마이크로 컨트롤러

- 외부 디지털 전압에 대한 입출력 기능
- 메모리 기능, 타이머 기능, PWM 펄스 생성 기능
- 입력신호 캡처 기능, A/D 변환 기능
- 통신 기능 (LIN, CAN, SPI, RS232, I2C

기계어 : 특정 비트에 특정 의미가 있는 2진값을 설정하는 명령어 나열
어셈블리어 : 사람이 연상하기 쉬운 니모닉과 연산 대상으로 바꿔 표현
어셈블러 : 어셈블리어를 기계어로 변환시키는 번역기

컴파일 : 고급언어를 기계어 목적 파일로 번역
어셈블 : 어셈블리어를 기계어 목적 파일로 번역
링크 : 여러 개의 목적 파일을 연결하여 통합 실행 파일로 만듦
라이브러리 : 공통으로 자주 사용하는 기능으로 컴파일 하여 목적 파일 생성
로딩 : 실행파일을 CPU에서 실행하기 위해 주 메모리에 탑재

범용 레지스터 : CPU 연산을 빠르게 처리하기 위해 ALU와 직접 연결 대상이 되는 오퍼랜드 값을 가짐
- 데이터, 주소 레지스터

특수 레지스터 : 명령어 실행 시 특수한 데이터 처리를 위함
- 프로그램 카운터 : 인출할 명령어가 있는 메모리의 주소를 갖음 (차례로 인출하여 프로그램 실행)
- 명령어 레지스터 : 프로그램 메모리에서 인출된 명령어를 기억하고 있음
- 명령어 디코더 : 명령어 레지스터의 명령코드를 디코딩하여 제어 신호 생성
- 상태 레지스터 : 명령어를 실행 후 연산 결과 정보를 기록
- 메모리 주소 레지스터, 버퍼 레지스터 : 데이터 인출/기록 과정에서 데이터 또는 주소를 임시로 저장하기 위함

명령어 실행 단계
명령어 인출 -> 명령어 분석 -> 오퍼랜드 인출 -> 연산 실행 및 결과 생성 -> 결과 기록

명령코드 : 명령어로 실행되어야 할 연산 동작 표시
오퍼랜드 : 연산에 필요한 데이터 표시
연산결과 : 연산 결과 값을 저장할 위치 표시

3주소 마이크로프로세서 명령어 : 명령어에 3개의 오퍼랜드 주소 포함
2주소 마이크로프로세서 명령어 : 명령어에 2개의 오퍼랜드 주소 포함
1주소 마이크로프로세서 명령어 : 어큐뮤레이터를 묵시적으로 이용하여 연산 수행
0주소 마이크로프로세서 명령어 : 오퍼랜드용 주소를 따로 두지 않고 stack에 있는 연산 대상

데이터 사용

어드레싱 모드
- 즉시 주소방식 : 명령어에 연산 대상이 되는 데이터 값을 갖음
- 직접 주소방식 : 명령어에 포함된 주소는 주 메모리의 연산 대상 데이터에 대한 주소
- 간접 주소방식 : 명령어에 기록된 주소는 오퍼랜드를 인출할 수 있는 주소
- 레지스터 직접 주소방식 : 명령어에 포함된 주소는 레지스터 파일 중 하나를 지칭하는 주소
- 레지스터 간접 주소방식 : 명령어에 포함된 주소는 레지스터 파일의 레지스터 지칭

가상 메모리 : 실행될 프로그램 또는 데이터는 디스크 메모리에 두었다가 실행이 시작되면 주 메모리로 이동 (OS가 관리)

RISC : 파이프라인으로 한 사이클에 한 명령어 실행
CISC : 가변 길이의 명령어 형식으로 명령어의 길이가 길고 다양한 어드레싱 모드를 사용

주변 장치
- 2개의 8비트 타이머/카운터 : 별도의 프리스케일러와 비교 모드 동작
- 2개의 16비트 타이머/카운터 : 별도의 프리스케일러, 비교 모드, 캡처 모드 동작
- 실시간 타이머 카운터
- 6개의 PWM 채널
- 출력 비교 모듈레이터
- 아날로그 비교기
- 10비트 A/D 변환기, A/D 변환기 : Analog to Digital, Digital to Analog
- TWI 직렬 인터페이스 : 직렬통신, 마스터와 슬레이브로 단 2가닥 선만을 이용하여 데이터를 전송
- 2개의 USART : 비동기 통신 (직렬), 송수신을 동시에 진행할 수 있는 양방향 통신
- SPI 직렬 인터페이스 : 직렬 통신, 동기식 데이터 전송 방식으로 마스터와 슬레이브를 정하여 통신
- 와치독 타이머 : code가 정해진 시간 안에 수행되지 않으면 주기적으로 카운터를 리셋하여 재실행
- CAN : CAN 통신
- 외부 인터럽트, 인터럽트+리셋벡터 : 우선순위에 따른 작업 처리를 위해 중간에 하던 작업을 끊음

LockStep : 1CPU는 작업 수행, 1CPU는 다른 CPU가 잘하고 있나 감지하여 문제 발생 시 해결

스택 포인터 : 임시 데이터, 로컬 변수, 호출된 함수의 복귀 주소 등을 저장한 스택의 top을 저장

메모리
- 플래시 프로그램 메모리 : 비휘발성 메모리로 프로그램을 저장
- SRAM : 프로그램에서 선언한 변수와 스택을 위해 읽고 쓰기를 빠르게 수행 (휘발성)
- EEPROM : 프로그램 실행 시 생성된 데이터를 전원 없이도 유지시키기 위함
- 외부 메모리 : 내장된 SRAM이 부족할 경우 추가 장착
- EPROM : 자외선으로 데이터를 소거 시 사용
- ISP : 마이크로컨트롤러를 장착한 PCB에 직접 장착하여 시스템 개발

8비트 타이머/카운터 : 256까지 max 카운터
16비트 타이머/카운터 : 65536까지 max 카운터

연속 시스템을 라플라스 변환을 통해 z변환을 하며 이산 시스템으로 바꿔서 계산

ELF : 실행 코드, 목적 코드, 공유 라이브러리를 포함하는 표준 파일 형식
- 시작코드, 바이트 수, 주소, 레코드 타입, 데이터, 체크섬이 포함

2의 보수 : 컴퓨터에서 음수를 표현하기 위한 체계
- 숫자에 비트 반전을 시키고 +1

툴체인 : 작업 중인 컴퓨터에서 다른 CPU에 실행될 기계어 프로그램을 생성하기 위한 프로그래밍 툴의 집합

make : 여러 개의 소스 파일로부터 자동으로 목적 파일을 만들 수 있도록 의존 관계와 파일 갱신 시간 정보로 특정 프로그램의 실행 지정

최적화 : 원시 프로그램이 타깃 코드로 변환될 때 컴파일러에 의해 같은 기능을 유지하고 크기는 작게, 빠르게 실행될 수 있게 변환
-O0 : 최적화 X, // O, O1 : 가장 기본적인 최적화 // O2 : 코드 크기 줄어듦
-O3 : 파일의 크기는 커질 수 있지만, 속도를 높이기 위해 시도하는 최적화 옵션
-Os : 코드의 크기를 줄이는 목적 (마이크로컨트롤러에서 주로 사용)

volatile : 최적화는 속도를 빠르게 하고 메모리 사용량을 줄일 수 있지만 인터럽트를 사용하는 경우 자칫 원하지 않는 최적화를 실행 할 수 있기 때문에 프로그램이 설계와 다르게 작동하는 것을 막기 위해 인터럽트 서비스 루틴에서 사용할 변수에 사용하여 오류를 막음

JTAG : 바운더리 스캔 방식을 이용하여 IC 내부를 모니터링과 변경, 플래시 프로그램 메모리에 다운로드 기능, 명령어 진행을 정지시키고 내부 레지스터 및 SRAM 데이터 관찰 (디버거)

unsigned : 양수와 0값만 가짐, signed : 양수, 0, 음수 모두 가짐
register : 컴파일러에게 선언되는 변수를 레지스터로 할당
volatile : 컴파일 시 최적화를 하지 않음
static : 함수가 종료되어도 변수값을 유지함
const : 초기화된 변수는 상수 취급, 수정할 수 없음

비트 OR 연산자 | : 동일한 위치의 비트끼리 OR 연산을 수행하여 둘 중 하나가 1이면 결과는 1, 둘 다 0일 때만 0
ex) a|control -> control이 0이면 a, control이 1이면 1

비트 XOR 연산자 ^ : 동일한 위치의 비트끼리 XOR 연산을 수행하여 둘이 서로 결과가 다르면 1, 둘 다 같으면 0
ex) a^control -> control이 0이면 a, control이 1이면 a반전

비트 AND 연산자 & : 동일한 위치의 비트끼리 AND 연산 수행하여 둘 중 하나가 0이면 결과가 0, 둘 다 1일 때만 1
ex) a & control -> control이 0이면 0, control이 1이면 a

왼쪽 비트 천이 연산자 << : 변수를 좌측으로 n비트 만큼 천이, 우측비트 0으로 채워짐, 2를 곱한 결과
ex) a = a << 2 -> a비트를 2비트만큼 MSB로 천이

오른쪽 비트 천이 연산자 >> : 변수를 우측으로 n비트 만큼 천이, 좌측비트 0으로 채워짐, 2를 나눈 결과
ex) a = a >> 2 -> a비트를 2비트만큼 LSB로 천이

자연신호를 전기신호로 변ㄴ환하고 A/D 변환을 통해 디지털 시스템에 입력, 디지털 시스템에서 출력 시 D/A 변환을 통해 구동 (요즘은 거의 D/A 사용 X)

공급 전압 5V에서 최소 공급 전압은 4.75V, 최대 공급 전압은 5.25V
HIGH 입력 전압은 최소 2V, LOW 입력 전압은 최대 0.5V
HIGH 출력 전류는 최대 -0.4mA, LOW 출력 전류는 최대 16mA

