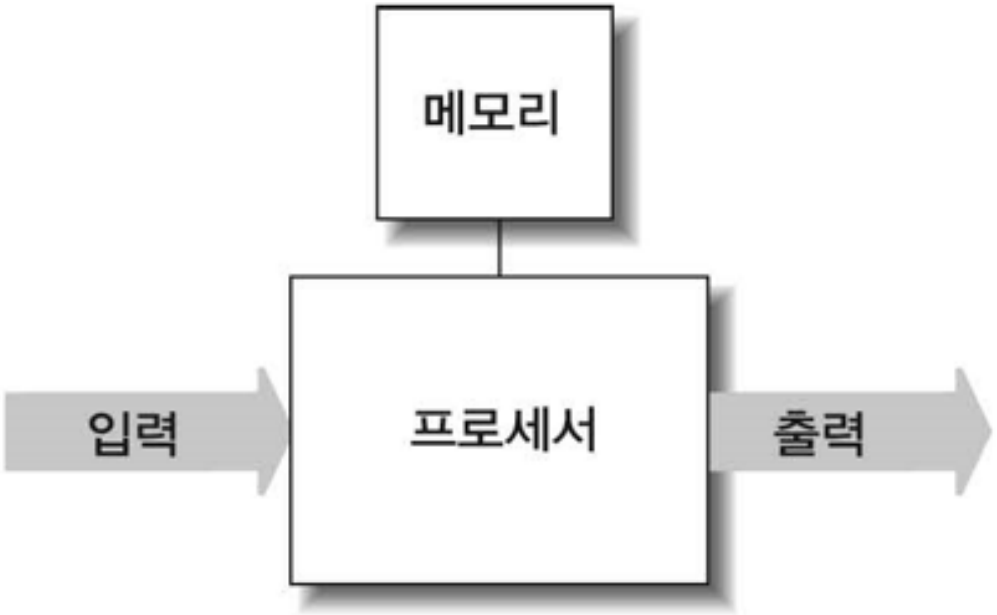
소프트웨어 개발자를 위한 임베디드 시스템 하드웨어

목차

- Embedded System 개요
- Embedded Processor 개요
- Memory
- BUS & Pipe-Line
- Interrupt & Exceptions
- 주변장치
- 부록: 시스템 개요

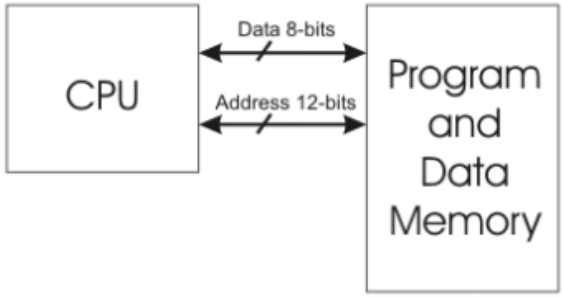
Embedded System 개요

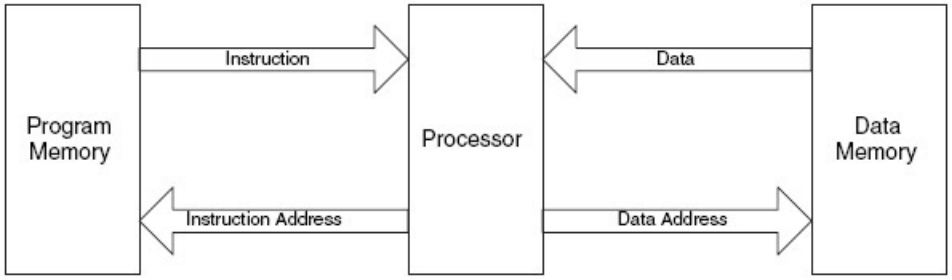
- Embedded System 이란
  - H/W와 S/W가 조합되어 **특정한 목적**을 수행하는 시스템
  - **큰 시스템의 구성요소**를 이루거나 사람의 개입없이 동작하는 H/W & S/W
  - PC는 Embedded System이라 하지 않는다.



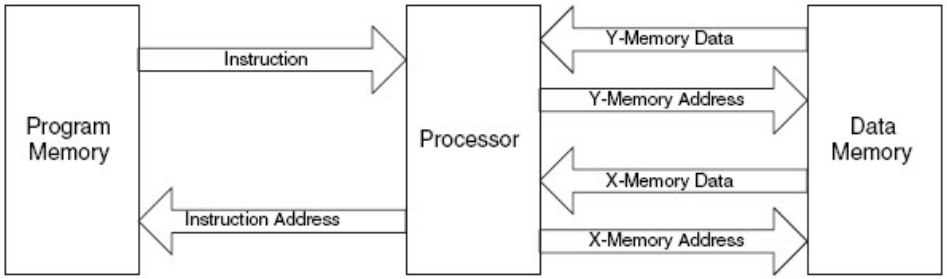
Embedded System 개요

- Von Neumann vs. Harvard Architecture





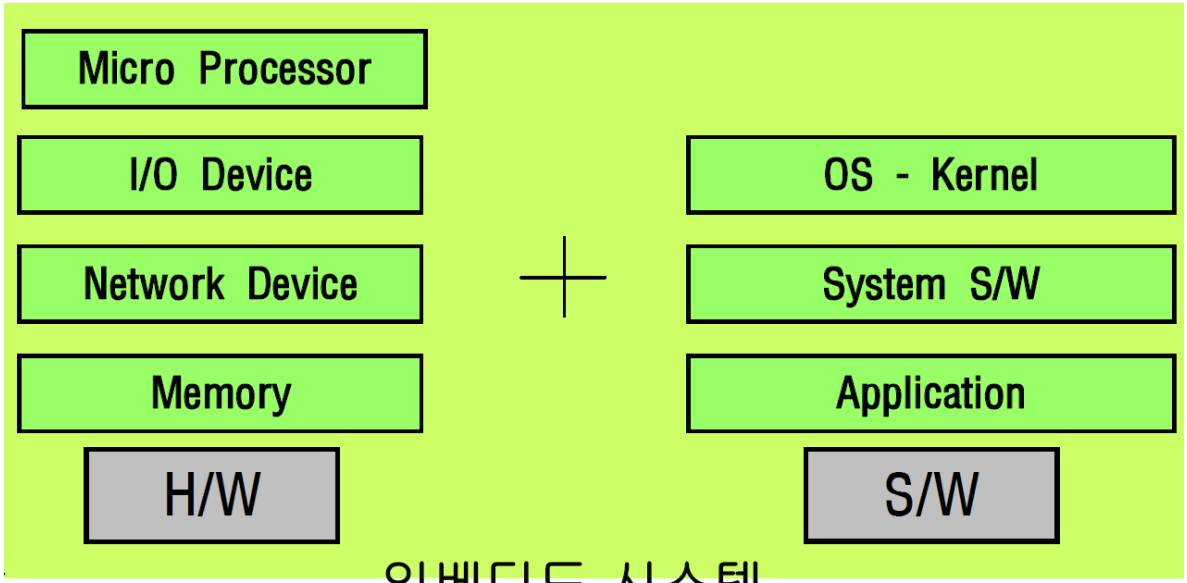
Pure Harvard Architecture



Modified Harvard Architecture

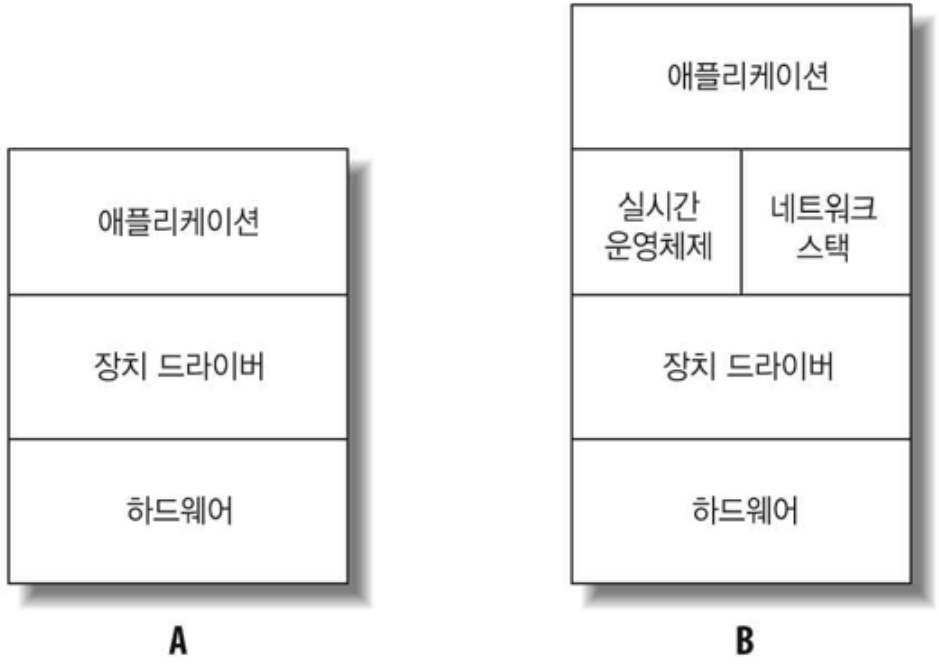
Embedded System 개요

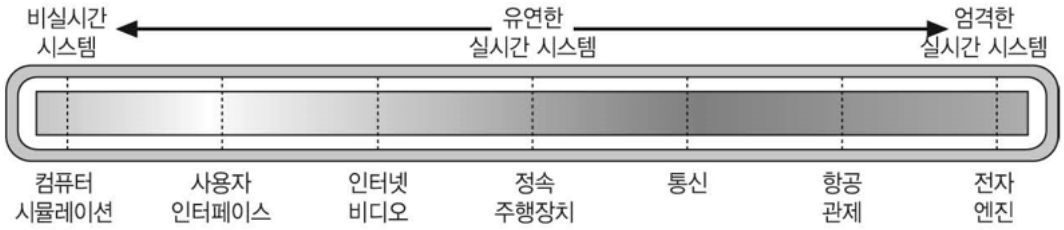
- 임베디드 시스템 구성
  - H/W
    - MCU / MPU, Memory, I/O, Network Device etc..
  - S/W
    - OS, Device Driver, Application FW...



Embedded System 개요

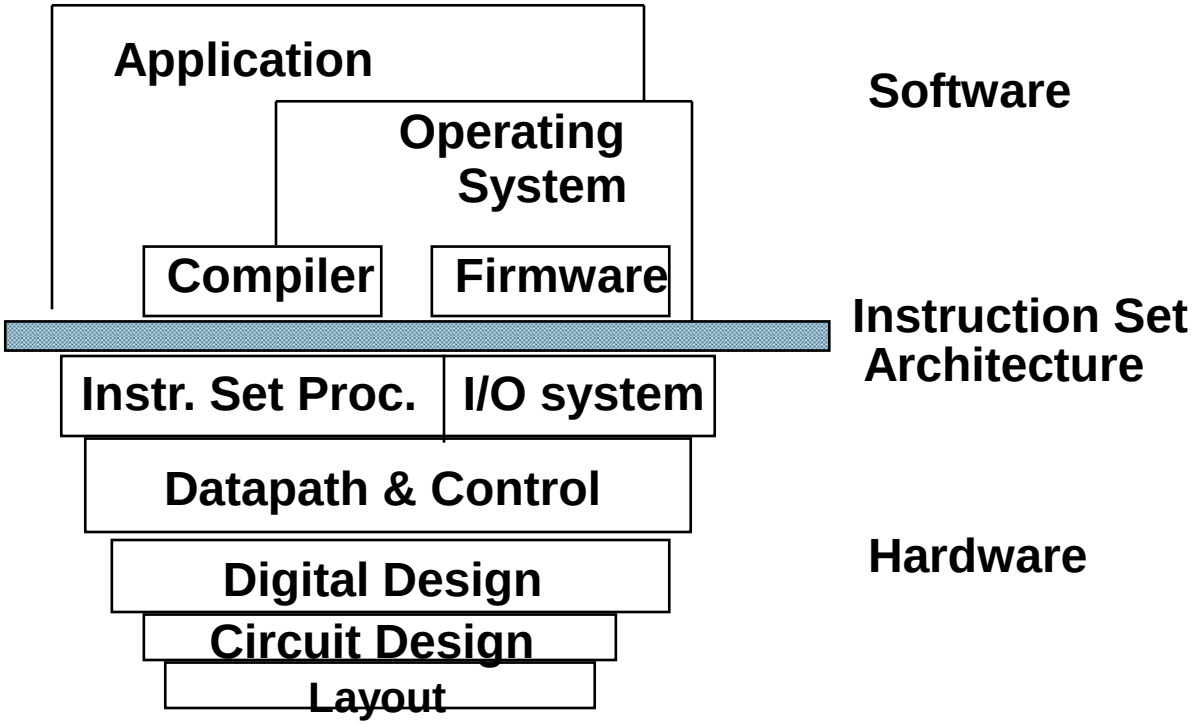
- 임베디드 S/W





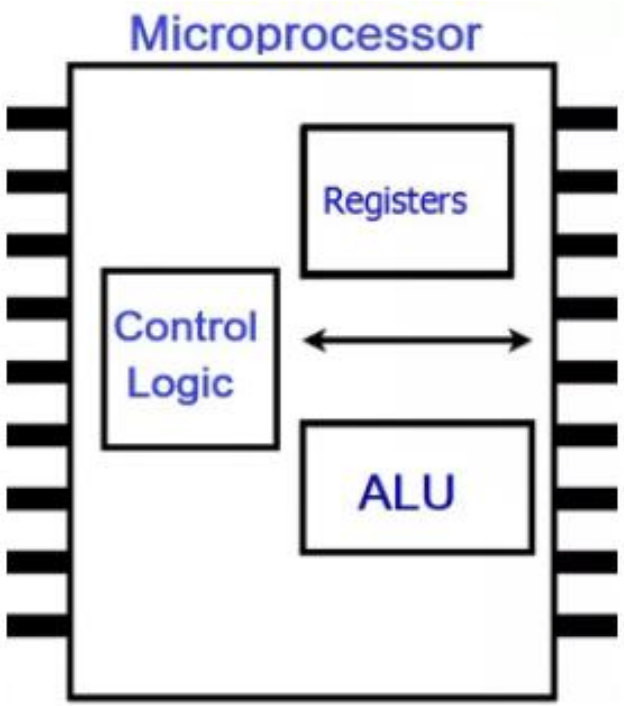
Embedded System 개요

- Computer Architecture 추상화



Embedded Processor 개요

- 기본구조
  - Datapath Unit : ALU, Register
    - Register를 이용하여 데이터 보관 및 저장
    - ALU등을 이용하여 Data 처리
  - Control Unit
    - 명령어 Operation Control
    - Datapath Uint Control



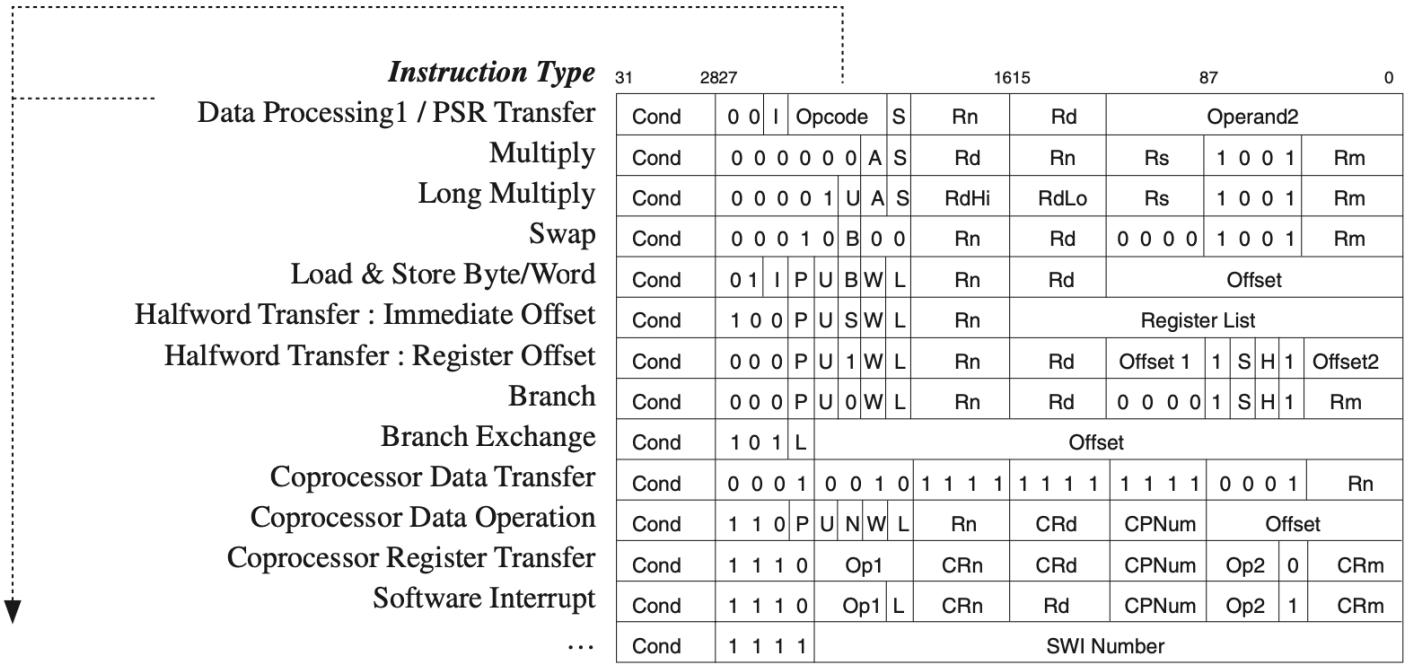
Embedded Process 개요

- ISA(Instruction Set Architecture)란?
  - S/W와 H/W 사이의 약속(Interface)
  - S/W와 H/W 사이를 소통하게 해주는 명령어들의 집합

