2 장 : 임베디드시스템 개발 방법

ATmega128 마이크로콘트롤러를 이용한 임베디드시스템 구현

JCnet 제이씨넷 신 상 석

목차 (Contents)

임베디드시스템 개발 절차

요구사항 정의

- 요구사항 정의 개념
  - 어떤 시스템을 만드려고 하는지에 대한 내용을 정의하는 것 (What)
  - 최종 사용자의 요구사항을 반영
  - 가능한 수치가 들어가 있으면 좋음
- 요구사항의 요소
  - 기능 : 요구 기능 (Must, May, 옵션), 프로세서 기능, OS 기능
  - 성능 : MIPS/FLOPS, 데이터전송속도(Mbit/sec) 등
  - 안정성 : MTBF, Fail Safety, 백업장치 등
  - 기타
    - 외관 : 크기, 재질, 디자인
    - 인증 : UL, CE, KS, ISO-9000 등
    - 기타 제약사항 : 일정, 비용, IP 소유권 등
- 결과물 : <요구사항정의서>

요구사항 정의가 중요한 이유

오류 수정 비용

1 : 10 : 100

설계시 시험시 시장배포후

시스템 설계

- 중요 H/W, S/W 스펙 선정
  - 전체구조, 프로세서, 메모리, I/O, 중요 chip, OS, middleware, 응용 프로그램, 개발환경 등
- 프로세서 선정
  - 기능 및 구조 : 데이터 크기, CISC vs RISC, 내장 I/O 콘트롤러 등
  - 성능 : MIPS/FLOPS/MOPS
  - OS와의 연계성, S/W Compatibility
  - 종류 : AVR계열, 8051계열, PIC계열, ARM계열, MIPS계열, M68K계열, PowerPC 계열
- OS 선정
  - OS 장착 vs OS 비장착
  - Realtime(soft, hard) or non-realtime
  - 소스 Open 여부, 라이선스 비용
  - 개발 환경
  - 종류 : Embedded Linux, Windows CE, VxWorks, ParmOS
- 결과물 : <시스템설계서>
- 주의 : 지속적으로 <요구사항 정의>를 만족하고 있는지 체크

요구사항정의를 지속적으로 체크해야 하는 이유

고객이 요구한 사항

개발자가 이해한 사항

* 문서화, 피드백 중요

앗!

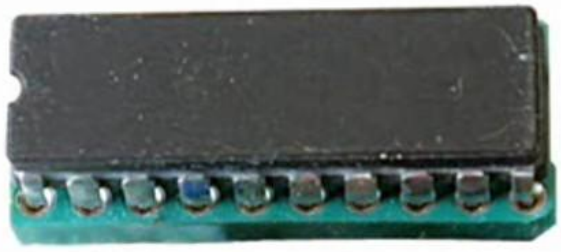
실제로 고객이 요구한 사항

프로세서 선정

- 프로세서 분류
  - 내장 형태에 따른 분류
    - 마이크로프로세서 vs 마이크로콘트롤러
  - 데이터 버스(레지스터) 크기에 따른 분류
    - 8비트 vs 16비트 vs 32비트 vs 64비트
  - 명령어세트 구조에 따른 분류
    - CISC vs RISC
  - 메모리 접근 방법에 따른 분류
    - 폰노이만 구조 vs 하버드 구조
  - 데이터 표현 방법에 따른 분류
    - Big Endian vs Little Endian

프로세서 선정

- 프로세서의 종류
  - **PIC 계열**
  - **8051 계열**
  - **AVR 계열**
  - ARM 계열 – ARM7, ARM9, Cortex-M3
  - PowerPC 계열
  - MIPS 계열
  - Intel 계열

