12. 전력 관리 및 열 설계

12.1 저전력 아키텍처 설계 (Clock Gating, Power Gating)

> 전력을 '사용할 때만' 쓰는 똑똑한 설계

🧠 전력 소모 구성

디지털 회로에서의 전력 소모는 크게 다음 세 가지로 나뉜다:

항목	설명	수식
동적 전력	스위칭에 따른 소비 (주로 클럭)	`P_dynamic = α·C·V²·f`
정적 전력	누설 전류 (MOSFET OFF 상태에서도 흐름)	`P_static = I_leak · V`
단락 전류	스위칭 중 순간 경로가 열릴 때 흐름	작지만 무시 못함

> 이 중 **동적 전력**이 전체의 대부분을 차지하기 때문에
> 이를 제어하는 **Clock Gating**이 1차 타겟이 되고,
> **Power Gating**은 그 다음 단계의 절전 기술이다.

🟩 1. Clock Gating (클럭 게이팅)

✅ 개념

불필요한 회로에 클럭을 차단하여 동작 자체를 멈추는 방식
→ 불필요한 플립플롭 토글을 막아 **동적 전력 절감**

✅ 원리

클럭 신호에 **Enable 제어 신호를 삽입**하여
회로 블록이 활동할 때만 클럭이 전달되게 함

```
// 기본 구조 (비효율적)
always @(posedge clk)
    if (enable) q <= d;

// Clock Gating 적용
assign gated_clk = clk & enable;
always @(posedge gated_clk)
    q <= d;
```

> 실제 구현 시엔 **Clock Gating Cell**을 사용하여
> 클럭 지터, 글리치 없이 안전하게 차단해야 함

✅ 위치 적용 예

적용 위치	예시
파이프라인 단계별	IF/ID/EX 레지스터에만 클럭 공급
캐시 컨트롤러	사용 중인 뱅크에만 클럭 공급
주변 장치	UART, SPI, GPIO 등이 동작할 때만 클럭 연결

🟥 2. Power Gating (파워 게이팅)

✅ 개념

회로 전체 또는 일부에 **전력 자체를 차단(전원 차단)**
→ 정적 전력(누설 전류)까지 제거 가능

✅ 구성

컴포넌트	설명
Sleep 트랜지스터	PMOS 또는 NMOS로 회로 전원을 끊는 스위치 역할
Power Domain	전력 차단이 가능한 블록 단위
Isolation Cell	파워오프된 블록이 주변에 신호 노이즈를 주지 않도록 차단
Retention Cell	필수 정보만 유지하고 나머지 전원 오프

✅ 동작 방식

1. 파워게이팅 신호 활성화
2. Sleep 트랜지스터로 전원 OFF
3. 필요 시 다시 ON → 리텐션 정보로 복원

📏 Clock vs Power Gating 비교

항목	Clock Gating	Power Gating
차단 대상	**클럭 신호**	**전원 공급**
절감 효과	동적 전력만	정적 + 동적 전력
리턴 시간	빠름 (몇 ns)	느림 (수 μs 이상)
리텐션	필요 없음	필요 (Retain Cells)
리스크	클럭 글리치 주의	데이터 손실 가능성 있음
구현 난이도	비교적 쉬움	SoC 수준의 설계 필요

💡 적용 사례

시스템	적용 예
ARM Cortex-M	Sleep, DeepSleep 모드에서 클럭/전원 부분 차단
ARM Cortex-A	각 코어에 Power Domain 설정 + Gating
Apple M1/M2	고성능 코어/저전력 코어 별도 Power Gating + DVFS
Qualcomm SoC	GPU/ISP/NPU 각 영역에 독립 파워게이팅
FPGA 설계	Gated Clock IP Core + Power Island 설계 가능

🛠️ 설계 시 고려사항

항목	설명
타이밍 검증	Gated Clock 경로 제외 설정 필요
글리치 방지	Gating Cell 사용 필수
클럭 도메인 관리	Gated 영역은 CDC 경계가 됨
Wake-up Latency	Power Gating은 복귀 시간 고려 필수
Retention 비용	유지할 정보 선별 및 비용 절충 필요

📌 요약 정리

항목	Clock Gating	Power Gating
절감 전력	동적 전력	동적 + 정적 전력
구현 난이도	쉬움 (Cell 삽입)	복잡 (PD/Isolation 필요)
복귀 시간	빠름 (ns)	느림 (μs)
적용 위치	플립플롭, 서브블록	Subsystem, IP 블록 단위
대표 기술	Gated Clock Cell	Sleep Transistor, Isolation Cell

12.2 동적 전압/주파수 스케일링 (DVFS)

🧠 DVFS란?

