Embedded System

Day-1

CH1.

* 프로세스: 컴퓨터에서 실행 중인 프로그램
* 스레드: CPU가 독립적으로 처리하는 작업의 단위
* 프로그램 생성시 프로세스 내부에 존재하는 수행 경로
  + 하나의 프로세스에서 여러 스레드 가능 (멀티스레딩)
* Bare-metal: 어떤 소프트웨어도 설치되어 있지 않은 하드웨어
* MCU(Micro Control Unit): 마이크로프로세서와 입출력 모듈이 하나의 칩으로 만들어져 정해진 기능을 수행하는 컴퓨터
* Microprocessor: ALU, Register 등의 연산 장치와 제어장치를 1개의 칩에 모아 놓은 장치
* MCU는 ALU, Register, Controller, Memory System등으로 이루어져 있다. Memory System은 외부 메모리에서 데이터를 가져오는 것을 의미한다. (Raspberry Pi같은 경우에는 MCU 내부에 있는 것 같다.)
* MCU에 외부 메모리에 주소를 주고 데이터를 주고받을 수 있는 실질적인 장치는 MCU에 붙어 있는 핀을 통한다. 핀은 Memory bus라고 생각해도 됨
* MCU의 비트 수는 Instruction이 다룰 수 있는 최대 비트 수
* SoC(System-on-Chip): 한 개의 칩 안에 완전 구동이 가능한 제품과 시스템이 들어있는 것

마이크로프로세서는 프로세서, 메모리, 통합 인터페이스의 다양성 때문에 실질적인 레퍼런스 아키텍쳐가 없다. 그럼에도 대부분의 마이크로 컨트롤러에는 **마이크로프로세서, RAM, 플래시 메모리, 직렬 트랜시버(transceiver)**[[1]](#footnote-1)는 포함되어 있다. 또한 더 많은 장치, 게이트웨이[[2]](#footnote-2)와 통신할 수 있는데, 일부 마이크로 컨트롤러는 임베디드 시스템의 제한(조건)을 충족시키기 위해 특정 프로토콜[[3]](#footnote-3)을 제공할 수 있다.

RAM, 플래시 메모리, 트랜시버의 제어 레지스터는 모두 같은 물리 공간에 Mapping되어 있어야 한다. Mapping 주소는 데이터 시트에 나와있다. 마이크로 컨트롤러는 실행중인 아키텍쳐에 적합한 바이너리 파일로 전달된 일련의 Instruction으로 코트를 실행할 수 있다.

프로세서[[4]](#footnote-4)는 보통 메모리의 0 또는 지정 주소에서 시작되어, Mapping된 내부 플래시 메모리 및 RAM에서 프로세서 특정 바이너리 형태로 저장된 Instruction을 실행할 목적으로 설계된다.

임베디드 마이크로 컨트롤러 환경에서 동작하는 소프트웨어를 컴파일하고 플래시 메모리에 로딩하려면 호스트 머신[[5]](#footnote-5)(?)이 필요하다. (최종 펌웨어는 보통 RAM보다 큰 플래시 메모리에 저장되고 전원 주입 및 재부팅 시에도 데이터가 유지된다. (RAM이 더 빠르긴 하다.) (RAM은 전원 꺼지면 지워진다.))

현재 모든 임베디드 시스템 플랫폼은 디버깅을 목적으로 JTAG 같은 매커니즘을 적어도 하나는 갖고 있고, 플래시 메모리에 설치한다. (JTAG란 CPU에서 제공하는 직렬 통신 방식이다. 시리얼보다는 빠르며, 부트로더를 [[6]](#footnote-6)올릴 때나, 프로그램 업로드, 데이터 교환시 사용된다.) 호스트 머신에서 디버깅 인터페이스에 접근하면 디버거가 장치와 상호작용이 가능해진다.

Day-2

임베디드 시스템의 제한 사항

1. 플래시 메모리가 구현하는데 필요 공간보다 작을 수 있다.
2. 구조체를 정의하는데 충분한 RAM이 없거나 대형 데이터 버퍼의 복사가 불가능 할 수 있다.
3. 프로세서가 데이터를 처리하기에 충분히 빠르지 않을 수 있다.
4. 더 적은 에너지를 소비해야 할 수도 있다.

