Functional Electrical Stimulation (FES)

Final Report

Chua Wang An
S10078666G

Lim Zi Chuan
S10075204K

Supervisors: Mr Chua Kok Poo
Mdm Tan Peck Ha

Year: 2011

Abstract

It has been known that long leg braces given to stroke patients frequently end up at home catching dust. They are difficult to don and remove. As the patients recover and return to the community independent in their wheelchairs, they will not bother to wear their braces. Some stroke patients use them occasionally for stretching knee and hip joints, but in general paraplegics do not use them to walk, even with assistive devices.

Thus, we decided to develop a computerized stimulator to furnish electro-stimulation to the paralyzed muscles of persons who have suffered traumatic upper motor neuron lesions of the spinal cord affecting their lower limbs and thus preventing them from standing and walking.

Contents

1. Objectives

- Research on stroke and its effects
- Research on spinal cord injuries and its effects
- Research on Human Central Nervous System (CNS)
- Research on physiology of nerve excitation
- Hardware implementations

2. Background research

2.1 Analysis of stroke

2.1.1 What is Stroke?

A stroke occurs when a blood clot blocks a blood vessel or artery, or when a blood vessel breaks and interrupts blood flow to an and bleeding occurs into an area of the brain.

Every stroke is different. The symptoms and effects vary according to the type of stroke, the part of the brain affected and the size of the damaged area. For some people the effects are severe, for some mild. Usually the symptoms come on suddenly but they may come on during sleep. Usually injury to one side of the brain affects the opposite side of the body.

2.1.2 Two major types of stroke – Haemorrhagic and Ischaemic

There are two major types of stroke:

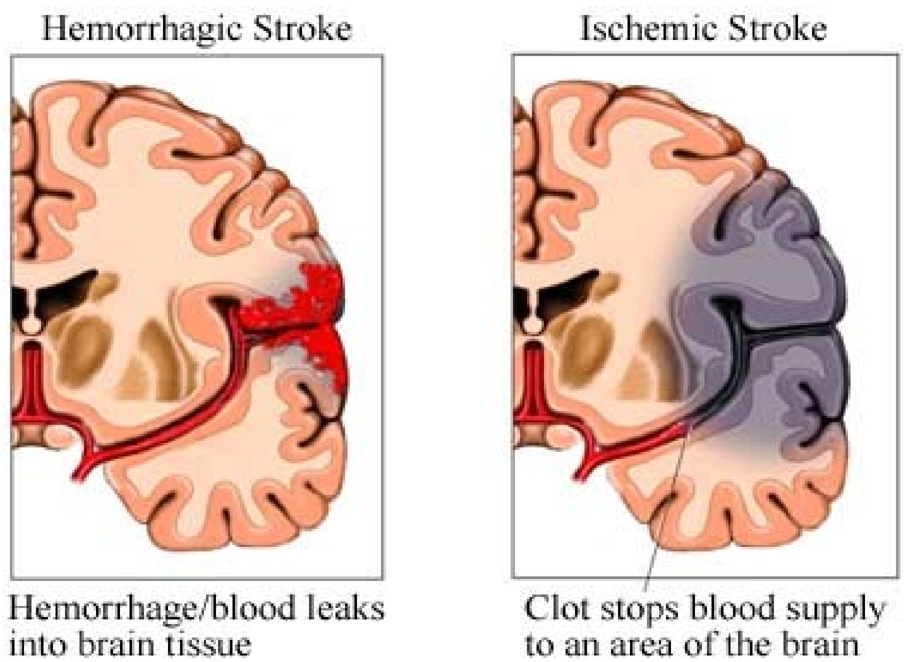


Figure 1: Haemorrhagic Stroke

- A **Haemorrhagic Stroke** Cerebral Haemorrhage occurs when a blood vessel ruptures within the brain (called an intra-cerebral haemorrhage) or into the space surrounding the brain (called a subarachnoid haemorrhage). Blood in the artery is under pressure and so, as it spurts out, it tears some of the soft brain tissue and forms a large clot (or haematoma) that squashes the surrounding brain. Brain tissue on the rim of thein and around the clot may therefore die.

- An **Ischaemic Stroke** occurs when an artery carrying blood to part of the brain is blocked. The brain needs the constant supply of oxygen and glucose that the blood brings. If this blood supply is blocked for more than a few minutes then that part of the brain stops working properly and brain tissue at the centre of the area affected begins to die. If the blockage is not cleared within a few hours then that all the part of the brain supplied by the blocked vessel may die; that is, it permanently ceases to work properly. This is called brain infarction. Ischaemic strokes are the most common type of stroke, occurring more than five times as often as haemorrhagic stroke cerebral haemorrhages.

2.1.3 Stroke in different position of brain

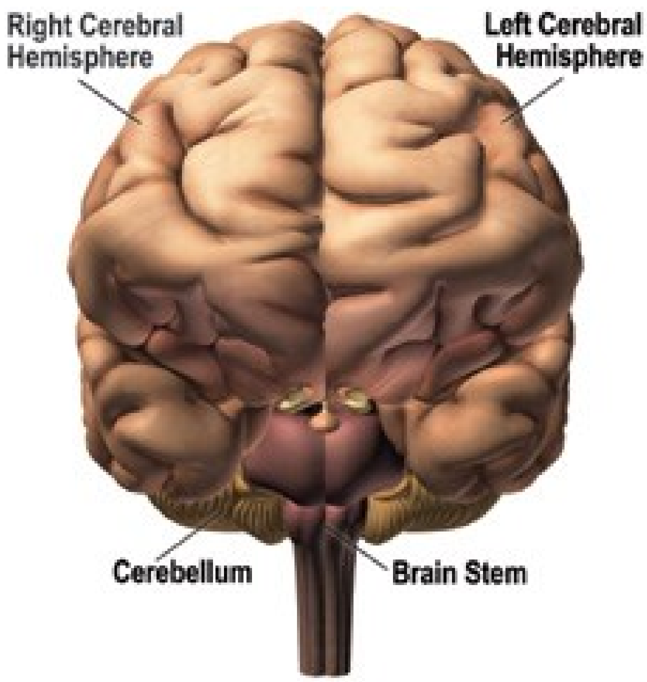


Figure 2: Frontal view of brain

Right Hemisphere Strokes

The right hemisphere of the brain controls the movement of the left side of the body so stroke in the right hemisphere often causes paralysis in the left side of the body. This is known as left hemiplegia.

Survivors of right-hemisphere strokes may also have problems with their spatial and perceptual abilities. This may cause them to misjudge distances (leading to a fall) or be unable to guide their hands to pick up an object, button a shirt or tie their shoes. They may even be unable to tell right side up from upside-down when trying to read.

Along with these physical effects, survivors of right-hemisphere strokes often have judgment difficulties that show up in their behaviour. They often act impulsively, unaware of their impairments and certain of their ability to perform the same tasks as before the stroke. This can be extremely dangerous. It may lead them to try to walk without aid or to try to drive a car.

Survivors of right-hemisphere strokes may also experience left-sided neglect. This is a result of visual difficulties that cause them to "forget" or "ignore" objects or people on their left side.

Some survivors of right-hemisphere strokes will experience problems with short-term memory. Although they may be able to recall a visit to the seashore that took place 30 years ago, they may be unable to remember what they ate for breakfast that morning.

Left Hemisphere Strokes

The left hemisphere of the brain controls the movement of the right side of the body. It also controls speech and language abilities for most people. A left-hemisphere stroke often causes paralysis of the right side of the body. This is known as right hemiplegia.

Someone who has had a left-hemisphere stroke may also develop aphasia. Aphasia is a catch all term used to describe a wide range of speech and language problems. These problems can be highly specific, affecting only one part of the patient's ability to communicate, such as the ability to move their speech-related muscles to talk properly. The same patient may be completely unimpaired when it comes to writing, reading or understanding speech.

In contrast to survivors of right-hemisphere stroke, patients who have had a left-hemisphere stroke often develop a slow and cautious behaviour. They may need frequent instruction and feedback to finish tasks.

Patients with left-hemisphere stroke may develop memory problems similar to those of right-hemisphere stroke survivors. These problems can include shortened retention spans, difficulty in learning new information and problems in conceptualising and generalising.

Cerebellum Strokes

The cerebellum controls many of our reflexes and much of our balance and coordination. A stroke that takes place in the cerebellum can cause abnormal reflexes of the head and torso, coordination and balance problems, dizziness, nausea and vomiting.

Brain Stem Strokes

Strokes that occur in the brain stem are especially devastating. The brain stem is the area of the brain that controls all of our involuntary functions, such as breathing rate, blood pressure and heart beat. The brain stem also controls abilities such as eye movements, hearing, speech and swallowing. Since impulses generated in the brain's hemispheres must travel through the brain stem on their way to the arms and legs, patients with a brain stem stroke may also develop paralysis in one or both sides of the body.

2.1.4 What are the Effects of Stroke?

- Weakness or lack of movement (paralysis) in legs and/or arms
- Shoulder pain
- Trouble swallowing
- Changes to way things are seen or felt (perceptual problems)
- Changes to the way things are felt when touched (sensory problems)
- Problems thinking or remembering (cognitive problems)
- Trouble speaking, reading or writing
- Incontinence
- Feeling depressed
- Problems controlling feelings
- Tiredness

The specific abilities that will be lost or affected by stroke depend on the extent of the brain damage and, most importantly, where, in the brain, the stroke occurred: the right hemisphere (or half), the left hemisphere, the cerebellum or the brain stem.

2.2 Spinal cord injury

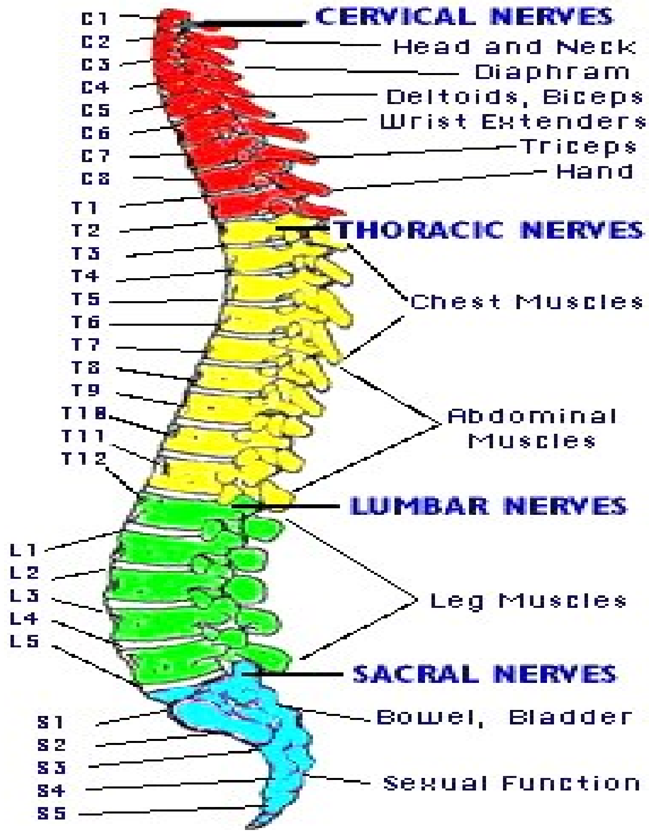


Figure 3: Anatomy of spinal cord

The spinal cord injury is a traumatic lesion of the spinal cord and its associated nerves.

Spinal nerves originate from all levels of the spinal cord, as follows:

C-1 to C-8 (cervical nerves);

T-1 to T-12 (thoracic nerves);

L-1 to L-5 (lumbar nerves);

S-1 to S-5 (sacral nerves)

2.2.1 What are the effects of Spinal Cord injury?

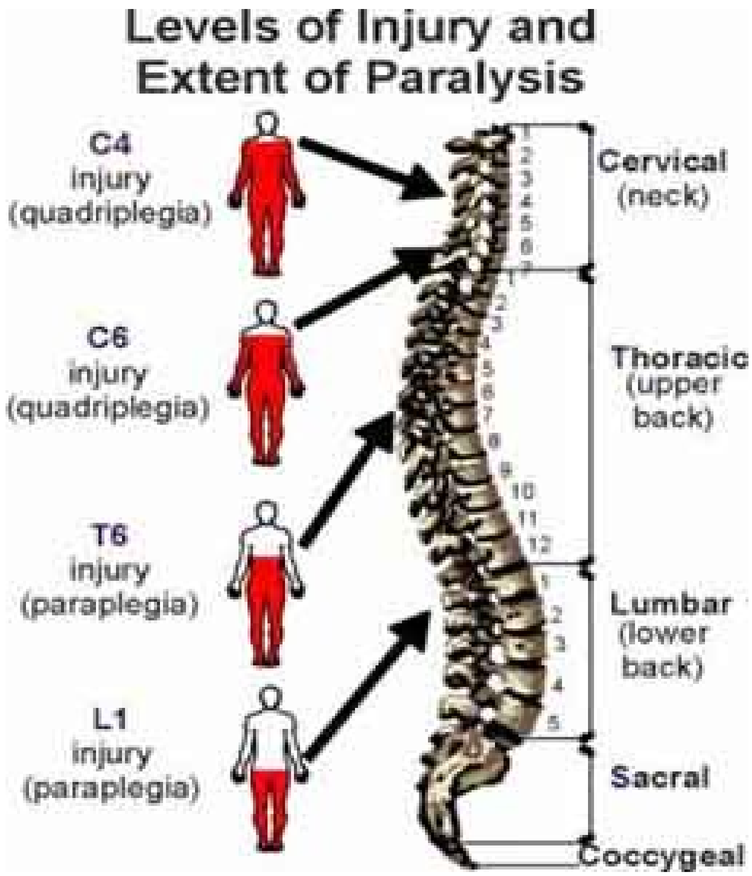


Figure 4: Anatomy of spinal cord

Neck (cervical) injuries usually result in quadriplegia. (C1 - C4 level): often require a ventilator to breathe. Usually results in hands and finger paralysis.

Spinal cord injuries to the thoracic level (T1 to T12) and below result in **paraplegia**: Usually losing control in lower part of the body. Along with the loss of sensation and motor functioning, people with spinal cord injuries experience other changes. A loss of bowel and bladder control may occur and sexual functioning is commonly affected. Other effects of spinal cord injury may include low blood pressure, reduced control of body temperature, inability to sweat below the level of injury and chronic pain.