DVFS(Dynamic Voltage and Frequency Scaling)는
CPU나 SoC의 **전압(V)과 클럭 주파수(f)**를 **실시간으로 조절**하여
성능과 전력 효율을 동적으로 균형 조정하는 기술이다.

🔌 왜 전압과 주파수를 조절해야 할까?

디지털 회로의 **동적 전력 소비 공식**은 다음과 같다:

```
P_dynamic = α × C × V² × f
```

항목	설명
α	스위칭 계수 (0~1)
C	부하 정전용량
V	공급 전압
f	클럭 주파수

> ⚠️ **전압(V)을 줄이면 전력은 제곱으로 감소**
> 하지만 **전압이 낮아지면 고속 연산이 어려워져서 f도 낮아져야 해**

🔄 DVFS 기본 동작 흐름

```
[Workload 감지]
      ↓
[필요 성능 분석]
      ↓
[최적 주파수(f) 결정]
      ↓
[그에 맞는 최소 전압(V) 설정]
      ↓
[PLL / Voltage Regulator 제어]
```

🧩 하드웨어 구성 요소

구성 요소	설명
PLL (Phase Locked Loop)	동작 주파수(f) 조절
PMIC (Power Management IC)	전압 조절 기능 제공
Voltage/Frequency Table (OPP)	가능한 조합 목록 (Operating Performance Point)
감시 회로	온도, 부하, 전압 안정성 모니터링

⚙️ 소프트웨어 제어 구조 (Linux 기준)

컴포넌트	역할
CPUFreq Driver	각 클럭/전압 상태 전환 함수 제공
Governor	정책 기반 동작 결정 (Ondemand, Performance, Powersave 등)

컴포넌트	역할
Thermal Driver	온도 기반 클럭 제어
devfreq	GPU, DSP 등 Non-CPU 디바이스의 DVFS 제어용

🔁 CPUFreq Governor 예시

Governor	설명
Performance	항상 최대 속도
Powersave	항상 최소 속도
Ondemand	부하가 급증하면 빠르게 증가
Conservative	천천히 오르내림
Schedutil	스케줄러 기반 예측 제어 (최신 우분투 등에서 기본)

🧠 OPP (Operating Performance Point)

DVFS는 임의 조합이 아니라 **안정적으로 검증된 조합만 사용**

Index	Frequency	Voltage
0	400 MHz	0.85 V
1	800 MHz	1.00 V
2	1.2 GHz	1.10 V
3	1.6 GHz	1.25 V

🧪 DVFS 적용 예: ARM big.LITTLE 구조

구성 요소	설명
big 코어	고성능, 높은 전력 (ex. Cortex-A76)
LITTLE 코어	저전력, 저성능 (ex. Cortex-A55)
→ 각각 DVFS로 독립 제어 가능	
→ 필요 시 Task migration 병행	

📱 적용 사례

플랫폼	DVFS 특징
스마트폰	화면 주사율, 앱 실행 상태 따라 클럭 조절

플랫폼	DVFS 특징
노트북	배터리 모드에서는 낮은 전압/클럭으로 유지
서버	가상화 상태에서 코어별 DVFS 적용
자동차 SoC	실시간성 영역은 고정 클럭, 비실시간 영역은 DVFS 적용

🔐 설계 시 고려사항

항목	설명
안정성	낮은 전압에서는 타이밍 오류 발생 가능 → 인증된 OPP만 사용
Wake-up Latency	주파수/전압 변경에는 수십~수백 μs 지연 발생
온도 제한	온도가 올라가면 최대 OPP 제한 필요
IR Drop	전압 낮춤에 따른 전류 부족 문제 가능성

📌 요약 정리

항목	설명
DVFS 목적	전력 효율 향상 + 발열 제어
전력 공식	$P \propto V^2 f$ (전압 조절이 가장 효과적)
제어 요소	PLL, PMIC, OPP 테이블
정책 기반 제어	Linux governor, thermal zone, workload 기반
활용 분야	모바일, 서버, 임베디드 SoC 전 분야
설계 조건	안정성, 응답 지연, 온도 고려 필수

12.3 서멀 쓰로틀링 및 발열 제어 회로

> "과열로부터 CPU를 지키는 마지막 방어선"

🧠 왜 발열 제어가 필요한가?

