# 집중교육 AUTOSAR 프로젝트

## 귀여운 고슴도치

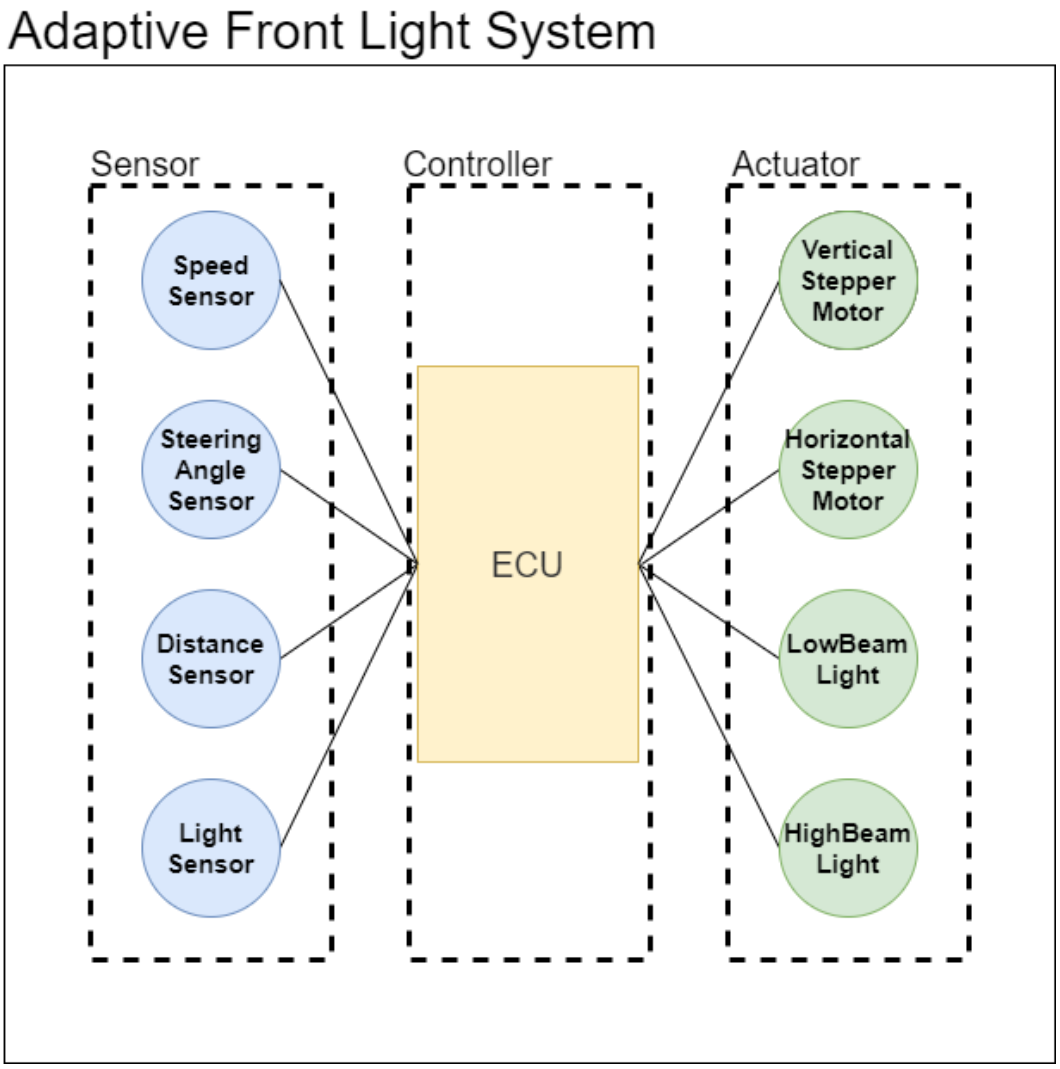
201420988 강현구
201421002 한수찬
201620486 오승민
201620950 강한결

# 목차

# 표 목차

# 1 발표 주제 시스템 소개

## 1.1 시스템 하드웨어 구조




**[그림 1] 시스템 하드웨어 구조도**

본 과제에서 개발하고자 하는 시스템의 하드웨어 구조는 [그림 1]과 같다. 개발하고자 하는 시스템은 크게 센서, 컨트롤러, 액추에이터 패턴을 적용하여 하드웨어 구조를 표현하였다. 센서에는 조도 센서(Light Sensor), 속도 센서(Speed Sensor), 스티어링 앵글 센서(Steering Angle Sensor), 거리센서(Distance Sensor)가 있다. 그리고 액추에이터에는 상향등(HighBeam Light), 하향등(LowBeam Light), 상하 스텝모터(Vertivcal or Up-Down Stepper Motor), 좌우 스텝모터(Horizontal or Left-RIght Stepper Motor)가 있다. 마지막으로, ECU는 컨트롤러에 해당한다.