- ISA 명령어
  - 종류
    - 산술 / 논리 명령어(add, Sub)
    - 데이터 전송 명령(load, store)
    - 흐름 제어 명령어(branch, call, trap)
    - 부동소수점 연산 명령어
  - 구성 : **Opcode + Operands**
  - Address Mode
    - Register : 레지스터에 저장된 값을 접근
    - Immediate : 명령어 자체에서 상수 값을 읽어 올때
    - Direct : 고정된 주소의 메모리 접근
    - Register Indirect : 레지스터에 있는 주소의 모리 접근
    - Displacement : 레지스터에 있는 주소에 일정 거리를 더한 주소의 메로리 접근
    - Index : 레지스터에 있는 주소에 다른 레지스터 값에 해당하는 거리를 더한 주소의 메모리를 접근할 때

Embedded Process 개요

- ISA 예제



Instruction Type	31	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data Processing1 / PSR Transfer	Cond		0	0	I	Opcode					S	Rn				Rd				Operand2										
Multiply	Cond		0	0	0	0	0	0	A	S	Rd				Rn				Rs				1	0	0	1	Rm			
Long Multiply	Cond		0	0	0	0	1	U	A	S	RdHi				RdLo				Rs				1	0	0	1	Rm			
Swap	Cond		0	0	0	1	0	B	0	0	Rn				Rd				0	0	0	0	1	0	0	1	Rm			
Load & Store Byte/Word	Cond		0	1	I	P	U	B	W	L	Rn				Rd				Offset											
Halfword Transfer : Immediate Offset	Cond		1	0	0	P	U	S	W	L	Rn				Register List															
Halfword Transfer : Register Offset	Cond		0	0	0	P	U	1	W	L	Rn				Rd				Offset 1				1	S	H	1	Offset2			
Branch	Cond		0	0	0	P	U	0	W	L	Rn				Rd				0	0	0	0	1	S	H	1	Rm			
Branch Exchange	Cond		1	0	1	L	Offset																							
Coprocessor Data Transfer	Cond		0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn			
Coprocessor Data Operation	Cond		1	1	0	P	U	N	W	L	Rn				CRd				CPNum				Offset							
Coprocessor Register Transfer	Cond		1	1	1	0	Op1				CRn				CRd				CPNum				Op2		0	CRm				
Software Interrupt	Cond		1	1	1	0	Op1			L	CRn				Rd				CPNum				Op2		1	CRm				
…	Cond		1	1	1	1	SWI Number																							

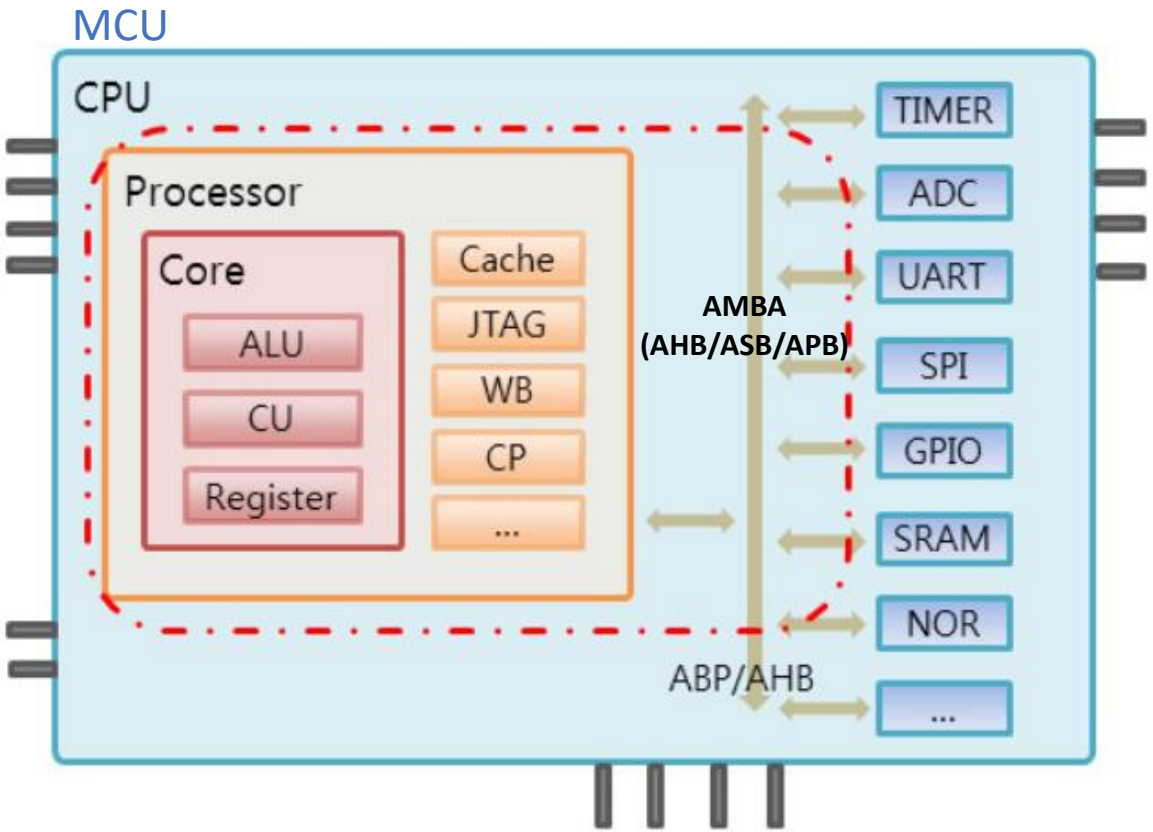
1 - Data Processing OpCodes

0000 = AND – Rd: = Op1 AND Op2
0001 = EOR – Rd: = Op1 EOR Op2
0010 = SUR – Rd: = Op1 – Op2
0011 = RSB – Rd: = Op2 – Op1
0100 = ADD – Rd: = Op1 + Op2
0101 = ADC – Rd: = Op1 + Op2 + C
0110 = SEC – Rd: = Op2 – Op1 + C –1
0111 = RSC – Rd: = Op2 – Op1 + C – 1
1000 = TST – set condition codes on Op1 AND Op2
1001 = TEQ – set condition codes on Op1 EOR Op2
1010 = CMP – set condition codes on Op1 – Op2
1011 = CMN – set condition codes on Op1 + Op2
1100 = ORR – Rd: = Op1 OR Op2
1101 = MOV – Rd: = Op2
1110 = BIC – Rd: = Op1 AND NOT Op2
1111 = MVN – Rd: = NOT Op2

ARM ISA Encoding

Embedded Processor 개요

- MCU : Micro Controller Unit
- MPU : Micro Process Unit
- DSP : Digital Signal Processor → MCU의 특화 Version

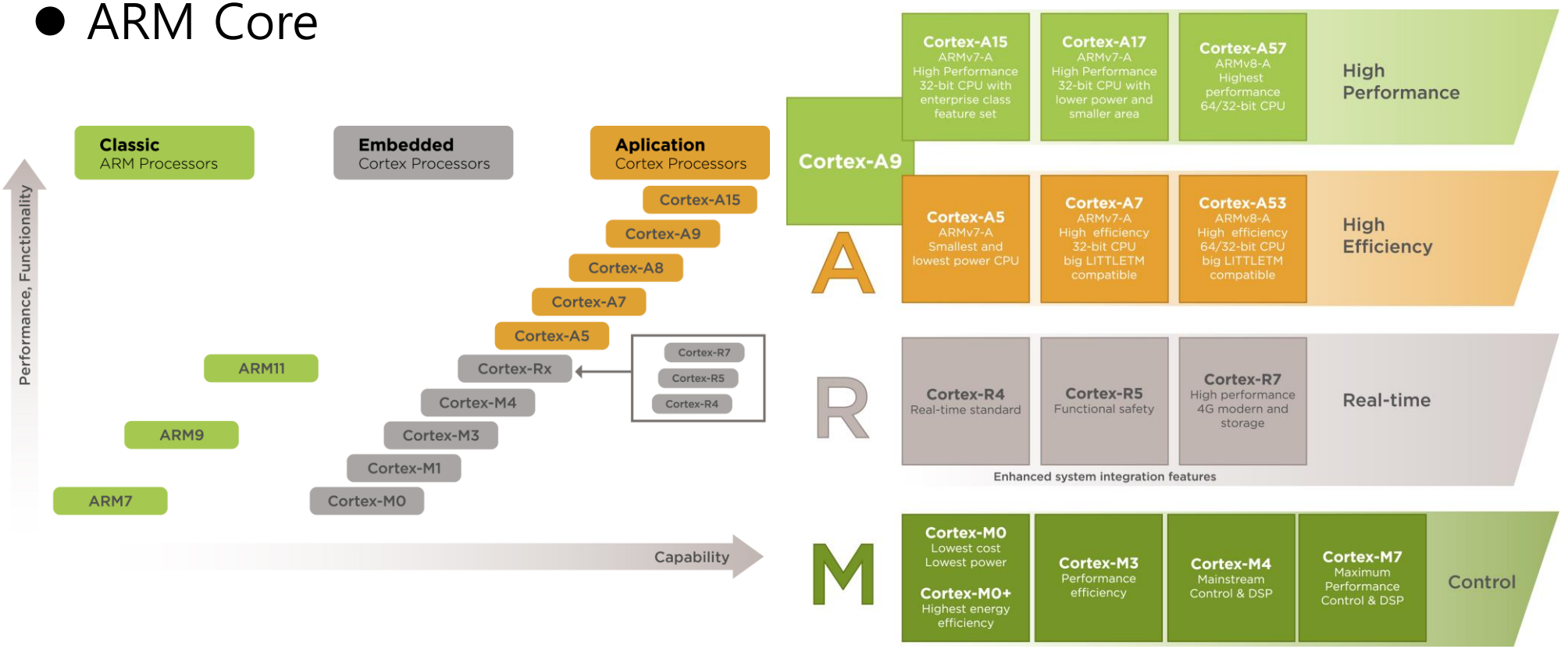


Embedded Processor 개요

	마이크로 프로세서	마이크로 컨트롤러
	마이크로 프로세서는 컴퓨터 시스템의 핵심 역할을합니다.	마이크로 컨트롤러는 임베디드 시스템의 핵심 역할을합니다.
	메모리와 I / O 출력 컴포넌트가 외부로 연결된 프로세서입니다.	메모리 및 I / O 출력 구성 요소가 내부에 존재하는 제어 장치입니다.
	메모리와 I / O 출력은 외부에서 연결되어야합니다. 따라서 회로가 더 복잡합니다.	온칩 메모리 및 I / O 출력 구성 요소를 사용할 수 있기 때문입니다. 따라서 회로는 덜 복잡합니다.
	컴팩트 시스템에서는 사용할 수 없습니다. 따라서 마이크로 프로세서는 비효율적입니다.	컴팩트 한 시스템에서 사용할 수 있습니다. 따라서 마이크로 컨트롤러가 더 효율적입니다.
	마이크로 프로세서에는 더 적은 레지스터가 있습니다. 따라서 대부분의 작업은 메모리 기반입니다.	마이크로 컨트롤러에는 더 많은 레지스터가 있습니다. 따라서 프로그램을 작성하는 것이 더 쉽습니다.
	상태 플래그가 XNUMX 인 마이크로 프로세서.	마이크로 컨트롤러에는 제로 플래그가 없습니다.
	주로 개인용 컴퓨터에 사용됩니다.	주로 세탁기, 에어컨 등에 사용됩니다.

Embedded Processor 개요

- ARM Core



Embedded Processor 개요

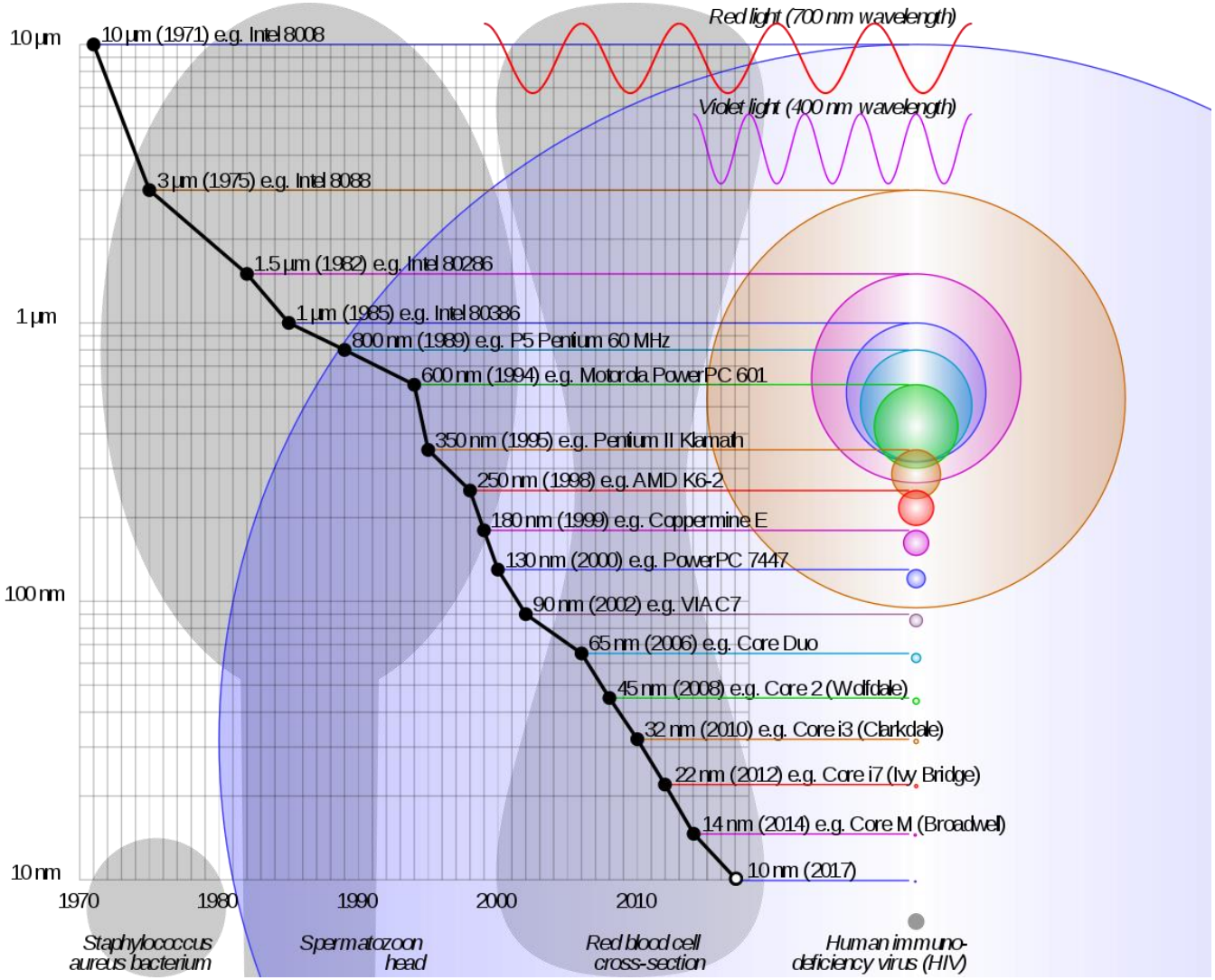
- 데이터 버스의 크기별 / ALU 및 Register의 크기별 분류
  - 8-bit
  - 16-bit
  - 32-bit
  - 64-bit

- 명령어 구조에 따른 분류
  - CISC(Complex Instruction Set Computer)
  - RISC(Reduced Instruction Set Computer)
  - EISC(Extended Instruction Set Computer)

Embedded Processor 개요

구분	CISC	RISC	EISC
CPU instruction	명령어 개수가 많고, 그 길이가 다양하며 실행사이클도 명령어 마다 다름	명령어길이는 고정적이며, 워드와 데이터 버스 크기가 모두 동일, 실행 사이클도 모두 동일	16비트 명령을 사용하여 32비트 데이터를 처리
회로 구성	복잡	단순	단순
메모리 사용	높은 밀도 메모리 사용이 효율적	낮은 밀도의 명령어 사용으로 메모리 사용이 비효율적	코드밀도가 높다(임베디드 시스템에 유리)
프로그램측면	명령어를 적게 사용	상대적으로 많은 명령어가 필요, 파이프라인 사용	RISC보다 더 깊은 파이프 라인 스테이지
컴파일러	다양한 명령을 사용하므로 컴파일러가 복잡해짐	명령어 개수가 적어서 단순한 컴파일러 구현 가능	국산 기술에 의해 개발(ADC corp.)

