WhiteBox test

▼ 결과

온도센서

테스트 케이스 이름	결과
TS-REQ001	PASS
TS-REQ002	PASS
TS-REQ003	PASS
TS-REQ004	PASS
TS-REQ005	PASS
TS-REQ006	PASS
TS-REQ007	PASS
TS-REQ008	PASS
TS-REQ009	PASS

테스트 케이스 이름	결과
TS-SM001	PASS
TS-SM011	PASS
TS-SM012	PASS
TS-SM021	PASS
TS-SM022	PASS
TS-SM023	PASS
TS-SM031	PASS
TS-SM032	PASS
TS-SM041	PASS
TS-SM042	PASS
TS-SM043	PASS
TS-SM051	PASS
TS-SM052	PASS
TS-SM053	PASS

1-wire

테스트 케이스 이름	결과
OW-SM001	PASS
OW-SM011	PASS
OW-SM012	PASS
OW-SM013	PASS
OW-SM014	PASS
OW-SM021	PASS

OW-SM022	**PASS**
OW-SM023	**PASS**
OW-SM031	**PASS**
OW-SM032	**PASS**
OW-SM033	**PASS**
OW-SM034	**PASS**

▼ 온도센서 계층

이 계층은 물리적인 통신은 전혀 관여하지 않고, 온도센서가 제공하는 통신기능을 단위명령으로 쪼개어 하위계층으로 전달한다.

하지만, 제공하는 기능 중에선 단지 물리적인 계층에서만 바라보면 안되는 기능들과 에러처리도 있기 때문에, 그 부분에 대한 흐름 컨트롤을 수행한다.

기능의 흐름을 중점으로 각 분기조건을 테스트 할 것이다.

전체 테스트를 2개의 객체로 본다.

- `tempSensor_reqCommand()` → 100ms 주기로 호출되는 것이 아님 (언제든 호출가능)
- `tempSensor_stMachine()` → 100ms 주기로 호출됨.

하지만 테스트 시나리오는 이 두 개를 따로따로 구분해서 만들어야 한다.

리퀘스트

유저는 필요한 기능이 있다면 이벤트상황이든 주기적으로든 명령을 내릴 수 있어야 한다.

함수에서 고려하는 요소

- 온도센서 객체
  - 현재상태
  - 이전상태
  - 에러코드
  - 최근 수신 데이터의 유효성 → 반복 수신요청 전송
- 전달되는 파라미터
  - 요청작업

- 작업에 필요한 파라미터 주소

조건 분기 확인

▼ **TS-REQ001**

- 목표 : 센서상태가 IDLE/PENDING/EXEC 이외 모두 reject
- 조건
  - currState = START/BUSY/ERROR
  - cmd = REQ_DATA
  - param = NULL
- 결과 : return false

▼ **TS-REQ002**

- 목표 : 정의된 명령어 번호가 아닐경우 reject
- 조건
  - currState = IDLE/PENDING/EXEC
  - cmd = -5
  - param = NULL
- 결과 : return false

▼ **TS-REQ003**

- 목표 : 요청작업이 SET_RESOL인데 param변수가 null 임 → reject
- 조건
  - currState = IDLE/PENDING/EXEC
  - cmd = SET_RESOL
  - param = NULL
- 결과 : return false

▼ **TS-REQ004**

- 목표 : 요청작업이 SET_RESOL인데 param변수가 정의된 값을 초과함 → reject
- 조건

    - currState = IDLE/PENDING/EXEC
    - cmd = SET_RESOL
    - param = 14
  - 결과 : return false

▼ **TS-REQ005**

- 목표 : pending 상태에서 막 빠져나왔고 CRC 매치를 확인함
- 조건
  - currState = IDLE
  - prevState = PENDING
  - lastData_IsValid = true
  - cmd = ANYTHING
  - param = NULL(SET_RESOL 제외)
- 결과
  - currCmd 설정 → 다음 실행할 명령이라는 뜻
  - *reqFailed = false
  - return true

▼ **TS-REQ006**

- 목표 : pending 상태에서 막 빠져나왔고 CRC 미스매치를 확인함
- 조건
  - currState = IDLE
  - prevState = PENDING
  - lastData_IsValid = false
  - cmd = ANYTHING
  - param = NULL(SET_RESOL 제외)
- 결과
  - return false → 명령무시 (이전 REQ_DATA 요청 다시 수행)

▼ **TS-REQ007**

- 목표 : 현재 상태가 idle 상태이고, 이전 상태가 pending 외 아무 상태일 때, 요청 수신 시 무조건 처리
- 조건
  - currState = IDLE
  - prevState != PENDING
  - cmd = ANYTHING
  - param = NULL(SET_RESOL 제외)
- 결과
  - currCmd 설정 → 다음 실행할 명령이라는 뜻
  - *reqFailed = false
  - return true

▼ **TS-REQ008**

- 목표 : 현재 상태가 idle 상태가 아닌 EXEC/PENDING 상태이지만, STOP 요청이 들어옴 → 이 요청은 바로 실행되어야 함
- 조건
  - currState = EXEC/PENDING
  - cmd = STOP
  - param = NULL
- 결과
  - currCmd 설정 → 다음 실행할 명령이라는 뜻
  - *reqFailed = false
  - return true

