Kicker Monitoring & Control User Guide

Andre Frankenthal

July 26, 2016

1 Introduction

This document is a detailed description of the monitoring and control system for the kicker in the new Muon g-2 experiment, located at Fermilab, in Batavia, Illinois. The main purpose is to instruct users of the system on how to operate the kicker and how to debug eventual issues that arise. An additional goal is to enable maintenance and further development of the monitoring and control (M&C) system by people not familiar with its initial conception.

The layout of this document is as follows: a brief overview of the kicker system is given, followed by a more in-depth description of the M&C. The description of the M&C will be further divided into a beginners' user guide, for people that simply want to use the system, followed by a more advanced description, for people who want to understand what it does and how to improve it.

2 A New Kicker For a New Experiment

3 The Monitoring and Control System

3.1 Overview

The M&C system's overarching concept is that of simplicity: it should be very easy for someone completely new to the apparatus to be able to operate it in a few minutes, at least for basic operations. It should also be very straightforward to install, requiring little effort from an already busy personnel to get it set up.

With these ideas in mind, the chosen design for the system was a web interface. This has several advantages: it is completely portable across operating systems, since it runs on a web browser. The only requirement to be able to run the M&C is to have a reasonably modern and up-to-date web browser. There are no installations required: as long as the M&C box and the computer are on the same sub-network, simply connecting to it via its IP address should do the job. In addition, the recent quick explosion of web API's allows one to have a beautifully designed interface with little manual effort. The advance of

system-on-a-chip boards further allows one to host a web server from very small computers (hand-sized), so that the physical space overhead is minimal.

All these factors led to a design decision to use web servers hosted on a small system-on-a-chip created by Intel, called Intel Galileo Gen 2 (just Galileo from now on – not to be confused with the scientist!). The Galileo is an Arduino-certified development board based on an Intel processor, the Quark, and runs a distribution of Linux underneath. It provides the 'the best of both worlds': an Arduino environment to manipulate IO pins with ease and communicate with peripheral devices and other boards in a simple manner, and also a full Linux environment that is able to run normal processes and languages like Javascript and Python. The Galileo is shown in Figure 1.

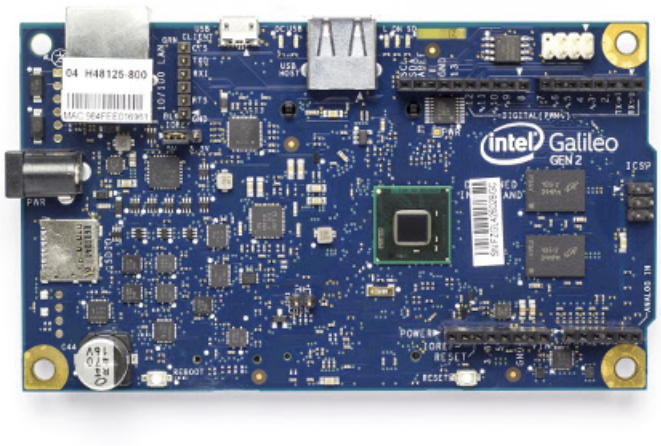

Figure 1: The Intel Galileo Gen 2. This board is used in the kicker M&C to host a webserver with monitoring updates and also provides the ability to change control states.

To actually handle the IO information we chose the Arduino Due board, which is a microcontroller based on the Arduino family, but with several enhancements from the original Arduino Uno microcontroller. The Arduino Due has 54 digital IO pins, 12 analog input pins, and 2 true analog output pins. In addition to the true analog outputs, 12 of the 54 digital pins can be operated in PWM (Pulse-Width Modulation) mode, which makes an analog voltage out of a square wave, with the duty cycle translating to an average voltage. The large number of pins was the major factor in the choice of the Due. In addition, this board runs at a clock of 84 MHz, which about 5 times faster than the average Arduino board (at 16 MHz). The Arduino Due is shown in Figure 2.