인텔 마이크로프로세서 역사

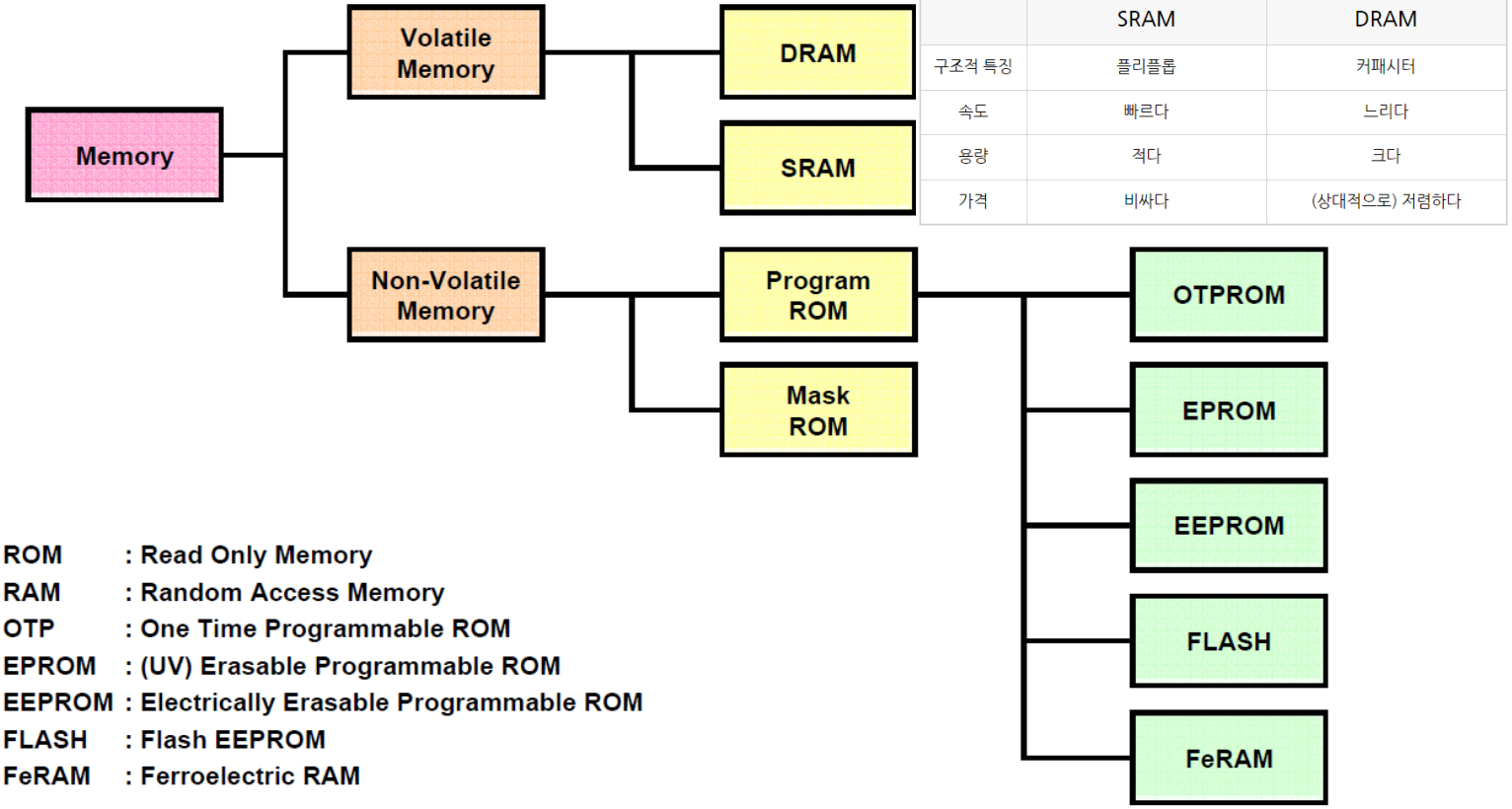


ACE Lab.

KNU

Memory

- Memory Type

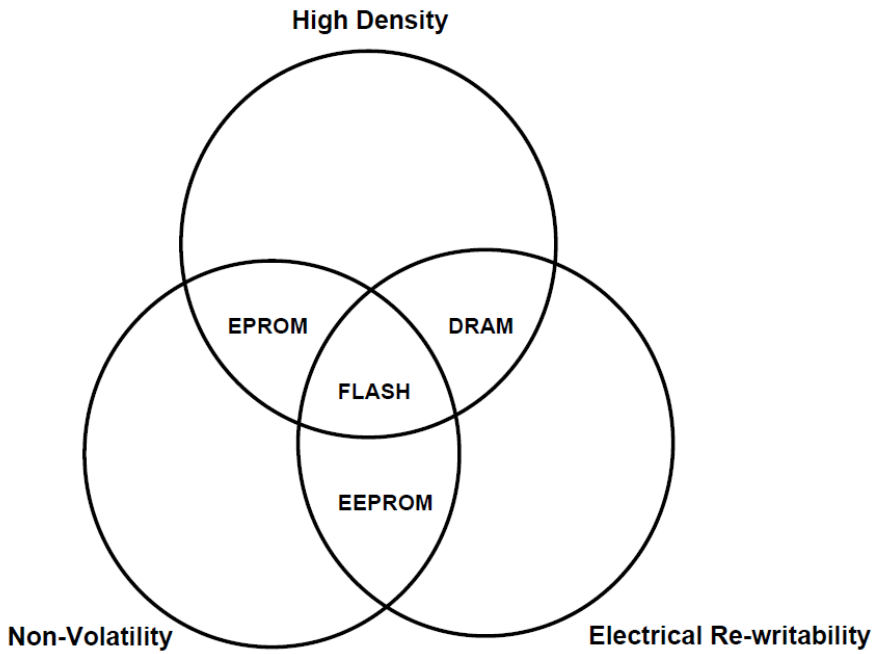


	SRAM	DRAM
구조적 특징	플리플롭	커패시터
속도	빠르다	느리다
용량	적다	크다
가격	비싸다	(상대적으로) 저렴하다

- ROM : Read Only Memory
- RAM : Random Access Memory
- OTP : One Time Programmable ROM
- EPROM : (UV) Erasable Programmable ROM
- EEPROM : Electrically Erasable Programmable ROM
- FLASH : Flash EEPROM
- FeRAM : Ferroelectric RAM

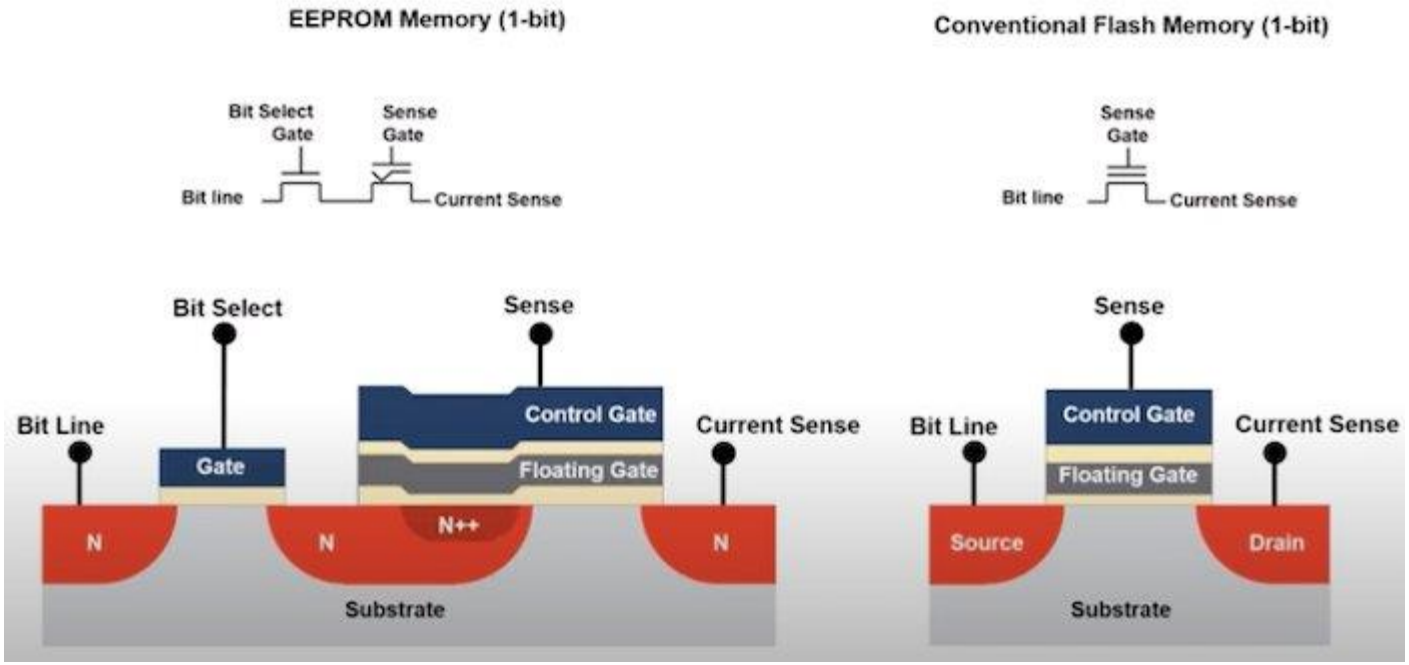
Memory

- Memory Type2



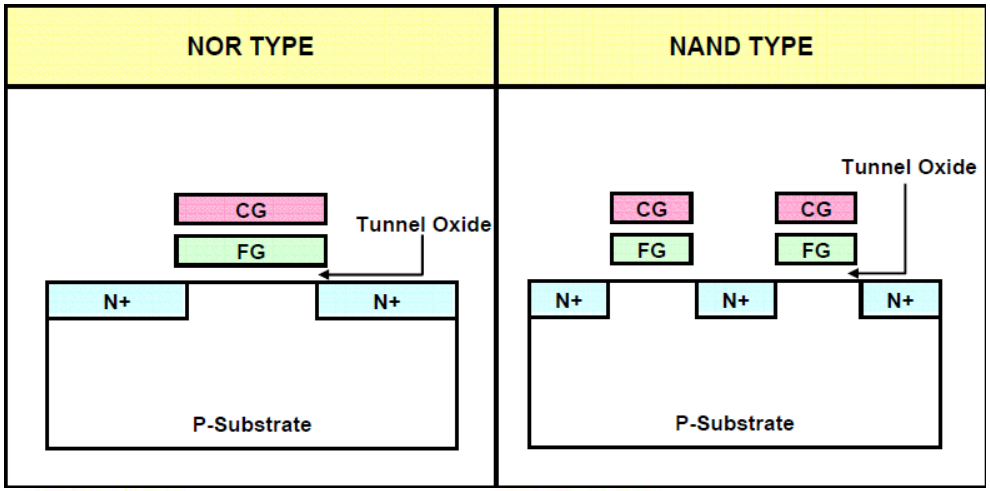
Memory

- EEPROM vs NOR Flash



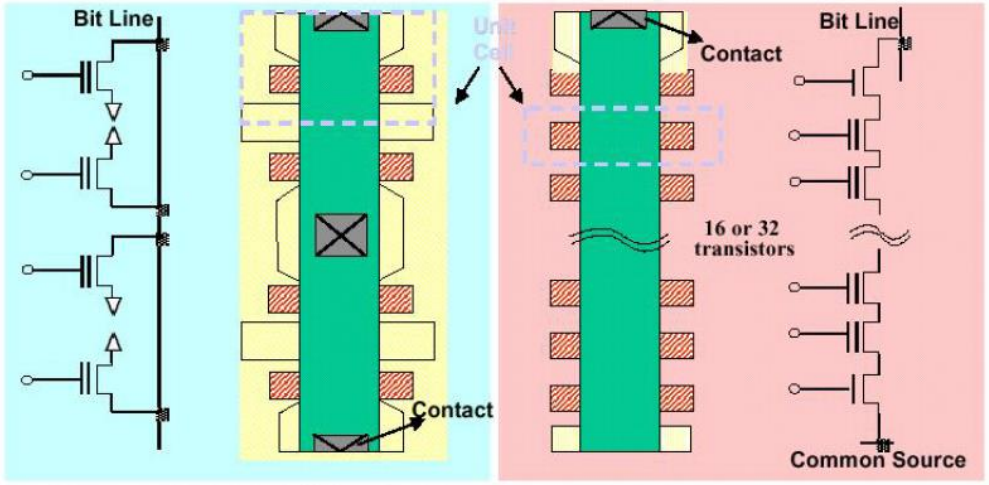
Memory

- NOR Flash vs NAND Flash 구조



• Large cell and fast random access

• Small cell, but fast sequential access



• Contact is the limiting factor for scale-down.

• Easy to Scale Down.

Memory

- NOR Flash vs NAND Flash 장단점

Merits of NOR

①High speed random access

②Byte programming

Merits of NAND

①High speed programming

②High speed erasing

③Small block size

Demerits of NOR

①Slow programming

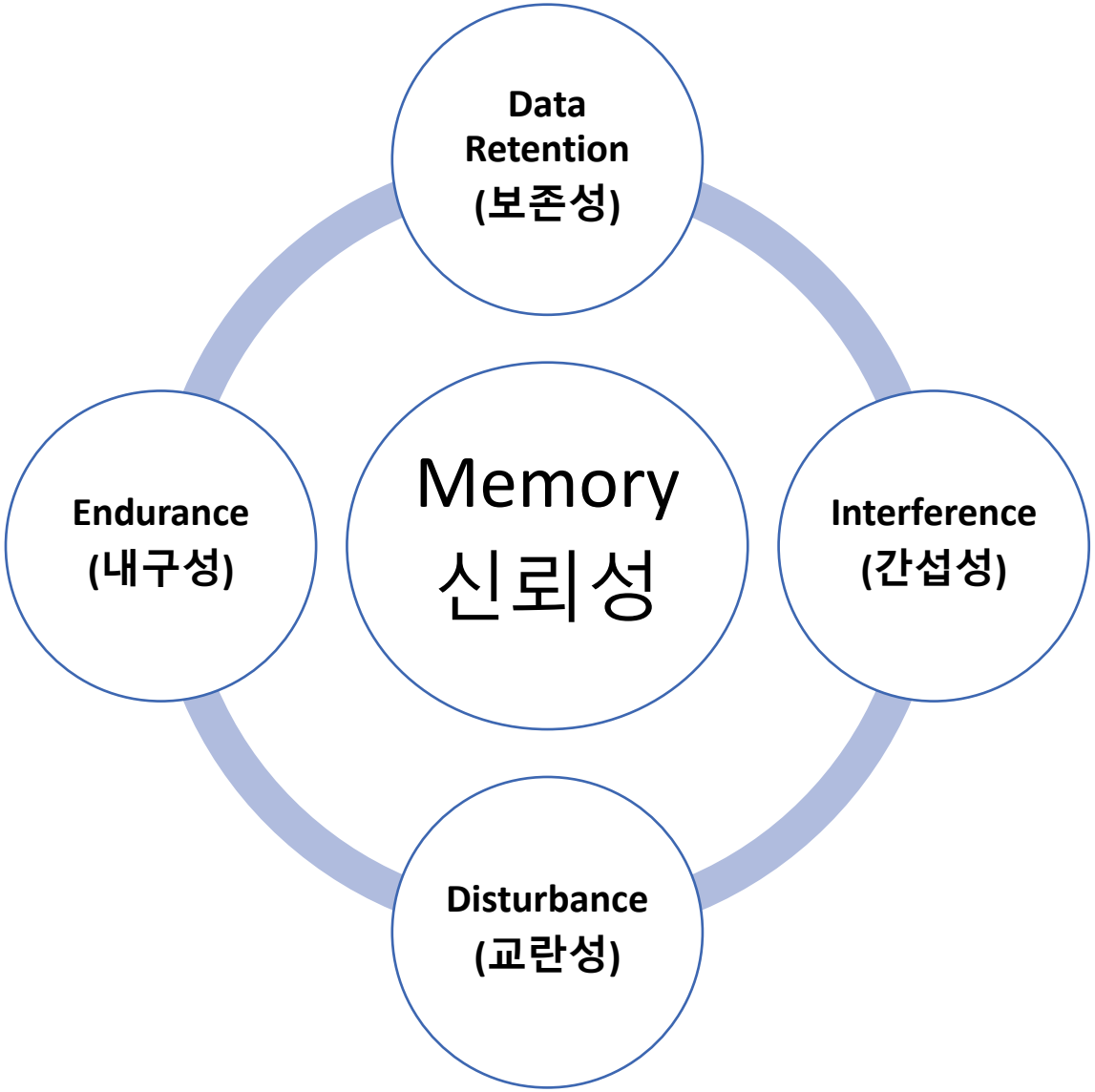
②Slow speed erasing

Demerits of NAND

①Slow random access

②Byte programming can not be performed

Memory


Data
Retention
(보존성)
Endurance
(내구성)
Memory
신뢰성
Interference
(간섭성)
Disturbance
(교란성)

Memory

- Data Retention
  - Floating Gate Data를 얼마나 오랫동안 저장이 가능한지에 대한 신뢰성요소
  - 온도에 반비례하고, Flash Cell 면적에 비례

- Endurance
  - Erase & Writing 반복 가능한 횟수를 나타내는 신뢰성 요소
  - Erase와 Writing 반복시 FG의 VT값이 shift됨.
  - Over Erase나 Over PGM을 통한 Damage에 의해 급속한 회수 제한이 생길 수도 있다.