MCU는 보통 MMU[[7]](#footnote-7)를 갖고 있지 않기 때문에 비휘발성 메모리가 부족하다. (커널, 애플리케이션, 라이브러리 저장을 위한) 따라서 임베디드 시스템은 특정 순서로 모든 데이터를 처리하고 통신하는 메인루프를 가진 단일 작업으로 이루어진다. 임베디드 시스템 설계는 부트 절차부터 애플리케이션 로직까지 전체 시스템을 맨 처음부터 구현해야한다.

단단한 Workflow[[8]](#footnote-8)는 중요하다. 여기엔 잘 정의된 테스트 사례, 주요 성능 지표 목록, 여러 도구와 절차, 효율적인 프로토 타입 단계가 포함된다.

* 멀티스레딩: 임베디드를 위한 MCU 사용의 장점은 time-sharing[[9]](#footnote-9)을 통해 별도의 실행 유닛에서 논리적으로 분리된 작업을 실행할 수 있다는 것이다.
* RAM: 임베디드 시스템에서는 각 주소 pool의 경계를 명확히 해야 하는데 잘못된 주소로 접근할 경우 전체 시스템이 망가질 수 있기 때문에 안전한 접근 방식이 필요하다.
* 플래시 메모리: 임베디드의 펌웨어는 보통 모든 소프트웨어 구성요소를 포함한 단일 바이너리 파일이어서, MCU 내부 플래시 메모리로 전송할 수 있다. 플래시 메모리는 메모리 영역의 고정 주소로 직접 매핑되기 때문에, 프로세서가 중간 단계없이 메모리에서 단일 Instruction을 순차적으로 가져와 사용할 수 있다. 이런 매커니즘을 XIP[[10]](#footnote-10)(execute in place)라고 한다.

마이크로 컨트롤러의 주소 영역에 Mapping된 내부 플래시 메모리는 쓰기를 위한 접근이 불가능하다. 내부 플래시 메모리의 변경은 블록 기반의 접근을 사용해야한다. (잘못 건들까봐 그러는 듯하다.) 쓰기를 위해 제조사들이 제공하는 블록 기반 플래시 메모리 접근 메커니즘을 IAP라 한다.

MCU 외부에 추가적인 비휘발성 장치가 있을 수 있다. Serial Peripheral Interface[[11]](#footnote-11)같은 특정 장치를 사용해 접근할 수 있다. 외부 플래시 메모리는 물리 주소 공간에 직접적인 Mapping을 허용하지 않으므로 펌웨어 저장에는 적합치 않다. 반면 쓰기 작업은 IAP보다 빠를 수 있다. 읽기 작업도 한번에 한 블록만 수행되기 때문에 이런 종류의 비휘발성 메모리는 설계에 따라 런타임에 검색된 데이터를 저장하는 데 이상적이다.

Day-3

* 인터페이스와 주변장치

마이크로 컨트롤러와 외부 핀은 일정 프로토콜을 이용하여 통신한다. 일반적인 인터페이스로는

1. 비동기 UART 기반 직렬 통신

Universal Asynchronous Receiver-Transmitter의 준말이다. 직렬 포트(Serial port) 종류의 인터페이스는 일반적으로 송신자 및 수신자를 동기화하기 위한 클럭 신호 공유가 필요치 않다. 하지만 사전에 정의된 클럭 속도로 작업하기에 비동기식[[12]](#footnote-12)이라 한다. 마이크로 컨트롤러는 여러 UART를 가질 수 있는데 비동기식 통신은 2개의 독립적인 와이어를 통해 각 종단점의 RX[[13]](#footnote-13)핀을 상대방의 TX[[14]](#footnote-14)핀에 연결해 전이중(양방향 통신이 가능한) 방식채널로 UART에 의해 제공된다.

두 종단점의 시스템은 같은 매개변수(와이어의 바이트 프레임, 프레임 속도 등등)를 사용하는 UART를 설정해야 한다. 또한 TTL-USB 직렬 변환기를 사용하면 호스트 머신의 콘솔에 UART를 쉽게 연결할 수 있다.

1. SPI(Serial Peripheral Interface) 버스

Serial Bus[[15]](#footnote-15) 중 하나다.

* + - 종단점(접속점) 동기화를 위한 직렬 클럭 라인
    - 마스터-슬레이브 프로토콜
    - 동일한 three-wire 버스를 통한 일대다 통신

마스터 장치인 마이크로 컨트롤러는 보통 하나 이상의 슬레이브 장치와 버스를 공유한다. MISO, MOSI, 클락선, SS(slave select)선, 총 네 개가 필요하다. 별도의 SS 신호가 버스에 연결된 각 슬레이브에 주소를 지정한는 데 사용된다. 버스는 단방향 데이터 전송을 위한 신호와 통신의 양단 간의 동기화를 위한 공유 클럭 라인을 위한 신호를 사용한다. 클럭 라인이 있기에 데이터의 전송은 더 안정적이고, 일반적인 UART보다 더 높은 비트 전송률을 가질 수 있다. SPI의 중요 요소는 슬레이브 설계에 요구되는 복잡성이 낮다는 것이다. I2C와의 차이점은 I2C는 무전기 같은 방식이고 SPI는 전화기 같은 방식이다.

