Remote Laser Controller

Based on Raspberr pi 2

Ang Li
Department of Electrical and Computer Engineer
Colorado State University
Fort Collins,USA
angli@rams.colostate.edu

***Abstract*—This project is intended to design a user friendly laser controller which can be remotely controlled. The controller is working as a web server and can be visited by any compatible browser such like google chrome, internet explorer or any other html5 browser. In order to realize this, one raspberry pi 2 was added to the originally motherboard and establish communication with the core FPGA by UART port. Some necessary changed also has been made in FPGA design file to enable communication between pi and the mother board. The web server is running on Pi 2 and can be visited by devices which connected to Wi-Fi access port provided by Pi 2.**

Keywords—Remoting Control, Web Serve, Real-time System

I. INTRODUCTION

Capillary discharge lasers have proven to be the highest average power tabletop coherent EUV sources currently available. Researchers at the Extreme Ultraviolet Engineering Research Center (EUV-ERC) have been performing experiments with these compact capillary discharge systems since 1994. Typical applications include but are not limited to, Nano-pattering, Nano-imaging, nanomachining, holography and spectroscopy. The size of these tabletop lasers has drastically reduced over the past two decades. XUV Lasers, Inc. has developed a very compact and fully enclosed soft x-ray source. The 46.9 nm, capillary discharge-pumped laser not only produces a high energy per pulse, 10 μJ, over a 1.5 ns pulse duration but can fit onto a regular sized desktop. The fully housed and fixed components of this system in combination with a comprehensive human interface allows for turnkey style operation during experiments.

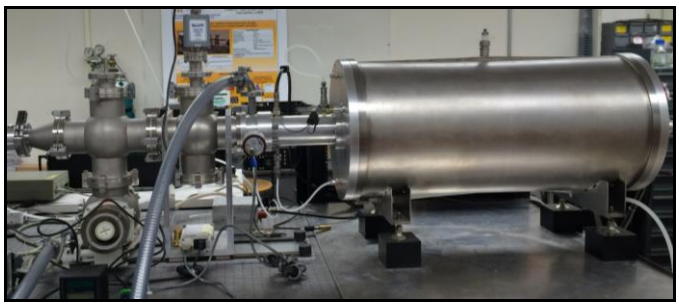


Figure 1: 46.9 nm Capillary Discharge Soft X-ray Laser

Comprised of eight closely stacked subsystems, an external electronics rack powers and controls the laser head. At the top of the rack, the Controller unit provides both local and remote (can only be controlled by local computer via USB to UART convertor) control for setup procedures and laser operation parameters. Below, the Heater/Oil Pump unit both pre-heats a thyratron for 10 mins and powers an oil pump fixed to the main laser head chamber. Following are the DC Preionization and Lesser High Voltage, LHV, units which output approximately 8kV to ionize the argon gas in the capillary. The Thyratron Trigger unit sits beneath the LHV and produces a 3kV output voltage to trigger current flow through the thyratron. Below the Thyratron Trigger, The High Voltage Supply Control unit regulates the amount of voltage to be output by the TDK-Lambda High Voltage Supply that it rests upon. The final unit is the RF Preionization unit which provides 30W CW and 300WPK power at 60 MHz to a coil surrounding the capillary for pre-ionization of the argon gas.

The current laser system had been tested for several thousand shots and operated correctly but not to all desired specifications. The final product must be capable of a 10 Hz firing rate which had been attempted but subsequently damaged components in the RF Periodization unit. To prevent this failure from repeating, both the RF unit and its matching network have been re-designed. The Controller prototype will also be re-designed to handle more interlock signals received from the other seven units in the electronics rack. Finally, XUV Lasers requires thorough documentation for laser head assembly and testing for company records and customer reference.

In order to provide a better user experience and safety operation environment, an additional control interface is necessary to develop in the future to enable cross-platform remoting control.

II. MOTIVATION

A. *Previous Work*

A DE-0 Nano FPGA development was chosen as an FPGA system board for this project. The resources on this development board that are relevant to this project are listed below.

