Real-Time Operating System (Day 4 Lab)

Jong-Chan Kim

Graduate School of Automotive Engineering

국민대학교
KOOKMIN UNIVERSITY

Cooperative Scheduling

- Non-preemptive에서 Scheduling Point 설정

- Preemptive에서 Scheduling Disable

Scheduling Points

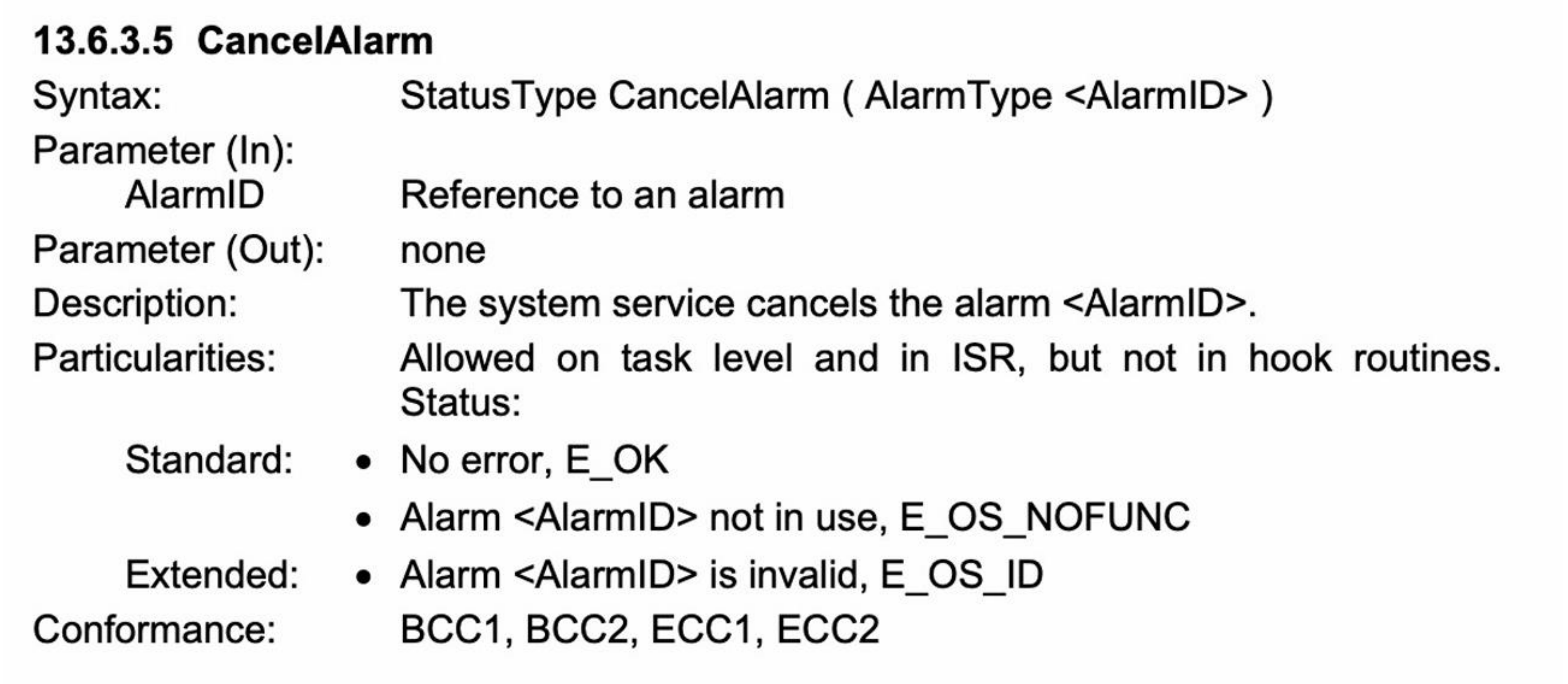
```
TASK(Task1)
{
      …
      Schedule();
      …
      Schedule();
      …
}
```

No scheduling allowed

```
TASK(Task1)
{
      …
      GetResource(RES_SCHEDULER);
      …
      ReleaseResource(RES_SCHEDULER);
      …
}
```

Alarm 기반 Activation의 문제점

- OSEK에서 Periodic Task 작성 방법
  - Counter와 연결된 Alarm에서 Activate하도록 OIL 설정
- 문제점
  - Alarm은 런타임에 취소/변경이 가능
  - 실수 혹은 악의적으로 Alarm 변경 시 Periodic Task 오동작 가능성
- Example)
  - CancleAlarm() 호출

