AutoCodeGen

# User Manual

Rivision 1.00

September 2013

# 목 차

# Ⅰ. 프로그램 소개

AutoCodeGen은 처음 Embedded System에서의 개발을 시작하거나 아직 초급 단계에서 구조를 이해하고 Firmware를 설계 / 개발하는데 어려움을 겪고 있는 사용자들을 위해 이를 손쉽게 해결해주기 위해 제작된 프로그램입니다.

우리 프로그램의 특징은 MS Windows를 기반으로 마우스 클릭 몇 번만으로 바로 응용이 가능한 기본 초기화 설정 및 API 함수 제공이 가능하여 누구나 어려움 없이 사용할 수 있도록 설계되었습니다.

또한, 초급 개발자를 목표로 하는 프로그램이기 때문에 생성된 결과가 ANSI-C를 기준으로 하는 문법을 사용하여 응용프로그램 개발자들도 사용법을 숙지하면 추가적인 학습 없이 일반적으로 응용프로그램을 개발하는 시간과 노력을 투자하여 Embedded System 환경에서의 새로운 결과물을 쉽게 얻어낼 수 있으며, AutoCodeGen에서 제공하는 모든 함수 및 초기화 내용은 .C 및 Header를 포함하여 Library 전체를 공개적으로 제공하기 때문에 초급 사용자가 이 프로그램을 통해 중급 개발자로 거듭날 수 있도록 개발 환경을 제공합니다.

우리 프로그램은 내부적인 구조가 추후 다양한 Chip Vendor의 Hardware Platform을 지원할 수 있도록 설계되어있으며, Embedded System에서의 공통적인 기능들( Data Acquisition and Quality Control, Motor Control 등 )에 대해서 동일한 방식으로 개발이 진행될 수 있도록 하여 사용자들이 다른 Platform으로의 이동에도 쉽게 적응 가능하도록 하였습니다.

초급 개발자 이외에도 사용자에 의해 설계된 초기화 내용과 함수 사용 내역에 대한 저장과 불러오기 기능을 제공하여 약간의 변동사항만 반영되는 반복적인 Firmware 설계에도 효율적인 작업 환경을 제공합니다.

다음은 AutoCodeGen의 대표적인 주요 특징들입니다.

- 초급 개발자들도 어려움 없이 Embedded System 환경에서 개발할 수 있습니다.
- 마우스 클릭 몇 번만으로 바로 응용이 가능한 초기화 및 API가 제공됩니다.
- 제공되는 모든 Library의 원본을 공개하고 있어 교육에 용이합니다.
- 다양한 Chip Vendor와 H/W Platform을 지원할 수 있는 구조로 구성되었습니다.
- 반복 작업 및 세부 응용작업에 효율적인 구조를 가지고 있습니다.

# Ⅱ. 프로그램 사용 환경

## 최소 요구 사양

- Windows XP SP2 32Bit 이상의 환경
- Target Board의 IDE(Integrated Development Environment)
- Target Board의 JTAG Debugger 및 Driver
- Visual Studio C++ 2008 Redistributable Package

## 권장 요구 사양

- Windows 7 32Bit 이상의 환경
- Target Board의 IDE(Integrated Development Environment)
- Target Board의 JTAG Debugger 및 Driver
- Visual Studio C++ 2010 Redistributable Package

# Ⅲ. 프로그램 구성

1 시작 화면

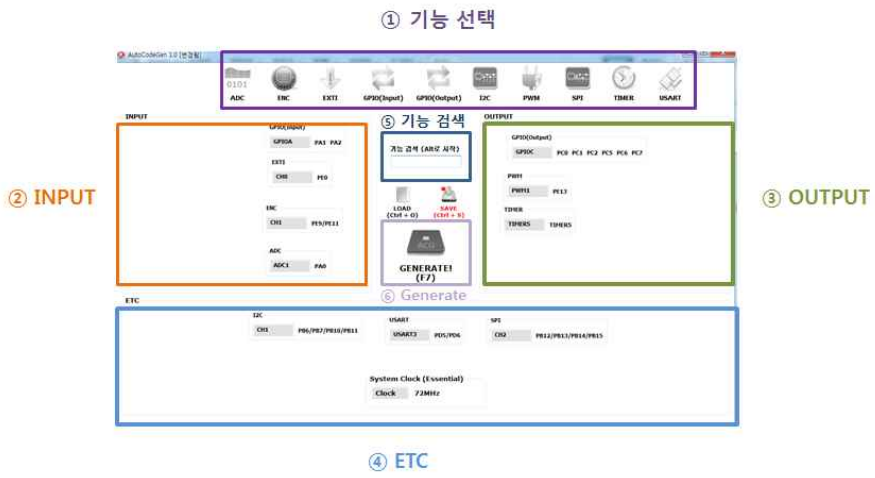

① Chip 종류 설정
- ARM, AVR, DSP 세가지 종류 중 한 가지를 선택

② Chip에 따른 Device 설정
- ARM : STM32F103
- AVR : ATmega128
- DSP : TR28335

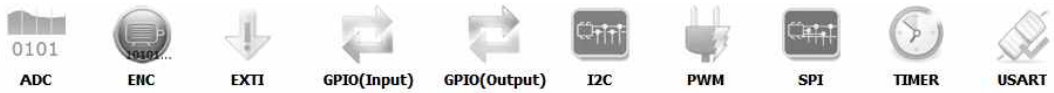
③ Project Name 설정
- 사용할 Project Name을 설정해 준다.

④ Project 생성

## 2 메인 화면




### ① 기능 선택




- 설정할 기능을 선택하여 레지스터를 설정 할 수 있다.

### ② INPUT

- MCU에 입력되는 기능들을 설정할 경우 INPUT 칸에 그림이 나타난다.
- 설정된 기능과 핀 정보를 간략하게 볼 수 있다.
- INPUT의 기능으로는 GPIO(Input), EXTI, ENC, ADC가 있다.

### ③ OUTPUT

- MCU에서 출력되는 기능들을 설정할 경우 OUTPUT 칸에 그림이 나타난다.
- 설정된 기능과 핀 정보를 간략하게 볼 수 있다.
- OUTPUT의 기능으로는 GPIO(Output), PWM, TIMER, ECAP(Dsp)이 있다.

### ④ ETC

- MCU에서 입출력을 제외한 기타 기능들을 설정할 경우 ETC 칸에 그림이 나타난다.
- 설정된 기능과 핀 정보를 간략하게 볼 수 있다.
- ETC의 기능으로는 I2C, USART, SPI가 있다.

### ⑤ 기능 검색

- 키보드로 키를 입력하여 기능을 검색 할 수 있다.
- 글자를 쳤을 때 해당하는 기능이 한 가지 밖에 없을 경우 자동으로 설정 화면으로 넘어간다.

### ⑥ Generate

- Generate를 누르면 설정한 레지스터들이 초기화된 소스파일이 자동으로 만들어 진다.

# Ⅳ. 지원되는 장비

## 1 Atmel AVR ATmega128

- 8Bit Microcontroller
- Up to 16MIPS Throughput at 16㎒
- 128KB of In-System Re-programmable Flash
- 4KB EEPROM / 4KB Internal SRAM
- Up to 64KB Optional External Memory Space

| Function | Max. Channel | Performance |
|---|---|---|
| ADC | 8 Channels | 10Bit |
| PWM | 8 Channels | |
| Timer | 4 Channels | Maximum 16Bit |
| External Interrupt | 8 Channels | |
| GPIO | 64 Pin | |
| USART | 2 Channels | |
| I2C | 1 Channels | |
| SPI | 1 Channels | |

## 2 STMicroelectronics ARM Cortex™-M3 (STM32F103x)

- 32Bit Microcontroller
- Up to 1.25 DMIPS Throughput at 72㎒
- 256 to 512KB of In-System Reprogrammable Flash
- Up to 64KB Internal SRAM
- Flexible Static Memory Controller

| Function | Max. Channel | Performance |
|---|---|---|
| ADC | 16 Channels | 12Bit |
| PWM | 28 Channels | |
| Timer | 6 Channels | Maximum 16Bit |
| External Interrupt | 16 Channels | |
| GPIO | 112 Pin | |
| USART | 5 Channels | |
| I2C | 2 Channels | |
| SPI | 3 Channels | |

## 3 Texas Instruments DSP TMS320F28335

- 32Bit Microcontroller
- Up to 150㎒ and FPU (Floating-Point Unit)
- 256K x 16 Internal SARAM
- 16/32Bit External Interface (XINTF)
- 8K x 16 Boot ROM for Software Boot-mode via SCI, SPI, CAN, I2C…

| Function | Max. Channel | Performance |
|---|---|---|
| ADC | 16 Channels | 12Bit |
| PWM | 12 Channels | |
| Timer | 3 Channels | Maximum 32Bit |
| External Interrupt | 7 Channels | |
| GPIO | 88 Pin | |
| USART | 2 Channels | |
| I2C | 1 Channels | |
| SPI | 1 Channels | |

# Ⅴ. 프로그램 사용 방법

## 1 System Clock (ACG_SYSCLK.h)

클록 펄스의 발생 주기. 컴퓨터 등 디지털 시스템에서 각 구성 요소의 모든 동작을 동기화하기 위해 사용되는 클록 펄스의 발생 주기를 클록 속도라고 하는데, 보통 메가헤르츠(MHz)로 표시하기 때문에 클록 주파수라고 한다. 1MHz는 초당 100만 주기의 속도를 나타낸다. 컴퓨터 시스템 클록(CPU 클록)의 클록 주파수는 컴퓨터의 전체적 처리 속도를 결정하는 주요한 요인의 하나이다. 주기억 장치나 디스크 구동 장치 등 다른 구성 요소가 CPU의 높은 주파수에 대응해야 하기 때문에, 무제한으로 클록 주파수를 높일 수는 없으나 CPU 자체의 처리 속도는 클록 주파수에 정비례한다. 예를 들어 클록 주파수가 20MHz인 CPU는 10MHz인 CPU에 비해 2배의 처리 속도를 갖는다. 최초의 IBM PC에 사용된 인텔 8088 마이크로프로세서의 클록 주파수는 약 5MHz였으나, 1995년에 발표된 펜티엄의 클록 주파수는 120MHz, 150MHz, 200MHz 등이며, 1998년에 발표된 펜티엄 Ⅱ의 클록 주파수는 330MHz, 350MHz, 400MHz 등으로 높아졌다.

### 1) ATmega128

Atmega128의 경우 외부크리스탈 16MHz를 받아 System Clock으로 사용하기 때문에 따로 설정해 주지 않는다.

### 2) STM32F103ZE

STM320F103ZE는 16MHz부터 72MHz까지 총 8가지로 System Clock을 설정할 수 있다.