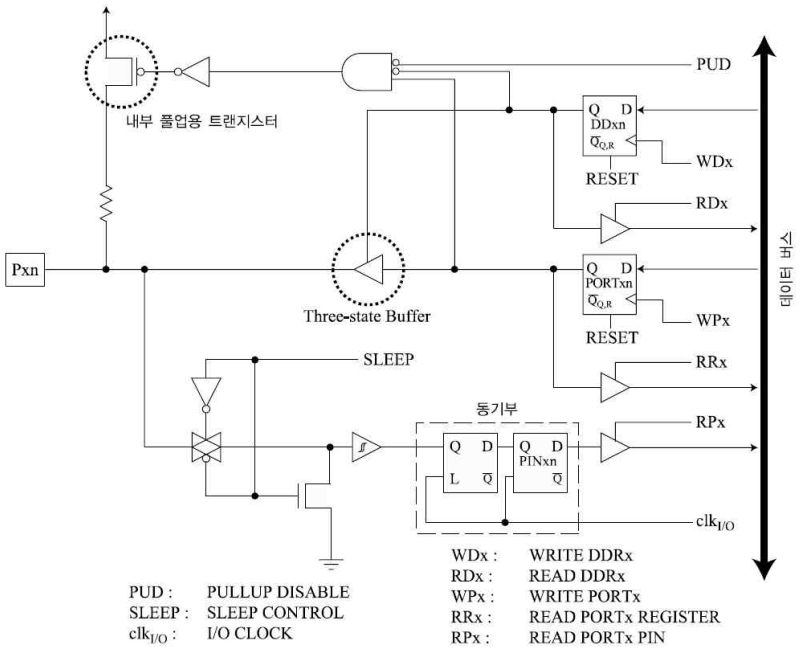
디지털 출력 전압 레벨 (1 : HIGH 출력, 0 : LOW 출력)
- 공급 전압 5V일 때 : HIGH : ~ 4.0V, LOW : 0.7V ~
- 공급 전압 3V일 때 : HIGH : ~ 2.2V, LOW : 0.5V ~

디지털 입력 전압 레벨 (1 : HIGH 입력, 0 : LOW 입력)
- 공급 전압 5V일 때 : HIGH : 5.5 ~ 3.0V, LOW : 1.0V ~ -0.5V
- 공급 전압 3V일 때 : HIGH : 3.3 ~ 1.8V, LOW : 0.6 ~ -0.5V

잡음 여유
- HIGH(1)을 전압레벨로 출력 시 : 4.0-3.0 = 1.0V
- LOW(0)을 전압레벨로 출력 시 : 1.0-0.7 = 0.3V

Source 모드 : PLC에서 전류가 나와서 입출력 소자로 감
Sink 모드 : 입출력 소자에서 전류가 PLC로 들어감

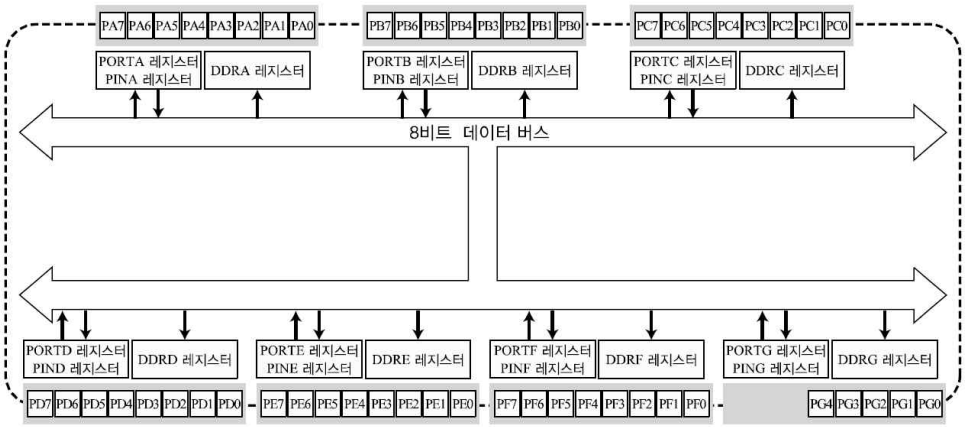
I/O 핀의 구조



- PUD : PullUp 레지스터(외부)

핀 방향 설정				출력 레벨/내부 풀업 설정			
방향	레지스터	비트명	값	출력 레벨/내부 풀업	레지스터	비트명	값
Pxn 핀 출력 설정	DDRX	DDxn	1	HIGH 출력	PORTx	PORTxn	1
				LOW 출력		PORTxn	0
Pxn 핀 입력 설정	DDRx	DDxn	0	내부 풀업 있음	PORTx	PORTxn	1
					SFIOR	PUD	0
				내부 풀업 없음	PORTx	PORTxn	0 또는 1

- Three-state Buffer : High, Low, z = High Impedance(신호 출력 소자와 신호 입력 소자만 연결하여야 하는데 복수 출력 회로가 동시에 접속되면 비정상 전류가 흐르기 때문에 소자가 파괴. 이를 방지하기 위해 출력 신호선에서 전기적으로 절연 상태)
- PUD가 0이고 PORTxn이 1이면 내부 풀업저항 연결
- PUD가 1이거나 PORTxn이 0이면 내부 풀업저항 개방
- DDRx 레지스터 : 데이터 방향 레지스터로서 포트에 포함된 핀의 입출력 방향 결정
- PORTx 레지스터 : 출력 방향(DDRx = 1)일 때, 논리값을 디지털 전압레벨로 출력
  입력 방향(DDRx = 0)일 때, 1로 설정되면 내부 풀업저항 연결
- PINx 레지스터 : 입력 핀 레지스터, 입력 방향일 때 핀의 디지털 전압레벨을 논리값으로 읽음



비트	7	6	5	4	3	2	1	0
DDRx	DDx7	DDx6	DDx5	DDx4	DDx3	DDx2	DDx1	DDx0
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

비트	7	6	5	4	3	2	1	0
PORTx	Px7	Px6	Px5	Px4	Px3	Px2	Px1	Px0
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

비트	7	6	5	4	3	2	1	0
PINx	PINx7	PINx6	PINx5	PINx4	PINx3	PINx2	PINx1	PINx0
읽기/쓰기	R	R	R	R	R	R	R	R
초깃값	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

- DDRx 레지스터의 초기값은 0이므로 최초 설정은 모두 입력
- 입력인 상황에서 PORTx 레지ㅣ스터의 초기값은 0
- Pxn핀은 풀업저항이 연결되지 않은 High-Impedance인 입력 핀, 초기 전압은 예측할 수 없어 PINx 초기값 N/A
- DDRx 레지스터 DDxn 출력 핀 위치에 1 기록 시 출력 디지털 전압레벨이 HIGH이면 PORTxn에 1 기록, 아니면 0 기록
- DDRx 레지스터 DDxn 입력 핀 위치에 0 기록 시 입력 핀 Pxn에 풀업저항이 연결되어 있다면 PORTxn에 1 기록/PUD 0 기록, 아니면 0 기록 -> PINxn 비트값을 읽어 취함
- PINxn 전압레벨 검사 : HIGH는 0이 아닌값, LOW는 0 (PINx & (1 << PINxn))

ex) DDRA = 0xF0 -> PA7,6,5,4는 출력, PA3,2,1,0은 입력 핀

ex) DDRA = DDRA | (1 << PA7) | (1 << PA6)

- PA7 = 7, PA0 = 0 // DDRA7 = 7, DDRA0 = 0

=> DDRA = 1 1 DDRA5 DDRA4 DDRA3 DDRA2 DDRA1 DDRA0

(7번, 6번 핀 출력, 나머지 사용 X)

ex) DDRA = 0xFF // PORTA = (PORTA & 0xF) | 1 << PA7 | 1 << PA6

=> DDRA는 모두 출력

=> PORTA = 1 1 0 0 PA3 PA2 PA1 PA0

(7번, 6번 핀 HIGH, 5번 4번 핀 LOW, 나머지 사용 X)

ex) DDRA = DDRA & ~(1<<PA7) & ~(1<<PA6) // value = PINA >> PA6

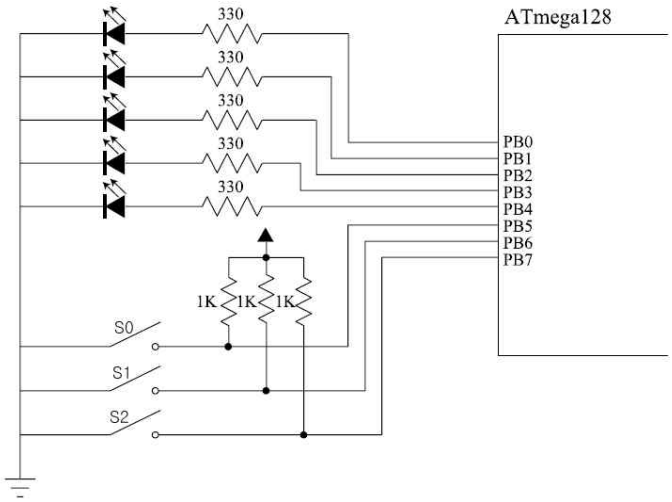
- DDRA = 0 0 DDRA5 DDRA4 DDRA3 DDRA2 DDRA1 DDRA0
  (7번과 6번 핀 입력으로 사용, 나머지 사용 X)
- 7번과 6번 핀이 입력이므로 PINA = PINA7 PINA6 ? ? ? ? ? ?
  value = 0 0 0 0 0 0 PIN7 PIN6

ex) PINA & (1<<PA6)

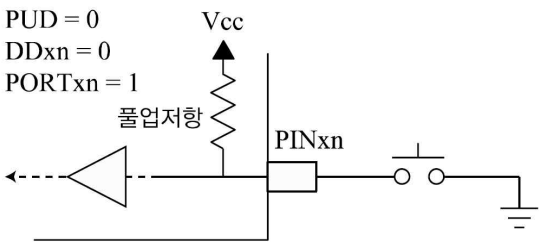
- PINA의 PINA6이 HIGH라면 PINA6, LOW라면 0

ex) DDRB = 0x1F

- PB0,1,2,3,4 : 출력 // PB5,6,7 : 입력
- DDRB = 1<<DDB4 | 1<<DDB3 | 1<<DDB2 | 1<<DDB1 | 1<<DDB0과 같음



- 캐패시터 : 입력값의 Noise를 막아줌
- 스위치가 눌리지 않으면 HIGH 전압레벨 / 눌리면 LOW 전압레벨
- LED는 스위치가 눌릴 때 LOW 전압레벨에서 동작



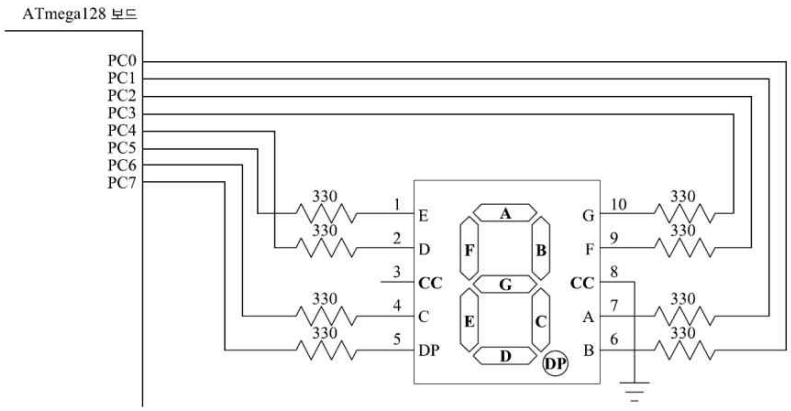
- PUD = 0이고 DDxn = 0, PORTxn = 1 : 해당 핀을 입력으로 사용, 내부 풀업 저항을 사용함.

