Firmware Specification

**GEN2, allears**

**Ver 0.1**

# **Revision History**

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **History** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# **Preface**

이 문서는 allears Gen2 DEMO 프로젝트 Firmware 알고리즘 및 명령어 해설 문서이다.

# **Terms and Definition**

## **Overview**

* 이 문서에서 사용되는 용어들을 정의한다.

## **Terms**

|  |  |
| --- | --- |
| **Term** | **Description** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# **Introduction**

1. Gen2는 allears의 2세대 버전이다.
2. 해당 실험 장치는 미주신경계를 자극하는 실험용 장치다.
3. UART로 명령을 하여 자극의 세기나 설정 값을 변경하고 상태를 확인한다.
4. 실험용 장치의 메인 컨트롤러는 Nucleo-64 형식의 STM32 L412RB MCU를 사용한다.

# **Version Format**

## **Format**



## **Number Meaning**

1. Major number
   * Numbering in order
2. Minor number

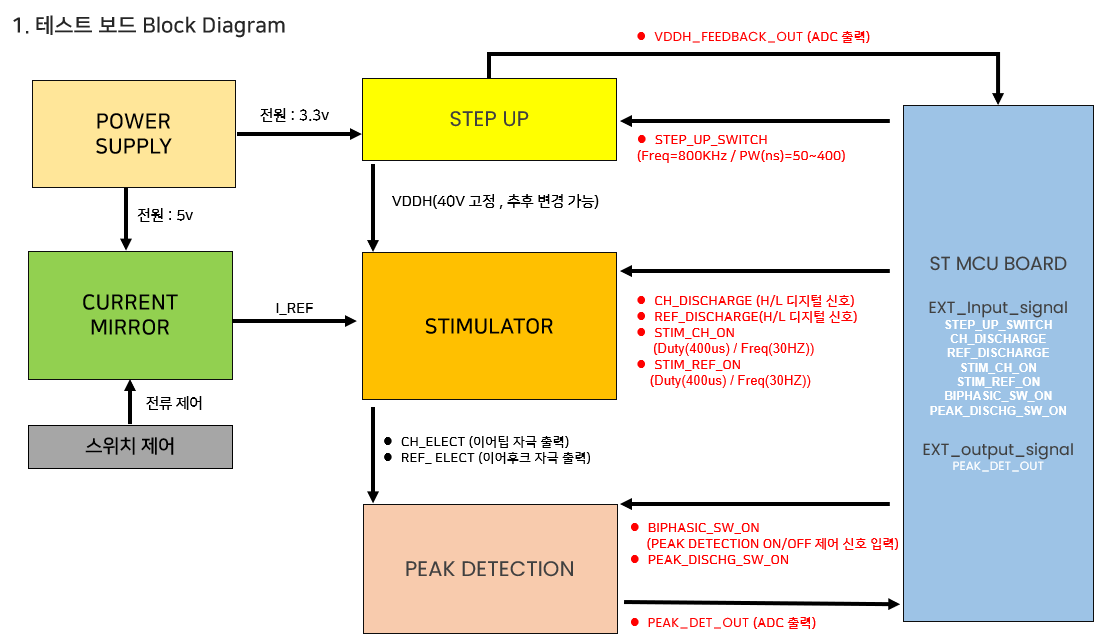
* Numbering in order

1. Sub-minor number
   * Numbering in order
2. ASCII ‘B’ or ‘R’
3. B: Beta version. Test or experimental version
4. R: Release version Officially

# **Basic Concept**

1. PC와 메인 MCU는 UART 통신으로 데이터를 주고받는다.
2. PC와 주고받은 데이터를 기반으로 MCU는 자극 회로를 제어한다.

# **Hardware Basic Diagram**

텍스트, 장치, 측정기, 게이지이(가) 표시된 사진

자동 생성된 설명

1. ST MCU 보드는 STEP UP에 PWM 출력의 전압을 제어하며 전압 데이터 값을 피드백 받는다.
2. ST MCU 보드는 STIMULATOR에 자극 파형을 출력한다.
3. STIMULATOR의 파형 출력이 PEAK DETECTION 블록의 입력으로 들어가며 PEAK DETECTOIN 블록에서 피크 값을 ST MCU 보드에 보내준다.
4. ST MCU 보드는 PEAK DETECTION에서 나오는 피크의 출력 유무를 제어할 수 있다.