#### API

| void AcgSystemClock_Setting(SysClk clk); |
|---|
| STM32F103ZE의 SystemClock을 초기화 하는 함수이다. |

| SysClk | |
|---|---|
| 이름 | 의미 |
| SYSCLK_16MHz | SystemClock을 16MHz로 설정합니다. |
| SYSCLK_24MHz | SystemClock을 24MHz로 설정합니다. |
| SYSCLK_32MHz | SystemClock을 32MHz로 설정합니다. |
| SYSCLK_40MHz | SystemClock을 40MHz로 설정합니다. |
| SYSCLK_48MHz | SystemClock을 48MHz로 설정합니다. |
| SYSCLK_56MHz | SystemClock을 56MHz로 설정합니다. |
| SYSCLK_64MHz | SystemClock을 64MHz로 설정합니다. |
| SYSCLK_72MHz | SystemClock을 72MHz로 설정합니다. |

### 3) TMS320F28335

| |
|---|
| TMS320F28335에서는 15MHz부터 150MHz 까지 총 10가지로<br>System Clock을 설정할 수 있다. |

### API

| |
|---|
| void AcgSystemClock_Setting(SysClk clk); |
| TMS320F28335의 SystemClock을 초기화 하는 함수이다. |

| |
|---|
| void Interrupt_Init(); |
| TMS320F28335의 Interrupt 설정을 초기화 하는 함수이다. |

| |
|---|
| void EnableGlobal_INT(); |
| TMS320F28335의 Global Interrupt핀을 Enable하는 함수이다.<br>모든 레지스터의 설정이 끝나고 이 함수를 써서 Global Interrupt를 Enable한다. |

| SysClk | |
|---|---|
| 이름 | 의미 |
| SYSCLK_15MHz | SystemClock을 15MHz로 설정합니다. |
| SYSCLK_30MHz | SystemClock을 30MHz로 설정합니다. |
| SYSCLK_45MHz | SystemClock을 45MHz로 설정합니다. |
| SYSCLK_60MHz | SystemClock을 60MHz로 설정합니다. |
| SYSCLK_75MHz | SystemClock을 75MHz로 설정합니다. |
| SYSCLK_90MHz | SystemClock을 90MHz로 설정합니다. |
| SYSCLK_105MHz | SystemClock을 105MHz로 설정합니다. |
| SYSCLK_120MHz | SystemClock을 120MHz로 설정합니다. |
| SYSCLK_135MHz | SystemClock을 135MHz로 설정합니다. |
| SYSCLK_150MHz | SystemClock을 150MHz로 설정합니다. |

## 2 ADC (ACG_ADC.h)

연속적인 신호인 아날로그 신호를 부호화된 디지털 신호로 변환하는 기계 장치. 실생활에서 연속적으로 측정되는 온도, 압력, 음성, 영상 신호, 전압 등의 신호를 컴퓨터에 입력하여 디지털화하는 장치이다.

### 1) ATmega128

| |
|---|
| ATmega128는 8채널 10Bit 분해능을 가진다.<br>PORTF에 8채널이 존재한다. |

### API

| |
|---|
| **void AcgADC_Setting();** |
| ATmega128의 ADC 설정을 초기화 해주는 함수이다.<br>주로 1/128의 분주비를 쓰므로 16MHz/128 = 125kHz의 Sampling Clock이 되고<br>이 값이 기본으로 설정되게 되어 있다. |
| **int AcgADC(ADCChannel ch);** |
| ADC 채널을 매개변수로 전달하면 그 채널에서 ADC 변환을 진행하여 int형 값으로 반환하는 함수이다.<br>사용자가 ADC 변환을 해서 값을 받고 싶은 부분에 이 함수를 사용해서 반환값으로 나온 값을 사용하면 된다.<br>**ex) while(1)문 안에 써서 ADC변환값을 계속 받아온다.** |

| ADCChannel | |
|---|---|
| 이름 | 의미 |
| ADC_00 | ADC 0번 채널 (PORTF0) |
| ADC_01 | ADC 1번 채널 (PORTF1) |
| ADC_02 | ADC 2번 채널 (PORTF2) |
| ADC_03 | ADC 3번 채널 (PORTF3) |
| ADC_04 | ADC 4번 채널 (PORTF4) |
| ADC_05 | ADC 5번 채널 (PORTF5) |
| ADC_06 | ADC 6번 채널 (PORTF6) |
| ADC_07 | ADC 7번 채널 (PORTF7) |

## 2) STM32F103ZE

| |
|---|
| STM32F103ZE에는 두 개의 ADC에 16채널 12Bit 분해능을 가진다.<br>두 개의 ADC 각 채널의 핀들은 같은 핀들을 사용한다. |

## API

| |
|---|
| **void AcgADCx_Setting();  (x = 1, 2)** |
| STM32F103ZE의 ADC 설정을 초기화 해주는 함수이다.<br>(System Clock)MHz/55 Sampling Clock이 기본값으로 설정되어 있다. |
| uint16_t AcgADCx(ADCChannel ch);  (x = 1, 2) |
| ADC 채널을 매개변수로 전달하면 그 채널에서 ADC 변환을 진행하여 uint16_t형으로 반환하는 함수이다.<br>사용자가 ADC 변환을 해서 값을 받고 싶은 부분에 이 함수를 사용해서 반환값으로 나온 값을 사용하면 된다.<br>**ex) while(1)문 안에 써서 ADC변환값을 계속 받아온다.** |

| ADCChannel | |
|---|---|
| 이름 | 의미 |
| ADC_00 | ADC 0번 채널 |
| ADC_01 | ADC 1번 채널 |
| ADC_02 | ADC 2번 채널 |
| ADC_03 | ADC 3번 채널 |
| ADC_04 | ADC 4번 채널 |
| ADC_05 | ADC 5번 채널 |
| ADC_06 | ADC 6번 채널 |
| ADC_07 | ADC 7번 채널 |
| ADC_08 | ADC 8번 채널 |
| ADC_09 | ADC 9번 채널 |
| ADC_10 | ADC 10번 채널 |
| ADC_11 | ADC 11번 채널 |
| ADC_12 | ADC 12번 채널 |
| ADC_13 | ADC 13번 채널 |
| ADC_14 | ADC 14번 채널 |
| ADC_15 | ADC 15번 채널 |

### 3) TMS320F28335

| |
|---|
| TMS320F28335는 16채널 12Bit 분해능을 가진다.<br>ADC 핀들이 다른 기능들과 독립적으로 존재한다.<br>ADCINA(0~7) , ADCINB(0~7) 으로 나누어져 있다. |

### API

| |
|---|
| void AcgADC_Setting(); |
| TMS320F28335의 ADC 설정을 초기화 해주는 함수이다.<br>기본적으로 6.25MHz로 ADC sampling clock이 설정되어있다. |
| Uint16 AcgADC(ADCChannel ch); |
| ADC 채널을 매개변수로 전달하면 그 채널에서 ADC 변환을 진행하여 Uint16형으로 반환하는 함수이다.<br>사용자가 ADC 변환을 해서 값을 받고 싶은 부분에 이 함수를 사용해서 반환값으로 나온 값을 사용하면 된다.<br>**ex) while(1)문 안에 써서 ADC변환값을 계속 받아온다.** |

| ADCChannel | |
|---|---|
| 이름 | 의미 |
| ADC_00 | ADC 0번 채널 (ADCINA0) |
| ADC_01 | ADC 1번 채널 (ADCINA1) |
| ADC_02 | ADC 2번 채널 (ADCINA2) |
| ADC_03 | ADC 3번 채널 (ADCINA3) |
| ADC_04 | ADC 4번 채널 (ADCINA4) |
| ADC_05 | ADC 5번 채널 (ADCINA5) |
| ADC_06 | ADC 6번 채널 (ADCINA6) |
| ADC_07 | ADC 7번 채널 (ADCINA7) |
| ADC_08 | ADC 8번 채널 (ADCINB0) |
| ADC_09 | ADC 9번 채널 (ADCINB1) |
| ADC_10 | ADC 10번 채널 (ADCINB2) |
| ADC_11 | ADC 11번 채널 (ADCINB3) |
| ADC_12 | ADC 12번 채널 (ADCINB4) |
| ADC_13 | ADC 13번 채널 (ADCINB5) |
| ADC_14 | ADC 14번 채널 (ADCINB6) |
| ADC_15 | ADC 15번 채널 (ADCINB7) |

## 3 EXTI (ACG_EXTI.h)

주변 장치의 읽기와 기록에 의해서 일어나는 인터럽트. 사용자가 키보드를 누름으로써 다음 동작으로 이동하거나 RS-232C 접속 장치(interface)로부터 보내온 데이터를 수용하는 등의 외부로부터 신호가 언제 들어올지 프로그램으로는 모르기 때문에 중앙 처리 장치가 일일이 감시하지 않아도 입력이 있으면 자동으로 프로그램의 실행을 일시 중단하고 외부 입력을 조사하여 그에 알맞은 처리를 하게 한 것.

### 1) ATmega128

ATmega128는 8개의 External Interrupt가 존재한다.
PORTD0 ~ PORTD3 , PORTE4 ~ PORTE7에 나누어져 있다.
Interrupt Trigger mode에는 Rising, Falling, Low, Toggle 4가지 방식이 있다.
Toggle 방식의 경우 PORTE4~ PORTE7에만 존재한다.

### API

**void AcgEXTI_Setting(EXTIChannel ch, TriggerType Trigger);**

ATmega128의 External Interrupt를 설정하고 초기화 해주는 함수이다.
EXTI 채널과 Trigger 방식을 매개변수로 받아 설정한다.

**void EXTI_0x_interrupt(); (x : 0~7)**

Interrupt handler 안에서 실행되는 함수로, Interrupt가 실행될 때 마다 이 함수가 실행이 된다.
사용자가 Interrupt가 실행될 때 수행하길 원하는 내용을 이 함수 안에 넣어주면 된다.
ACG_interrupt.handler.c 에 있다가 EXTI가 설정이 되면 그 설정된 부분의 함수만 main으로 빠져나가게 된다.

| EXTIChannel | |
|---|---|
| 이름 | 의미 |
| EXTI_00 | EXTI 0번 채널 (PORTD0) |
| EXTI_01 | EXTI 1번 채널 (PORTD1) |
| EXTI_02 | EXTI 2번 채널 (PORTD2) |
| EXTI_03 | EXTI 3번 채널 (PORTD3) |
| EXTI_04 | EXTI 4번 채널 (PORTF4) |
| EXTI_05 | EXTI 5번 채널 (PORTF5) |
| EXTI_06 | EXTI 6번 채널 (PORTF6) |
| EXTI_07 | EXTI 7번 채널 (PORTF7) |

| TriggerType | |
|---|---|
| 이름 | 의미 |
| Rising | EXTI핀의 Rising edge가 Interrupt를 트리거한다. |
| Falling | EXTI핀의 Falling edge가 Interrupt를 트리거한다. |
| Low | EXTI핀의 Low 신호 입력이 Interrupt를 트리거한다. |
| Toggle | EXTI핀의 Rising, Falling edge가 Interrupt를 트리거한다. |

## 2) STM32F103ZE

| |
|---|
| STM32F103ZE는 16개의 External Interrupt가 존재한다.<br>PORTF0 ~ PORTF15 사용한다. |

### API

| |
|---|
| void AcgEXTI_Setting(EXTIChannel ch, TriggerType Trigger); |
| STM32F103ZE의 External Interrupt를 설정하고 초기화 해주는 함수이다.<br>EXTI 채널과 Trigger 방식을 매개변수로 받아 설정한다. |
| void EXTI_x_interrupt(); (x : 0~15) |
| Interrupt handler 안에서 실행되는 함수로, Interrupt가 실행될 때 마다 이 함수가 실행이 된다.<br>사용자가 Interrupt가 실행될 때 수행하길 원하는 내용을 이 함수 안에 넣어주면 된다.<br>ACG_interrupt.handler.c 에 있다가 EXTI가 설정이 되면 그 설정된 부분의 함수만 main으로 빠져나가게 된다. |

| EXTIChannel | |
|---|---|
| 이름 | 의미 |
| EXTI_00 | EXTI 0번 채널 (PORTF0) |
| EXTI_01 | EXTI 1번 채널 (PORTF1) |
| EXTI_02 | EXTI 2번 채널 (PORTF2) |
| EXTI_03 | EXTI 3번 채널 (PORTF3) |
| EXTI_04 | EXTI 4번 채널 (PORTF4) |
| EXTI_05 | EXTI 5번 채널 (PORTF5) |
| EXTI_06 | EXTI 6번 채널 (PORTF6) |
| EXTI_07 | EXTI 7번 채널 (PORTF7) |
| EXTI_08 | EXTI 8번 채널 (PORTF8) |
| EXTI_09 | EXTI 9번 채널 (PORTF9) |
| EXTI_10 | EXTI 10번 채널 (PORTF10) |
| EXTI_11 | EXTI 11번 채널 (PORTF11) |
| EXTI_12 | EXTI 12번 채널 (PORTF12) |
| EXTI_13 | EXTI 13번 채널 (PORTF13) |
| EXTI_14 | EXTI 14번 채널 (PORTF14) |
| EXTI_15 | EXTI 15번 채널 (PORTF15) |

| TriggerType | |
|---|---|
| 이름 | 의미 |
| Rising | EXTI핀의 Rising edge가 Interrupt를 트리거한다. |
| Falling | EXTI핀의 Falling edge가 Interrupt를 트리거한다. |
| Toggle | EXTI핀의 Rising, Falling edge가 Interrupt를 트리거한다. |