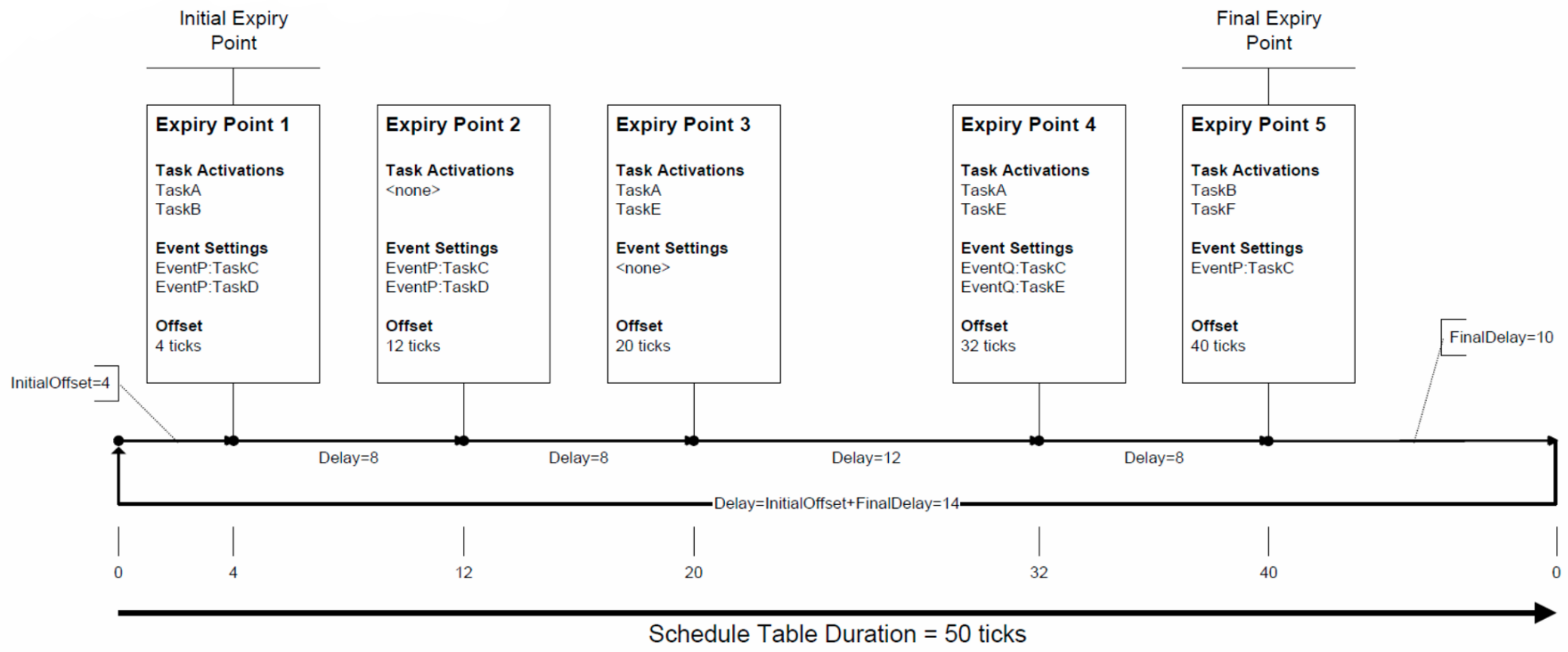


13.6.3.5 CancelAlarm

Syntax:		StatusType CancelAlarm (AlarmType <AlarmID>)
Parameter (In):		
AlarmID		Reference to an alarm
Parameter (Out):		none
Description:		The system service cancels the alarm <AlarmID>.
Particularities:		Allowed on task level and in ISR, but not in hook routines. Status:
Standard:	•	No error, E_OK
	•	Alarm <AlarmID> not in use, E_OS_NOFUNC
Extended:	•	Alarm <AlarmID> is invalid, E_OS_ID
Conformance:		BCC1, BCC2, ECC1, ECC2

Schedule Table Concepts

- Duration: Schedule table의 사이클 타임
- Expiry points: Duration 안에서의 상대 시간 (Activate, SetEvent 가능)
- Initial Offset: 첫 Expiry point
- Delay: Expiry point 사이의 간격



22-1. Schedule Table

```
SCHEDULETABLE SchedTab1 {
    COUNTER = counter1;
    DURATION = 10;
    REPEATING = TRUE;
    AUTOSTART = TRUE {
        START_VALUE = 5;
    };
    EXPIRE_POINT = ACTION {
        EXPIRE_VALUE = 0;
        ACTION = ACTIVATETASK { TASK = TaskH; };
        ACTION = ACTIVATETASK { TASK = TaskL; };
    };
    EXPIRE_POINT = ACTION {
        EXPIRE_VALUE = 5;
        ACTION = ACTIVATETASK { TASK = TaskH; };
        ACTION = SETEVENT { TASK = TaskL; EVENT = Event1; };
    };
};
```