The overall design of the system is shown in Figure 3. It can be seen that the IO lines are connected to the Arduino Due, which handles trigger generation, switch control, and acquisition of monitoring data with an ADC. It then communicates with the Intel Galileo, which receives the collected data and hands

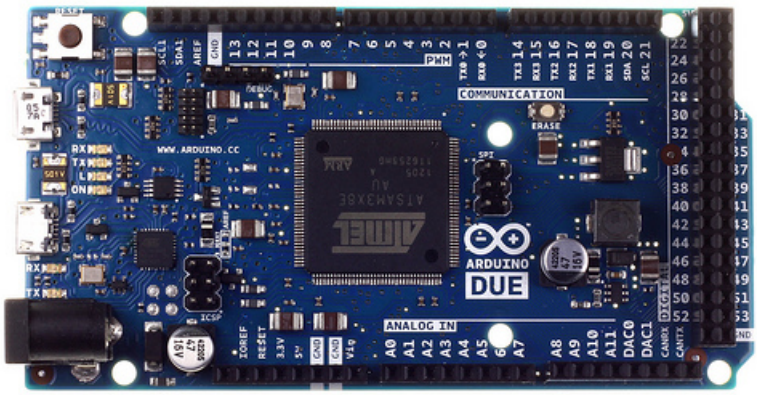

Figure 2: The Arduino Due. This board is used to connect all IO lines, and to send it to the Galileo for display.

it on to the webserver for display, and also sends back the new commands for the Arduino to execute, such as the timing of the triggers and the status of the switches, which was collected from the webserver and hence from the user.

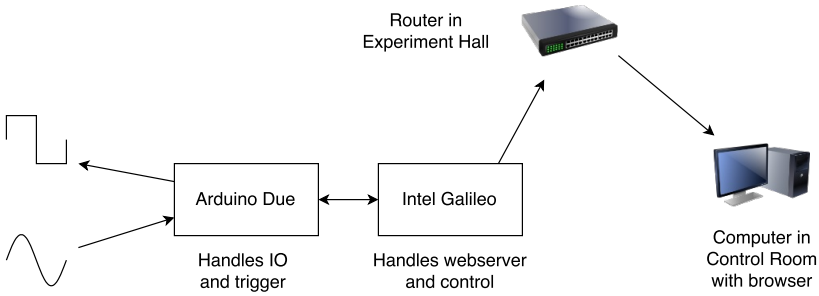

Figure 3: The overall design of the system. The Arduino handles all the IO, including triggers, switches, and ADC readouts, while the Intel Gailleo handles the communication with the client computer in the Control Room.

3.2 Beginner's User Guide

Once the system is in place, with the power supply connected and an Ethernet cable connected to a computer in the Control Room, all that is needed to do to access the interface is type in a web browser the IP address given to the Intel Galileo board, either by some DHCP server or statically. Note that the port of the server is 8081. So one only has to type (say) `192.168.214.36:8081` on the browser to connect to the web interface. The result of this action is shown in

Figure 4.

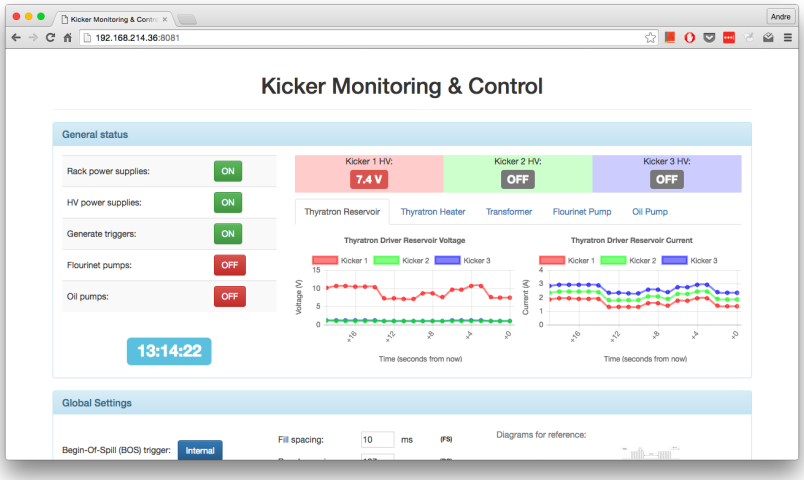

Figure 4: The interface of the webserver hosted by the Intel Galileo. The IP address used here is local and will change according to the location of the boards and the network configuration.

On the left of the interface one can find a table with switch controls, labeled, respectively, “Rack power supplies”, “HV power supplies”, “Generate triggers”, “Flourinet pumps”, and “Oil pumps”. If all switches are off and one wants to turn the kicker system on, the first thing to do is to switch on the rack power supplies. Note that the other buttons are disabled while the rack supply is off, to prevent inadvertent operation of the kicker. Once the rack supply is on, then the HV power supply switch becomes enabled, and one can then turn it on. After that operation, a timer starts just below the trigger switch, and when the timer is finished the switch becomes enabled as well. This timer is necessary to ensure that the thyratron drivers have enough time to warm up before they are issued triggers. Also, after the HV power supply is on, the pumps can be turned on as well.