### 3) TMS320F28335

| |
|---|
| TMS320F28335는 7채널의 External Interrupt를 가진다.<br>EXTIA 채널 (GPIO_00 ~ GPIO_31)중에 2개 EXTIB 채널 (GPIO_32 ~ GPIO_63)중에 5개를 임의로 골라 설정할 수 있다.<br>Interrupt Trigger mode에는 Rising, Falling 2가지 방식이 있다. |

### API

| |
|---|
| **void AcgEXTI_Setting(EXTIChannel ch, TriggerType trigger, Uint32 pin);** |
| ATmega128의 External Interrupt를 설정하고 초기화 해주는 함수이다.<br>EXTI 채널과 Trigger 방식과 EXTI로 사용할 핀을 매개변수로 받아 설정한다. |
| **void EXTI_0x_interrupt(); (x : 1~7)** |
| Interrupt handler 안에서 실행되는 함수로, Interrupt가 실행될 때 마다 이 함수가 실행이 된다.<br>사용자가 Interrupt가 실행될 때 수행하길 원하는 내용을 이 함수 안에 넣어주면 된다.<br>ACG_interrupt.handler.c 에 있다가 EXTI가 설정이 되면 그 설정된 부분의 함수만 main으로 빠져나가게 된다. |

| EXTIChannel | |
|---|---|
| 이름 | 의미 |
| EXTI_01 | EXTI 1번 채널 (GPIO_00 ~ GPIO_31 중 택 1) |
| EXTI_02 | EXTI 2번 채널 (GPIO_00 ~ GPIO_31 중 택 1) |
| EXTI_03 | EXTI 3번 채널 (GPIO_32 ~ GPIO_63 중 택 1) |
| EXTI_04 | EXTI 4번 채널 (GPIO_32 ~ GPIO_63 중 택 1) |
| EXTI_05 | EXTI 5번 채널 (GPIO_32 ~ GPIO_63 중 택 1) |
| EXTI_06 | EXTI 6번 채널 (GPIO_32 ~ GPIO_63 중 택 1) |
| EXTI_07 | EXTI 7번 채널 (GPIO_32 ~ GPIO_63 중 택 1) |

| TriggerType | |
|---|---|
| 이름 | 의미 |
| Rising | EXTI핀의 Rising edge가 Interrupt를 트리거한다. |
| Falling | EXTI핀의 Falling edge가 Interrupt를 트리거한다. |

## 4 GPIO (ACG_GPIO.h)

프로세서나 컨트롤러 등에서 일반 목적으로 사용하도록 준비된 입력, 또는 출력 포트. 다용도 입출력 포트(GPIO)를 소프트웨어와 연동시키면 전기적 입력을 받거나 출력으로 특정 디바이스를 제어하게 할 수 있다.

### 1) ATmega128

ATmega128는 PORTA ~ PORTG 까지 있으며 PORT별로 GPIO로 사용할 수 있는 8개의 핀이 있다. 단, PORTG에는 5개의 핀만이 존재한다.
따라서 ATmega128에는 총 53개의 GPIO핀이 있다.

### API

| |
|---|
| void AcgGPIO_OutputSetting(GPIOPort GPIOx ,char GPIO_Pin);<br>void AcgGPIO_InputSetting(GPIOPort GPIOx ,char GPIO_Pin); |
| ATmega128의 GPIO를 입력과 출력으로 설정해주는 함수이다.<br>매개변수로 GPIOPort와 GPIO_Pin을 입력받아 설정한다.<br>GPIOPort는 PORTA ~ PORTG까지고 각 채널별로 최대 8Bit가 있으므로<br>GPIO_Pin은 char형으로 16진수로 받게 된다. ex) 0xFF; (8Bit) |
| void AcgGPIO_OutPort(GPIOPort GPIOx,char GPIO_Pin); |
| GPIOx에 GPIO_Pin값을 출력하는 함수이다.<br>매개변수로 GPIOPort로 출력할 PORT를 받고<br>GPIO_Pin 값은 char형으로 16진수로 받는다.<br>사용자가 직접 원하는 곳에 함수를 호출해서 사용한다.<br>**ex) AcgGPIO_OutPort(GPIO_A, 0xFF); //PORTA에 모든 핀 출력** |
| void AcgGPIO_OutPin(GPIOPort GPIOx,int pin, int data); |
| pin을 한 개씩 지정하여 입력받은 data 값을 출력하는 함수이다.<br>매개변수로 GPIOPort로 출력할 PORT를 받고<br>int형 pin으로 PORT의 몇 번 핀(0~7)을 선택할지 정한다.<br>int형 data로 0 또는 1의 값을 받아서 1이면 출력, 0이면 출력하지 않는다.<br>사용자가 직접 원하는 곳에 함수를 호출해서 사용한다.<br>**ex) AcgGPIO_OutPin(GPIO_A,1,1); //PORTA의 1번 핀을 출력** |
| char AcgGPIO_InPort(GPIOPort GPIOx); |
| GPIOPort로 입력 받을 PORT를 받고, 그 PORT의 입력되는 값을<br>char형으로 반환하는 함수이다. PORT의 전체 입력되는 핀 값이 반환되기 때문에<br>주로 Masking을 해서 사용한다.<br>사용자가 직접 원하는 곳에 함수를 호출해서 사용한다.<br><br>**ex) if( AcgGPIO_InPort(GPIO_A) & 0x01);** |

| //PORTA의 입력되는 값을 0번 핀과 Masking한다. 즉, PORTA에 0번 핀에 값이 입력 //되는지를 확인한다. |
|---|
| int AcgGPIO_InPin(GPIOPort GPIOx,int pin); |
| GPIOPort에 입력받은 PORT의 한 개 pin(0~7 중 택 1)의 입력으로 들어오는 값을 반환하는 함수이다. 그 pin에 값이 입력되면 1을 반환, 입력되지 않으면 0을 반환한다.<br>사용자가 직접 원하는 곳에 함수를 호출해서 사용한다.<br><br>ex) if( AcgGPIO_InPin(GPIO_A, 0)); //PORTA의 0번 핀에 값이 입력되는지 확인 |

| GPIOPort | |
|---|---|
| 이름 | 의미 |
| GPIO_A | PORTA (0~7) |
| GPIO_B | PORTB (0~7) |
| GPIO_C | PORTC (0~7) |
| GPIO_D | PORTD (0~7) |
| GPIO_E | PORTE (0~7) |
| GPIO_F | PORTF (0~7) |
| GPIO_G | PORTG (0~4) |

## 2) STM32F103ZE

STM32F103ZE는 PORTA ~ PORTG 까지 있으며 PORT별로 GPIO로 사용할 수 있는 15개의 핀이 있다.
따라서 STM32F103ZE에는 총 104개의 GPIO핀이 있다.

## API

| |
|---|
| void AcgGPIO_OutputSetting(GpioPort GPIOx ,uint16_t GPIO_Pin);<br>void AcgGPIO_InputSetting(GpioPort GPIOx ,uint16_t GPIO_Pin); |
| STM32F103ZE의 GPIO를 입력과 출력으로 설정해주는 함수이다.<br>매개변수로 GPIOPort와 GPIO_Pin을 입력받아 설정한다.<br>GPIOPort는 PORTA ~ PORTG까지고 각 채널별로 최대 16Bit가 있으므로<br>GPIO_Pin은 **uint16_t**형으로 16진수로 받게 된다. ex) 0xFFFF; (16Bit) |
| void AcgGPIO_OutPort(GpioPort GPIOx, uint16_t GPIO_Pin); |
| GPIOx에 GPIO_Pin값을 출력하는 함수이다.<br>매개변수로 GPIOPort로 출력할 PORT를 받고<br>GPIO_Pin 값은 uint16_t형으로 16진수로 받는다.<br>사용자가 직접 원하는 곳에 함수를 호출해서 사용한다.<br>**ex) AcgGPIO_OutPort(GPIO_A, 0xFFFF); //PORTA에 모든 핀 출력** |
| void AcgGPIO_OutPin(GpioPort GPIOx, uint16_t pin, uint8_t data); |
| pin을 한 개씩 지정하여 입력받은 data 값을 출력하는 함수이다.<br>매개변수로 GPIOPort로 출력할 PORT를 받고<br>uint16_t형 pin으로 PORT의 몇 번 핀(0~15)을 선택할지 정한다.<br>uint8_t형 data로 0 또는 1의 값을 받아서 1이면 출력, 0이면 출력하지 않는다.<br>사용자가 직접 원하는 곳에 함수를 호출해서 사용한다.<br>**ex) AcgGPIO_OutPin(GPIO_A,1,1); //PORTA의 1번 핀을 출력** |
| uint16_t AcgGPIO_InPort(GpioPort GPIOx); |
| GPIOPort로 입력 받을 PORT를 받고, 그 PORT의 입력되는 값을<br>char형으로 반환하는 함수이다. PORT의 전체 입력되는 핀 값이 반환되기 때문에<br>주로 Masking을 해서 사용한다.<br>사용자가 직접 원하는 곳에 함수를 호출해서 사용한다.<br><br>**ex) if( AcgGPIO_InPort(GPIO_A) & 0x01);**<br>**//PORTA의 입력되는 값을 0번 핀과 Masking한다. 즉, PORTA에 0번 핀에 값이 입력**<br>**//되는지를 확인한다.** |
| uint8_t AcgGPIO_InPin(GpioPort GPIOx,int pin); |
| GPIOPort에 입력받은 PORT의 한 개 pin(0~7 중 택 1)의 입력으로 들어오는<br>값을 반환하는 함수이다. 그 pin에 값이 입력되면 1을 반환, 입력되지 않으면 |

0을 반환한다.

사용자가 직접 원하는 곳에 함수를 호출해서 사용한다.

**ex) if( AcgGPIO_InPin(GPIO_A, 0)); //PORTA의 0번 핀에 값이 입력되는지 확인**

| GPIOPort | |
|---|---|
| 이름 | 의미 |
| GPIO_A | PORTA (0~15) |
| GPIO_B | PORTB (0~15) |
| GPIO_C | PORTC (0~15) |
| GPIO_D | PORTD (0~15) |
| GPIO_E | PORTE (0~15) |
| GPIO_F | PORTF (0~15) |
| GPIO_G | PORTG (0~15) |

### 3) TMS320F28335

TMS320F28335는 GPIO_00 ~ GPIO_87 까지 총 88개의 GPIO가 존재한다.
**GPIO_A** (GPIO_00 ~ GPIO_31), **GPIO_B** (GPIO_32 ~ GPIO_63)
**GPIO_C** (GPIO64 ~ GPIO_87) 세 개의 채널로 나누어져 있다.

### API

```
void AcgGPIO_OutputSetting(GPIOPort GPIOx, unsinged long GPIO_Pin);
void AcgGPIO_InputSetting(GPIOPort GPIOx, unsigned long GPIO_Pin);
```

TMS320F28335의 GPIO를 입력과 출력으로 설정해주는 함수이다.
매개변수로 GPIOPort와 GPIO_Pin을 입력받아 설정한다.
GPIOPort는 GPIO_A ~ GPIO_C까지고 각 채널별로 최대 32Bit가 있으므로
GPIO_Pin은 unsinged long형으로 16진수로 받게 된다. ex) 0xFF00FFFF; (32Bit)

```
void AcgGPIO_OutPort(GPIOPort GPIOx, unsigned long GPIO_Pin);
```

GPIOx에 GPIO_Pin값을 출력하는 함수이다.
매개변수로 GPIOPort로 출력할 PORT를 받고
GPIO_Pin 값은 unsinged long형으로 16진수로 받는다.
사용자가 직접 원하는 곳에 함수를 호출해서 사용한다.
**ex) AcgGPIO_OutPort(GPIO_A, 0xFFFFFFFF; //PORTA에 모든 핀 출력**

```
void AcgGPIO_OutPin(GPIOPort GPIOx,int pin, int data);
```

pin을 한 개씩 지정하여 입력받은 data 값을 출력하는 함수이다.
매개변수로 GPIOPort로 출력할 PORT를 받고
int형 pin으로 PORT의 몇 번 핀(0~31)을 선택할지 정한다.
int형 data로 0 또는 1의 값을 받아서 1이면 출력, 0이면 출력하지 않는다.
사용자가 직접 원하는 곳에 함수를 호출해서 사용한다.
**ex) AcgGPIO_OutPin(GPIO_A,1,1); //PORTA의 1번 핀을 출력**

```
unsigned long AcgGPIO_InPort(GPIOPort GPIOx);
```

GPIOPort로 입력 받을 PORT를 받고, 그 PORT의 입력되는 값을
unsigned long형으로 반환하는 함수이다. PORT의 전체 입력되는 핀 값이 반환되기 때문에 주로 Masking을 해서 사용한다.
사용자가 직접 원하는 곳에 함수를 호출해서 사용한다.

