Electronic Build Project:

"A Computer Based Master for ITR & IBM Minute Impulse Secondary Clock Movements"

By Joe Fox, KD4MS

I have seen many requests from people on various forums, asking for information on how to get an old IBM slave clock working. This article may help some of you. Movements, like the one in Photo 1, are often referenced as International Time Recording (ITR) or International Business Machine (IBM) slave clocks but this is a misnomer. The correct name for these movements is "Secondary Movement". IBM minute impulse secondary movements, along with time and attendance recorders, time stamps, event programmers, buzzers, bells, and other time recording devices were used for decades in many schools, factories, and businesses for coordinating time and attendance.

Photo 1: Art-Deco style 1930 International Time Recording model type 561-B, that was installed in the Executive offices of Woodlands Division International Paper Company, Panama City FL. This unit features fancy hands, a brass cursive "International" logo, sixty brass minute markers (12 diamond and 48 rectangular shaped), and fifteen brass numerals; each individually mounted on a nickle face with brass screws. See some of the numerous mounting screws in photo 2, on the right.

Photo 2: Secondary on the left, showing the hinged soft copper shell and wood blocking for support.

Electronic Master:

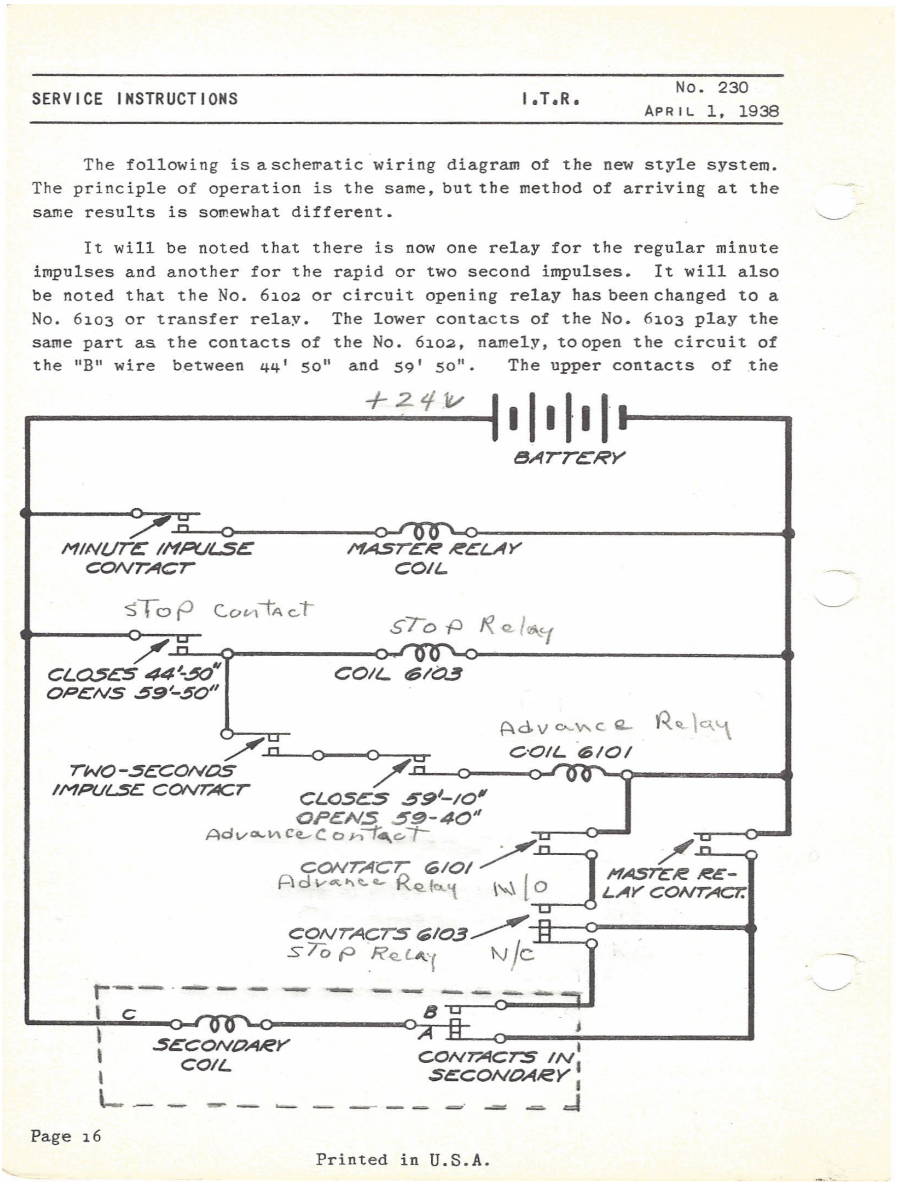
This article describes how IBM's time control system works and discusses three different versions of a computerized equivalent master clock. A master is needed to accurately operate one or a hundred IBM minute impulse self-regulating secondary movements as timepieces. The electronic masters detailed here control three-wire self-correcting minute impulse secondaries (notice no second hand).

