exception Handler(예외 핸들러로 진입)

1. 현재 명령어를 완료합니다 (긴 명령어는 제외).
2. 인터럽트 벡터를 조회하고 예외 처리기의 진입 주소로 분기 합니다.
3. 현재 스택(MSP 또는 PSP)에 컨텍스트를 푸시 합니다.
4. PC(프로그램 카운터)에 예외 처리기의 주소를 로드 합니다.
5. LR(링크 레지스터)에 EXC\_RETURN 코드를 로드 합니다.
6. IPSR(Interrupt Program Status Register)에 예외 번호를 로드 합니다.
7. 예외 처리기의 첫 번째 명령어 실행을 시작합니다.
  - 일반적으로 예외 요청부터 처리기의 첫 번째 명령어 실행까지 16 사이클이 소요 됩니다.
  - 인터럽트 지연(latency)은 인터럽트에 진입하기 전까지의 시간 지연으로, 이는 최소화해야 할 오버헤드 입니다.

# Exiting an Exception Handler

- 주변 장치의 인터럽트 요청을 다음과 같이 클리어합니다:
  - 타이머는 인터럽트 요청을 클리어하는 데 사용되는 클리어 레지스터를 가질 수 있습니다.
  - 또는, 특정 조건에서 주변 장치 자체가 인터럽트 요청을 자동으로 클리어할 수 있습니다. 예를 들어, UART는 수신 FIFO에서 모든 데이터를 읽어 들인 후 인터럽트 요청을 클리어할 수 있습니다.
- 컨텍스트 복원: 스택에서 레지스터를 팝업 합니다.
- IPSR(Interrupt Program Status Register)을 업데이트합니다.
- PC(프로그램 카운터)에 반환 주소를 로드 합니다.
- 이전 프로그램의 코드를 계속 실행합니다.

# 실습3 – Timer&Interrupt 생성 및 clear

- SoC에서의 AHBTIMER와 prescaler를 이용하여 timer interrupt가 발생하면 해당 인터를 처리하고 clear하는 동작을 시뮬레이션 해보시오
- 테스트 벤치는 다음장을 참조 하십시오
- 기존에 설계된 AHBTIMER를 이용하십시오.
- Prescaler.v
- AHBTIMER.v
- tb\_AHBTIMER2.v