SPI는 센서 장치, LCD 디스플레이, 플래시 메모리 컨트롤러 및 네트워크 인터페이스에 사용된다.

<https://www.kocoafab.cc/tutorial/view/101>

1. I2C (inter-integrated circuit) 버스

I2C는 여러 마이크로 컨트롤러와 여러 슬레이브가 동일한 two-wire버스에 연결하는 것을 염두에 두고 설계되었다. I2C는 SCL(serial clock)과 SDA(serial data) 두 가지 신호가 있다. SPI와 UART와는 달리 두 방향의 흐름이 같은 신호를 공유하기 때문에 반이중 버스다. (무전기 같은 듯) 프로토콜에 통합된 7 bits slave-addressing[[16]](#footnote-16) 매커니즘을 사용하여 슬레이브 선택에 추가적인 신호가 필요치 않다. 중재 로직에 따라 모든 마스터를 고려하기 때문에 여러 마스터에 같은 라인을 허용한다. 구조가 복잡하고 별도의 회로를 구성해줘야 하는 단점이 있다.

<https://namu.wiki/w/I2C>

1. USB (Universal Serial Bus)

호스트에 다양한 주변기기를 연결할 수 있게 해주는 버스 규격이다.

<https://engschool.tistory.com/entry/USB01?category=451977>

‘

Day-4

* 연결 시스템

주변 시스템과 인터넷 네트워크 또는 트래픽을 라우팅하는 게이트웨이를 통해 통신할 수 있다. IoT는 임베디드 장치들이 네트워크 프로토콜을 사용해 통신하기 위한 것이다. 즉, 임베디드 시스템에서 PC나 모바일과 같은 방식으로 네트워크의 프로토콜을 사용해 데이터를 교환하는 것이다. 인터넷 표준인 TCP/IP 프로토콜을 사용해 얻는 이점은 다른 기기와 통신할 때 라우팅 변환 매커니즘이 필요치 않다는 것이다. 하지만 TCP/IP 프로토콜은 복잡하고, 크기가 크며, 메모리 요구 사항이 높아 사용하기 힘들다. RAM과 플래시 메모리의 요구 사항이 다른 라이브러리와의 통합을 위해 설계단계에서 갱신되어야 한다.

1. 올바른 기술 및 프로토콜 설계
2. 비트 전송률, 패킷 크기 및 매체 접근의 제한 사항
3. 노드의 가용성
4. 토폴로지의 단일 장애 지점
5. 라우트의 설정
6. 관련 호스트의 인증
7. 매체를 통한 통신 기밀 유지
8. 네트워크 속도, 대기 시간 및 RAM 사용률에 대한 버퍼링의 영향
9. 프로토콜 스택 구현의 복잡성

* 레퍼런스 플랫폼

임베디드 시스템은 우선 RISC를 중심으로 한다. 마이크로 컨트롤러에 가이드라인인 몇 가지 레퍼런스가 있다.

1. 레지스터 및 주소에 사용되는 워드 크기(8, 16, 32, 64비트)
2. 인스트럭션 세트
3. 레지스터 환경설정
4. 엔디언[[17]](#footnote-17)(endianness)
5. 확장된 CPU 기능(인터럽트 컨트롤러, FPU[[18]](#footnote-18), MMU)
6. 캐시 전략
7. 파이프라인 설계

* ARM 레퍼런스 설계

임베디드 시장의 최고의 설계 공급자로, ARM Cortex-M 제품군이 가장 유명하다.