# **Electrical Stimulation**

## **Concepts**



## **Electrical Stimulation Parameters**



1. 하나의 자극 펄스는 하나의 Cathode와 하나의 Anode로 이루어진다.
2. 1s / Period = stimulation pulse Frequency
3. Cathode와 Anode의 Pulse width는 같다.
4. Cathode가 먼저 출력 되고, Dead Time 이후에 Anode를 출력한다.
5. 전기 자극이 출력되는 동안에는 충전 전압을 방전하고, 전기 자극이 멈추는 동안에 전압을 충전한다.

## **Electrode Charging**

1. 전기 자극에서 사용되는 전압을 충전한다.
2. 충전되는 전압은 일정하게 유지되어야 한다.
3. Control points

|  |  |
| --- | --- |
| **Control points** | **Description** |
| volt\_ctrl | 전압 수준을 제어 |
| monitor\_volt | 충전된 전압 수준을 확인/현재 전압 출력 수준 |
| stimul\_volt | 전기 자극으로 출력되는 전압. 방전 시 전압 출력 목표 전압 |

## **Electrical Stimulation**

1. 전기 자극의 수준들 제어한다.
2. 출력되는 전기 자극의 정도는 착용 상태를 확인하는데 사용된다.
3. Control points

|  |  |
| --- | --- |
| **Control points** | **Description** |
| ca\_ctrl | cathode 전극의 파형 출력을 제어 |
| an\_ctrl | anode 전극의 파형 출력을 제어 |
| cathode | 출력 되는 전기 자극의 전극의 하나 |
| anode | 출력 되는 전기 자극의 전극의 하나 |

## **Electrical Stimulation Level**

|  |  |  |
| --- | --- | --- |
| **자극 레벨** | **출력 전압 (V)** | **펄스 폭(us)** |
| Level 0 | 0 | 0 |
| Level 1 | 40 | 50 |

## **Peak Detection**

1. 자극의 피크 감지기능의 작동 유무를 제어한다.
2. 출력된 피크의 정보를 기반으로 임피던스를 측정하는데 사용한다.
3. Control points

|  |  |
| --- | --- |
| **Control points** | **Description** |
| peak\_val | 입력되는 Peak 값 |
| peak\_pwr\_on | Peak 감지 기능의 전원부를 제어 |
| peek\_on | Peak 감지 기능의 출력부를 제어 |

## **Peak Detection Level**

|  |
| --- |
| **Voltage (V)** |
| 0 ~ 42 (여유2V 줌) |

# **System Peripheral Index**

1. ADC – ADC DMA
2. TIMER – Timer Update Event / PWM PULSE
3. GPIO – LED / Button / Current DAC Control
4. UART – Shell Command Read/Write
5. FLASH – DATA SAVE / LOAD

# **코드의 폴더 구조 – 파일 별 기능 설명**

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

## **Td\_adc.c / td\_adc.h**

* ADC 관련 FSM이 돌아가는 곳이다. ADC 관련 계산을 수행한다.

## **Td\_btn.c / td\_btn.h**

* 버튼에 대한 상태 값을 제어한다.

## **Td\_flash\_memory.c / Td\_flash\_memory.h**

* Flash 메모리에 값을 저장하거나 읽어 들이는 기능을 수행한다.

## **Td\_led.c / Td\_led.h**

* LED에 대한 상태 값을 제어한다.

## **Td\_retarget.c**

* UART 출력을 printf 함수로 출력할 수 있도록 재 지정한다.

## **Td\_private.h / td\_schdule.h**

* 펌웨어 버전, 딜레이 시간 값을 Define 한다.

