8. 고급 제어 로직 설계

8.1 상태 전이(State Machine) 기반 설계

(PLC에서 순차적 공정, 동작 흐름, 자동화 로직 구현을 위한 핵심 제어 패턴)

✅ 개요

상태 전이(State Machine) 또는 **상태 기반 제어**는
공정의 각 단계를 상태(State)로 정의하고, 조건(Transition)에 따라 상태를 이동시키며 전체 동작을 제어하는 방식이다.

이는 특히 **순차 제어**, **로봇 동작 제어**, **자동화 설비 단계별 운전**, **인터페이스 제어** 등에 널리 활용된다.

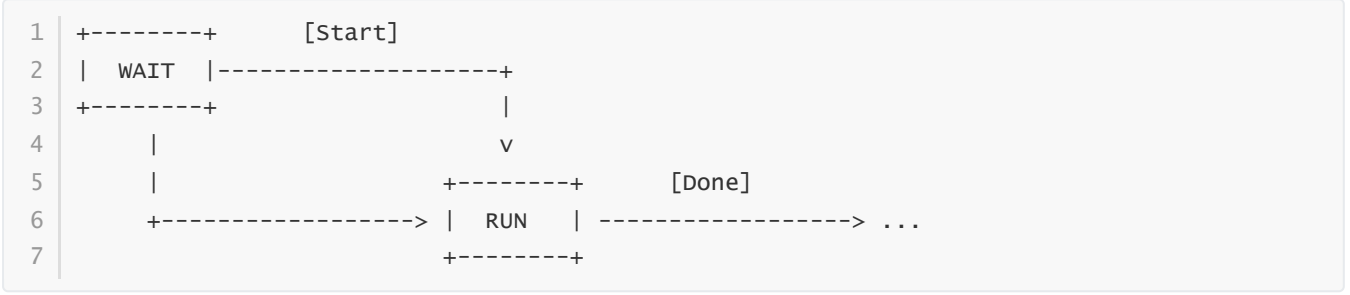
✅ 1. 핵심 개념

요소	설명
State (상태)	현재 시스템의 동작 단계 (예: 대기, 작동, 정지 등)
Event / Condition	상태 전환 조건 (예: 버튼 입력, 센서 신호)
Transition (전이)	조건이 충족되면 상태가 바뀌는 것
Action (동작)	각 상태에서 수행되는 제어 동작

📘 예시

상태	전이 조건	다음 상태	상태 동작
`WAIT`	시작 버튼	`RUN`	대기등 점등
`RUN`	완료 센서 ON	`FINISH`	모터 작동
`FINISH`	리셋 버튼	`WAIT`	알람 점등

✅ 2. 일반적인 상태 전이 구조



✅ 3. PLC 구현 방법 (Structured Text 예)

```
CASE State OF
    0:  // WAIT
        IF StartButton THEN
            State := 1;
        END_IF;

    1:  // RUN
        Motor := TRUE;
        IF DoneSensor THEN
            Motor := FALSE;
            State := 2;
        END_IF;

    2:  // FINISH
        Alarm := TRUE;
        IF ResetButton THEN
            Alarm := FALSE;
            State := 0;
        END_IF;
END_CASE;
```

- `State`는 INT형 변수
- 조건 발생 시 상태 전환
- 각 상태에 맞는 출력 동작 실행

✅ 4. 상태 전이 예: 컨베이어 제어

상태 번호	상태명	설명
`0`	INIT	시스템 초기화
`1`	IDLE	대기 상태
`2`	FEED	자재 이송
`3`	PROCESS	가공 동작
`4`	EJECT	배출 동작
`5`	ERROR	에러 정지

✅ 5. 상태 정의 테크닉

- `ENUM` 타입을 사용해 가독성을 높일 수 있음

```
TYPE StateEnum :
(
  INIT := 0,
  IDLE := 1,
  FEED := 2,
  PROCESS := 3,
  EJECT := 4,
  ERROR := 5
);
```

- `IF`보다 `CASE` 문을 적극적으로 활용

✅ 6. 상태 전이 조건 설계 팁

설계 포인트	설명
전이 조건은 명확하게	모호한 조건은 상태 충돌 유발
타이머 사용	일정 시간 경과 후 상태 전환 시 필수
상태 복귀 조건 정의	재시작, 리셋 조건 반드시 명시
예외 상태 처리	에러, 긴급 정지 상태도 별도로 정의해야 함