Item	Specification
FPGA	Altera Cyclone IVEP4CE22F17C6N
Configuration Flash	EPCS64 (64-Mbit)
SDRAM	32MB
EEPROM	2Kb
GPIO	88 Ports

A mother board for the controller was designed by Mark Woolston and laid out by David (Aaron) Yazdani. The timing engine memory structure and the instruction set were designed by Mark Woolston and Valentin Aslanyan. The Python program for remote control mode was designed by Valentin Aslanyan. A power monitor circuit, which is able to maintain the working system when a power failure occur, was designed by Tyson Urrutia. The NIOS II soft core and the driver programs for the VFD display and the keypad were designed by Ce Guo.

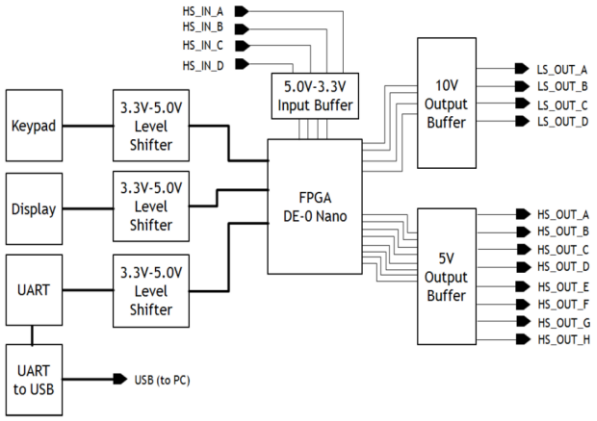


Figure 2: Previous System Controller Hardware Diagram

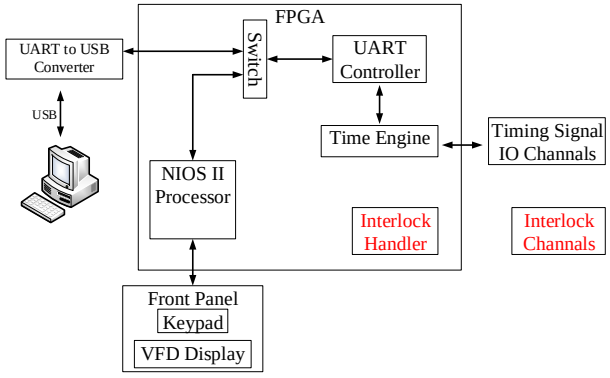


Figure 3: Previous On-Chip Communication Diagram

Before this project was taken over, both the software and hardware part of the new interlock handler hadn't been completely embedded into the existing system yet.

B. *Current System Level Connection*

It's easy to observe that additional local PC can be connected with the FPGA via UART port, however the local PC doesn't have any interface to control the laser. The previous set up only can set up the parameters of the laser by terminal. And it's really inconvenient to change model from remote mode and local control model since it's needs burn in firmware to FPGA.

In order to address above problems, attaching a raspberry pi 2 to the mother board and made it connected with the FPGA. Since the Raspberry Pi 2 has its own UART port which made the whole connection easier.

It's not hard to make connection between Pi 2 and current FPGA, the UART port can be connected by GPIO 14 and GPIO 15 on Pi 2.

III. HARDWARE CONNECTION

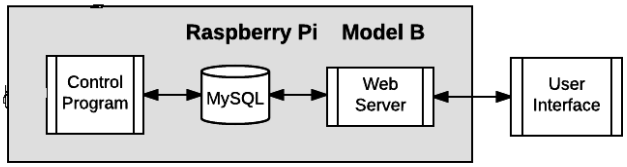


Figure 4 Overall System Design

The overall system diagram can be seen in Figure 4. The raspberry Pi 2 Model B is the central unit in the control system. Other units are the FPGA core, Nois II soft core and previous local control program. On startup, the Pi initializes an Apache web server, a PHP interpreter, and a MySQL database. The web server acts as a go-between for the database and the user interface. The database is the central resource, storing the interrupt information and control settings. The control program reports information from the FPGA core to the database, and pulls control parameters for laser to control systems. The control system will refresh the database as frequent as once per second.