### 1.2 시스템 기능

차량 전조등 자동 제어 시스템(Adjustive Front light System : AFS)을 주제로 하였다. 차량의 실제 주행 환경에 따라 전조등의 조사각과 세기를 조절하여 운전자에게 최적의 전방 시야를 제공한다. 조사각은 상하각도 뿐만 아니라 좌우 각도도 조절된다. 차량이 커브를 돌 때에는 조향 방향에 맞추어 전조등이 좌우로 회전하고 고속 운전시에는 시야 확보를 위해 전조등의 하향각이 작아진다. 또한 실제 조도에 따라 전조등의 밝기가 조절된다. 상향등을 사용하는 상황에서는 전방에 차량이 감지되면 자동으로 하향등으로 전환하고, 감지된 차량이 사라지면 다시 상향등으로 전환한다.

## 2 요구사항 정의

### 2.1 시스템 요구사항

| SR-001 | |
|---|---|
| 이름 | 스티어링 각도에 따른 전조등 좌우 각도 조절 |
| 정의 | 시스템은 차량 주행시 핸들의 조향 방향에 맞추어 전조등의 조사 방향을 변경할 수 있어야 한다. |
| HW | 센서 : Steering Angle Sensor / 액추에이터 : Horizontal Stepper Motor |

**[표 1] 시스템 요구사항 1**

| SR-002 | |
|---|---|
| 이름 | 운행 속도에 따른 전등 상하 각도 조절 |
| 정의 | 시스템은 차량 주행 속도에 따라 하향등의 조사 각이 상승하여 고속 주행 시 전방시야 확보에 도움을 줄 수 있어야 한다.. |
| HW | 센서 : Speed Sensor / 액추에이터 : Vertical Stepper Motor |

**[표 2] 시스템 요구사항 2**

| SR-003 | |
|---|---|
| 이름 | 주행 중 조도에 따른 전조등 밝기 조절 |
| 정의 | 시스템은 실제 조도를 측정하여 전조등의 밝기를 조절할 수 있어야 한다. |
| HW | 센서 : Light Sensor / 액추에이터 : LowBeam Light |

**[표 3] 시스템 요구사항 3**

| SR-004 | |
|---|---|
| 이름 | 전방 차량 유무에 따른 상향등 자동 제어 |
| 정의 | 시스템은 상향등 사용 시 전방 차량이 감지되면 자동으로 하향등으로 변경하고 감지된 차량이 사라지면 다시 상향등으로 변경할 수 있어야 한다. |
| HW | 센서 : Distance Sensor / 액추에이터 : HighBeam Sensor |

**[표 4] 시스템 요구사항 4**

## 2.2 소프트웨어 기능 요구사항

| FR-001 | 관련 요구사항 | SR-001 |
|---|---|---|
| 이름 | Steering Sensor Component로 측정된 전압 변화량 값 전달 | |
| 정의 | Steering Sensor Component는 ECU_ABS에게 Steering Angle Sensor로부터 측정된 전압 변화량 값을 요청하고 받을 수 있어야 한다. | |

**[표 5] 소프트웨어 기능 요구사항 1**

| FR-002 | 관련 요구사항 | SR-001 |
|---|---|---|
| 이름 | Motor Application Component로 수신받은 전압 변화량 값 전달 | |
| 정의 | Steering Sensor Component는 ECU_ABS에게 수신받은 전압 변화량 값을 Motor Appliciation Component로 전달할 수 있어야 한다. | |

**[표 6] 소프트웨어 기능 요구사항 2**

| FR-003 | 관련 요구사항 | SR-001 |
|---|---|---|
| 이름 | Motor Application Component에서 조향각 값 검출 | |
| 정의 | Motor Application Component는 Steering Sensor Component로부터 전달받은 전압 변화량 값으로 조향각 값을 검출할 수 있어야 한다. | |

**[표 7] 소프트웨어 기능 요구사항 3**

| FR-004 | 관련 요구사항 | SR-001 |
|---|---|---|
| 이름 | Left-Right Motor Actuator Component로 검출된 조향각 값 전달 | |
| 정의 | Motor Appliciation Component는 Left-Right Motor Actuator Component에게 검출된 조향각 값을 전달할 수 있어야 한다. | |

**[표 8] 소프트웨어 기능 요구사항 4**

| FR-005 | | 관련 요구사항 | SR-001 |
|---|---|---|---|
| 이름 | ECU_ABS로 검출된 조향각 값 전달 | | |
| 정의 | ECU_ABS는 Left-Right Motor Actuator Component에게 검출된 조향각 값을 요청하고 받을 수 있어야 한다. | | |

**[표 9] 소프트웨어 기능 요구사항 5**

| FR-006 | | 관련 요구사항 | SR-002 |
|---|---|---|---|
| 이름 | Speed Sensor Component로 측정된 RPM 값 전달 | | |
| 정의 | Speed Sensor Component는 ECU_ABS에게 Speed Sensor로부터 측정된 회전수 값을 요청하고 받을 수 있어야 한다. | | |

**[표 10] 소프트웨어 기능 요구사항 6**

| FR-007 | | 관련 요구사항 | SR-002 |
|---|---|---|---|
| 이름 | Motor Application Component로 수신받은 회전수 값 전달 | | |
| 정의 | Speed Sensor Component는 ECU_ABS에게 수신받은 회전수 값을 Motor Appliciation Component로 전달할 수 있어야 한다. | | |