디지털 회로는 연산이 증가할수록 발열이 따라오고,
온도가 **기준치를 초과**하면 다음 문제가 발생한다:

문제	설명
Gate Delay 증가	높은 온도 → 트랜지스터 동작 속도 저하
Leakage 증가	정적 전력 소모 증가 (특히 FinFET 이하 공정)
전기적 손상	고온 상태 지속 → 트랜지스터 열화

문제	설명
시스템 불안정	오류율 증가, 시스템 재부팅 또는 다운

🌡️ 온도 감지 회로 (Thermal Sensor)

✅ 센서 종류

센서 종류	설명
온도 다이오드	PN 접합 전압(Vf)이 온도에 따라 감소
Bandgap 기반 센서	온도 보정 회로 내장, 정확도 높음
Digital Temp Sensor (DTS)	ADC 내장, 디지털 신호로 직접 추출 가능

✅ 센서 위치

- 각 CPU Core 내부 (Core-level)
- GPU, NPU, Memory Controller 근처
- Die 중앙 및 주변 Thermal Hotspot

🔁 Thermal Throttling 동작 흐름

```
[온도 상승 감지]
    ↓
[정해진 임계치(T1) 도달]
    ↓
[DVFS or 클럭 Gating → 성능 저하]
    ↓
[T2 이상이면 하드웨어 강제 Shutdown]
```

🔥 예시 임계치

단계	임계 온도	조치
T1	80°C	DVFS 적용, 클럭/전압 감소
T2	95°C	클럭 정지 또는 CPU off
T3	≥105°C	하드웨어 강제 종료 (Thermal Trip)

🧩 스로틀링 기법 종류

방식	설명
DVFS	클럭과 전압을 낮춰 전력 소모를 줄임
Idle Injection	주기적으로 연산 중단 (일종의 Sleep)

방식	설명
Core Parking	일부 코어를 비활성화
클럭 정지 (Clock Stop)	위험 구간에서는 클럭을 아예 멈춤
Power Gating	전원 자체를 차단 (심한 경우)

🛠️ 하드웨어 보호 회로: Thermal Trip

구성 요소	기능
Thermal Comparator	온도센서 출력이 임계값을 초과하는지 비교
Trip Signal	SoC 내부 회로를 차단하는 신호 생성
PMIC 연동	전원 차단 제어 또는 리셋 신호 발생

→ 이 과정은 **OS 개입 없이 하드웨어 레벨에서 동작**함

⚙️ OS 연동: Linux Thermal Framework

구성 요소	설명
thermal_zone	온도센서와 연계된 측정 영역
cooling_device	클럭, DVFS, 팬, CPU 오프 등 제어 대상
trip_point	스로틀링 또는 shutdown을 유발하는 온도
governor	어떤 방식으로 반응할지 정책 지정

📄 예시 (Device Tree):

```
trip-point@0 {
    temperature = <80000>; // 80°C
    type = "passive";
    cooling-device = <&cpu0 0 1>;
};
```

📱 실제 적용 예

✅ ARM Cortex-A 계열

- DTS 내장 (Digital Temp Sensor)
- `TSENS` 모듈로 온도 정보 전달
- `CPUFreq` + `Thermal` 연동

✅ Intel Core / Xeon

- `Digital Thermal Sensor` 내장
- `PROCHOT` 핀: 발열 시 외부 회로로 알림
- `THERMTRIP` 핀: 비상시 하드웨어 전원 차단

✅ Apple M 시리즈

- 고성능 코어와 저전력 코어 각각 Thermal 관리
- passive + aggressive throttling 조합

📏 스로틀링 vs 샷다운 비교

항목	Throttling	Thermal Shutdown
온도 범위	경고 수준 (70~90°C)	위험 수준 (≥100~110°C)
목적	발열 완화	하드웨어 보호
반응 속도	즉시 적용, 점진적 완화	즉시, 강제 시스템 종료
OS 개입	있음 (thermal framework)	없음 (하드웨어 전용)

📌 요약 정리

항목	설명
온도 센서	디지털 or 아날로그 방식, 다이 내부 센싱
스로틀링 방식	DVFS, Idle Injection, 클럭 정지
Thermal Trip	하드웨어 차단 보호 회로
Linux 연동	thermal_zone + cooling_device 기반
적용 시스템	ARM, Intel, Apple 등 모든 SoC 필수
설계 목적	시스템 안정성 확보 + 장기적 신뢰성 보장

12.4 전력 프로파일링 및 측정 기법

> '측정하지 못하면 최적화할 수 없다'

🧠 전력 프로파일링이란?