A. *Establish connection between Pi and Time Engine*

The connection between Raspberry Pi 2 and controller were established via UART port. On Raspberry Pi 2, the UART port can be opened by

port = serial.Serial("/dev/ttyAMA0", baudrate=9600)

The timing engine in the controller has an instruction set which were implemented by circuits:

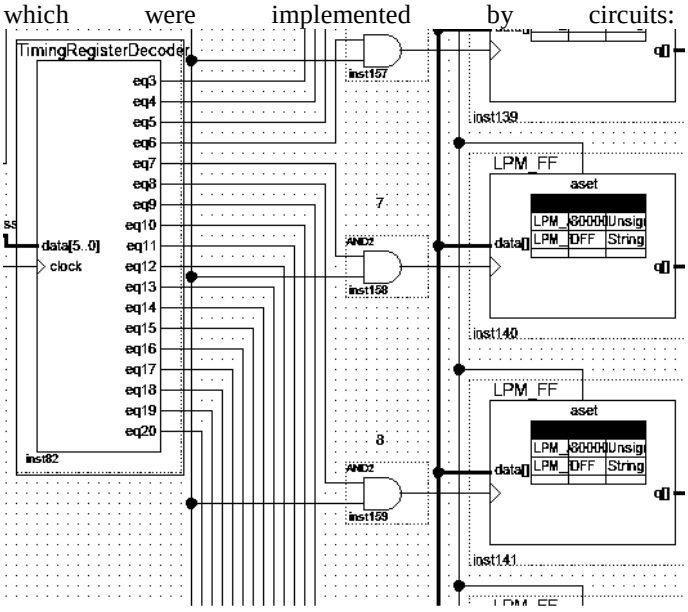


Figure 5: Timing Engine Instruction Decoding

The memory, which has a word length of 32 bits, is embedded inside the timing engine. It stores all the required parameters (e.g. repetition rate, number of shots, timing parameters). As the Fig 6 stated, the instruction decoder use a mux to decode different instructions in the instructions set, and them fire the laser. The timing engine would be able to get parameters in its RAM memory and then generate delay for each channel. Parameters are mapped to this memory as list below.

Address	Parameter
0x00	Reserved
0x01	Repetition Rate
0x02	Number of Shots
0x03	Chnl. 1 Start Time
0x04	Chnl. 1 End Time
0x05	Chnl. 2 Start Time
0x06	Chnl. 2 End Time
…	…
0x13	Chnl. 9 Start Time
0x14	Chnl. 9 End Time

Table 1 Timing Engine Instructions

Therefore, the timing engine can be programmed by changing the content of this memory. An instruction set is designed such that a computer can access timing engine parameters, as well as trigger the firing sequence, by sending the corresponding instruction through the UART connection.

There are four functional instructions in the instruction set. Each construction consists of 11 ASCII characters. Here is the format of an instruction.

Byte	Content	Description
0	0x02	Start of an instruction
1	W/R/F/S	Instruction type
2	0x00~0x14	Address (Don't care for
3	*	Reserved for checksum
4	Data[3]	Data (Don't care for R/F/S)
5	Data[2]	
6	Data[1]	
7	Data[0]	
8	*	Reserved for checksum
9	0x04	End of an instruction
10	0x04	

Table 2 Instruction Format Details

The python program has been written to write instruction to timing engine memory:

Function WriteRAM and ReadRAM is used to send message via UART port to time engine, it has format like:

```
ser.write(chr(2)+'R'+chr(address*4)+'xxxxxx'+chr(4)+chr(4))
```

ASCII code **chr(2)** means start of the instruction and **R** means read, **address** is the parameter passed in which indicates which channel to read. Ex, channel 1 means Repetition rate of the laser.

The write command uses the similar format:

```
char1=int('{0:032b}'.format(message)[0:8],2)
char2=int('{0:032b}'.format(message)[8:16],2)
char3=int('{0:032b}'.format(message)[16:24],2)
char4=int('{0:032b}'.format(message)[24:32],2)
ser=serial.Serial(port_name)
ser.write(chr(2)+'W'+chr(address*4)+'x'+chr(char1)+chr(char2)+chr(char3)+chr(char4)+'x'
```

The message is the parameter passed in which indicates what value want to set in that channel. All other instruction format program has been attached in Appendix A.

B. Building Pi as a Wifi Access Port

On startup, the Pi establishes an access point on the integrated wlan0 interface using the hostapd service. It uses the dnsmasq service to provide DHCP and DNS services. When the pi has finished its startup routine, a user can connect to the broadcasted ssid. Directing their browser to http://lasercontrol.local will take the user to the user interface page for the laser controller. I may also at some point develop a "shell" app that handles login and connection automatically, but that is a tertiary goal.

The Wi-Fi access port has been developed by some other tutorial, which has been attached in Appendix B.