✅ 7. 고급 구현 기법

기법	설명
이벤트 큐 방식	입력 이벤트를 큐에 넣고 순차 처리
병렬 상태 처리	동시에 여러 상태를 추적하는 구조 (Sub-StateMachine)
SFC (Sequential Function Chart)	상태 전이 시각화 및 PLC용 표준 도식
데이터 기반 상태 머신	상태, 조건, 동작을 데이터 테이블로 분리하여 관리

✅ 8. 상태 전이 vs 순차 회로

비교 항목	상태 전이	순차 회로 (래더 타이머 기반)
설계 방식	논리적으로 상태 추적	단순 순서대로 처리
변경 용이성	**높음**	낮음 (타이머 수동 변경 필요)

비교 항목	상태 전이	순차 회로 (래더 타이머 기반)
복잡도 대응	**고성능** (조건 분기 용이)	단일 흐름에 적합
디버깅	상태 추적 쉬움	타이머 추적 복잡

✅ 9. 정리

- **상태 전이 제어는 공정 자동화의 핵심 제어 패턴**으로, 각 단계의 정확한 흐름과 조건 분리를 도와줌
- PLC에서는 `CASE` 문, `INT` 변수, `ENUM`, `타이머`, `센서 조건` 등을 통해 유연하게 구현 가능
- 실무에서는 **HMI 상태 표시**, **알람 연동**, **로그 기록**까지 함께 처리

8.2 복수 장치 동기화

(PLC에서 여러 장치 간의 타이밍 및 동작 상태를 정밀하게 맞추는 제어 기법)

✅ 개요

복수 장치 동기화(Multi-Device Synchronization)란,
하나의 공정에서 여러 기계나 설비가 **정해진 순서 및 조건에 따라 정확히 동시에 혹은 일정 시차를 두고 동작하도록 제어하는 기술**이다.

이 개념은 **이송 → 가공 → 배출**, 또는 **복수 모터, 컨베이어, 밸브** 등의 협조 동작에서 핵심적으로 적용된다.

✅ 1. 대표 예시

적용 분야	동기화 대상	설명
생산라인	피더 + 가공기 + 배출기	피더가 자재를 넣고, 가공기 작동 후 배출기 작동
병렬 제어	2축 모터 제어	두 모터가 동일 속도/위상으로 작동
로봇 협동	다관절 로봇 + 컨베이어	로봇이 컨베이어 움직임에 맞춰 Pick & Place 수행

✅ 2. 동기화 방식

방식	설명	적용 예
신호 기반	특정 장치가 완료 신호(센서, 출력)를 보내고 다음 장치가 작동	전형적인 시퀀스 제어
시간 기반	타이머나 클럭에 따라 특정 시간 후 동작	이송 후 n초 후 가공
위상 기반	엔코더나 모션 컨트롤러로 위치 동기화	동기 회전 모터 제어
중앙 제어형	마스터 장치가 전체 순서를 제어	상태 머신 기반 제어기
분산 협조형	각 장치가 상태 공유하고 자율 협조	PLC간 통신 + 상태 공유

✅ 3. PLC 구현 구조

(1) 기본 시퀀스 기반 동기화 (단순 구조)

```
IF Step = 0 AND StartButton THEN
    Feeder := TRUE;
    IF FeederSensor THEN
        Step := 1;
    END_IF;
END_IF;

IF Step = 1 THEN
    Machine := TRUE;
    IF ProcessDone THEN
        Step := 2;
    END_IF;
END_IF;
```

(2) 마스터 상태 기반 제어

```
CASE State OF
    INIT:
        Feeder := TRUE;
        IF Sensor1 THEN State := PROCESS;

    PROCESS:
        Machine := TRUE;
        IF Sensor2 THEN State := UNLOAD;

    UNLOAD:
        Ejector := TRUE;
        IF Done THEN State := INIT;
END_CASE;
```

✅ 4. 실시간 동기화 예 (동기 모터, 서보 등)

- **위상 동기화**
  → 엔코더를 통해 **Master-Slave 모터 동기화**
  → 예: Yaskawa, Mitsubishi 서보 PLC 연동
- **PLC 연산**으로 속도 피드백 보정

```
Error := MasterPos - SlavePos;
SlaveSpeed := BaseSpeed + Kp * Error;
```

✅ 5. 통신 기반 복수 PLC 동기화

■ PLC ↔ PLC 간 신호 연동 (예: Profibus, Ethernet/IP)

항목	설명
동작 명령 송신	마스터 PLC가 "시작" 명령 전송
상태 응답 수신	슬레이브 PLC에서 "완료" 응답
에러 연동	한쪽에서 오류 발생 시 전체 중단