**ex) if( AcgGPIO_InPort(GPIO_A) & 0x00000001);**
**//PORTA의 입력되는 값을 0번 핀과 Masking한다. 즉, PORTA에 0번 핀에 값이 입력//되는지를 확인한다.**

```
int AcgGPIO_InPin(GPIOPort GPIOx,int pin);
```

GPIOPort에 입력받은 PORT의 한 개 pin(0~7 중 택 1)의 입력으로 들어오는
값을 반환하는 함수이다. 그 pin에 값이 입력되면 1을 반환, 입력되지 않으면

0을 반환한다.
사용자가 직접 원하는 곳에 함수를 호출해서 사용한다.

ex) if( AcgGPIO_InPin(GPIO_A, 0)); //PORTA의 0번 핀에 값이 입력되는지 확인

| GPIOPort | |
|---|---|
| 이름 | 의미 |
| GPIO_A | GPIO_00 ~ GPIO_31 |
| GPIO_B | GPIO_32 ~ GPIO_63 |
| GPIO_C | GPIO_64 ~ GPIO_87 |

# 5 USART (ACG_USART.h)

입출력 기기 혹은 데이터 회선과 처리 장치와의 사이에 있으며, 전이중 및 비트 직렬 방식으로 데이터 전송을 다루는 기능 단위. 보통 단일의 패키지로서 구성되는 LSI 회로이며, 타이프라이터와 같은 저속도의 입출력 기기에는 비동기식으로, CRT 단말과 같은 고속도의 입출력 기기에 대해서는 동기식으로 데이터의 송수신 제어나 그에 부수하는 기능을 수행한다.

## 1) ATmega128

ATmega128는 두 개의 USART 포트가 존재한다.
BaudRate는 8가지가 있고, 주로 9600을 많이 쓴다.
WordLength는 5 ~ 9까지 설정할 수 있다.

## API

```
void AcgUSARTx_Setting(BaudRate baudRate, WordLength wordLength, StopBit stopBit, ParityBit parity);  ( x :  0,1 )
```

ATmega128의 USART 통신을 위한 레지스터를 초기화 해주는 함수이다.
매개변수로 BaudRate는 전송속도를 설정해주고
WordLength는 한 번에 보낼 Data의 Bit수를 설정해 준다.
StopBit는 StopBit를 몇 Bit 사용할지 설정해 주고
ParityBit는 parityBit를 어떻게 사용할지 설정해 준다.

```
void AcgUSARTx_txc(char tx);   ( x : 0,1 )
```

사용자가 Data를 전송할 때 호출하여 사용하는 함수이다.
매개변수로 char형으로 Data를 받아 전송한다.
**ex) AcgUSART0_txc('A');  //문자 'A'를 송신한다.**

```
char AcgUSARTx_rxc();      ( x : 0,1 )
```

수신 받은 Data 값을 반환하는 함수이다.
char형으로 Data를 반환한다.
이 함수를 호출하면 수신 값을 받을 때 까지 기다린다.
**ex)  rxData = AcgUSART0_rxc();  //수신받은 값을 rxData에 넣는다.**

| BaudRate | |
|---|---|
| 이름 | 의미 |
| BAUDRATE_2400 | 1초에 2400 Bit를 전송합니다. |
| BAUDRATE_4800 | 1초에 4800 Bit를 전송합니다. |
| BAUDRATE_9600 | 1초에 9600 Bit를 전송합니다. |
| BAUDRATE_14400 | 1초에 14400 Bit를 전송합니다. |
| BAUDRATE_19200 | 1초에 19200 Bit를 전송합니다. |
| BAUDRATE_38400 | 1초에 38400 Bit를 전송합니다. |
| BAUDRATE_57600 | 1초에 57600 Bit를 전송합니다. |
| BAUDRATE_115200 | 1초에 115200 Bit를 전송합니다. |

| WordLength | |
|---|---|
| 이름 | 의미 |
| WORDLENGTH_5 | 5Bit 단위로 data를 전송합니다. |
| WORDLENGTH_6 | 6Bit 단위로 data를 전송합니다. |
| WORDLENGTH_7 | 7Bit 단위로 data를 전송합니다. |
| WORDLENGTH_8 | 8Bit 단위로 data를 전송합니다. |
| WORDLENGTH_9 | 9Bit 단위로 data를 전송합니다. |

| StopBit | |
|---|---|
| 이름 | 의미 |
| STOPBIT_1 | StopBit를 1로 설정합니다. |
| STOPBIT_2 | StopBit를 2로 설정합니다. |

| ParityBit | |
|---|---|
| 이름 | 의미 |
| PARITY_NONE | Parity Bit를 사용하지 않습니다. |
| PARITY_EVEN | Even Parity Bit를 사용합니다. |
| PARITY_ODD | Odd Parity Bit를 사용합니다. |

## 2) STM32F103ZE

STM32F103ZE는 3 개의 USART 포트가 존재한다.
BaudRate는 12가지가 있고, 주로 115200을 많이 쓴다.
WordLength는 8, 9까지 설정할 수 있다.

## API

**void AcgUSARTx_Setting(BaudRate baudRate, WordLength wordLength, StopBit stopBit, ParityBit parity, HWFlowControl hardwareFlowControl); ( x : 1 ~ 3 )**

STM32F103ZE의 USART 통신을 위한 레지스터를 초기화 해주는 함수이다.
매개변수로 BaudRate는 전송속도를 설정해주고
WordLength는 한 번에 보낼 Data의 Bit수를 설정해 준다.
StopBit는 StopBit를 몇 Bit 사용할지 설정해 주고
ParityBit는 parityBit를 어떻게 사용할지 설정해 준다.

**void AcgUSARTx_txc(uint16_t tx); ( x : 1 ~ 3 )**

사용자가 Data를 전송할 때 호출하여 사용하는 함수이다.
매개변수로 char형으로 Data를 받아 전송한다.
**ex) AcgUSART0_txc('A'); //문자 'A'를 송신한다.**

**uint16_t AcgUSARTx_rxc(); ( x : 1 ~ 3 )**

수신 받은 Data 값을 반환하는 함수이다.
uint16_t형으로 Data를 반환한다.
이 함수를 호출하면 수신 값을 받을 때 까지 기다린다.
**ex) rxData = AcgUSART0_rxc(); //수신받은 값을 rxData에 넣는다.**

| BaudRate | |
|---|---|
| 이름 | 의미 |
| BAUDRATE_1200 | 1초에 1200 Bit를 전송합니다. |
| BAUDRATE_2400 | 1초에 2400 Bit를 전송합니다. |
| BAUDRATE_4800 | 1초에 4800 Bit를 전송합니다. |
| BAUDRATE_9600 | 1초에 9600 Bit를 전송합니다. |
| BAUDRATE_14400 | 1초에 14400 Bit를 전송합니다. |
| BAUDRATE_19200 | 1초에 19200 Bit를 전송합니다. |
| BAUDRATE_38400 | 1초에 38400 Bit를 전송합니다. |
| BAUDRATE_56000 | 1초에 56000 Bit를 전송합니다. |
| BAUDRATE_57600 | 1초에 57600 Bit를 전송합니다. |
| BAUDRATE_115200 | 1초에 115200 Bit를 전송합니다. |
| BAUDRATE_128000 | 1초에 128000 Bit를 전송합니다. |
| BAUDRATE_256000 | 1초에 256000 Bit를 전송합니다. |

| WordLength | |
|---|---|
| 이름 | 의미 |
| WORDLENGTH_8 | 8Bit 단위로 data를 전송합니다. |
| WORDLENGTH_9 | 9Bit 단위로 data를 전송합니다. |

| StopBit | |
|---|---|
| 이름 | 의미 |
| STOPBIT_0_DOT_5 | StopBit를 0.5로 설정합니다. |
| STOPBIT_1 | StopBit를 1로 설정합니다. |
| STOPBIT_1_DOT_5 | StopBit를 1.5로 설정합니다. |
| STOPBIT_2 | StopBit를 2로 설정합니다. |

| ParityBit | |
|---|---|
| 이름 | 의미 |
| PARITY_NONE | Parity Bit를 사용하지 않습니다. |
| PARITY_EVEN | Even Parity Bit를 사용합니다. |
| PARITY_ODD | Odd Parity Bit를 사용합니다. |

| Hardware Flowcontrol | |
|---|---|
| 이름 | 의미 |
| HWFLOWCONTROL_NONE | Hardware Flowcontrol를 사용하지 않습니다. |
| HWFLOWCONTROL_RTS | Hardware Flowcontrol RTS를 사용합니다. |
| HWFLOWCONTROL_CTS | Hardware Flowcontrol CTS를 사용합니다. |
| HWFLOWCONTROL_CRTS | Hardware Flowcontrol RTS, CTS를 사용합니다. |

### 3) TMS320F28335

TMS320F28335에는 세 개의 USART 포트가 존재한다.
BaudRate는 레지스터 설정 값에 따라 여러 값이 나올 수 있는데,
AutoCodeGen에서는 주로 쓰는 8가지로 BaudRate를 설정 할 수 있게 하였다.
WordLength는 1~8 Bit까지 설정할 수 있다.

### API

**void AcgUSARTx_Setting(BaudRate baudRate, WordLength wordLength, StopBit stopBit, ParityBit parity); ( x : 0,1,2 )**

TMS320F28335의 USART 통신을 위한 레지스터를 초기화 해주는 함수이다.
매개변수로 BaudRate는 전송속도를 설정해주고
WordLength는 한 번에 보낼 Data의 Bit수를 설정해 준다.
StopBit는 StopBit를 몇 Bit 사용할지 설정해 주고
ParityBit는 parityBit를 어떻게 사용할지 설정해 준다.

**void AcgUSARTx_txc(char tx); ( x : 0,1,2 )**

사용자가 Data를 전송할 때 호출하여 사용하는 함수이다.
매개변수로 char형으로 Data를 받아 전송한다.
**ex) AcgUSART0_txc('A'); //문자 'A'를 송신한다.**

**char AcgUSARTx_rxc(); ( x : 0,1,2 )**

수신 받은 Data 값을 반환하는 함수이다.
char형으로 Data를 반환한다.
이 함수를 호출하면 수신 값을 받을 때 까지 기다린다.
**ex) rxData = AcgUSART0_rxc(); //수신받은 값을 rxData에 넣는다.**

| BaudRate | |
|---|---|
| 이름 | 의미 |
| BAUDRATE_2400 | 1초에 2400 Bit를 전송합니다. |
| BAUDRATE_4800 | 1초에 4800 Bit를 전송합니다. |
| BAUDRATE_9600 | 1초에 9600 Bit를 전송합니다. |
| BAUDRATE_14400 | 1초에 14400 Bit를 전송합니다. |
| BAUDRATE_19200 | 1초에 19200 Bit를 전송합니다. |
| BAUDRATE_38400 | 1초에 38400 Bit를 전송합니다. |
| BAUDRATE_57600 | 1초에 57600 Bit를 전송합니다. |
| BAUDRATE_115200 | 1초에 115200 Bit를 전송합니다. |

| WordLength | |
|---|---|
| 이름 | 의미 |
| WORDLENGTH_1 | 1Bit 단위로 data를 전송합니다. |
| WORDLENGTH_2 | 2Bit 단위로 data를 전송합니다. |
| WORDLENGTH_3 | 3Bit 단위로 data를 전송합니다. |
| WORDLENGTH_4 | 4Bit 단위로 data를 전송합니다. |
| WORDLENGTH_5 | 5Bit 단위로 data를 전송합니다. |

| | |
|---|---|
| WORDLENGTH_6 | 6Bit 단위로 data를 전송합니다. |
| WORDLENGTH_7 | 7Bit 단위로 data를 전송합니다. |
| WORDLENGTH_8 | 8Bit 단위로 data를 전송합니다. |
| WORDLENGTH_9 | 9Bit 단위로 data를 전송합니다. |

| StopBit | |
|---|---|
| 이름 | 의미 |
| STOPBIT_1 | StopBit를 1로 설정합니다. |
| STOPBIT_2 | StopBit를 2로 설정합니다. |

| ParityBit | |
|---|---|
| 이름 | 의미 |
| PARITY_NONE | Parity Bit를 사용하지 않습니다. |
| PARITY_EVEN | Even Parity Bit를 사용합니다. |
| PARITY_ODD | Odd Parity Bit를 사용합니다. |

## 6 TIMER (ACG_TIM.h)

외부(예를 들어 클록)에서 펄스 신호를 카운트하여 특정한 간격으로 삽입하여 신호를 발생시키는 장치를 말한다.