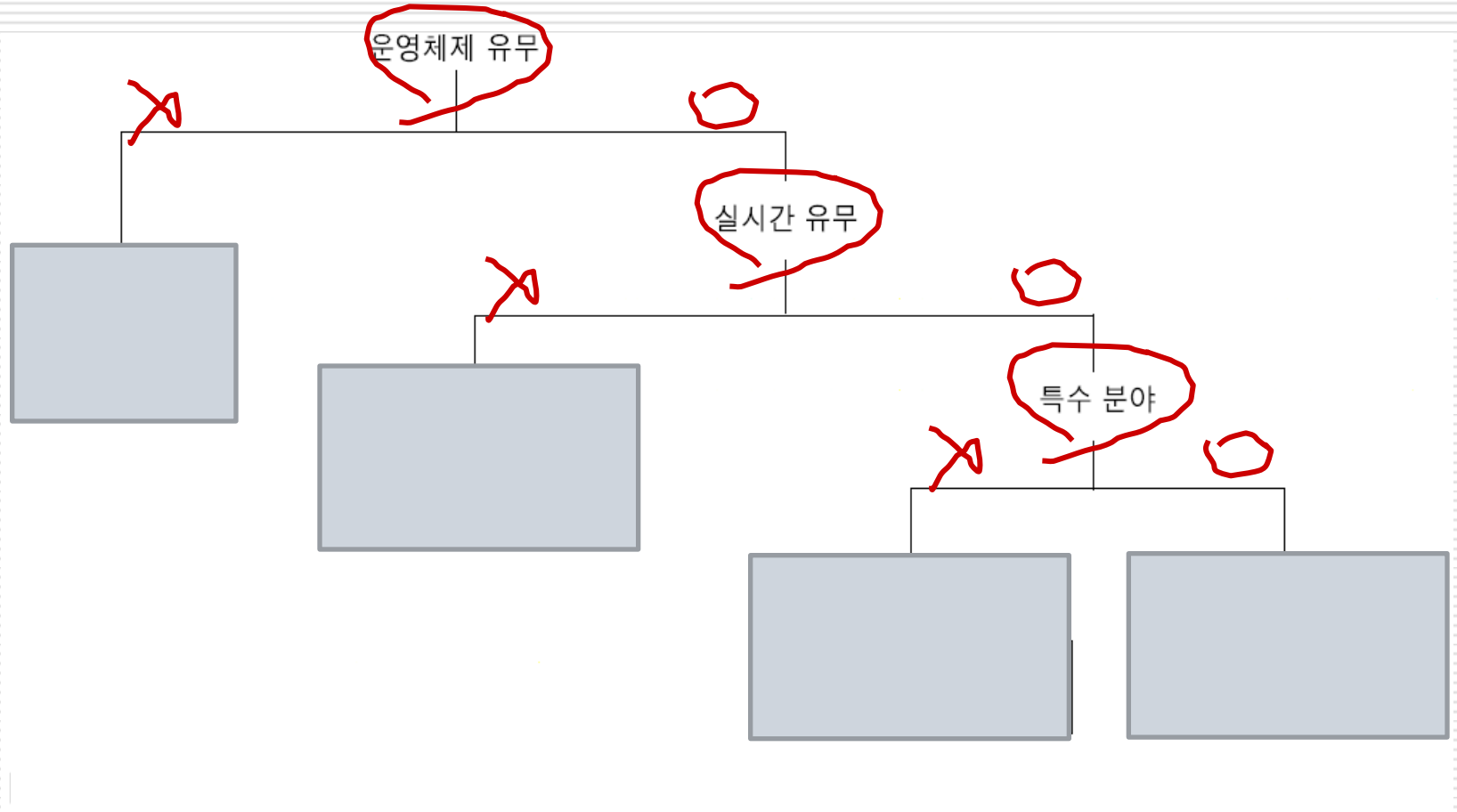


OS 선정

- OS의 분류
  - 범용 vs 임베디드용
  - Realtime vs non-Realtime
  - 상용 vs 비상용
- OS의 탑재 필요성
  - 시스템 고기능화
  - 빠른 개발과 유지 보수 용이
  - 쉬운 사용자 인터페이스

% 기능이 복잡하지 않은 경우는 굳이 OS를 탑재할 필요가 없음!!