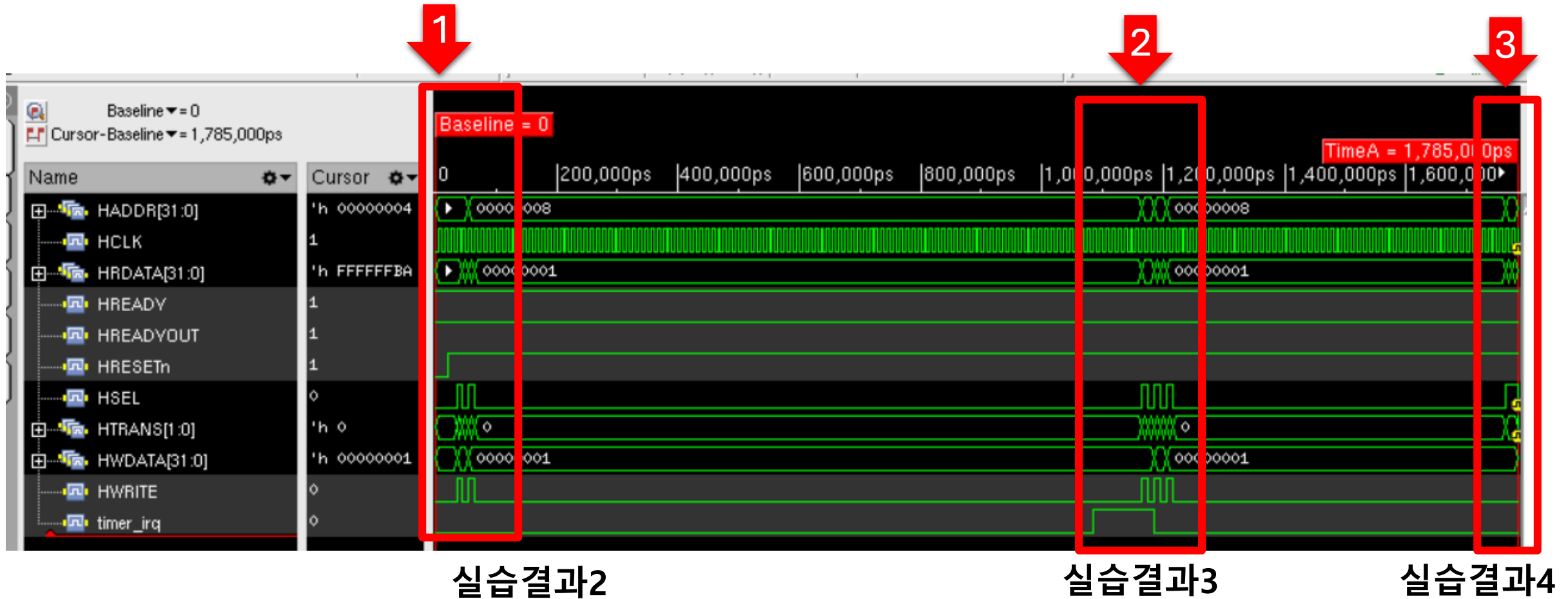
```
class5]$ xrun -gui -access +rw AHBTIMER.v tb_AHBTIMER2.v prescaler.v  
4)      24.03-s005: Started on Mar 16, 2025 at 22:10:24 KST  
3-s005: (c) Copyright 1995-2024 Cadence Design Systems, Inc.
```