### 1) ATmega128

ATmega128에는 8Bit Timer 2개 (Timer0,2) 16Bit Timer 2개 (Timer1,3)가 존재한다. 8Bit Timer는 100us ~ 10ms까지 만들 수 있고, 16Bit Timer는 100us ~ 4s까지 만들 수 있다.

#### API

| |
|---|
| void AcgTIMERx_Setting(long period); ( x : 0,1,2,3 ) |
| ATmega128의 Timer 설정을 초기화해주는 함수이다.<br>period는 us 단위이다. |
| void TIMER_0x_interrupt(void){} ( x : 0,1,2,3 ) |
| 사용자가 설정한 주기마다 Timer Interrupt가 걸리게 된다.<br>Interrupt Vector안에 호출되는 함수로써 사용자가 원하는 내용을<br>이 함수 안에다 넣어 쓸 수 있다.<br>ACG_interrupt_handler.c 안에 있다가 Timer가 쓰이게 되면<br>main으로 빠져나오게 된다. |

### 2) STM32F103ZE

TMS320F28335는 6개의 Timer가 존재한다.
Timer는 100us ~ 5s까지 만들 수 있다.

#### API

| |
|---|
| void AcgTIMERx_Setting(long period); ( x : 1 ~ 5, 8) |
| ATmega128의 Timer 설정을 초기화해주는 함수이다.<br>period는 us 단위이다. |
| void TIMER_0x_interrupt(void){} ( x : 1 ~ 5, 8) |
| 사용자가 설정한 주기마다 Timer Interrupt가 걸리게 된다.<br>Interrupt Vector안에 호출되는 함수로써 사용자가 원하는 내용을<br>이 함수 안에다 넣어 쓸 수 있다.<br>ACG_interrupt_handler.c 안에 있다가 Timer가 쓰이게 되면<br>main으로 빠져나오게 된다. |

### 3) TMS320F28335

| |
|---|
| TMS320F28335는 3개의 32Bit Timer가 존재한다.<br>32Bit의 주기 레지스터가 존재해서 그 곳에 주기를 넣어주면<br>Timer가 넣어준 주기로 동작을 한다.<br>따라서 범위는 0 ~ $2^{32}$ us 이다. |

### API

| |
|---|
| void AcgTIMERx_Setting(long period); ( x : 0,1,2 ) |
| TMS320F28335 Timer 설정을 초기화해주는 함수이다.<br>period는 us 단위이고 범위는 0 ~ $2^{32}$us 이다. |
| void TIMER_0x_interrupt(void){} ( x : 0,1,2 ) |
| 사용자가 설정한 주기마다 Timer Interrupt가 걸리게 된다.<br>Interrupt Vector안에 호출되는 함수로써 사용자가 원하는 내용을<br>이 함수 안에다 넣어 쓸 수 있다.<br>ACG_interrupt_handler.c 안에 있다가 Timer가 쓰이게 되면<br>main으로 빠져나오게 된다. |

## 7 PWM (ACG_PWM.h)

펄스 변조 방식의 하나로, 변조 신호의 크기에 따라서 펄스의 폭을 변화시켜 변조하는 방식. 그림에서 보는 바와 같이 신호파의 진폭이 클 때는 펄스의 폭이 넓어지고, 진폭이 작을 때는 펄스의 폭이 좁아진다. 단, 펄스의 위치나 진폭은 변하지 않는다.

### 1) ATmega128

ATmega128에는 Timer0,2 에는 PWM 채널이 1개, Timer1,3에는 PWM 채널이 3개씩 존재한다. 3개의 PWM 채널은 하나의 주기를 공유한다.

### API

| void AcgPWMx_Setting(TIMERChannel ch, long period); ( x : 0,1,2,3 ) |
|---|
| ATmega128의 PWM 설정을 초기화 해주는 함수이다.<br>PWM0,2에는 채널이 1개, PWM1,3에는 채널에 3개씩 있다.<br>주기는 us 단위이다. |
| **void AcgPWMx(TIMERChannel ch, long dutycycle) ( x : 0,1,2,3 )** |
| PWM의 dutycycle을 설정해주는 함수이다.<br>dutycycle은 %단위로 0 ~ 100까지 설정 가능하다.<br>프로그램 내에서 사용자가 PWM을 사용하기 전에 직접 써서<br>설정해 주어야 한다. |

| TIMERChannel | |
|---|---|
| 이름 | 의미 |
| TIMER_01 | PWM 1번 채널 (OCxA) |
| TIMER_02 | PWM 2번 채널 (OCxB) |
| TIMER_03 | PWM 3번 채널 (OCxC) |

### 2) STM32F103ZE

STM32F103ZE는 PWM1 ~ PWM6, PWM8까지 6개가 있고 각 PWM 마다 4개의 채널이있어 총 24개의 PWM을 사용할 수 있다.

### API

void AcgPWMx_Setting(PWMChannel ch, int period); ( x : 1 ~ 5, 8 )

STM32F103ZE PWM 설정을 초기화해주는 함수이다.
PWMChannel로 1 ~ 4 채널인지를 받고
주기를 us 단위로 받는다.
주기는 1us ~ 5s 까지 설정 가능하다.

| PWMChannel | |
|---|---|
| 이름 | 의미 |
| TIMER_01 | PWM_1 채널 |
| TIMER_02 | PWM_2 채널 |
| TIMER_03 | PWM_3 채널 |
| TIMER_04 | PWM_4 채널 |

### 3) TMS320F28335

TMS320F28335는 PWM1 ~ PWM6 까지 6개가 있고 각 PWM 마다 2개의 채널이 있어 총 12개의 PWM을 사용할 수 있다.
GPIO_00 ~ GPIO11 까지 사용한다.

### API

void AcgPWMx_Setting(PWMChannel ch, Uint16 period); ( x : 1 ~ 6 )

TMS320F28335 PWM 설정을 초기화해주는 함수이다.
PWMChannel로 PWM_A 채널인지 PWM_B 채널인지를 받고
주기를 us 단위로 받는다.
주기는 1us ~ 50ms 까지 설정 가능하다.

| PWMChannel | |
|---|---|
| 이름 | 의미 |
| TIMER_01 | PWM_A 채널 |
| TIMER_02 | PWM_B 채널 |

## 8 SPI (ACG_SPI.h)

Serial to Peripheral Interface의 약자로 HW/FW 통신 프로토콜이다. 장치들은 마스터 슬레이브 모드로 통신하며 여기서 마스터 장치는 데이터 프레임을 초기화한다. 여러 슬레이브 장치들은 개별 슬레이브 설렉트 라인과 함께 동작 할 수 있다.

SPI 버스는 4가지 논리 신호로 통신을 한다.

**SCLK** : 직렬 클럭 (마스터로부터의 출력)
**MOSI** : 마스터 출력, 슬레이브 입력
**MISO** : 마스터 입력, 슬레이브 출력
**SS** : 슬레이브 설렉트

### 1) ATmega128

ATmega128에는 1개의 SPI 포트가 존재한다.
4MHz, 1MHz, 250kHz, 125kHz의 4가지 통신 속도를 제공한다.
데이터 전송 Bit 수는 8Bit 이다.
SS : PB0
SCK : PB1
MOSI : PB2
MISO : PB3

### API

```
void AcgSPI_Setting(SerialClock clock,SPIMode spiMode,CharLength charLength);
```

ATmega128의 SPI 통신 설정을 초기화해주는 함수이다.
매개변수로 SerialClock을 통해 통신 속도를 결정하고 SPIMode를 통해 통신 방식을 결정한다.
CharLength를 통해 데이터 전송 Bit 수를 정한다.
ATmega128에서는 8Bit만 지원한다.

| SerialClock | |
|---|---|
| 이름 | 의미 |
| SCLK_4000000 | 16MHz / 4 = 4 MHz로 SPI 통신을 한다. |
| SCLK_1000000 | 16MHz / 16 = 1 MHz로 SPI 통신을 한다. |
| SCLK_250000 | 16MHz / 64 = 250 kHz로 SPI 통신을 한다. |
| SCLK_125000 | 16MHz / 128 = 125 kHz로 SPI 통신을 한다. |

| SPIMode | | | |
|---|---|---|---|
| 이름 | | Leading edge | Trailing edge |
| MODE_01 | CPOL=0, CPHA=0 | Rising | Falling |
| MODE_02 | CPOL=0, CPHA=1 | Rising | Falling |
| MODE_03 | CPOL=1, CPHA=0 | Falling | Rising |
| MODE_04 | CPOL=1, CPHA=1 | Falling | Rising |

| CharLength | |
|---|---|
| 이름 | 의미 |
| CHARLENGTH_8 | 8Bit 단위로 Data를 전송한다. |

## 2) STM32F103ZE

STM32F103ZE에는 3개의 SPI 포트가 존재한다.
기본적으로 2개의 통신 속도를 지원한다.
데이터 전송 Bit 수는 8Bit, 16Bit 까지 지원한다.

### API

**void AcgSPIx_Setting(SerialClock clock,SPIMode spiMode,CharLength charLength);**
**(x = 1 ~ 3)**

STM32F103ZE의 SPI 통신 설정을 초기화해주는 함수이다.
매개변수로 SerialClock을 통해 통신 속도를 결정하고
SPIMode를 통해 통신 방식을 결정한다.
CharLength를 통해 데이터 전송 Bit 수를 정한다.

[redacted]

| | |
|---|---|
| SCLK_Divide_128 | (System Clock)/124 kHz로 SPI 통신을 한다. |
| SCLK_Divide_256 | (System Clock)/256 kHz로 SPI 통신을 한다. |

[redacted]

| CK PHASE |
|---|
| 0 |
| 1 |
| 0 |
| 1 |

| CharLength | |
|---|---|
| 이름 | 의미 |
| CHARLENGTH_8 | 8Bit 단위로 Data를 전송한다. |
| CHARLENGTH_16 | 16Bit 단위로 Data를 전송한다. |

### 3) TMS320F28335

TMS320F28335에는 1개의 SPI 포트가 존재한다.
AutoCodeGen에서는 기본적으로 11개의 통신 속도를 지원한다.
데이터 전송 Bit 수는 1Bit ~ 16Bit 까지 지원한다.
MOSI : GPIO_16
MISO : GPIO_17
SCLK : GPIO_18
SS : GPIO_19

### API

```
void AcgSPI_Setting(SerialClock clock,SPIMode spiMode,CharLength charLength);
```

TMS320F28335의 SPI 통신 설정을 초기화해주는 함수이다
매개변수로 SerialClock을 통해 통신 속도를 결정하고
SPIMode를 통해 통신 방식을 결정한다.
CharLength를 통해 데이터 전송 Bit 수를 정한다.