OS 선정시 기본 고려사항

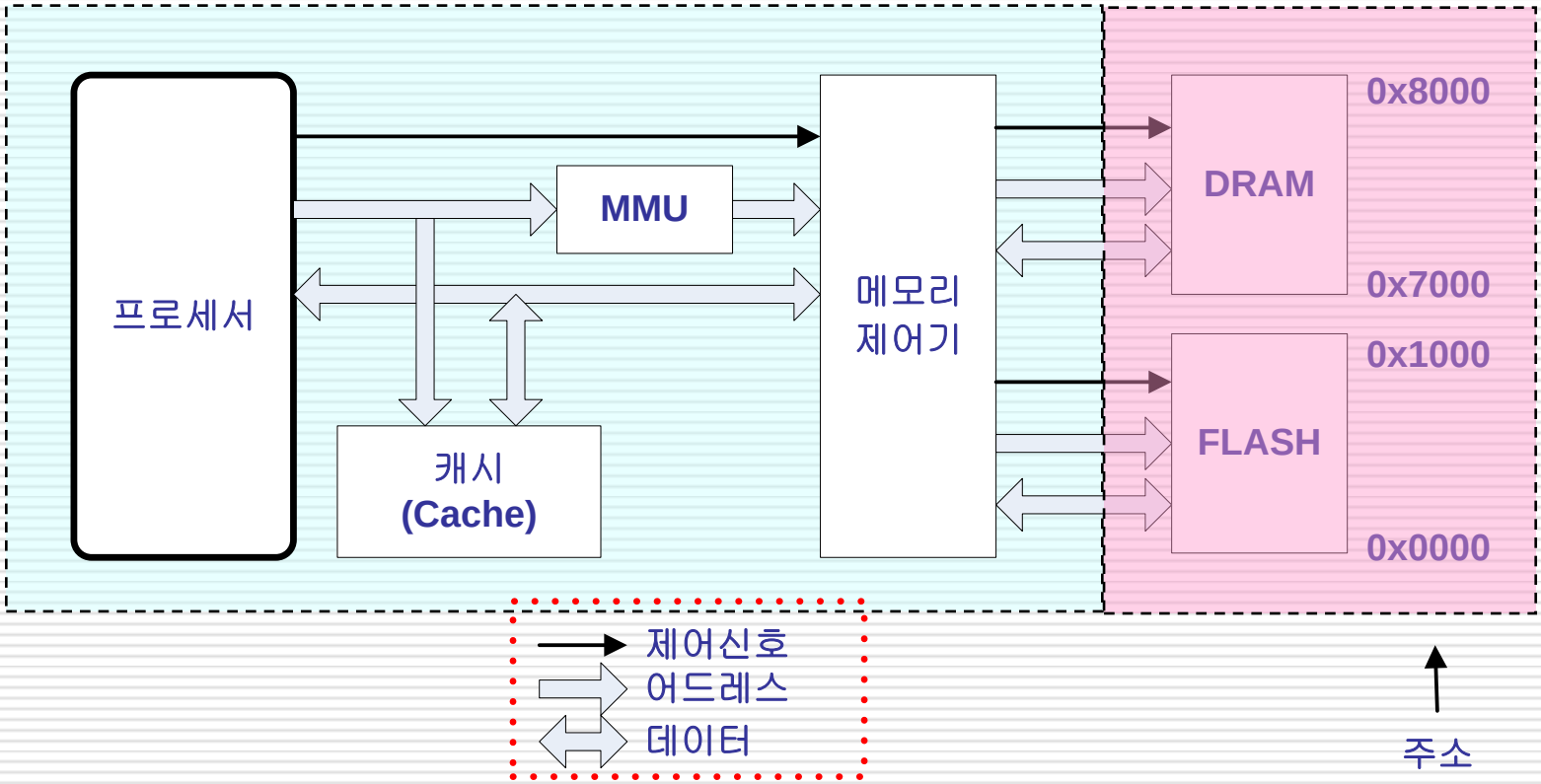


임베디드 OS 비교

- 모바일(Smart) OS
  - iOS (iOS3, iOS4, iOS5) vs Android (Proyo, Ginger Bread, Icecream Sandwitch)
- Embedded Linux
  - Open된 Linux 소스를 기반으로 각 임베디드시스템에 알맞게 포팅
  - 다양한 플랫폼, 다양한 개발환경 제공
  - 라이선스 무료로 점점 증가추세임
- Windows CE (WinCE), Window Embedded XP
  - Intel x86 계열 지원
  - 마이크로소프트사에서 각종 환경 지원(ActiveX, Win32 API등)
- VxWorks (Intel(예전 Windriver)의 상용 OS)
  - hard realtime, multi-thread, preemptive 특성을 갖는 상용 RTOS
  - 시장 점유율은 점점 떨어지는 추세
- 기타 OS : eCOS, freeOS 등 마이크로시스템용 OS도 다양함