## **Td\_shell.c / td\_shell.h**

* UART로 특정 명령어가 들어오면 실행할 기능을 가진 함수를 호출한다.
* 일정시간동안 UART로 입력이 없으면 들어왔던 데이터를 초기화한다.
* 특정 형태로 출력할 함수를 내포하고 있다. Ex) HEX 형태

## **Td\_stim\_setting.c / Td\_stim\_setting.h**

* **UART로 들어온 특정한 명령을 수행한다. 자세한건 UART Command 참고**
* UART로 들어온 데이터를 파싱 한다. 추출된 데이터를 반영하여 제어한다.

## **Td\_sys\_common.c / Td\_sys\_common.h**

* 장치의 프로그램이 시작될 때 초기 값을 설정한다.
* 오류가 발생할 때 처리하도록 한다.

## **Td\_sys\_fsm\_state.c / Td\_sys\_fsm\_state.h**

* 장치의 시스템을 FSM 디자인 기반으로 PWM 펄스 제어, LED 출력, ADC계산 시작 등을 제어한다.

## **Td\_system\_timing.c**

* 함수 내에서 ADC에서 계산된 값을 기반으로 하여Step-up을 특정 시간마다 피드백 제어를 한다.
* 이 함수는 타이머 기반으로 돌아가며 주기는 100ms다.

## **Td\_uart2.c**

* UART2번에 대한 입력을 처리한다.
* 들어온 데이터를 인터럽트로 읽으며 읽은 데이터는 Queue에 넣는 기능을 한다.
* UART2번의 Queue 관리를 도맡아서 한다.

# **Main 스케줄러**

Main의 스케줄러는 시스템에서 사용하는 주변장치의 스케줄을 관리한다. 스케줄러 내부의 각 주변 장치는 FSM과 스케줄러에 따라서 상태를 업데이트하고 동작하게 된다.

## 구성요소

FSM Handler, Button Handler, LED Handler, ADC Handler, UART Input Prints



# **System FSM Diagram**



* INIT 상태
* 시스템에서 전원을 넣고 넘어오는 상태다.
* 해당 상태에서는 내부 메모리에 저장한 데이터를 불러오며, 이를 실패할 경우 기본 값으로 설정하게 된다.
* IDLE 상태
* INIT 상태가 끝나고 아무것도 하지 않는 상태다.
* 버튼 입력 이벤트 또는 UART 시작 커맨드로 RUN 상태에 진입할 수 있다.
* UART 명령어를 통해 파형의 주기, 펄스 폭, Dead Time을 **설정하고 저장하며, 볼 수 있다.**
* RUN 상태
* 자극 파형이 나오는 상태다.
* 버튼 입력 이벤트 또는 UART 정지 명령어로 IDLE 상태에 진입할 수 있다.
* **설정한 값을 볼 수 있으며, 해당 상태에서는 값 설정변경이 불가능하다.**
* ERROR 상태
* 칩 자차에서 발생한 오류로 인해 아무런 동작이 되지 않는 상태다. 해당 상태에서는 시스템 리셋을 실행한다.

# **Button Handler**

스케줄러 내부에서 버튼 Handler가 돌아가며 입력에 따라서 FSM 상태가 변경이 된다.

|  |  |  |
| --- | --- | --- |
| **Button Event** | **State** | **Button action** |
| Pressed | IDLE | Start stimulation |
| Pressed | RUN | Stop |
| Pressed | ERROR | No action |

# **LED Handler**

스케줄러 내부에서 LED Handler가 돌아가며 FSM 상태에 따라서 LED 상태가 변한다.

|  |  |  |
| --- | --- | --- |
| **LED** | **State** | **Indication** |
| Green | IDLE | LED OFF |
| Green | RUN | LED ON |
| Green | ERROR | LED blinks 10 times in 3sec |

# **UART 구현 방법 - Shell Command**

## **Queue 입력 처리**

UART 입력은 시스템의 가장 중요한 부분이며 이는 데이터를 읽는 이벤트가 있을 때 우선적으로 먼저 처리해야 하므로 인터럽트로 방식으로 데이터를 읽어 들여야만 한다. 명령어를 처리하는 순서는 다음과 같다.