2.3 The human Central Nervous System (CNS)

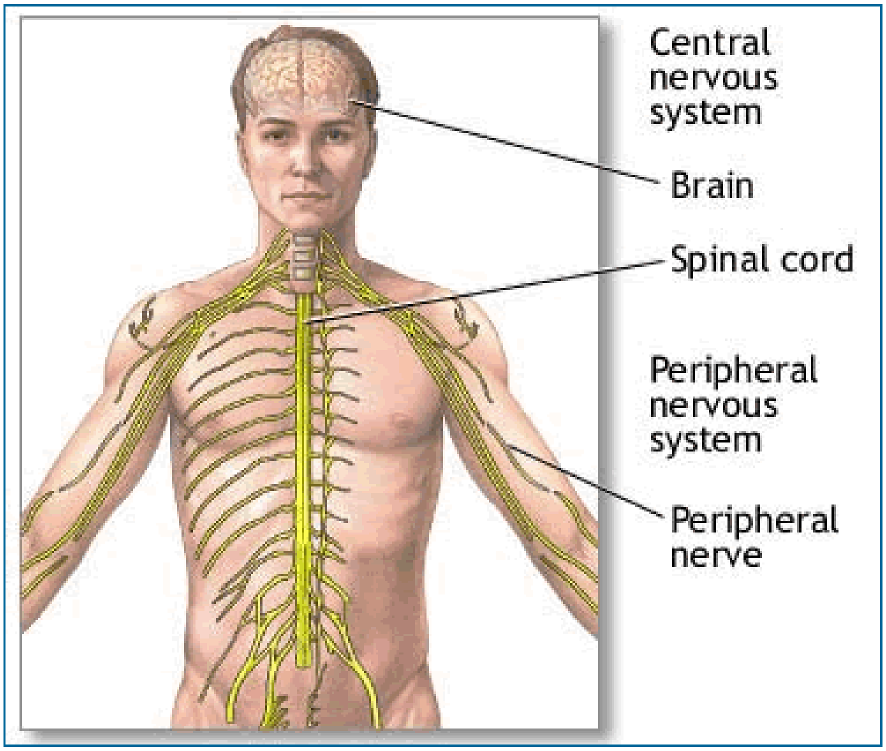


Figure 5: Human Central Nervous System

The Central Nervous System (CNS) consists of the brain and the spinal cord and is in charge of sending and receiving signals from the brain to all other nerve bundles in the body. It is important to understand that muscles contract not simply because we want to lift something heavy or run really fast, but because the nerve bundles that control the motor units (bundles of muscle fibers) tell it to contract. Therefore, when the mind wants the limbs to move, it sends electrical signals down through the spinal cord, from nerve to nerve until the signal makes its way to the appropriate muscle and actually stimulates it.

2.4 The human motor system

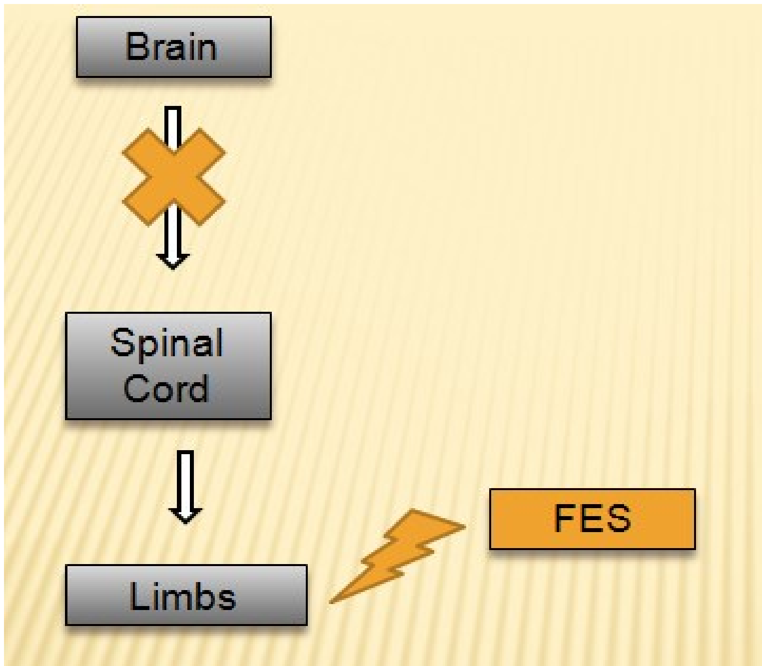


Figure 6: human motor system

Spinal cord injury or stroke results in paralysis. The figure above shows that motor neurons from the brain are not able to reach the limb muscles. However, the muscles are intact and functional while still being able to contract. Therefore by using external electrical pulse to stimulate the muscle or nerve group will enable it to contract and move. Using electricity to stimulate and control a part of the body is also known as Functional Electrical Stimulation or FES.

3. Introduction of FES

3.1 Project Description

Functional Electrical Stimulator (FES) artificially generates neural activity to activate muscles to overcome impaired muscles functions in paralyzed persons.

In most cases, to control body movement and perform basic bodily functions, electrical signals travel from the brain down through the spinal cord to the corresponding muscles or organs. When damage occurs to the spinal cord or when the person suffers a major stroke, he often loses control of his body. The electrical signals from the brain cannot reach the intended destination. The muscles and organs may be intact and healthy, but the brain cannot stimulate or control them. However, by-passing the spinal cords and brain by applying a small electrical current directly to the intended nerve or muscle group, the desired function can be triggered.

Stimulating the correct muscles can make a person's limbs move, while stimulation to the bladder or diaphragm can return a basic function. Using electricity to stimulate and control a part of the body is known as Functional Electrical Stimulation or FES.

Hence, in order to solve this problem, a programmable functional electrical stimulator controlled by a micro-processor is proposed. The stroke patient will have the stimulator attached to the impaired muscle group and this system will be able to send in pulse of current to 'exercise' the muscle group in order for it to move again.

3.2 Project Statement

Design and implement a system which generates an electrical pulse to aid the user in stimulating the targeted muscles group. It will aid the stroke victims reduce muscle atrophy by constantly exercising the targeted muscle group and eventually aiding them to stand and walk again.

3.3 System Specifications

- Step up regulator to step up voltage from 6V – 200V
- A current control circuit to drive a pulse into through the electrodes into the patient's muscles
- Design a micro-controller system
- A micro-processor for overall control
- Uses 4 'AA' batteries

3.4 Summary & Future

Developing an orthotic device used to attach to the targeted muscle group of the limb. It will be controlled with a programmed controller which can adjust the intensity of the electrical pulse for stimulation of the muscles.

Figure 7: Final product visualization

4. Project planning

4.1 Work Breakdown Structure (WBS)

		Resources Requirements	Comments
1	**Project Planning**		
1.1	Understanding about FES and different type of switching and step up circuit	Internet, reference books	Research online (google)
1.2	Project proposal	Laptop (Microsoft word)	
1.3	Purchase components		Element14, RS, PSU
2	**Hardware Design & Implementations**		
2.1	Design/implement step up regulator	Power inductor, resistor, capacitor, MOSFET, Altium Designer	Purchase from element 14
2.2	Design/implement switching circuit	H-bridge using Dual channel MOSFET, Altium Designer	Supertex TC6320
2.3	Design/Implement a current drive circuit	Using transistors and Op-amp, Altium Designer	With feedback to the micro-controller
2.4	Design/Implement an overall micro-controller system	MSP programming	PSU, Farnell, RS
2.5	Integration of all hardware components	Altium Designer	
3	**Microprocessor**		
3.1	Programming	Internet, books, lecturer	MSP programming
3.3	Troubleshoot the program		Laptop

4.2 Responsibility Assignment Matrix (RAM)

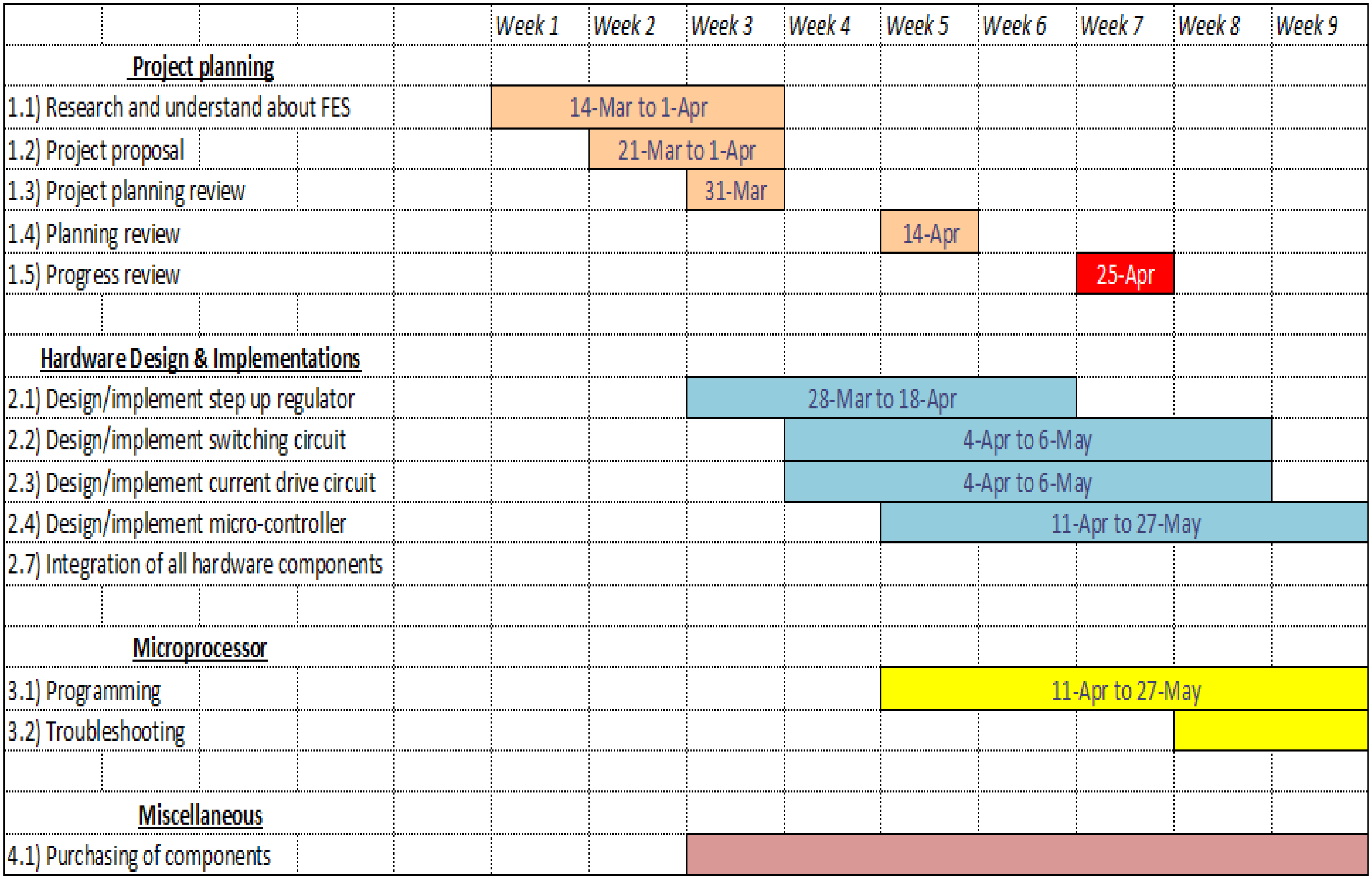
No.	Tasks	WangAn	ZiChuan
1	**Project Planning**		
1.1	Understanding about FES and different type of switching and step up circuit	P	P
1.2	Project proposal	P	S
1.3	Purchasing of components	S	P
2	**Hardware design**		
2.1	Design/implement step up regulator	P	P
2.2	Design/implement switching circuit	P	P
2.3	Design/Implement a current control circuit	S	P
2.4	Design/Implement an overall micro-controller system	P	S
2.5	Integration of all hardware components	P	P
3	**Microprocessor implementation**		
3.1	Programming	P	P
3.2	Troubleshoot the program	S	P

4.3 Precedence list

Task code	Work Product Description	Prerequisite	Duration*(day)*
1	**Project Preparation**		
1.1	Understanding about FES and different type of switching and step up circuit	-	7
1.2	Project proposal	After 1.1	1
1.3	Purchasing of components	After 1.2	60
2	**Hardware Design**		
2.1	Design a step up regulator	After 1.2	3
2.2	Design a switching circuit	After 2.1	3
2.3	Design t a current control circuit	After 2.2	3
2.4	Design an overall micro-controller system	After 2.3	3
2.5	Integration of all hardware components	After 2.4	10
3	**Software Design and Implementation**		
3.1	Study of MSP430 LaunchPad	After 2.5	5
3.2	Programming timer from MSP430G2231	After 3.1	3
3.3	Test and troubleshooting	After 3.2	1
4	**Implementation of Step up regulator**		
4.1	Calculation & Measurement of the inductor and capacitor (Including frequency to use)	After 1.2	3
4.2	Draft-up for step up regulator	After 4.1	3
4.3	Breadboard testing of step up regulator	After 4.2	1
4.4	New improve design of step up regulator	After 4.3	2
4.5	Breadboard testing of improved step up regulator	After 4.4	1
4.6	Troubleshoot	After 4.5	2
4.7	Design of schematic and PCB	After 4.6	3

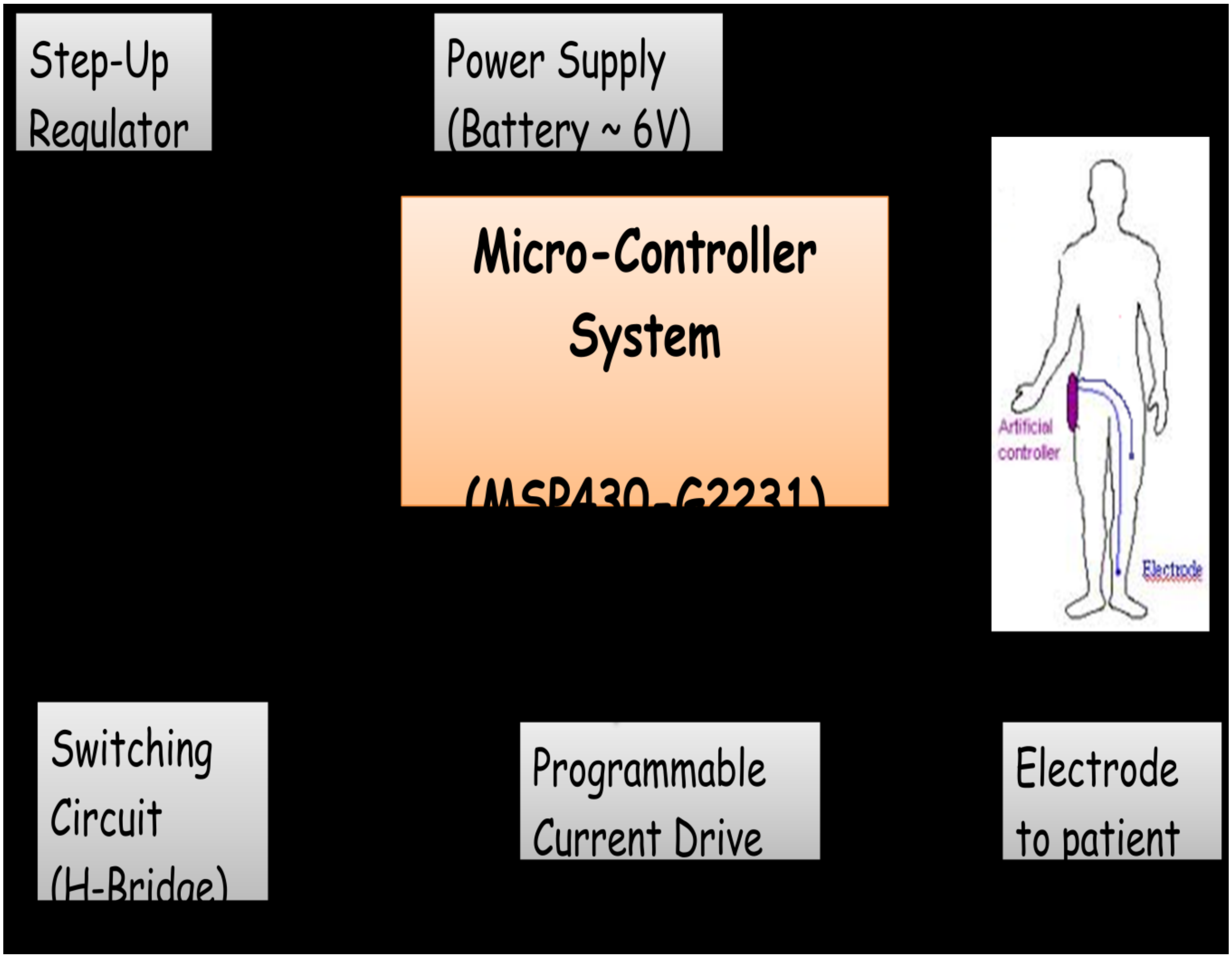
5	**Implementation of H-bridge**		
5.1	Circuit design	After 2.2	3
5.2	Calculation of resistor value to use	After 5.1	3
5.3	Implementation of H-bridge driver	After 5.2	3
5.4	Integration of driver + H-bridge	After 5.3	4
5.5	Testing and troubleshooting	After 5.4	5
5.6	Design of schematic and PCB	After 5.5	3
6	**Current control circuit**		
6.1	Circuit design	After 2.3	3
6.2	Breadboard testing	After 6.1	3
6.3	Testing with overall integrated circuit	After 6.2	3
6.4	Troubleshoot	After 6.3	4
7	Design of overall schematic and PCB		
8	Overall Troubleshoot	After 6.4	4
9	Overall Integration	After 7	4
10	**Final Review**		
10.1	Final report	After 9	-
10.2	Final Presentation preparation	After 9	-