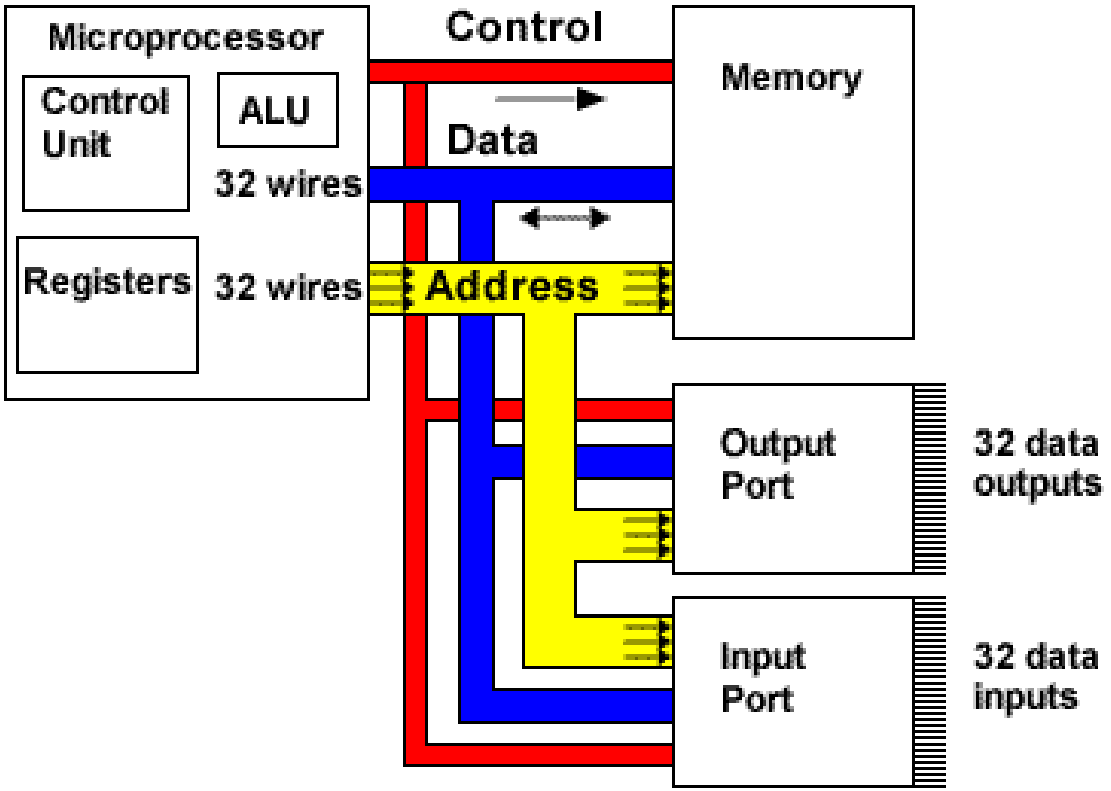
Memory

- Disturbance
  - 전기적 Stress에 의해 정상 Data가 변동되는 현상
    0 → 1, 1 → 0
  - Drain Disturb / Gate Disturb / Read Disturb

- Interference
  - 주변의 Cell의 일시적 또는 장시간 전자 유입으로 인해 정상 Data가 변동되는 현상
  - Wafer 제작과정에서 미세 particle등에 의해 발생하며,
    특별한 원인 규명이 되지 않는 경우가 많음.

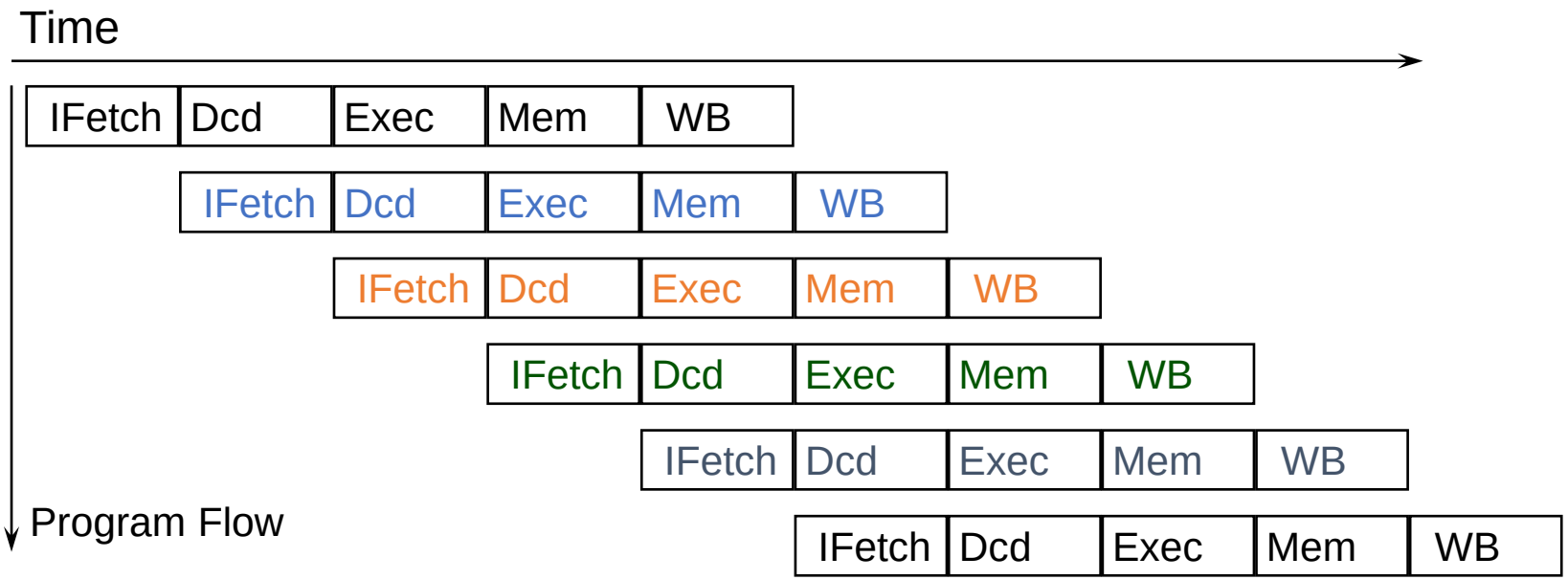
BUS & Pipe-Line

- BUS
  - Digital 회로에서 동일한 기능을 수행하는 선들의 집합.
  - Adddress Bus
  - Data bus
  - Control Bus



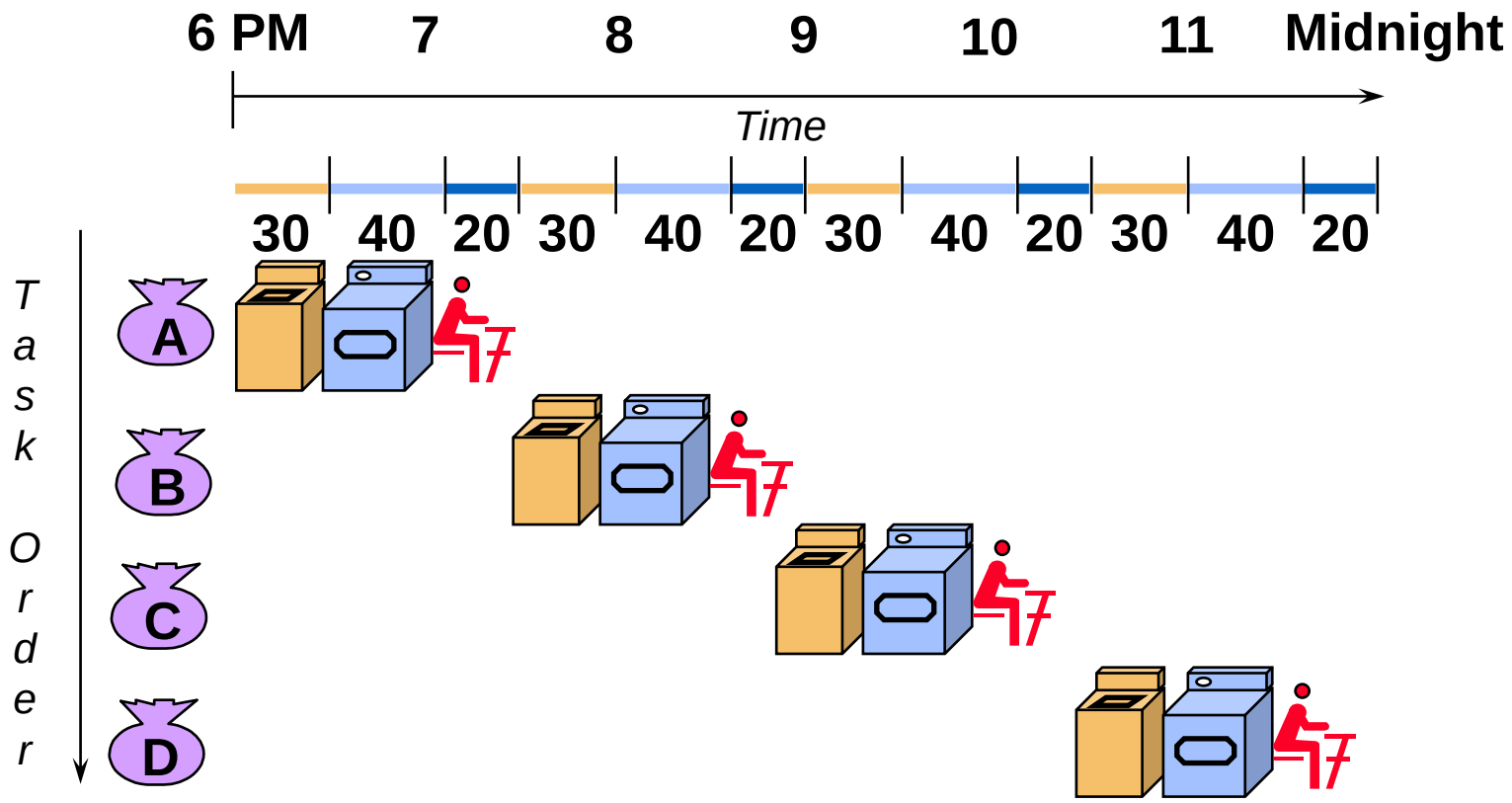
BUS & Pipe-Line

- Pipe-Line
  - 하나의 Process를 서로다른 기능을 가진 여러 개의 Sub-Process로 나눔.
  - 각 Sub-Process가 동시에 서로 다른 데이터를 취급하도록 하는 기법.
  - 1 Clock에 동시에 명령어 처리를 가능하게 하는 구조