1. 시리얼 터미널 창에서 데이터를 인터럽트로 읽어 Queue에 데이터를 넣는다.
2. Main문의 Shell CMD Handler에서 pooling방식으로 들어온 데이터 중 엔드 문자가 들어오면 Queue에서 명령어에 해당하는 문자가 순서대로 들어왔는지 확인한다.
3. 확인이 되면 해당 명령어를 실행한다. 이후 Queue 데이터를 초기화한다.



## **명령어 처리 순서**

1. 먼저 Queue에 들어온 데이터가 메시지 명령 규칙대로 들어왔는지 확인한다.
2. “#set”, “#get” 또는 admin 명령이 들어온 것이 확인되면 명령어의 길이, 뒤에 들어온 문자를 비교 연산하여 해당하는 입력이 어떤 명령어인지를 판단한다.
3. 명령어 판단이 되면 해당하는 데이터 처리 함수를 호출하여 구문 분석을 한 뒤 들어온 문자열에서 데이터를 추출한다.

EX>>

1. 추출한 데이터를 반영해야 될 곳에 반영을 한 뒤 추출한 값을 “#res” 형태로 출력한다.
2. 이후 끝나면 Queue가 초기화 된다.

## **출력 처리**

UART 출력은 시스템에서 디버깅 및 동작 확인용으로 사용한다. 따라서 입력보다는 우선순위가 낮으므로 DMA방식 출력이 아닌 일반적인 UART 출력을 사용한다.

단 시리얼 터미널에서 문자 1개씩 입력할 때 마다 입력한 문자를 출력해줘야 한다. 이는 main문에서 스케줄러 내부에서 1개씩 출력하게 된다.

# **UART Command**

자극 관련 명령어와 응답 메시지를 UART를 통해 송수신한다.

## **메지지 및 명령어 규칙**

1. 모든 명령어는 접두어로 “#”을 사용한다.
2. 시스템의 중요한 명령어 5개는 “#”을 제외한 접두어가 없다.

* 시작, 정지, 저장, 공장초기화, 버전정보

1. 값을 세팅할 때는 접두어로 “#set” 가 붙는다.

* Ex) #setDT

1. 설정한 값을 읽어올 때는 접두어로 “#get” 가 붙는다.

* Ex) #getDT

1. #set, #get에 대한 응답 메시지는 접두어로 “#res”가 붙어 나온다.

* Ex) #resDT

1. 에러 코드는 접두어로 “#Error” 가 붙으며 에러 코드번호는 “-“로 구분한다.

* Ex) #Error-1

1. 그 외 시스템 메시지는 형식이 없다.

## **Command and Response Format**

### **Admin Command**

|  |  |
| --- | --- |
| **Command type** | **Description** |
| #start | 자극 시작 명령어 |
| #stop | 자극 중지 명령어 |
| #save | 설정한 펄스 폭, 주기, Dead Time 등의 데이터를 저장하는 명령어 |
| #factoryreset | 설정된 값을 초기 값으로 설정하는 공장초기화 명령어 |
| #help | 명령어 매뉴얼 출력 |
| #version | 버전 정보를 확인 |

### **Parameter Command**

|  |  |
| --- | --- |
| **Parameter type** | **Description** |
| #setDT,xxx | 입력범위 1 ~ 100us, 10us 단위  펄스와 펄스 폭 사이의 간격 시간을 설정하는 명령어  EX) #setDT,100 => Dead Time 100us 로 설정함 |
| #setPW,xxx | 입력범위 1 ~ 1000us, 100us 단위  펄스 폭을 설정하는 명령어  EX) #setPW,1000 => 펄스 폭 1000us 로 설정함 |
| #setHZ,xxx | 입력범위 1 ~ 30Hz, 1단위  펄스의 주기를 설정하는 명령어  EX) #setHZ,30 => 30Hz 설정함 |
| #setVPW | step-up 관련 펄스 폭을 설정함. **(0~30)/디버깅시에만 사용 권장함** |
| #setVOL | **출력되어야 할 Set-up 전압 값을 설정함 (0~45)** |
| #setDAC | 전류세기를 단계별로 제어함 (0 ~ 15)  해당 명령은 입력 값에 따라서 GPIO를 4Bit 형태로 제어하게 된다.   |  |  |  |  | | --- | --- | --- | --- | | DAC3 | DAC2 | DAC1 | DAC0 | | 0 | 0 | 0 | 0 |   > 참고: 제일 낮은 BIT가 DAC 0번이다.   |  |  | | --- | --- | | 10진수 | GPIO 형태(0:LOW / 1: HIGH) | | 0 | 0 0 0 0 | | 1 | 0 0 0 1 | | 2 | 0 0 1 0 | | 3 | 0 0 1 1 | | 4 | 0 1 0 0 | | 5 | 0 1 0 1 | | 6 | 0 1 1 0 | | 7 | 0 1 1 1 | | 8 | 1 0 0 0 | | 9 | 1 0 0 1 | | 10 | 1 0 1 0 | | 11 | 1 0 1 1 | | 12 | 1 1 0 0 | | 13 | 1 1 0 1 | | 14 | 1 1 1 0 | | 15 | 1 1 1 1 | |