4.4 Schedule Chart



	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16	Week 17	Week 18	Week 19	Week 20
Project planning											
1.1) Research and understand about FES											
1.2) Project proposal											
1.3) Project planning review											
1.4) Planning review											
1.5) Progress review											
Hardware Design & Implementations											
2.1) Design/implement step up regulator											
2.2) Design/implement switching circuit											
2.3) Design/implement current drive circuit											
2.4) Design/implement micro-controller											
2.7) Integration of all hardware components						16-May to 25-Jul					
Microprocessor											
3.1) Programming											
3.2) Troubleshooting						2-May to 25-Jul					
Miscellaneous											
4.1) Purchasing of components					28-Mar to 1-Jul						

4.5 System Block Diagram



5. Calculations

5.1.1 Frequency for Step-Up Regulator (0-200V)

Calculation for resonance frequency

$$\omega = \frac{1}{\sqrt{LC}}$$

W = 1 / √ [(44x10^-6 H) x (2x10^-6 F)]

= 106600 Hz = 106K Hz

5.1.2 Frequency Calculation for Step-Up Regulator (555 timer)

Frequency of the pulses is:

$$f = \frac{1.44}{(R1 + 2R2) \times C}$$

The period of pulses, t, is given by:

$$t = \frac{1}{f} = 0.69(R1 + 2R2) \times C$$

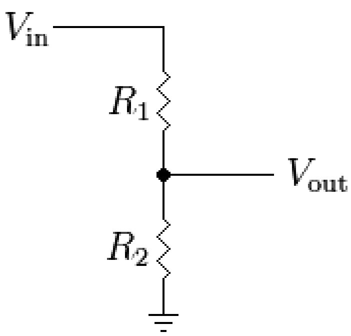
Duty Cycle of the pulses:

$$\text{duty cycle} = \frac{\text{HIGH time}}{\text{pulse period time}}$$

The HIGH and LOW times of each pulse can be calculated from:

$$\text{HIGH time} = 0.69(R1 + R2) \times C \qquad \text{LOW time} = 0.69(R2 \times C)$$

5.2 Resistor Calculation for feedback current and voltage



$$V_{\text{out}} = \frac{R_2}{R_1 + R_2} \cdot V_{\text{in}} \qquad I = \frac{V}{R}$$

Output Current Control

a) V=IR

Let Load between electrode = 1K ohm

Output Voltage 200V //Step up Votlage

Iout= 200V/ 1K Ω = 200mA // Maximum output current to from the electrode

Vref = 3V

Iref = 200mA

R= V/I

R= 3V/200mA = 15 Ω // Resistor use for feedback of the op-amp for current control.

b) Voltage Control for Step up Regulator

Calculation: Vout = 200V

Vref = 3V

Let Iref = 0.1mA //Constant Current

R2 = 3V / 0.1mA = 30K Ω

Using Voltage Divider Rule:

Vout = [R2/(R1+R2)] x Vin

200V = [30K / (R1 +30K)] x 3V

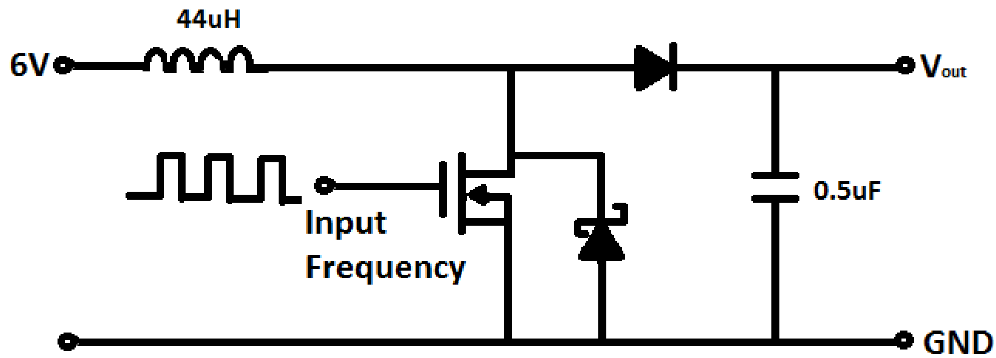
Therefore;

R1 = 2M Ω, R2 = 15 Ω // Resistor use for feedback of the op-amp for voltage control

6. Hardware Implementations

6.1 Step-up regulator

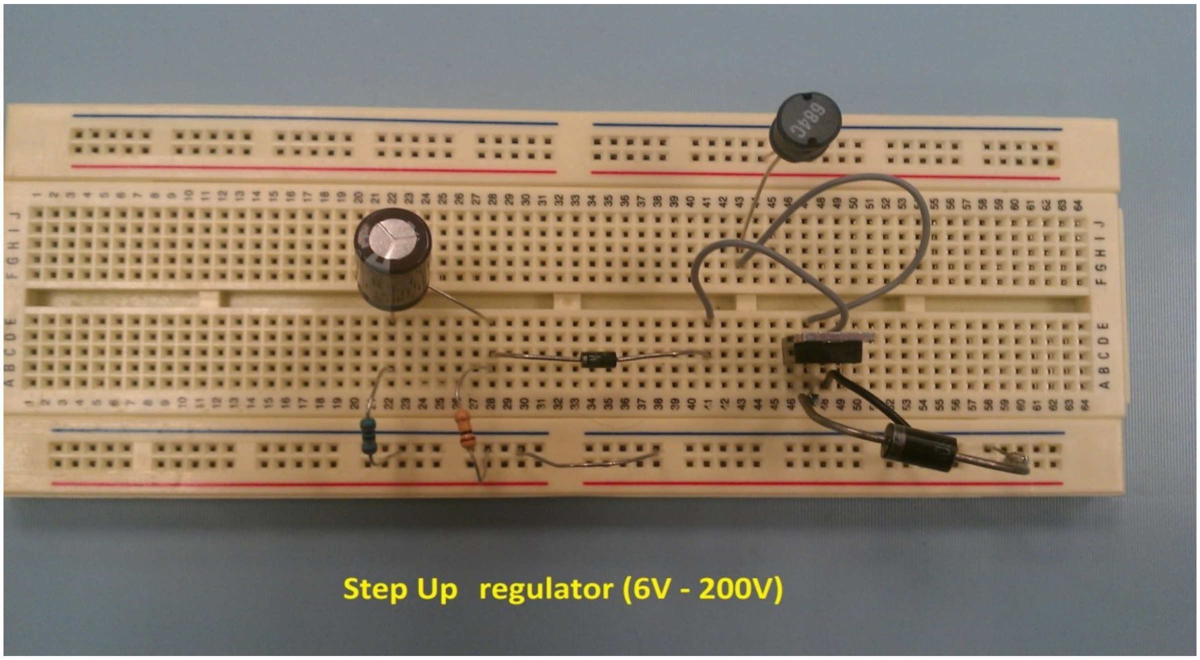
6.1.1 Simple step-up regulator circuit diagram



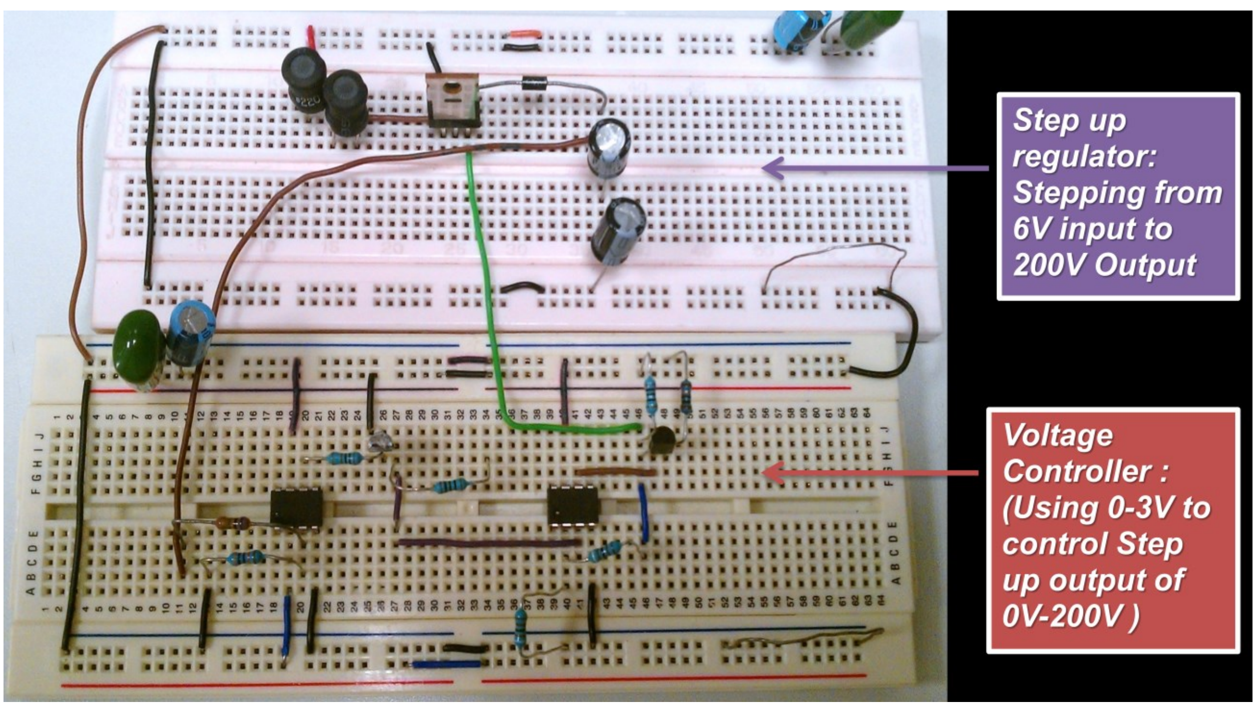
Firstly, we use power inductor to step up voltage from 6V to 200V. The circuit diagram above shows the step-up regulator implemented in our design.

Following that, we recorded the waveform of the output and confirmed that the step-up regulator from 6V-200V is working and can be implemented into our overall project design.