**[표 11] 소프트웨어 기능 요구사항 7**

| FR-008 | | 관련 요구사항 | SR-002 |
|---|---|---|---|
| 이름 | Motor Application Component에서 속도 값 검출 | | |
| 정의 | Motor Application Component는 Speed Sensor Component로부터 전달받은 회전수 값으로 속도 값을 검출할 수 있어야 한다. | | |

**[표 12] 소프트웨어 기능 요구사항 8**

| FR-009 | | 관련 요구사항 | SR-002 |
|---|---|---|---|
| 이름 | Up-Down Motor Actuator Component로 상하 조사각 단계 값 전달 | | |
| 정의 | Motor Appliciation Component는 Up-Down Motor Actuator Component에게 검출된 속도 값으로부터 구분된 조사각 단계 값을 전달할 수 있어야 한다. | | |

**[표 13] 소프트웨어 기능 요구사항 9**

| FR-010 | | 관련 요구사항 | SR-002 |
|---|---|---|---|
| 이름 | ECU_ABS로 검출된 조사각 값 전달 | | |
| 정의 | ECU_ABS는 Up-Down Motor Actuator Component에게 검출된 조사각 값을 요청하고 받을 수 있어야 한다. | | |

**[표 14] 소프트웨어 기능 요구사항 10**

| FR-011 | | 관련 요구사항 | SR-003 |
|---|---|---|---|
| 이름 | Light Sensor Component로 측정된 저항 값 전달 | | |
| 정의 | Light Sensor Component는 ECU_ABS에게 Light Sensor로부터 측정된 저항 값을 요청하고 받을 수 있어야 한다. | | |

**[표 15] 소프트웨어 기능 요구사항 11**

| FR-012 | | 관련 요구사항 | SR-003 |
|---|---|---|---|
| 이름 | Low Beam Application Component로 수신 받은 전압 변화량 값 전달 | | |
| 정의 | Light Sensor Component는 ECU_ABS에게 수신 받은 저항 값을 Low Beam Application Component로 전달할 수 있어야 한다. | | |

**[표 16] 소프트웨어 기능 요구사항 12**

| FR-013 | | 관련 요구사항 | SR-003 |
|---|---|---|---|
| 이름 | Low Beam Application Component에서 필요한 밝기 값 검출 | | |
| 정의 | Low Beam Application Component는 Light Sensor Component로부터 전달받은 저항 값을 필요한 밝기 값으로 검출할 수 있어야 한다. | | |

**[표 17] 소프트웨어 기능 요구사항 13**

| FR-014 | | 관련 요구사항 | SR-003 |
|---|---|---|---|
| 이름 | Low Beam Actuator Component로 검출된 밝기 값 전달 | | |
| 정의 | Low Beam Application Component는 Low Beam Actuator Component에게 검출된 밝기 값을 전달할 수 있어야 한다. | | |

**[표 18] 소프트웨어 기능 요구사항 14**

| FR-015 | | 관련 요구사항 | SR-003 |
|---|---|---|---|
| 이름 | ECU_ABS로 검출된 밝기 값 전달 | | |
| 정의 | ECU_ABS는 Low Beam Actuator Component에게 검출된 밝기 값을 요청하고 받을 수 있어야 한다. | | |

**[표 19] 소프트웨어 기능 요구사항 15**

| FR-016 | | 관련 요구사항 | SR-004 |
|---|---|---|---|
| 이름 | Distance Sensor Component로 측정된 펄스 값 전달 | | |
| 정의 | Distance Sensor Component는 ECU_ABS에게 Light Sensor로부터 측정된 펄스 값을 요청하고 받을 수 있어야 한다. | | |

**[표 20] 소프트웨어 기능 요구사항 16**

| FR-017 | | 관련 요구사항 | SR-004 |
|---|---|---|---|
| 이름 | High Beam Application Component로 수신 받은 전압 변화량 값 전달 | | |
| 정의 | Distance Sensor Component는 ECU_ABS에게 수신 받은 펄스 값을 High Beam Application Component로 전달할 수 있어야 한다. | | |

**[표 21] 소프트웨어 기능 요구사항 17**

| FR-018 | | 관련 요구사항 | SR-004 |
|---|---|---|---|
| 이름 | High Beam Application Component에서 거리 값 검출 | | |
| 정의 | High Beam Application Component는 Distance Sensor Component로부터 전달 받은 펄스 값을 필요한 거리 값으로 검출할 수 있어야 한다. | | |

**[표 22] 소프트웨어 기능 요구사항 18**

| FR-019 | | 관련 요구사항 | SR-004 |
|---|---|---|---|
| 이름 | High Beam Actuator Component로 High Beam Mode 값 전달 | | |
| 정의 | High Beam Application Component는 High Beam Actuator Component에게 검출된 거리 값으로부터 구분된 High Beam Mode 값을 전달할 수 있어야 한다. | | |

**[표 23] 소프트웨어 기능 요구사항 19**

| FR-020 | | 관련 요구사항 | SR-004 |
|---|---|---|---|
| 이름 | ECU_ABS로 검출된 거리 값 전달 | | |
| 정의 | ECU_ABS는 High Beam Actuator Component에게 구분된 High Beam Mode값을 요청하고 받을 수 있어야 한다. | | |

**[표 24] 소프트웨어 기능 요구사항 20**

## 2.3 품질 속성

### 2.3.1 기능성

- 시스템은 상황에 따라 최적의 전방 시야를 제공해야 한다.