|  |  |
| --- | --- |
| #getDT | 설정된 Dead Time 출력 |
| #getPW | 설정된 펄스 폭 출력 |
| **#**getHZ | 설정된 펄스 주기 출력 |
| #getVPW | 설정된 step-up 관련 펄스 폭을 설정함. |
| #getVOL | 설정된 출력되어야 할 Set-up 전압 값을 설정함 |
| #getDAC | 설정된 전류세기를 단계를 출력함 |
| #getALLPRM | 설정된 모든 값 출력 |

### **Parameter Response Message**

|  |  |
| --- | --- |
| **Message type** | **Description** |
| #resDT: XXX | #setDT, #getDT 출력에 대한 형식이다.  XXX는 설정된 값 |
| #resPW: XXX | #setPW, #getPW 출력에 대한 형식이다.  XXX는 설정된 값 |
| #resHZ: XXX | #setHZ, #getHZ 출력에 대한 형식이다.  XXX는 설정된 값 |
| #resVPW: XXX | #setVPW, #getVPW 출력에 대한 형식이다.  XXX는 설정된 값 |
| #resVOL: XXX | #setVOL, #getVOL 출력에 대한 형식이다.  XXX는 설정된 값 |
| #resDAC: XXX | #setDAC, #getDAC 출력에 대한 형식이다.  XXX는 설정된 값 |
| #resALLPRM  DT: XXX  PW: XXX  HZ: XXX  VPW: XXX  VOL: XXX | #getALLPRM 출력에 대한 형식이다.  XXX는 설정된 값 |

### **Error Codes**

|  |  |
| --- | --- |
| **Error code** | **Description** |
| #Error-1 | System Fault Exception  에러 처리: 이후 시스템을 리셋을 한다. |
| #Error-2 | 설정한 값을 내부 메모리에 저장 실패  에러 처리: 사용자가 저장을 재 시도한다. |

# **ADC Handler구현 – ADC FSM**

## **참고사항**

1. ADC는 시스템과 마찬가지로 FSM으로 구현되었다. ADC Handler의 존재 의의는 ADC FSM 상태가 변경될 때 업데이트를 하는데 의의가 있다.
2. ADC는 2개의 채널을 사용하며 1개는 Step-up 1개는 Peak Detection을 위해 사용한다.

* **Step UP 피드백: ADC – Feedback, TIMER Step-up Pulse 참고 바람**
* **Peak Detection: TIMER – Peak Detection 참고 바람**

1. Step-up 관련 ADC는 FSM에 따라 동작을 하고 Peak Detection은 타이머 이벤트에 따라 동작을 하게 된다.
2. 비고: ADC DMA Conversion 시간은 20us가 채 되지 않습니다.
3. 설정한 값에 따라 채널 1의 ADC DMA는 시작하면 연속적으로 읽는 것이 아닌 ADC Conversion 할 때 마다 코드에서 설정한 버퍼 크기만큼 읽어 들입니다.
4. 설정한 값에 따라 채널 2의 ADC DMA는 타이머 주기마다 Conversion을 하여 연속적으로 ADC를 읽어 들입니다. 해당하는 타이머를 시작할 때 같이 Conversion 해주면 됩니다.