# 실습3 – Timer&Interrupt 생성 및 clear 시뮬레이션

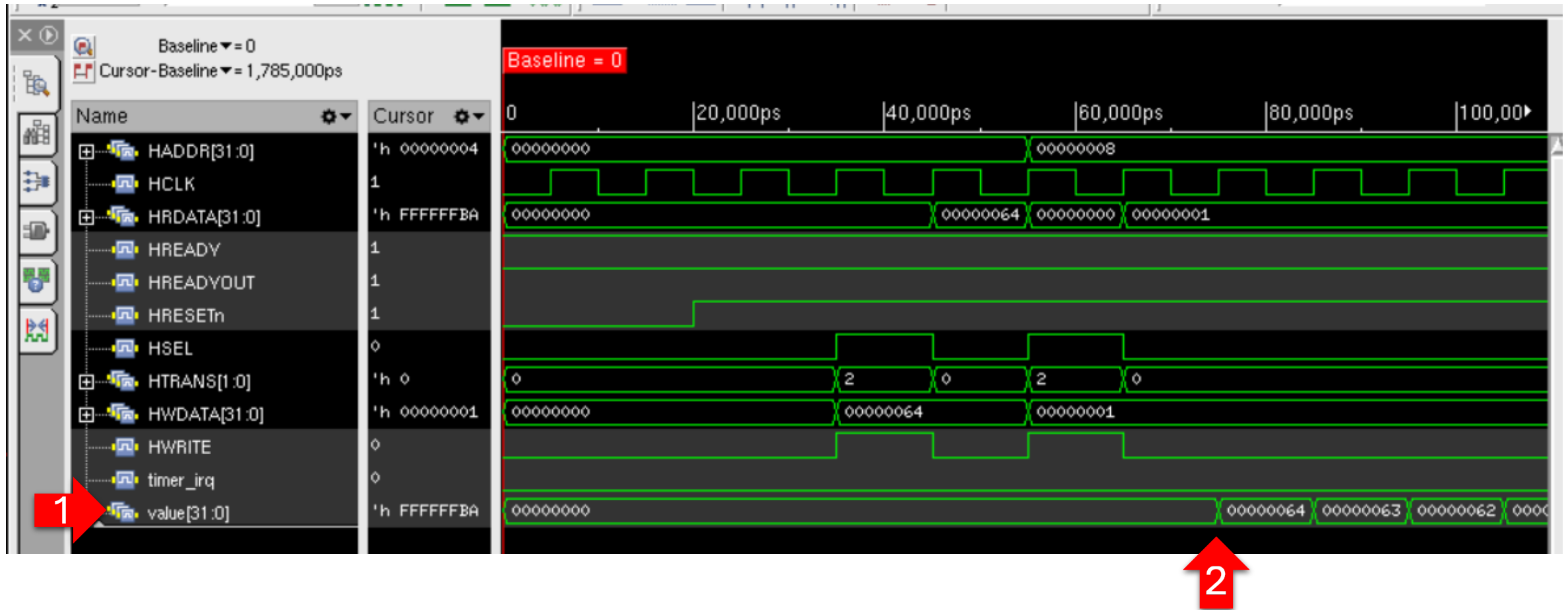
- 시뮬레이션 테스트 벤치 코드를 완성 하시오

```
62 // 타이머 동작 관찰
63 #1100; // 타이머가 카운트다운하는 동안 대기
64
65 $display("타이머 인터럽트 상태: %b", timer_irq);
66
67 if (timer_irq) begin
68     $display("타이머 인터럽트가 활성화되었습니다. 인터럽트를 클리어합니다.");
69     AHB_WRITE(32'h0000000C, 32'b1); // IRQCLRADDR에 클리어 명령 전달
70     #10; // 클리어 후 대기
71     $display("타이머 인터럽트 상태 (클리어 후): %b", timer_irq);
72
73     // 다시 타이머 로드값 설정 및 재시작
74     $display("타이머를 재설정하고 다시 시작합니다.");
75     AHB_WRITE(32'h00000000, 32'd50); // LDADDR에 새로운 로드값 50 설정
76     #10;
77     AHB_WRITE(32'h00000008, 32'b0001); // CTLADDR에 제어 값 설정 (Enable)
78
79     #550; // 새로운 타이머 카운트다운 대기
80
81     $display("다시 발생한 타이머 인터럽트 상태: %b", timer_irq);
82     if (timer_irq) begin
83         $display("다시 타이머 인터럽트가 발생했습니다.");
84     end
85 end
86
87 // 타이머 값 읽기
88 AHB_READ(32'h00000004); // VALADDR에서 현재 값 읽기
89
90 $finish; // 시뮬레이션 종료
91 end
```