On the upper central part of the screen is the monitoring for the HV power supply, one for each kicker. Note that each kicker has a unique color, which is always the same throughout the interface. The monitor reports the current value for the power supply voltage, and compares it with the set voltage: if they are the same (to about 10%), then the box is colored green, to indicate everything is fine with that power supply; if the monitor voltage is different from the set voltage, the box turns red to indicate something is wrong. If that particular kicker is turned off, the box turns gray and displays ‘OFF’.

On the central part of the screen the monitoring of the thyratron driver reservoir is displayed. The left chart is the reservoir voltage, and the right one

is the reservoir current. Again the three colors correlate to the three kickers. It is also possible to hide a given kicker plot, by simply clicking on it in the legend above the charts. To show it again just click on it again. One can also see other monitoring charts by clicking on the appropriate tab.

Figure 5 shows the same page, but scrolled down to display the remainder of the interface.

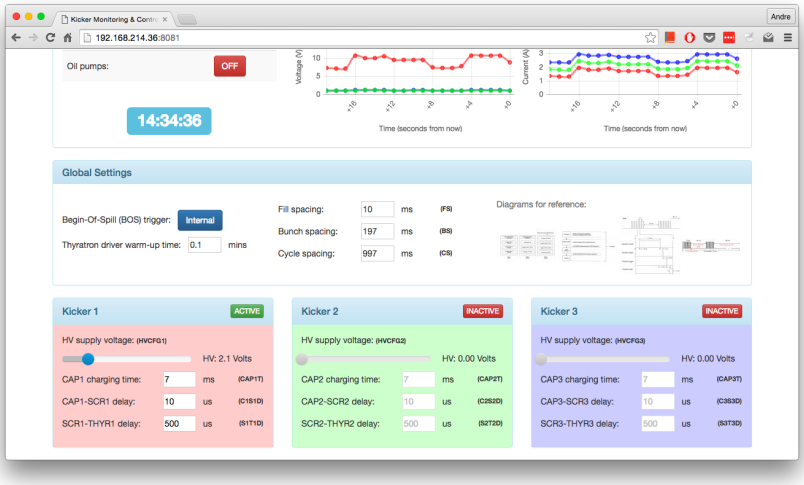

Figure 5: The bottom part of the interface of the webserver hosted by the Intel Galileo, showing the control options for the trigger and the voltages.

The image shows the three panels to configure trigger and voltage properties for each individual kicker (bottom part of the page). The slider enables change of the kicker power supply voltage, while the text boxes control the spacing and duration of different trigger pulses. In the middle of the page one can see the global settings panel, which allows one to change the properties of the Begin-Of-Spill (BOS) trigger, such as spacing between each fill, between each bunch, and between each cycle. Note that to the right of these text boxes one can find a helpful set of images and diagrams explaining in detail what each setting does. Just click on a thumbnail to enlarge the picture. Finally, to the left of the text boxes there is a button to toggle a switch between internal and external BOS trigger: if the button says 'Internal' then the C&M system generates its own BOS trigger; in this case, the "BOS Trigger" pin in the box will output the BOS trigger for synchronization. If the user toggles the switch, it changes the setting to 'External', and now the same "BOS Trigger" pin becomes an input and receives the BOS trigger from elsewhere, thereby initializing the trigger sequence to the kicker.

3.3 In-depth Description of M&C

3.3.1 Circuits and diagrams

All the circuits and pieces of the kicker M&C system are inside the box shown in Figure 6.