## **설정 값**

1. **ADC 1채널 파라미터**텍스트이(가) 표시된 사진

   자동 생성된 설명
2. **ADC 1채널 NVIC**

**테이블이(가) 표시된 사진

자동 생성된 설명**

1. **ADC 1 채널 DMA**

텍스트이(가) 표시된 사진

자동 생성된 설명

1. **ADC 2 채널 - TRGO**텍스트이(가) 표시된 사진

   자동 생성된 설명

ADC 1번 채널과 그 외 나머지는 같다.

1. **ADC 2 채널 – TRGO 타이머**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

타이머에서 Trigger Event Selection TRGO를 해당하는 타이머에서 활성화한다.

이리 하면 타이머 관련 이벤트가 있을 때 마다 ADC Conversion을 하게 된다.

## **ADC FSM**



* ADC INIT 상태
* 시스템에서 전원을 넣고 넘어오는 상태다.
* ADC IDLE 상태
* INIT 상태가 끝나고 아무것도 하지 않는 상태다.
* 버튼 입력 이벤트 또는 UART 시작 커맨드로 RUN 상태에 진입할 수 있다.
* ADC RUN 상태
* 자극 파형이 나오는 상태다.
* 버튼 입력 이벤트 또는 UART 정지 명령어로 AD IDLE 상태에 진입할 수 있다.
* 또한 ADC를 Conversion을 하면 ADC Conv OK 상태로 넘어간다.
* ADC Conv OK상태
* ADC의 데이터를 읽어 Conversion 한 상태다.
* 해당 상태에서는 ADC 읽기를 멈추게 되며 타이머 기반으로 돌아가는 Data Print 이벤트를 기다린다.
* 해당 상태는 Step up 관련하여 Feedback 제어를 하게 된다. Step-up Feedback 참고 바람.
* ADC Print OK상태
* Conversion 한 데이터를 타이머 기반으로 돌아가는 출력 이벤트를 통해 출력하고 난 다음 출력을 완료한 상태이다. **타이머는 100ms 마다 ADC 데이터를 출력하게 된다.**
* 따라서 ADC를 읽고 출력하는데 100ms 주기로 실행한다. 이는 Step up Feedback 제어 주기와 관련이 있다.
* 이후 다음 사이클 때 상태를 ADC RUN 상태로 전환해 다시 Conversion 한다.
* ADC ERROR 상태
* ADC에서 Error Callback이 발생한 오류로 아무런 동작이 되지 않는 상태다. 해당 상태에서는 ADC 자체를 리셋 한다.

# **ADC – Feed Back Voltage Calculation**

## **서론**

Step up에서 입력한 전압 값과 실측한 전압 값을 비교 연산하여 펄스 값을 가감하게 된다.

이에 따라 펄스 값을 가감하기 전에 Feedback 해야 할 전압 값을 계산해야 하는데 이 ADC 값이라는 것은 칩에서 다른 IO를 사용하지 않고 매우 안정된 회로에서는 3.3 V == 4095 (ADC 12bit) 같이 매우 이상적인 값이 나오지만 사용하는 시스템 전원이 불안정 하던가 아님 MCU의 전력을 다른데 소모하여 ADC 값이 괴상하게 나온다든가, 온도, 습도 등으로 각 소자가 영향을 받아 이상적인 값을 계산하기에는 매우 힘든 상황이 발생할 것이다.

이 상황을 해결하기 위해서는 **전압 값을 실측하고 이 해당하는 전압의 ADC 값을 일일이 측정해 레퍼런스 값을 만들어야 한다.**

따라서 ADC 값은 절대적인 값의 계산이 아니라 특정 전압일 때 ADC의 특정 범위에 있는 것을 실측하여 테이블을 만들어 계산하는 것이 훨씬 정확하다.

요약하자면 3.3 V == 4095 (ADC 12bit) 라는 생각을 집어치우란 말이다.