> 타이밍 정합이 중요한 경우 실시간성 높은 프로토콜(RT-Ethernet 등) 사용 권장

✅ 6. 주요 설계 고려 사항

항목	설명
각 장치의 상태 피드백 확보	센서, 인버터 상태, 출력 확인 등
각 동작 시간 오차 허용 고려	타이밍 여유값 설정
예외 처리 설계	장치 불능, 응답 없음 대비
우선순위와 제약조건 정의	병렬 장치 간 충돌 방지
시각화 연동(HMI)	동기화 상태 모니터링 구현

✅ 7. 고급 적용 사례

적용 예	기술 설명
다축 동기 모션 제어	고속 엔코더 + 위치 동기화
로봇/AGV 연동 제어	MQTT, OPC-UA 기반 상태 공유
멀티 라인 컨트롤	메인 PLC가 Sub-PLC 다수 제어
동기화 오류 탐지	타임아웃, 오차 검출 로직 포함

✅ 8. 실전 팁

팁	내용
타이머에 의존하는 구조 피하기	항상 센서 기반 전이 우선
상태 머신과 병행 사용	복잡한 동기 제어는 상태 관리가 핵심
리셋 시 상태 동기화 필요	모든 장치 동일 시점으로 초기화 필수

팁	내용
고속 제어 시 PLC 주기 고려	Scan Time과 타이밍 민감도 분석 필요

✅ 정리

- 복수 장치 동기화는 **생산 라인의 흐름 제어 핵심**
- 신호, 시간, 위상, 통신 등 다양한 방식으로 구성 가능
- **상태 기반 로직과 센서 입력 중심 제어가 안정적**
- 고급 동기화는 **서보 제어, 통신 프로토콜, 마스터-슬레이브** 구조 활용

8.3 다중 시퀀스 처리

(PLC에서 복수의 시퀀스를 병렬 또는 조건적 순차로 동시에 제어하는 설계 기법)

✅ 개요

다중 시퀀스 처리(Multi-Sequence Control)는 여러 개의 독립적이거나 상호 관련된 공정 순서를 **동시에 병렬로**, 혹은 **조건적 순차 흐름**으로 처리하는 제어 기법이다.

이는 **다수의 장비, 유닛, 서브 프로세스**가 하나의 PLC 내에서 각자 별도의 제어 흐름을 갖거나, 공정 단계에서 병렬로 작동할 필요가 있을 때 반드시 요구된다.

✅ 1. 적용 사례

분야	다중 시퀀스 항목	설명
조립라인	모듈별 유닛 (피더, 가공, 검사)	독립 제어 필요
포장 시스템	로봇 + 컨베이어 + 용접	각각 병렬로 동작
로봇 셀	로봇팔 + 그리퍼 + 회전대	각 장치의 개별 시퀀스
멀티 트랙 생산	라인 A, B, C 동시 운전	각 라인의 시퀀스 분리

✅ 2. 설계 원리

✅ (1) 시퀀스 분리 설계

각 서브 시퀀스를 **별도의 상태 변수, 로직 블록**으로 분리해 처리

```
// 시퀀스 A
CASE SeqAState OF
  0: IF A_Start THEN SeqAState := 1;
  1: A_Device := TRUE; IF A_Sensor THEN SeqAState := 2;
  ...
END_CASE;

// 시퀀스 B
CASE SeqBState OF
  0: IF B_Start THEN SeqBState := 1;
  1: B_Device := TRUE; IF B_Sensor THEN SeqBState := 2;
  ...
END_CASE;
```

> → **독립적으로 실행되므로 병렬 동작 가능**

✅ (2) 공용 자원 제어 구조 포함

두 시퀀스가 **공통 장비(예: 로봇, 컨베이어)를 공유**하는 경우 **인터록** 또는 **자원 중재(Mutex)** 필요

```
IF NOT ConveyorBusy THEN
  IF SeqA_Req THEN
    ConveyorBusy := TRUE;
    ConveyorOwner := "A";
  ELSIF SeqB_Req THEN
    ConveyorBusy := TRUE;
    ConveyorOwner := "B";
  END_IF;
END_IF;
```

✅ 3. 병렬 시퀀스 상태 구조 예

유닛	상태 변수	상태 흐름 예시
Feeder	`FeedState`	INIT → FEED → WAIT
Press	`PressState`	INIT → DOWN → UP
Sorter	`SortState`	IDLE → CHECK → SORT

→ 이 상태 변수들은 각각 독립적으로 흐름을 갖고, **PLC의 한 스캔 주기 내에서 병렬 처리**된다.