Cycle Time

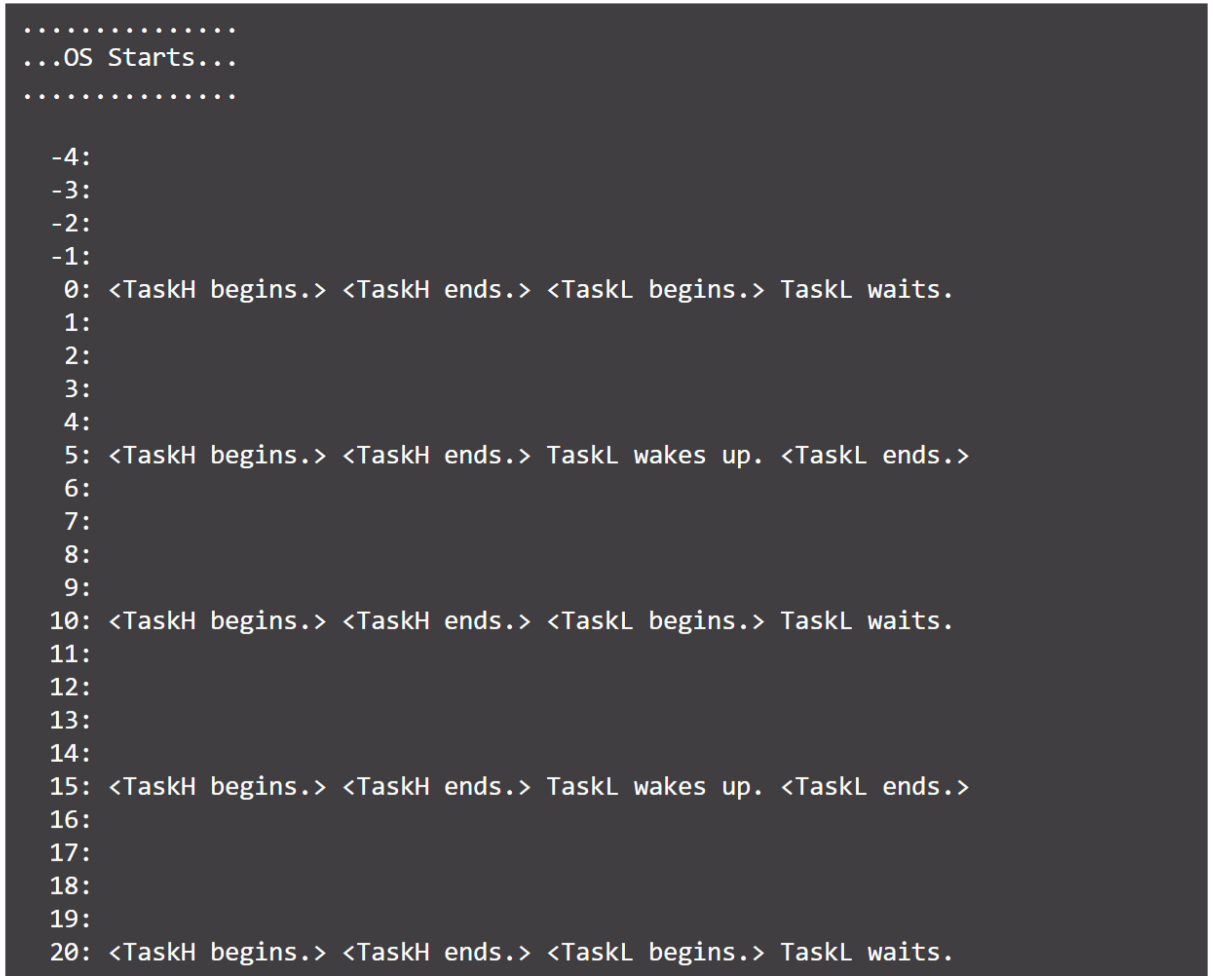
Start Offset

Task Activation

Event Setting

22-1. Schedule Table

• 실행 결과

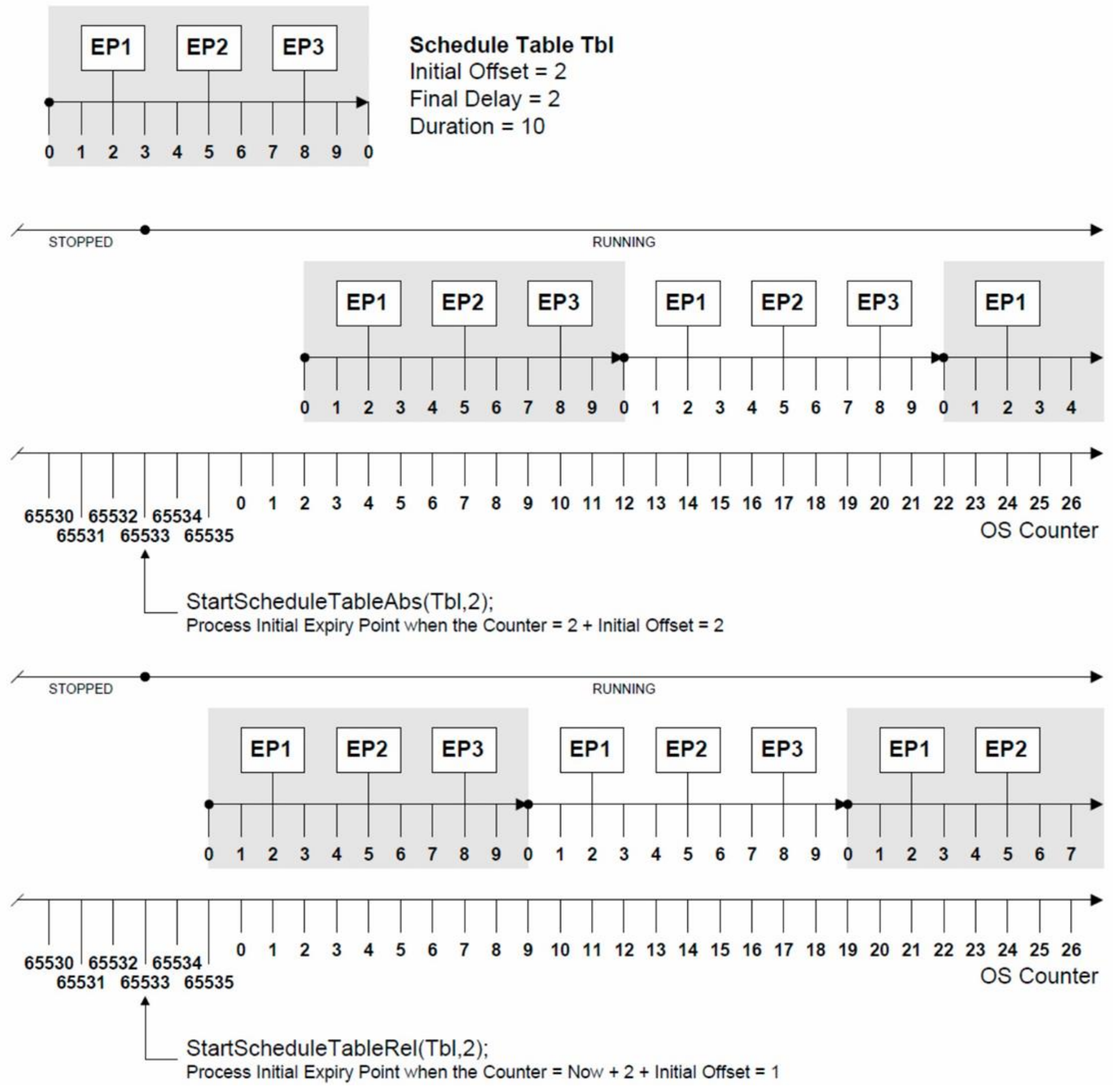


```
...............
...OS Starts...
...............

  -4:
  -3:
  -2:
  -1:
   0: <TaskH begins.> <TaskH ends.> <TaskL begins.> TaskL waits.
   1:
   2:
   3:
   4:
   5: <TaskH begins.> <TaskH ends.> TaskL wakes up. <TaskL ends.>
   6:
   7:
   8:
   9:
  10: <TaskH begins.> <TaskH ends.> <TaskL begins.> TaskL waits.
  11:
  12:
  13:
  14:
  15: <TaskH begins.> <TaskH ends.> TaskL wakes up. <TaskL ends.>
  16:
  17:
  18:
  19:
  20: <TaskH begins.> <TaskH ends.> <TaskL begins.> TaskL waits.
```