[redacted]

| | |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

| | |
|---|---|
| | |
| | CK PHASE |
| | 0 |
| | 1 |
| | 0 |
| | 1 |

| CharLength | |
|---|---|
| 이름 | 의미 |
| CHARLENGTH_1 | 1Bit 단위로 Data를 전송한다. |
| CHARLENGTH_2 | 2Bit 단위로 Data를 전송한다. |
| CHARLENGTH_3 | 3Bit 단위로 Data를 전송한다. |
| CHARLENGTH_4 | 4Bit 단위로 Data를 전송한다. |
| CHARLENGTH_5 | 5Bit 단위로 Data를 전송한다. |
| CHARLENGTH_6 | 6Bit 단위로 Data를 전송한다. |

| | |
|---|---|
| CHARLENGTH_7 | 7Bit 단위로 Data를 전송한다. |
| CHARLENGTH_8 | 8Bit 단위로 Data를 전송한다. |
| CHARLENGTH_9 | 9Bit 단위로 Data를 전송한다. |
| CHARLENGTH_10 | 10Bit 단위로 Data를 전송한다. |
| CHARLENGTH_11 | 11Bit 단위로 Data를 전송한다. |
| CHARLENGTH_12 | 12Bit 단위로 Data를 전송한다. |
| CHARLENGTH_13 | 13Bit 단위로 Data를 전송한다. |
| CHARLENGTH_14 | 14Bit 단위로 Data를 전송한다. |
| CHARLENGTH_15 | 15Bit 단위로 Data를 전송한다. |
| CHARLENGTH_16 | 16Bit 단위로 Data를 전송한다. |

## ⑨ I2C (ACG_I2C.h)

Inter-Intergrated Circuit의 약자로 필립스에서 개발한 직렬 컴퓨터 버스이다. 저속 주변 기기를 연결하기 위해 사용된다.

직렬 데이터(SDA)와 직렬 클럭(SCL)이라는 두 개의 양 방향 오픈 컬렉터 라인을 사용한다. 가장 일반적으로 사용되는 I2C 버스의 모드는 표준 모드인 100kBit/s 와
저속 모드인 10kBit/s가 사용된다.

### 1) ATmega128

ATmega128에서는 1개의 I2C 포트가 존재한다.
Fast Mode (400kbps) 와 Standard Mode (100kbps)를 지원한다.
SCL : PD0
SDA : PD1

### API

```
void AcgI2C_Setting(BitRate bitRate);
```

ATmega128의 I2C 통신을 초기화 해주는 함수이다.
매개변수로 BitRate를 받아서 통신속도를 설정해준다.
ATmega128에서는 Fast Mode (400kbps)와 Standard Mode (100kbps)를
지원한다.

| BitRate | |
|---|---|
| 이름 | 의미 |
| FAST | 400kbps로 I2C 통신을 한다. |
| STANDARD | 100kbps로 I2C 통신을 한다. |

### 2) STM32F103ZE

STM32F103ZE에서는 2개의 I2C 포트가 존재한다.
Fast Mode (400kbps) 와 Standard Mode (100kbps)를 지원한다.

#### API

**void AcgI2Cx_Setting(BitRate bitRate);  (x = 1, 2)**

STM32F103ZE의 I2C 통신을 초기화 해주는 함수이다.
매개변수로 BitRate를 받아서 통신속도를 설정해준다.
STM32F103ZE에서는 Fast Mode (400kbps)와 Standard Mode (100kbps)를 지원한다.

| BitRate | |
|---|---|
| 이름 | 의미 |
| FAST | 400kbps로 I2C 통신을 한다. |
| STANDARD | 100kbps로 I2C 통신을 한다. |

### 3) TMS320F28335

TMS320F28335에서는 1개의 I2C 포트가 존재한다.
Fast Mode (400kbps) 와 Standard Mode (100kbps)를 지원한다.
SCL :  GPIO_33
SDA :  GPIO_32

#### API

**void AcgI2C_Setting(BitRate bitRate);**

TMS320F28335의 I2C 통신을 초기화 해주는 함수이다.
매개변수로 BitRate를 받아서 통신속도를 설정해준다.
TMS320F28335에서는 Fast Mode (400kbps)와 Standard Mode (100kbps)를 지원한다.

| BitRate | |
|---|---|
| 이름 | 의미 |
| FAST | 400kbps로 I2C 통신을 한다. |
| STANDARD | 100kbps로 I2C 통신을 한다. |

## 10 Encoder (ACG_ENC.h)

모터의 회전수와 속도를 감지할 수 있다.

### 1) ATmega128

ATmega128에는 Encoder가 존재하지 않는다.

### 2) STM32F103ZE

STM32F103ZE에서는 6개의 2상 Encoder가 존재한다.

#### API

| void AcgENCx_Setting(int max);  (x = 1 ~ 5, 8) |
|---|
| STM32F103ZE의 Encoder를 설정하는 함수이다.<br>매개변수로 카운터의 최대값을 받는다 |

| Uint32 AcgENCx(void);  (x = 1 ~ 5, 8) |
|---|
| 엔코더의 카운터 값을 반환하는 함수이다.<br>사용자가 원하는 곳에 써서 값을 받아 올 수 있다. |

### 3) TMS320F28335

TMS320F28335에서는 2개의 4상 Encoder가 존재한다.

#### API

| void AcgENC1_Setting(Uint32 max); |
|---|
| TMS320F28335의 Encoder를 설정하는 함수이다.<br>매개변수로 카운터의 최대값을 받는다 |

| Uint32 AcgENC1(void); |
|---|
| 엔코더의 카운터 값을 반환하는 함수이다.<br>사용자가 원하는 곳에 써서 값을 받아 올 수 있다. |

## 11 ECAP (ACG_ECAP.h)

PWM과 같이 펄스를 내보낸다.
보조 PWM 역할을 한다.

### 1) ATmega128

ATmega128에는 ECAP이 존재하지 않는다.

### 2) STM32F103ZE

STM32F103ZE에는 ECAP이 존재하지 않는다.

### 3) TMS320F28335

TMS320F28335에서는 6개의 ECAP 채널이 존재한다.
ECAP_01 : GPIO_34
ECAP_02 : GPIO_37
ECAP_03 : GPIO_26
ECAP_04 : GPIO_27
ECAP_05 : GPIO_48
ECAP_06 : GPIO_49

#### API

`void` AcgECAPx_Setting(Uint32 period); ( x : 1~6 )

TMS320F28335의 ECAP을 초기화 해주는 함수이다.
period는 us 단위이다.
총 6개의 채널이 있다.

`void` AcgECAPx(int dutycycle); ( x : 1~6 )

ECAP의 dutycycle을 설정해주는 함수이다.
dutycycle은 %단위로 0 ~ 100까지 설정 가능하다.
프로그램 내에서 사용자가 ECAP을 사용하기 전에 직접 써서
설정해 주어야 한다.

`void` ECAP_0x_interrupt() { } ( x : 1~6 )

ECAP에서 한 주기마다 Interrupt가 걸린다.

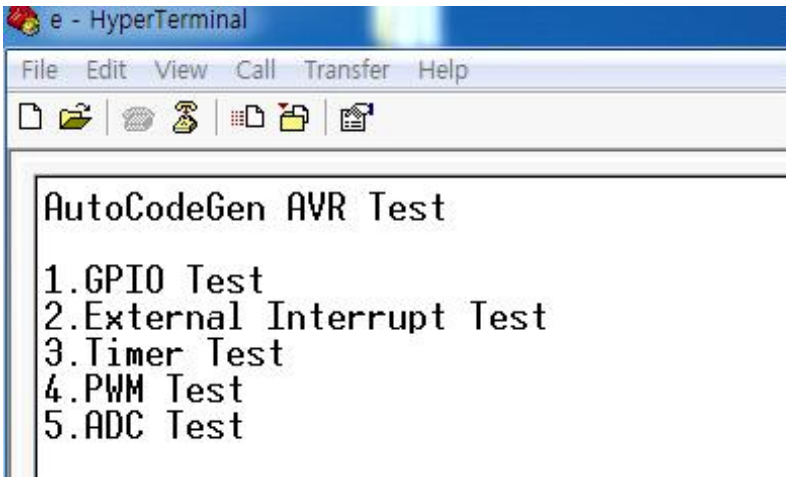
Interrupt handler안에서 호출되는 함수로 ACG_interrupt_handler.c에
있다가 ECAP이 사용되면 main으로 빠져나온다.
사용자가 원하는 내용을 이 함수안에 써주면 된다.

| BitRate | |
|---|---|
| 이름 | 의미 |
| FAST | 400kbps로 I2C 통신을 한다. |
| STANDARD | 100kbps로 I2C 통신을 한다. |

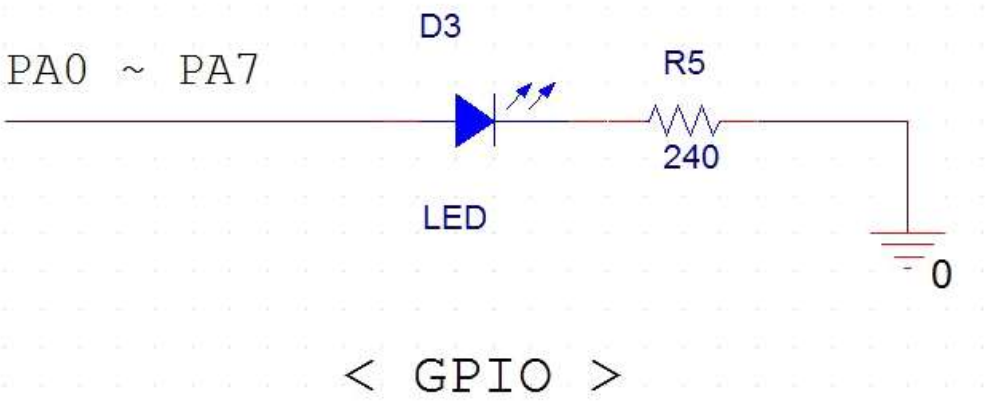
# Ⅵ. 생성된 결과물의 응용

## 1 Atmega128 Test Board

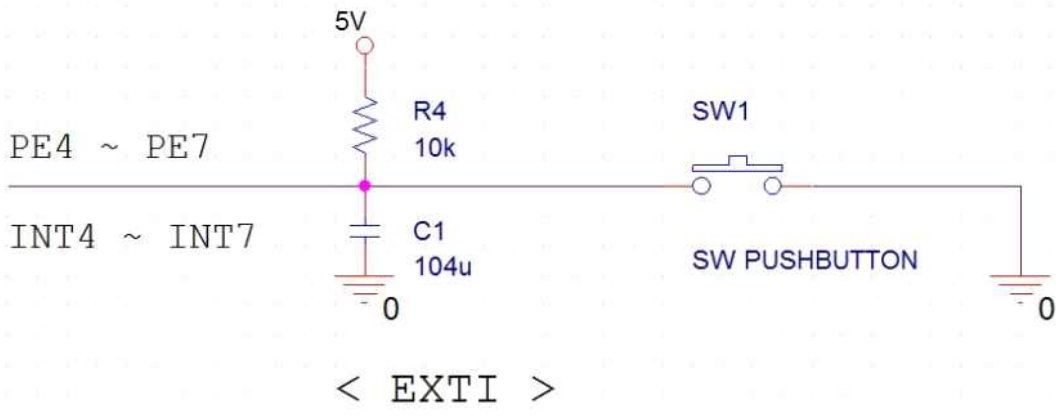
ATmega128의 Test Board이다. GPIO, EXTI, USART, PWM, TIMER, ADC 기능을 테스트 해볼 수 있다.