▼ **TS-REQ009**

- 목표 : 현재 상태가 idle 상태도 아니고 STOP 명령도 아님
- 조건
  - currState = EXEC/PENDING
  - cmd = SAVE_RESOL
  - param = NULL

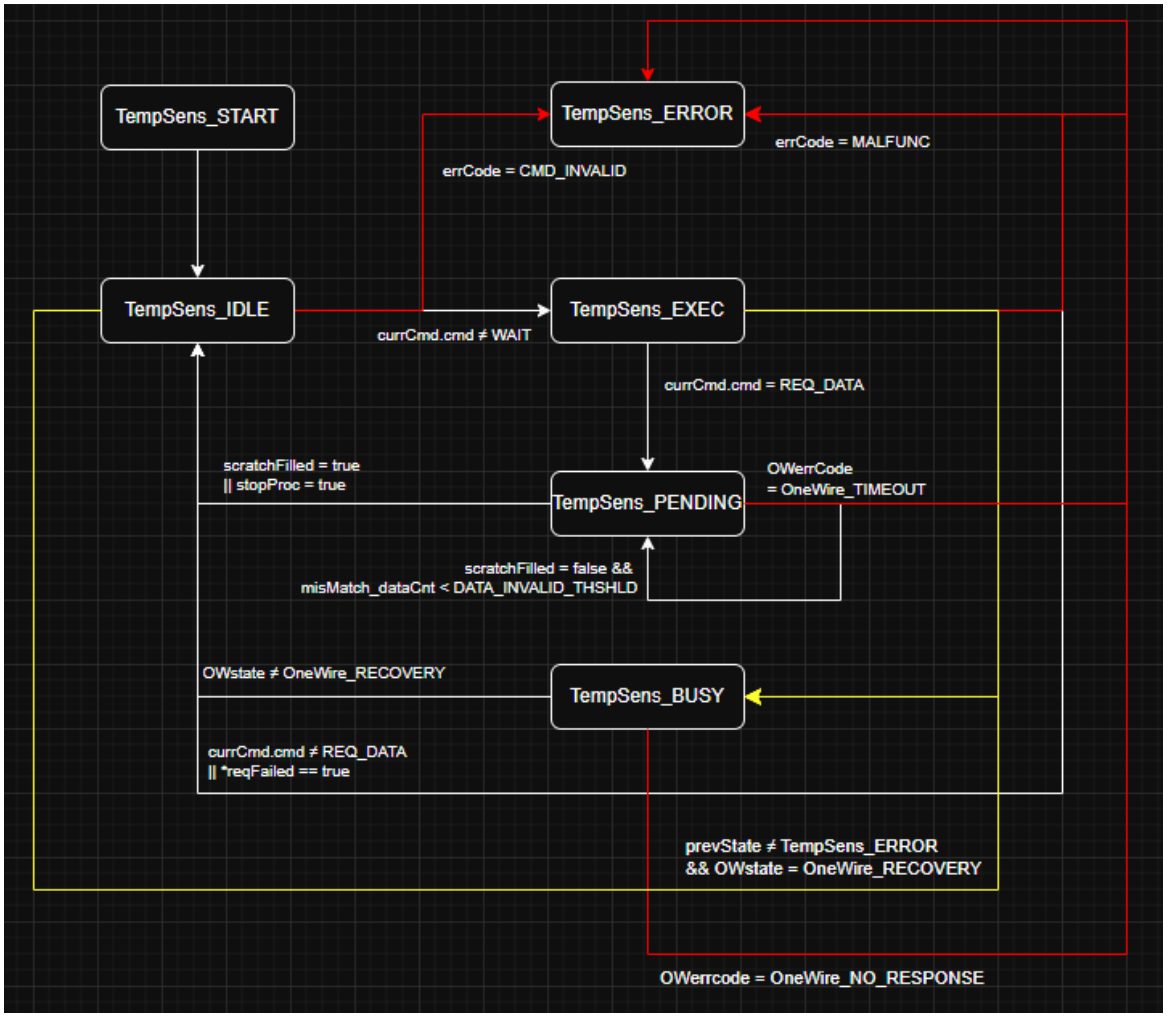
- 결과
    - return false

상태머신

온도센서의 상태머신은 초기화가 이루어 진 후, 100ms의 주기로 동작하는 태스크이다.

상태 커버리지 보다는 전이 커버리지를 이용하여 상태의 전이가 올바르게 이루어지는 지를 중점으로 확인한다.