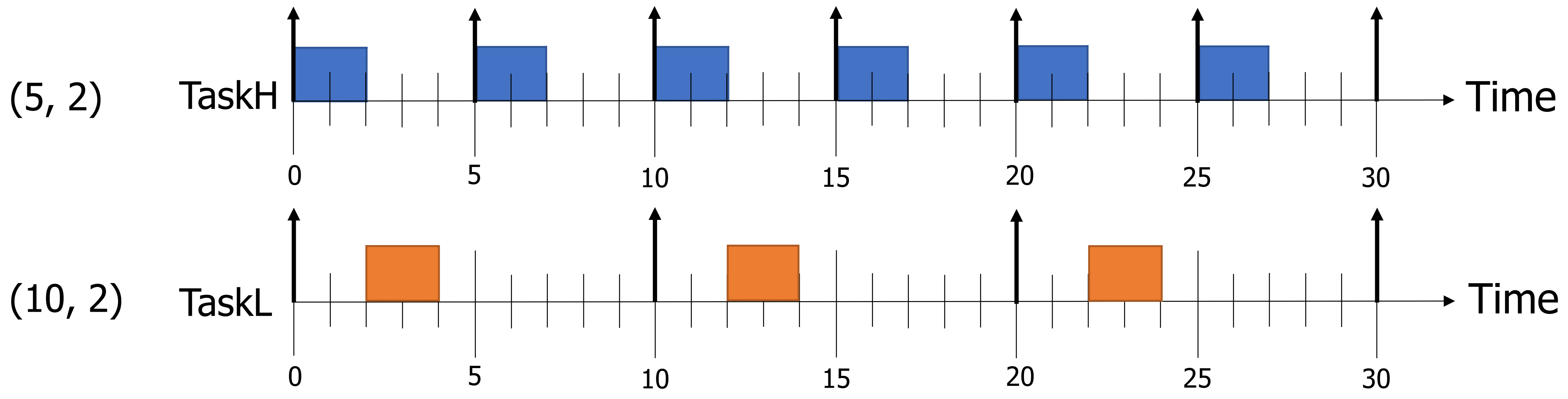
Schedule Table Handling Functions

- StartScheduleTableRel()
- StartScheduleTableAbs()
- StopScheduleTable()


EP1
EP2
EP3
Schedule Table Tbl
Initial Offset = 2
Final Delay = 2
Duration = 10
STOPPED
RUNNING
OS Counter
StartScheduleTableAbs(Tbl,2);
Process Initial Expiry Point when the Counter = 2 + Initial Offset = 2
StartScheduleTableRel(Tbl,2);
Process Initial Expiry Point when the Counter = Now + 2 + Initial Offset = 1