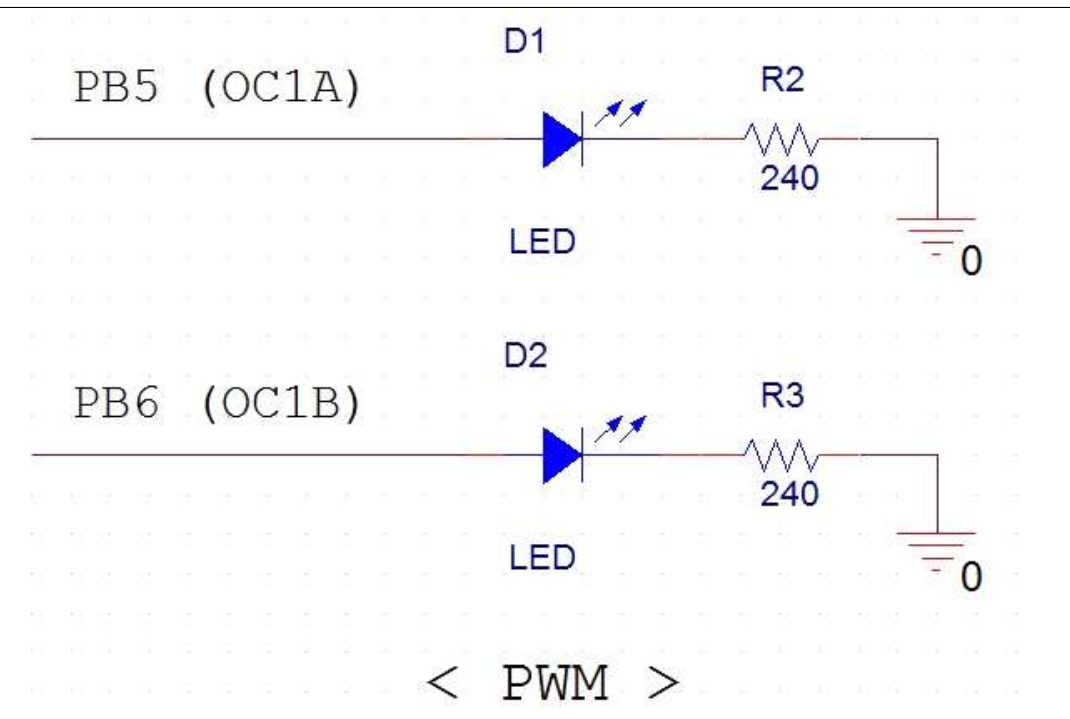

USART를 통한 테스트 프로그램이다. 기본적으로 USART를 이용하기 때문에 테스트 프로그램이 정상적으로 동작을 하면 USART가 제대로 동작함을 알 수 있다.



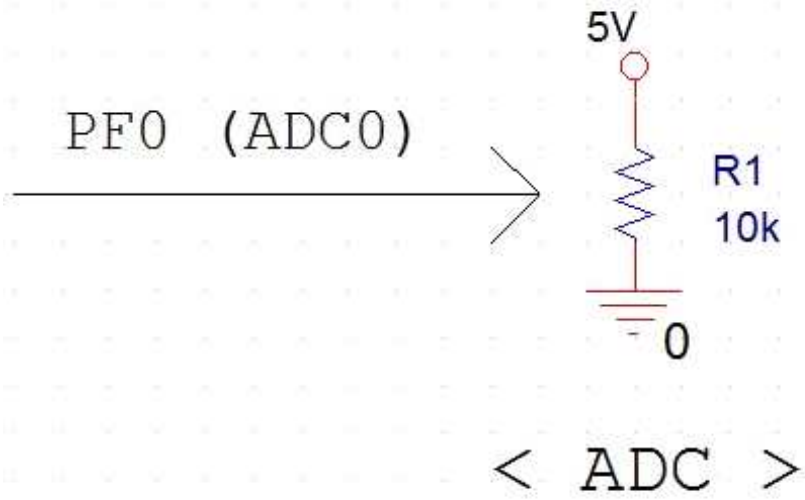

PA0 ~ PA7 까지 LED를 달아서 LED를 켜고 끔으로써 **GPIO** 테스트를 해볼 수 있다.
일정 주기로 TIMER를 설정해서 LED를 켜고 끔으로써 **TIMER** 또한 테스트 해볼 수 있다.




PE4 ~ PE7 까지 스위치를 달아서 **EXTI**를 테스트 해볼 수 있다.



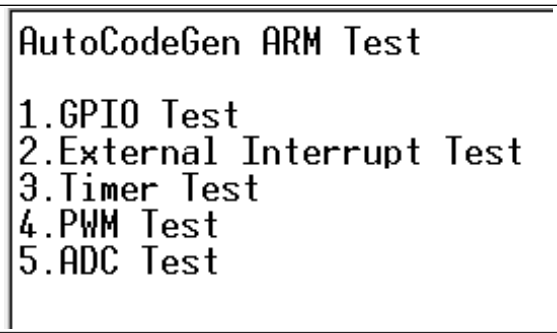

PWM 출력 핀 PB5와 PB6에 고휘도 LED를 달아서 **PWM** 테스트를 할 수 있다.
duty rate를 바꾸면 LED 불빛의 밝기가 달리짐을 알 수 있다.



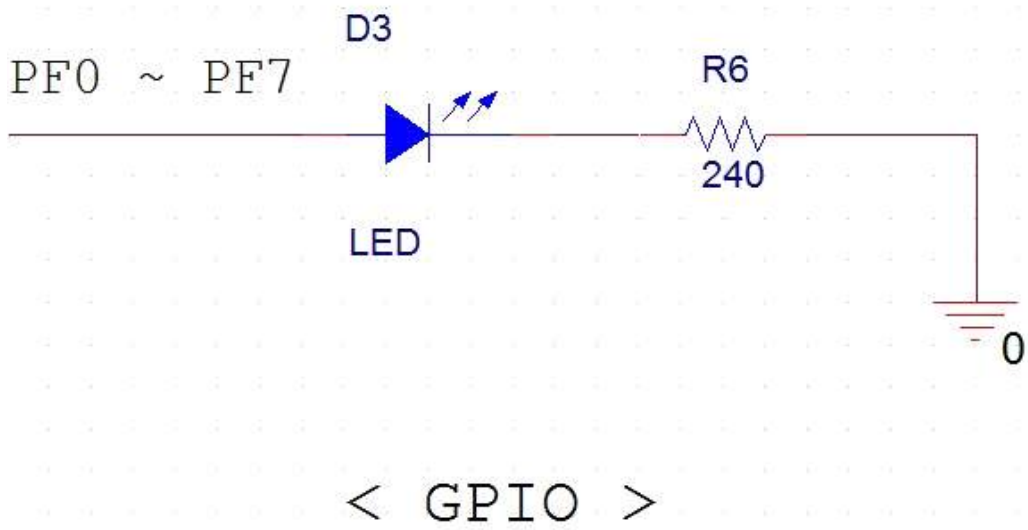

PF0에 가변저항을 연결하여 **ADC** 테스트를 할 수 있다. 가변저항을 돌림에 따라 ADC 되어 나오는 값이 달리짐을 알 수 있다.

## 2 STM32F103ZE Test Board

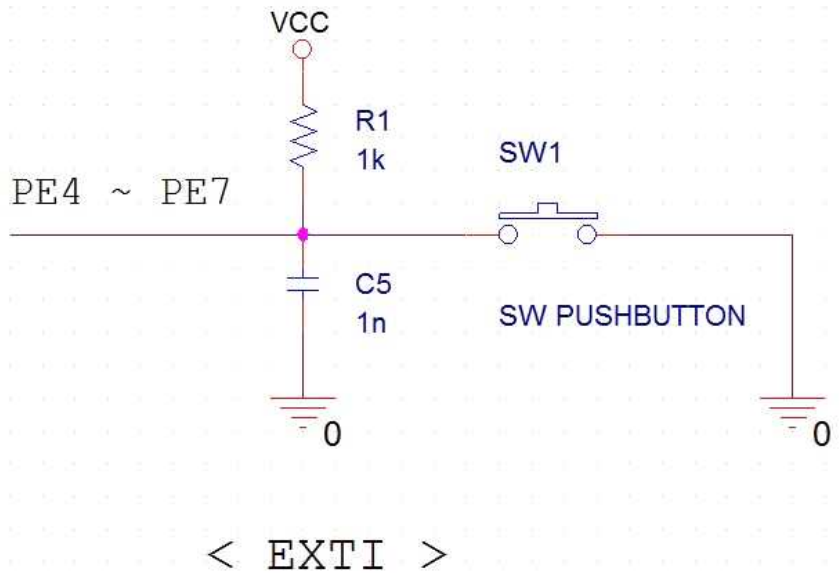
STM32F103ZE의 Test Board이다. GPIO, EXTI, USART, PWM, TIMER, ADC 기능을 테스트 해볼 수 있다.




USART를 통한 테스트 프로그램이다. 기본적으로 USART를 이용하기 때문에 테스트 프로그램이 정상적으로 동작을 하면 USART가 제대로 동작함을 알 수 있다.

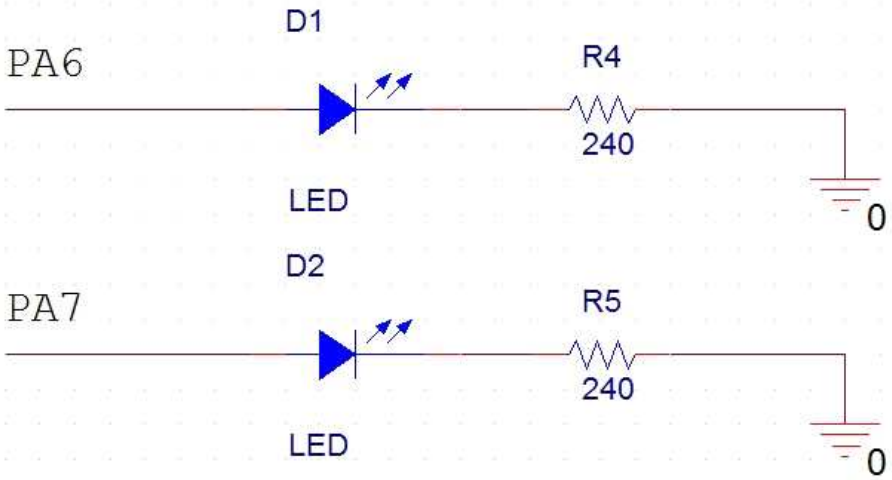



PA0 ~ PA7 까지 LED를 달아서 LED를 켜고 끔으로써 **GPIO** 테스트를 해볼 수 있다.
일정 주기로 TIMER를 설정해서 LED를 켜고 끔으로써 **TIMER** 또한 테스트 해볼 수 있다.



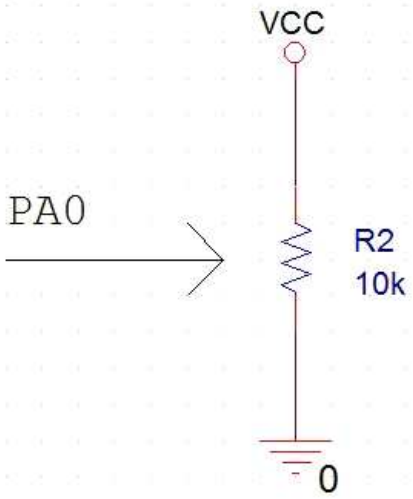

< EXTI >

PE4 ~ PE7 까지 스위치를 달아서 **EXTI**를 테스트 해볼 수 있다.




< PWM >

PWM 출력 핀 PB5와 PB6에 고휘도 LED를 달아서 **PWM** 테스트를 할 수 있다.
duty rate를 바꾸면 LED 불빛의 밝기가 달리짐을 알 수 있다.