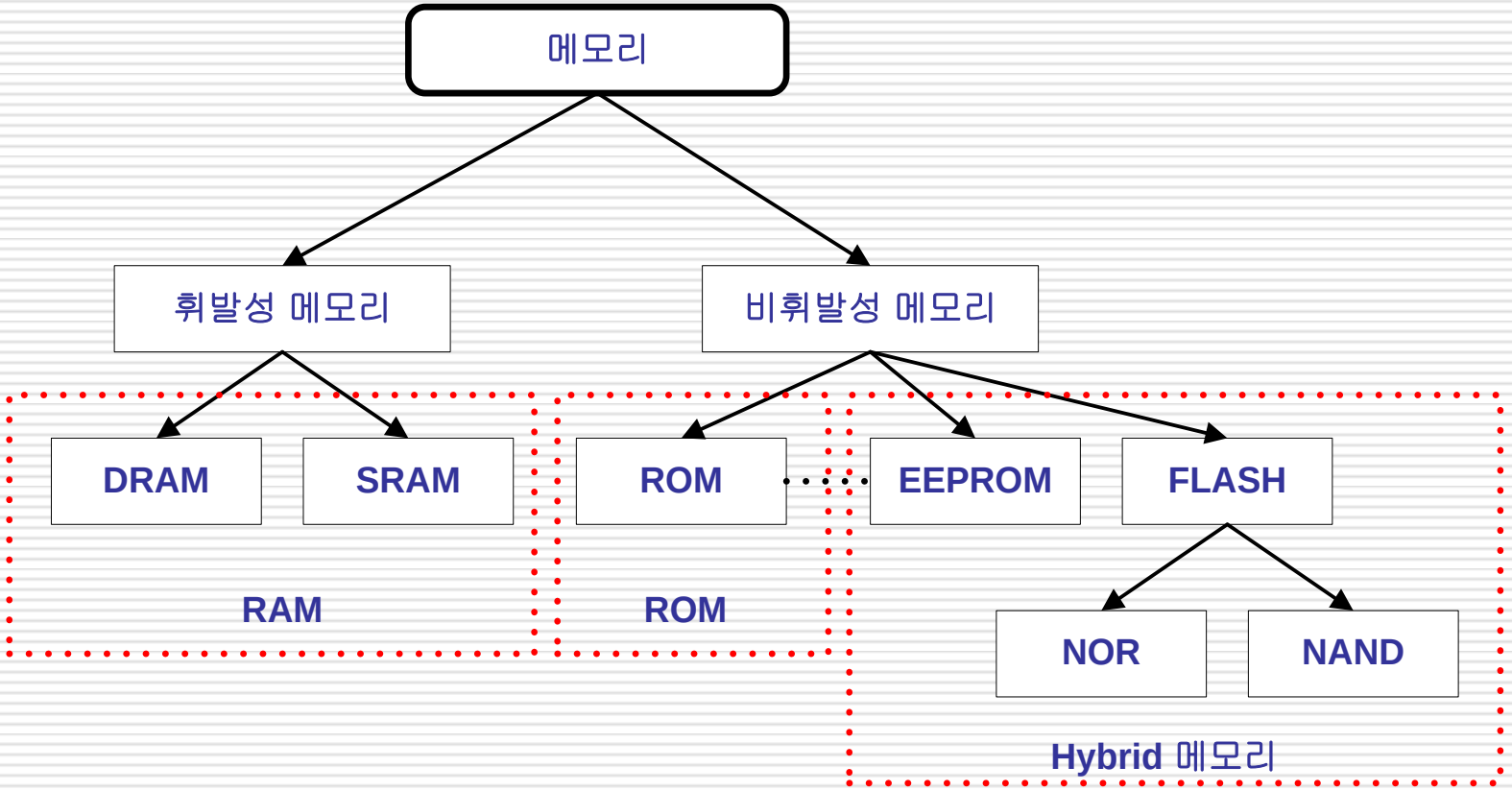
하드웨어 유니트 설계 및 구현

메모리 구조

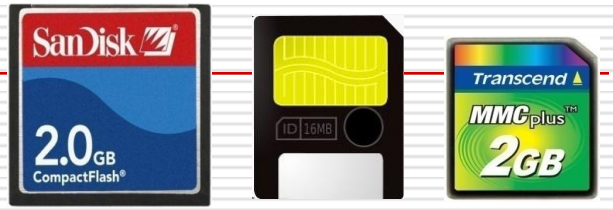


하드웨어 유니트 설계 및 구현

- 메모리 종류



하드웨어 유니트 설계 및 구현



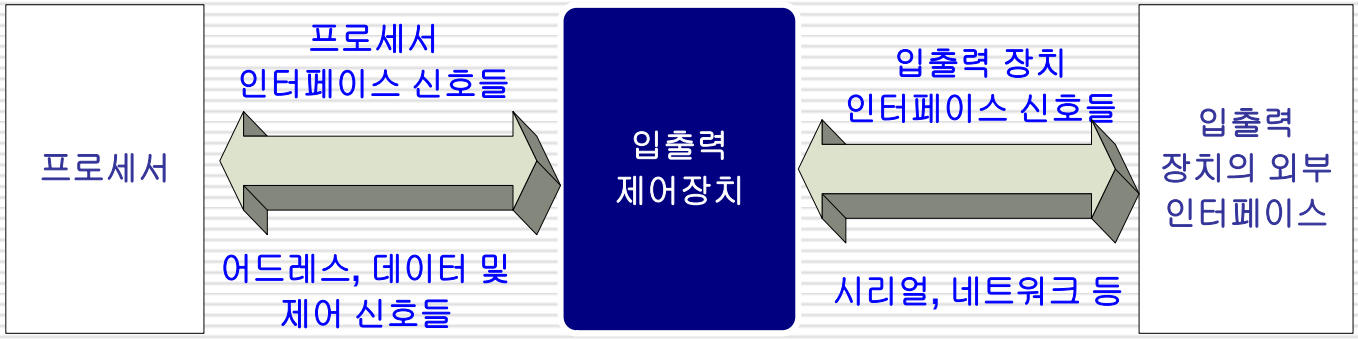
◻ 메모리 종류

구 분			속 도	가 격	용 도	특 징
휘발성 (Volatile Memory)	SRAM (Static)		수ns,고속	고가	캐시 등	집적도가 낮음
	DRAM (Dynamic)		수십 ns	저렴	주기억 장치	집적도가 높음
비휘발성(Non-Volatile Memory)	EEPROM (Electrically Erasable)		수십 ns	고가	소용량 데이터나 프로그램 저장용	NVRAM
	Flash	NAND	수십 ns	저렴	대용량 데이터 저장	블록 단위 읽기 쓰기
		NOR	수십 ns	고가	프로그램 저장 데이터 저장	

하드웨어 유니트 설계 및 구현

입출력 장치

- 프로세서와 외부를 연결하여 정보를 교환하는 장치
- 디지털 신호 또는 아날로그 신호를 포함
- 어드레스, 데이터 및 제어 신호를 통해서 연결됨
- 외부 인터페이스로는 Serial 연결이 주로 사용됨
- USART, USB, SPI, I2C(TWI), CAN, ADC, DAC, Ethernet 등



메모리 맵 방식과 I/O 맵 방식

구분	메모리 맵 방식	I/O 맵 방식
대표적인 프로세서	ARM, MIPS, PowerPC, M68K …	x86 계열