22-2. Schedule Table

- [예제] 2개 Task 생성 및 Scheduling 실험
- Schedule Table을 활용하여 아래 Timing diagram 구현하기



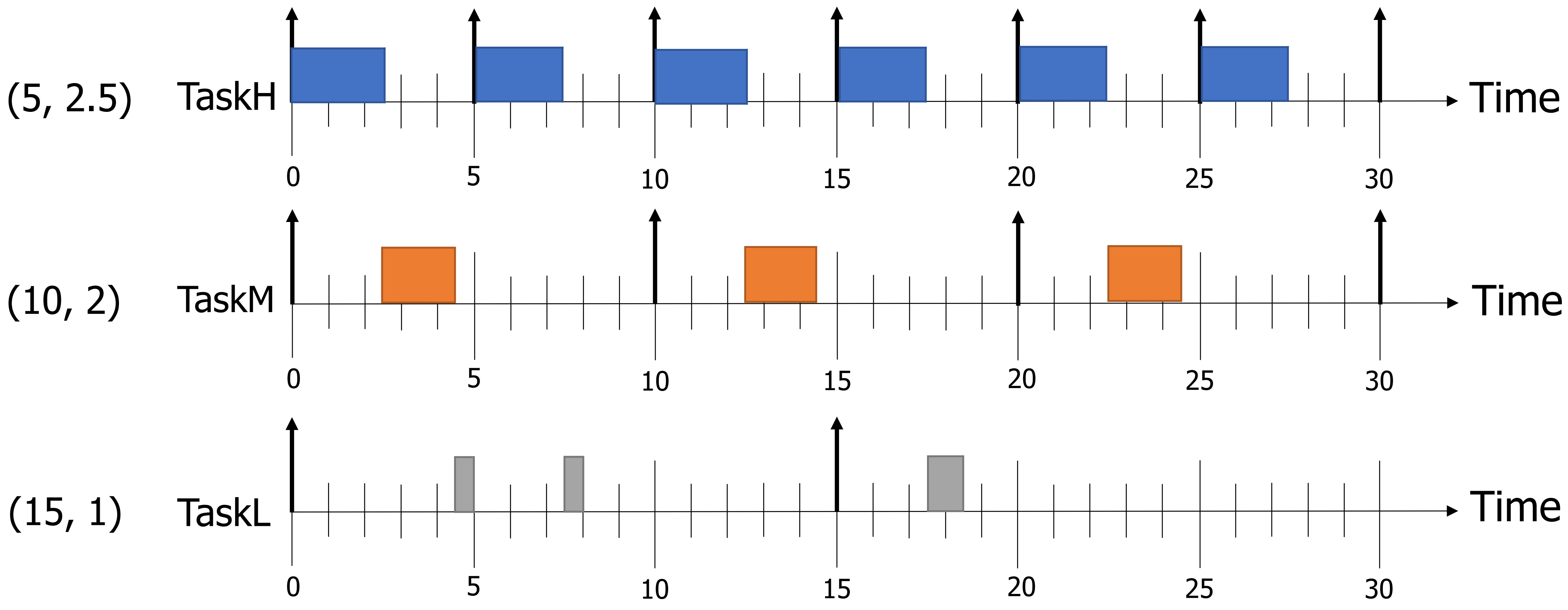
22-2. Schedule Table

- [예제] 2개 Task 생성 및 Scheduling 실험

```
SCHEDULETABLE SchedTab1 {
       COUNTER = mycounter;
       DURATION = 10;
       REPEATING = TRUE;
       AUTOSTART = TRUE {
           START_VALUE = 5;
       };
       EXPIRE_POINT = ACTION {
           EXPIRE_VALUE = 0;
           ACTION = ACTIVATETASK { TASK = TaskH; };
           ACTION = ACTIVATETASK { TASK = TaskL; };
       };
       EXPIRE_POINT = ACTION {
           EXPIRE_VALUE = 5;
           ACTION = ACTIVATETASK { TASK = TaskH; };
       };
   };
```

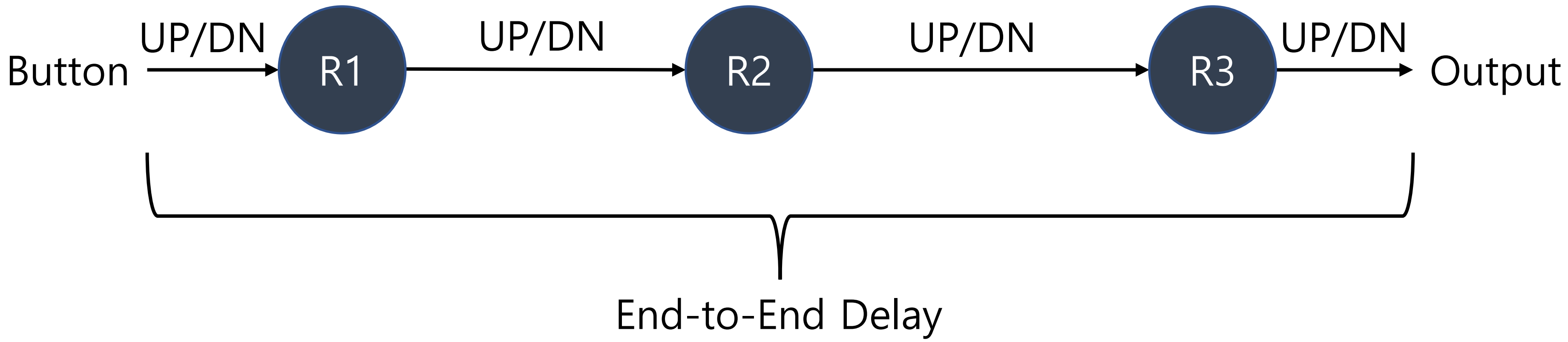
22-3. Schedule Table

- [예제] 3개 Task 생성 및 Scheduling 실험
- Schedule Table을 활용하여 아래 Timing diagram 구현하기



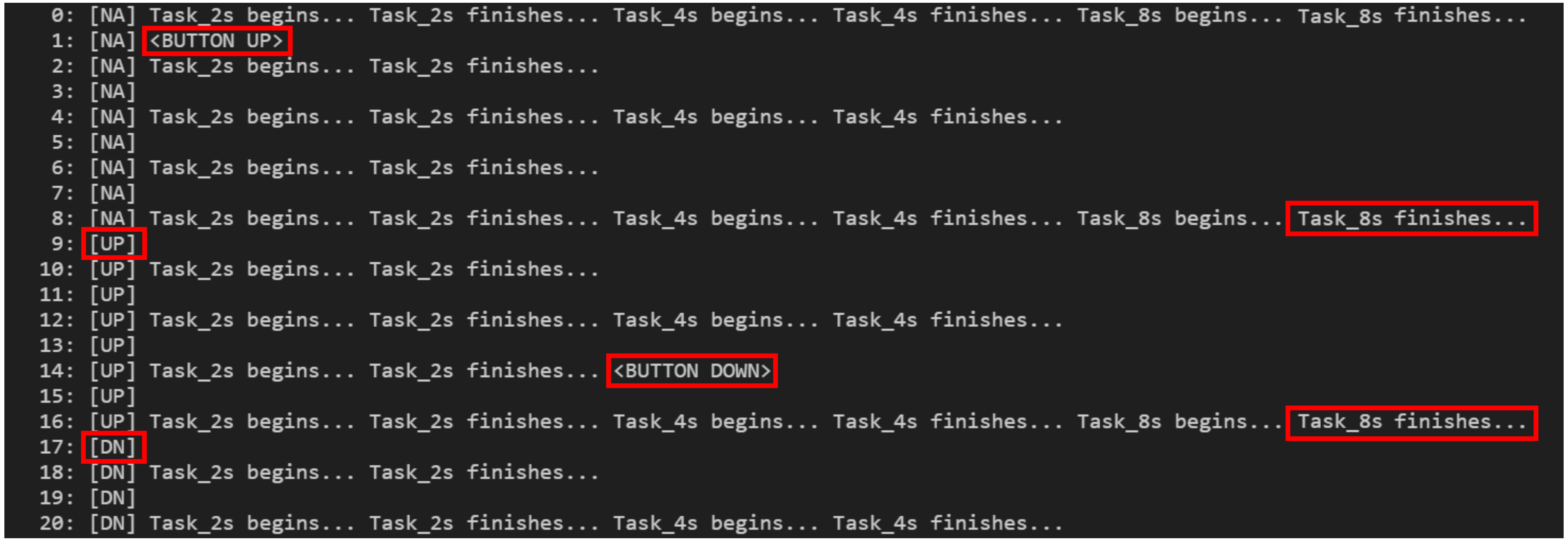
23-1. End-to-End Delay

- AUTOSAR 기반 DAG (Directed Acyclic Graph) SW 구조
- Runnable to Task 매핑 & 시퀀싱
- Sensor에서 Actuator까지 End-to-End Delay 관찰