✅ 4. 제어 전략

전략	설명
상태 변수 다중화	각 시퀀스별 별도 `State` 변수 유지

전략	설명
공용 자원 인터록	공유 장비 충돌 방지 논리 필요
타이밍 정합 고려	서로 의존 관계 있는 경우 대기 조건 삽입
HMI 분리 구성	각 시퀀스를 별도 제어 및 감시 가능하도록 설계

✅ 5. 고급 설계 기법

기법	설명
Function Block 단위 분할	각 시퀀스를 함수 블록으로 분리하여 재사용
타스크(Task) 단위 분리 (RT PLC)	실시간 PLC에서는 Task 1/2/3으로 나눠 병렬 처리
비동기 이벤트 기반 처리	이벤트 큐 기반의 병렬 상태 제어 구조
우선순위 기반 시퀀스 스케줄링	자원 충돌 시 우선순위로 실행 순서 결정

✅ 6. 예외 상황 처리 전략

- 병렬 처리 중 하나의 시퀀스에서 오류 발생 시,
  - 전체 정지? or 해당 유닛만 정지?
  - 재시작 조건은?
  - 인터페이스 상태는 유지될 것인가?

→ 이 전략은 **시스템 설계 철학**에 따라 다르며, **로컬 에러 처리 → 전체 에러 전파** 여부 결정 필요

✅ 7. 시각화 연동 구조

- HMI 구성 시 각 시퀀스를 개별 패널로 나눔
- 각 상태 변수 및 타이머, 알람 표시
- 공용 자원 사용 여부 실시간 표시 (Busy 상태 등)

✅ 정리

- **다중 시퀀스 처리는 복잡한 제어 시스템에 필수 구조**
- 각 시퀀스를 **상태 기반으로 분리**하여 병렬 처리하며, 공용 자원은 **중재/인터록** 필요
- 안정성과 유지보수를 위해 **함수 블록 분리, 상태 추적, 타이밍 정합** 설계가 중요
- HMI 및 알람 시스템도 병렬 제어 구조에 맞춰 구성되어야 함

8.4 트리거 조건 기반 인터럽트 처리

(PLC에서 외부 이벤트나 긴급 상황 발생 시 즉시 반응하는 인터럽트 구조 및 설계 기법)

✅ 개요

PLC에서는 일반적으로 **스캔 주기 방식(scan cycle)**으로 프로그램을 실행한다.
하지만 긴급한 이벤트나 특정 조건이 발생했을 때 **즉시 반응해야 하는 경우**,
트리거 기반 인터럽트 처리를 적용한다.

이는 **센서 이벤트, 안전 스위치, 고속 카운터, 긴급 정지 버튼(E-Stop)** 등에서 핵심적으로 사용된다.

✅ 1. 인터럽트 처리란?

> 특정한 **입력 조건(Trigger Condition)**이 발생했을 때
> 메인 루프와 **무관하게 즉시 특정 코드 블록을 실행**하는 방식

- 일반 주기 제어는 `주기적으로 평가`
- 인터럽트는 `발생 즉시, 우선적으로 실행`

✅ 2. PLC에서 인터럽트 적용 예

사례	트리거 조건	처리 동작
긴급 정지	E-Stop 스위치 ON	전체 설비 정지
고속 센서 감지	포토센서 ON	카운터 증가
모터 과전류	인버터 알람 ON	즉시 정지 후 알람 표시
재료 도착 감지	근접 센서 ON	벨트 정지, 가공기 작동

✅ 3. 인터럽트 유형

분류	설명
하드웨어 인터럽트	물리적 입력 조건을 PLC 인터럽트 입력 포트에 연결 (즉시 반응)
소프트웨어 인터럽트	스캔 주기 내 특정 조건 판단 시 강제로 제어 흐름 전환
고속 카운터 인터럽트	펄스 수신 시 카운터가 실시간으로 동작 (엔코더 등)

✅ 4. 구조도 (하드웨어 기반 예)

```
[ Sensor Input ]
      ↓
[ 인터럽트 조건 (예: 상승엣지) 발생 ]
      ↓
[ 인터럽트 루틴 실행 ]
      ↓
[ 지정된 처리 수행 (예: 정지, 상태 저장) ]
      ↓
[ 메인 루틴으로 복귀 ]
```

✅ 5. IEC 61131-3 언어에서의 예 (ST)