- 스위치를 누르면 Vcc가 ground로 빠지기 때문에 MCU에 0V가 걸리고 스위치를 떼면 Vcc가 MCU 내부로 들어가 5V가 걸림 => 스위치를 누르면 MCU로 LOW값

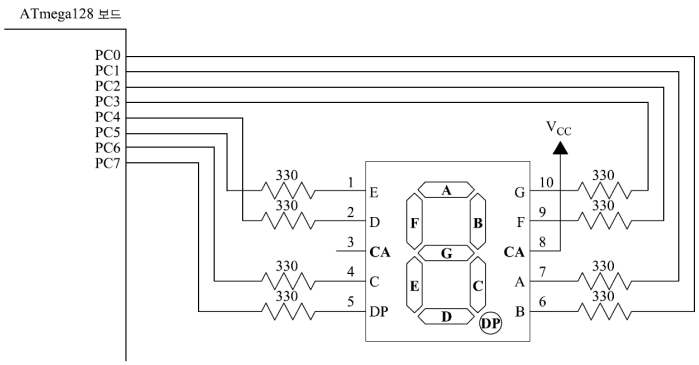
직접 구동 : 마이크로컨트롤러의 디지털 출력 핀을 사용하여 외부 장치를 직접 제어
간접 구동 : 마이크로컨트롤러의 출력 신호를 중간 장치를 통해 전달하여 외부 장치를 제어
- 릴레이와 트랜지스터를 사용하여 간접구동

7-세그먼트 LED

공통 캐소드 LED 구동 회로

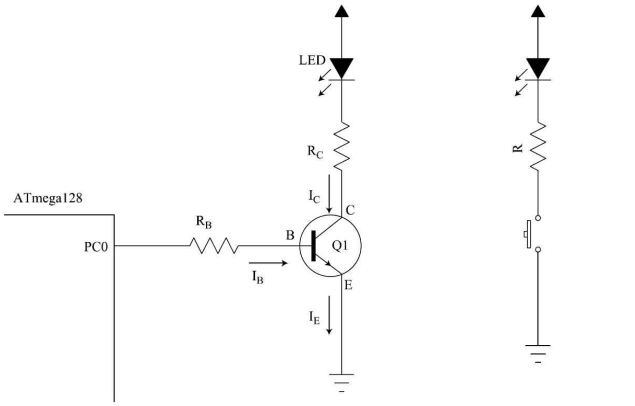


ex) LED로 1을 표현 시 PC0 = 1, PC6 = 1, 나머지 0 => B와 C High, 나머지 Low
=> Source 전류가 흐름 (외부 인터페이스는 접지되어 있어 무조건 0V이기 때문에 공급 전압이 5V인 MCU에서 1로 High를 보내주면 바뀌면서 LED 출력)



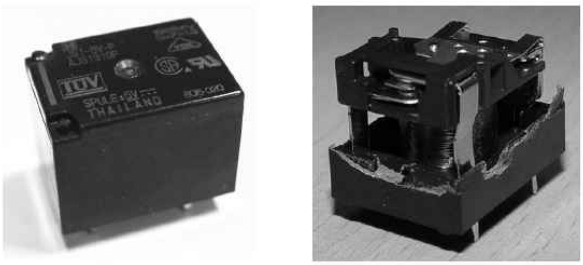
ex) LED로 1을 표현 시 PC0 = 0, PC6 = 0, 나머지 1 => B와 C Low, 나머지 High
=> Sink 전류가 흐름 (외부 인터페이스에서 5V가 공급전압이 되고 있어 무조건 5V이기 때문에 접지 되어 있는 MCU에서 LOW를 보내주면 바뀌면서 LED 출력)

NPN 트랜지스터에서 전압을 증폭시켜 트랜지스터에 높은 전압이 걸리면 LED가 어두워지고 낮은 전압이 걸리면 LED가 밝아짐 (전압강하 조절)
=> 트랜지스터를 사용하여 공급전압이 아무리 커져도 상관 X (큰 전류가 흐름)
=> 스위치를 사용하면 공급전압이 커지면 포화됨



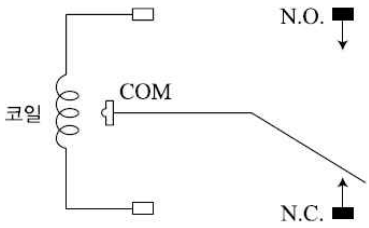
(a) 트랜지스터가 연결된 LED　　(b) 기계적인 스위치가 연결된 LED

릴레이 동작

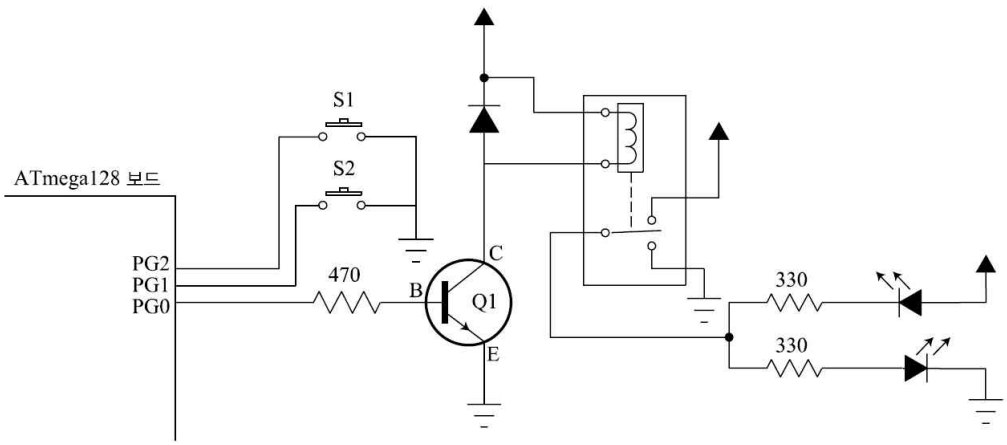


(a) 외관

(b) 내부 구조

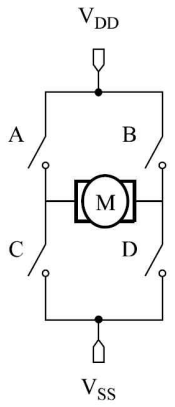


(c) 접점 구조

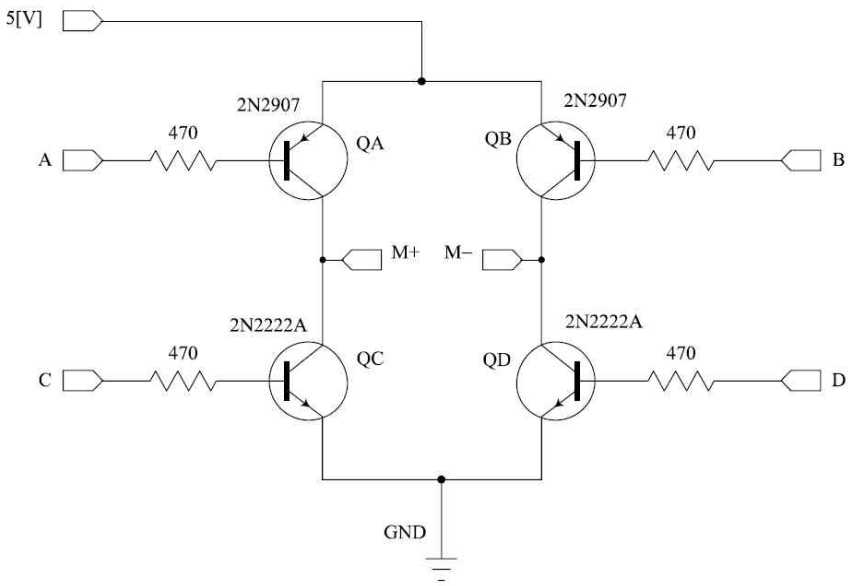


스위치 바운싱 현상 : 스위치를 여러번 누르게 되면 오동작 할 수 있기 때문에 SW적으로 스위치 디바운싱 역할을 함(스위치 변화를 처음 감지 후, 일정시간 동안 변화를 무시함

소형 모터 구동을 위한 H-브리지 회로



[그림 5-18] H-브리지 회로와 스위치



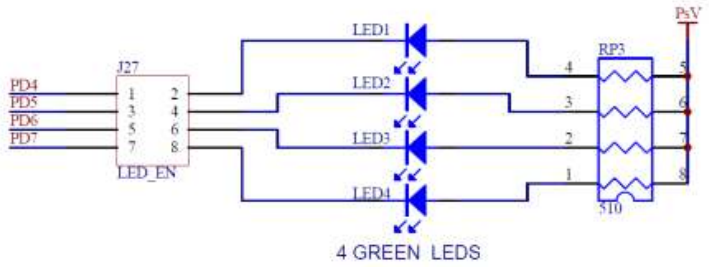
타이머/카운터의 역할

- 시간 측정 : 시간을 정밀하게 측정하여 시간 지연 및 정밀한 타이밍 요구사항 충족 시 중요
- 이벤트 타이밍 : 이벤트 간의 시간 간경을 측정하거나 이벤트의 타이밍을 조절하는데 사용
- 주기적 작업 : 주기적으로 반복되는 작업을 스케쥴링하는 주기적인 작업 수행에 사용
- 펄스 생성 : 정확한 주파수의 디지털 펄스 생성 시 사용
- 인터럽트 생성 : 일정 시간이 경과하면 인터럽트를 생성하여 프로세서에 중요한 이벤트를 알림
- 타이머/카운터 모드 : PWM 생성, 카운트 업/다운, 입력 캡처 등

GPIO : General-Purpose Input/Output

- 마이크로컨트롤러의 디지털 핀들을 입력, 출력으로 사용
- 디지털 신호를 처리하는데 사용

LED 회로



ıt

11.5.2.8 Pad Configuration Registers (PCR[0:107])

The Pad Configuration Registers allow configuration of the static electrical and functional characteristics associated with I/O pads. Each PCR controls the characteristics of a single pad.

Address: Base + 0x0040 (PCR0) Access: User read/write

...

Base + 0x0116 (PCR107) 108 registers

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SMC	APC	0	PA[1:0]		OBE	IBE	0	0	ODE	0	0	SRC	WPE	WPS
W																
Reset[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-10. Pad Configuration Registers 0–107 (PCR[0:107])

[1] See Table 11-11.