So if you are lacking a functional master, build one of the versions of this project and get your IBM impulse secondary working accurately and electronically without ruining your pocket book.

There are three choices of electronic processors offered here; each with different complexity's and costs.

1- An Arduino Uno or Mega, plus an Ethernet shield, including their open-source clones,
2- An ESP8266-12E NodeMCU wireless IoT network processor (the cheapest @$6.50),
3- A Raspberry Pie Zero W(ireless networked) computer. (as low as $5.00 plus memory)

Each version uses a pair of Insulated-Gate Bipolar Transistor MOSFET devices (IGBT) to drive the impulse secondary's coil, and each includes their own version of software [in downloadable files]. The first two processors are programmed using Arduino's free Integrated Development Environment (IDE) software, and the third uses master clock software ported to Linux and executed at boot time.



SERVICE INSTRUCTIONS I.T.R. No. 230 APRIL 1, 1938

The following is a schematic wiring diagram of the new style system. The principle of operation is the same, but the method of arriving at the same results is somewhat different.

It will be noted that there is now one relay for the regular minute impulses and another for the rapid or two second impulses. It will also be noted that the No. 6102 or circuit opening relay has been changed to a No. 6103 or transfer relay. The lower contacts of the No. 6103 play the same part as the contacts of the No. 6102, namely, to open the circuit of the "B" wire between 44' 50" and 59' 50". The upper contacts of the

Page 16

Printed in U.S.A.

Schematic 1: Page 16, from ITR Service Instructions Number 230, April 1, 1938. Reprinted by permission from Reference Desk, IBM Corporate Archives, Route 100/CSB, Somers, NY, 10589

Schematic 1 shows the most basic and latest wiring of an IBM self-regulating time control system that was produced as of 1938 and is the basis of this computerized master clock. IBM (formerly International Time Recording Co.) produced these master clocks and impulse secondaries from around the 1880's to 1958, in the United States. Photo 3 shows the mahogany master clock that was originally installed at International Paper Company's Panama City, FL mill. I purchased it in 1966, from a fellow IBMer; however the clock's contacts and wiring were removed when it was traded-in to IBM.

Photo 3: My 1930 IBM master clock, originally installed at the International Paper Company's Southern Kraft plant in Panama City Florida. This master has a 60 beat escapement, motorized auto-winding drive weights , a temperature compensating Invar-steel pendulum rod and mercury pendulum.