전력 프로파일링(Power Profiling)은
시스템의 각 기능/시간대별로 **전력 소비 패턴을 시각화하고 분석**하여
최적화 포인트를 찾는 데 사용되는 기법이다.

> 🔍 목적:

- 전력 소모가 많은 루틴 식별

- DVFS, Power Gating 적용 효과 분석
- 설계 단계에서 전력 예산 검증
- 실시간 제어(서멀/배터리 수명) 기반

🔌 1. 측정 방식 분류

✅ A. 하드웨어 기반 측정

방식	설명
샘플링 전류센서	전원선에 저항(R_shunt) 연결 후 전압 측정
I²C/SMBus 전력칩	INA219, INA231, MAX34407 등으로 전압/전류 모니터링
오실로스코프 + 프로브	실시간 전력 파형 측정 가능
FPGA 내 전력 모듈	Xilinx Power Monitor, Intel PMBus 모듈 등

```
[전원] ──[Shunt R]──[SoC]
              |
              ▼
      [INA219 / ADC] → [I2C / USB]
```

✅ B. 소프트웨어 기반 추정

방식	설명
Linux PowerTop	CPU, 디바이스 상태 기반 통계 예측
Android Battery Historian	각 앱의 전력 사용 추정
PMU (Performance Monitoring Unit)	명령어 수, 메모리 접근 → 전력 추정
SoC 드라이버 내계측	각 Power Domain의 DVFS 상태, Idle 상태 기록

⏱️ 2. 시간 기반 프로파일링

- 주기적으로 샘플링하여 시간축으로 **전력 소비 변화 시각화**
- GPU, NPU, DDR, CPU 사용량과 **전력 패턴을 상관 분석**

```
Time (ms) | CPU (mW) | GPU (mW) | DDR (mW)
----------|----------|----------|---------
0         | 50       | 10       | 20
10        | 110      | 40       | 30
20        | 95       | 60       | 35
```

🧩 SoC 전력 도메인(Power Domain) 별 분석

도메인	측정 항목	예시
CPU	DVFS 상태, 활성 시간, 사용률	Core0~Core3
GPU	워크로드 기반 사용 시간	Mali, Adreno
NPU/DSP	연산량, Idle 비율	Vision DSP
DDR	Access 패턴, Bandwidth	LPDDR4/5
Peripherals	켜짐/꺼짐 상태 시간	USB, UART, SPI 등

→ 이 데이터는 `/sys/class/powercap/`, `/sys/class/thermal/`, `devfreq`, `cpuinfo` 등에서 수집 가능

🛠️ 3. 실전 측정 도구

도구	환경	설명
PowerTop	Linux	C-States, Device Activity, Power Hint
ARM Streamline	ARM SoC	CPU/GPU/Memory 프로파일링 통합 도구
INA231 with I2C	하드웨어	CPU 전원선의 mA 단위 실측
Xilinx XPower Analyzer	FPGA	구현 전 RTL 수준에서 소비 예측
Android Battery Historian	Android	앱별 배터리 소비 추정 그래프화
Jetson Power GUI	NVIDIA Jetson	CPU/GPU/NVDEC/NVENC 소비 실시간 확인

📏 4. 단위

단위	설명
W (Watt)	전력
mW/ms	시간 단위 소비량
mAh	배터리 용량 단위 (전력 x 시간)
Efficiency (%)	유효 연산 대비 전력 비율

📉 5. 전력 최적화 피드백 루프

```
[측정] → [분석] → [Hotspot 식별] → [DVFS / Gating 적용] → [재측정]
```

→ 이 과정을 반복하며 점진적으로 최적화

🧠 고급 항목: 전력 이벤트 기반 태깅

- 프로파일링 시 **특정 코드 구간을 태그(tag)**하여
  해당 루틴이 실행될 때 전력 소비만 추출 가능

```
gpio_set(POWER_TAG1, 1);
critical_function();
gpio_set(POWER_TAG1, 0);
```

→ 오실로스코프나 GPIO 로거에서 해당 구간의 소비만 정밀 분석 가능

📌 요약 정리

항목	설명
전력 프로파일링	시간/기능별 전력 패턴 시각화 및 분석
하드웨어 방식	샘플링 센서, INA231, PMIC
소프트웨어 방식	Linux PowerTop, Android Stats
도메인별 분석	CPU, GPU, NPU, DDR, IO 장치
적용 도구	ARM Streamline, INA231, Jetson GUI 등
피드백 루프	측정 → 최적화 → 재측정