온도센서 계층의 상태도



태스크에서 고려하는 요소

상태 머신에서는 3가지의 요소를 이용한다

- 온도센서 객체의 내부변수
    - 현재상태

    - 이전상태
    - 마지막 수신 데이터의 유효 플래그

      ⇒ 수신 받은 CRC값과 직접 계산한 CRC 값이 맞다면 true
    - 마지막 수신 데이터의 CRC 미스매치 카운트

      ⇒ 수신 받은 CRC값과의 미스매치가 발생하면 이전 명령을 재수행하는 횟수 → 횟수가 기준치를 넘어가면 에러상태로 빠짐
- 하위계층의 정보
    - 하위 계층의 상태

      ⇒ 하위 계층이 명령을 받을 준비가 되면 사용자가 요청한 작업을 하위계층에서 실행 할 수 있는 단위로 분해하여 enqueue 시킴
    - 하위 계층의 에러

      ⇒ 하위 계층이 어떤 에러정보를 갖고 있는지에 따라 센서의 상태를 결정함
- 사용자가 요청하는 작업정보
    - 위의 `tempSensor_reqCommand` 함수를 제시하여 사용자에게서 작업/부가정보 등을 입력받음

상태 전이 기반 테스트

사전준비

모든 테스트 케이스는 하위계층의 상태머신이 먼저 정상적으로 호출되고, 초기화 작업이 정상적으로 이루어진 후, 상태머신이 호출되었다는 상황을 전제로 한다.

또한, 각 계층의 초기화 함수가 초기화 이후 다시는 호출 되지 않는 상황도 전제로 한다.

START_STATE

▼ TS-SM001

목적 : START 상태가 정상적인 초기상태일 때 IDLE로 넘어가야 한다.

준비

- vTempSensor_obj.prevState = TEMPSENS_STATE_START
- vTempSensor_obj.errCode = TEMPSENS_ERR_OKAY
- vTempSensor_obj.OWcurrState ≠ OW_STATE_RECOVERY || OW_STATE_ERROR
- vTempSensor_obj.OWerrCode = OW_ERR_OKAY

결과

- process : START → IDLE
- START exit : X
- IDLE entry :
  - vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_WAIT;
  - vTempSensor_obj.currCmd.param = NULL;
  - vTempSensor_obj.currCmd.reqFailed = NULL;
- IDLE do : X

IDLE_STATE

▼ TS-SM011

목적 : 아무 명령도 들어오지 않은 상태일 땐, 자신의 상태를 유지한다.

준비

- vTempSensor_obj.prevState = TEMPSENS_STATE_IDLE
- vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_WAIT
- vTempSensor_obj.OWcurrState ≠ OW_STATE_RECOVERY || OW_STATE_ERROR
- vTempSensor_obj.OWerrCode = OW_ERR_OKAY

결과

- process : IDLE → IDLE
- IDLE do : X

▼ TS-SM012

목적 : 명령이 들어온 상태일땐, EXEC 상태로 천이한다.

준비

- vTempSensor_obj.prevState = TEMPSENS_STATE_IDLE
- vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_REQ_DATA
- vTempSensor_obj.OWcurrState ≠ OW_STATE_RECOVERY || OW_STATE_ERROR
- vTempSensor_obj.OWerrCode = OW_ERR_OKAY

결과

- process : IDLE → EXEC
- IDLE exit : X
- EXEC entry : X
- EXEC do : tempSensor_execCMD 함수 호출

  ⇒ 해당 함수는 단순히, 요청번호에 대응된 처리함수를 실행

EXEC_STATE

▼ TS-SM021

목적 : REQ_DATA 요청이 아닌 다른 명령일 경우 다시 IDLE 상태로 천이한다.

준비 :

- vTempSensor_obj.prevState = TEMPSENS_STATE_EXEC
- vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_SAVE_RESOL
- vTempSensor_obj.OWcurrState ≠ OW_STATE_RECOVERY || OW_STATE_ERROR
- vTempSensor_obj.OWerrCode = OW_ERR_OKAY

결과 :

- process : EXEC → IDLE
- EXEC exit : X
- IDLE entry ：
  - vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_WAIT;
  - vTempSensor_obj.currCmd.param = NULL;
  - vTempSensor_obj.currCmd.reqFailed = NULL;
- IDLE do : X

▼ TS-SM022

목적 : REQ_DATA 요청일 경우 polling 유사한 작업을 위해 PENDING 상태로 천이한다.

준비 :

- vTempSensor_obj.prevState = TEMPSENS_STATE_EXEC

- vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_REQ_DATA
- vTempSensor_obj.OWcurrState ≠ OW_STATE_RECOVERY || OW_STATE_ERROR
- vTempSensor_obj.OWerrCode = OW_ERR_OKAY

결과 :

- process : EXEC → PENDING
- EXEC exit : X
- PENDING entry :
  - 스크래치 패드 raw 배열 0xff로 초기화
  - vTempSensor_obj.scratchpadFilled = false
- PENDING do :
  - 사용자가 중단요청을 내릴경우, 중단요청 처리함수 실행 후, 중단요청 플래그 CLR
  - 아닐경우, 스크래치 패드 raw 배열들이 하나라도 초기화 값을 가질 시, scratchpadFilled = false
  - 모두 채워졌을 경우, scratchpadFilled = true

▼ TS-SM023

목적 : 만약, 하위계층으로 단위 명령들을 enqueue하는 과정이 실패하면 무조건 idle 상태로 천이한다.