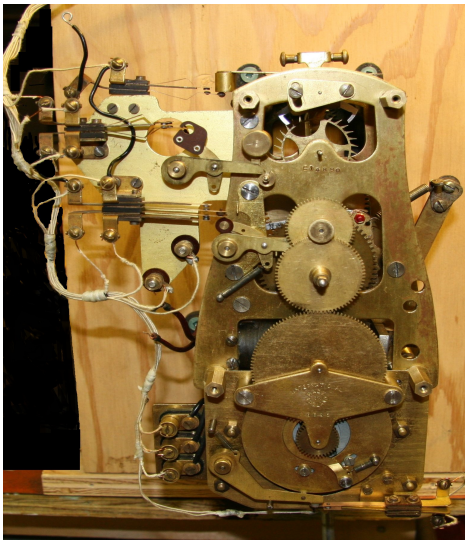


Photo 4: 1950's IBM master clock movement showing original wiring and contacts. It has a 60 beat escapement and an impulse wound spring drive. It also has the IBM accumulator feature attached at the bottom of the frame (bottom 1/3rd). This feature automatically corrects all secondaries after a power loss.

Early model master circuits were made obsolete by the changes of April 1, 1938 and without their original system documentation, troubleshooting a problem on a mechanical master clock can be a real time consuming adventure, and they often are a constant maintenance headache. Whereas a computerized master clock using a simple IGBT transistor driver circuit can make the electronic master clock virtually maintenance free. A really nice thing about IGBT drivers is that they stay cool and report “no audible noise”. The master and duration relays in a mechanical master clock make an incredible amount of noise. Over time, these electronic masters (Photos 11, 12, 13, and 14) will keep better time than the brass, nickle, and mercury master clock living in my foyer.

It was the gift of an Arduino Mega from Phil, my son-in-law, that prompted me to finally develop an electronic master clock. Two of my most cherished secondaries, Photos 1 and 5, came from the same paper mill and another, Photo 6, from Cove Elementary School where I attended the second through sixth grades. You can see original wiring and the contact assembly of a master I repaired for a friend in Photo 4, and without these contacts my master can not run my secondaries. Since the day I acquired my IBM Master clock I've wanted my own clock system. Now I have three masters to choose from.

A good reference you should know about is the *Reference Manual of Electrical Characteristics*, IBM Form # 231-6401-0. This has important electrical information for IBM time equipment devices, including model types, wire size, wire distances, voltage, current, and power requirements, as well as tables needed to calculate parameters of a complete time system installation.

Published in this document is Table II, "*Amperes Required By DC Impulse Magnets*", on page 9. Here you will find, of particular interest, a list of electrical current requirements by DC voltages of twenty six model types using coil voltages of 12, 24, 48,115, and 230 volts DC. This same document is also printed as part of the *IBM (Time Division's) Sales Manual*, as the Electrical Appendix, Copyright IBM, December 1951. Please request an electronic copy of the reference manual from the Reference Desk, IBM Corporate Archives, Somers, New York, 10589.

Self-Regulating Operation:
All three versions of the computerized master clock are programmed to output two signals to their General Purpose Input Output (GPIO) data pins. This configuration uses the same logic as the wiring of the relay technology wiring design as shown in the 1938 IBM schematic (See Schematic 1).

Starting at the top of the hour, an A and B pulse is presented, for a duration of 0.4 seconds ON, then OFF, once each minute, with the following exceptions:

- The A signal is raised once per minute at zero-seconds, and on every odd second between 10 and 50 during the 59th minute. Therefore the A line will get an extra 21 pulses during the 59th minute, totaling 80 pulses per hour.
- The B signal is raised once per minute at zero-seconds for each minute except for minutes 50 to 59, where the B signal is stopped. Therefore the B pulse only occurs 50 times per hour.

Secondaries that are late (slow) [due to mechanical or electrical failure] will receive the A pulse once per minute until the Master is at the 59th minute, then receive up to 21 rapid A pulses every other second, until late secondaries reach their 59th minute. After the 59th minute, the secondaries will switch from their Normally Closed (N/C) "A" contact to their Normally Open (N/O) "B" contact, and will only advance to the top of the hour on the next B pulse. Most secondaries will switch back to the "A" contact at four minutes after the hour, depending on the model of the secondary.