트리거 조건 감지 방식 (에지 검출 + 소프트웨어 인터럽트 유사 처리)

```
VAR
    PrevSensor: BOOL := FALSE;
    EdgeRise: BOOL;
END_VAR

EdgeRise := (Sensor = TRUE) AND (PrevSensor = FALSE);
PrevSensor := Sensor;

IF EdgeRise THEN
    // 인터럽트 처리 블록
    Alarm := TRUE;
    StopMotor := TRUE;
END_IF;
```

> ※ 스캔 주기 안에서 "즉시 반응에 근접한 감지"를 구현할 수 있음
> 진정한 실시간 반응은 **PLC 고속 인터럽트 입력**을 사용

✅ 6. 고속 인터럽트 입력 설정 예 (Mitsubishi FX / Siemens S7 등)

항목	설명
PLC 설정 메뉴에서 고속 입력 포트 지정	X0, X1 등 고속 모드 활성화
상승/하강 엣지 감지 지정	Rising Edge, Falling Edge
인터럽트 루틴 지정	Interrupt OB (Siemens) 또는 Task (Omron)
루틴 내 단순 연산/저장/정지 로직	예: 카운터 증가, 비상 정지

✅ 7. 예시: E-Stop 즉시 정지 처리

```
IF E_Stop THEN
    GlobalStop := TRUE;
    Motor := FALSE;
    Alarm := TRUE;
END_IF;
```

> 단, E_Stop은 일반적인 디지털 입력보다 **하드와이어 릴레이 차단 구조 병행** 필요 (이중 안전 구조)

✅ 8. 주의 사항

항목	설명
인터럽트 우선순위 설정 필요	여러 이벤트 발생 시 어느 것을 먼저 처리할지 정의
중복 실행 방지	인터럽트 조건이 연속으로 발생하지 않도록 처리 상태 유지 변수 사용
인터럽트 처리 시간 최소화	짧고 간결한 코드만 배치 (Block 안에서 대기나 루프 금지)
타이밍 조건 고려	노이즈에 의한 오작동 방지 (디바운싱, 필터링 등)

✅ 9. HMI/로깅과 연계

- 인터럽트 발생 시 **이벤트 로그 저장** (발생 시각, 센서 상태, 시스템 상태)
- HMI에 실시간으로 표시 (E-Stop 발생 → 빨간 화면, 부저 등)
- 상태 해제 조건도 명확히 표시 (Reset 버튼 등)

✅ 정리

- 트리거 조건 기반 인터럽트 처리는 **즉시 반응해야 하는 상황에 필수적**
- **센서 엣지 감지 + 조건 검사 + 인터럽트 루틴 실행**으로 구성됨
- 진정한 실시간 처리는 **PLC 고속 입력** 또는 **하드웨어 인터럽트 포트**로 구성
- **안전, 성능, 안정성** 모두를 고려하여 설계해야 함

8.5 조건 분기와 우선순위 설계

(PLC 프로그램에서 복수 조건의 충돌 방지 및 실행 순서를 명확히 하기 위한 제어 설계 기법)

✅ 개요

조건 분기와 우선순위(Priority)는 PLC 제어에서 **여러 조건이 동시에 성립할 수 있는 상황에서,
어떤 동작을 우선적으로 실행할지 결정하고 제어 흐름을 명확히 정의하는 핵심 원칙**이다.

특히 **병렬 조건, 다중 입력, 알람 vs 정상 동작의 충돌 방지**, **긴급 제어 조건**을 다룰 때 매우 중요하다.

✅ 1. 왜 우선순위가 필요한가?

상황	문제점
여러 조건이 동시에 참일 때	서로 다른 동작을 하려 하여 충돌 발생
비상 정지와 정상 동작이 겹칠 때	동작이 멈추지 않거나 위험 발생
조건 분기 없이 나열된 경우	실행 순서가 모호하고 유지보수 어려움

✅ 2. 조건 분기 구조 설계 기본

(1) `IF - ELSIF - ELSE` 방식 사용

```
IF Emergency THEN
    StopAll := TRUE;

ELSIF Alarm THEN
    ShowAlarm := TRUE;

ELSIF Start THEN
    StartMotor := TRUE;

ELSE
    Idle := TRUE;
END_IF;
```

> → 우선순위: Emergency > Alarm > Start > Idle

---

(2) 조건 가중치 논리 설계 (우선순위 테이블)

조건	우선순위 (낮음→높음)	설명
정상 가동	1	모든 조건 충족 시 실행
공정 종료	2	완료 감지 시 전환
경고 발생	3	일시정지 or 알람 표시
비상 정지	4	최상위. 즉시 전체 정지