준비 :

- vTempSensor_obj.prevState = TEMPSENS_STATE_EXEC
- vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_REQ_DATA
- *vTempSensor_obj.currCmd.reqFailed = true
- vTempSensor_obj.OWcurrState ≠ OW_STATE_RECOVERY || OW_STATE_ERROR
- vTempSensor_obj.OWerrCode = OW_ERR_OKAY

결과 :

- process : EXEC → IDLE
- EXEC exit : X

- IDLE entry ：
  - vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_WAIT;
  - vTempSensor_obj.currCmd.param = NULL;
  - vTempSensor_obj.currCmd.reqFailed = NULL;
- IDLE do : X

PENDING_STATE

▼ TS-SM031

목적 : 아직 데이터의 일부가 초기화 값이어서 상태를 유지한다

준비 :

- vTempSensor_obj.prevState = TEMPSENS_STATE_PENDING
- vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_REQ_DATA
- scratchpadFilled = false
- vTempSensor_obj.OWcurrState ≠ OW_STATE_RECOVERY || OW_STATE_ERROR
- vTempSensor_obj.OWerrCode = OW_ERR_OKAY

결과 :

- process : PENDING → PENDING
- PENDING do :
  - 사용자가 중단요청을 내릴경우, 중단요청 처리함수 실행 후, 중단요청 플래그 CLR
  - 아닐경우, 스크래치 패드 raw 배열들이 하나라도 초기화 값을 가질 시, scratchpadFilled = false
  - 모두 채워졌을 경우, scratchpadFilled = true

▼ TS-SM032

목적 : 모든 데이터가 초기화 값이 아님 → 데이터가 수신되었다고 판단 (CRC 결과도 만족)

준비 :

- vTempSensor_obj.prevState = TEMPSENS_STATE_PENDING
- vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_REQ_DATA

- scratchpadFilled = true
- vTempSensor_obj.lastData_IsValid_flag = true
- vTempSensor_obj.OWcurrState ≠ OW_STATE_RECOVERY || OW_STATE_ERROR
- vTempSensor_obj.OWerrCode = OW_ERR_OKAY

결과 :

- process : PENDING → IDLE
- PENDING exit : postProc 함수 호출

  ⇒ 해당 함수에서는 CRC 계산 및 오류가 아닌 값을 수신했다고 판단할 경우, 객체에 현재 온도 값 및 센서의 분해능을 업데이트 → 사용자가 받아서 쓸 수 있음
- IDLE entry :
  - CRC 정합을 확인하였음
  - vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_WAIT;
  - vTempSensor_obj.currCmd.param = NULL;
  - vTempSensor_obj.currCmd.reqFailed = NULL;
- IDLE do : X

CRC MISMATCH FLOW

▼ TS-SM041

목적 : 데이터를 수신하였지만, CRC 값이 맞지 않음

준비 :

- vTempSensor_obj.prevState = TEMPSENS_STATE_PENDING
- vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_REQ_DATA
- scratchpadFilled = true
- vTempSensor_obj.lastData_IsValid_flag = false
- vTempSensor_obj.mismatch_dataCnt = 3
- vTempSensor_obj.OWcurrState ≠ OW_STATE_RECOVERY || OW_STATE_ERROR
- vTempSensor_obj.OWerrCode = OW_ERR_OKAY

결과 :

- process : PENDING → IDLE
- PENDING exit : postProc 함수 호출

  ⇒ 해당 함수에서 CRC 계산을 수행했지만, 계산 CRC값과 넘겨받은 CRC 값이 서로 맞지 않음 → 재수신 필요 mismatch_dataCnt++ (=4)
- IDLE entry :
  - CRC 부정합을 확인하였음
  - currCmd 멤버는 초기화 하지 않고, 이전 명령 정보를 재수행
- IDLE do : X

▼ TS-SM042

목적 : 이전에 사용자가 요청한 읽기 명령을 수행했지만, CRC의 연속적인 부정합으로 인해 센서에 문제가 발생했다고 판단하고 에러상태로 빠짐

준비 :

- vTempSensor_obj.prevState = TEMPSENS_STATE_PENDING
- vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_REQ_DATA
- scratchpadFilled = true
- vTempSensor_obj.lastData_IsValid_flag = false
- vTempSensor_obj.mismatch_dataCnt = 4
- vTempSensor_obj.OWcurrState ≠ OW_STATE_RECOVERY || OW_STATE_ERROR
- vTempSensor_obj.OWerrCode = OW_ERR_OKAY

결과 :

- process : PENDING → IDLE
- PENDING exit : postProc 함수 호출

  ⇒ 해당 함수에서 CRC 계산을 수행했지만, 계산 CRC값과 넘겨받은 CRC 값이 서로 맞지 않음 → 자동 재수신 과정 필요 mismatch_dataCnt++ (=5)

  ⇒ mismatch_dataCnt가 한계에 다다름 (에러천이 필요)

  ⇒ vTempSensor_obj.errCode = TEMPSENS_ERR_MALFUNC

- IDLE entry :
  - CRC 부정합을 확인하였음
  - currCmd 멤버는 초기화 하지 않고, 이전 명령 정보를 재수행
- IDLE do : X

next tick

- process : IDLE → ERROR
- IDLE exit : X
- ERROR entry : X
- ERROR do : X

▼ TS-SM043

목적 : 재요청 루프 안에서 사용자가 중단 요청을 보낸 경우엔 모든 과정을 중단하고 IDLE로 천이해야 한다.