### 2.3.2 신뢰성

- 시스템은 센서의 값을 실시간으로 측정할 수 있어야 한다.
- 시스템은 센서의 이상 값을 구별해 낼 수 있어야 한다.

### 2.3.3 정확성

- 센서 측정값의 오차범위는 5% 이내여야한다.

2.3.4 사용성

- 시스템은 모두 자동으로 동작하여야 한다.

2.3.5 유지보수성

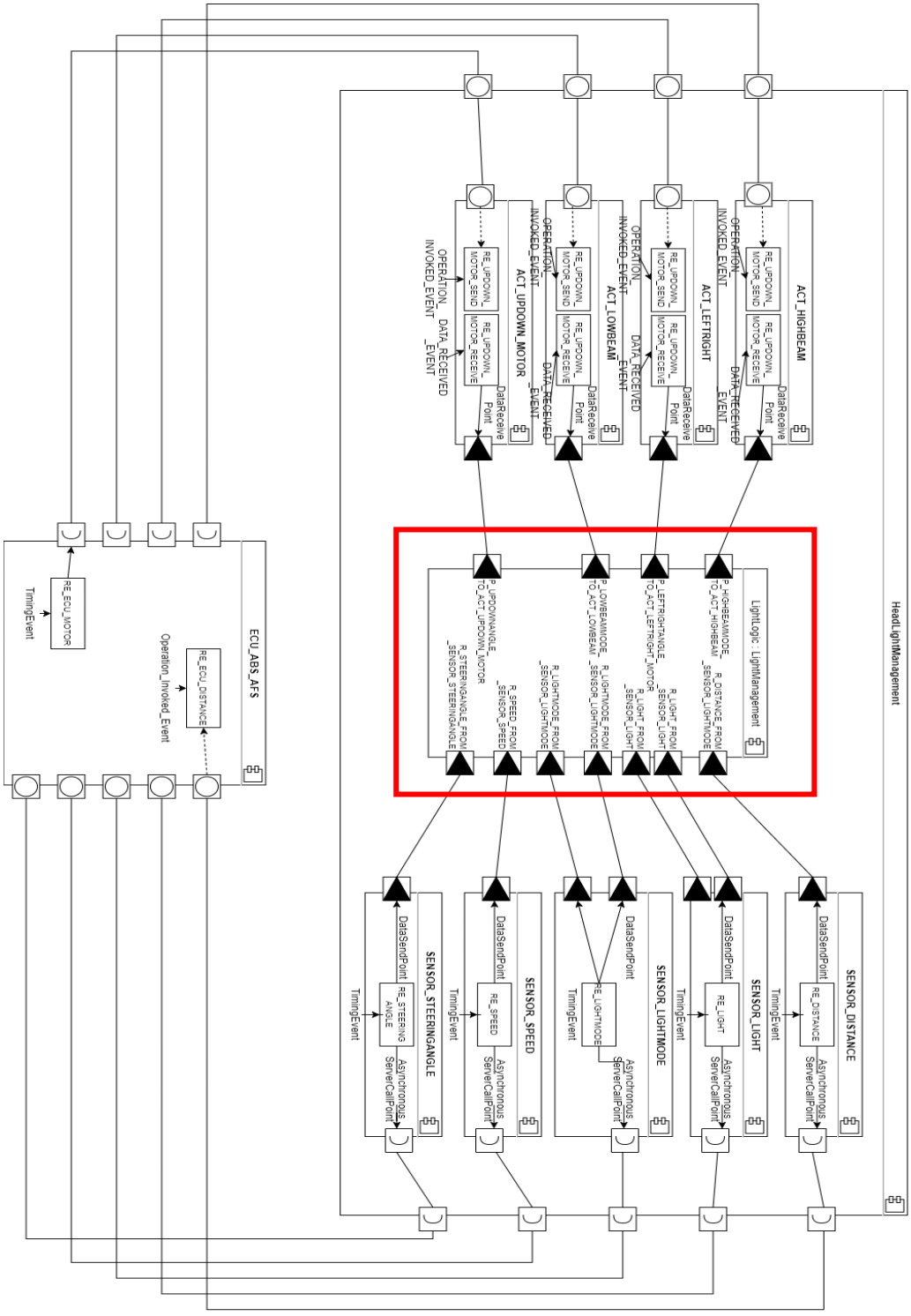
- 시스템은 컴포넌트화 되어 수정하기 쉽게 구현되어야 한다.

2.3.6 이식성

- 시스템은 다른 ECU상에서도 동일하게 작동해야 한다.