입출력 장치의 영역	메모리의 일부를 I/O 장치로 사용	메모리 영역과는 별도의 I/O 번지 영역이 존재
명령어	메모리와 I/O 장치 모두 메모리 동작 명령으로 억세스 하며, 각 영역의 구분은 어드레스로 한다.	메모리 억세스 명령과 I/O 억세스 명령(in/out)이 구분
하드웨어	어드레스를 해석하는 디코더 회로에 따라 메모리 혹은 I/O 장치가 선택	메모리 번지와 I/O 번지를 구분하는 신호가 존재.
주의 사항	- I/O 영역은 Non-cacheable로 설정해야 한다 - I/O 영역 변수는 volatile type으로 선언해야 한다.	

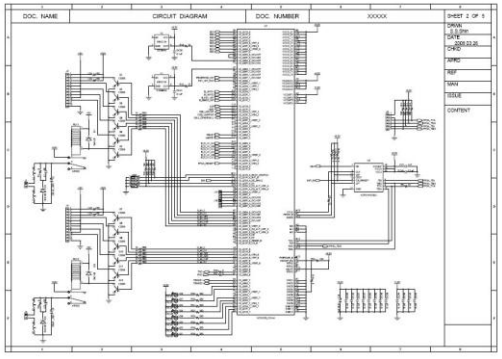
하드웨어 유니트 설계 및 구현

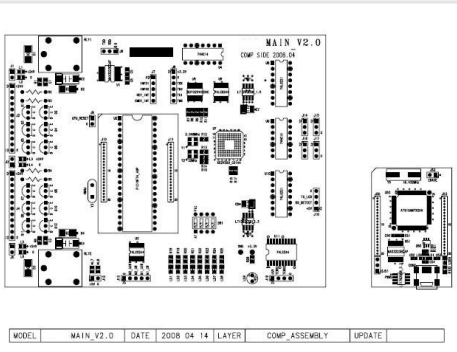
- 입출력장치의 자원 관리 방식
  - 폴링 (Polling) 방식
    - 한 프로그램이나 장치에서 다른 프로그램이나 장치들이 어떤 상태에 있는지를 순차적으로 검사하여 처리하는 방식
    - Sequential 형태이고, Priority가 없음
    - 단순하나 효율성이 떨어짐
    - OS가 없는 간단한 임베디드시스템에 사용
  - 인터럽트 (Interrupt) 방식
    - 한 프로그램이나 장치는 주어진 일을 수행하고, 필요시에 주변 장치에서 이를 인터럽트로 알려 처리하게 하는 방식
    - Multi-tasking 형태가 가능하며, Priority가 있음
    - 조금 복잡하나 효율성이 높음
    - OS가 있거나 조금 복잡한 임베디드시스템에 사용