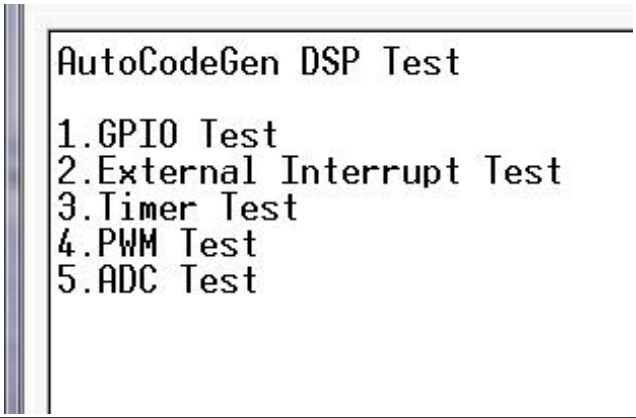



< ADC >

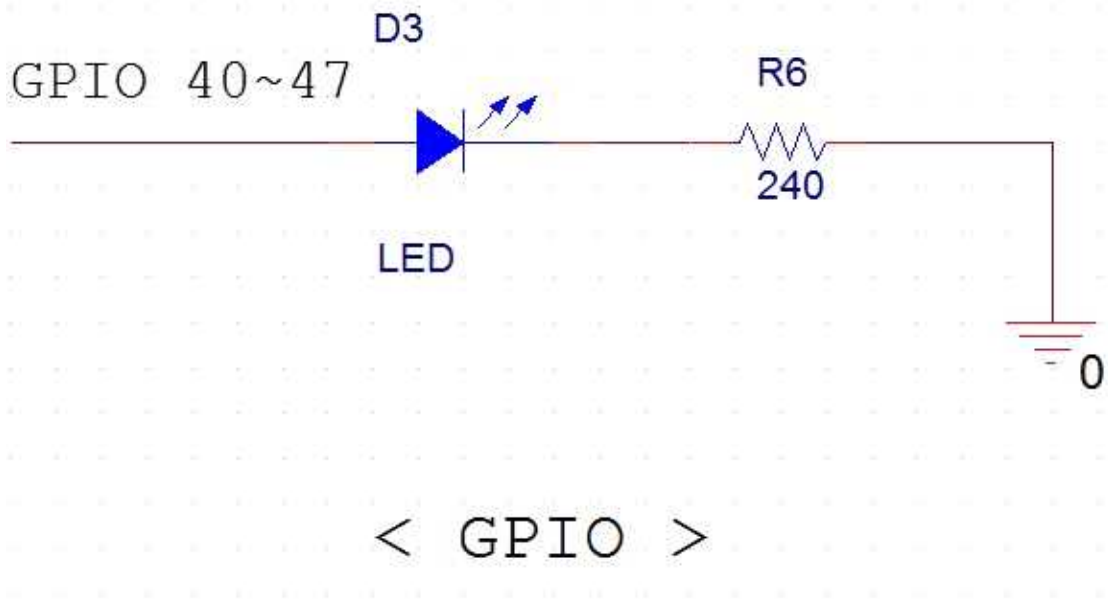
PF0에 가변저항을 연결하여 ADC 테스트를 할 수 있다. 가변저항을 돌림에 따라 ADC 되어 나오는 값이 달리짐을 알 수 있다.

## 3 TMS320F28335 Test Board

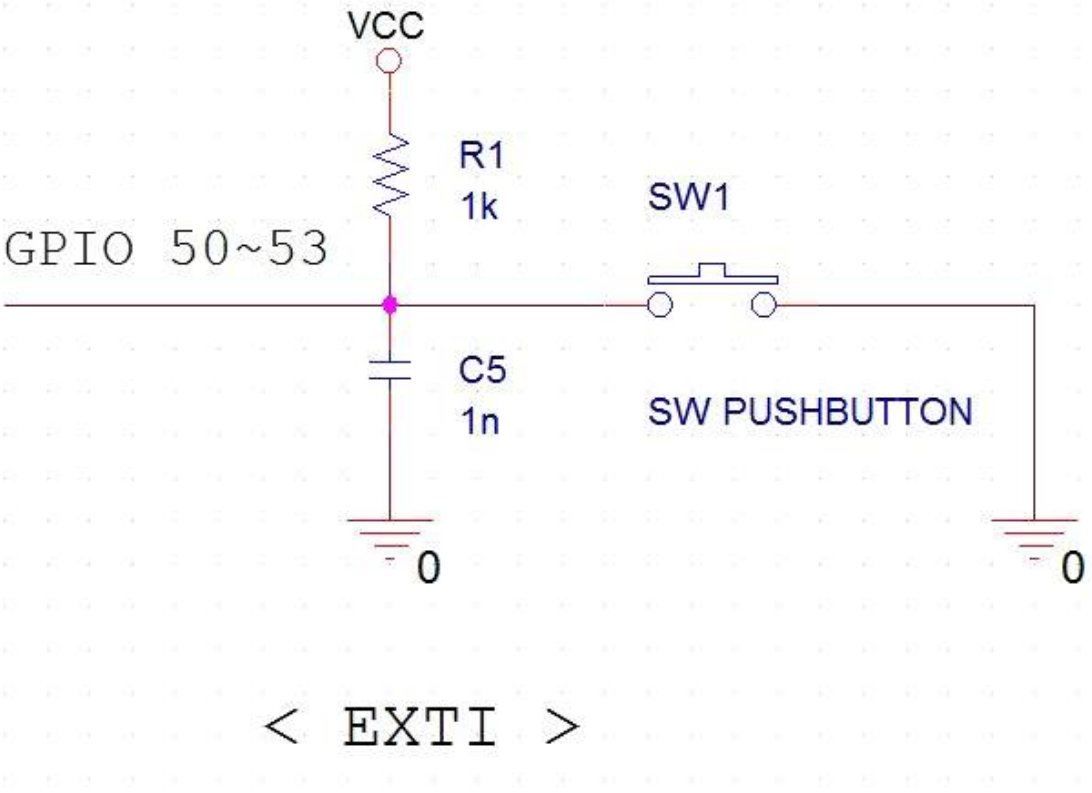
TMS320F28335의 Test Board이다. GPIO, EXTI, USART, PWM, TIMER, ADC 기능을 테스트 해볼 수 있다.



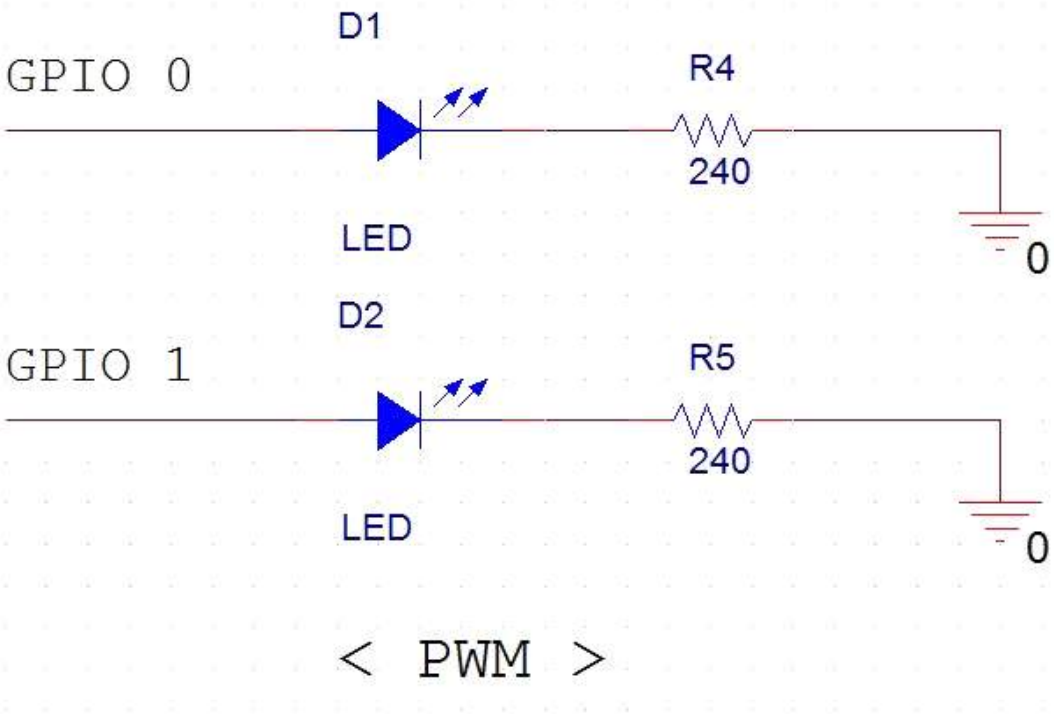

USART를 통한 테스트 프로그램이다. 기본적으로 USART를 이용하기 때문에 테스트 프로그램이 정상적으로 동작을 하면 USART가 제대로 동작함을 알 수 있다.



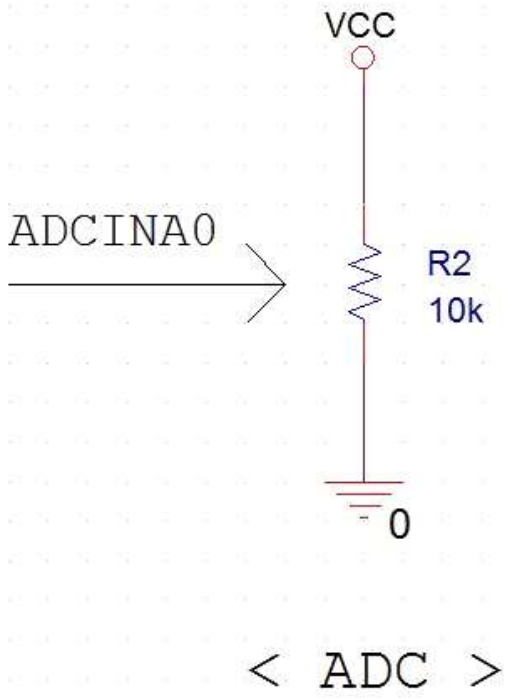

PA0 ~ PA7 까지 LED를 달아서 LED를 켜고 끔으로써 **GPIO** 테스트를 해볼 수 있다.
일정 주기로 TIMER를 설정해서 LED를 켜고 끔으로써 **TIMER** 또한 테스트 해볼 수 있다.




PE4 ~ PE7 까지 스위치를 달아서 **EXTI**를 테스트 해볼 수 있다.




PWM 출력 핀 PB5와 PB6에 고휘도 LED를 달아서 **PWM** 테스트를 할 수 있다.
duty rate를 바꾸면 LED 불빛의 밝기가 달리짐을 알 수 있다.




< ADC >

PF0에 가변저항을 연결하여 **ADC** 테스트를 할 수 있다. 가변저항을 돌림에 따라 ADC 되어 나오는 값이 달리짐을 알 수 있다.

# Ⅶ. 용어 설명

| Word | Abbreviation | Content |
|---|---|---|
| ACG | AutoCodeGen | Automatically Embedded-C Code Generation.<br>It also mean our program title. |
| ADC | Analog to Digital Converter | Converts a analog signal to digital value. |
| Baudrate | | Transfer speed as a digitally modulated signal or bits per second. |
| ENC | Rotary Encoder | It also called Shaft Encoder.<br>Converts the angular position or motion of shaft or axle to an analog or digital code. |
| EXTI | External Interrupts | It can be using a Button Click, Signal Interrupt, Switch event, etc. |
| GPIO | General Purpose I/O | I/O through a non-purposed pin.<br>It can be using a digital input or output. |
| LM | Luminary Co., Ltd. | |
| Prescaler | | It is an electronic counting circuit used to reduce a high frequency electrical signal to a lower frequency by integer division. |
| PWM | Pulse Width Modulation | Generate a Pulse with some parameters.<br>Period is product of pulse counter and pre-scaler constant. |
| Stop Bits | | This value used to distinction of a sampling data. |
| ST(M) | STMicroelectronics Co., Ltd. | |
| SysClk | System Clock | |
| Timer | Timer Interrupts | Periodical interrupt when the count reach the maximum pulse count. |
| USART | Universal Synchronous Rx/Tx<br>Universal Asynchronous Rx/Tx | The One of communication protocol.<br>UART is Rx/Tx onto RS232, RS485, etc,… |
| ECAP | Enhanced Capture Module | It is used in systems where accurate timing of external events is important. It also generate a pulse. |
| DSP | Digital Signal Processor | |
| SPI | Serial to Peripheral Interface | SPI bus is a synchronous serial data link defacto standard named by Motorola, that operates in full duplex mode. |
| I2C | Inter-Intergrated Circuit | It is a multimaster serial single-ended computer bus invented by Philips used for attaching low-speed peripherals to a motherboard, embedded system, cellphone, or other electronic device. |