NOTE

Table 11-10. PCR[0:107] field descriptions

Field	Description
PA[1:0]	Pad Output Assignment This field selects the function that is allowed to drive the output of a multiplexed pad. The PA field size can vary from 0 to 2 bits, depending on the number of output functions associated with this pad. 00: Alternative mode 0: GPIO 01: Alternative mode 1 (see Chapter 3, "Signal Description) 10: Alternative mode 2 (see Chapter 3, "Signal Description) 11: Alternative mode 3 (see Chapter 3, "Signal Description) **Note:** The number of bits in the PA bitfield depends of the number of actual alternate functions provided for each pad. Please see the *MPC5604P Datasheet (MPC5604P)*.
OBE	Output Buffer Enable This bit enables the output buffer of the pad in case the pad is in GPIO mode. 0: Output buffer of the pad disabled when PA = 00 1: Output buffer of the pad enabled when PA = 00
IBE	Input Buffer Enable This bit enables the input buffer of the pad. 0: Input buffer of the pad disabled 1: Input buffer of the pad enabled

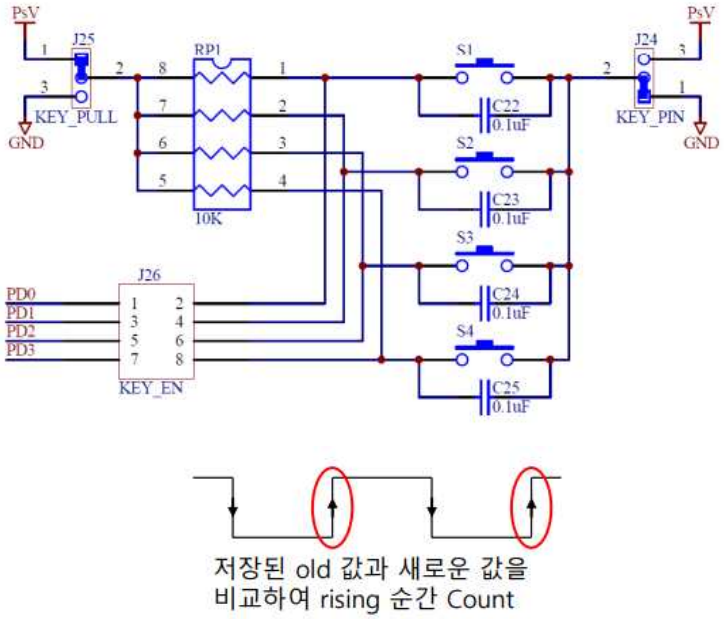
GPIO의 기능 설정

Output 설정

Input 설정

- LED로 사용할 4개의 핀을 Output으로 설정함

Switch 회로



ADC : Analog to Digital Converter (아날로그(전압) 신호를 디지털 신호로 변환)

- Resolution : 10bit

ADC 초기화

24.4.2.1 Main Configuration Register (MCR)

The Main Configuration Register (MCR) provides configuration settings for the ADC.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OWREN	WLSIDE	MODE	0	0	0	0	NSTART	0	JTRGEN	JEDGE	JSTART	0	0	CTUEN	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	ADCLKSEL	ABORTCHAIN	ABORT	ACKO	0	0	0	0	PWDN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 24-8. Main Configuration Register (MCR)

Table 24-12. MCR field descriptions

Field	Description
OWREN	Overwrite enable This bit enables or disables the functionality to overwrite unread converted data. 0 Prevents overwrite of unread converted data; new result is discarded 1 Enables converted data to be overwritten by a new conversion
WLSIDE	Write left/right-aligned 0 The conversion data is written right-aligned. 1 Data is left-aligned (from 15 to (15 – resolution + 1)). The WLSIDE bit affects all the CDR registers simultaneously. See Figure 24-23 and Figure 24-23.
MODE	One Shot/Scan 0 One Shot Mode—Configures the normal conversion of one chain. 1 Scan Mode—Configures continuous chain conversion mode; when the programmed chain conversion is finished it restarts immediately.
JTRGEN	Injection external trigger enable 0 External trigger disabled for channel injection 1 External trigger enabled for channel injection
JEDGE	Injection trigger edge selection Edge selection for external trigger, if JTRGEN = 1. 0 Selects falling edge for the external trigger 1 Selects rising edge for the external trigger
JSTART	Injection start Setting this bit will start the configured injected analog channels to be converted by software. Resetting this bit has no effect, as the injected chain conversion cannot be interrupted.
CTUEN	Cross trigger unit conversion enable 0 CTU triggered conversion disabled 1 CTU triggered conversion enabled
ADCLKSEL	Analog clock select This bit can only be written when ADC in Power-Down mode 0 ADC clock frequency is half Peripheral Set Clock frequency 1 ADC clock frequency is equal to Peripheral Set Clock frequency
ABORTCHAIN	Abort Chain When this bit is set, the ongoing Chain Conversion is aborted. This bit is reset by hardware as soon as a new conversion is requested. 0 Conversion is not affected 1 Aborts the ongoing chain conversion
ABORT	Abort Conversion When this bit is set, the ongoing conversion is aborted and a new conversion is invoked. This bit is reset by hardware as soon as a new conversion is invoked. If it is set during a scan chain, only the ongoing conversion is aborted and the next conversion is performed as planned. 0 Conversion is not affected 1 Aborts the ongoing conversion
ACKO	Auto-clock-off enable If set, this bit enables the Auto clock off feature. 0 Auto clock off disabled 1 Auto clock off enabled
PWDN	Power-down enable When this bit is set, the analog module is requested to enter Power Down mode. When ADC status is PWDN, resetting this bit starts ADC transition to IDLE mode. 0 ADC is in normal mode 1 ADC has been requested to power down

ADC sampling 수에 따른 변환 시간 설정

24.4.6 Conversion timing registers CTR[0]

CTR0 = associated to internal precision channels (from 0 to 15)

Address: Base + 0x0094 (CTR0) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INPLATCH	0	OFFSHIFT		0	INPCMP		0	INPSAMP							
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1

Figure 24-19. Conversion timing registers CTR[0]

Field	Description
INPLATCH	Configuration bit for latching phase duration
OFFSHIFT	Configuration for offset shift characteristic 00 No shift (that is the transition between codes 000h and 001h) is reached when the A_{VIN} (analog input voltage) is equal to 1 LSB. 01 Transition between code 000h and 001h is reached when the A_{VIN} is equal to1/2 LSB 10 Transition between code 00h and 001h is reached when the A_{VIN} is equal to 0 11 Not used **Note:** Available only on CTR0
INPCMP	Configuration bits for comparison phase duration
INPSAMP	Configuration bits for sampling phase duration

ADC Channel 사용 설정

24.4.7.2 Normal Conversion Mask Registers (NCMR[0])

NCMR0 = Enable bits of normal sampling for channel 0 to 15 (precision channels)

Address: Base + 0x00A4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-20. Normal Conversion Mask Register 0 (NCMR0)

ADC 데이터 저장 변수 초기화

24.4.9.2 Channel Data Register (CDR[0..15])

CDR[0..15] = precision channels

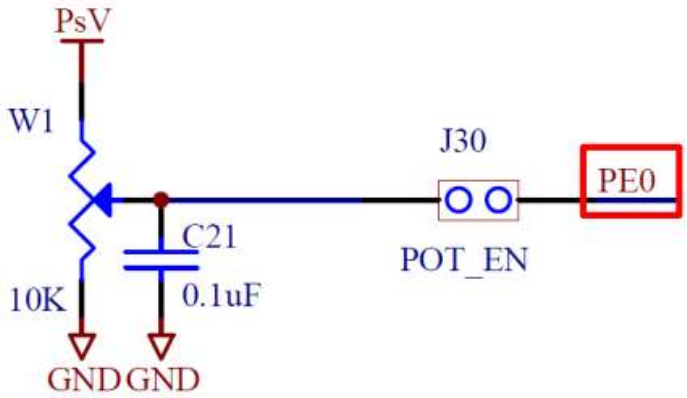
Each data register also gives information regarding the corresponding result as described below.

Address: See Table 24-10 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	VALID	OVERW	RESULT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	CDATA[0:9] (MCR[WLSIDE] = 0)									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

가변 저항 회로



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SMC	APC	0	PA[1:0]		OBE	IBE	0	0	ODE	0	0	SRC	WPE	WPS
W																
Reset[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Port pin	Pad configuration register (PCR)	Alternate function[1,2]	Functions	Peripheral[3]	I/O direction[4]	Pad speed[5]		Pin No.	
						SRC = 0	SRC = 1	100-pin	144-pin
E[0]	PCR[64]	ALT0 ALT1 ALT2 ALT3 —	GPIO[64] — — — AN[5]	SIUL — — — ADC_1	Input only	—	—	46	68

Interrupt

- Interrupt : 어떤 조건 하에서 조건을 만족하면 Flag를 띄워 사건을 처리
  ex) PWM Interrput
- PIT : 일정 주기를 가지고 실행되는 Interrupt (Periodic Interrupt Timer)
  * 설정 주기마다 발생
- 모든 주변장치는 clock을 사용하여 운영

PIT Enable Resister Setting

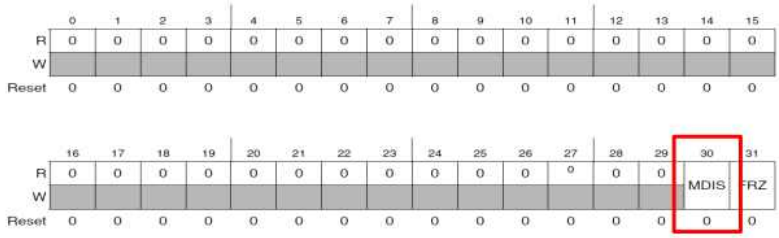


Figure 36-2. PIT Module Control Registers (PITMCR)

Table 36-3. PITMCR Field Descriptions

Field	Description
MDIS	Module Disable. This is used to disable the module clock. This bit should be enabled before any other setup is done. 0 Clock for PIT Timers is enabled (default) 1 Clock for PIT Timers is disabled
FRZ	Freeze. Allows the timers to be stopped when the device enters debug mode. 0 = Timers continue to run in debug mode. 1 = Timers are stopped in debug mode.

PIT Period Resister Setting

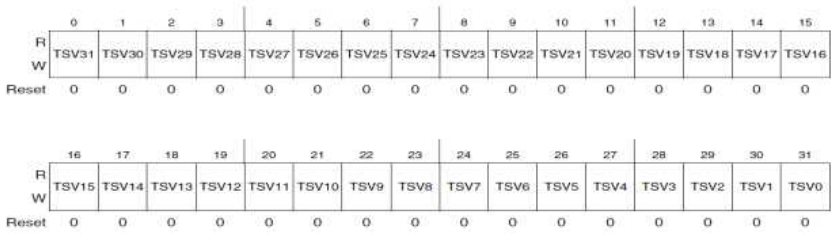


Figure 36-3. Timer Load Value Register (LDVAL)

Table 36-4. LDVAL Field Descriptions

Field	Description
TSV*n*	Time Start Value Bits. These bits set the timer start value. The timer will count down until it reaches 0, then it will generate an interrupt and load this register value again. Writing a new value to this register will not restart the timer, instead the value will be loaded once the timer expires. To abort the current cycle and start a timer period with the new value, the timer must be disabled and enabled again (see Figure 36-8).