6.1.2 Bread-boarding of step-up regulator

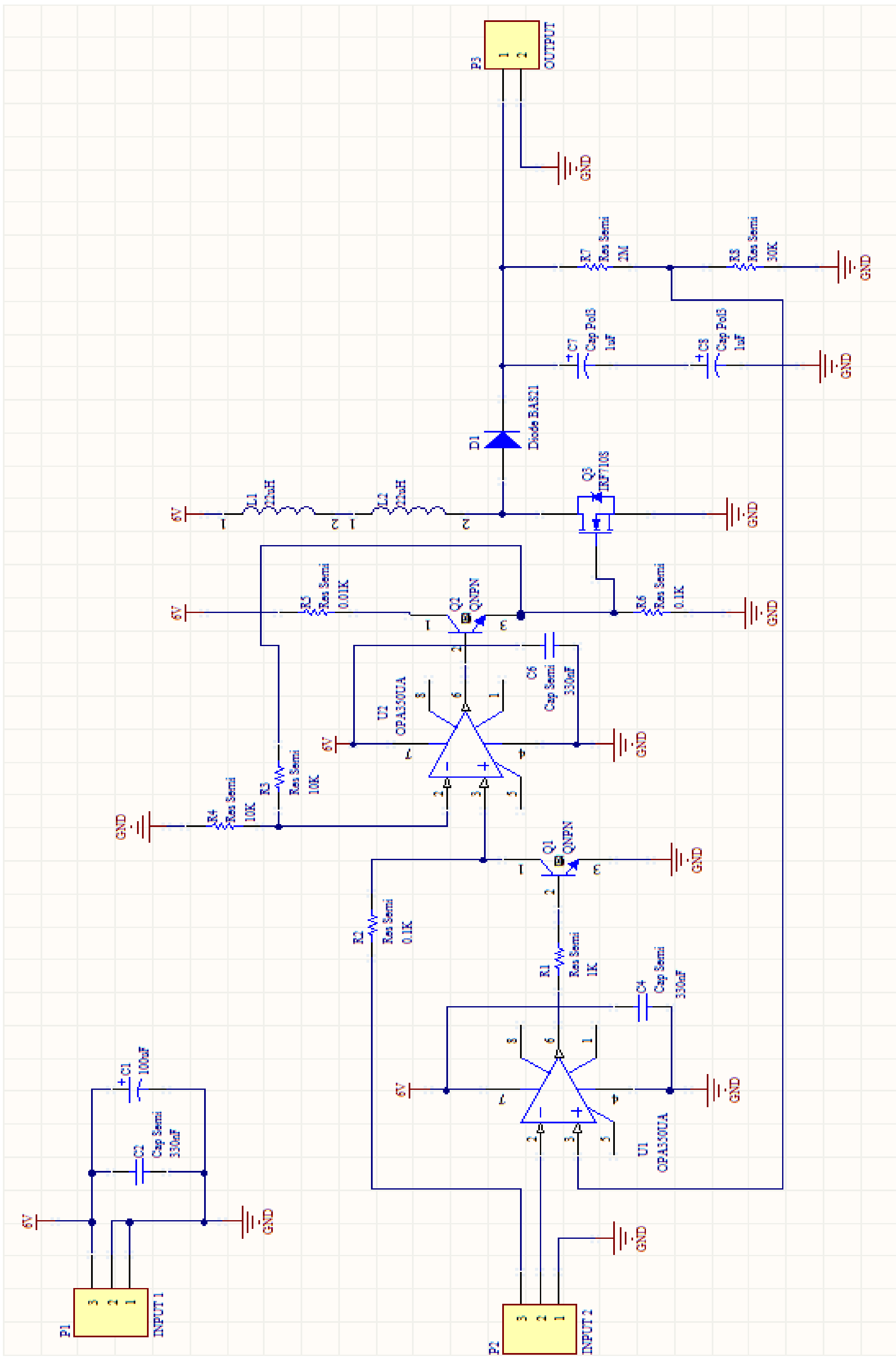


Initial design

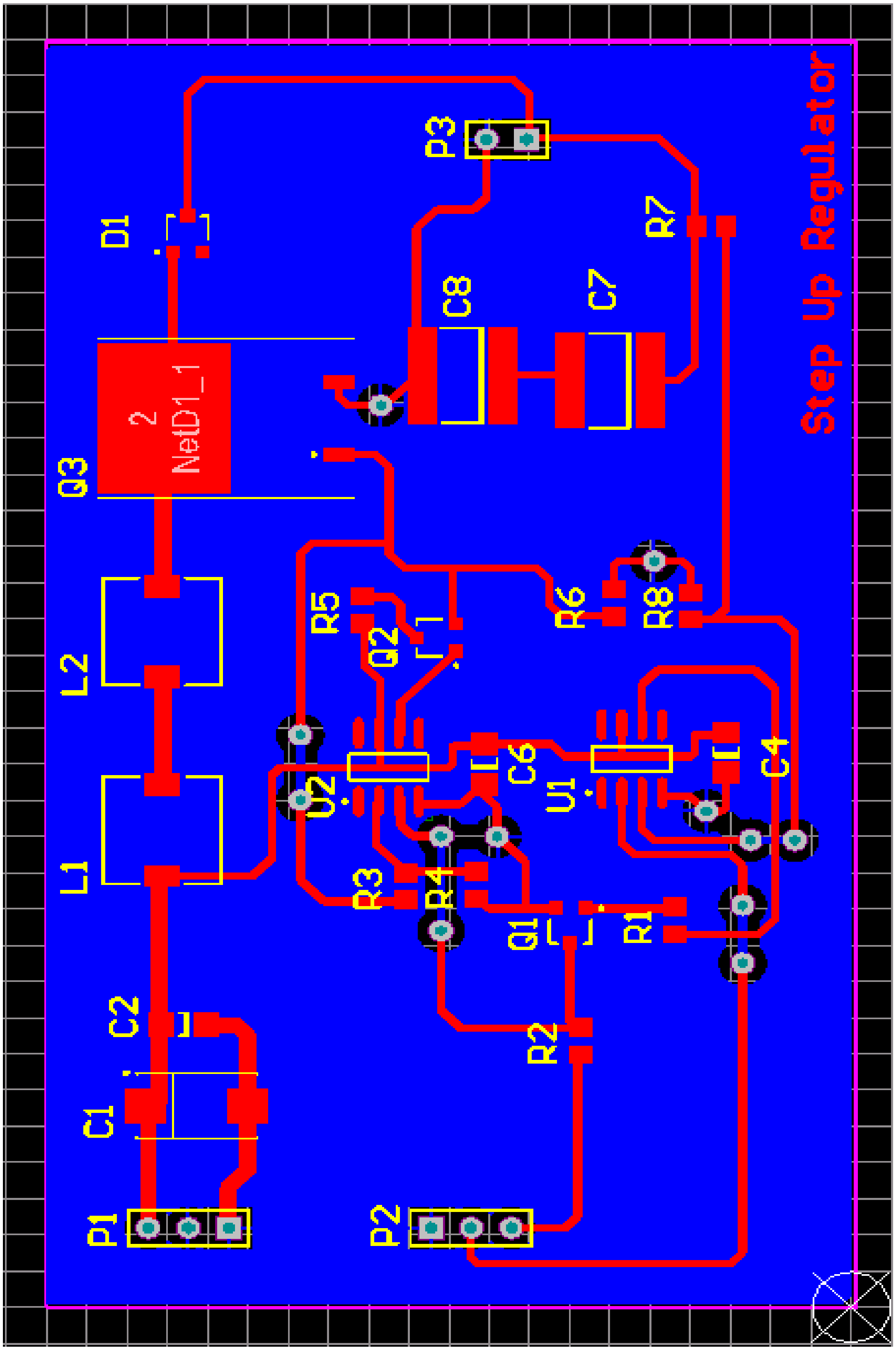


New improved design (with voltage regulator)