✅ 3. 상태 기반 우선순위 분기

```
CASE State OF
  INIT:
    IF Emergency THEN State := ERROR;
    ELSIF Start THEN State := RUN;
```

```
  RUN:
    IF Emergency THEN State := ERROR;
    ELSIF Alarm THEN State := WARNING;

  WARNING:
    IF Reset THEN State := RUN;

  ERROR:
    IF Reset THEN State := INIT;
END_CASE;
```

> → 상태(State)에 따라 조건이 달라지고, 각 조건에도 우선순위가 내장됨

✅ 4. 래더 다이어그램 예 (LD)

```
| Emergency |──[ ]──┬──( Stop All )
                    |
| Alarm     |──[ ]──┘──( Show Alarm )

| Start     |──[ ]──────( Start Motor )
```

→ 이 구조에서는 **위에서부터 먼저 true가 된 조건만 실행**되므로, 위치가 곧 우선순위
→ 따라서 **긴급 조건은 가장 위에 배치**

✅ 5. 충돌 방지 설계 패턴

패턴	설명
Mutual Exclusion	동시에 두 조건이 활성화되면 하나만 작동하도록 논리 분리
Flag Clear 후 전환	조건 해제 후 다른 조건으로 넘어감 (e.g. Emergency 해제 후 Start 가능)
Delay / 타이머 기반 전환	분기 사이에 딜레이 삽입해 충돌 방지

✅ 6. 고급 응용: 우선순위 스케줄러 설계

```
Priority := 0;

IF Emergency THEN
    Priority := 4;
ELSIF Alarm THEN
    Priority := 3;
ELSIF Start THEN
    Priority := 2;
ELSE
    Priority := 1;
END_IF;
```

```
12
13  CASE Priority OF
14      4: StopAll := TRUE;
15      3: ShowAlarm := TRUE;
16      2: StartMotor := TRUE;
17      1: Idle := TRUE;
18  END_CASE;
```

→ 복잡한 조건을 정수값 하나로 압축 → **유지보수성 향상**

✅ 7. 상태 및 분기 조건 시각화

- HMI에 각 조건 및 우선순위를 아이콘으로 표시
- 현재 활성화된 상태를 하이라이트
- 사용자가 조건 간 충돌 여부를 직관적으로 확인

✅ 8. 실무 팁

항목	조언
긴급/정지 조건은 항상 최상위에	하위 조건보다 항상 먼저 실행
우선순위 구조 명시	설계 문서나 주석에 우선순위 명시
중복 실행 방지 논리 삽입	동일 동작 중복 트리거 방지 (`동작 완료 플래그` 활용)
조건 해제 후 전환 구조 명확화	모든 조건이 자동으로 다시 실행되지 않도록 함

✅ 정리

- PLC에서 조건 분기와 우선순위는 **안전성과 논리 일관성 확보에 필수적**
- `IF - ELSIF`, `CASE`, `State + Priority` 방식으로 제어 흐름을 구조화
- **우선순위 설계 기준을 명확히 수립**하고, **코드에 반영 + 문서화 + HMI 연동**까지 고려해야 안정적인 시스템 운영 가능

8.6 실시간 제어 주기 고려

(PLC에서 신뢰성 있는 제어를 위한 Scan Time, 주기 설계 및 실시간성 확보 전략)

✅ 개요

PLC는 프로그램을 반복적으로 실행하는 **스캔 주기(scan cycle)** 기반 제어 시스템이다.
실시간성이 요구되는 제어에서는 이 **스캔 시간과 주기(Time Determinism)**를 정확히 고려해 설계해야 한다.

특히 **센서 감지 지연, 출력 응답 지연, 제어 불안정 현상** 등은 주기 설정의 미흡에서 기인하는 경우가 많다.

✅ 1. 스캔 주기(Scan Cycle)란?

> PLC는 기본적으로 아래 4단계의 루프를 반복 수행한다.