- Resister에 설정된 Value에서 시스템 Clock의 주기로 0까지 Count down
- 이후 Interrupt 발생 후 다시 원래 Value로 돌아가서 반복

$$\text{PIT Period} = \frac{1}{SystemClock} * LDVAL\ Resister\ Value$$

Example)

System Clock = 64Mhz

PIT.CH[0].LDVAL.R = 6400; // $\frac{1}{SystemClock} * 6400 = 100us$

PIT.CH[1].LDVAL.R = 32000; // $\frac{1}{SystemClock} * 32000 = 500us$

PIT.CH[2].LDVAL.R = 64000; // $\frac{1}{SystemClock} * 64000 = 1ms$

PIT 각각의 Timer Enable 및 Start Resister Setting

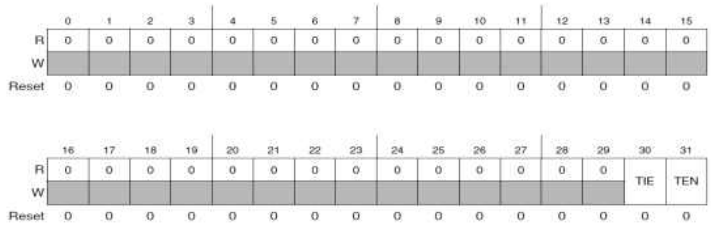


Figure 36-5. Timer Control Register (TCTRL)

Table 36-6. TCTRL Field Descriptions

Field	Description
TIE	Timer Interrupt Enable Bit. 0 Interrupt requests from Timer x are disabled 1 Interrupt will be requested whenever TIF is set When an interrupt is pending (TIF set), enabling the interrupt will immediately cause an interrupt event. To avoid this, the associated TIF flag must be cleared first.
TEN	Timer Enable Bit. 0 Timer will be disabled 1 Timer will be active

- TIE BIT : 각각의 Timer의 Interrupt enable
- TEN BIT : 각각의 Timer의 Timer start

우선순위

59	0x03B0	16	PITimer Channel 0	PIT	PIT
60	0x03C0	16	PITimer Channel 1	PIT	PIT
61	0x03D0	16	PITimer Channel 2	PIT	PIT
62	0x03E0	16	ADC_EOC	ADC_0	ADC_0

179	0x0B30	16	RF0	FlexPWM_0
180	0x0B40	16	COF0	FlexPWM_0
181	0x0B50	16	CAF0	FlexPWM_0
182	0x0B60	16	RF1	FlexPWM_0
183	0x0B70	16	COF1	FlexPWM_0
184	0x0B80	16	CAF1	FlexPWM_0
185	0x0B90	16	RF2	FlexPWM_0

PWM Interrupt

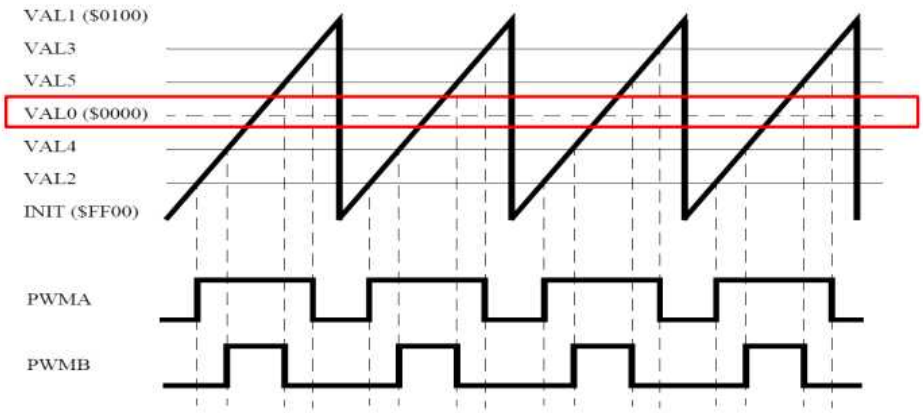
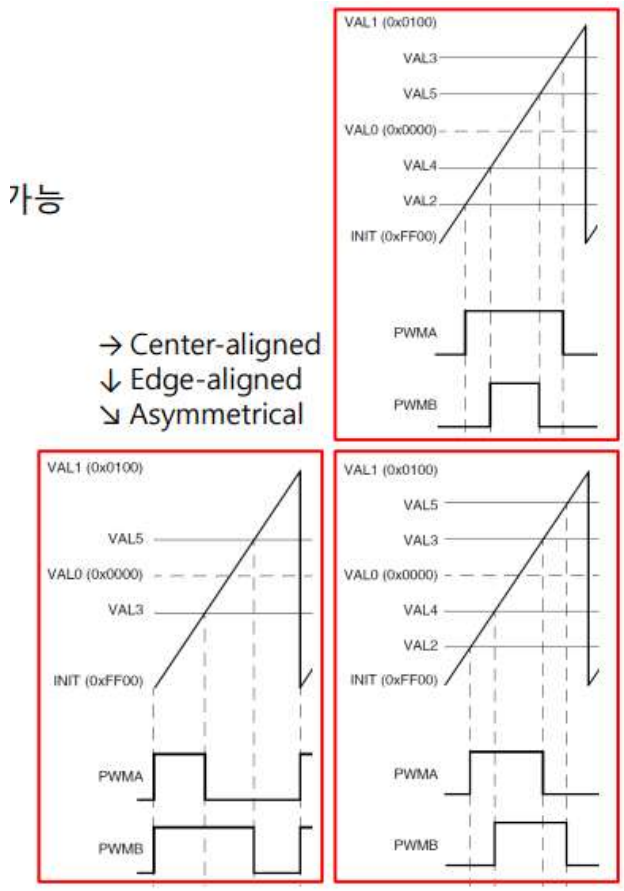
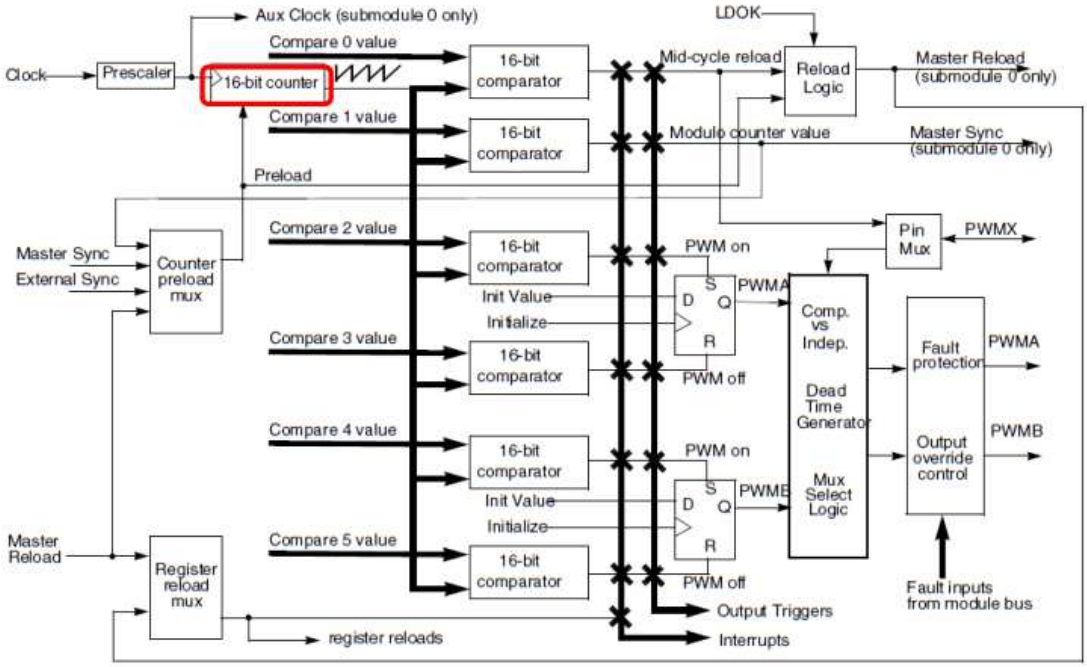


Figure 25-3. Center Aligned Example

PWM

- FlexPWM
  - * Resolution : 16bit, 각각의 PWM 출력의 모든 Edge에 대해 독립제어
  - * PWMX는 Capture기능





General Configuration

2.1.1 각 모듈 A, B, X 출력 설정

26.6.4.1 Output Enable register (OUTEN)

Figure 26-27. Output Enable register (OUTEN)

3번 모듈 | 2번 모듈 | 1번 모듈 | 0번 모듈

```
FLEXPWM_0.OUTEN.B.PWMA_EN = 0b0111;    // PWM A Output Enabled Submodule 0:2
FLEXPWM_0.OUTEN.B.PWMB_EN = 0b0111;    // PWM B Output Enabled Submodule 0:2
FLEXPWM_0.OUTEN.B.PWMX_EN = 0b0000;    // PWM X Output Disable
```

- 0 : 출력 X, 1 : 출력 O

2.1.2 각 모듈 A, B, X 출력 Mask

26.6.4.2 Mask register (MASK)

※ 설정시 강제로 PWM 출력을 0으로 만듦

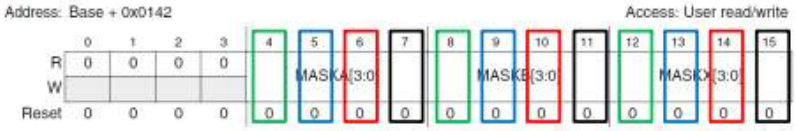


Figure 26-28. Mask register (MASK)

3번 모듈 | 2번 모듈 | 1번 모듈 | 0번 모듈

```
FLEXPWM_0.MASK.R = 0x0000;             // PWM All Mask Disable
```

2.1.3 각 모듈 A, B Software 출력 설정 ※ DTSRCSEL의 SELA/B가 0b10으로 되어있을 때 사용

26.6.4.3 Software Controlled Output Register (SWCOUT)

Address: Base + 0x0144 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	OUTA_3	OUTB_3	OUTA_2	OUTB_2	OUTA_1	OUTB_1	OUTA_0	OUTB_0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-29. Software Controlled Output Register (SWCOUT)

```
FLEXPWM_0.SWCOUT.B.OUTA_3 = 0;          // S/W Controlled O/P OUTA_3 Disable
FLEXPWM_0.SWCOUT.B.OUTB_3 = 0;          // S/W Controlled O/P OUTB_3 Disable
FLEXPWM_0.SWCOUT.B.OUTA_2 = 1;          // S/W Controlled O/P OUTA_2 Enable
FLEXPWM_0.SWCOUT.B.OUTB_2 = 1;          // S/W Controlled O/P OUTB_2 Enable
FLEXPWM_0.SWCOUT.B.OUTA_1 = 1;          // S/W Controlled O/P OUTA_1 Enable
FLEXPWM_0.SWCOUT.B.OUTB_1 = 1;          // S/W Controlled O/P OUTB_1 Enable
FLEXPWM_0.SWCOUT.B.OUTA_0 = 1;          // S/W Controlled O/P OUTA_0 Enable
FLEXPWM_0.SWCOUT.B.OUTB_0 = 1;          // S/W Controlled O/P OUTB_0 Enable
```

2.1.4 각 모듈 A, B Deadtime Source 설정

26.6.4.4 Deadtime Source Select Register (DTSRCSEL)

Address: Base + 0x0146 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SELA_3		SELB_3		SELA_2		SELB_2		SELA_1		SELB_1		SELA_0		SELB_0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-30. Deadtime Source Select Register (DTSRCSEL)

0b00 : Generated PWMA/B_x signal is used by the deadtime logic.
0b01 : Inverted generated PWMA/B_x signal is used by the deadtime logic.
0b10 : OUTA/B_x bit is used by the deadtime logic.