IV. SOFTWARE DESIGN

A. *Web Server*

Initially the Raspberry Pi 3 was configured to use Flask, a python framework for a simple web server. This proved to be more complex and less robust than simply installing a traditional LAMP software bundle consisting of a Linux operating system, an Apache web server, a MySQL server, and the PHP programming language used by the web server. When an event occurs on the user interface, the client sends a message to the server through a JQuery request to a corresponding php file which responds by querying the database, either to update control settings, or to respond with log information or a file handle for an experiment's results.

B. *Database System*

There are three tables in the database. It is largely complete though some modification may be necessary as system requirements evolve. Descriptions of the database layout and the SQL script used to generate the database can be found in Appendix C.

CONTROL:

id	name	target
1	frequency	5
2	lhv	37500
3	rf	0
4	hv	0
5	dc_pre	0
6	main	0

The control table is used to store target parameter for all parameters and as well as the time to start fire. The basically working procedure is:

- User interface capture user's input
- Web server write information to database
- Control program check database and send instruction to timing engine
- Control program get back from time engine
- Control program update database table 'Current'
- User interface update information according to database

The current configure table receives log updates from the control program. Currently it is used only to log the current parameters conditions for reporting to the user interface; however, it will later be used in a calibration phase by the control program as well. Finally, the trial table contains fields for storing information about an individual trial: start and stop time, the name of the experiment the trial belongs to, and the location of interrupt logging information etc…

C. *User Interface*

The interface is built around the concept of platform independence: the interface needs to work on small screen devices such as smartphones as well as tablets as well as laptop and desktop PCs. To that end I used Bootstrap for the broad design since it is centered around dynamically adjusting to screen resolution. The interface consists of a set of tiles which collapse and expand when the user clicks on a title bar. They are rearranged to match the size of the window. The whole UI on google chrome browser attached on Appendix D and user interface on small screen smart phone attached:

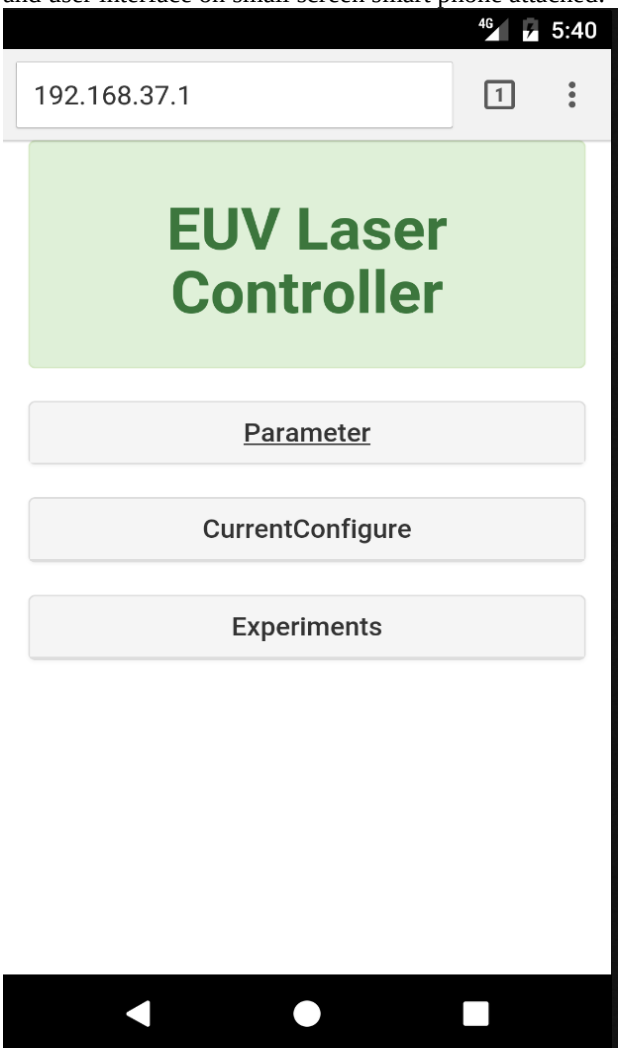


Figure 6: GUI on Android Mobile Phone

There were totally 3 tabs on the user interface web, which are: Parameters, Current Configure and Experiment. The first two tabs are used to set up laser firing parameter and experiment tab is used to fire and pause the laser controller. Parameter tab allows the user to send control information to the database and retrieve the current control configuration state. In addition to explicit readings reported numerically, a current configure tab is also used to tell user current configure of the laser controller.