# 3 모델 구조

## 3.1 Component구조




**[그림 2] 전체 시스템 구조도**

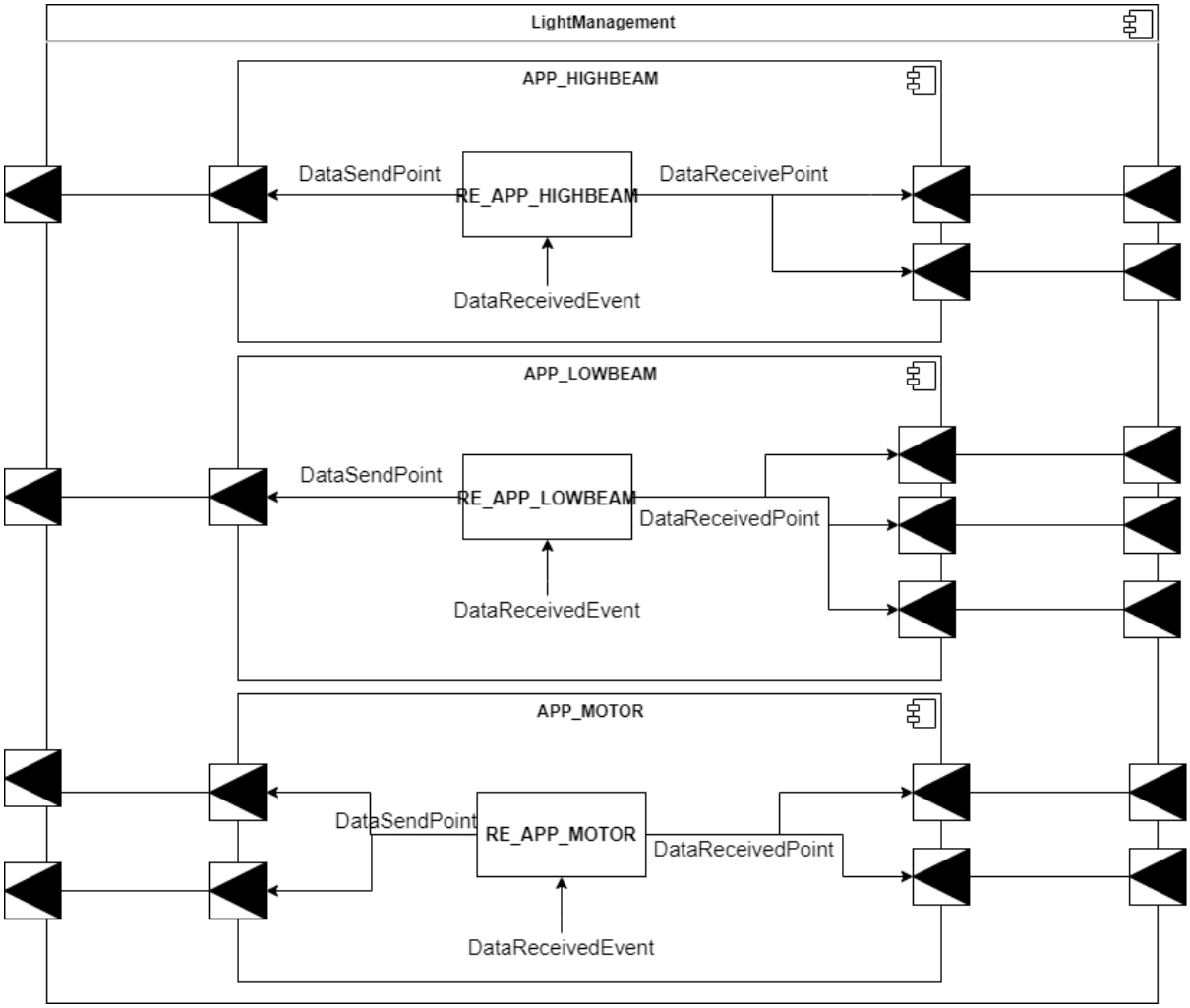
프로젝트에서 설계한 AFS모델은 크게 **Sensor Component, Actuator Component, SW Application Component, ECU ABS Component** 로 구분된다. 위 그림은 AFS모델을 설계한 [그림 2]이다. **HeadLightManagement**라는 Type안에 Sensor 값을 통해 기능을 구현하는 **LightManagement Prototype**, 센서의 값을 읽는 **SENSOR_*** 네이밍 형태의 prototype, **ACT_*** 네이밍 형태의 prototype 등이 존재한다. 여기서 **LightManagement**는 **APP_HIGHBEAM, APP_LOWBEAM, APP_MOTOR**의 Atomic SW Component로 구성되는 Composite Component이다. 이때, **ECU_ABS_AFS**는 별도의 Atomic Component로 나누어 설계하였다.

**ECU ABS Component**는 Sensor HW나 Actuator HW의 값을 읽고 쓰기 위해 모든 **Sensor Actuator Component**에 연결되어 있다. 모두 **Client-Server Interface**로 연결되어 있으며 Actuator와 통신할 때에는 Client로써 Timing Event를 통해 0.1초마다 Call Request를 보내어 데이터를 받는다. Sensor와 통신할 때에는 Server로써 역할을 통해 데이터를 전송해준다. 이 때에는 **Operation Invoked Event**를 통해 RE가 깨어난다.