2.1.5 PWM 값 load ※ PWM count값 입력 후 load시 사용

26.6.4.5 Master Control Register (MCTRL)

Address: Base + 0x0148 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	IPOL				RUN				0	0	0	0	LDOK			
W									CLDOK							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-31. Master Control Register (MCTRL)

IPOL : A, B를 대칭으로 출력, 독립모드에서는 무시
RUN : PWM 발생 enable
CLDOK : LDOK를 clear
LDOK : PWM값을 load

```
FLEXPWM_0.MCTRL.B.LDOK |= 0xF;          // Load config values into buffers
FLEXPWM_0.MCTRL.B.RUN |=0xF;            //1, 2, 3, 4 RUN
```

Submodule Configuration

2.2.1 PWM count 초기 값

26.6.3.2 Initial Count Register (INIT)

Address: Base + 0x0002 (Submodule 0)
Base + 0x0052 (Submodule 1)
Base + 0x00A2 (Submodule 2)
Base + 0x00F2 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	INIT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-4. Initial Count Register (INIT)

```
FLEXPWM_0.SUB[0].INIT.R = 0xffff-FSW_CLOCK_PERIOD+1; //62336 in 10kHz,63936 in 20kHz
```

2.2.2 PWM count 중간 값

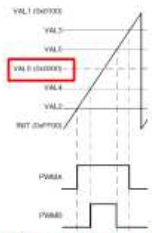
26.6.3.5 Value register 0 (VAL0)

Address: Base + 0x0008 (Submodule 0)
Base + 0x0058 (Submodule 1)
Base + 0x00A8 (Submodule 2)
Base + 0x00F8 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VAL0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-7. Value Register 0 (VAL0)



```
FLEXPWM_0.SUB[0].VAL[0].R = 0;                    //Mid-cycle Reload Point for the PWM in PWM clock periods
```

2.2.3 PWM count 최대 값

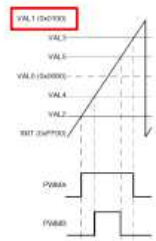
26.6.3.6 Value register 1 (VAL1)

Address: Base + 0x000A (Submodule 0)
Base + 0x005A (Submodule 1)
Base + 0x00AA (Submodule 2)
Base + 0x00FA (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VAL1															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-8. Value Register 1 (VAL1)



```
FLEXPWM_0.SUB[0].VAL[1].R = FSW_CLOCK_PERIOD;    // Modulo(Maximum) Count Value for the submodule counter
```

2.2.4 PWM_A 상승 값

26.6.3.7 Value register 2 (VAL2)

Address: Base + 0x000C (Submodule 0)
Base + 0x005C (Submodule 1)
Base + 0x00AC (Submodule 2)
Base + 0x00FC (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VAL2															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-9. Value register 2 (VAL2)

2.2.5 PWM_A 하강 값

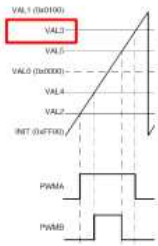
26.6.3.8 Value register 3 (VAL3)

Address: Base + 0x000E (Submodule 0)
Base + 0x005E (Submodule 1)
Base + 0x00AE (Submodule 2)
Base + 0x00FE (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL3															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-10. Value register 3 (VAL3)



2.2.6 PWM_B 상승 값

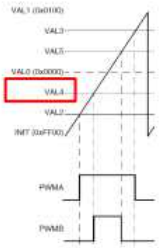
26.6.3.9 Value register 4 (VAL4)

Address: Base + 0x0010 (Submodule 0)
Base + 0x0060 (Submodule 1)
Base + 0x00B0 (Submodule 2)
Base + 0x0100 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL4															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-11. Value register 4 (VAL4)



2.2.7 PWM_B 하강 값

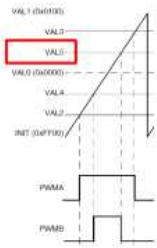
26.6.3.10 Value register 5 (VAL5)

Address: Base + 0x0012 (Submodule 0)
Base + 0x0062 (Submodule 1)
Base + 0x00B2 (Submodule 2)
Base + 0x0102 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL5															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-12. Value register 5 (VAL5)



2.2.8 Submodule Control 2

DBGEN : Debug 모드에서 동작 설정
WAITEN : WAIT/HALT 모드에서 동작 설정
INDEP : A와 B를 독립적/대칭으로 발생
PWMA_INIT : PWMA의 초기값
PWMB_INIT : PWMB의 초기값
PWMX_INIT : PWMX의 초기값
INIT_SEL : Counter로 가는 초기 Signal Source를 설정
FRCEN : FORCE_OUT 신호의 초기화
FORCE : FORCE_SEL이 000일 때 사용
FORCE_SEL : Force Source 선택
RELOAD_SEL : Reload Source 선택
CLK_SEL : Clock Source 선택

```
FLEXPWM_0.SUB[0].CTRL2.B.DBGEN       = 0;      // Debug Enable : Disable
FLEXPWM_0.SUB[0].CTRL2.B.WAITEN      = 0;      // Wait Enable : Disable
FLEXPWM_0.SUB[0].CTRL2.B.INDEP       = 1;      // Pair Operation : Independent
FLEXPWM_0.SUB[0].CTRL2.B.PWMA_INIT   = 0;      // PWMA Initial Value : 0
FLEXPWM_0.SUB[0].CTRL2.B.PWMB_INIT   = 0;      // PWMB Initial Value : 0
FLEXPWM_0.SUB[0].CTRL2.B.PWMX_INIT   = 0;      // PWMX Initial Value : 0
FLEXPWM_0.SUB[0].CTRL2.B.INIT_SEL    = 0b00;   // Initialization Control : Local sync
FLEXPWM_0.SUB[0].CTRL2.B.FRCEN       = 0;      // Force out event Initialization : Disable
FLEXPWM_0.SUB[0].CTRL2.B.FORCE       = 0;      // Force Initialization : Disable
FLEXPWM_0.SUB[0].CTRL2.B.FORCE_SEL   = 0b000;  // Force Source : Local force
FLEXPWM_0.SUB[0].CTRL2.B.RELOAD_SEL  = 0;      // Reload Source : Local
FLEXPWM_0.SUB[0].CTRL2.B.CLK_SEL     = 0b00;   // Clock Source : IPBus clock
```

2.2.9 Submodule Control 1

26.6.3.4 Control 1 Register (CTRL1)

Address: Base + 0x0006 (Submodule 0)
Base + 0x0056 (Submodule 1)
Base + 0x00A6 (Submodule 2)
Base + 0x00F6 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LDFQ				HALF	FULL	DT		0	PRSC			0	0	0	DBL EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-6. Control 1 Register (CTRL1)

LDFQ : 새로운 PWM 값을 반영하는 시기 설정
HALF : Half Cycle Reload(Val0에서 reload)
FULL : Full Cycle Reload(Val1에서 reload)
DT : Deadtime의 끝에서 PWMX의 샘플링 된 값을 보여준다
PRSC : PWM clock 주파수 설정
DBLEN : Double Switching 설정

```
FLEXPWM_0.SUB[0].CTRL.B.LDFQ   = 0b0000;   // Load Frequency : Every PWM
FLEXPWM_0.SUB[0].CTRL.B.HALF   = 1;        // Half Cycle Reload : Enable
FLEXPWM_0.SUB[0].CTRL.B.FULL   = 1;        // Full Cycle Reload : Enable
FLEXPWM_0.SUB[0].CTRL.B.DT     = 0b00;     // Deadtime from PWMX(DT[0], DT[1]) : None
FLEXPWM_0.SUB[0].CTRL.B.PRSC   = 0b000;    // Prescaler : fclk
FLEXPWM_0.SUB[0].CTRL.B.DBLEN  = 0;        // Double Switching : Disable
```

2.2.10 Submodule 출력 설정

26.6.3.11 Output Control register (OCTRL)

Address: Base + 0x0018 (Submodule 0)
Base + 0x0068 (Submodule 1)
Base + 0x00B8 (Submodule 2)
Base + 0x0108 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PWMA_IN	PWMB_IN	PWMX_IN	0	0	POLA	POLB	POLX	0	0	PWMAFS		PWMBFS		PWMXFS	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-13. Output Control register (OCTRL)

PWMA_IN : PWMA 입력으로 구동되는 Logic Value
PWMB_IN : PWMB 입력으로 구동되는 Logic Value
PWMX_IN : PWMX 입력으로 구동되는 Logic Value
POLA : PWMA의 출력 반전을 설정
POLB : PWMB의 출력 반전을 설정
POLX : PWMX의 출력 반전을 설정
PWMAFS : Fault 발생시 PWMA의 상태 설정
PWMBFS : Fault 발생시 PWMB의 상태 설정
PWMXFS : Fault 발생시 PWMX의 상태 설정

```
FLEXPWM_0.SUB[0].OCTRL.B.PWMA_IN = 0;       // PWM A Input Logic Value
FLEXPWM_0.SUB[0].OCTRL.B.PWMB_IN = 0;       // PWM B Input Logic Value
FLEXPWM_0.SUB[0].OCTRL.B.PWMX_IN = 0;       // PWM X Input Logic Value
FLEXPWM_0.SUB[0].OCTRL.B.POLA    = 0;       // PWM A Output Polarity : PWMA output not Inverted
FLEXPWM_0.SUB[0].OCTRL.B.POLB    = 0;       // PWM B Output Polarity : PWMB output not Inverted
FLEXPWM_0.SUB[0].OCTRL.B.POLX    = 0;       // PWM X Output Polarity : PWMX output not Inverted
FLEXPWM_0.SUB[0].OCTRL.B.PWMAFS  = 0b00;    // PWM A Fault State : Output is forced to logic 0
FLEXPWM_0.SUB[0].OCTRL.B.PWMBFS  = 0b00;    // PWM B Fault State : Output is forced to logic 0
FLEXPWM_0.SUB[0].OCTRL.B.PWMXFS  = 0b00;    // PWM X Fault State : Output is forced to logic 0
```