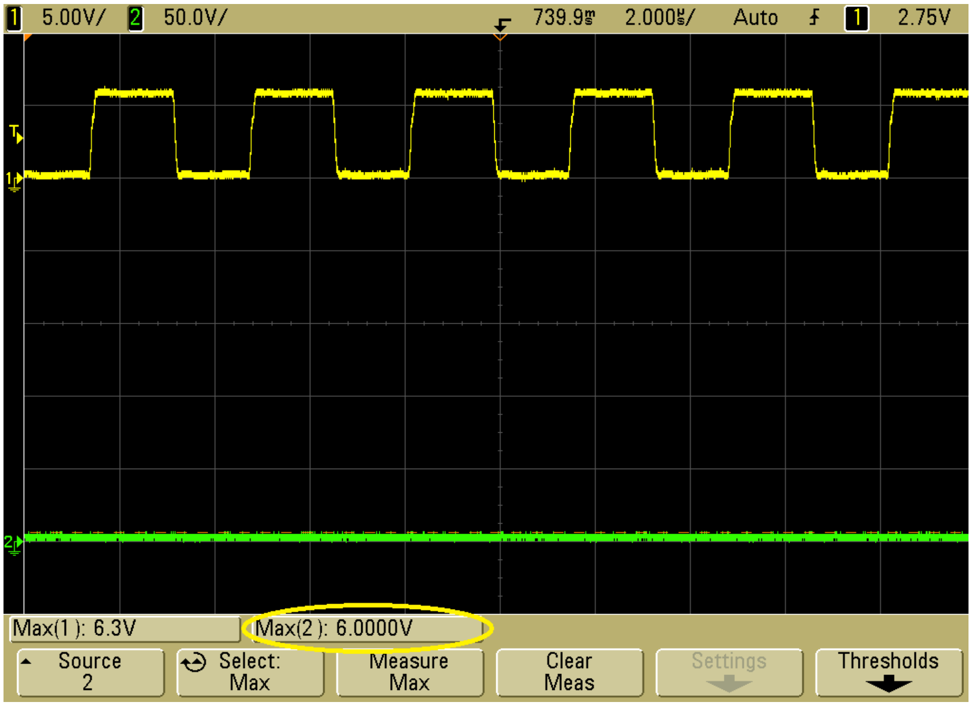
6.1.3 Schematic of step-up regulator



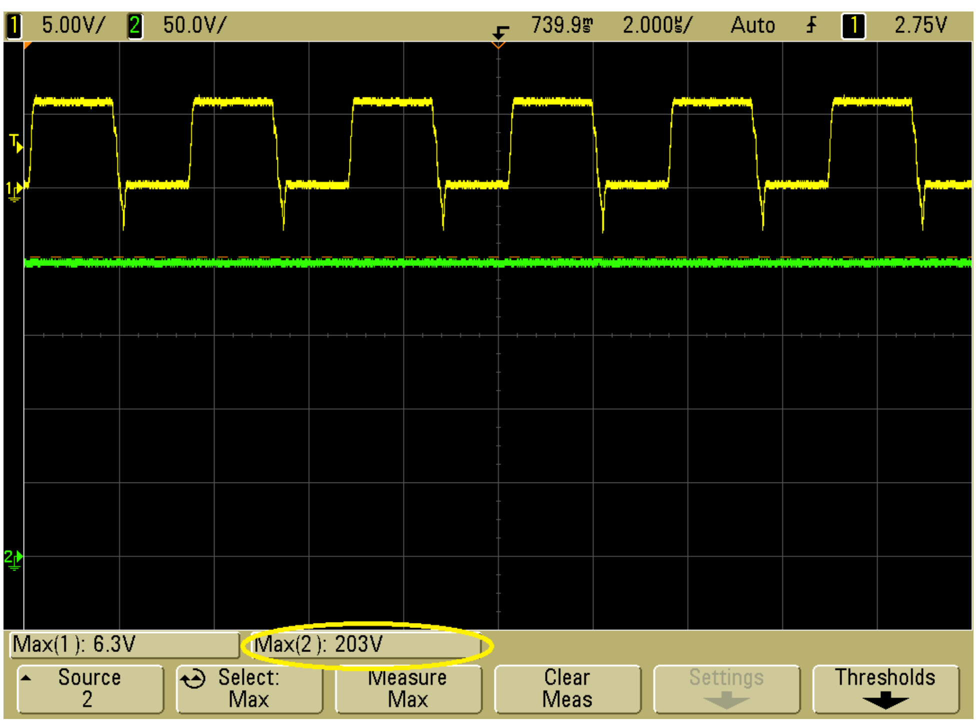
6.1.4 PCB layout of step-up regulator



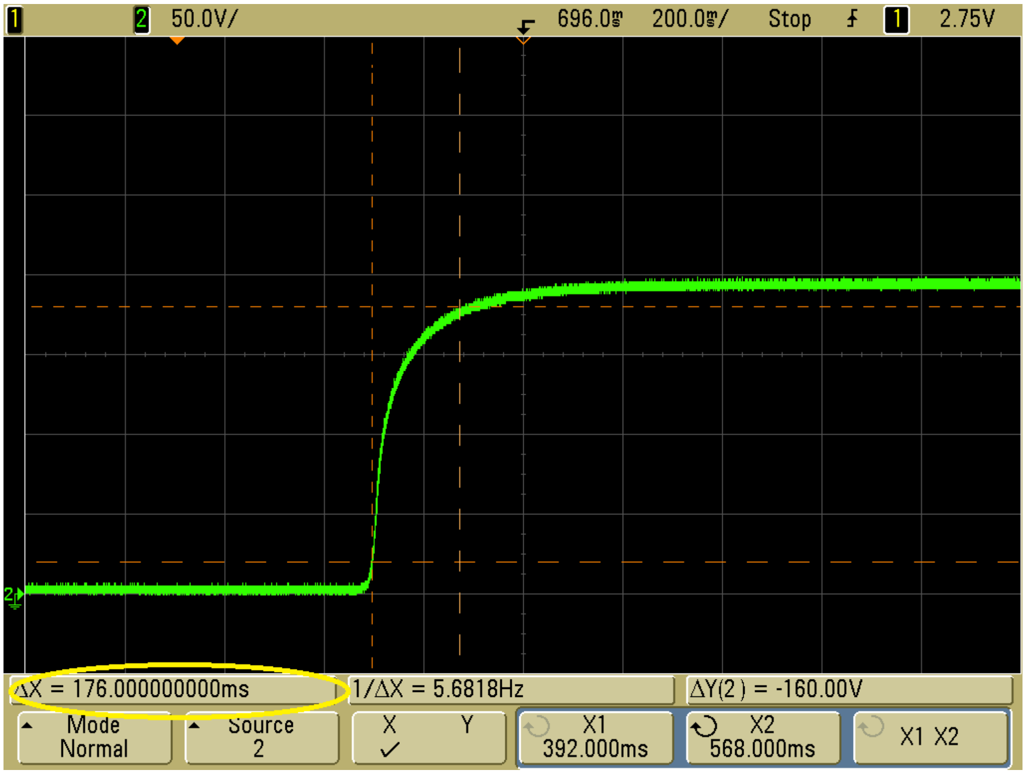
6.1.5 Waveform of step-up regulator



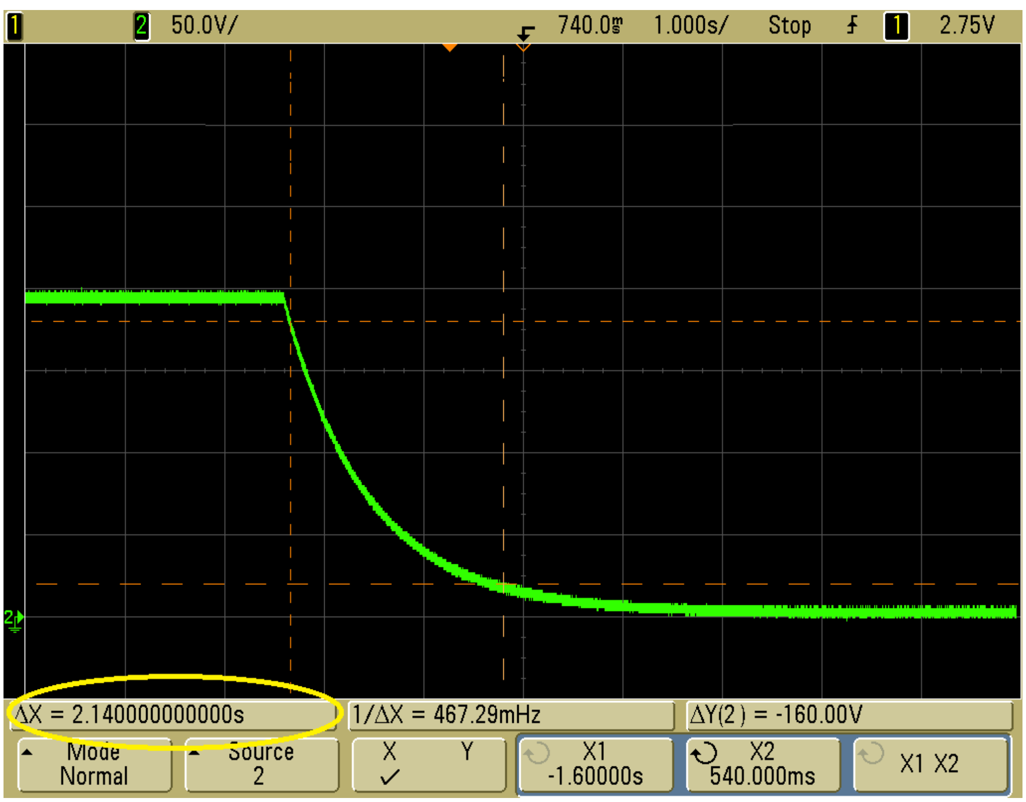
Before Stepping up



After Stepping up



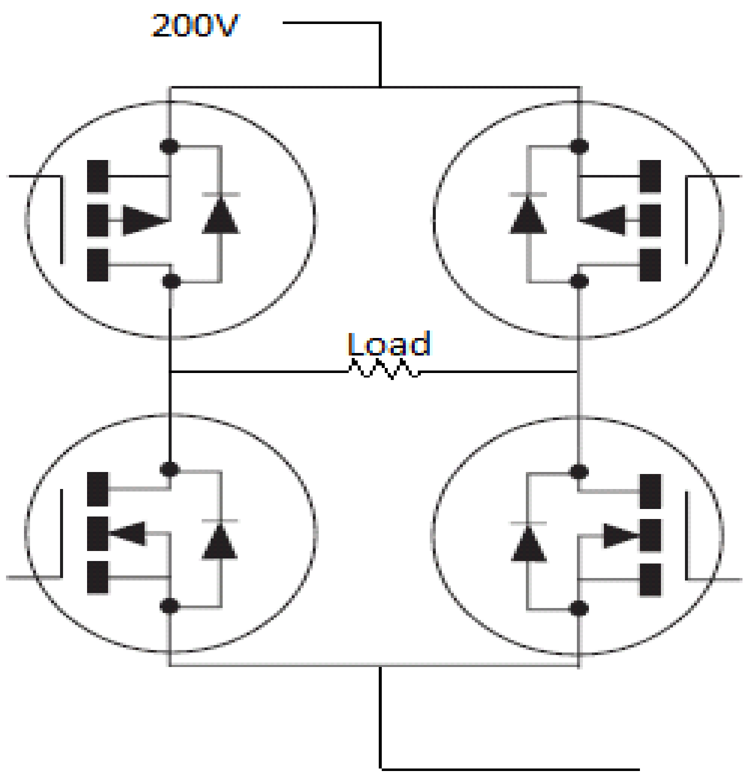
Charging time



Discharging time

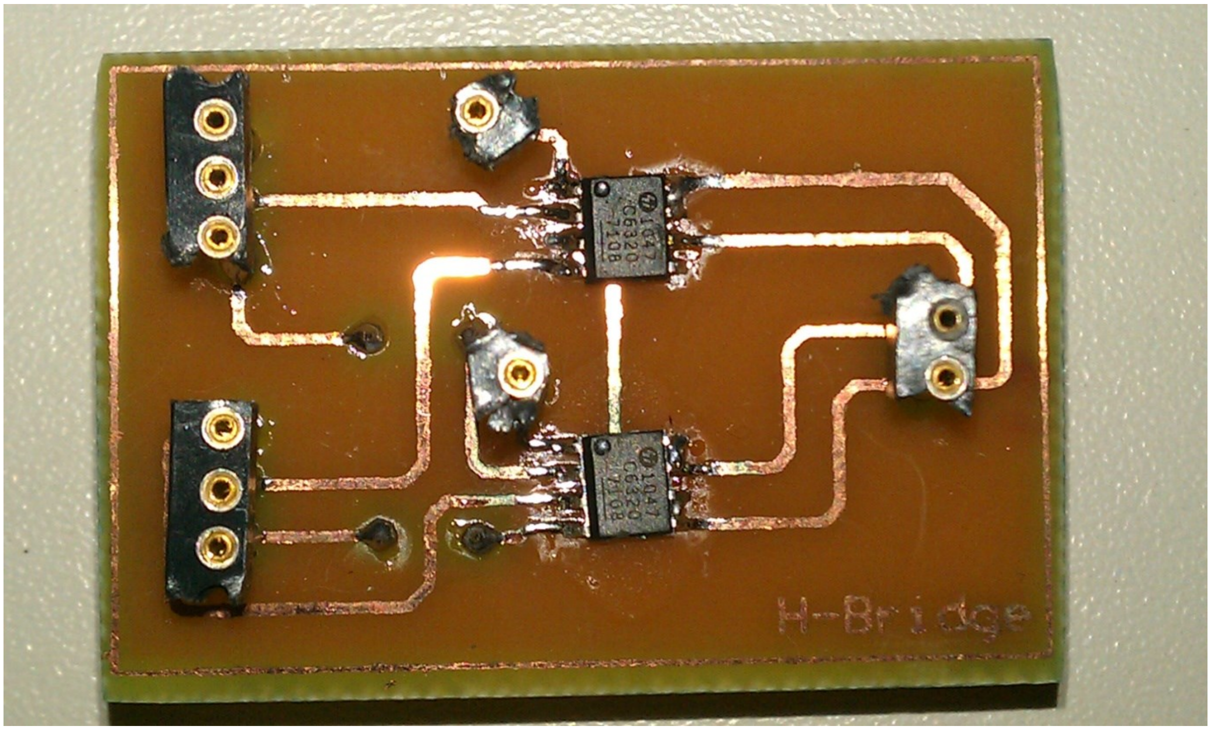
6.2 Switch (H-bridge)

6.2.1 Simple switching circuit diagram



An H-bridge is an electronic circuit which enables a voltage to be applied across a load in either direction. These circuits are often used in robotics and other applications to allow DC motors to run forwards and backwards.

6.2.2 Circuit design of H-bridge

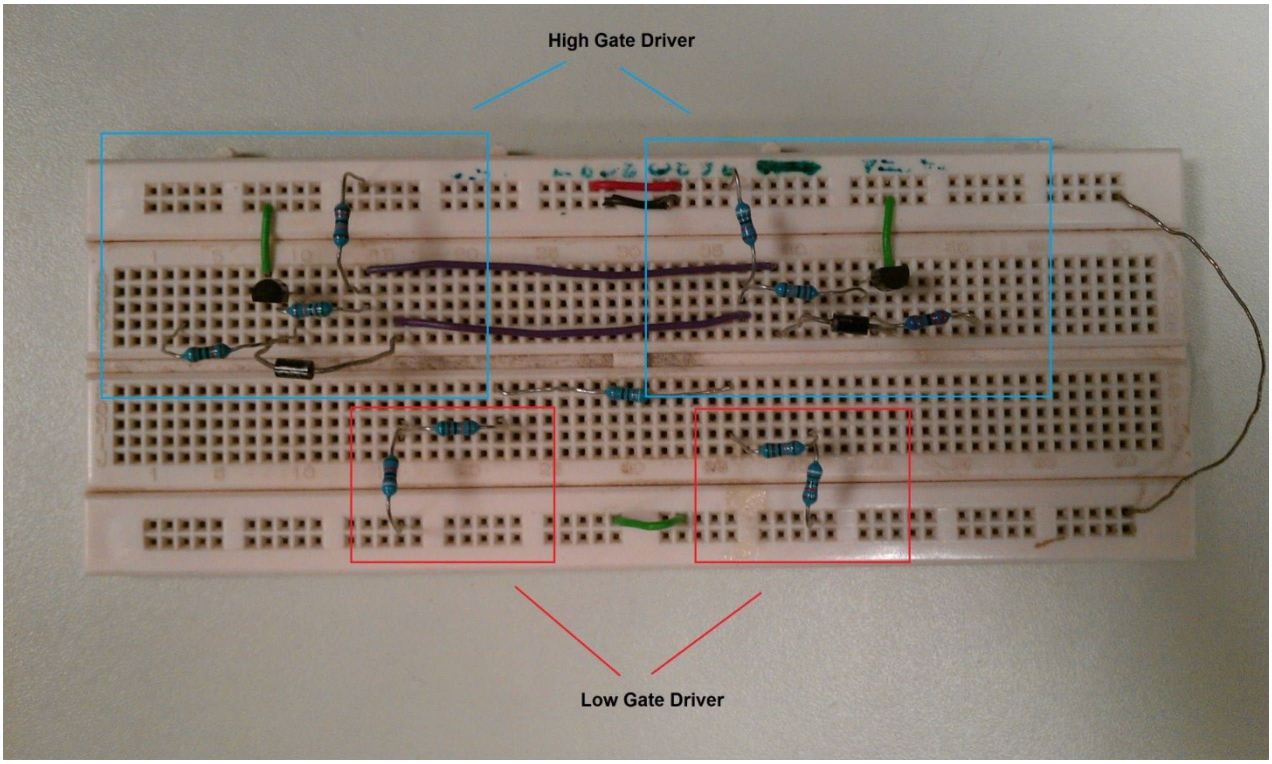


Due to the high breakdown voltage (200V) that we need, we could not find an overall IC chip that has the specifications we needed and the whole of H-bridge.

Thus, we decided to buy half bridge IC chip with breakdown voltage of 200V and integrate it into a full bridge.

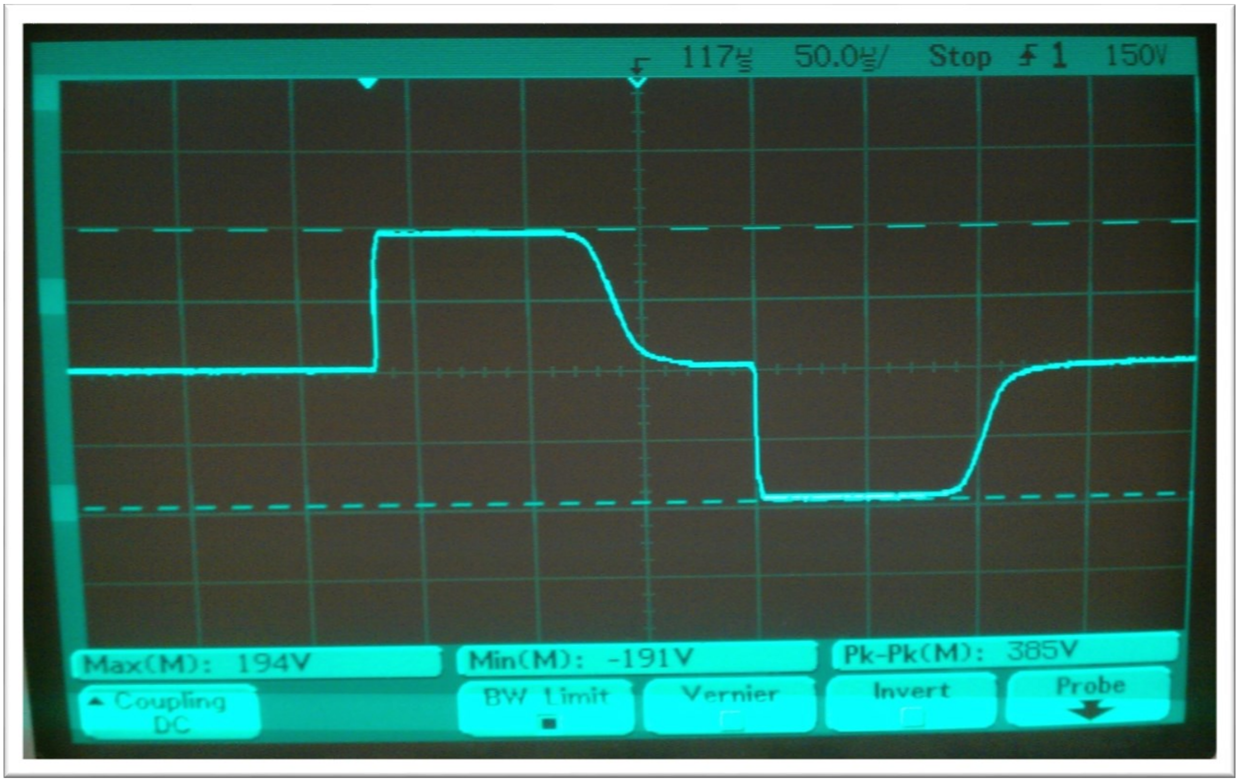
Furthermore, this model is not manufacturing anymore, thus it does not have DIP packaging, so we cannot test it on breadboard and has to resort to using PCB to test it.

6.2.3 Breading of driver for H-bridge switching



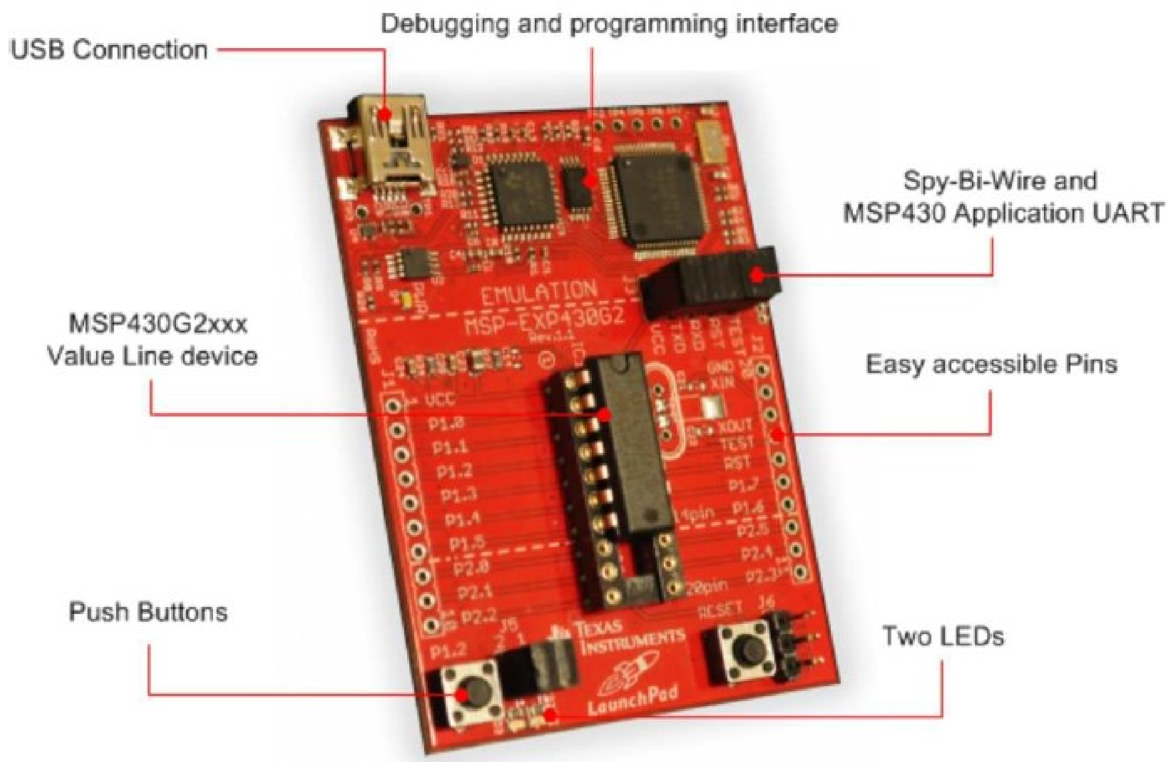
6.2.4 Bi-Polar Waveform

Result after driving the H-bridge with the driver and with an input of 200V from step-up regulator. A period of 330ms with pulse width of 30us pulse is send into 2 different gates. The delay between 2 pulses is approximately 20us.