23-1. End-to-End Delay

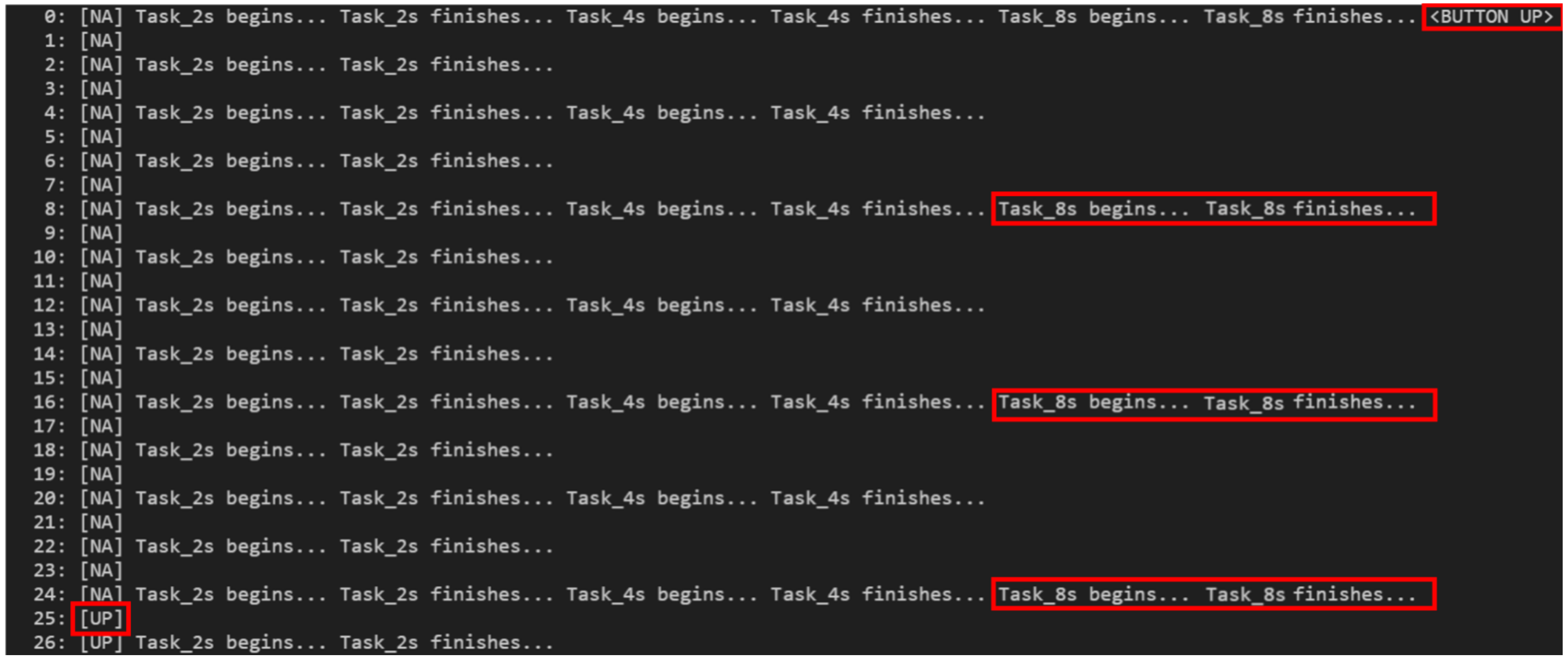
- 버튼 입력 시각에 따른 End-to-End Delay 차이 존재



```
 0: [NA] Task_2s begins... Task_2s finishes... Task_4s begins... Task_4s finishes... Task_8s begins... Task_8s finishes...
 1: [NA] <BUTTON UP>
 2: [NA] Task_2s begins... Task_2s finishes...
 3: [NA]
 4: [NA] Task_2s begins... Task_2s finishes... Task_4s begins... Task_4s finishes...
 5: [NA]
 6: [NA] Task_2s begins... Task_2s finishes...
 7: [NA]
 8: [NA] Task_2s begins... Task_2s finishes... Task_4s begins... Task_4s finishes... Task_8s begins... Task_8s finishes...
 9: [UP]
10: [UP] Task_2s begins... Task_2s finishes...
11: [UP]
12: [UP] Task_2s begins... Task_2s finishes... Task_4s begins... Task_4s finishes...
13: [UP]
14: [UP] Task_2s begins... Task_2s finishes... <BUTTON DOWN>
15: [UP]
16: [UP] Task_2s begins... Task_2s finishes... Task_4s begins... Task_4s finishes... Task_8s begins... Task_8s finishes...
17: [DN]
18: [DN] Task_2s begins... Task_2s finishes...
19: [DN]
20: [DN] Task_2s begins... Task_2s finishes... Task_4s begins... Task_4s finishes...
```