# 실습3 – Timer&Interrupt 생성 및 clear 결과1

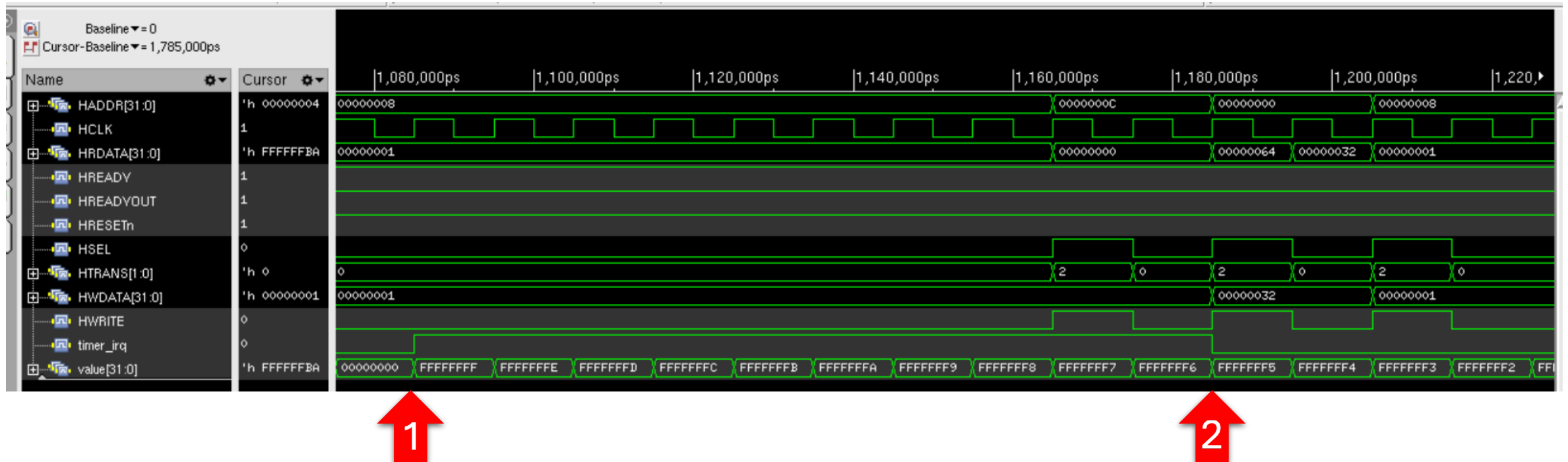


# 실습3 – Timer&Interrupt 생성 및 clear 결과2

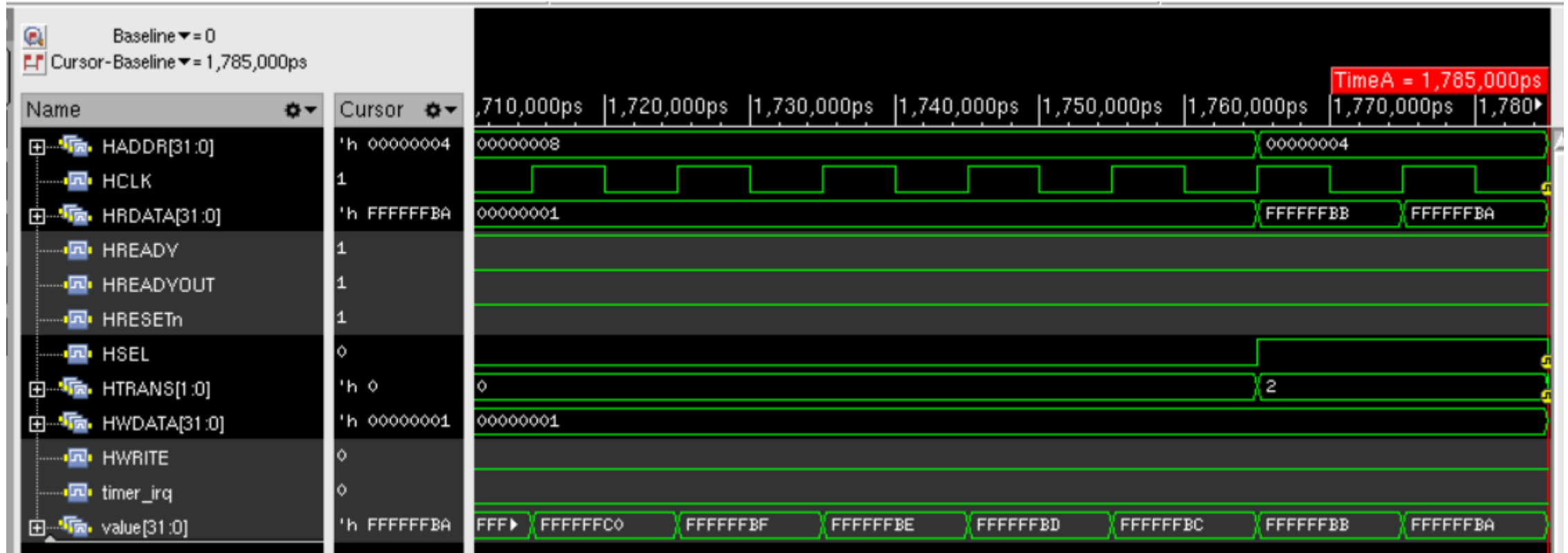




# 실습3 – Timer&Interrupt 생성 및 clear 결과3



# 실습3 – Timer&Interrupt 생성 및 clear 결과4



**Thanks**