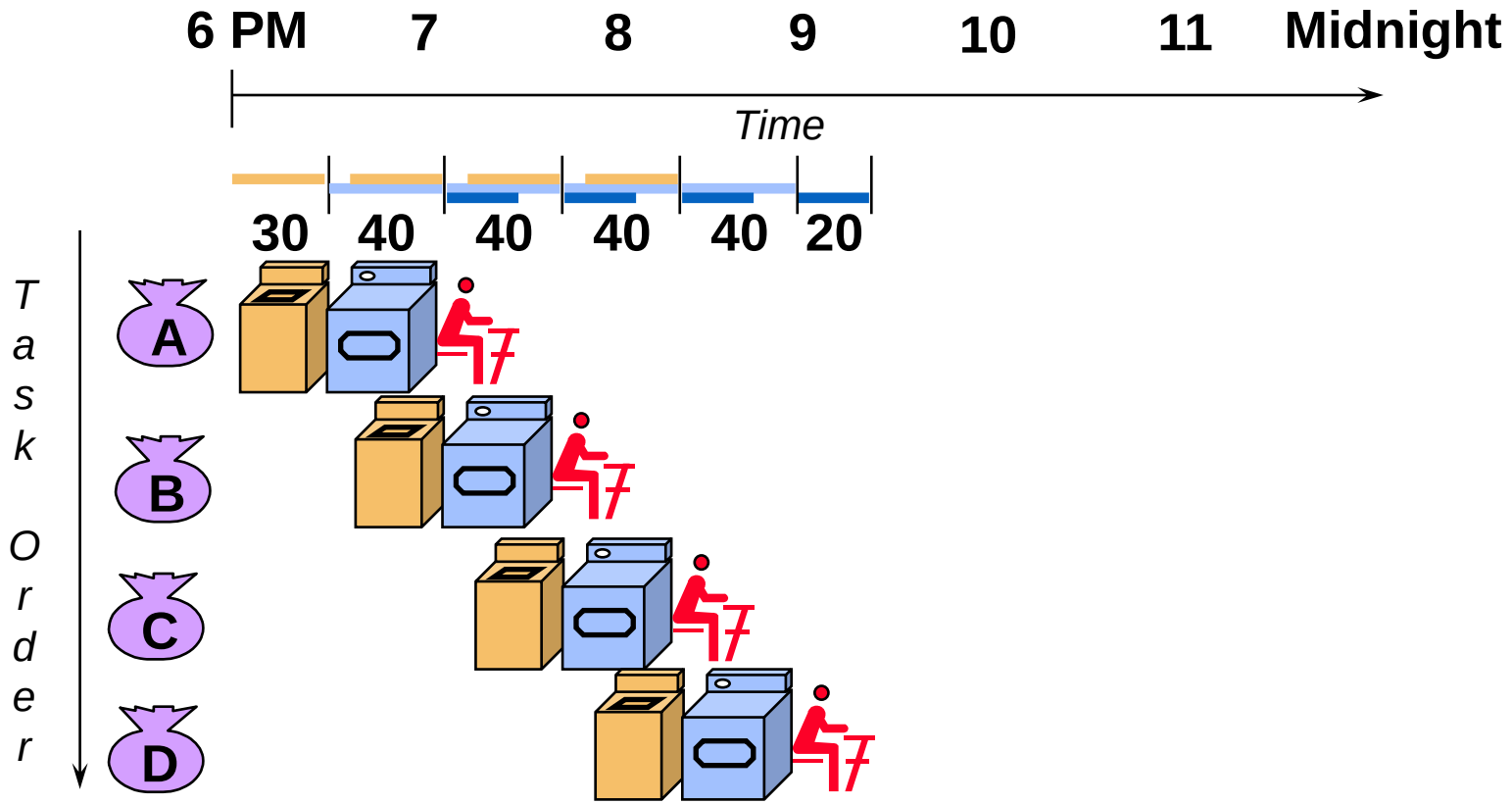
BUS & Pipe-Line

- Sequential Laundry



BUS & Pipe-Line

- Pipelined Laundry



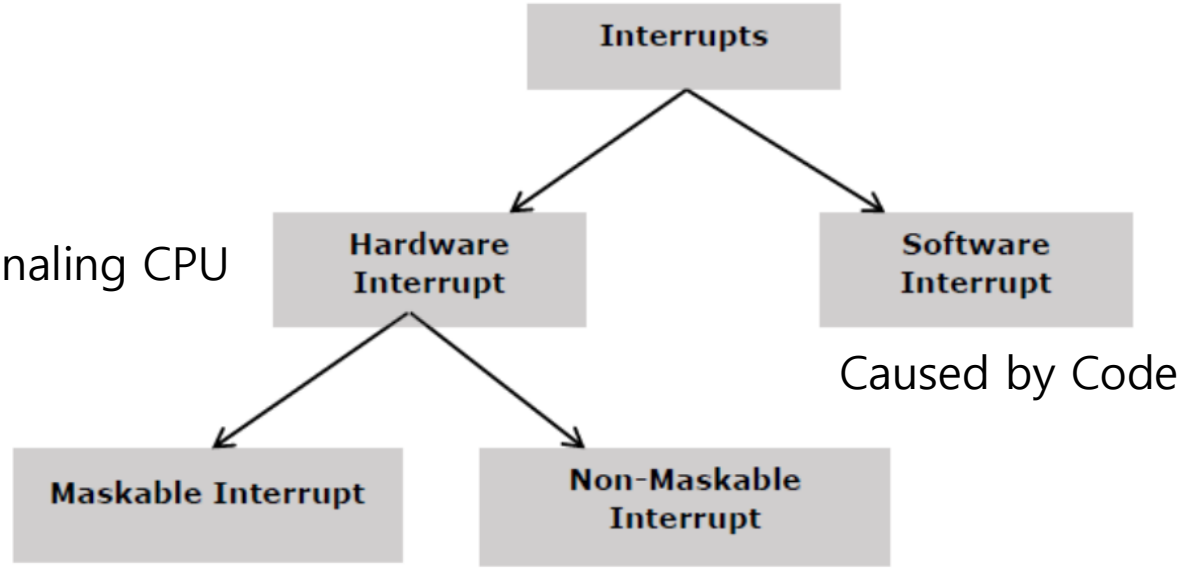
인터럽트 개요

Exception

Unintentional Software interrupt

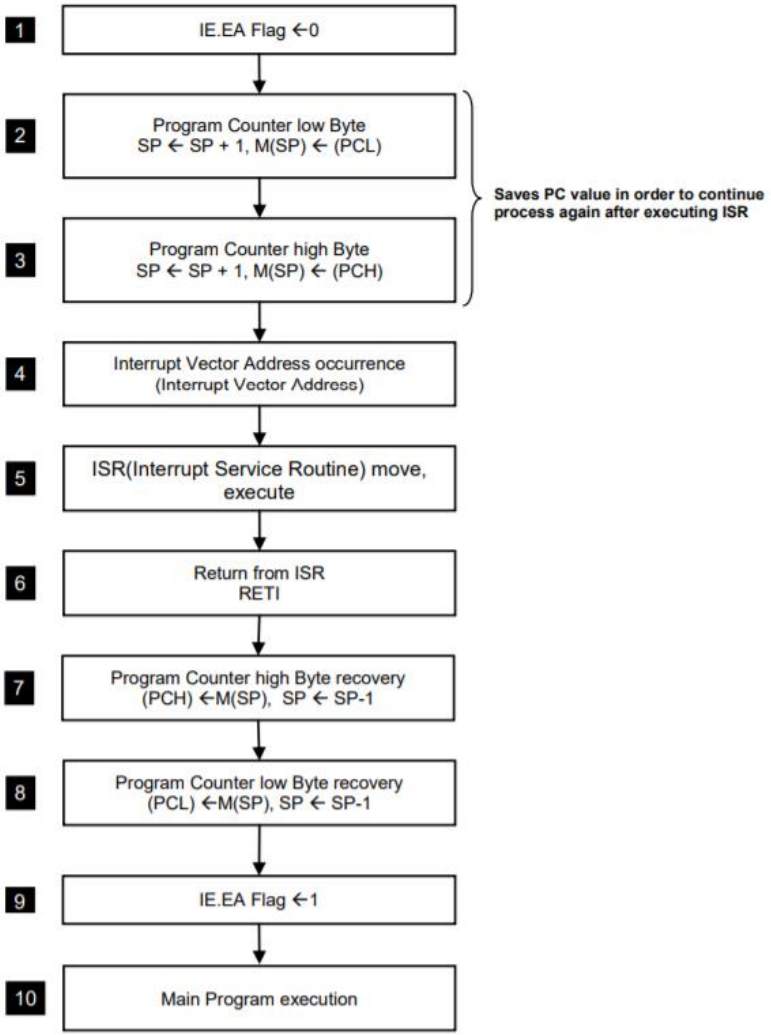
Interrupt

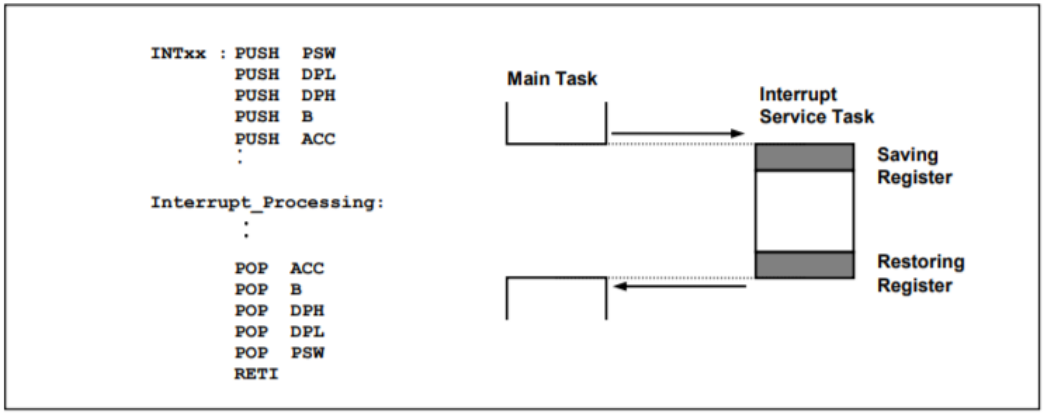
Caused by Devices signaling CPU



Caused by Code

인터럽트 실행 과정


1
IE.EA Flag ←0
2
Program Counter low Byte
SP ← SP + 1, M(SP) ← (PCL)
3
Program Counter high Byte
SP ← SP + 1, M(SP) ← (PCH)
Saves PC value in order to continue process again after executing ISR
4
Interrupt Vector Address occurrence
(Interrupt Vector Address)
5
ISR(Interrupt Service Routine) move, execute
6
Return from ISR
RETI
7
Program Counter high Byte recovery
(PCH) ←M(SP), SP ← SP-1
8
Program Counter low Byte recovery
(PCL) ←M(SP), SP ← SP-1
9
IE.EA Flag ←1
10
Main Program execution

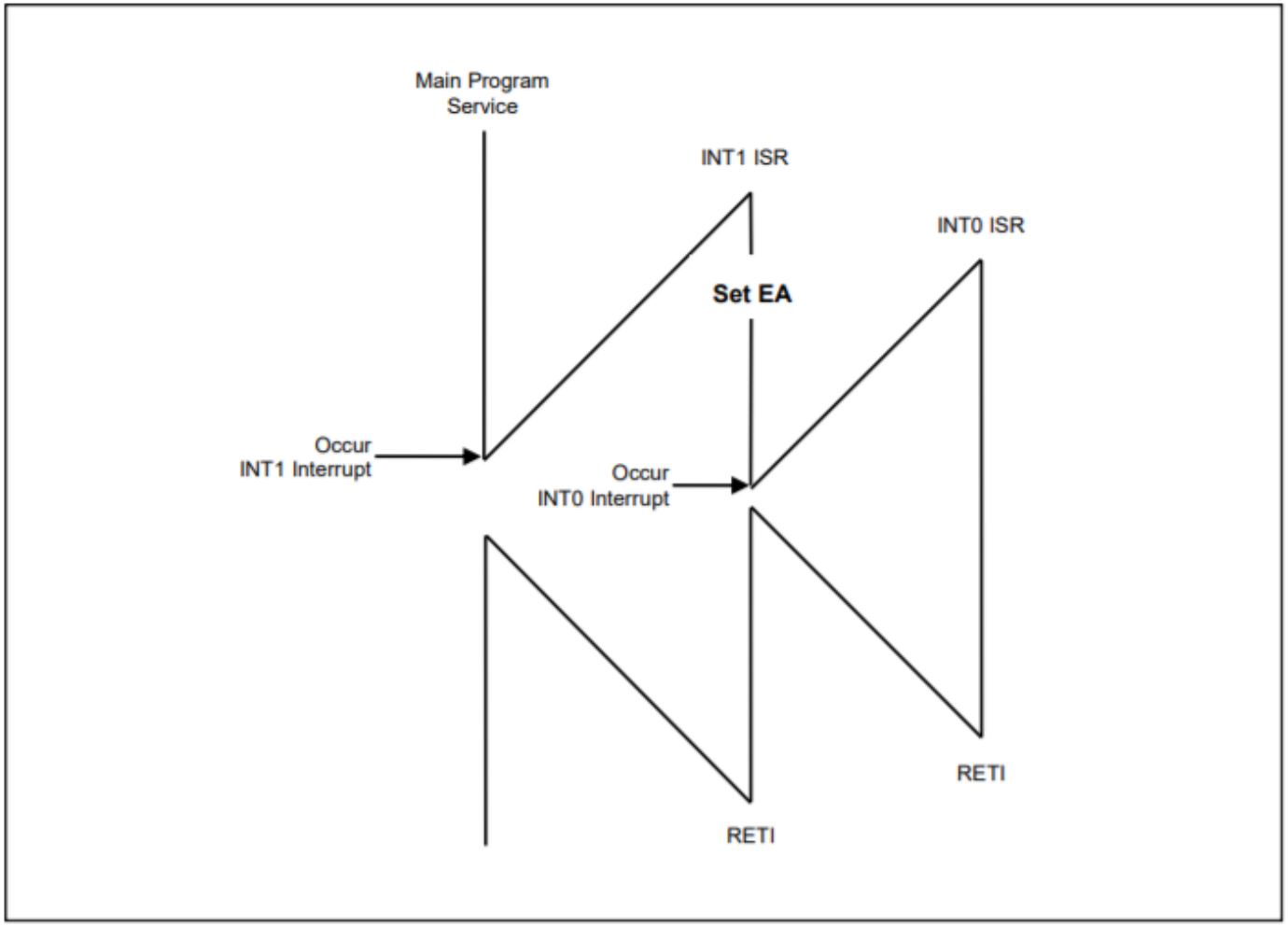

INTxx : PUSH PSW
PUSH DPL
PUSH DPH
PUSH B
PUSH ACC
Interrupt_Processing:
POP ACC
POP B
POP DPH
POP DPL
POP PSW
RETI
Main Task
Interrupt Service Task
Saving Register
Restoring Register

중첩 인터럽트(Nested Interrupt)

- Disable interrupts
  - Processor will ignore further interrupts while processing one interrupt
  - Interrupts remain pending and are checked after first interrupt has been processed
  - Interrupts handled in sequence as they occur

- Define priorities
  - Low priority interrupts can be interrupted by higher priority interrupts
  - When higher priority interrupt has been processed, processor returns to previous interrupt

Multi Interrupt 동작



Case Study: Read, Modify and Write & Interrupt (2/5)

interrupt.c

```
int_req |= 0x00000001;
```

```
PUSH {R0-R2}
LD R1,#0x20000000
LD R2,#0x00000001
LD R0,R1
OR R0,R0,R2
ST R0,R1

~

POP {R0-R2}
RETI
```

PUSH {Rn-Rm}	Rn 부터 Rm을 Stack에 저장
POP {Rn-Rm}	Stack에서 Rn-Rm 꺼냄
LD Rn,#Imm	Rn에 #Imm 저장
LD Rn,Rm	Rm 주소를 읽어서 Rn 저장
ST Rn,Rm	Rm 주소에 Rn 저장
OR Rn,Rs0,Rs1	Rn = Rs0 | Rs1

main.c

```
int_req  = 0x00000000;

~


int_req |= 0x00010000;
```

```
LD R1,#0x20000000
LD R0,#0x00000000
ST R0,R1

~

LD R1,#0x20000000
LD R2,#0x00010000
LD R0,R1
OR R0,R0,R2
ST R0,R1
```

(0x2000_0000) = 0x00010001

Case Study: Read, Modify and Write & Interrupt (3/5)

interrupt.c

```
int_req |= 0x00000001;
```

```
PUSH {R0-R2}
LD R1,#0x20000000
LD R2,#0x00000001
LD R0,R1
OR R0,R0,R2
ST R0,R1

~

POP {R0-R2}
RETI
```

PUSH {Rn-Rm}	Rn 부터 Rm을 Stack에 저장
POP {Rn-Rm}	Stack에서 Rn-Rm 꺼냄
LD Rn,#Imm	Rn에 #Imm 저장
LD Rn,Rm	Rm 주소를 읽어서 Rn 저장
ST Rn,Rm	Rm 주소에 Rn 저장
OR Rn,Rs0,Rs1	Rn = Rs0 | Rs1

main.c

```
int_req  = 0x00000000;

~


int_req |= 0x00010000;
```

```
LD R1,#0x20000000
LD R0,#0x00000000
ST R0,R1

~

LD R1,#0x20000000
LD R2,#0x00010000
LD R0,R1
OR R0,R0,R2
ST R0,R1
```

(0x2000_0000) = 0x00010000

Case Study: Read, Modify and Write & Interrupt (4/5)

interrupt.c

```
int_req |= 0x00000001;
```

```
PUSH {R0-R2}
LD R1,#0x20000000
LD R2,#0x00000001
LD R0,R1
OR R0,R0,R2
ST R0,R1

~

POP {R0-R2}
RETI
```

PUSH {Rn-Rm}	Rn 부터 Rm을 Stack에 저장
POP {Rn-Rm}	Stack에서 Rn-Rm 꺼냄
LD Rn,#Imm	Rn에 #Imm 저장
LD Rn,Rm	Rm 주소를 읽어서 Rn 저장
ST Rn,Rm	Rm 주소에 Rn 저장
OR Rn,Rs0,Rs1	Rn = Rs0 | Rs1

main.c

```
int_req  = 0x00000000;

~


int_req |= 0x00010000;
```

```
LD R1,#0x20000000
LD R0,#0x00000000
ST R0,R1

~

LD R1,#0x20000000
LD R2,#0x00010000
LD R0,R1
OR R0,R0,R2
ST R0,R1
```

(0x2000_0000) = 0x00010000

Case Study: Read, Modify and Write & Interrupt (5/5)

interrupt.c

```
int_req |= 0x00000001;
```

```
PUSH {R0-R2}
LD R1,#0x20000000
LD R2,#0x00000001
LD R0,R1
OR R0,R0,R2
ST R0,R1

~

POP {R0-R2}
RETI
```

main.c

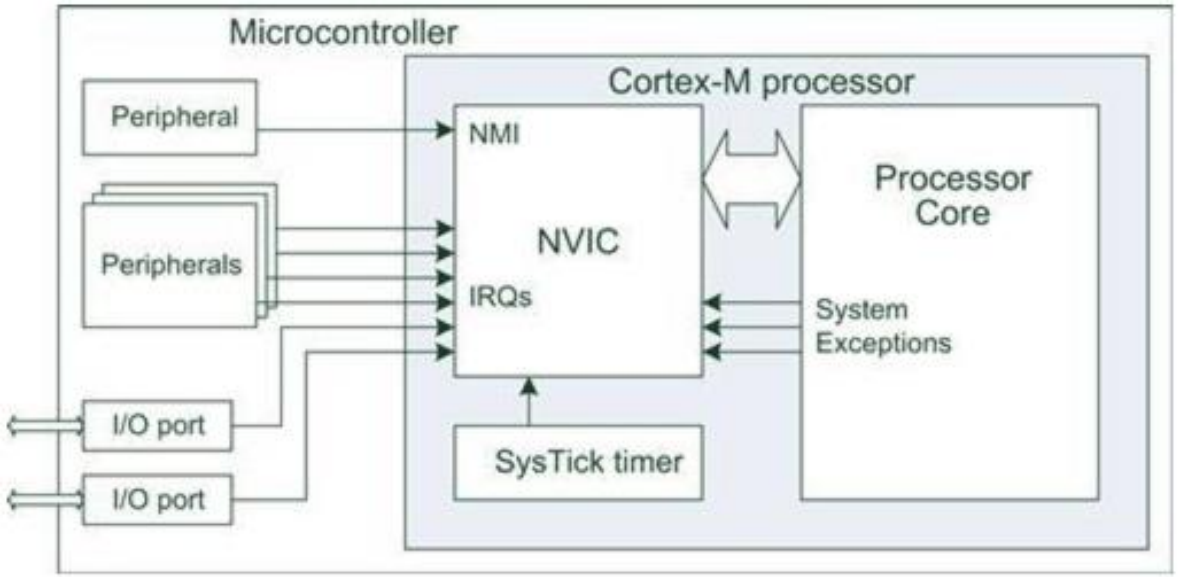
```
int_req  = 0x00000000;

~


Interrupt_Disable();
int_req |= 0x00010000;
Interrupt_Enable();
```

```
LD R1,#0x20000000
LD R0,#0x00000000
ST R0,R1

~

LD R1,#0x20000000
LD R2,#0x00010000
LD R0,R1
OR R0,R0,R2
ST R0,R1
```

(0x2000_0000) = 0x00010001

인터럽트 예제: ARM Cortex-M4

- Typical sequence to handle an exception
  - The peripheral asserts an interrupt request to the processor
  - The processor suspends the currently executing task
  - The processor executes an ISR to service the peripheral, and optionally clear the interrupt request by software if needed
  - The processor resumes the previously suspended task



Various sources of exceptions in a typical microcontroller

ARM Cortex-M4 Exception Types

Exception Number	Exception Type	Priority	Descriptions
1	Reset	−3 (Highest)	Reset
2	NMI	−2	Non-Maskable Interrupt (NMI), can be generated from on chip peripherals or from external sources.
3	Hard Fault	−1	All fault conditions, if the corresponding fault handler is not enabled
4	MemManage Fault	Programmable	Memory management fault; MPU violation or program execution from address locations with XN (eXecute Never) memory attribute.
5	Bus Fault	Programmable	Bus error; usually occurs when AHB interface receives an error response from a bus slave (also called *prefetch abort* if it is an instruction fetch or *data abort* if it is a data access). Can also be caused by other illegal accesses.