23-1. End-to-End Delay

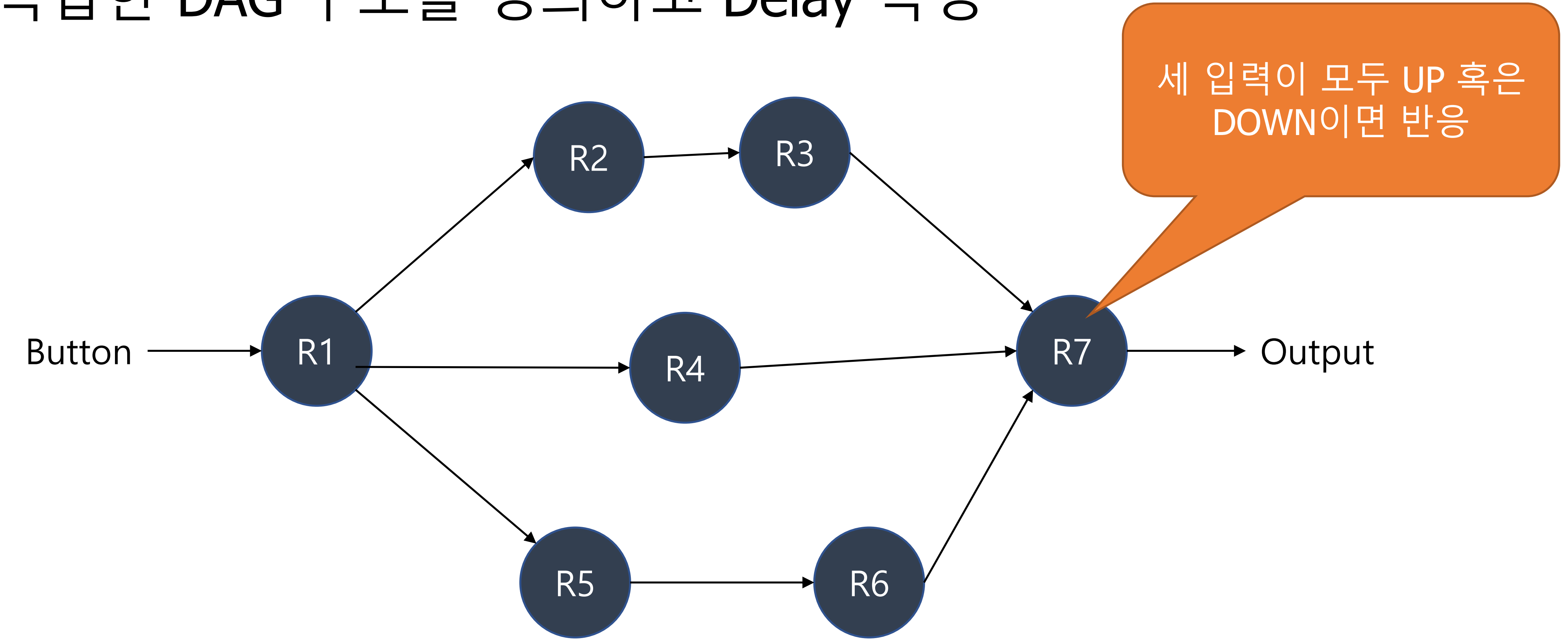
- Runnable 순서를 반대로 변경
- 순서에 따른 Reaction 지연 발생 확인



```
 0: [NA] Task_2s begins... Task_2s finishes... Task_4s begins... Task_4s finishes... Task_8s begins... Task_8s finishes... <BUTTON UP>
 1: [NA]
 2: [NA] Task_2s begins... Task_2s finishes...
 3: [NA]
 4: [NA] Task_2s begins... Task_2s finishes... Task_4s begins... Task_4s finishes...
 5: [NA]
 6: [NA] Task_2s begins... Task_2s finishes...
 7: [NA]
 8: [NA] Task_2s begins... Task_2s finishes... Task_4s begins... Task_4s finishes... Task_8s begins... Task_8s finishes...
 9: [NA]
10: [NA] Task_2s begins... Task_2s finishes...
11: [NA]
12: [NA] Task_2s begins... Task_2s finishes... Task_4s begins... Task_4s finishes...
13: [NA]
14: [NA] Task_2s begins... Task_2s finishes...
15: [NA]
16: [NA] Task_2s begins... Task_2s finishes... Task_4s begins... Task_4s finishes... Task_8s begins... Task_8s finishes...
17: [NA]
18: [NA] Task_2s begins... Task_2s finishes...
19: [NA]
20: [NA] Task_2s begins... Task_2s finishes... Task_4s begins... Task_4s finishes...
21: [NA]
22: [NA] Task_2s begins... Task_2s finishes...
23: [NA]
24: [NA] Task_2s begins... Task_2s finishes... Task_4s begins... Task_4s finishes... Task_8s begins... Task_8s finishes...
25: [UP]
26: [UP] Task_2s begins... Task_2s finishes...
```