**Sensor Component**는 **LightManagement**라는 **SW Application Component**와 **ECU_ABS_AFS**라는 **ECU ABS Component**와 통신한다. **LightManagement**와 통신할 때는 **Sender-Receiver Interface**로 통신을 한다. Sender의 역할을 하는 **Sensor Component**는 Data Send Point를 통해 하여 Sensing한 값을 넘겨준다. **ECU ABS Component**와 통신할 때는 **Client-Server Interface**를 통해 통신하며 Asynchronous Server Call Point를 이용하여 Client의 역할을 한다. 여기서 Asynchronous 방식을 사용한 이유는 AFS 시스템은 사용자의 안전과 직결되는 기능이 아닌 편의를 위한 부가적인 기능이기 때문에 데이터 무결성의 중요도가 낮다고 판단하였기 때문이다. 따라서 정밀한 작동 보다는 작동 유무에 초점을 두어 Asynchronous 방식을 이용하였다. **Sensor Component**의 Runnable Entity는 Timing Event를 통해 0.1초마다 작동한다.

**Actuator Component**역시 **LightManagement**와 **ECU_ABS_AFS**와 연결된다. **Sender-Receiver Interface**로 **LightManagement**와 통신하며 Data Received Point를 이용해 Receiver로써의 역할을 한다. **LightManagement**가 data를 전송하면 **Data Received Event**를 통해 RE가 깨어난다. **ECU_ABS_AFS**와는 **Client-Server Interface**로 통신하며 Synchronous Server Call Point를 이용하여 Server로써의 역할을 한다. 여기서 Synchronous 방식을 사용한 이유는 실제 값이 있을 때 Actuator를 작동시켜야 하기 때문이다. **ECU_ABS_AFS**가 Call Request를 보내면 **Operation Invoked Event**로 RE가 깨어

나서 Data를 send해준다.




**[그림 3] LightManagement Component 구조도**

위 [그림 3]은 [그림 2]의 붉은 사각형 내부의 구조도이다.

**LightManagement Component**는 **APP_HIGHBEAM, APP_LOWBEAM, APP_MOTOR**라는 3개의 Atomic SW Application Component로 구성된 **Composite Component**이다. Sensor Component가 아닌 Application Component에 센서 값을 정제하는 기능을 포함시킴으로써 Sensor Component에서의 Asynchronous 호출속도 지연을 감소시키고 Component 독립성을 높여 재사용성 및 유지보수성을 향상시켰다.

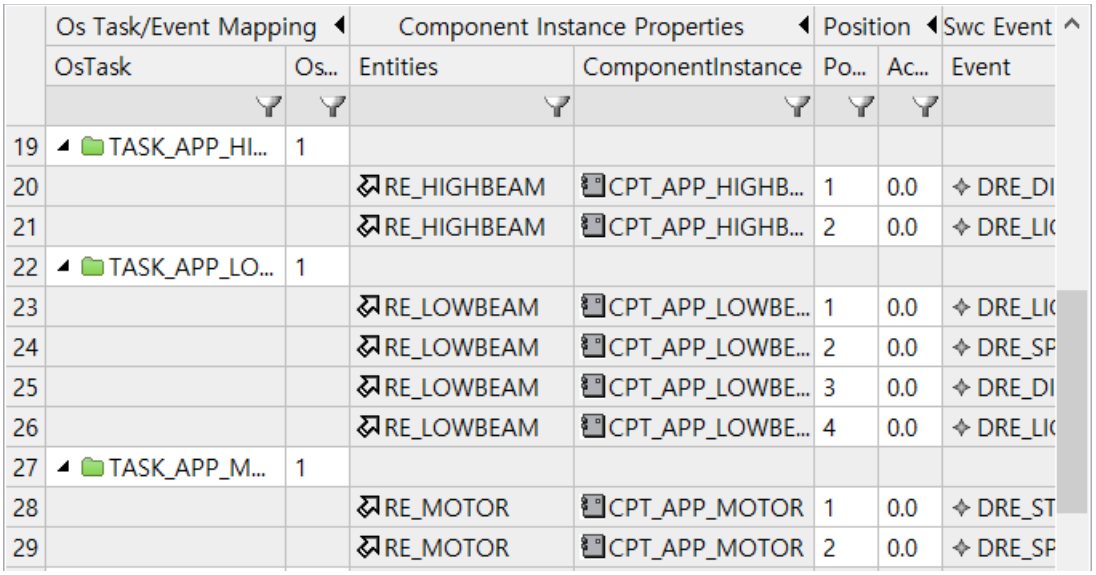
**APP_HIGHBEAM Component**는 조건에 따라 상향등의 On/Off를 조절한다. 조건들은 상

대 차량과의 거리를 알기 위한 **SENSOR_DISTANCE**와 운전자가 상향등을 켰는지 하향등을 켰는지에 대한 상태를 알기 위한 **SENSOR_LIGHTMODE**를 통해 Receive되며 이 때 **Sender-Receiver Interface**를 사용한다. Data Reveiced Point를 이용해 Sensor로부터 값을 읽고, 상황을 판단한 뒤, **ACT_HIGHBEAM**으로 Data Send Point를 통해 값을 전송한다. 이 RE은 Sensor로부터 값을 받았을 때만 작동하도록 하여 **Data Received Event**를 통해 깨어난다. **SENSOR_DISTANCE**로부터 받은 거리값은 윈도우 크기가 5인 median 알고리즘을 사용하여 보정함으로써 신뢰성을 높였다.