The whole user interface contains three part: PHP to write/read database, html5/CSS to build the page itself and JavaScript to get user's event on the interface. The final interface is an platform independent web which can be visited by any html5 browser such like Google Chrome, Internet Explorer and Firefox etc. All the source code was created by WordPress and has been attached in the submitted folder.

V. Result and Analysis

After done with the user interface, the wholes controller can be used remotely via any html 5 compatible browser. But there also exist some issues about the controller such like remaining fire number and interrupt handling. Over all, this system works good on original design goals and are able to provide a remotely control on current laser system but definitely it needs more work to make it a reliable controller.

VI. Conclusion and Future Work

The project still needs to be improved in following aspects:

- User interface should provide parameter range check
- Webpage should refresh itself after user done with clicking
- Interrupt information should be provided as log for each experiment

And this system is not able to handle interrupt since it's a non-real time system.

References

[1] P. Martin, "Using your new Raspberry Pi 3 as a WiFi access point with hostapd," *Frillip's Blog*, 04-Mar-2016. [Online]. Available: https://frillip.com/using-your-raspberry-pi-3-as-a-wifi-access-point-with-hostapd/. [Accessed: 06-Dec-2016].

[2] R. Brown, "HOWTO: Setup dnsmasq as DNS DHCP," *Beware Here Be Musings*, 29-Dec-2013. [Online]. Available: http://blogging.dragon.org.uk/howto-setup-dnsmasq-as-dns-dhcp/. [Accessed: 07-Dec-2016].

[3] S. Kelley, "DNSMASQ," *Main page of DNSMASQ*, 18-May-2016. [Online]. Available: http://www.thekelleys.org.uk/dnsmasq/docs/dnsmasq-man.html. [Accessed: 07-Dec-2016].

[4] J. Malinen, "hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator," *hostapd*, 12-Jan-2013. [Online]. Available: https://w1.fi/hostapd/. [Accessed: 07-Dec-2016].

```
import numpy
import serial
import binascii
import time

def set_RepRate(port_name,RepRate):
        WriteRAM32Bits(port_name,int(1),RepRate)
        receivedRepRate=int(ReadRAM32Bits(port_name,int(1)))
        if receivedRepRate!=RepRate:
                print 'Error!'
        return receivedRepRate

def set_NumberOfShots(port_name,NumberOfShots):
        WriteRAM32Bits(port_name,int(2),NumberOfShots)
        receivedNumberOfShots=int(ReadRAM32Bits(port_name,int(2)))
        if receivedNumberOfShots!=NumberOfShots:
                print 'Error!'
        return receivedNumberOfShots

def set_ShotCounter(port_name):
        WriteRAM32Bits(port_name,int(15),int(0))
        WriteRAM32Bits(port_name,int(16),int(3990000))
        return


def ReadRAM32Bits(port_name,address):
    ser=serial.Serial(port_name,timeout=3.0)
    ser.write(chr(2)+'R'+chr(address*4)+'xxxxxx'+chr(4)+chr(4))
    data=ser.read(7)
    ser.close()
    time.sleep(0.1)
    return ord(data[2])+2**8*ord(data[3])+2**16*ord(data[4])+2**24*ord(data[5])


def WriteRAM32Bits(port_name,address,message):
    char1=int('{0:032b}'.format(message)[0:8],2)
    char2=int('{0:032b}'.format(message)[8:16],2)
    char3=int('{0:032b}'.format(message)[16:24],2)
    char4=int('{0:032b}'.format(message)[24:32],2)
    ser=serial.Serial(port_name)
    ser.write(chr(2)+'W'+chr(address*4)+'x'+chr(char1)+chr(char2)+chr(char3)+chr(char4)+'x'+chr(4)+chr(4))
    ser.close()
    time.sleep(0.1)
    return

def Fire(port_name):
    ser=serial.Serial(port_name)
    ser.write(chr(2)+'F'+'xxxxxxx'+chr(4)+chr(4))
    ser.close()
    time.sleep(0.1)
    return

def Stop(port_name):
    ser=serial.Serial(port_name)
    ser.write(chr(2)+'S'+'xxxxxxx'+chr(4)+chr(4))
    ser.close()
    time.sleep(0.1)
    return
```

Appendix A Python Implementation for Time Engine Instruction

This function has already been finished by following steps:

$ sudo apt-get update

$ sudo apt-get install dnsmasq hostapd

Modify /etc/dhcpcd.conf by adding the following line to the bottom of the file:

denyinterfaces wlan0

Change the wlan0 interface in /etc/network/interfaces to:

allow-hotplug wlan0
iface wlan0 inet static
address 172.24.1.1
netmask 255.255.255.0
network 172.24.1.0
broadcast 172.24.1.255

Create a new file: /etc/hostapd/hostapd.conf and add the following:

interface=wlan0
driver=nl80211
ssid=lasercontrol
hw_mode=g
channel=6
ieee80211n=1
wmm_enabled=1
ht_capab=[HT40][SHORT-GI-20][DSSS_CCK-40]
macaddr_acl=0
ignore_broadcast_ssid=0
#rsn_pairwise=CCMP
#auth_algs=1

Open /etc/default/hostapd and change

#DAEMON_CONF=""
to DAEMON_CONF="/etc/hostapd/hostapd.conf"
: sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig
edit /etc/dnsmasq.conf as follows:
interface=wlan0
listen-address=172.24.1.1
bind-interfaces
bogus-priv
dhcp-range=172.24.1.50, 172.24.1.150, 12h
no-resolv
local=/LaserControll.com/
no-hosts #see above
addn-hosts=/etc/dnsmasq_static_hosts.conf
expand_hosts
domain=example.com
dhcp-option=option:router,172.24.1.1

Create and edit /etc/dnsmasq_static_hosts.conf as:

172.24.1.1 laser

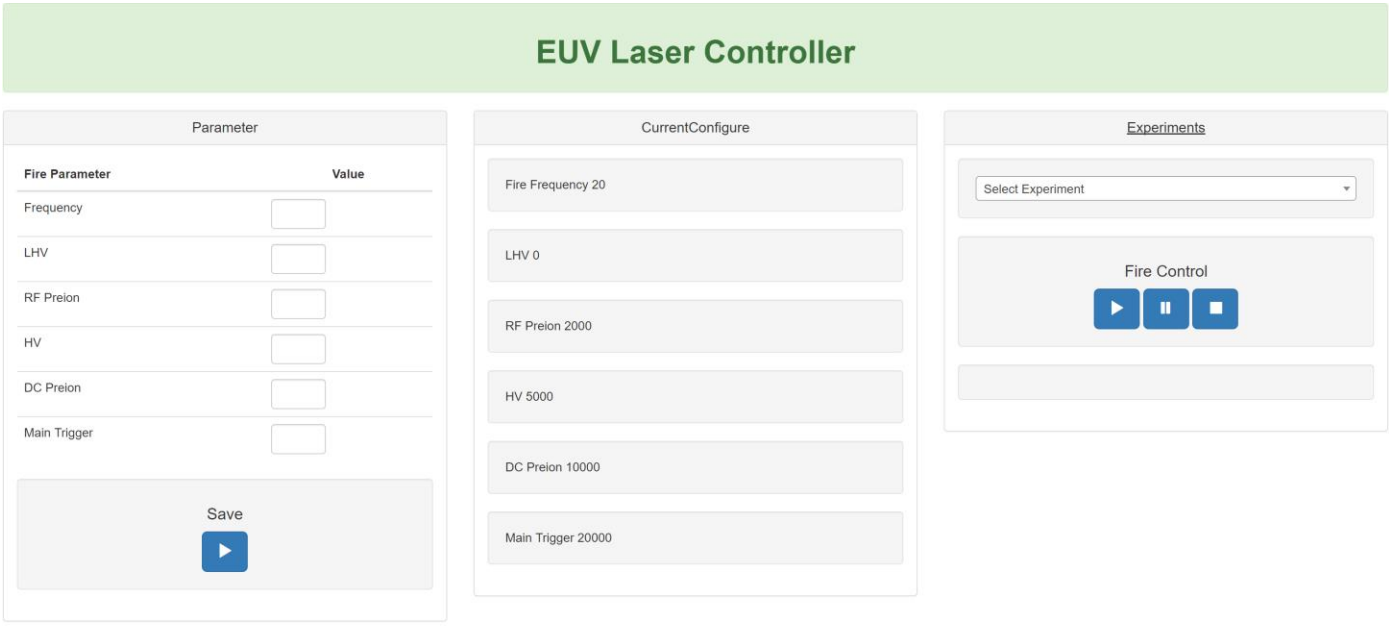
```
CREATE DATABASE laser;
USE laser;
CREATE TABLE current(
        id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
        time TIMESTAMP,
        frequency FLOAT,
        lhv FLOAT,
        rf FLOAT,
        hv FLOAT,
        dc_pre FLOAT,
        main FLOAT
);

CREATE TABLE control(
        id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
        name VARCHAR(255),
        target FLOAT NOT NULL DEFAULT 0,
);

INSERT INTO control (name) VALUES ('O2');
INSERT INTO control (name) VALUES ('temp');
INSERT INTO control (name) VALUES ('stagex');
INSERT INTO control (name) VALUES ('stagey');
INSERT INTO control (name) VALUES ('stagez');
INSERT INTO control (name) VALUES ('focus');
INSERT INTO control (name) VALUES ('log_interval');

CREATE TABLE trials(
        trial INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
        experiment VARCHAR(255),
        time_start DATETIME,
        time_stop DATETIME,
        data_dir VARCHAR(255)
);
```