조건 :

- vTempSensor_obj.prevState = TEMPSENS_STATE_PENDING
- vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_REQ_DATA
- scratchpadFilled = true
- vTempSensor_obj.lastData_IsValid_flag = false
- **stopProc = true**
- vTempSensor_obj.OWcurrState ≠ OW_STATE_RECOVERY || OW_STATE_ERROR
- vTempSensor_obj.OWerrCode = OW_ERR_OKAY

결과 :

- process : PENDING → IDLE
- PENDING exit :
  - mismatch_dataCnt = 0
  - 하위계층 flush 명령 수행
- IDLE entry :
  - vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_WAIT;

    - vTempSensor_obj.currCmd.param = NULL;
    - vTempSensor_obj.currCmd.reqFailed = NULL;

ONEWIRE

▼ TS-SM051

목적 : 만약 하위계층이 일시적인 무응답으로 복구 상태에 있을 때는 BUSY 상태로 천이되어야 한다.

준비 :

- vTempSensor_obj.OWcurrState = OW_STATE_RECOVERY
- vTempSensor_obj.OWerrCode = OW_ERR_NOT_RESPONDING

결과 (IDLE)

- process : IDLE → BUSY
- IDLE exit : X
- BUSY entry :
- BUSY do : X

결과 (PENDING)

- process : PENDING → BUSY
- PENDING exit : mismatch_dataCnt = 0
- BUSY entry :
- BUSY do : X

결과 (BUSY)

- process : BUSY → BUSY
- BUSY do : X

▼ TS-SM052

목적 : 하위 계층의 무응답 복구 대기 프로세스 중, 고장으로 판정되었음. 에러상태로 천이될 것

준비 :

- vTempSensor_obj.OWcurrState = OW_STATE_ERROR
- vTempSensor_obj.OWerrCode = OW_ERR_NO_RESPONSE

결과 :

- process : BUSY → ERROR
- BUSY exit : X
- ERROR entry : X
- ERROR do : X

▼ TS-SM053

목적 : PENDING 상태는 하위계층이 RDY_CHK 상태일때가 대부분임. 하지만, RDY_CHK가 정상적으로 이루어지지 않아 TIME_OUT 상황으로 판단되면, 센서계층도 에러로 천이될 것.

준비 :

- vTempSensor_obj.prevState = TEMPSENS_STATE_PENDING
- vTempSensor_obj.currCmd.cmd = TEMPSENS_CMD_REQ_DATA
- scratchpadFilled = false
- vTempSensor_obj.lastData_IsValid_flag = false
- vTempSensor_obj.OWcurrState = OW_STATE_ERROR
- vTempSensor_obj.OWerrCode = OW_ERR_TIMEOUT

결과 :

- process : PENDING → ERROR
- PENDING exit :
  - scratchpadFilled = false
  - mismatch_dataCnt = 0
- ERROR entry : X
- ERROR do : X

▼ OneWire 계층

기능의 흐름을 중점으로 각 분기조건을 테스트 할 것이다.