Secondaries that are early (fast) [due to mechanical or electrical failure] will receive the A pulse once per minute until their 59th minute. Then they switch from their N/C "A" contact to their N/O "B" contact, and will ignore the A pulses while waiting for the next B pulse, before advancing.

The next B pulse will step all the secondaries forward from their 59th minute to the top of the hour and will continue to advance the secondaries with the B pulse until the secondaries switch their cam operated contacts back to the N/C "A" contact.

There are some early model secondaries that switch back to the B contact at 14 minutes past the hour and these are compatible with this master clock scheme since the B signal continues until 50 minutes after the hour. If a late secondary is more than 20 minutes late, but less than 50 minutes late, then correction will continue to the next hour's synchronization cycle. The service manual says, severely out of time secondaries could take up to three hours to become synchronized. Movements with more than a few minutes early or late without reason are considered in need of repair.

Electronic Operation:
I am referencing the Arduino in the following text as the driver circuits and operations are the same except for the GPIO pin identifications (see schematic 2). Differences and additions will be noted in the descriptions of the ESP8266-12E NodeMCU and Raspberry Pi Zero W sections later. Here is how the IGBT driver interfaces the secondaries. The Arduino outputs a +5 volt logic level at output pin "D9" (see schematic 2).

This powers and illuminates the “A” LED for a visual indicator and presents a +5 volt electromagnetic field (EMF) at the gate of the IGBT. Like a switch, this electromagnetic field turns ON, or opens the gate of the IGBT, causing the source to drain junction to become a virtual short circuit (0.046 ohms – less than half of a 1/10th of an ohm), effectively grounding the A line. Electron current flow is from the negative terminal of the 24 volt power supply, to ground, into the transistor's Source, out the transistors Drain, through the A line, through the (N/C) A contact, through the secondary's coil, through the C line, and back to the positive terminal of the power supply. This completes the circuit and magnetizes the coil's steel core. The core's electromagnetic field now causes the secondary's armature to rotate approximately 70 degrees. Then 0.4 seconds later, output pin “D9” returns to zero voltage, the LED turns off and the gate closes. The transistor stops conducting, stopping current flow, causing the core's magnetic field to collapse, allowing the armature to return to it's resting position and moving the minute and hour hands forward one minute.

Photo 5: This model type 561-2 secondary was installed in 1930 in the Engineering Department of International Paper Company's Panama City, FL plant. International Time Recording (ITR) logo, face rim made of soft copper and painted Black, face painted White, with a flat glass cover.

Photo 6: This model type 565-2 was installed in February, 1955, at Cove Elementary School, 2nd grade classroom, Panama City, FL. (I was in the 3rd grade!) This movement has an IBM logo on a steel face painted white, a domed glass cover, and is surrounded by a wrinkled finish painted rim around a steel back plate.

Be aware, the system only corrects the secondary's minute, not the hour. To correct the hour you need to manually advance the secondaries. The standard Run/Advance Switch feature found on all IBM master clock systems was added to my computerized master clock version in July, 2016, both in software and hardware, with the addition of a N/O switch. A GPIO pin is defined as a digital input for this added hardware switch. The Run/Advance switch is used to advance all secondaries forward manually, at a one second pulse rate by pulsing both A and B lines without correction. Normal operating position is “Run” and should be in this position when booting. This is very handy for daylight savings time changes and manual setting of movements when working on the system or testing a secondary. When resetting movements because of daylight savings time, it will take eleven minutes to advance all secondaries eleven hours to fall back to standard time in the fall, but only one minute to spring forward one hour in the spring. Note: this advance is twice as fast as IBM's mechanical master.

Checking out a Secondary, or other minute impulse device:
Using information from previous paragraphs, you can easily check out an impulse secondary manually and electrically. First, make sure that 12 o'clock is in the up position.