**APP_LOWBEAM Component**는 조건에 따라 하향등의 빛의 세기를 조절한다. 빛의 세기는 차량의 속도, 조도 센서, 하향등 전원 유무에 따라 조절된다. **SENSOR_SPEED, SENSOR_LIGHT, SENSOR_LIGHTMODE**에서 조건에 대한 값을 Receive 한다. 이 때 역시 **Sender-Receiver Interface**를 사용하며 이 외에도 IB Modeling에 대한 설정은 위의 **APP_HIGHBEAM**의 설정과 같다. **APP_LOWBEAM**은 센서에서 받은 값을 통해 정해진 로직에 따라서 하향등의 세기를 어떻게 조절할지 판단한 뒤 **ACT_LOWBEAM**으로 값을 send한다. **SENSOR_LIGHT**로부터 받은 조도값은 윈도우 크기가 5인 median 알고리즘을 사용하여 보정함으로써 신뢰성을 높였다.

**APP_MOTOR Component**는 조건에 따라 하향등의 방향을 조절한다. 하향등의 방향은 Steering angle과 차량의 속도에 의해 조절된다. Steering angle이 커지면 angle의 크기만큼 하향등도 좌우로 조절해주고 차량의 속도가 빠르면 하향등을 위로 올려 더 멀리까지 볼 수 있게 한다. IB모델링은 **LightManagement**의 prototype component들과 같다.

### 3.2 Task 구조



| | Os Task/Event Mapping | | Component Instance Properties | | Position | | Swc Event |
|---|---|---|---|---|---|---|---|
| | OsTask | Os... | Entities | ComponentInstance | Po... | Ac... | Event |
| 19 | TASK_APP_HI... | 1 | | | | | |
| 20 | | | RE_HIGHBEAM | CPT_APP_HIGHB... | 1 | 0.0 | DRE_DI |
| 21 | | | RE_HIGHBEAM | CPT_APP_HIGHB... | 2 | 0.0 | DRE_LI |
| 22 | TASK_APP_LO... | 1 | | | | | |
| 23 | | | RE_LOWBEAM | CPT_APP_LOWBE... | 1 | 0.0 | DRE_LI |
| 24 | | | RE_LOWBEAM | CPT_APP_LOWBE... | 2 | 0.0 | DRE_SP |
| 25 | | | RE_LOWBEAM | CPT_APP_LOWBE... | 3 | 0.0 | DRE_DI |
| 26 | | | RE_LOWBEAM | CPT_APP_LOWBE... | 4 | 0.0 | DRE_LI |
| 27 | TASK_APP_M... | 1 | | | | | |
| 28 | | | RE_MOTOR | CPT_APP_MOTOR | 1 | 0.0 | DRE_ST |
| 29 | | | RE_MOTOR | CPT_APP_MOTOR | 2 | 0.0 | DRE_SP |

**[그림 4] Task 맵핑 구조도**

Task는 [그림 4] 의 Task 3개를 제외하고는 모두 Task와 Runnable Entity가 1대1로 맵핑되었다. Runnable Entity를 여러 개 포함한 Task들은 해당 Task를 실행하여 얻는 하나의 Output을 위해 여러 데이터가 같은 영역에 필요하기 때문에 한 Task에 맵핑하였다. 그 외 Task들은 재사용성과 독립성을 위하여 Runnable Entity와 1대1로 맵핑하였다.

# 4 설계 근거

## 4.1 인터페이스 선택 이유

Server-Client : 실제 센서로부터 측정된 값을 센서 컴포넌트가 받아오는 과정에 필요하다.

Sender-Receiver : 컴포넌트간 데이터 전송을 위해 필요하다.

## 4.2 인터페이스 활용 이유

Server-Client : 센서 컴포넌트에서 실제 센서에서 측정된 값을 받아오기 위해서는 ECU_ABS에 존재하는 센서 측정 API를 호출해야 한다. 따라서 센서 컴포넌트에서 ECU_ABS로 요청을 보내야 하고 이를 위해 Server – Client Interface를 활용하였다. 또한 실제 액추에이터를 작동시키기 위해서는 액추에이터 컴포넌트로부터 값을 받아야 하는데 ECU_ABS에서 액추에이터 컴포넌트에 값을 요청하여 실에 액추에이터로 보내주게 된다. 마찬가지로 Server -Client Interface를 활용하여 이를 구현하였다.

Sender-Receiver : 컴포넌트간 데이터 송수신을 구현하기 위해 Sender – Reveiver Interface를 활용하였다.