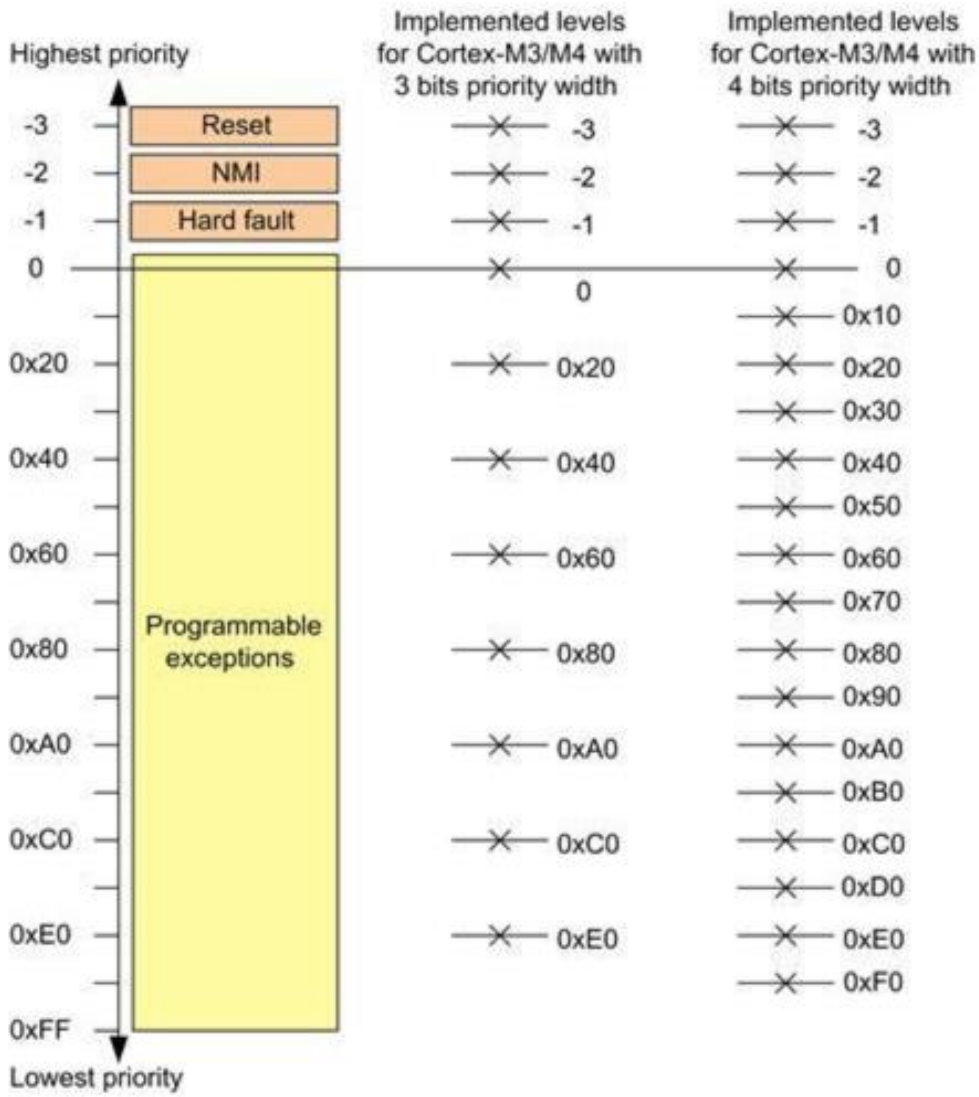
ARM Cortex-M4 Exception Types

6	Usage Fault	Programmable	Exceptions due to program error or trying to access co-processor (the Cortex-M3 and Cortex-M4 processor do not support co-processors).
7–10	Reserved	NA	–
11	SVC	Programmable	SuperVisor Call; usually used in OS environment to allow application tasks to access system services.
12	Debug Monitor	Programmable	Debug monitor; exception for debug events like breakpoints, watchpoints when software based debug solution is used.
13	Reserved	NA	–
14	PendSV	Programmable	Pendable service call; An exception usually used by an OS in processes like context switching.
15	SYSTICK	Programmable	System Tick Timer; Exception generates by a timer peripheral which is included in the processor. This can be used by an OS or can be used as a simple timer peripheral.

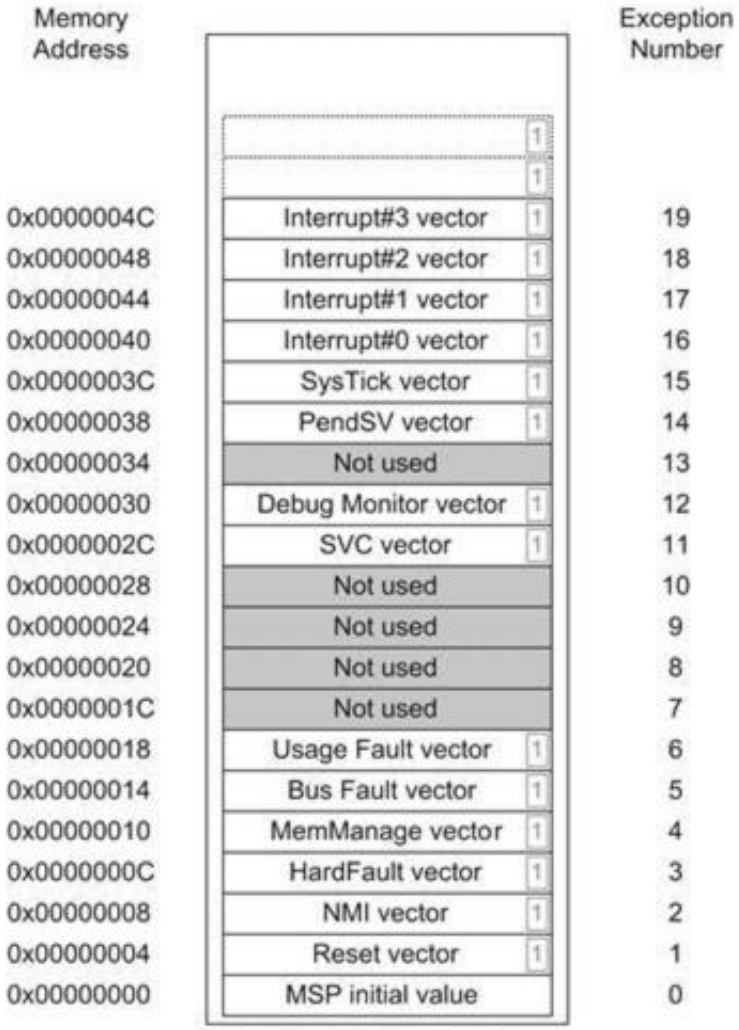
ARM Cortex-M4 List of Interrupts

Exception Number	Exception Type	Priority	Descriptions
16	Interrupt #0	Programmable	It can be generated from on chip peripherals or from external sources.
17	Interrupt #1	Programmable	
…	…	…	
255	Interrupt #239	Programmable	

ARM Cortex-M4 Interrupt Priority Levels



ARM Cortex-M4 Vector Table

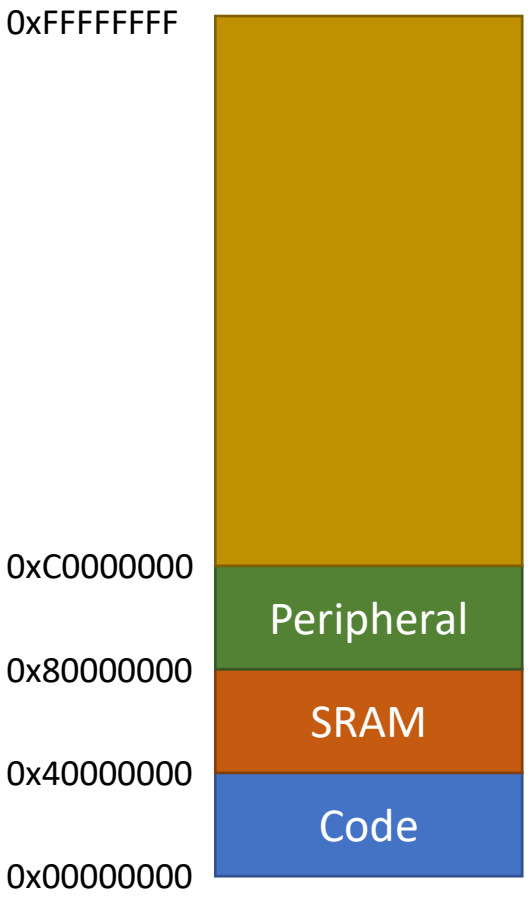


주변장치 (Peripheral)

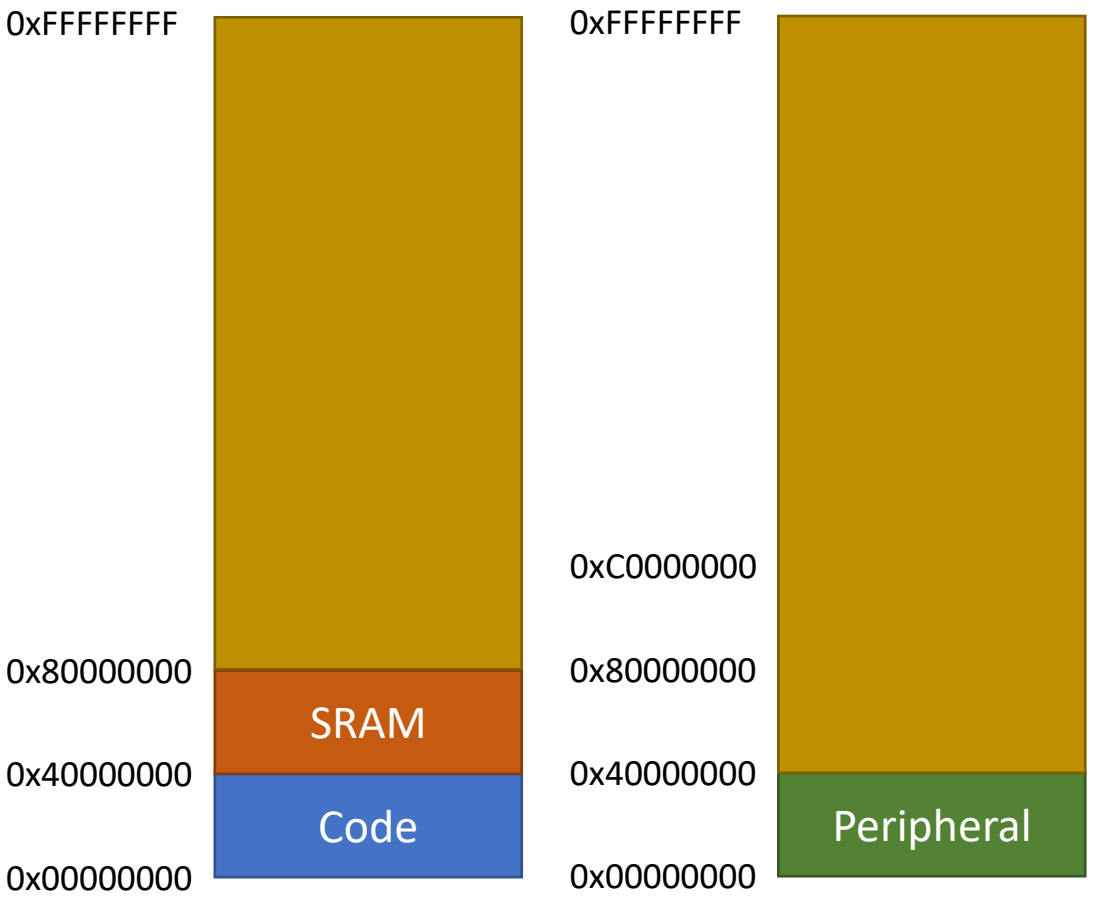
- 데이터 전송
  - Memory-mapped IO
  - Isolated IO
- 장치 접근
  - Polling IO
  - Interrupt-driven IO
- 제어 방식
  - Program-controlled IO
  - DMA (Direct Memory Access)

데이터 전송

Memory-mapped IO



Isolated IO



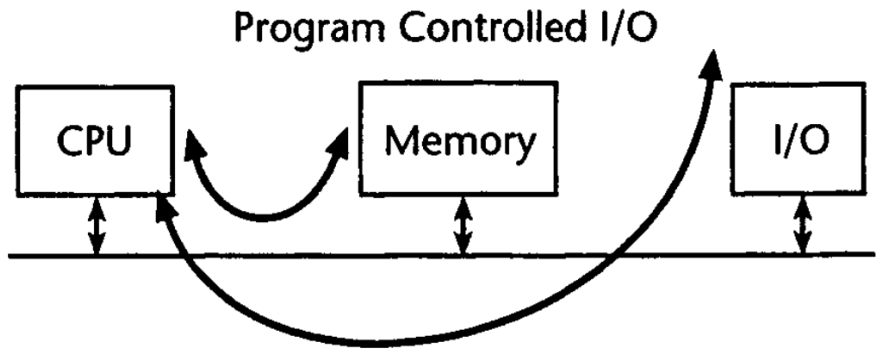
장치 접근

- Polling IO
  Status bit를 이용하여 CPU가 Read해서 확인함.

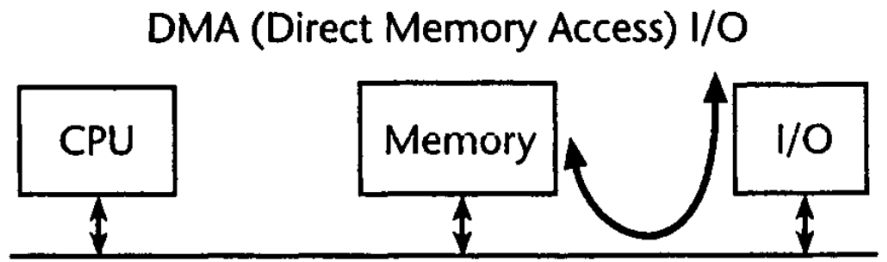
- Interrupt-driven
  Status bit를 Trigger로 Interrupt를 발생시킴.

제어방식

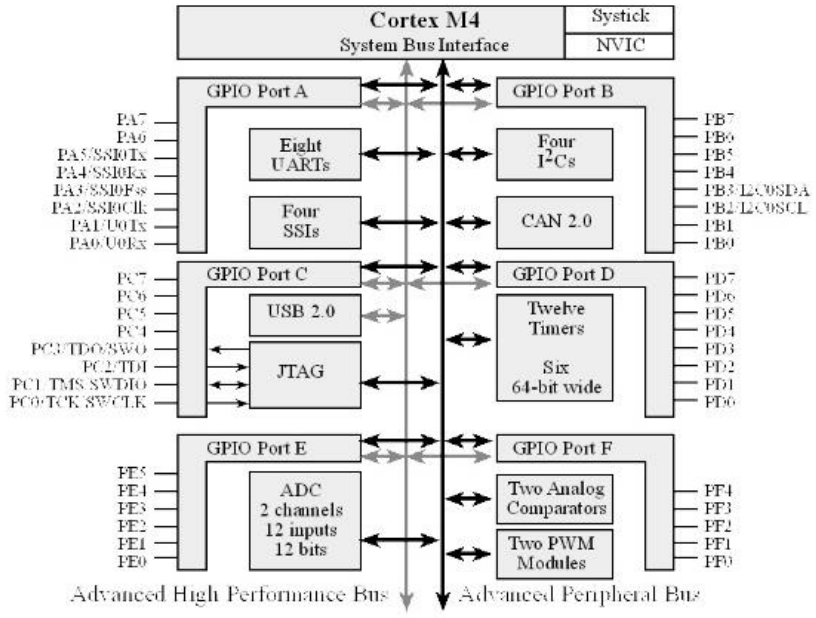
- Program-Controlled IO
  - Control register
  - Data register
  - Status register



- DMA



GPIO (General Purpose Input/Output)



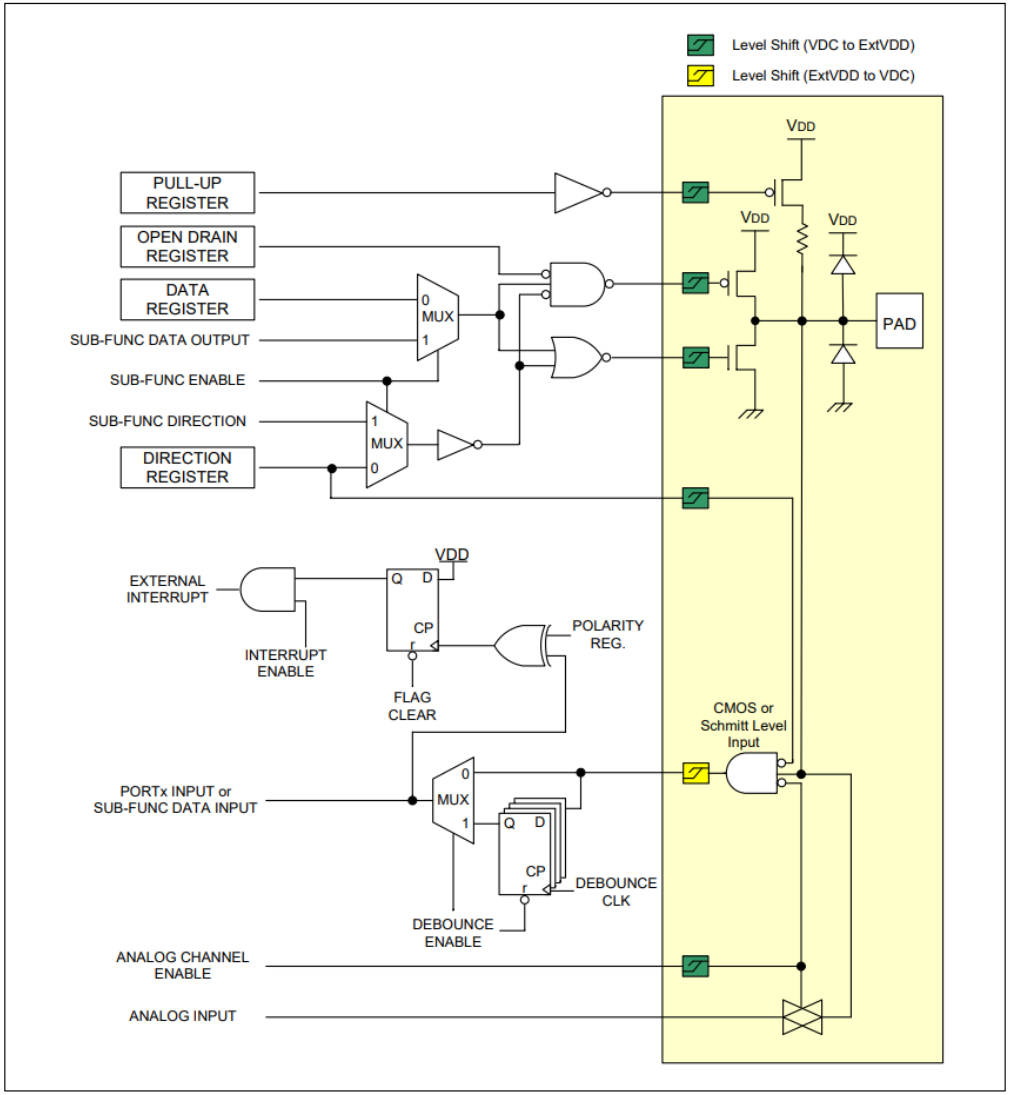
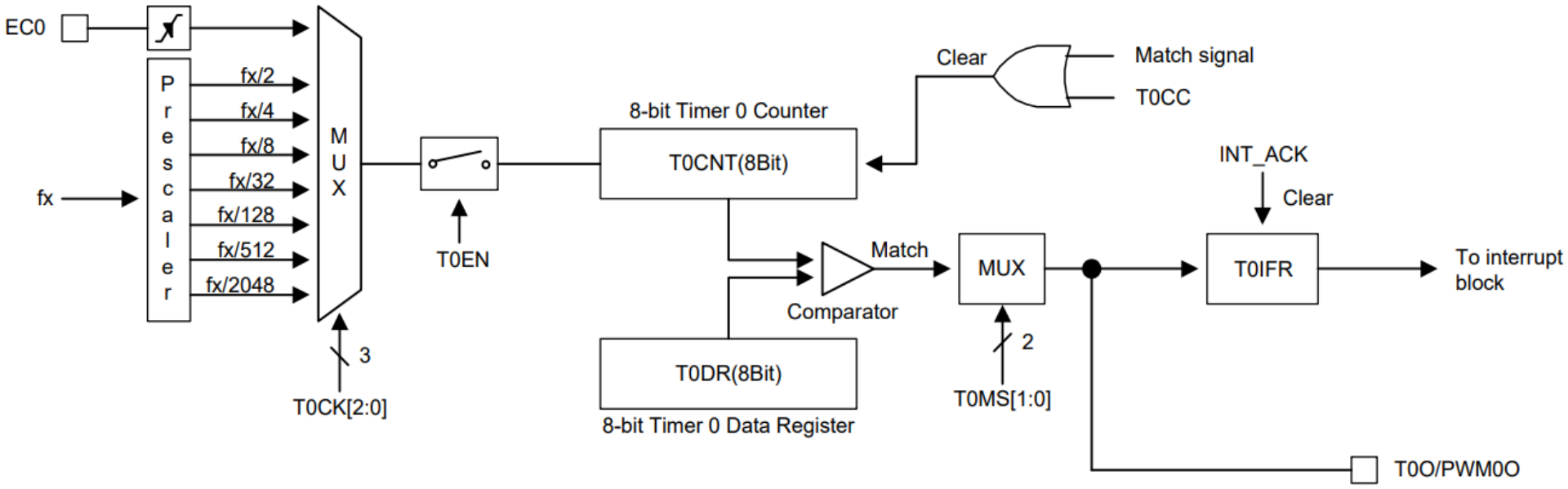