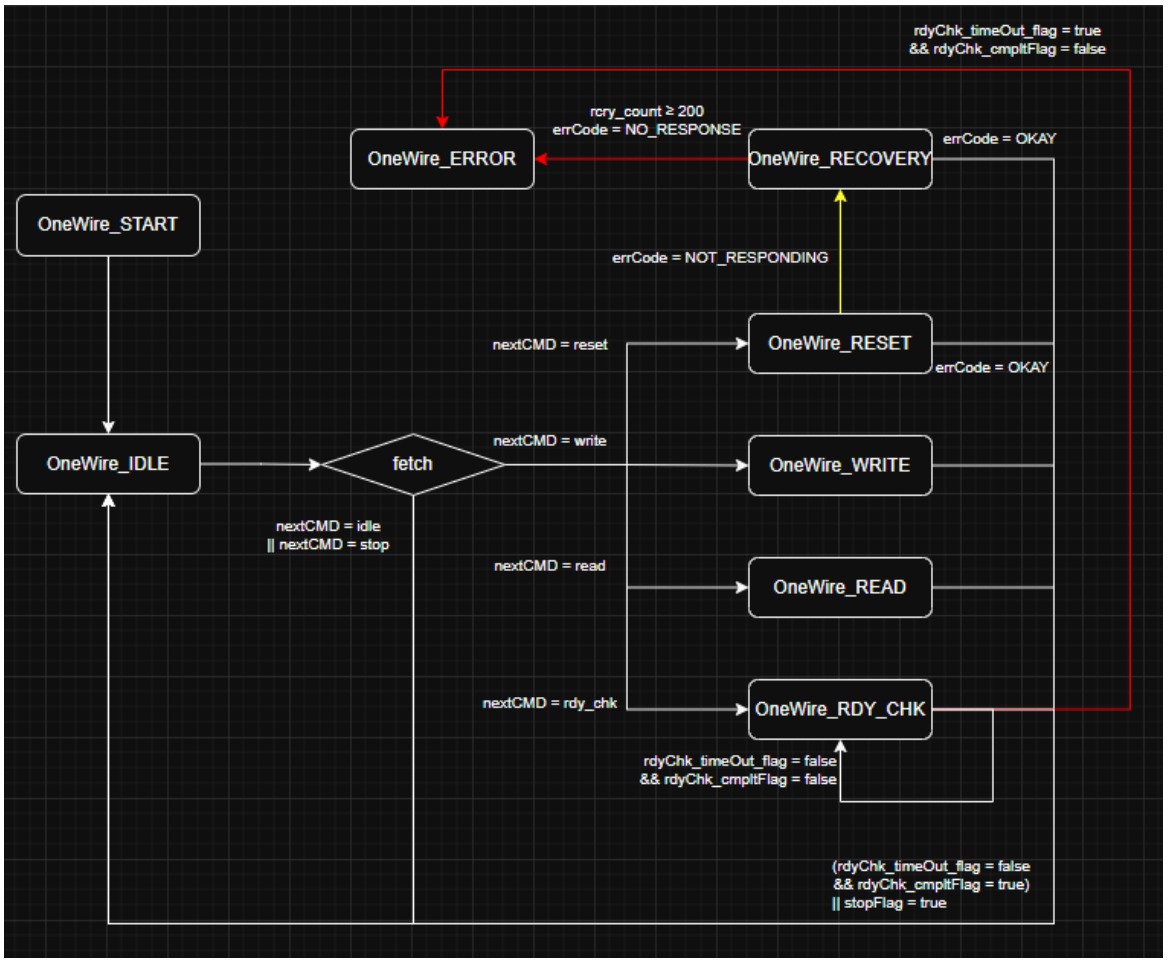
상태머신

원와이어 계층은 해당 온도센서와의 입출력을 물리적인 상황에서 묶을 수 있는 단위적인 명령들만 수행한다.

만약 물리적인 상황에서 에러가 발생한다면 그 에러도 직접 처리해서 이 계층과 상호작용하는 상위계층에 알린다.

이 계층은 5ms 주기로 호출된다.

oneWire 계층 상태도



상태머신에서 고려하는 요소

- oneWire 객체의 내부 변수
  - 현재 상태
  - 이전 상태
  - 리커버리 시도 횟수
  - 센서 준비 tick 횟수
  - 센서 준비상태 thrsld

- 상위계층에서 받은 명령들
  - 명령 큐
  - 명령 종류
  - 명령을 수행하는데 필요한 파라미터

상태 전이 기반 testCase

사전준비

모든 테스트 케이스는 초기화 작업이 정상적으로 이루어진 후, 상태머신이 호출되었다는 상황을 전제로 한다.

또한, 각 계층의 초기화 함수가 초기화 이후 다시는 호출 되지 않는 상황도 전제로 한다.

이로써, 초기화 관련 문제는 철저히 배제한다.

마지막으로, 상위계층에 에러가 발생하지 않고 정상적으로 동작하는 상황을 전제로 한다.

START_STATE

▼ OW-SM001

목적 : START 상태에서 IDLE 상태로 전이된다.

준비 :

- vOneWire_obj.prevState = OW_STATE_START
- vOneWire_obj.errCode = OW_ERR_OKAY

결과 :

- process : START → IDLE
- START exit : X
- IDLE entry : X
- IDLE do : X

IDLE_STATE

▼ OW-SM011

목적 : `oneWire_cmdQueue_fetchCMD` 를 수행하여, dequeue 한 명령이 reset이어서 RESET 상태로 천이됨

준비 :

- vOneWire_obj.prevState = OW_STATE_IDLE

- vOneWire_obj.cmdNext.cmd = OW_CMD_RESET
- vOneWire_obj.errCode = OW_ERR_OKAY

결과 :

- process : IDLE → RESET
- IDLE exit : X
- RESET entry : X
- RESET do : `oneWire_reset` 함수 실행
  - 데이터 시트에서 요구한 사항 그대로 실행

▼ OW-SM012

목적 : `oneWire_cmdQueue_fetchCMD` 를 수행하여, dequeue 한 명령이 write이어서 WRITE 상태로 천이됨

준비 :

- vOneWire_obj.prevState = OW_STATE_IDLE
- vOneWire_obj.cmdNext.cmd = OW_CMD_WRITE
- vOneWire_obj.cmdNext.param ≠ NULL
- vOneWire_obj.errCode = OW_ERR_OKAY

결과 :

- process : IDLE → WRITE
- IDLE exit : X
- WRITE entry : X
- WRITE do : `oneWire_writeByte` 함수 실행

▼ OW-SM013

목적 : `oneWire_cmdQueue_fetchCMD` 를 수행하여, dequeue 한 명령이 readycheck여서 RDY_CHK 상태로 천이됨

준비 :

- vOneWire_obj.prevState = OW_STATE_IDLE
- vOneWire_obj.cmdNext.cmd = OW_CMD_RDY_CHK
- vOneWire_obj.cmdNext.param = 96

- vOneWire_obj.errCode = OW_ERR_OKAY

결과 :