[테이블이(가) 표시된 사진

자동 생성된 설명](https://todoc1-my.sharepoint.com/:x:/g/personal/woojin_ahn_to-doc_com/EXG-wd2FqedBtaUjmH0OUzMBulbQJaiZHnURyqJCSOmFLw)

위 그림은 STEP UP 회로의 PWM 값을 변화하며 실측한 전압 값을 테이블 표로 만든 것이다. 표에 있는 것처럼 PWM 값을 변화하였을 때 전압의 MIN, MAX 값의 차이가 최대한 안정적인 테이블을 사용해야 한다. 또한 그 특정 전압일 때 ADC의 평균적인 값을 산출하여 테이블에 넣어야 한다. **(타이머 CNT 값: PWM 펄스 폭)**

## **Table로 구한 ADC값으로 전압 값 계산하기**

# **TIMER – Stimulation Pulse**

## **참고 문서**

* [test\_pwm\_delay.docx](https://todoc1-my.sharepoint.com/:w:/g/personal/woojin_ahn_to-doc_com/EUS9gLG--C1CvZ6t0egA2tsBHP3LGbVgMmf9NGsr4V8MFA), [test\_stimul\_pulse.docx](https://todoc1-my.sharepoint.com/:w:/g/personal/woojin_ahn_to-doc_com/EZ7EjmuwoF9Cn9sDO-N5arcBsq3KRXv-HFSOdnKBuICpSg) 참고



1. PWM2: TIM2 CH2 – Anode Control
2. OC4: TIM2 CH4 – Cathode Control
3. OC1: TIM2 CH1 – Current Source Control
4. OC들은 전부 PWM이 아니지만 PWM 펄스가 SW에 의해 출력이 된다.
5. PWM DMA 방식으로 구현이 되며 START CMD 또는 버튼 입력으로 시작한다.

## **조건**

1. Anode와 Cathode의 펄스폭은 PW + 2s고 d ≥ 2s 다.
2. Current 펄스폭은 “PW” 이며PWM 펄스 유형이 아니다.
3. 펄스 타이밍 계산표

|  |  |
| --- | --- |
| **Points** | **Timing** |
| S | 0 |
| T1 | s |
| T2 | Pw + s |
| T3 | Pw + d |
| T4 | Pw + d + s |
| T5 | 2pw + d + s |
| T6 | 2pw+d+2s |

# **TIMER – Step-up Pulse**

## 참고 문서

* [test\_stepup\_pulse.docx](https://todoc1-my.sharepoint.com/:w:/g/personal/woojin_ahn_to-doc_com/EYLS8CaPQa1Dkek_2qtEhTEBIbyvhbu653spdraChLK-cQ)

## **Condition**

1. 원하는 전압 값을 입력 받은 후 펄스 폭을 조정하여 전압 값을 제어합니다.
2. 전압 값을 Feed back 받아 입력한 값을 유지할 수 있도록 합니다.
3. 타이머 이벤트 발생시 ADC FSM State가 ADC\_CONV\_OK일 때 전압 값을 계산하여 Feed back 받아 펄스 폭을 가감할 수 있도록 만듭니다. (피드백)