Figure 6.2 External Interrupt I/O Port

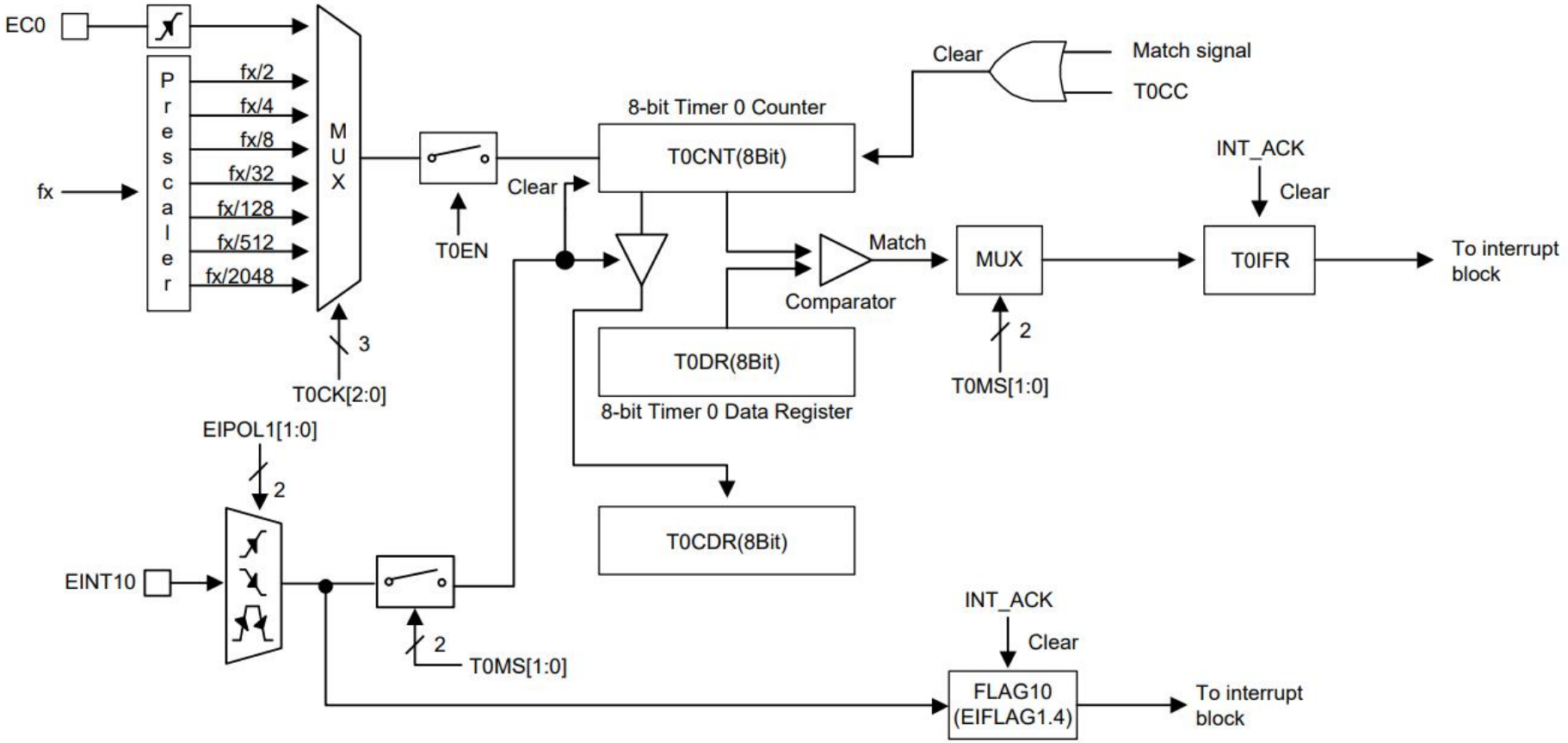
Timer

- Counter-Compare Mode
- Capture Mode
- PWM Mode

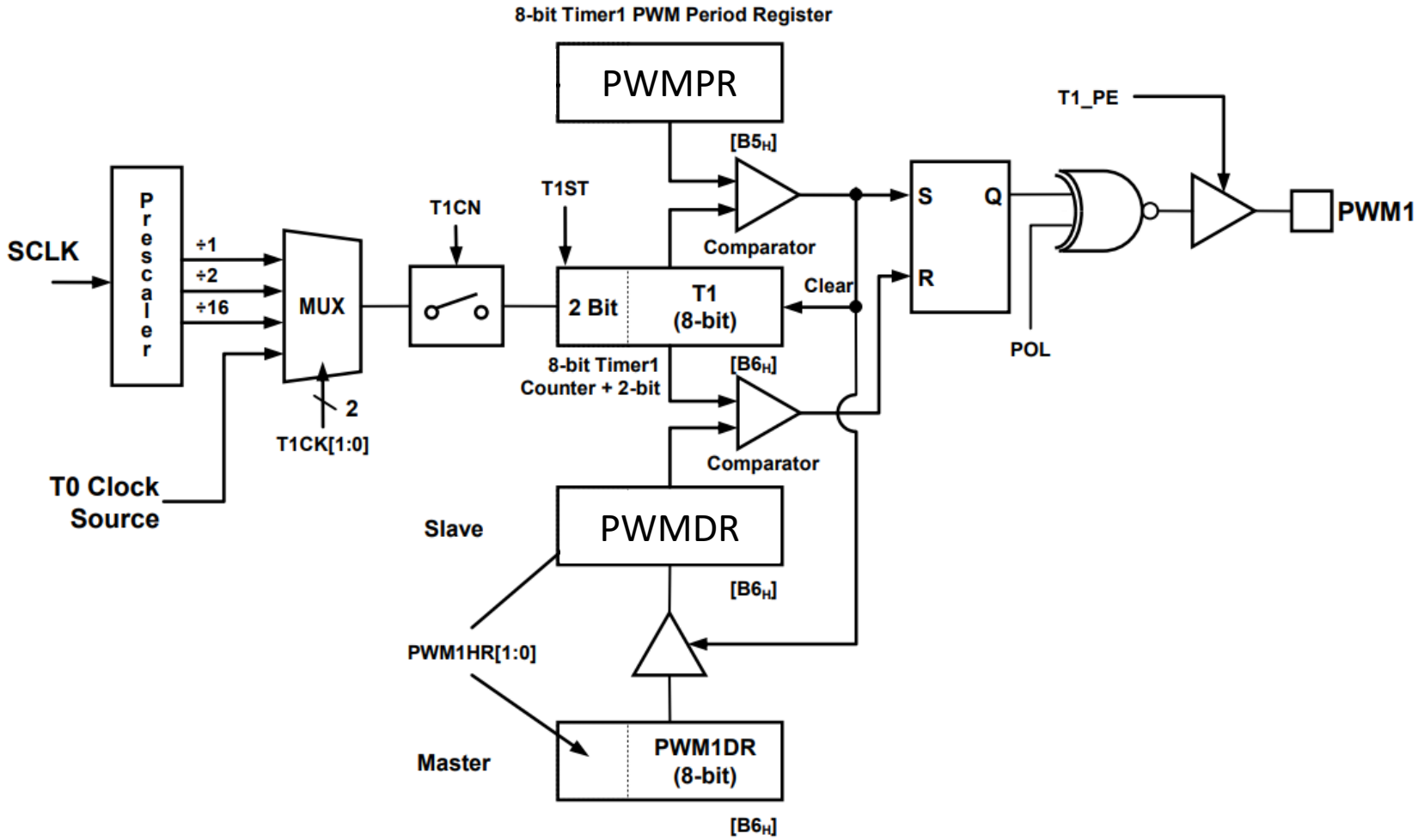
Timer –Count-Compare Mode



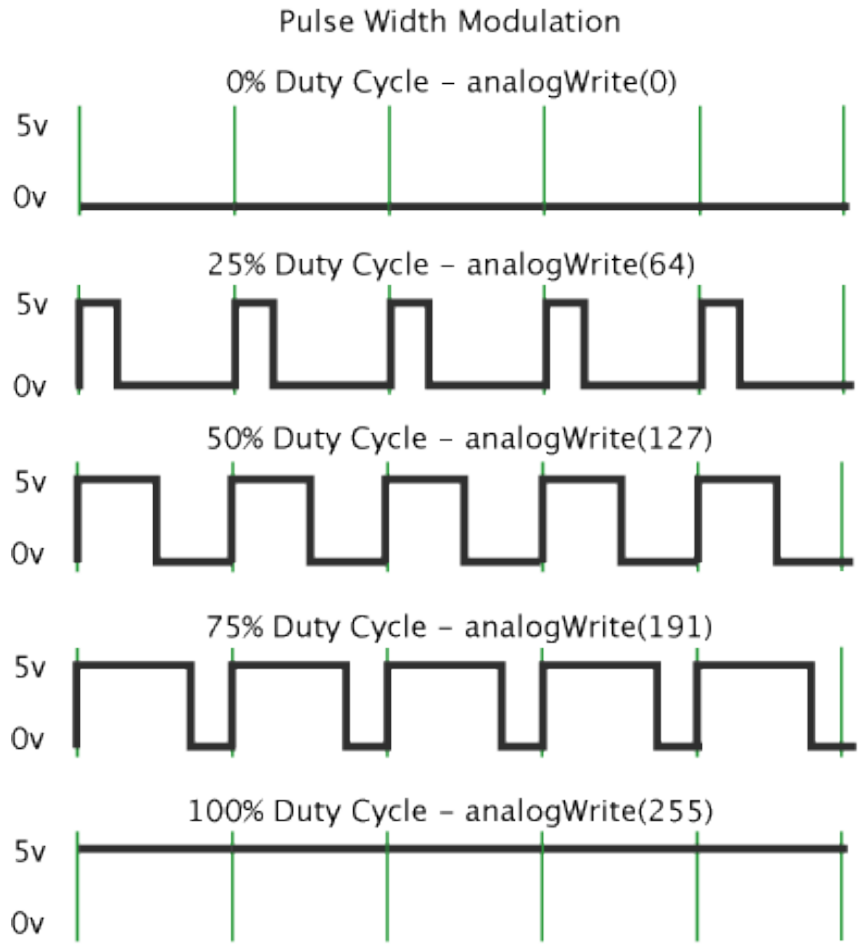
Timer –Capture Mode



Timer – PWM


8-bit Timer1 PWM Period Register
PWMPR
[B5H]
T1_PE
SCLK
Prescaler
÷1
÷2
÷16
MUX
T1CN
T1ST
Comparator
S
Q
R
PWM1
Clear
2 Bit
T1
(8-bit)
POL
8-bit Timer1
Counter + 2-bit
[B6H]
2
T1CK[1:0]
Comparator
T0 Clock
Source
Slave
PWMDR
[B6H]
PWM1HR[1:0]
Master
PWM1DR
(8-bit)
[B6H]

PWM (Pulse-Width Modulation)

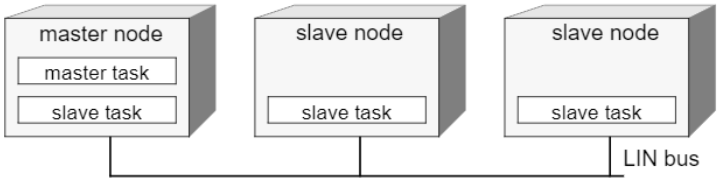


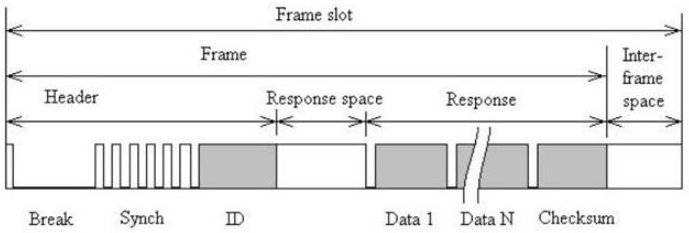
Serial Communication

- Asynchronous
  - 1-line Communication
    - ✓ LIN(Local Interconnect Network)
  - 2-line Communication
    - ✓ DSI3(Digital Serial Interface 3)
    - ✓ UART
      (Universal Asynchronous serial Receiver and Transmitter)
    - ✓ CAN(Control Area Network)
    - ✓ USB(Universal Serial Bus)

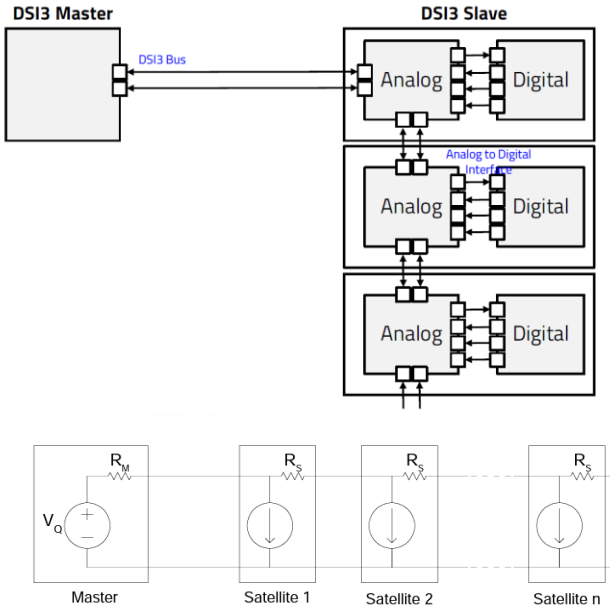
- Synchronous
  - I2C(Inter-Integrated Circuit)
  - SPI(Serial Peripheral Interface)

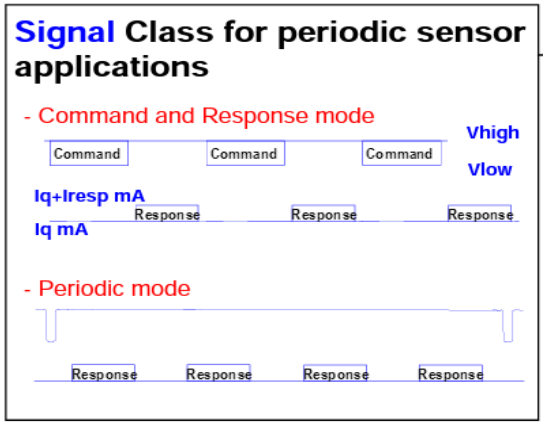
LIN/DSI3

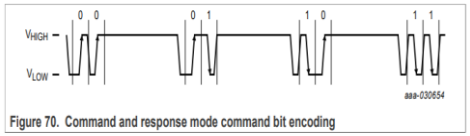


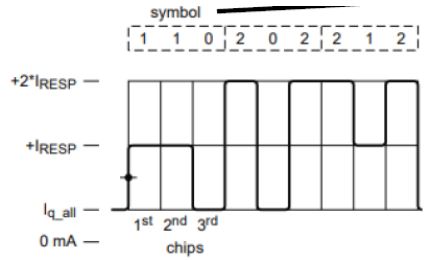


LIN







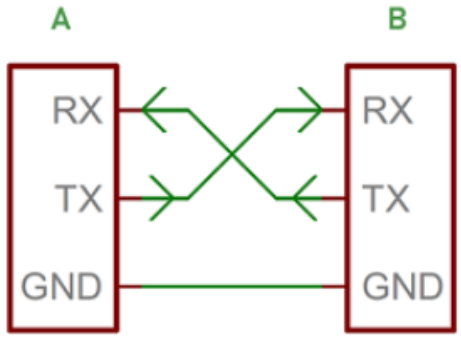


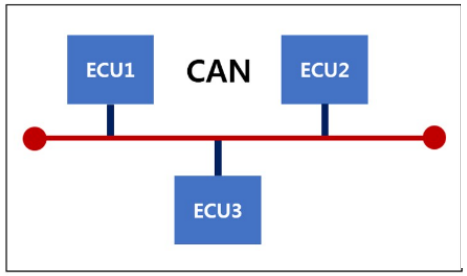
DSI3

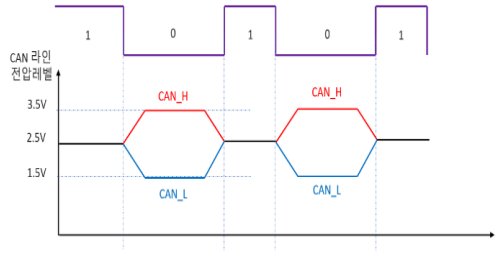
ACE Lab.