- process : IDLE → RDY_CHK
- IDLE exit : X
- RDY_CHK entry :
  - rdyChk_Cnt_5ms 멤버 초기화
  - 파라미터로 받은 변수를 5ms의 틱으로 계산하기 (나머지는 전부 올림 계산)
    - rdyChk_TO_5ms ⇒ 96/5 + (96%5 ≠ 0) = 19 + 1 = 20
  - rdyChk_cmpltFlag, rdyChk_timeOut_flag 초기화 (false)
- READ do : `oneWire_readByte` 함수 실행

▼ OW-SM003

목적 : `oneWire_cmdQueue_fetchCMD` 를 수행하여, dequeue 한 명령이 read여서 READ 상태로 천이됨

준비 :

- vOneWire_obj.prevState = OW_STATE_IDLE
- vOneWire_obj.cmdNext.cmd = OW_CMD_READ
- vOneWire_obj.cmdNext.param ≠ NULL
- vOneWire_obj.errCode = OW_ERR_OKAY

결과 :

- process : IDLE → READ
- IDLE exit : X
- READ entry : X
- READ do : `oneWire_readByte` 함수 실행

▼ OW-SM014

목적 : `oneWire_cmdQueue_fetchCMD` 를 수행하여, dequeue 한 명령이 stop이어서, 명령큐를 비우고 상태유지

준비 :

- vOneWire_obj.prevState = OW_STATE_IDLE

- vOneWire_obj.cmdNext.cmd = OW_CMD_STOP
- vOneWire_obj.errCode = OW_ERR_OKAY

결과 :

- process : IDLE → IDLE
  - `oneWire_cmdQueue_flush` 실행
- IDLE do : X

RECOVERY_STATE

▼ OW-SM021

목적 : `oneWire_reset` 함수 실행 중, 센서 무응답 상황이 발생함. 에러코드를 띄울 것

준비 :

- vOneWire_obj.prevState = OW_STATE_IDLE
- vOneWire_obj.cmdNext.cmd = OW_CMD_RESET
- vOneWire_obj.errCode = OW_ERR_OKAY

결과 :

- process : IDLE → RESET
- IDLE exit : X
- RESET entry : X
- RESET do : `oneWire_reset` 함수 실행
  - presence 신호 대기시간동안 high 상태 유지
  - 에러코드 OW_ERR_NOT_RESPONDING 발생 (사실상 warning)

nextTick

- process : RESET → RECOVERY
- RESET exit : X
  - vOneWire_obj.cmdNext.cmd = OW_CMD_IDLE
  - vOneWire_obj.cmdNext.param = NULL
- RECOVERY entry :
  - rcvrFailCount 초기화
- RECOVERY do :

    - `oneWire_cmdQueue_flush` 함수를 실행
    - `oneWire_recoveryBus` 함수를 실행

▼ OW-SM022

목적 : 임시 무응답 상태에서 반복적으로 reset을 수행하다가 presence 신호를 받음 → IDLE 상태로 천이

준비 :

- vOneWire_obj.prevState = OW_STATE_RECOVERY
- vOneWire_obj.cmdNext.cmd = OW_CMD_WAIT
- vOneWire_obj.errCode = OW_ERR_NOT_RESPONDING
- vOneWire_obj.rcvrFailCount ≠ 0

결과 :

- process : RECOVERY → RECOVERY
- RECOVERY do :
    - `oneWire_cmdQueue_flush` 함수를 실행
    - `oneWire_recoveryBus` 함수를 실행
        - `oneWire_reset` 함수를 호출하여 errCode가 OKAY 상태로 전환된 것을 확인

nextTick

- process : RECOVERY → IDLE
- RECOVERY exit : X
- IDLE entry : X
- IDLE do : X

▼ OW-SM023

목적 : 무응답 복구 상태에서 반복적으로 reset을 수행하다가 카운트 문턱값을 넘도록 presence 신호를 못 받음 → ERROR 상태로 천이

준비 :

- vOneWire_obj.prevState = OW_STATE_RECOVERY
- vOneWire_obj.cmdNext.cmd = OW_CMD_WAIT
- vOneWire_obj.errCode = OW_ERR_NOT_RESPONDING

- vOneWire_obj.rcvrFailCount = 199

결과 :

- process : RECOVERY → RECOVERY
- RECOVERY do :
  - `oneWire_cmdQueue_flush` 함수를 실행
  - `oneWire_recoveryBus` 함수를 실행
    - `oneWire_reset` 함수를 호출하여 rcvrFailCount++ 후, 문턱 값에 도달해서 errCode = OW_ERR_NO_RESPONSE 로 결정

nextTick

- process : RECOVERY → ERROR
- RECOVERY exit : X
- ERROR entry : X
- ERROR do : X

RDY_CHK_STATE

▼ OW-SM031

목적 : 특정 커맨드(온도변환)를 명령했고 매 tick 마다 대기하는 과정 (not bit read). RDY_CHK 상태유지

준비 :

- vOneWire_obj.prevState = OW_STATE_RDY_CHK
- vOneWire_obj.cmdNext.cmd = OW_CMD_RDY_CHK
- vOneWire_obj.cmdNext.param = 96
- vOneWire_obj.errCode = OW_ERR_OKAY
- vOneWire_obj.rdyChk_TO_5ms = 20
- vOneWire_obj.rdyChk_Cnt_5ms = 19
- vOneWire_obj.rdyChk_timeOut_flag = false
- vOneWire_obj.rdyChk_cmpltFlag = false