## 4.3 컴포넌트 및 Runnable Entity 구성 이유

AFS System에서 Component는 Sensor, Actuator, Application, ECU등의 기능별로 가장 크게 분류된다. 그 다음 Sensor별로, Actuator별로 Application 로직 별로 각각의 컴포넌트들을 다시 나누어 주었다. 이렇게 여러 개의 Component로 나눈 이유는 Exchangeability 와 Reusability를 위해서이다. Component를 작게 나눌수록 다른 project에 가져가서 사용할 수도 있고 로직이 바뀌었을 때도 필요한 부분만 쉽게 찾아서 수정해주면 다른 코드들은 수정하지 않아도 되기 때문에 일의 효율적인 측면에서 매우 긍정적인 결과를 얻을 수 있다. 또한, Runnable Entity를 합한 경우는 ECU_ABS_AFS에서 ACT_* Component들의 값을 읽기 위해 Call을 보내는 경우가 있는데 이 때는 모든 ACT의 값을 항상 읽어와야 함으로 모든 Event를 한번에 실행시켜도 괜찮다. 이렇게 RE를 합칠 경우 Task의 개수를 줄일 수 있어서 구조를 더 간단하게 만들 수 있는 장점이 있다. 한 편, RE를 각각의 Event마다 나누는 경우도 있었는데, SENSOR_* Component에서 ECU_ABS_AFS에게 Request Call을 보낼 때

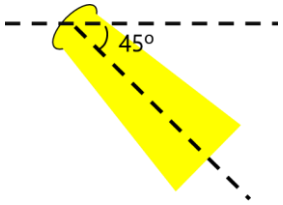
Operation Invoked Event로 이벤트가 발생하는데 이 때에는 각각의 센서들마다 값을 읽어오도록 동작하는 것이 하나의 센서가 값을 요청할 때 필요 없는 센서들의 값까지 읽어오는 것 보다 효율적이므로 Runnable Entity를 나누어 구현하였다.

# 5 시뮬레이션 결과

시뮬레이션 시나리오

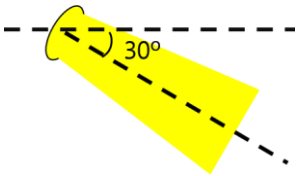
1. 정지된 차량에서 하향등을 킨다.

```
Status.Read: [ECU] 로우빔의 세기는 1 입니다.
Status.Lines: 3
Status.Read: [ECU] 좌우모터가 0 각도 만큼 변경되었습니다.
Status.Lines: 4
Status.Read: [ECU] 상하모터가 45 각도 만큼 변경되었습니다.
Status.Lines: 5
```



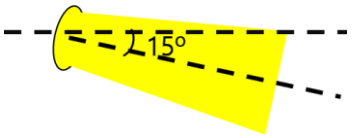

2. 하향등을 킨 상태로 시속 60km로 직진한다.

```
Status.Read: [ECU] 상하모터가 30 각도 만큼 변경되었습니다.
Status.Lines: 6
```



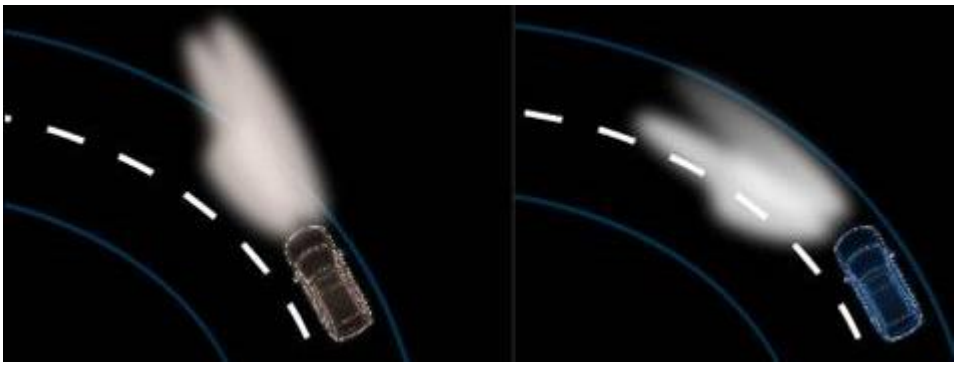

3. 속력을 110km로 올린다.

```
Status.Read: [ECU] 상하모터가 15 각도 만큼 변경되었습니다.
Status.Lines: 7
```




4. 좌회전을 한다.

```
Status.Read: [ECU] 좌우모터가 40 각도 만큼 변경되었습니다.
Status.Lines: 8
```



5. 날이 어두워져 조도센서의 값이 올라간다.

Status.Read: [ECU] 로우빔의 세기는 1 입니다.
Status.Lines: 9
Status.Read: [ECU] 로우빔의 세기는 3 입니다.
Status.Lines: 10

6. 직진 상태로 변경하고 상향등을 킨다.

Status.Read: [ECU] 좌우모터가 0 각도 만큼 변경되었습니다.
Status.Lines: 4
Status.Read: [ECU] 로우빔이 꺼졌습니다.
Status.Lines: 5
Status.Read: [ECU] 하이빔이 켜졌습니다.
Status.Lines: 6

7. 맞은편에 차량이 나타난다.

Status.Read: [ECU] 하이빔이 꺼졌습니다.
Status.Lines: 17
Status.Read: [ECU] 전방에 차가 등장했습니다. 하이빔을 끄고 로우빔을 키겠습니다
Status.Lines: 18
Status.Read: [ECU] 로우빔의 세기는 3 입니다.
Status.Lines: 19

8. 맞은편 차량이 지나간다.

Status.Read: [ECU] 하이빔이 켜졌습니다.
Status.Lines: 15
Status.Read: [ECU] 로우빔이 꺼졌습니다.
Status.Lines: 16