Manually on a model type 561, advance the movement by first pressing the armature horizontally (clockwise) with your finger, then release the armature to advance the minute hand (see photos 17 & 18). On a model type 565-2, rotate the armature clockwise by pushing a pole piece with your finger, then release, advancing the movement one minute. Looking at photo 8; the armature is shown as nine (seven visible), 90 degree formed tabs on a rotating plate and seen as the lighter white inter ring of dashes in this photo.

Photo 7: This circa 1930, ITR Type 561-C secondary movement has a white face with the INTERNATIONAL logo printed on it. Like all 561 model types, it has a drive and check pawl drive system, a rotating armature, 24 volt and part number coil markings, a cam operated contact set switching at fifty-nine minutes from A to B, returning to A at four minutes after the hour.

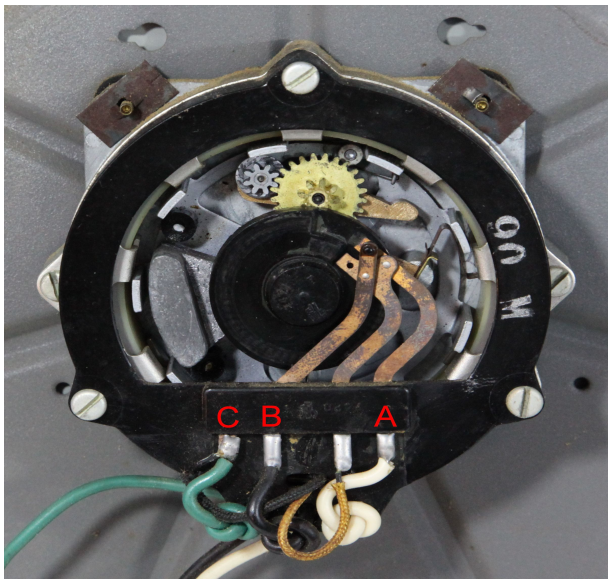


Photo 8: 1955 Type 565-2, back of secondary movement of Photo 6. Note the rotating armature appearing as the lighter white circle of dashes inside the (black) circular coil. (seven visible) The darker dashes are the electromagnetic poles of the coil. The center black disk is the "A" and"B" contact cam.

To advance a movement electrically, you should first perform continuity checks with an Ohm meter using the resistance values listed in my table 1, for open, shorted, or grounded (to frame) coils. "Using a meter" section described later in this article may be helpful. Then simply apply +24 volts (or +12 volts) at the C terminal and momentarily connecting terminal A (or B if the minute hand is between the 59th minute and the fifth minute) to the negative power supply connection. See the example in photos 19 and 20. The C wire from the movement is the one going directly to the coil. Grounding the A wire energizes the coil and operates the rotating armature for about 70 degrees of rotation. When you remove the connection to the ground terminal, the armature returns to its resting position, pulling the minute hand forward. Repeating this procedure using the A terminal will advance the hand until the 59th minute, then the movement will stop. Now connecting the B terminal to the ground terminal will advance the movement to the top of the hour and for approximately four more advances, then stop again when the cam operated contacts are switched back to the "A" normally closed position. The IGBT's simply ground the A and/or B line. These things are slow! That is why there is a 0.4 second pulse width, to ensure a long enough signal for the armature to rotate.

Important, be sure the secondary is vertical with the 12 o'clock position up, because drive and check pawl engagement is operated by gravity and will not engage correctly when laying down. Any impulse device such as an accumulator, program device, impulse recorder or dial recorder can be tested electrically as above.