6.3 Micro-processor

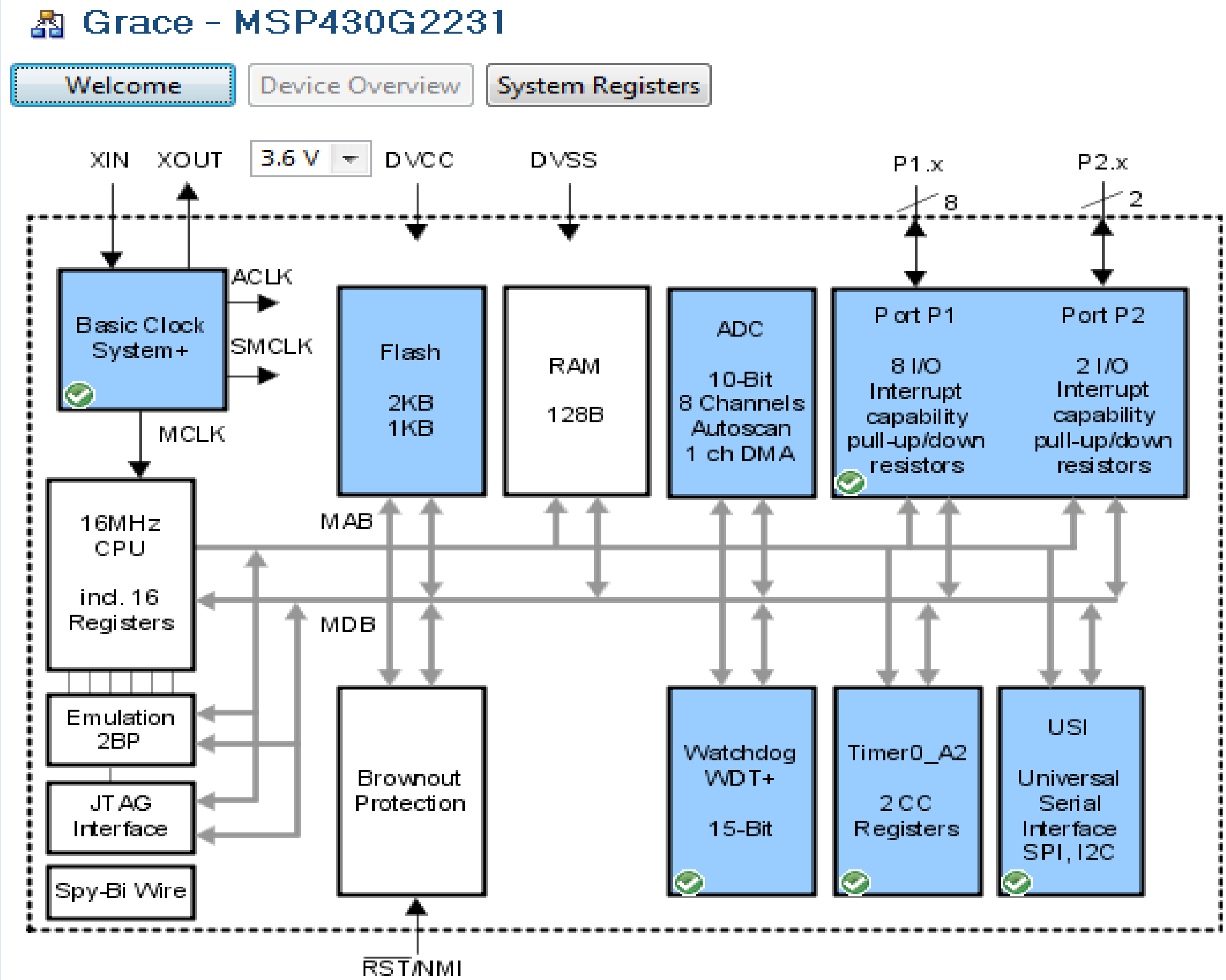
6.4.1 Simple switching circuit diagram

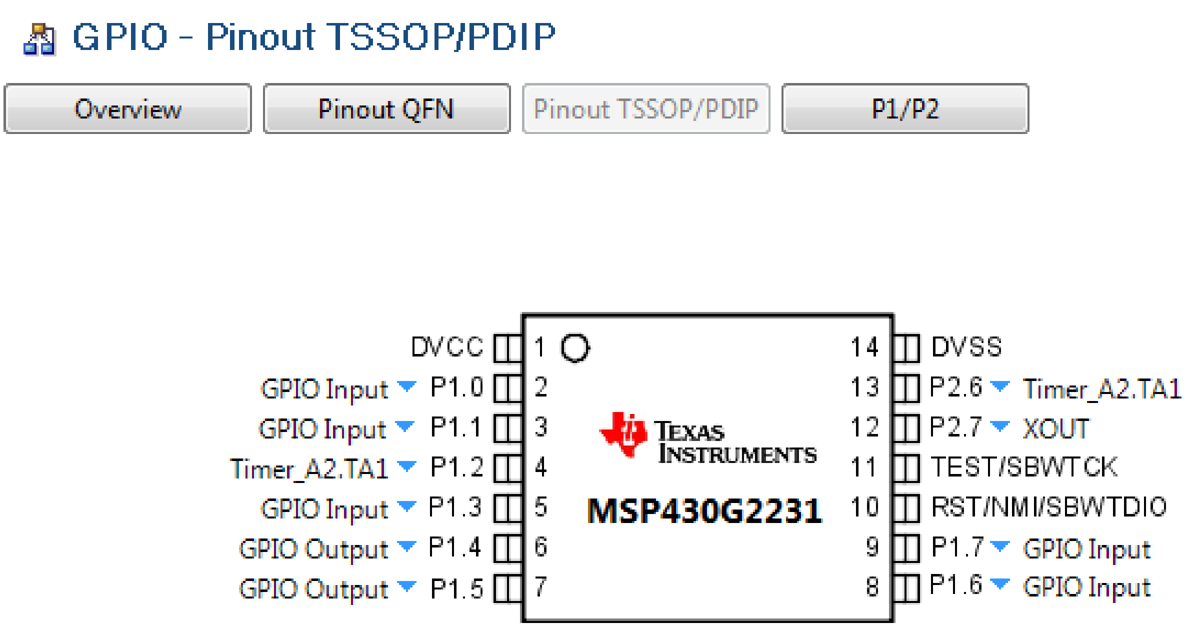


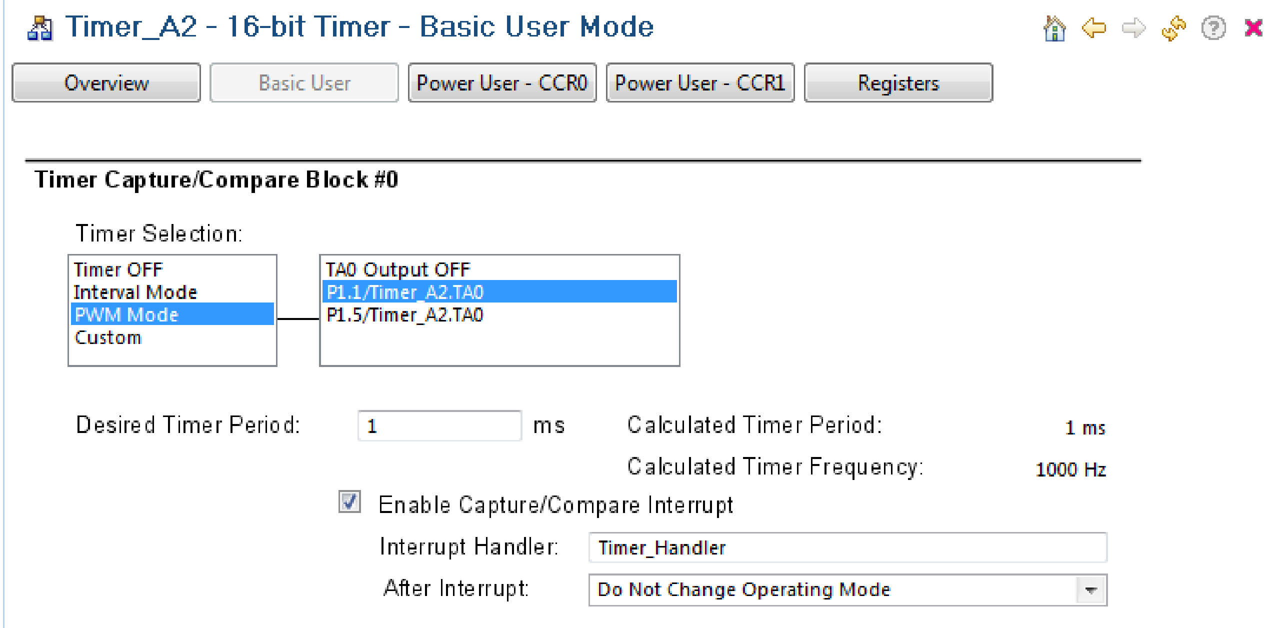
MSP-EXP430G2 LaunchPad Overview

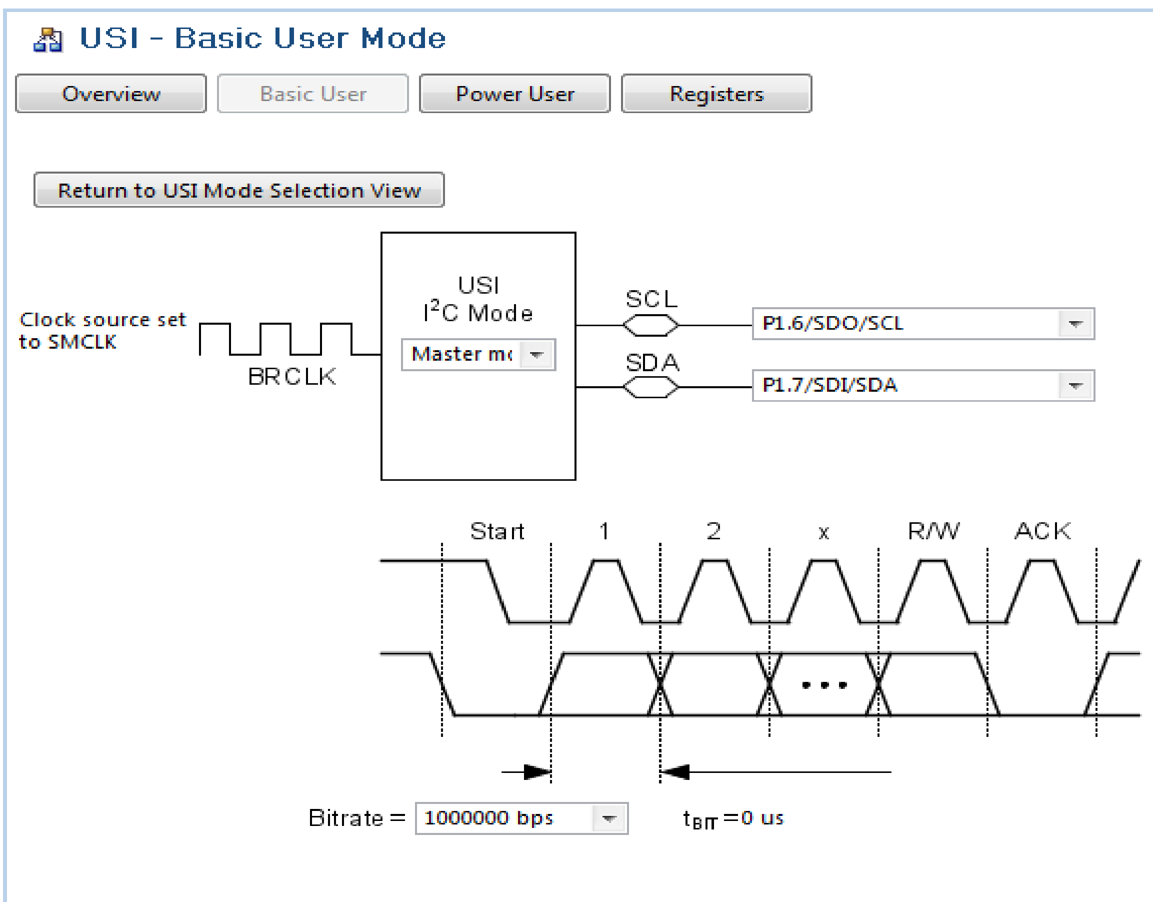
The above MSP430 LaunchPad is very easy to use and it is cheap. They provide free and downloadable software with integrated development environments (IDEs). They also provide sample coding for us to reference.

We used the code and program to generate 2 pulses with duration of 330ms and a width of 30us and a delay of 20us in between each pulse. The pulse is send to the gate of the P-Channel mosfet of the H-Bridge.


Grace - MSP430G2231
Welcome
Device Overview
System Registers
XIN
XOUT
3.6 V
DVCC
DVSS
P1.x
8
P2.x
2
ACLK
SMCLK
MCLK
Basic Clock System+
Flash 2KB 1KB
RAM 128B
ADC 10-Bit 8 Channels Autoscan 1 ch DMA
Port P1 8 I/O Interrupt capability pull-up/down resistors
Port P2 2 I/O Interrupt capability pull-up/down resistors
16MHz CPU incl. 16 Registers
MAB
MDB
Emulation 2BP
JTAG Interface
Spy-Bi Wire
Brownout Protection
Watchdog WDT+ 15-Bit
Timer0_A2 2 CC Registers
USI Universal Serial Interface SPI, I2C
RST/NMI


GPIO - Pinout TSSOP/PDIP
Overview
Pinout QFN
Pinout TSSOP/PDIP
P1/P2
DVCC 1
GPIO Input P1.0 2
GPIO Input P1.1 3
Timer_A2.TA1 P1.2 4
GPIO Input P1.3 5
GPIO Output P1.4 6
GPIO Output P1.5 7
14 DVSS
13 P2.6 Timer_A2.TA1
12 P2.7 XOUT
11 TEST/SBWTCK
10 RST/NMI/SBWTDIO
9 P1.7 GPIO Input
8 P1.6 GPIO Input
TEXAS INSTRUMENTS
MSP430G2231


Timer_A2 - 16-bit Timer - Basic User Mode
Overview
Basic User
Power User - CCR0
Power User - CCR1
Registers
Timer Capture/Compare Block #0
Timer Selection:
Timer OFF
Interval Mode
PWM Mode
Custom
TA0 Output OFF
P1.1/Timer_A2.TA0
P1.5/Timer_A2.TA0
Desired Timer Period:
1
ms
Calculated Timer Period:
1 ms
Calculated Timer Frequency:
1000 Hz
Enable Capture/Compare Interrupt
Interrupt Handler:
Timer_Handler
After Interrupt:
Do Not Change Operating Mode

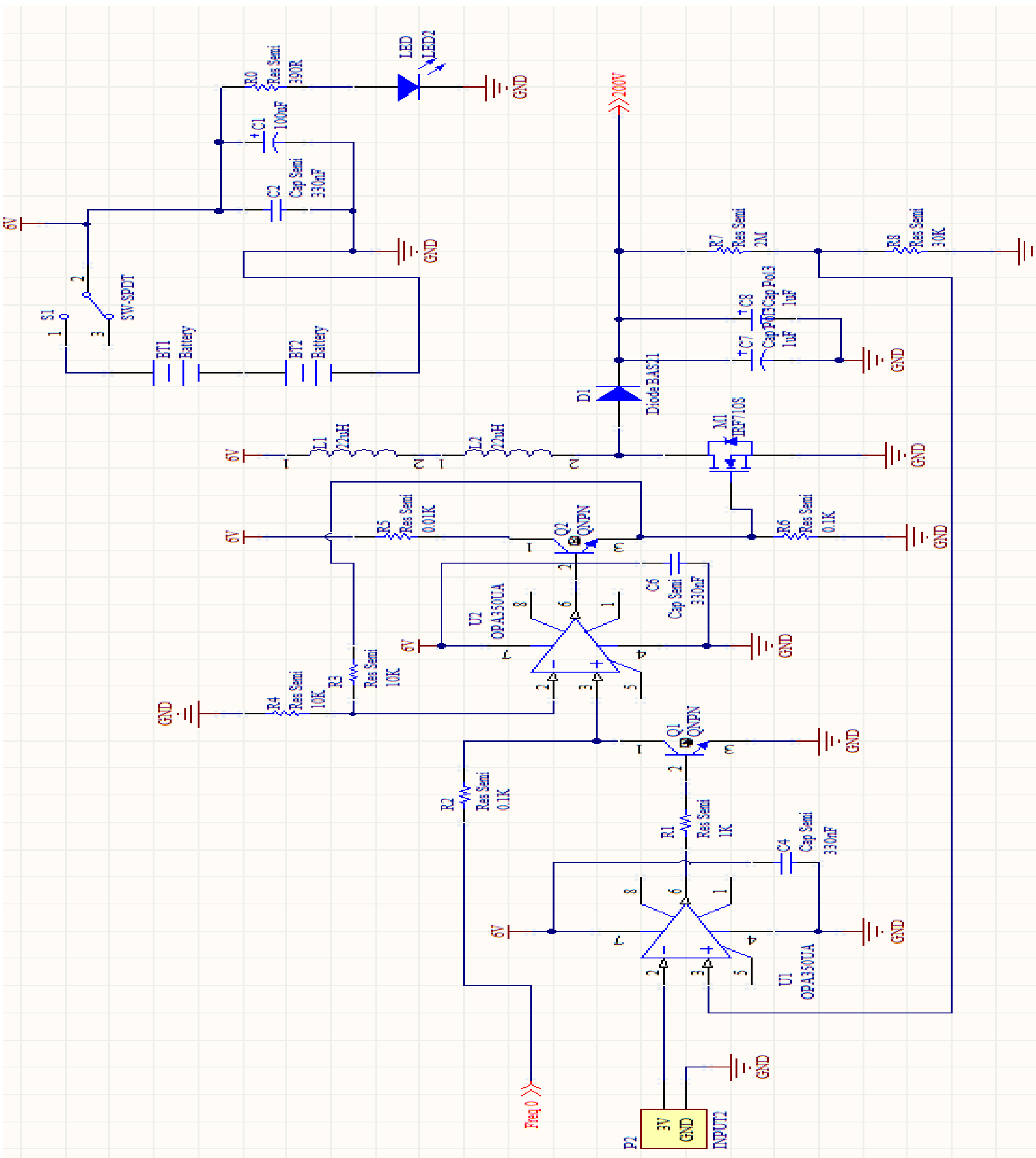

USI - Basic User Mode
Overview
Basic User
Power User
Registers
Return to USI Mode Selection View
Clock source set to SMCLK
BRCLK
USI
I2C Mode
Master mo
SCL
P1.6/SDO/SCL
SDA
P1.7/SDI/SDA
Start
1
2
x
R/W
ACK
Bitrate =
1000000 bps
tBIT =0 us

6.4 Final PCB Design and Schematic Design

6.4.1 Schematic of step-up regulator with Voltage Control

Green LED: To indicate that the circuit is switch on.