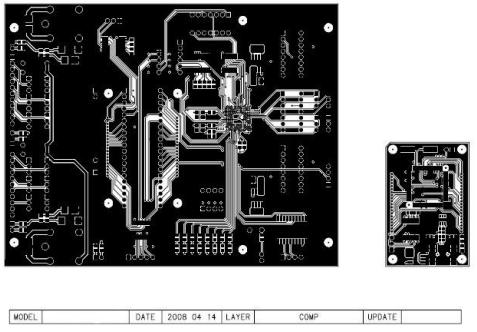
하드웨어 유니트 설계 및 구현

☐ 하드웨어 유니트 구현

- 설계된 내용을 OrCAD, Mentor 등의 CAD Tool을 이용하여 구현
- 회로도(Schematic) 작성
- 배치(Layout)
- 배선(Routing)
- PCB(Printed Circuit Board) 제작
- PBA(Printed Board Assembly) 조립







하드웨어 유니트 시험

- 하드웨어 유니트 시험
  - 전원, 클록, 리셋은 기본적으로 확인한 후 기능 시험 실시
  - 펌웨어를 다운로드 하여 마이크로콘트롤러를 동작시킨 후, 이후 기본적인 기능 시험을 위하여 가장 먼저 LED, FND, RS232C 등 눈에 보이는 기능을 먼저 셋업하고, 이후 차례로 기능을 시험
  - 문제 발생시 디버깅을 통하여 코드를 수정하고 재시험
  - 시험의 편의 등을 위하여 필요에 따라 BIOS(Basic Input Output System) 프로그램 또는 모니터프로그램을 작성하여야 할 필요가 있음

소프트웨어 유니트 설계 및 구현

- 임베디드시스템 소프트웨어 개발환경 셋업
  - IDE (Integrated Development Environment)
    - 컴파일러, 디버거 에디터 등이 통합된 CASE(Computer Aided Software Engineering) 툴
  - 교차 툴 체인 (Cross Tool Chain)
    - 개발된 소스 프로그램을 다른 machine의 기계어로 번역하는 프로그램 환경
    - 예 : GNU ARM Cross Tool Chain
  - ICE (In Circuit Emulator)
    - 호스트의 디버거와 함께 Target 시스템의 레지스터나 메모리의 내용을 읽거나 변경할 수 있고, Break Point나 Watch Point를 설정할 수도 있으며, 프로그램을 step-by-step으로 실행 할 수도 있게 해주는 장치
    - 보통 JTAG 인터페이스로 연결됨

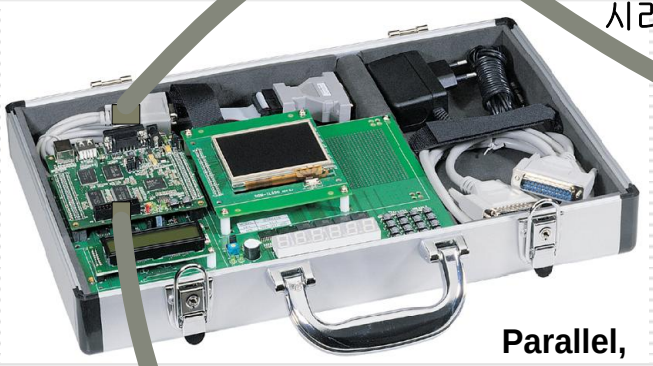
소프트웨어 유니트 설계 및 구현

- 임베디드시스템 소프트웨어 개발환경 셋업
  - IDE (Integrated Development Environment)
    - 컴파일러, 디버거 에디터 등이 통합된 CASE(Computer Aided Software Engineering) 툴
  - 교차 툴 체인 (Cross Tool Chain)
    - 개발된 소스 프로그램을 다른 machine의 기계어로 번역하는 프로그램 환경
    - 예 : GNU ARM Cross Tool Chain
  - ICE (In Circuit Emulator)
    - 호스트의 디버거와 함께 Target 시스템의 레지스터나 메모리의 내용을 읽거나 변경할 수 있고, Break Point나 Watch Point를 설정할 수도 있으며, 프로그램을 step-by-step으로 실행 할 수도 있게 해주는 장치
    - 보통 JTAG 인터페이스로 연결됨

임베디드시스템 소프트웨어 개발 환경

타겟 시스템
(AVR, Linux)

호스트 시스템
(Pentium, Linux)