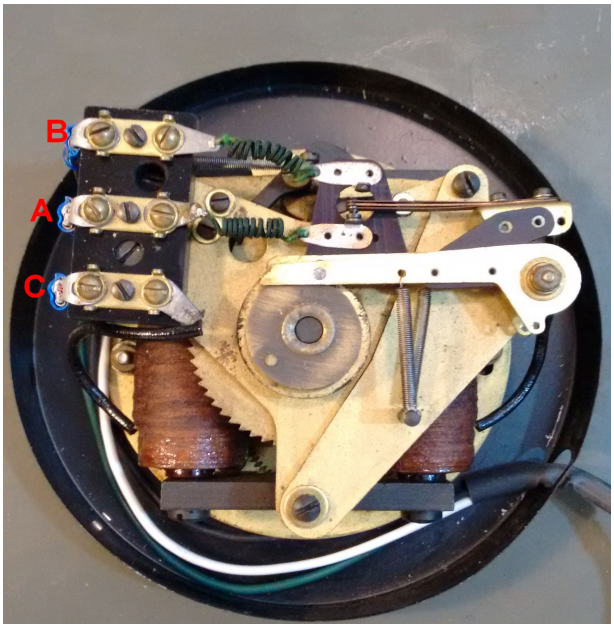


Photo 10: 1930 model 561-2, back of secondary movement of photo 5, showing the terminals for connecting the A, B, and C system wires. Notice the different styles of contact mechanisms between this photo and photos 7 and 10.

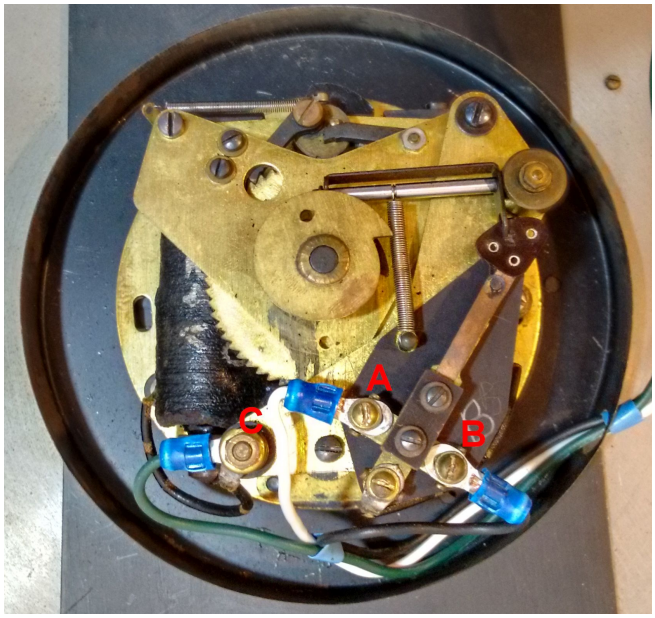


Photo 9: 1930 model 561-2, back of secondary of photo 1. Showing the terminals for connecting the A, B, and C system wires and the cam operated N/C and N/O contacts. Note: This photo was taken with the movement upside down so the check pawl, which is operated by gravity, is out of position. It is a photo edited blow up and 180 degree rotation of photo 2.

Determining an unknown coil voltage of a secondary:

Often the the paper label pasted to the back of all units, showing the model, serial, voltage, etc., of the movement, is either too deteriorated or has been partially removed by the seller, thinking it will sell better without this critical information. Table 1 shows measured values of some of my type 561-2 and type 565-2 models and some hints as what to do to determine your model's coil voltage if it is unknown. Type 561s should have the voltage and part number printed on the coil's linen wrapper in white ink, so look closely. The following procedure should also be helpful. Measure the coils resistance to ensure continuity. This is a DC circuit and we are measuring copper wire resistance of the coil which will determine the current through the coil. However, the amount of core steel matters as to how strong the magnetic field will be when energized and is an unknown factor. We want to be sure that the coil is not open, shorted, or that there is not a problem with the A and B cam operated contacts. Using a 12 volt DC “wall wart” type power source with a current rating of 300 mA [or greater] will do fine. First connect the secondary as detailed in photo 20, + voltage to the C wire and (–) ground to the A wire. The armature should rotate solidly. If it does not move or if it is slow and/or does not completely hit its stop every time you open and close the circuit, then it is not a 12 volt coil. The service manual says the coil should work with ¾ rated voltage. If the procedure failed with 12 volts, then repeat the procedure with a 24 volt DC power source. You can also measure the current as described below, to make sure the current is near the 0.170 amps for a 12 volt coil or 0.085 amps for 24 volt coil, as indicated in Table 2. Although my measurement of a 565-2's resistance and current seems correct compared to table 2 values with 12 volts applied, there does not seem to to be enough magnetic field produced to make the armature operate properly. Conclusion, if the voltage is unknown, connect a 12 volt power source first and see of the armature operates completely and reliably. If it keeps time then it is probably a 12 volt coil. If not, then try connecting to a 24 volt power source and measure the DC current.