**비고: ADC DMA Conversion 시간은 20us가 채 되지 않습니다.**

## **Description**

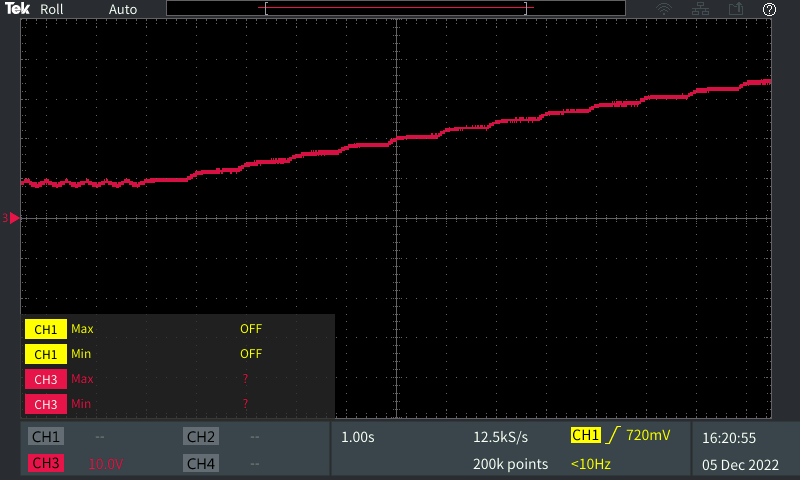


## **STEP UP VOLTAGE Description**

1. T1: Step-up voltage rising time
2. T2: Voltage keeping Time
3. T3: Step-up voltage falling time

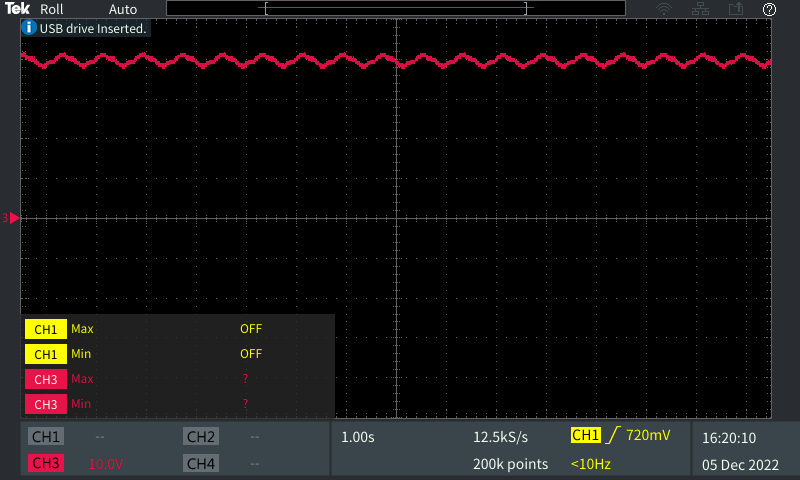
## **T1 Description**

1. 전압 값이 올라갈 때는 타이머 업데이트 이벤트가 10번 발생하여 1초 주기로 ADC 상태가 **ADC\_CONV\_OK** 일 때 펄스 폭을 증가시킵니다. \*(**ADC\_CONV\_OK: ADC 변환 완료)**
2. 입력한 전압 값에 도달하면 T2로 넘어가 100ms 주기로 Feedback 합니다.



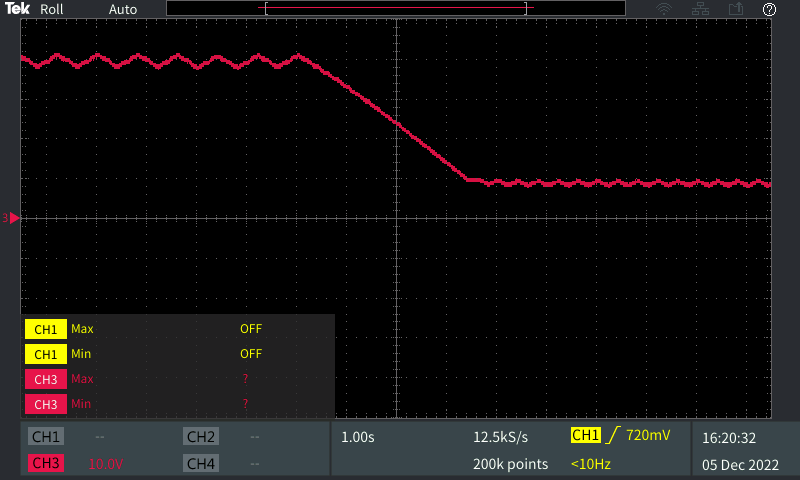
## **T2 Description**

1. A: 상한 범위 값: 42.0v
2. B: 목표 값: 40.0v
3. C: 하한 범위 값: 38.0v
4. D: Feedback 주기: 100ms
5. EFFECT RANGE: 1.4v
6. 타이머 업데이트 이벤트가 100ms 주기로 ADC 상태가 ADC\_CONV\_OK 일 때 펄스 폭을 가감합니다. (피드백)



## **T3 Description**

1. 전압 값이 내려갈 때는 타이머 업데이트 이벤트가 100ms 주기로 ADC 상태가 ADC\_CONV\_OK 일 때 펄스 폭을 감소시킵니다. (피드백)



# **TIMER – Peak Detection**