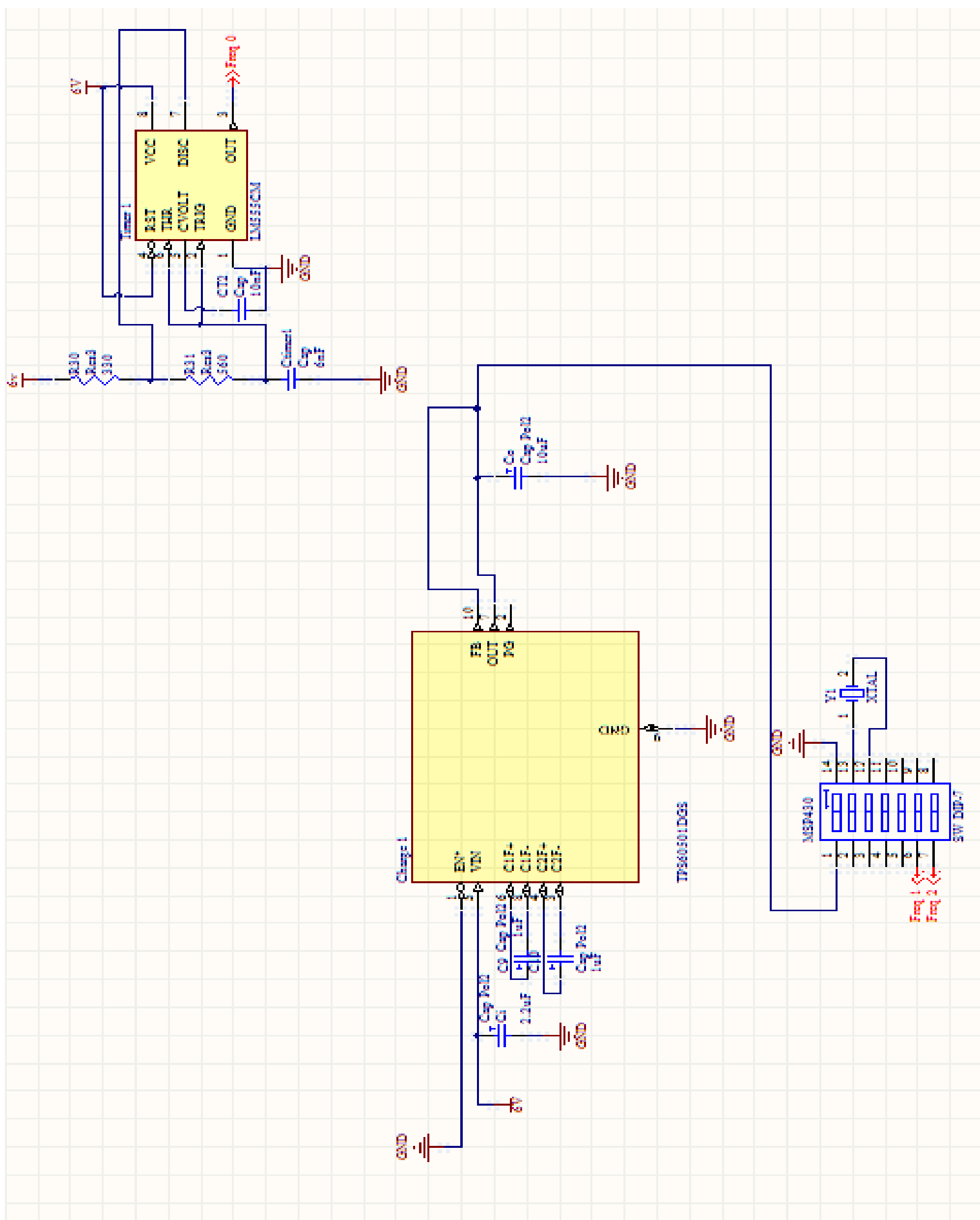
Voltage Control: Using 0-3V to control the output voltage from 0-200V.



6.4.2 Charge Pump with MSP430 with Charge Pump

TPS60501: Step down charge pump, set from 6v input to 3.3V output. It is used to supply the micro-controller MSP430.

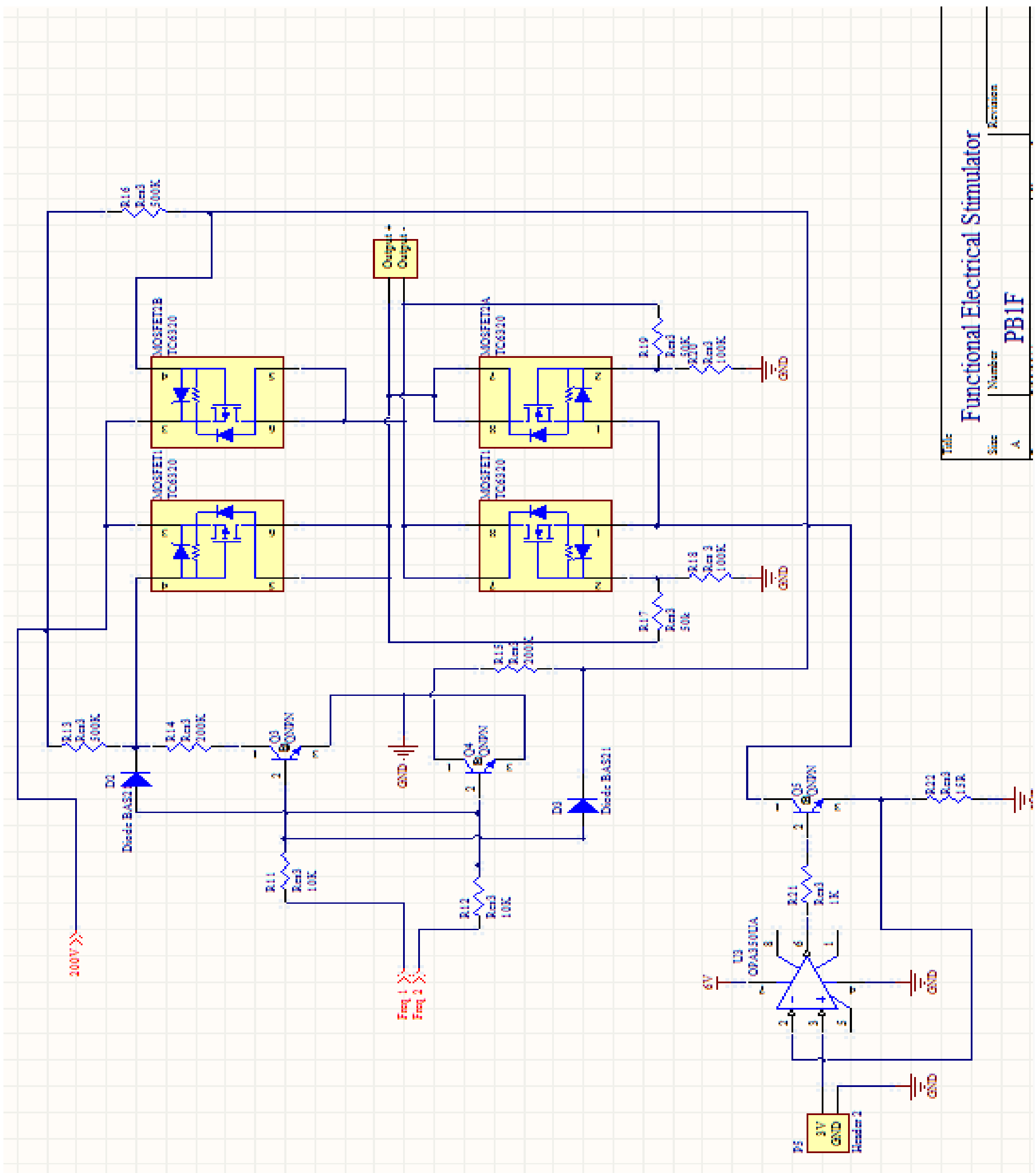
LM555 Timer: To generate an output pulse of 110KHz with a duty cycle of 55%. It is then use to control supply for voltage generator to generate an output of 200V.



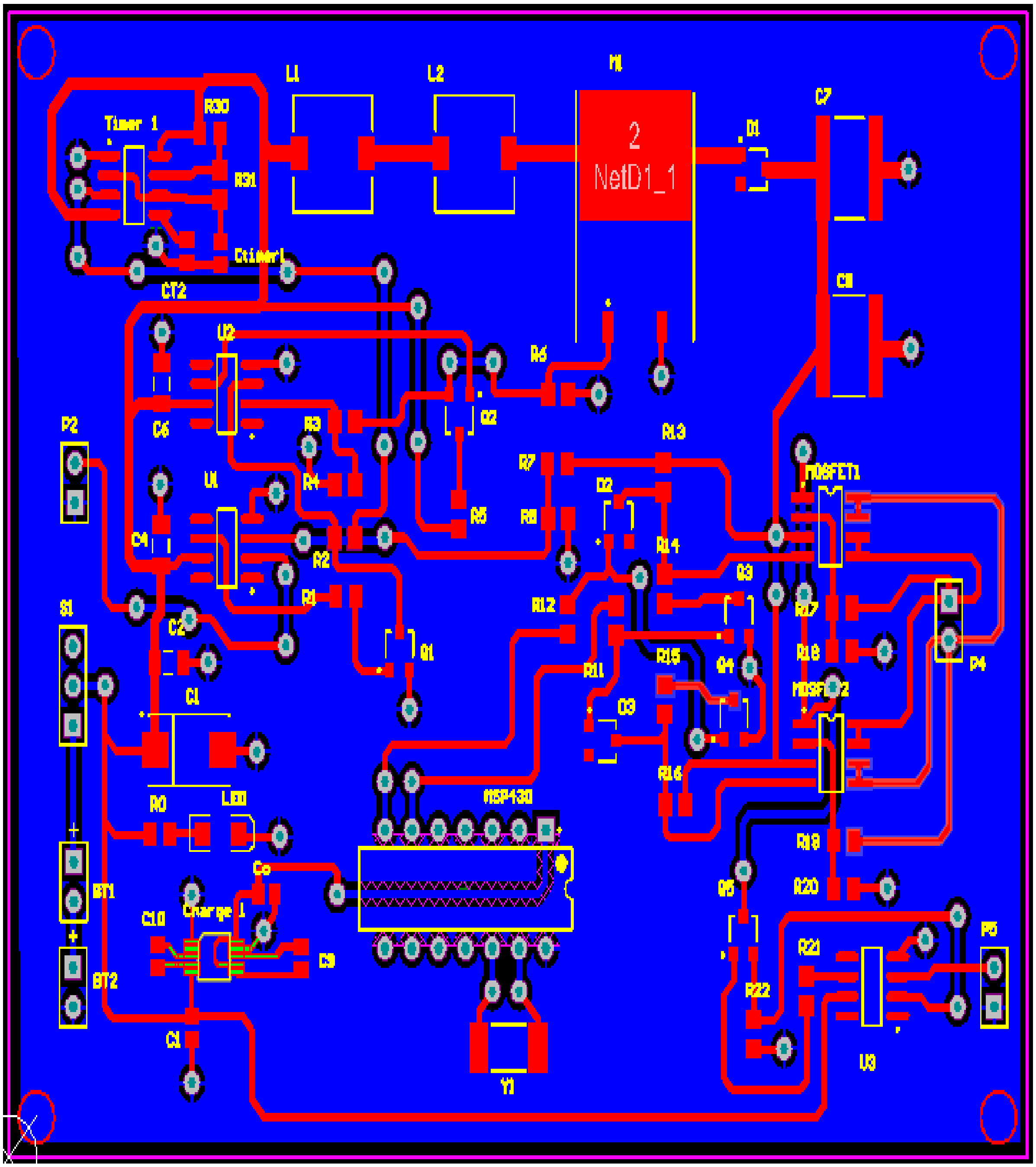
6.4.3 H-Bridge with Current Control

H-Bridge: To generate Bi-Polar Pulses.

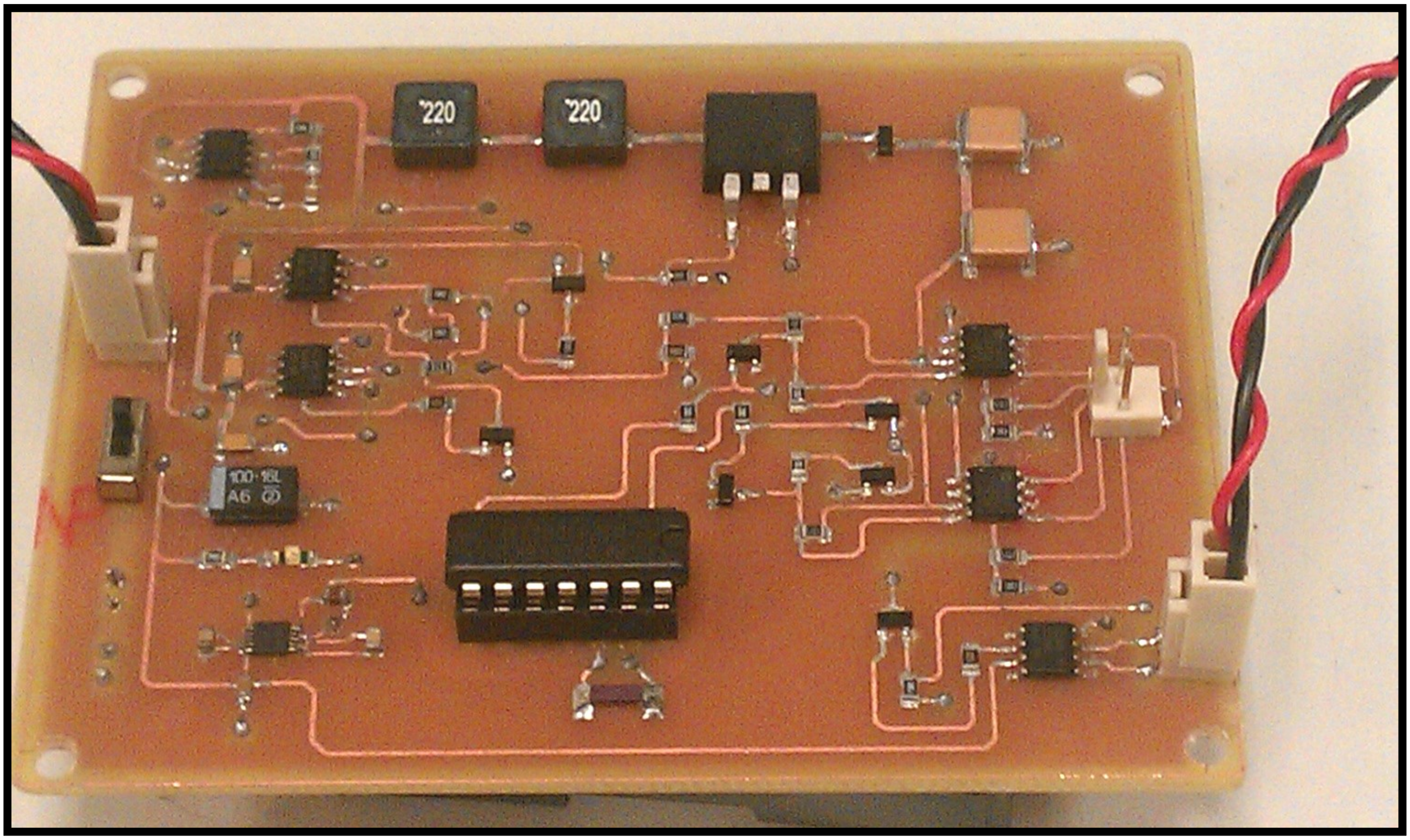
Current Control: To control the amount of current flowing through the body with a voltage reference of 0-3V, so that the output current to the patient can varies from 0-200mA.