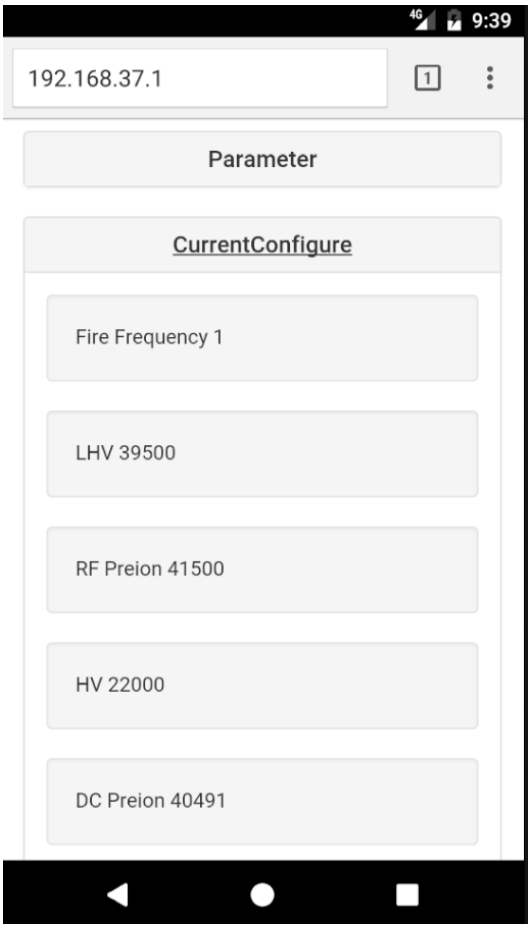
Appendix C.1: Database generation SQL script

```
+----+--------------+--------+--------+
| id | name         | target |
+----+--------------+--------+--------+
|  1 | frequency          |      5 |
|  2 | lhv          |   0 |
|  3 | rf       |      0 |
|  4 | hv       |      0 |
|  5 | dc_pre     |      0 |
|  6 | main  |      0 |
+----+--------------+--------+--------+
```