23-2. End-to-End Delay

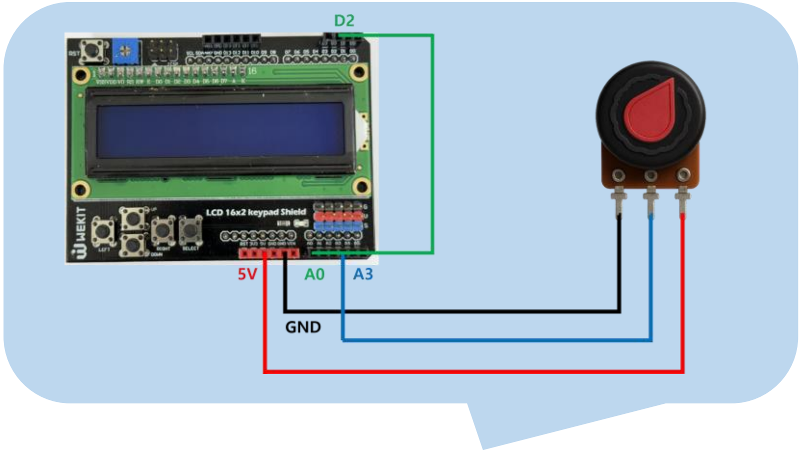
- 아래 복잡한 DAG 구조를 정의하고 Delay 측정

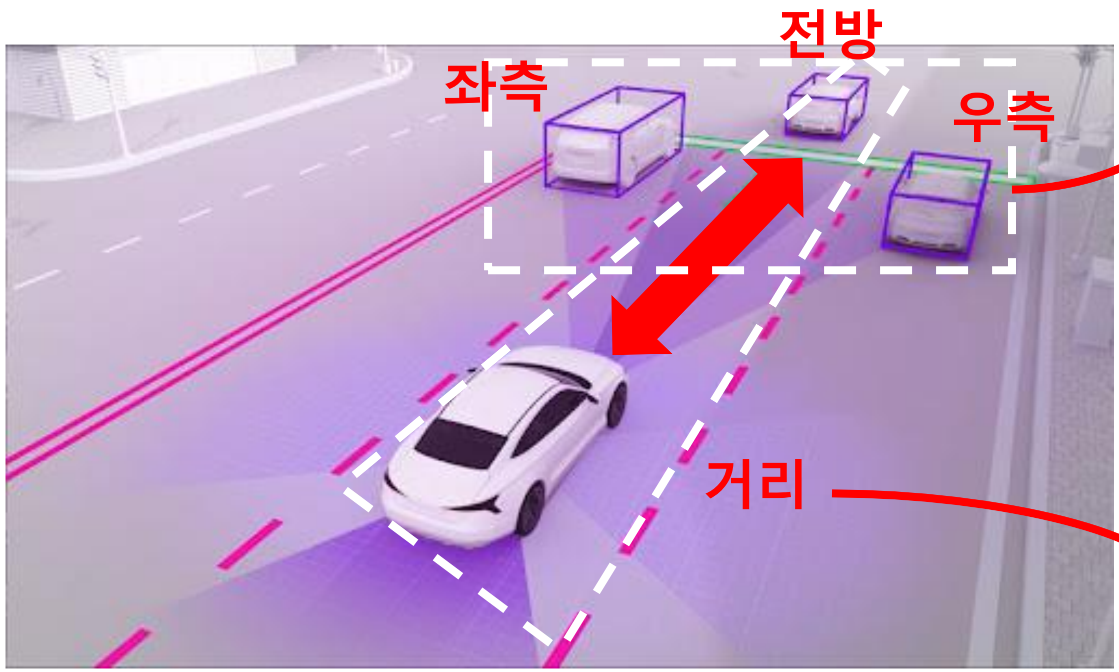


Team Project

- RTOS 기반 자율주행 장애물 감지 및 회피 시스템
- 목표:
  - ✓ RTOS에서 ISR, Task, Event, Resource 구조 이해
  - ✓ 자율주행 시스템처럼 Event 기반 판단 및 회피 전략 구성
  - ✓ 실제 센서(Button/가변저항)와 연동된 Task 스케줄링 실습

가변저항 연결 (점퍼선 3개)





Sensor 1

<LCD키패드쉴드>
Camera
객체 방향

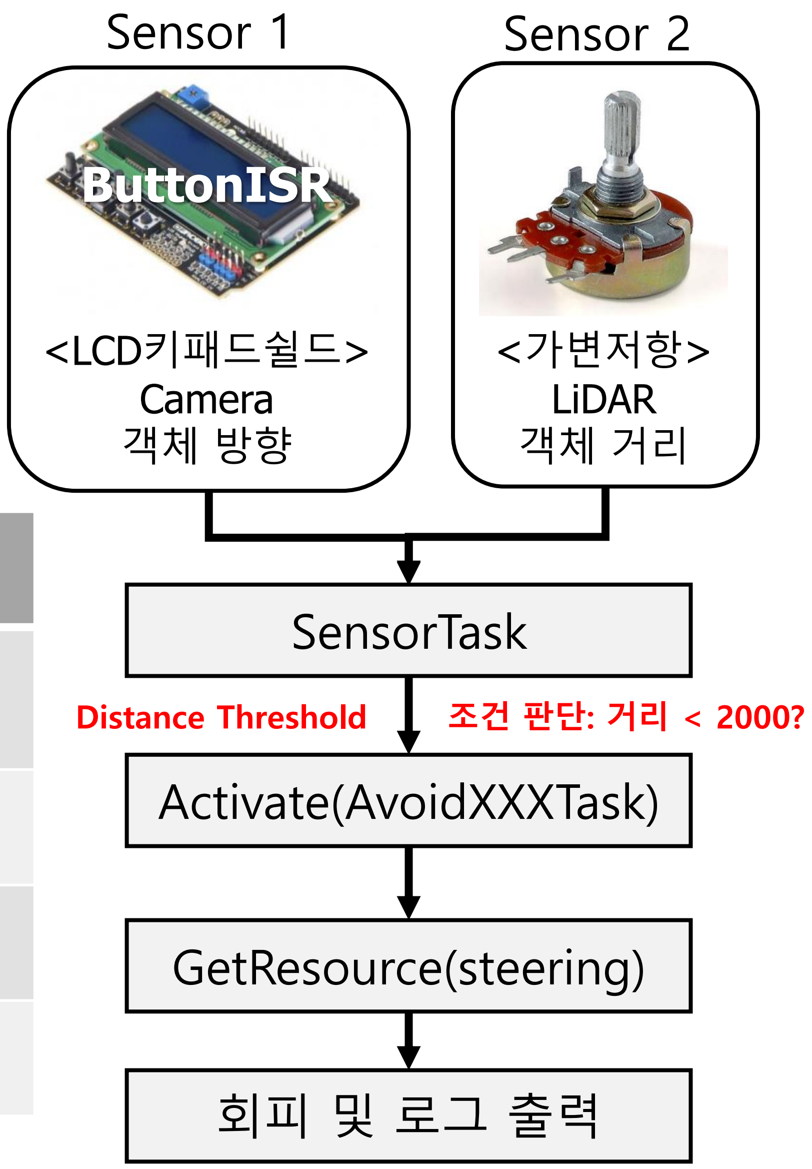
Sensor 2

<가변저항>
LiDAR
객체 거리

Team Project

- 시스템 시나리오
  1. 자율주행 차량이 주행 중
  2. 전방 / 좌측 / 우측에 장애물이 감지되면
     - → 거리가 일정 값 미만이면 회피
     - → 그렇지 않으면 무시
  3. 회피 동작은 반드시 자원을 점유하고 단독 실행

Task 이름	우선순위	기능 설명	공유 자원
SensorTask	3	방향 및 거리 판단, AvoidTask 결정	없음
AvoidFrontTask	4	전방 회피	steering_control
AvoidLeftTask	2	좌측 회피	steering_control
AvoidRightTask	2	우측 회피	steering_control



Questions