시리얼 케이블

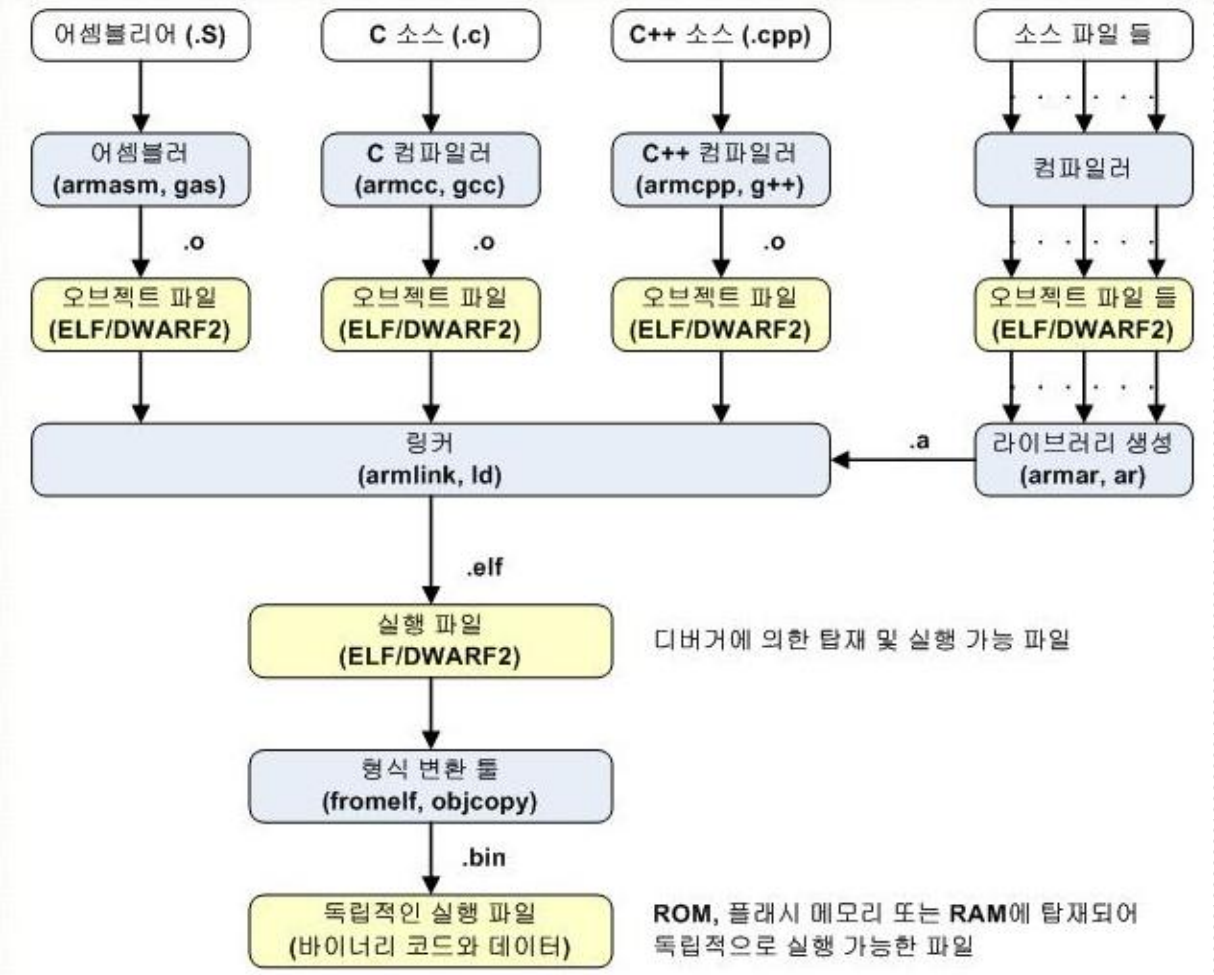
Parallel,
Ethernet, USB
케이블

JTAG 인터페이스

ICE

Cross-Development 환경

크로스툴체인 활용 : 예) GNU ARM의 경우



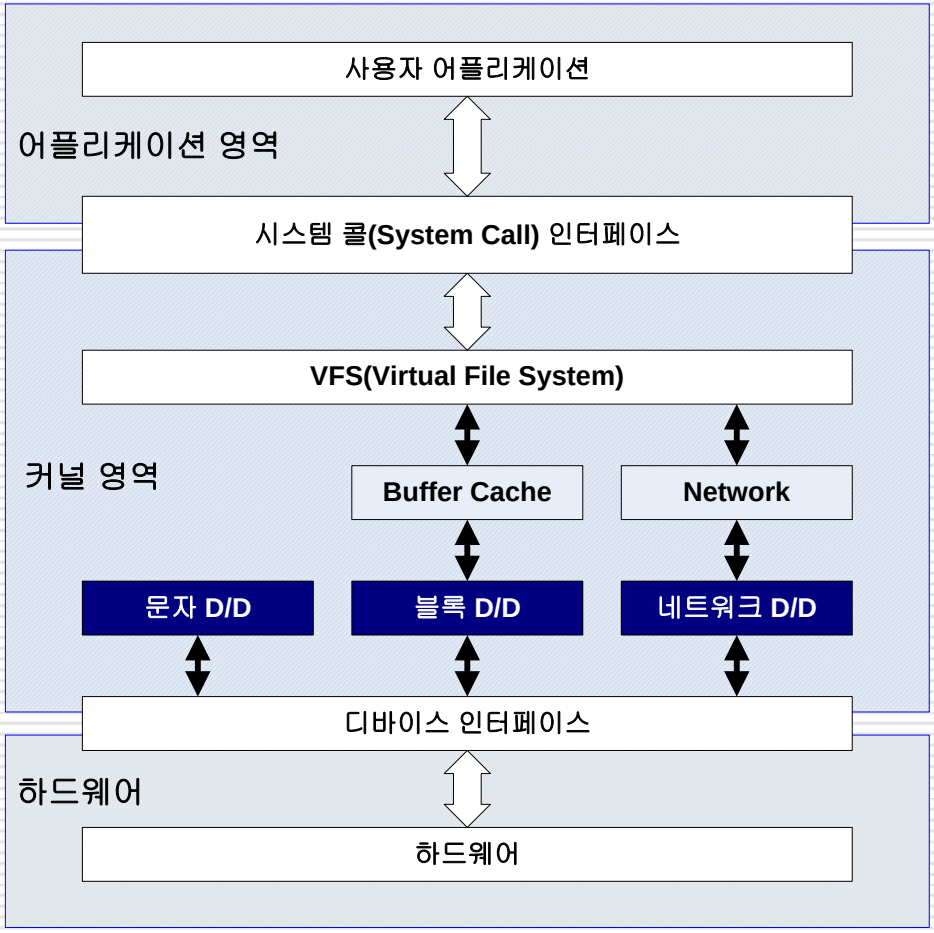
소프트웨어 유니트 설계 및 구현

- OS(커널) 포팅
  - 선정된 OS 커널이 개발하려는 시스템에서 동작 할 수 있도록 , 프로세서, 메모리, 트랩(Trap or Exception), 인터럽트와 타이머 등을 맞추어 주는 작업
  - OS의 Scheduler를 정상적으로 동작 할 수 있도록 함
  - 커널을 포팅하는 개발자는 타켓 시스템의 하드웨어 및 소프트웨어에 대하여 잘알고 있어야 함
  - 근래에는 반도체 또는 SoC 개발 회사에서 커널을 포팅하여 제공하는 추세

소프트웨어 유니트 설계 및 구현

- 디바이스 드라이버 설계
  - 디바이스
    - 네트워크 어댑터, LCD 디스플레이, Audio, 터미널, 키보드, 하드 디스크, 플로피디스크, 프린터 등과 같은 입출력 장치들을 말함
  - 디바이스 드라이버
    - 실제 장치 부분을 추상화 시켜 사용자 프로그램이 정형화된 인터페이스를 통해 디바이스를 접근할 수 있도록 해주는 프로그램
    - 디바이스 관리에 필요한 정형화된 인터페이스 구현에 요구되는 함수와 자료구조의 집합체