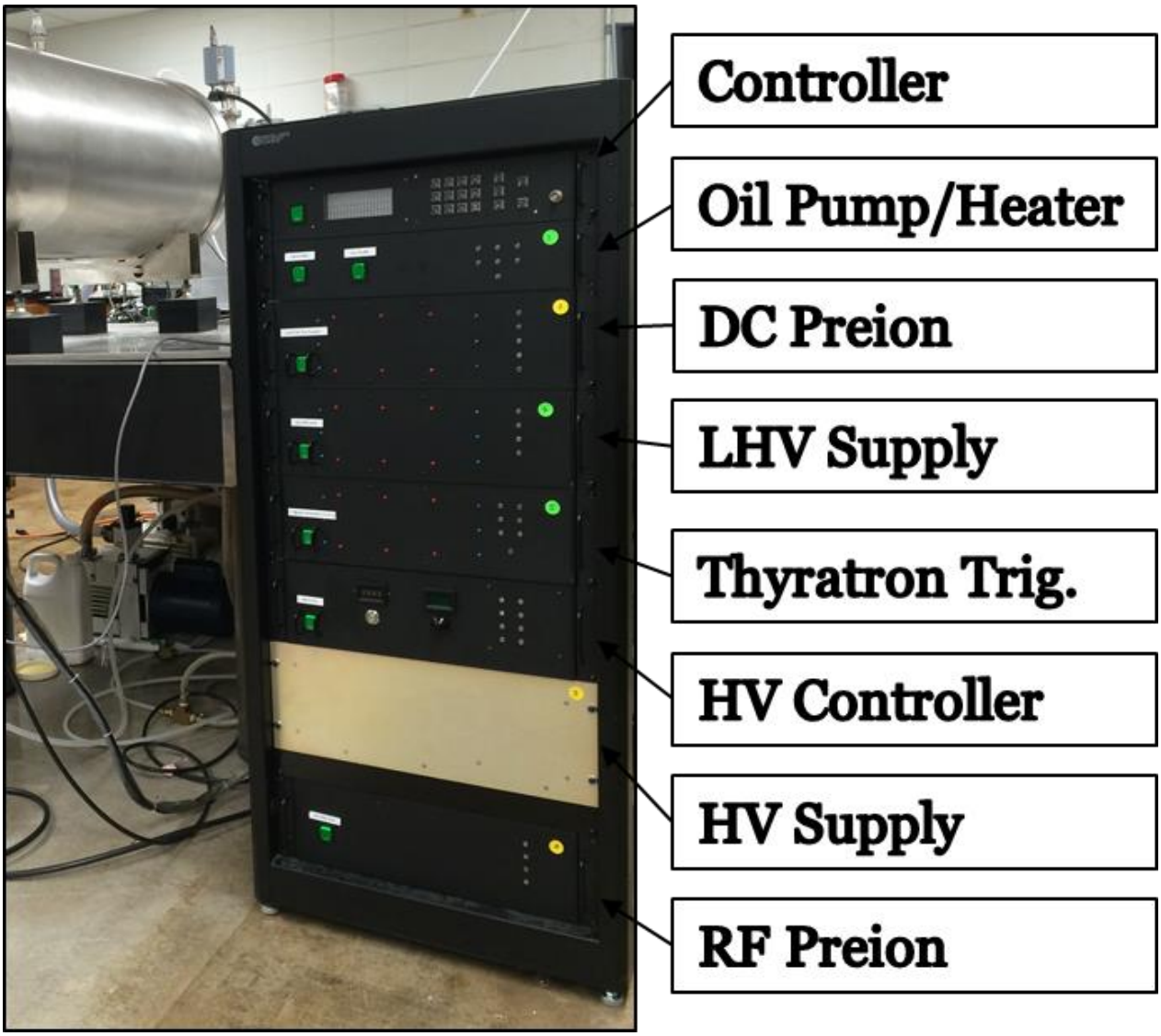
Appendix C.2: Control Table Contents

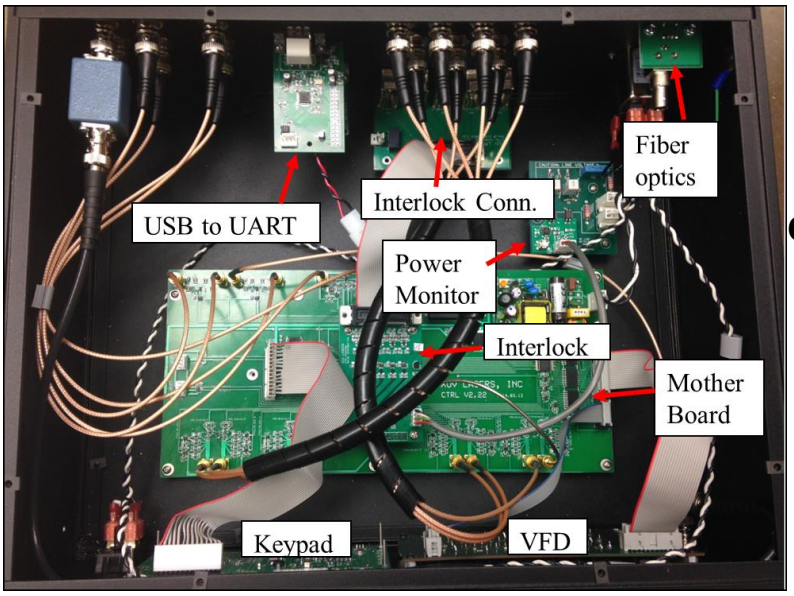
Appendix C Database Structure





Appendix D GUI Pictures





Controller Layout