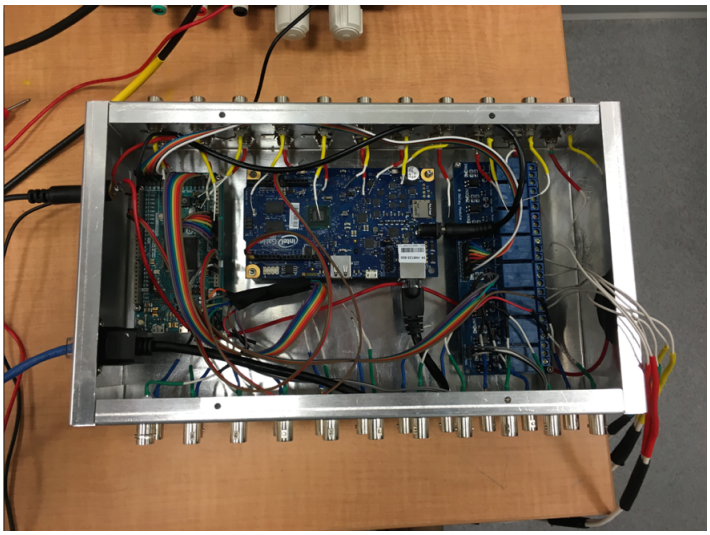

Figure 6: The box containing the circuitry of the M&C system. On the left one can see the Arduino Due, on the middle the Intel Galileo, and on the right a relay module, connected to the Due.

The box takes one power supply, connected by a regular 2.1 mm DC barrel jack, and distributes the power to all components appropriately. Note that the whole chassis of the box is grounded. On the same end as the DC jack there is also an Ethernet jack, which connects the Intel Galileo to the control room where the client browser will be running.

Figure 7 shows a different view of the box, with a large number of BNC connectors. These are used to send trigger signals to and receive monitoring signals from the kicker. The central pin of each BNC is connected to a pin in the Arduino Due, and the other terminal is connected to the chassis, and thus grounded. The only exceptions are the three BNC's on the bottom right of the figure, which are connected to the kicker power supplies by means of a relay – in this case, the BNC needs to be isolated from the chassis to complete the relay circuit (one can see a white isolating material separating the chassis from the BNC for these connectors).

For reference purposes a diagram of the wire connections inside the box is shown in Figure 8. Note that only one half of the BNC connectors is currently operational. The other side of the box also has BNC's that haven't been wired up yet (there is also space for spares).

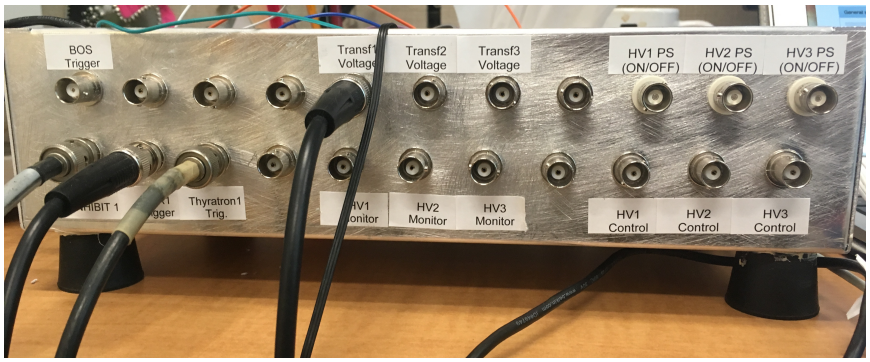

Figure 7: The front of the box showing the BNC connectors. Notice that the 3 top right connectors are isolated from the chassis.

3.3.2 Software and Linux

There are three distinct pieces of code in the kicker M&C system that need to be dissected separately: the Arduino program running inside the Arduino Due, the Arduino program running inside the Intel Galileo, and the HTML/CSS/Javascript web server also running inside the Intel Galileo. The logical organization of the software follows the overall design of the system: the Arduino code in the Due is responsible for generating the trigger signals, according to the timing and delay specifications that are programmed by the user, and also for reading the ADC values of the analog channels, and for sending that information to the Intel Galileo.

The codebase is available at GITHUB:afrankenthal/kicker-arduino-code and GITHUB:afrankenthal/kicker-server. Here we shall describe the main pieces of each code.

Arduino Due Code: The file kickerControlArduinoV2.ino contains the Arduino code for the Due. The main idea is to use a schedulizer library called TaskScheduler to set up independent tasks for each of the roles the Arduino Due has to perform. So, for example, there are tasks to control the trigger logical operation ('HV', 'CS', 'BS', and 'FS'), a task to read data from all enabled channels in the ADC, and a task to send all this data to the Intel Galileo and read new configuration parameters from it, using the SPI (Serial Peripheral Interface) communication protocol. There are some intricate pieces of code to configure the SPI and the ADC correctly for the purposes of the kicker monitoring and control.