    - 응용프로그램이 하드웨어를 제어할 수 있도록 인터페이스 제공
    - 하드웨어와 독립적인 프로그램 작성을 가능하게 함

디바이스 드라이버 구조


사용자 어플리케이션
어플리케이션 영역
시스템 콜(System Call) 인터페이스
VFS(Virtual File System)
커널 영역
Buffer Cache
Network
문자 D/D
블록 D/D
네트워크 D/D
디바이스 인터페이스
하드웨어
하드웨어

소프트웨어 유니트 시험

- 응용 소프트웨어 로드 및 시험
  - 코딩이 완료되면 개발툴을 이용하여 응용 소프트웨어를 타켓에 로드하고 기능이 제데로 동작하는지 시험
  - 문제 발생시 디버깅을 통하여 코딩을 수정하고 재시험

시스템(통합) 시험

- <시스템설계서> 전 항목 만족 여부 확인 (checklist)
- <요구사항정의서> 전 항목 만족 여부 확인 (checklist)
- 기능 시험
  - White Box 테스트 : 개발자 테스트, verification
  - Black Box 테스트 : 사용자 테스트, validation
- 성능 시험
  - Benchmark 테스트
  - Sample Case 테스트
- 안정성 시험
  - Longrun 테스트
  - Burn-In 테스트
  - Field 테스트
- Beta 테스트, Alpha 테스트
- 공인규격, 공인시험기관 인증 시험

묻고 답하기

Q & A