6.4.4 Overall PCB Design



Final Product



Problems Faced

One of the problems, we faced was the H-Bridge IC did not work properly. We used many ways to route the PCB board before we can get it working. Some of the H-Bridge IC did not function well as it might be faulty or due to the wrong PCB routing which causes one of the side to be faulty when during test.

Another problem is the H-Bridge Driver that we intend to use to power up the H-Bridge using external 2 N-channel and it did not succeed. So we change to another design where we found online. It is using BJT to operate as an external driver to drive the H-Bridge.

One other problem we face is the step up regulator. When we combine with the H-bridge the capacitor cannot store the charge for a long time and causes the overall circuit not to work. So we decrease our frequency and increase the capacitance value to improve the overall charge and discharge for the H-bridge.

User manual

Features of FES

- Output on electrodes – Stimulation can be used to treat various parts of the body simultaneously or separately.
- 3 main treatment; Assist in standing, walking and preventing muscle atrophy
- The wave form and frequency will be constant
- The electrodes pad are flexible which can be affixed following the contour of the body

Use of FES for

- Paralysis of lower body.
- Stroke
- Recovery of muscle fatigue
- Increase muscular activity
- Improves blood circulation

Important preliminary information

Warning

Refrain from using in the following situations:

1. Heart diseases
2. Users of pacemakers
3. High fever
4. Acute diseases
5. Abnormal blood pressure
6. Menstruation, Pregnancy
7. Skin diseases
8. Physical cut or wounds on the targeted part
9. When targeted part is sweating or wet
10. Any major diseases

Precautionary Instructions

Caution

Read, understand and practice the precautionary and operating instructions. Know the limitations and hazards associated with using any electrical stimulation

Consult your doctor/physician prior using this device or feeling unwell after using.

Correct Usage

Installing the battery

When replacing the batteries, be sure that the power is off.

Flip over to the battery compartment,

Insert the battery and make sure that the (+) and (-) polarities match the diagram inside the compartment.

Confirm that the battery is completely secured.

Preparing the padding

1. Correctly identify the position of the intended muscle group.
2. Use a damp towel to wipe the area of skin on which will be affixing the pads to remove any oil, cosmetic or dirt.
3. Should the padding be soiled, clean it with a damp cloth.
4. Wear the padding at the area.

Cautions

1. Never wear the padding with the power supply on. Doing so may result in you being subjected to a sudden strong shock.

2. Cease usage immediately if feeling unwell when using this device, consult your doctor immediately.

3. Do not assume that by increasing the intensity of the stimulation that there will be a stronger effect. This may well not be the case. However, it is possible that excessive stimulation to the skin will result in irritation or red eruptions.

Maintenance and storage

1. Do not use or store the unit where there are magnetic fields or electric wave to prevent interference (near TV sets or speakers).

2. Do not place the device in areas of high temperature, high humidity, or under direct sunlight.

3. Store the device where there is no moisture as far as possible.

4. Keep out of reach of children age 5 years and below.

6. Remove batteries if unit will not be used for extended periods of time.

Appendix

a. Using Interrupt timer to generate a pulse

```
/*
 * ======== Standard MSP430 includes ========
 */
#include <msp430.h>

/*
 * ======== Grace related includes ========
 */
#include <ti/mcu/msp430/csl/CSL.h>

int i,T1=1,T2=10 ;
void Timer_Handler (void)
{

        P1OUT = BIT4;           // on set output to p1.4
        for (i = 0; i < T1; i++); // pulse width for p1.2
        P1OUT=0x00;
        for (i = 0; i < T2; i++); // pulse width delay
        P1OUT = BIT5; // on set output to p1.5
        for (i = 0; i < T1; i++); // pulse width for p1.6
        P1OUT = 0x00;
}
/*
 *  ======== main ========
 */
int main(int argc, char *argv[])
{
   CSL_init();                  // Activate Grace-generated configuration
   __enable_interrupt();        // Set global interrupt enable

   // >>>>> Fill-in user code here <<<<<

   return (0);
}
```

b. DAC coding for Master to Slave

```
//******************************************************************************
*
//  MSP430G2x21/G2x31 Demo - I2C Master Transmitter / Reciever, Repeated Start
//
//  Description: I2C Master communicates with I2C Slave using
//  the USI. Master data should increment from 0x55 with each transmitted byte
//  and Master determines the number of bytes transmitted and recieved, set by
//  the Number_of_TX_Bytes and Number_of_RX_bytes values. These values will
//  determine how many bytes are Txed then RXed with repeated starts in-between.
//  LED off for address or data Ack; LED on for address or data NAck.
//  ACLK = n/a, MCLK = SMCLK = Calibrated 1MHz
//
//
//  ***THIS IS THE MASTER CODE***
//
//                  Slave                      Master
//     (msp430g2x21_usi_15.c)
//             MSP430G2x21/G2x31          MSP430G2x21/G2x31
//             -----------------          -----------------
//         /|\|              XIN|-    /|\|              XIN|-
//          | |                 |      | |                 |
//          --|RST          XOUT|-     --|RST          XOUT|-
//            |                 |        |                 |
//      LED <-|P1.0             |        |                 |
//            |                 |        |             P1.0|-> LED
//            |         SDA/P1.7|------->|P1.6/SDA         |
//            |         SCL/P1.6|<-------|P1.7/SCL         |
//
//  Note: internal pull-ups are used in this example for SDA & SCL
//
//  D. Dang
//  Texas Instruments Inc.
//  October 2010
//  Built with CCS Version 4.2.0 and IAR Embedded Workbench Version: 5.10
//******************************************************************************
*

#include <msp430g2221.h>


#define number_of_TX_bytes 2                 // How many bytes do you want to TX?
#define number_of_RX_bytes 3                 // How many bytes do you want to RX?

void Master_RPT(void);

void Master_Transmit(void);
void Master_Recieve(void);

void Setup_USI_Master_TX(void);
void Setup_USI_Master_RX(void);

char MST_Data = 0x55;                        // Variable for transmitted data
char SLV_Addr = 0x90;
int I2C_State, Bytecount, Transmit, number_of_bytes, repeated_start = 0;
```

```
void Data_TX (void);
void Data_RX (void);
void main(void)
{
  volatile unsigned int i;                  // Use volatile to prevent removal

  WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog
  if (CALBC1_1MHZ ==0xFF || CALDCO_1MHZ == 0xFF)
  {
    while(1);                               // If calibration constants erased
                                            // do not load, trap CPU!!
  }
  BCSCTL1 = CALBC1_1MHZ;                    // Set DCO
  DCOCTL = CALDCO_1MHZ;

  P1OUT = 0xC0;                             // P1.6 & P1.7 Pullups, others to 0
  P1REN |= 0xC0;                            // P1.6 & P1.7 Pullups
  P1DIR = 0xFF;                             // Unused pins as outputs
  P2OUT = 0;
  P2DIR = 0xFF;



  while(1)
  {

   Bytecount = 0;
   Master_RPT();

 }
}

/******************************************************
// USI interrupt service routine
// Data Transmit : state 0 -> 2 -> 4 -> 10 -> 12 -> 14
// Data Recieve  : state 0 -> 2 -> 4 -> 6 -> 8 -> 14
******************************************************/
#pragma vector = USI_VECTOR
__interrupt void USI_TXRX (void)
{
  switch(__even_in_range(I2C_State,14))
    {
      case 0: // Generate Start Condition & send address to slave
              P1OUT |= 0x01;                // LED on: sequence start
              Bytecount = 0;
              USISRL = 0x00;                // Generate Start Condition...
              USICTL0 |= USIGE+USIOE;
              USICTL0 &= ~USIGE;
              if (Transmit == 1){
                USISRL = 0x90;              // Address is 0x48 << 1 bit + 0 (rw)
              }
               if (Transmit == 0){
               USISRL = 0x91;               // 0x91 Address is 0x48 << 1 bit
                                            // + 1 for Read
               }
              USICNT = (USICNT & 0xE0) + 0x08; // Bit counter = 8, TX Address
              I2C_State = 2;                // next state: rcv address (N)Ack
```

```
                break;

        case 2: // Receive Address Ack/Nack bit
                USICTL0 &= ~USIOE;              // SDA = input
                USICNT |= 0x01;                 // Bit counter=1, receive (N)Ack
bit
                I2C_State = 4;                  // Go to next state: check (N)Ack
                break;

        case 4: // Process Address Ack/Nack & handle data TX

 if(Transmit == 1){
                USICTL0 |= USIOE;               // SDA = output
                if (USISRL & 0x01)              // If Nack received...
                { // Send stop...
                  USISRL = 0x00;
                  USICNT |=  0x01;              // Bit counter=1, SCL high, SDA low
                  I2C_State = 14;               // Go to next state: generate Stop
                  P1OUT |= 0x01;                // Turn on LED: error
                }
                else
                { // Ack received, TX data to slave...
                USISRL = MST_Data++;              // Load data byte
                USICNT |=  0x08;                // Bit counter = 8, start TX
                I2C_State = 10;                 // next state: receive data (N)Ack
                Bytecount++;
                P1OUT &= ~0x01;                 // Turn off LED
                break;
                }
 } if(Transmit == 0){

                 if (USISRL & 0x01)              // If Nack received
                { // Prep Stop Condition
                  USICTL0 |= USIOE;
                  USISRL = 0x00;
                  USICNT |=  0x01;              // Bit counter= 1, SCL high, SDA
low
                  I2C_State = 8;                // Go to next state: generate Stop
                  P1OUT |= 0x01;                // Turn on LED: error
                }
                else{ Data_RX();}               // Ack received


}
                break;

case 6: // Send Data Ack/Nack bit
                USICTL0 |= USIOE;               // SDA = output
                if (Bytecount <= number_of_bytes-2)
                {                               // If this is not the last byte
                  USISRL = 0x00;                // Send Ack
                  P1OUT &= ~0x01;               // LED off
                  I2C_State = 4;                // Go to next state: data/rcv again
                  Bytecount++;
                  }

                else //last byte: send NACK
```

```
          {
            USISRL = 0xFF;                // Send NAck
            P1OUT |= 0x01;                // LED on: end of comm
            I2C_State = 8;                // stop condition
          }
          USICNT |= 0x01;                 // Bit counter = 1, send (N)Ack bit
          break;

      case 8: // Prep Stop Condition
          USICTL0 |= USIOE;               // SDA = output
          USISRL = 0x00;
          USICNT |=  0x01;                // Bit counter= 1, SCL high, SDA
low
          I2C_State = 14;                 // Go to next state: generate Stop
          break;

      case 10: // Receive Data Ack/Nack bit
          USICTL0 &= ~USIOE;              // SDA = input
          USICNT |= 0x01;                 // Bit counter = 1, receive (N)Ack
bit
          I2C_State = 12;                 // Go to next state: check (N)Ack
          break;

      case 12: // Process Data Ack/Nack & send Stop

          USICTL0 |= USIOE;
          if (Bytecount == number_of_bytes){// If last byte

            if(repeated_start == 1){

            USISRL = 0xFF;                  // this will prevent a stop cond
            USICTL0 |= USIOE;               // SDA = output
            I2C_State = 14;                 // Go to next state: generate
Stop
            USICNT |=  0x01;     }          // set count=1 to trigger next
state

          else{
          USISRL = 0x00;

          I2C_State = 14;                 // Go to next state: generate Stop
          P1OUT |= 0x01;
          USICNT |=  0x01;     }          // set count=1 to trigger next
state
          }else{
            P1OUT &= ~0x01;               // Turn off LED
            Data_TX();                    // TX byte
          }
          break;

      case 14:// Generate Stop Condition
          USISRL = 0x0FF;                 // USISRL = 1 to release SDA
          USICTL0 |= USIGE;               // Transparent latch enabled
          USICTL0 &= ~(USIGE+USIOE);      // Latch/SDA output disabled
          I2C_State = 0;                  // Reset state machine for next xmt
          LPM0_EXIT;                      // Exit active for next transfer
          break;
```

```
    }

  USICTL1 &= ~USIIFG;                        // Clear pending flag
}


void Data_TX (void){

              USISRL = MST_Data++;           // Load data byte
              USICNT |=  0x08;               // Bit counter = 8, start TX
              I2C_State = 10;                // next state: receive data (N)Ack
              Bytecount++;
}

void Data_RX (void){
      USICTL0 &= ~USIOE;                   // SDA = input --> redundant
        USICNT |=  0x08;                     // Bit counter = 8, RX data
        I2C_State = 6;                       // Next state: Test data and (N)Ack
        P1OUT &= ~0x01;                      // LED off
        }


void Setup_USI_Master_TX (void)
{
  _DINT();
  Transmit = 1;
  USICTL0 = USIPE6+USIPE7+USIMST+USISWRST;  // Port & USI mode setup
  USICTL1 = USII2C+USIIE;                    // Enable I2C mode & USI interrupt
  USICKCTL = USIDIV_7+USISSEL_2+USICKPL;     // USI clk: SCL = SMCLK/128
  USICNT |= USIIFGCC;                        // Disable automatic clear control
  USICTL0 &= ~USISWRST;                      // Enable USI
  USICTL1 &= ~USIIFG;                        // Clear pending flag
  _EINT();
}


void Setup_USI_Master_RX (void)
{
  _DINT();
  Transmit = 0;
  USICTL0 = USIPE6+USIPE7+USIMST+USISWRST;  // Port & USI mode setup
  USICTL1 = USII2C+USIIE;                    // Enable I2C mode & USI interrupt
  USICKCTL = USIDIV_7+USISSEL_2+USICKPL;     // USI clks: SCL = SMCLK/128
  USICNT |= USIIFGCC;                        // Disable automatic clear control
  USICTL0 &= ~USISWRST;                      // Enable USI
  USICTL1 &= ~USIIFG;                        // Clear pending flag
  _EINT();

}

void Master_Transmit(void){
  number_of_bytes = number_of_TX_bytes;
Setup_USI_Master_TX();
    USICTL1 |= USIIFG;                       // Set flag and start communication
    LPM0;                                    // CPU off, await USI interrupt
}
void Master_Recieve(void){
```

```
  number_of_bytes = number_of_RX_bytes;
  Setup_USI_Master_RX();
  USICTL1 |= USIIFG;                        // Set flag and start communication
  LPM0;                                     // CPU off, await USI interrupt
}

void Master_RPT(void){
    repeated_start =1;

    Master_Transmit();
    _NOP();

    Master_Recieve();
    _NOP();

    repeated_start =0;

}
```