The ADC needs to be set up in free-running mode, which means it continuously acquires data on the enabled channels. This is necessary to maximize the sampling rate of the ADC, which is nominally 1 MSPS. Note that the Arduino Due has only one ADC for all its analog channels, which means for each new sampling point, the ADC switches the channel it is acquiring the data from.

This naturally reduces the effective sampling rate per channel. In the kicker case, we aim for 3 channels where we require faster sampling rate (signals with timescales of microseconds), with other channels only requiring slow sampling rates, on the order of 1 Hz. This means our 3 fast channels will have effective sampling rates of about 333 KSPS.

To read the ADC data, we have several read buffers to which the ADC sends its free-running data, regardless of triggers. The ADC cycles through all the enabled channels, which are configured directly using ADC configuration registers. The user can enable or disable channels using the web interface, which are relayed to the Due via the Intel Galileo. After each ADC buffer is full, it is added to a different output buffer, which is used to send this raw data to the Galileo via SPI. The Arduino Due doesn't do any processing of the raw data. This is on purpose to ensure that the code is as efficient as possible. The tasks of the Due are time-critical, and any extra overhead processing could cause it to slow down to the point of losing real-time functionality, like issuing triggers at the right time. The raw data consists of a 16-bit number, a 12-bit ADC reading plus a 4-bit channel identifier, indicating which channel the ADC read that number from.

The SPI is used to exchange data between the Galileo and the Due synchronously, with the Galileo acting as master and therefore setting the clock speed, and the Due as slave. To set up the Due SPI interface as slave requires some direct modifications to the SPI registers, as it's usually configured to act as a master to several peripherals. The Due sets up the output buffer ready to send out whenever the master (the Intel Galileo) decides to initiate a transfer.

Since the Due has such intensive data in/out overhead, it would consume all of its processing capabilities if it had to take care of all the data input and output itself. This would leave no room for the microprocessor to handle the other tasks, such as issuing triggers. Fortunately the Due includes a feature called DMA (Direct Memory Access), which handles data processing from peripherals to memory and vice versa. The DMA is a module independent from the microprocessor, which connects both to the ADC and to SPI, and automatically transfers data from the ADC to memory and from memory to SPI and then out to the Galileo. This frees up the processor to handle other tasks simultaneously.

If interested, one can find more detailed comments about the program inside the code itself.

Intel Galileo Arduino Code: The file kickerControlGalileoV2.ino contains the interface for transferring data between the Galileo and the Due, on the side of the Galileo. Its basic function is to read the control data from the web server and relay it to the Due via SPI, and conversely to obtain the monitoring data from the Due and relay it to the user via the web. The communication between the Arduino program and the webserver happens via text files with the JSON format, so in the Galileo Arduino code we use a library called ArduinoJson that allows for JSON parsing easily. The flow of information follows the pattern: JSON file $\rightarrow$ char array $\rightarrow$ JSON object $\rightarrow$ condensed output buffer to the

Due. The Arduino code is also responsible for setting up the SPI protocol and initiating a transfer between the Due and the Galileo.

A noteworthy comment is that the SPI in the Galileo (as far as we have been able to gather) can unfortunately only transmit 8 bits of data per packet, so any data that requires more than 8 bits (such as ADC readings, which consist of 12-bit readings plus 4-bit channel id's) must be manually split into two packets before sending and then reassembled after receiving the data on the other end. This makes the process a bit more cumbersome, but not an overall impediment to the use of SPI. The advantages of using SPI include its speed, which is faster than any other communication protocol available for the Due and the Galileo, and the fact that it's a synchronous transmission, which means that we don't need a 'handshaking' protocol to establish the communication, thereby reducing the amount of necessary overhead.

Intel Galileo HTML/CSS/Javascript webserver code: The purpose of the webserver is to collect the monitoring data received from the Arduino side of things and to display it in a convenient and friendly way to the user. The code reads the JSON file containing monitoring data and makes it available via graphs and charts, according to the description in Section 3.2. It also collects the control information that the user set into a control JSON file to be processed by the Arduino code and then sent to the Due.

The web interface allows for nice and user-friendly content, and requires minimal installation from the user side. Currently the server runs at port 8081 of the Intel Galileo. So to access the server from a client computer, just point the browser to the address `http://GALILEO_IP:8081`, where `GALILEO_IP` is the IP address of the Intel Galileo that should have been configured previously by the network administrator.