* Cortex-M 마이크로프로세서

Cortex-M 32bits 제품군의 특성

1. 16 범용 CPU 레지스터
2. 코드 밀도 최적화를 위한 thumb 16비트용 Instruction[[19]](#footnote-19)
3. 8 ~ 16 우선순위 수준을 갖는 내장 중첩 벡터 인터럽트 컨트롤러(NVIC[[20]](#footnote-20))
4. ARMv6 또는 ARMv7 아키텍쳐
5. 선택적 8구역 MPU(Memory Protection Unit)

1. 이더넷에 접속할 수 있게 해주는 기기 [↑](#footnote-ref-1)
2. 복수의 컴퓨터와 근거리 통신망(LAN)등을 상호 접속할 때 접속하는 장치(라우터와 비슷) 라우터는 서로 다른 네트워크를 연결해 주는 장치. 연결만이 아니라 효율적인 길도 알려준다. [↑](#footnote-ref-2)
3. 컴퓨터 간의 정보를 주고받을 때의 규칙(ex) TCP/IP) [↑](#footnote-ref-3)
4. 특정 기능, 처리를 하는 소프트웨어. 컴퓨터 시스템 전체를 하나의 프로세서라 부를 수도 있다. [↑](#footnote-ref-4)
5. 특정 하드웨어 모음과 소프트웨어 도구들이 있다. [↑](#footnote-ref-5)
6. CPU나 메모리가 처음 공장에서 나오면 기본 부팅이나 프로그램들을 받을 수 있도록 기본 프로그램들을 설치하는 과정 [↑](#footnote-ref-6)
7. 가상 주소와 물리 주소의 변환을 담당하는 장치다. 물리 주소: 컴퓨터의 메인 메모리에 접근할 때 사용되는 주소. 실제 메모리 공간과 같다. 가상 주소: 물리 주소만으로는 메인 메모리보다 큰 프로그램을 메모리에 적재할 수 없다. 또한 컴파일할 때 물리 주소를 할당하기에 동시에 여러 프로그램을 수행하기 힘들다. 따라서 각 프로그램에 실제 메모리 주소가 아닌 가상 메모리 주소를 주는 방식이 나타났다. [↑](#footnote-ref-7)
8. 일련의 업무 흐름을 의미한다. 일련의 엄무들을 정의하고 일정한 시간 내에서 자동적으로 이루어질 수 있도록 만든 소프트웨어다. [↑](#footnote-ref-8)
9. 하나의 장치에서 두개 이상의 실행을 시간을 쪼개어 상호 교환 배치시키도록 하는 기법 [↑](#footnote-ref-9)
10. RAM에 실행프로그램을 복사하는 것보단 비휘발성 메모리에 직접적으로 실행하는 방식이다. 필요한 총 메모리 양을 줄이기 위한 공유 메모리의 확장이다. 펌웨어에서 변경 불가능한 부분은 RAM에 로딩될 필요가 없고 direct addressing으로 접근한다. XIP를 지원하기 위해서는 몇 가지 조건을 충족시켜야 한다. <https://en.wikipedia.org/wiki/Execute_in_place> [↑](#footnote-ref-10)
11. 4선을 이용하여 주변장치와 연결하는 전이중(양방향 통신이 가능한) 동기식 인터페이스다. 2개의 데이터 회선과 2개의 제어 회선으로 되어있으며 주종관계 통신을 행한다. [↑](#footnote-ref-11)
12. 송신과 수신 측의 시점을 일치시키지는 않지만 블록을 기본 단위로 적당한 단위마다 동기를 이루는 것이다. 시작과 정지 신호로 동기를 이루는 방법과 일정한 크기로 동기를 이루는 방식이 있다. [↑](#footnote-ref-12)
13. Receive data 데이터 수신 [↑](#footnote-ref-13)
14. 데이터를 송신하는 핀 <https://blog.naver.com/yjc192/40009776202> [↑](#footnote-ref-14)
15. 연속적으로 채널이나 버스를 거쳐 한번에 하나의 비트를 단위로 데이터를 전송하는 과정 [↑](#footnote-ref-15)
16. 7개의 슬레이브 주소 비트와 1개의 read/write 비트로 이루어져 있다. <https://www.totalphase.com/support/articles/200349176#7bit> [↑](#footnote-ref-16)
17. 컴퓨터 메모리 같은 1차원 공간에 여러 개의 연속된 대상을 배열하는 방법. 큰 단위가 앞에 오는 빅-엔디언과 작은 단위가 앞에 오는 리틀-엔디언 등이 있다. [↑](#footnote-ref-17)
18. Floating Point Unit으로 부동 소수점을 계산하기 위한 장치다. [↑](#footnote-ref-18)
19. 자원의 효율적인 사용을 위해 32비트 대신 16비트 thumb 명령어를 사용한다. [↑](#footnote-ref-19)
20. 인터럽트를 관리하는 컨트롤러. 우선 순위 할당 등을 지원한다. [↑](#footnote-ref-20)