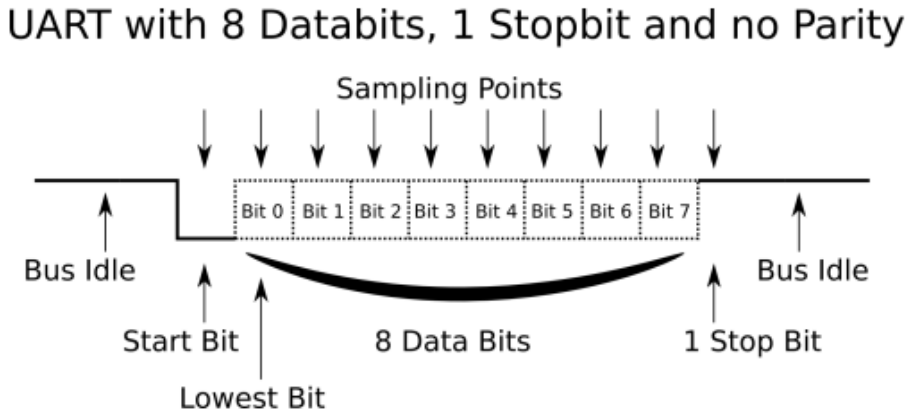
KNU

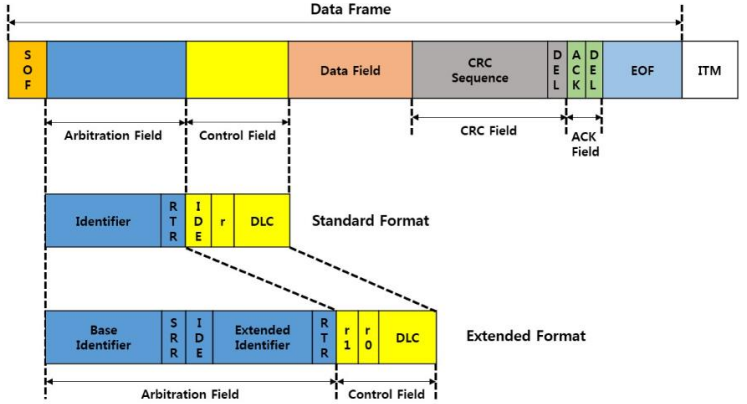
UART/CAN







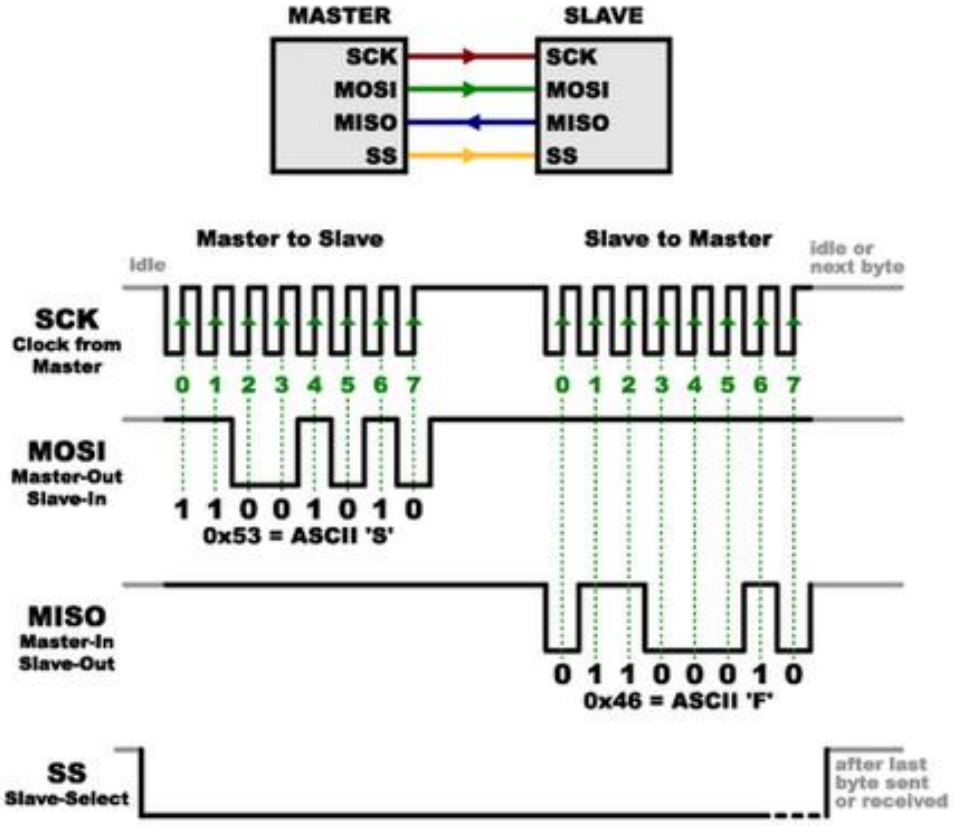


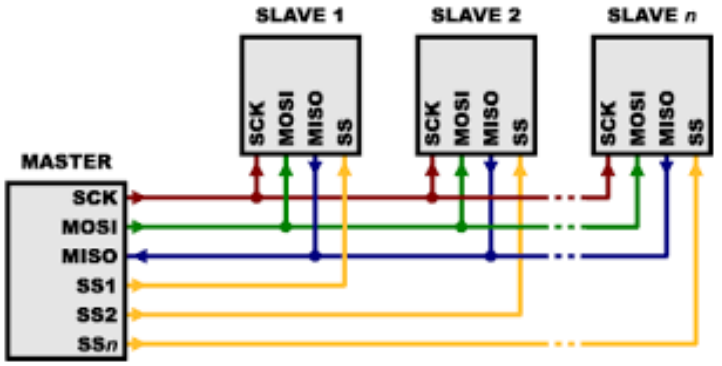


UART

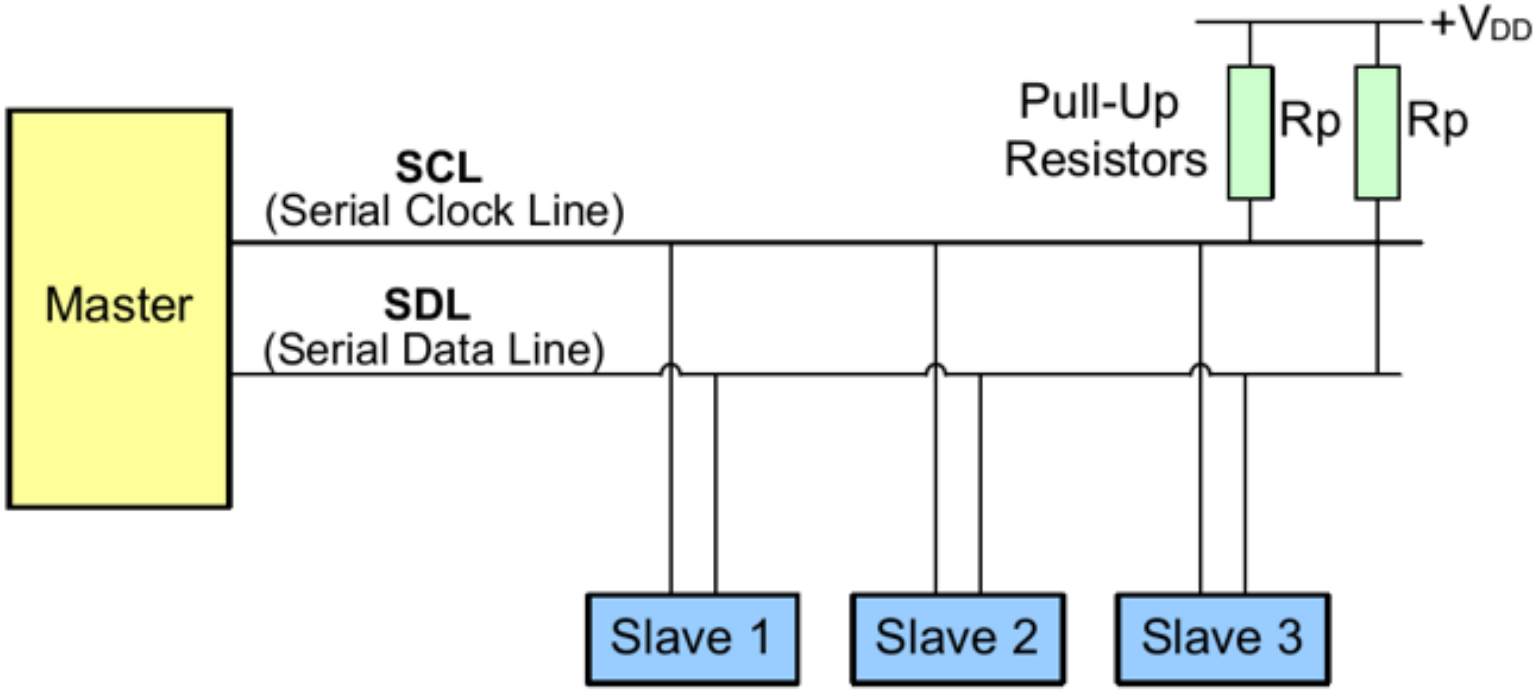
CAN

SPI (Serial Peripheral Interface)

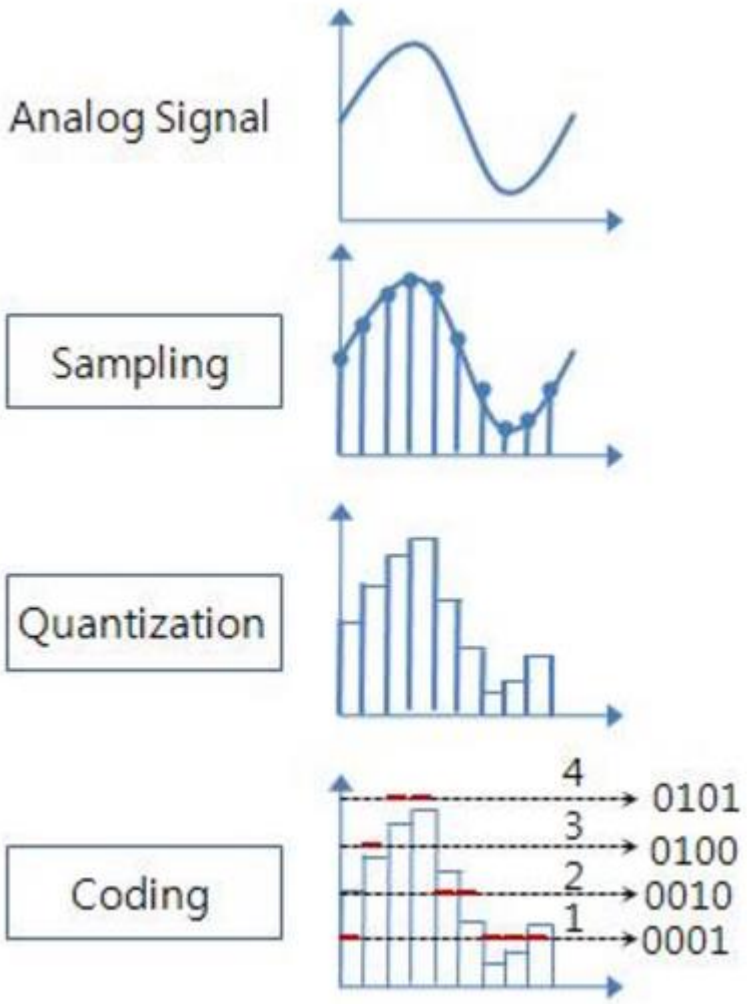


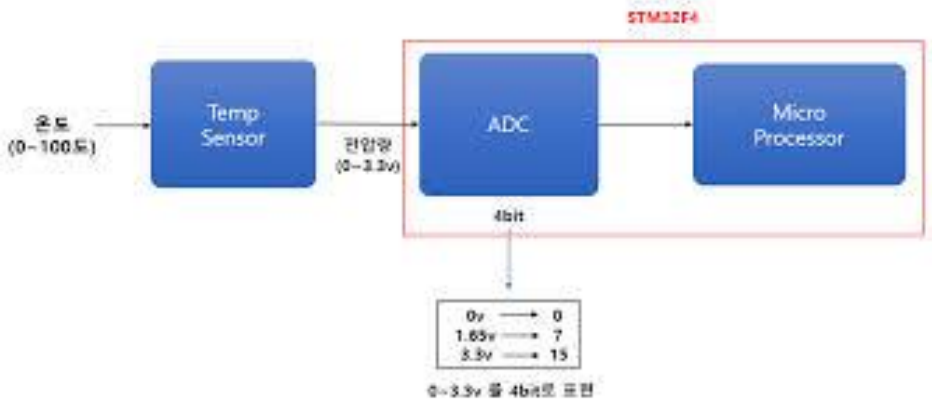


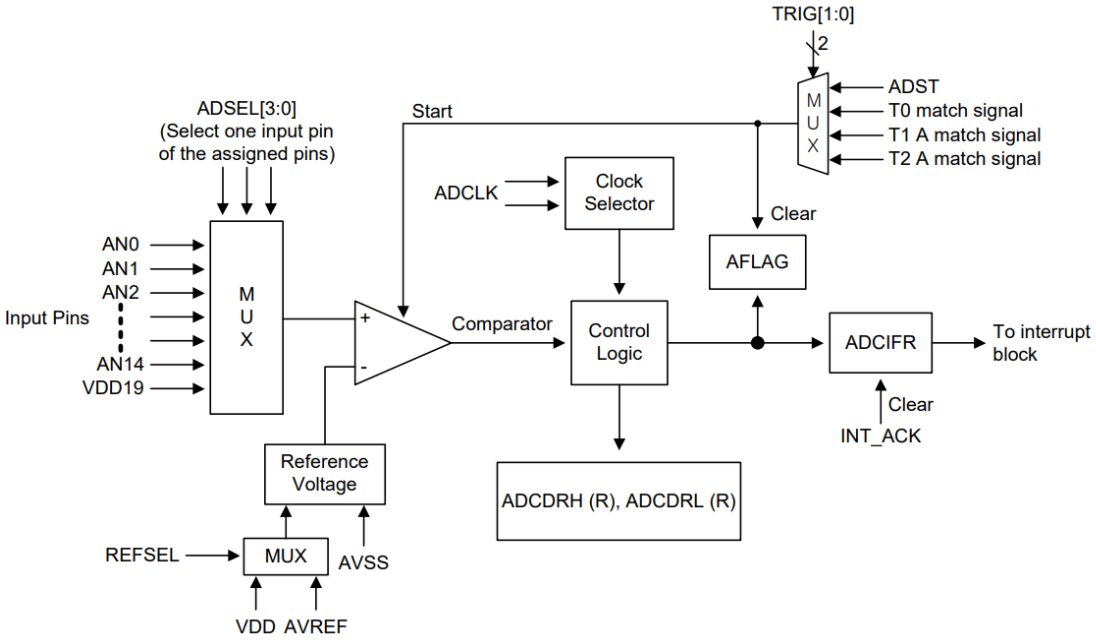
I2C(Inter-Integrated Circuit)



ADC (Analog to Digital Convertor)

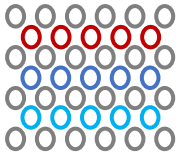






Q & A

Thank you for your attention



Architecture and Compiler
for Embedded Systems Lab.

School of Electronics Engineering, KNU

ACE Lab. (jcho@knu.ac.kr)

ACE Lab.

KNU