The interface contains several elements to help the user control the kicker. For example, if at any point the server loses connectivity with the client an error message is displayed at the top of the page to warn the user. Also, before beginning to issue triggers, the user must wait a predetermined (and configurable) time for the thyratron switches to warm up. This wait is enforced by a timer that starts when the power supply button switch is turned on. The trigger button is disabled until the timer ends, at which point it becomes clickable to the user. In addition there are helpful diagrams to illustrate which controls in the interface act on which pieces of the apparatus.

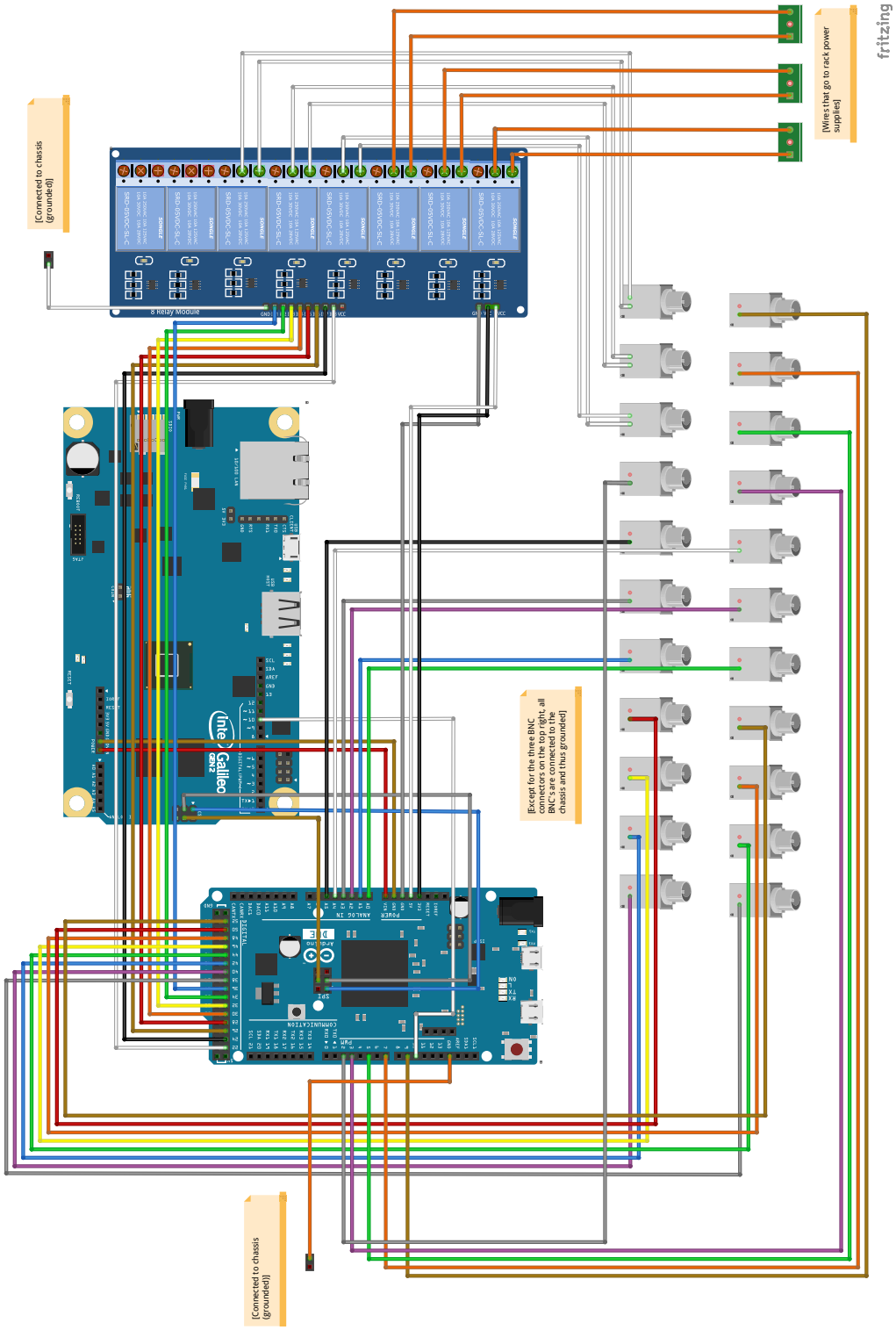

Figure 8: Wiring diagram of the circuits inside the box.