2.2.11 Fault Mask

26.6.3.16 Fault Disable Mapping register (DISMAP)

Address: Base + 0x0022 (Submodule 0)
Base + 0x0072 (Submodule 1)
Base + 0x00C2 (Submodule 2)
Base + 0x0112 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	1	1	1	1	DISX[3:0]				DISB[3:0]				DISA[3:0]			
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 26-18. Fault Disable Mapping register (DISMAP)

```
FLEXPWM_0.SUB[0].DISMAP.B.DISX = 0b1111;    // PWM X Fault Mask : Turn off
FLEXPWM_0.SUB[0].DISMAP.B.DISB = 0b0000;    // PWM B Fault Mask : Turn on
FLEXPWM_0.SUB[0].DISMAP.B.DISA = 0b0000;    // PWM A Fault Mask : Turn on
```

2.2.12 Deadtime

26.6.3.17 Deadtime Count registers (DTCNT0, DTCNT1)

Address: Base + 0x0024 (Submodule 0)
Base + 0x0074 (Submodule 1)
Base + 0x00C4 (Submodule 2)
Base + 0x0114 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	DTCNT0										
W																
Reset	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1

Figure 26-19. Deadtime Count Register 0 (DTCNT0)

Address: Base + 0x0026 (Submodule 0)
Base + 0x0076 (Submodule 1)
Base + 0x00C6 (Submodule 2)
Base + 0x0116 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	DTCNT1										
W																
Reset	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1

Figure 26-20. Deadtime Count register 1 (DTCNT1)

```
FLEXPWM_0.SUB[0].DTCNT0.R = DTIME_CLOCK_PERIOD;     // PWM A Deadtime Count
FLEXPWM_0.SUB[0].DTCNT1.R = DTIME_CLOCK_PERIOD;     // PWM B Deadtime Count
```

2.3 출력 Pin 설정

```
SIU.PCR[58].R = 0x0600;      // PWM A[0]
SIU.PCR[59].R = 0x0600;      // PWM B[1]
SIU.PCR[61].R = 0x0600;      // PWM A[1]
SIU.PCR[62].R = 0x0600;      // PWM B[1]
SIU.PCR[12].R = 0x0A00;      // PWM A[2]
SIU.PCR[13].R = 0x0A00;      // PWM B[2]
```

● PORT 세팅

- OUTEN : B[3] Enable
- MASK : 0000 -> Mask를 하지 않겠다
- LDOK : Load Start
- RUN : PWM Start
- INIT : 캐리어파의 시작점
- VAL[0] : 캐리어파의 중간값 (3200)
- VAL[1] : 캐리어파의 끝값 (6400)
- B.INDEP : B와 A를 독립적으로 사용
- HALF : PWM의 값을 Half에서 update
- FULL : PWM의 값을 Full에서 update
- DISB, DISA : Fault 시 데이터를 차단

● LED Switch

```
#include "MPC5604P_M26V.h"
#include "freemaster.h"
#include "init_base.h"

void RegisterO_set(void);
void RegisterI_set(void);
void LED_ctrl(void);
void Switch(void);

char cnt = 0;   // switch의 conut
char SW[4] = {0, 0, 0, 0};
char SWold[4] = {0, 0, 0, 0}; // switch의 이? 값


char LED[4] = {1, 1, 1, 1};

int main(void)
{
	initModesAndClock();
	disableWatchdog();
	enableIrq();
	initOutputClock();
	FMSTR_Init();
	init_INTC();
	init_Linflex0();

	RegisterO_set();
	RegisterI_set();

	/* Loop forever */
	for (;;)
	{
		FMSTR_Recorder();
		FMSTR_Poll();

		Switch();
		LED_ctrl();
	}
}
```

```
void RegisterO_set(void)
{
        SIU.PCR[52].R = 0x200; // LED1
        SIU.PCR[53].R = 0x200; // LED2
        SIU.PCR[54].R = 0x200; // LED3 //.B => Register의 특정 비트의 특정 위치
        SIU.PCR[55].B.OBE = 0b1; // LED4 //.R => Register all
}

void LED_ctrl(void)
{
        SIU.GPDO[52].B.PDO = LED[0]; //LED1; // LED1
        SIU.GPDO[53].B.PDO = LED[1]; //LED2; // LED2
        SIU.GPDO[54].B.PDO = LED[2]; //LED3; // LED3 //.B => Register의 특정 비트
의 특정 위치
        SIU.GPDO[55].B.PDO = LED[3]; //LED4; //.R => Register all
}


void RegisterI_set(void)
{
        SIU.PCR[48].R = 0x100; // LED1
        SIU.PCR[49].R = 0x100; // LED2
        SIU.PCR[50].R = 0x100; // LED3 //.B => Register의 특정 비트의 특정 위치
        SIU.PCR[51].B.IBE = 0b1; // LED4 //.R => Register all
}

void Switch(void)
{
        SWold[0] = SW[0];
        SWold[1] = SW[1];
        SWold[2] = SW[2];
        SWold[3] = SW[3];

        SW[0] = SIU.GPDI[48].B.PDI;
        SW[1] = SIU.GPDI[49].B.PDI;
        SW[2] = SIU.GPDI[50].B.PDI;
        SW[3] = SIU.GPDI[51].B.PDI;

        if(!SWold[0] && SW[0]) cnt += 1;
        if(!SWold[1] && SW[1]) cnt += 2;
```

```
        if(!SWold[2] && SW[2]) cnt += 4;
        if(!SWold[3] && SW[3]) cnt += 8;

        cnt %= 16;

        LED[0] = !(cnt&0x01);
        LED[1] = !(cnt&0x02);
        LED[2] = !(cnt&0x04);
        LED[3] = !(cnt&0x08);

}
```

● 가변저항 LED ON/OFF

```
#include "MPC5604P_M26V.h"
#include "freemaster.h"
#include "init_base.h"

void RegisterO_set(void);
void LED_ctrl(void);
void init_ADC1(void);
void ADCRead_1(void);

char SW[4] = {0, 0, 0, 0};
char SWold[4] = {0, 0, 0, 0}; // switch의 이? 값

char LED[4] = {1, 1, 1, 1};

int R_adc = 0;
int LED_dis = 0;

int main(void)
{
	initModesAndClock();
	disableWatchdog();
	enableIrq();
	initOutputClock();
	FMSTR_Init();

	init_INTC();
	init_Linflex0();

	init_ADC1();

	RegisterO_set();

	/* Loop forever */
	for (;;)
	{
		FMSTR_Recorder();
		FMSTR_Poll();
		ADCRead_1();
	}
```

```
}

void RegisterO_set(void)
{
        SIU.PCR[52].R = 0x0200; // LED1
        SIU.PCR[53].R = 0x0200; // LED2
        SIU.PCR[54].R = 0x0200; // LED3 //.B => Register의 특정 비트의 특정 위치
        SIU.PCR[55].B.OBE = 0b1; // LED4 //.R => Register all

        SIU.PCR[64].R = 0x2400; // LED2
}

void LED_ctrl(void)
{
        SIU.GPDO[52].B.PDO = LED[0]; //LED1; // LED1
        SIU.GPDO[53].B.PDO = LED[1]; //LED2; // LED2
        SIU.GPDO[54].B.PDO = LED[2]; //LED3; // LED3 //.B => Register의 특정 비트
의 특정 위치
        SIU.GPDO[55].B.PDO = LED[3]; //LED4; //.R => Register all
}

void init_ADC1(void)
{
        ADC_1.MCR.B.ABORT = 1;

        ADC_1.MCR.B.OWREN = 0; //disable overwritting
        ADC_1.MCR.B.WLSIDE = 0;
        ADC_1.MCR.B.MODE = 0;
        ADC_1.MCR.B.CTUEN = 0;
        ADC_1.MCR.B.ADCLKSEL = 0;
        ADC_1.MCR.B.ACK0 = 0;
        ADC_1.MCR.B.PWDN = 0;

        ADC_1.CTR[0].R = 0x00008208;
        ADC_1.NCMR[0].R = 0x00000020;
        ADC_1.CDR[0].R = 0x00000000;

        ADC_1.MCR.B.ABORT = 0; //Exit Abort ADC_1

}
```

```
void ADCRead_1(void)
{
        ADC_1.MCR.B.NSTART = 1;
        asm("nop");

        while(ADC_1.MCR.B.NSTART) asm("nop");
        R_adc = ADC_1.CDR[5].B.CDATA;

        LED_dis = R_adc>>6;

        LED[0] = !(LED_dis&0x08);
        LED[1] = !(LED_dis&0x04);
        LED[2] = !(LED_dis&0x02);
        LED[3] = !(LED_dis&0x01);

        LED_ctrl();
}
```

● PIT로 LED Enable

```
#include "MPC5604P_M26V.h"
#include "freemaster.h"
#include "init_base.h"

void RegisterO_set(void);
void LED_ctrl(void);
void init_PIT(void);
void PIT0ISR(void);
void PIT1ISR(void);

char LED[4] = {1, 1, 1, 1};

int LED_dis = 0;

uint32_t Pit0cnt = 0;
uint32_t Pit1cnt = 0;

int main(void)
{
	initModesAndClock();
	disableWatchdog();
	enableIrq();
	initOutputClock();
	FMSTR_Init();

	init_INTC();
	init_Linflex0();
	init_PIT();

	//INTC_InstallINTCInterruptHandler(PIT0ISR, 59, 6);
	INTC_InstallINTCInterruptHandler(PIT1ISR, 60, 6);

	RegisterO_set();

	/* Loop forever */
	for (;;)
	{
		FMSTR_Recorder();
		FMSTR_Poll();
```

```
                LED_ctrl();
        }
}

void RegisterO_set(void)
{
        SIU.PCR[52].R = 0x0200; // LED1
        SIU.PCR[53].R = 0x0200; // LED2
        SIU.PCR[54].R = 0x0200; // LED3 //.B => Register의 특정 비트의 특정 위치
        SIU.PCR[55].B.OBE = 0b1; // LED4 //.R => Register all
}

void LED_ctrl(void)
{
        SIU.GPDO[52].B.PDO = LED[0]; //LED1; // LED1
        SIU.GPDO[53].B.PDO = LED[1]; //LED2; // LED2
        SIU.GPDO[54].B.PDO = LED[2]; //LED3; // LED3 //.B => Register의 특정 비트
의 특정 위치
        SIU.GPDO[55].B.PDO = LED[3]; //LED4; //.R => Register all
}

void init_PIT(void)
{
        PIT.PITMCR.R = 0x00000001;
        PIT.CH[0].LDVAL.R = 64000000;
        PIT.CH[1].LDVAL.R = 64000000;  //1sec

        PIT.CH[0].TCTRL.R = 0x00000003;
        PIT.CH[1].TCTRL.R = 0x00000003;
}

void PIT0ISR(void)
{
        Pit0cnt++;

        PIT.CH[0].TFLG.B.TIF = 1; //추가하지 않으면 interrput가 계속 살아 있어서 통신
interrupt를 받지 못함

        LED_dis = Pit0cnt>>6;
```

```
	LED[0] = !(Pit0cnt&0x08);
	LED[1] = !(Pit0cnt&0x04);
	LED[2] = !(Pit0cnt&0x02);
	LED[3] = !(Pit0cnt&0x01);
}

void PIT1ISR(void)
{
	Pit1cnt++;

	PIT.CH[1].TFLG.B.TIF = 1; //추가하지 않으면 interrput가 계속 살아 있어서 통신 interrupt를 받지 못함

	LED_dis = Pit1cnt>>6;

	LED[0] = !(Pit1cnt%5>=1);
	LED[1] = !(Pit1cnt%5>=2);
	LED[2] = !(Pit1cnt%5>=3);
	LED[3] = !(Pit1cnt%5>=4);
}
```