단계	설명
① 입력 스캔	센서 등 입력 상태 읽기
② 논리 연산	프로그램 실행 (LD, ST 등)
③ 출력 갱신	결과를 출력 포트로 전송
④ 내부 처리	통신, 진단, 유지보수 등

➡️ 이 한 바퀴를 도는 데 걸리는 시간이 **스캔 주기 (Scan Time)**

✅ 2. 스캔 시간 확인 방법

- **PLC 제조사별 도구 제공**
  - Siemens: `OB1`, `Cycle Time` 모니터
  - Mitsubishi: `SD520` 레지스터
  - Omron: `A195` 시스템 메모리 등
- 예: 평균 5ms, 최대 10ms → 즉 1초에 약 100~200번 루프 가능

✅ 3. 스캔 주기와 제어 응답성 관계

항목	영향
스캔 시간이 너무 길면	입력에 대한 반응이 늦음 → 실시간성 저하
스캔 시간이 너무 짧으면	연산 오버헤드, 통신 불안정 발생 가능
시간 민감한 제어	반드시 고정된 주기에서 안정적 실행 필요 (PID, 고속 계수 등)

✅ 4. 실시간 제어 주기 설계 시 고려 사항

항목	설명
최소 응답 시간 요구	예: 센서 트리거 → 20ms 내 밸브 작동
PLC 프로그램 복잡도	스캔 시간은 로직량에 비례 증가함
인터럽트 및 고속 입력 사용 여부	고속 이벤트는 별도로 처리해야 정확도 보장
통신 딜레이 포함 여부	외부 장치와 통신 시간도 포함하여 고려해야 함

✅ 5. 예시: 응답 지연 발생 구조

```
IF Sensor = TRUE THEN
    Valve := TRUE;
END_IF;
```

- 위 코드가 있는 스캔 루프가 30ms 걸린다면?
  - 센서가 ON → 최대 30ms 후에야 Valve가 ON
  - 실제 기계에서는 눈에 띄는 지연으로 인식될 수 있음

✅ 6. 실시간성 확보를 위한 전략

전략	설명
프로그램 분리 구조 (OB/Task 분리)	시간 민감한 로직만 독립 Task로 구성
인터럽트 루틴 사용	고속 트리거는 인터럽트/고속 입력으로 처리
타이머 기반 주기 실행 제어	`IF Timer.Q THEN …` 방식으로 정해진 시간마다만 수행
PID 등은 고정 주기 실행 필수	예: 100ms마다 정확히 실행되어야 함

🔧 예: 100ms마다 고정 주기 동작 블록

```
IF CycleTimer.Q THEN
    CycleTimer(IN := TRUE, PT := T#100ms);
    // 주기 실행 로직
    MotorSpeed := MotorSpeed + 1;
END_IF;
```

✅ 7. 시스템 레벨 주기 설계 예시

기능 블록	주기	설명
PID 제어	100ms	고정 주기 필수
통신 처리	200~500ms	너무 자주하면 부하 발생
모터 제어	50~100ms	실시간 반응 필요
로깅	1~5초	비실시간 대상으로 느슨하게

✅ 8. Scan Time 안정성 확보 방법

방법	설명
프로그램 최적화	불필요한 루프, 중복 조건 제거
조건 분기 구조 간결화	IF-ELSE 중첩 줄이기
함수 블록화 및 분산	로직을 기능 단위로 나누어 관리
고속 처리 기능 활용	제조사 제공 고속 처리 블록 활용 (Fast OB 등)

✅ 9. 시각화 (HMI 등)에서 고려할 점

- HMI 주기와 PLC 주기가 다르면 **데이터 왜곡 발생**
- HMI가 2초 주기인데 PLC는 50ms라면?
  → 빠른 변화 감지 불가
  → **중간 버퍼링/평균값 처리 필요**

✅ 정리

- 실시간 제어에서 **PLC 스캔 주기(Scan Time)**는 시스템의 반응성과 신뢰성을 결정하는 핵심 요소
- **제어 요구 응답 시간**, **로직 복잡도**, **인터럽트 사용 여부**를 고려해 설계
- **시간 민감한 블록은 고정 주기로 실행**, **나머지는 느슨한 루프로 분리**
- **Scan Time은 짧을수록 좋지만 너무 짧으면 안정성 저하**

8.7 장애 발생시 리셋/보존 설계

(PLC 시스템에서 에러 발생 후의 안전한 복구, 상태 유지, 리셋 전략)

✅ 개요

PLC 제어 시스템에서 **전원 장애, 통신 오류, 센서 이상, 예외 조건 등**이 발생했을 때,
기기의 상태를 어떻게 복구할 것인가? 리셋은 어떤 방식으로 할 것인가?는
안전성과 시스템 신뢰도에 직결된다.

장애 이후에도 **정확한 상태 복원**, 또는 **안전한 정지와 명확한 리셋**이 설계되지 않으면
기계 파손, 인명 사고, 공정 오류 등이 발생할 수 있다.