결과 :

- process : RDY_CHK→ RDY_CHK

- RECOVERY do :
  - `oneWire_readyCheck` 함수를 실행
  - rdyChk_Cnt_5ms(19) < rdyChk_TO_5ms (20)+10ms여분(2) 이다.
  - rdyChk_Cnt_5ms(19) < rdyChk_TO_5ms (20) 이기에 준비완료 bit는 올 수 없음
  - rdyChk_Cnt_5ms++

▼ OW-SM032

목적 : 특정 커맨드(온도변환)를 명령했고 매 tick 마다 대기하는 과정 (bit read) . RDY_CHK 상태유지

준비 :

- vOneWire_obj.prevState = OW_STATE_RDY_CHK
- vOneWire_obj.cmdNext.cmd = OW_CMD_RDY_CHK
- vOneWire_obj.cmdNext.param = 96
- vOneWire_obj.errCode = OW_ERR_OKAY
- vOneWire_obj.rdyChk_TO_5ms = 20
- vOneWire_obj.rdyChk_Cnt_5ms = 20
- vOneWire_obj.rdyChk_timeOut_flag = false
- vOneWire_obj.rdyChk_cmpltFlag = false

결과 :

- process : RDY_CHK→ RDY_CHK
- RECOVERY do :
  - `oneWire_readyCheck` 함수를 실행
  - rdyChk_Cnt_5ms(20) < rdyChk_TO_5ms (20)+10ms여분(2) 이다.
  - rdyChk_Cnt_5ms(20) = rdyChk_TO_5ms (20) 이기에 준비완료 bit가 올 시간이 됨
  - `oneWire_readBit` 함수를 호출 → 아직 준비완료 bit를 못받음

- rdyChk_Cnt_5ms++

▼ OW-SM033

목적 : 특정 커맨드(온도변환)를 명령했고 드디어 준비완료 bit를 수신함. IDLE 상태로 천이될 것

준비 :

- vOneWire_obj.prevState = OW_STATE_RDY_CHK
- vOneWire_obj.cmdNext.cmd = OW_CMD_RDY_CHK
- vOneWire_obj.cmdNext.param = 96
- vOneWire_obj.errCode = OW_ERR_OKAY
- vOneWire_obj.rdyChk_TO_5ms = 20
- vOneWire_obj.rdyChk_Cnt_5ms = 21
- vOneWire_obj.rdyChk_timeOut_flag = false
- vOneWire_obj.rdyChk_cmpltFlag = false

결과 :

- process : RDY_CHK→ RDY_CHK
- RECOVERY do :
  - `oneWire_readyCheck` 함수를 실행
  - rdyChk_Cnt_5ms(21) < rdyChk_TO_5ms (20)+10ms여분(2) 이다.
  - rdyChk_Cnt_5ms(21) ≥ rdyChk_TO_5ms (20) 이기에 준비완료 bit가 올 시간이 됨
  - `oneWire_readBit` 함수를 호출 → 준비완료 bit를 수신함
    - rdyChk_cmpltFlag = true

nextTick

- process : RDY_CHK→ IDLE
  - vOneWire_obj.rdyChk_cmpltFlag == true
- RDY_CHK exit :

    - vOneWire_obj.cmdNext.cmd = OW_CMD_IDLE
    - vOneWire_obj.cmdNext.param = NULL
  - IDLE entry : X
  - IDLE do : X

▼ OW-SM034

목적 : 특정 커맨드(온도변환)를 명령했지만, 제한 시간내에 준비완료 bit를 받지 못함 ERROR 상태로 천이될 것

준비 :

- vOneWire_obj.prevState = OW_CMD_RDY_CHK
- vOneWire_obj.cmdNext.cmd = OW_CMD_RDY_CHK
- vOneWire_obj.cmdNext.param = 96
- vOneWire_obj.errCode = OW_ERR_OKAY
- vOneWire_obj.rdyChk_TO_5ms = 20
- vOneWire_obj.rdyChk_Cnt_5ms = 22
- vOneWire_obj.rdyChk_timeOut_flag = false
- vOneWire_obj.rdyChk_cmpltFlag = false

결과 :

- process : RDY_CHK→ RDY_CHK
- RECOVERY do :
  - `oneWire_readyCheck` 함수를 실행
  - rdyChk_Cnt_5ms(22) == rdyChk_TO_5ms (20)+10ms여분(2) 이다. (timeout 처리)
  - rdyChk_timeOut_flag = true
  - errCode = OW_ERR_TIMEOUT

nextTick

- process : RDY_CHK→ ERROR
  - vOneWire_obj.rdyChk_cmpltFlag == false
  - vOneWire_obj.rdyChk_timeOut_flag == true

- RDY_CHK exit :
  - vOneWire_obj.cmdNext.cmd = OW_CMD_IDLE
  - vOneWire_obj.cmdNext.param = NULL
- ERROR entry : X
- ERROR do : X