● 가변저항을 읽어 PWM 디지털신호로 바꾸어 LED 밝기 조정

```
#include "MPC5604P_M26V.h"
#include "freemaster.h"
#include "init_base.h"


void RegisterO_set(void);
void LED_ctrl(void);
void init_ADC1(void);
void ADCRead_1(void);
void Init_FlexPWM(void);
void PWM_out(void);

char LED[4] = {1, 1, 1, 1};

int R_adc = 0;
int LED_dis = 0;

uint16_t PWM_LED1 = 6400;

int main(void)
{
	initModesAndClock();
	disableWatchdog();
	enableIrq();
	initOutputClock();
	FMSTR_Init();

	init_INTC();
	init_Linflex0();

	init_ADC1();
	Init_FlexPWM();

	RegisterO_set();

	/* Loop forever */
	for (;;)
	{
		FMSTR_Recorder();
```

```
                FMSTR_Poll();
                ADCRead_1();

                PWM_out();
        }
}

void RegisterO_set(void)
{
        SIU.PCR[52].R = 0x0E00; // LED1
        SIU.PCR[53].R = 0x0200; // LED2
        SIU.PCR[54].R = 0x0200; // LED3 //.B => Register의 특정 비트의 특정 위치
        SIU.PCR[55].B.OBE = 0b1; // LED4 //.R => Register all

        SIU.PCR[64].R = 0x2400; // LED2        //가변저항
}

void LED_ctrl(void)
{
        SIU.GPDO[52].B.PDO = LED[0]; //LED1; // LED1
        SIU.GPDO[53].B.PDO = LED[1]; //LED2; // LED2
        SIU.GPDO[54].B.PDO = LED[2]; //LED3; // LED3 //.B => Register의 특정 비트
의 특정 위치
        SIU.GPDO[55].B.PDO = LED[3]; //LED4; //.R => Register all
}

void init_ADC1(void)
{
        ADC_1.MCR.B.ABORT = 1;

        ADC_1.MCR.B.OWREN = 0; //disable overwritting
        ADC_1.MCR.B.WLSIDE = 0;
        ADC_1.MCR.B.MODE = 0;
        ADC_1.MCR.B.CTUEN = 0;
        ADC_1.MCR.B.ADCLKSEL = 0;
        ADC_1.MCR.B.ACK0 = 0;
        ADC_1.MCR.B.PWDN = 0;

        ADC_1.CTR[0].R = 0x00008208;
        ADC_1.NCMR[0].R = 0x00000020;
        ADC_1.CDR[0].R = 0x00000000;
```

```
        ADC_1.MCR.B.ABORT = 0; //Exit Abort ADC_1

}

void ADCRead_1(void)
{
        ADC_1.MCR.B.NSTART = 1;
        asm("nop");

        while(ADC_1.MCR.B.NSTART) asm("nop");
        R_adc = ADC_1.CDR[5].B.CDATA;
}


void Init_FlexPWM(void)
{
        //FLEXPWM_0.OUTEN.B.PWMA_EN = 0b1111;
        FLEXPWM_0.OUTEN.B.PWMB_EN = 0b1000;
        //FLEXPWM_0.OUTEN.B.PWMX_EN = 0b0000;

        FLEXPWM_0.MASK.R = 0x0000;

        //FLEXPWM_0.SWCOUT.B.OUTA_3 = 1;
        FLEXPWM_0.SWCOUT.B.OUTB_3 = 1;

        //FLEXPWM_0.SWCOUT.B.OUTA_2 = 1;
        //FLEXPWM_0.SWCOUT.B.OUTB_2 = 1;

        //FLEXPWM_0.SWCOUT.B.OUTA_1 = 1;
        //FLEXPWM_0.SWCOUT.B.OUTB_1 = 1;

        //FLEXPWM_0.SWCOUT.B.OUTA_0 = 1;
        //FLEXPWM_0.SWCOUT.B.OUTB_0 = 1;

        // module 활성화
        FLEXPWM_0.MCTRL.B.LDOK |= 0xF;
        FLEXPWM_0.MCTRL.B.RUN |= 0xF;

        //1kHZ로 update
        FLEXPWM_0.SUB[3].INIT.R = 0;
```

```
        FLEXPWM_0.SUB[3].VAL[0].R = 3200;
        FLEXPWM_0.SUB[3].VAL[1].R = 6400;

        //독립 제어
        FLEXPWM_0.SUB[3].CTRL2.B.INDEP = 1;

        //full, half enable
        FLEXPWM_0.SUB[3].CTRL.B.HALF = 1;
        FLEXPWM_0.SUB[3].CTRL.B.FULL = 1;

        FLEXPWM_0.SUB[3].DISMAP.B.DISB = 0;
        FLEXPWM_0.SUB[3].DISMAP.B.DISA = 0;

}

void PWM_out(void)
{
        PWM_LED1 = (unsigned short)(6400*R_adc/1023);
        FLEXPWM_0.SUB[3].VAL[4].R = 0; //4, 5번 관장 => PWMB
        FLEXPWM_0.SUB[3].VAL[5].R = PWM_LED1;

        FLEXPWM_0.MCTRL.B.LDOK |= 0xF;
        FLEXPWM_0.MCTRL.B.RUN |= 0xF;
        LED_ctrl();

}
```

● 아날로그 신호를 입력받아 PWM 출력을 내보내는 프로그램

- PWM인터럽터에서 AD변환(10bit)을 하고 결과값에 따라 PWM 듀티를 0~100%까지 출력.

```
#include "MPC5604P_M26V.h"
#include "freemaster.h"
#include "init_base.h"


void RegisterO_set(void);
void init_ADC1(void);
void ADCRead_1(void);
void Init_FlexPWM(void);
void PWM_out(void);
void PIT0ISR(void);
void PIT1ISR(void);
void init_PIT(void);

int R_adc = 0;
int Pit0cnt = 0;
int Pit1cnt = 0;
uint16_t PWM_X = 3200;


int main(void)
{
        initModesAndClock();
        disableWatchdog();
        enableIrq();
        initOutputClock();
        FMSTR_Init();

        init_INTC();
        init_Linflex0();
        init_PIT();

        init_ADC1();
        Init_FlexPWM();

        INTC_InstallINTCInterruptHandler(PIT0ISR, 59, 5);
        INTC_InstallINTCInterruptHandler(PIT1ISR, 60, 6);
```

```
        RegisterO_set();

        /* Loop forever */
        for (;;)
        {
                FMSTR_Recorder();
                FMSTR_Poll();
        }
}

void RegisterO_set(void)
{
   SIU.PCR[58].R = 0x0600; //PWM A[0]
   SIU.PCR[59].R = 0x0600; //PWM B[0]

   SIU.PCR[64].R = 0x2400; // LED2    //가변저항

}


void init_ADC1(void)
{
        ADC_1.MCR.B.ABORT = 1;

        ADC_1.MCR.B.OWREN = 0; //disable overwritting
        ADC_1.MCR.B.WLSIDE = 0;
        ADC_1.MCR.B.MODE = 0;
        ADC_1.MCR.B.CTUEN = 0;
        ADC_1.MCR.B.ADCLKSEL = 0;
        ADC_1.MCR.B.ACK0 = 0;
        ADC_1.MCR.B.PWDN = 0;

        ADC_1.CTR[0].R = 0x00008208;
        ADC_1.NCMR[0].R = 0x00000020;
        ADC_1.CDR[0].R = 0x00000000;

        ADC_1.MCR.B.ABORT = 0; //Exit Abort ADC_1

}
```

```
void ADCRead_1(void)
{
	ADC_1.MCR.B.NSTART = 1;
	asm("nop");

	while(ADC_1.MCR.B.NSTART) asm("nop");
	R_adc = ADC_1.CDR[5].B.CDATA;
}

void init_PIT(void)
{
	PIT.PITMCR.R = 0x00000001;
	PIT.CH[0].LDVAL.R = 64000;
	PIT.CH[1].LDVAL.R = 6400;

	PIT.CH[0].TCTRL.R = 0x00000003;
	PIT.CH[1].TCTRL.R = 0x00000003;
}

void PIT0ISR(void)
{
	Pit0cnt++;

	PIT.CH[0].TFLG.B.TIF = 1; //추가하지 않으면 interrput가 계속 살아 있어서 통신
interrupt를 받지 못함

	ADCRead_1();

}

void PIT1ISR(void)
{
	Pit1cnt++;
	PIT.CH[1].TFLG.B.TIF = 1; //추가하지 않으면 interrput가 계속 살아 있어서 통신
interrupt를 받지 못함
	PWM_out();
}

void Init_FlexPWM(void)
{
	FLEXPWM_0.OUTEN.B.PWMA_EN = 0b1000;
```

```
        FLEXPWM_0.OUTEN.B.PWMB_EN = 0b1000;
        FLEXPWM_0.SWCOUT.B.OUTA_3 = 1;
        FLEXPWM_0.SWCOUT.B.OUTB_3 = 1;

        // module 활성화
        FLEXPWM_0.MCTRL.B.LDOK |= 0xF;
        FLEXPWM_0.MCTRL.B.RUN |= 0xF;

        //독립 제어
        FLEXPWM_0.SUB[3].CTRL2.B.INDEP = 1;

        //full, half enable
        FLEXPWM_0.SUB[3].CTRL.B.HALF = 1;
        FLEXPWM_0.SUB[3].CTRL.B.FULL = 1;

        FLEXPWM_0.SUB[3].DISMAP.B.DISB = 0;
        FLEXPWM_0.SUB[3].DISMAP.B.DISA = 0;

        //1kHZ로 update
        FLEXPWM_0.SUB[3].INIT.R = 0xffff - 0x0C80 + 1; //3200
        FLEXPWM_0.SUB[3].VAL[0].R = 0;
        FLEXPWM_0.SUB[3].VAL[1].R = 0x0C80;
}

void PWM_out(void)
{
        PWM_X = (uint16_t)(3200*R_adc/1023);
        FLEXPWM_0.SUB[3].VAL[2].R = -PWM_X; //4, 5번 관장 => PWMB
        FLEXPWM_0.SUB[3].VAL[3].R = PWM_X;

        FLEXPWM_0.MCTRL.B.LDOK |= 0xF;
        FLEXPWM_0.MCTRL.B.RUN |= 0xF;

}
```