✅ 1. 장애 유형별 분류

장애 종류	예시	설계 고려 사항
전원 장애	갑작스러운 정전	복귀 시 이전 상태 복원 또는 초기화 필요
통신 장애	HMI/센서/로봇 연결 끊김	재접속 및 재동기화 구조 필요

장애 종류	예시	설계 고려 사항
입출력 오류	센서 고장, 액추에이터 미동작	에러 감지 및 대기 상태 전환
소프트웨어 예외	변수 범위 초과, 루프 오버플로	예외 처리 및 재시작 논리 필요

✅ 2. 상태 보존 vs 초기화 전략

전략	특징	적용 시점
상태 보존	이전 동작 상태를 유지하여 재가동	정전 후 자동 복구 필요한 공정
상태 초기화	모든 변수/동작을 초기 상태로 복귀	안전 우선, 수동 리셋이 요구되는 경우

> 실무에서는 **안전 관련 회로는 초기화**, **공정 관련 로직은 보존**하는 **혼합 설계**가 일반적

✅ 3. 전원 OFF 후 상태 보존 기법

✅ (1) Retain 변수 사용

```
VAR RETAIN
  ConveyorState: BOOL;
  StepCounter: INT;
END_VAR
```

- PLC 전원 재가동 후에도 위 변수는 값 유지
- Siemens, Mitsubishi, Omron 등 모든 PLC에서 지원 (단, 설정 필요)

✅ (2) EEPROM / Flash 저장

- 일정 주기로 중요한 상태를 EEPROM/Flash에 저장
- PLC 전원 복귀 시 읽어와서 초기화 단계에 복원
- 단점: 저장 주기가 짧으면 수명 단축 가능

✅ 4. 장애 발생 시 상태 스냅샷 구조

```
IF ErrorDetected THEN
    ErrorState := TRUE;
    SavedStep := CurrentStep;
    SavedMotorStatus := MotorStatus;
END_IF;
```

- → 리셋 조건 충족 시 아래와 같이 복구

```
IF ErrorReset THEN
    ErrorState := FALSE;
    CurrentStep := SavedStep;
    MotorStatus := SavedMotorStatus;
END_IF;
```

✅ 5. 리셋 동작 설계 구조

리셋 조건	동작
수동 리셋 버튼 누름	모든 변수 초기화, 안전 대기 상태 전환
자동 리셋 조건 충족	에러 해제 후 자동 복귀
조건부 리셋	특정 상태에서만 복귀 가능 (예: 제품 제거 후만 리셋 허용)

✅ 6. 리셋 로직 예시 (Ladder Diagram)

```
| ErrorState |──[ ]──────( R ) ResetOutputs
| ResetBtn   |──[↑]──────( S ) InitState
```

- ↑: 상승 엣지 트리거
- `(S)` : Set, `(R)` : Reset

✅ 7. 보존이 필요한 상태 종류 예

항목	설명
생산 순번	몇 번째 공정 중이었는지
모터 상태	ON/OFF, 속도 등
타이머 값	잔여 시간 등
공정 단계	FSM 기반 상태 변수

✅ 8. HMI 연동: 에러 복구 흐름

1. 에러 발생 시 → 상태 저장 + 알람 출력
2. HMI에 복구 버튼 또는 경고 메시지 출력
3. 사용자가 수동 리셋 or 조건 만족 시 자동 복구
4. 복구 상태도 로그로 저장 (누가, 언제, 어떤 에러를 어떻게 복구했는지)

✅ 9. 안전 설계 고려사항

항목	설명
리셋 전에 반드시 안전 확인	오퍼레이터가 물리적으로 상황을 확인해야 복귀 가능
보존된 상태로 복귀가 위험할 경우 무시	예: 모터가 갑자기 다시 동작하면 위험한 경우
단계별 리셋 허용	전체가 아닌 단계적으로 복원하여 안정성 확보

✅ 10. 예외 설계 팁

- 장애 상태를 **별도 변수 (`ErrorState`, `RecoveryFlag`)**로 추적
- 복구 성공 여부도 기록 (e.g. `RecoverySuccess`)
- EEPROM 기록은 주기 제한 필수 (예: 30초 이상 간격)

✅ 정리

- 장애 발생 시 복구 전략은 **"무조건 초기화" vs "상태 보존 복귀"**의 균형이 핵심
- 안전에 직결되는 회로는 **초기화**, 공정의 연속성이 중요한 경우는 **보존 후 복원**
- **Retain 변수, EEPROM 저장, 복구 로직**을 조합하여 시스템의 신뢰성과 사용성을 모두 확보