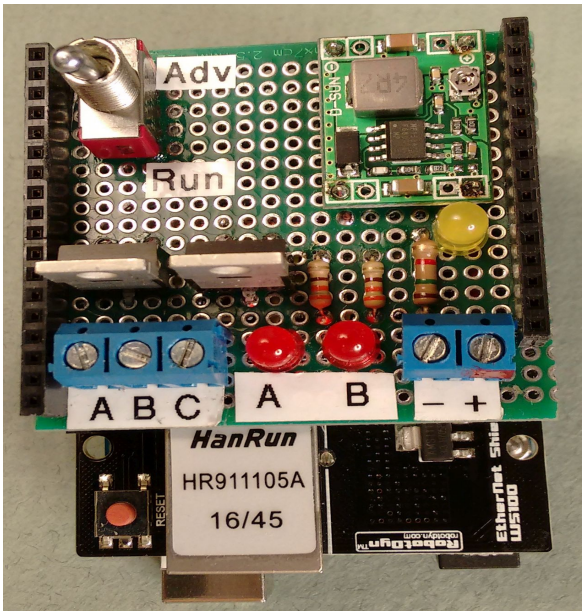


Photo 11: Version 1, Top view of the Arduino Uno, RobotDyn Ethernet shield, and DIY IGBT driver shield, stacked. Yellow LED is a 12/24 volt "Power ON", indicator. This LED is dimmer at 12 volts.

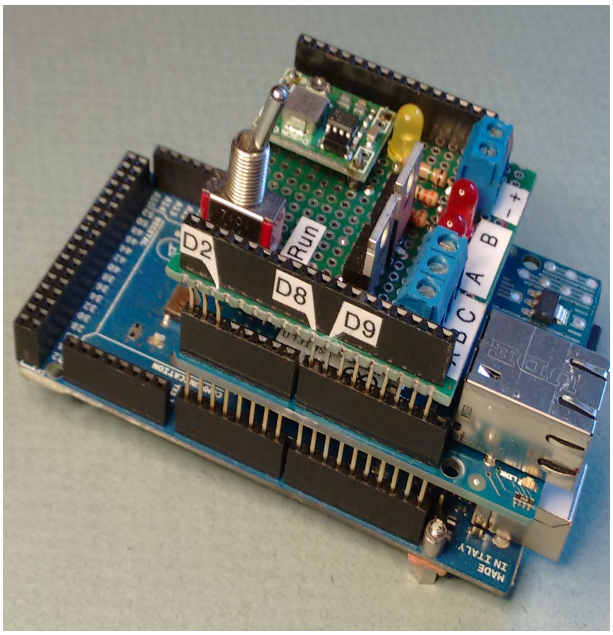


Photo 12: Version 1, view showing the Arduino Mega, a clone Ethernet shield, and DIY driver shield, stacked. Note the GPIO pins: D2, D8, and D9.

Using a Meter:

A word about digital multi-meters. There are many Digital meters on Amazon.com for under $20.00. Some meters have auto-ranging and auto-off features. You can find free digital meters occasionally from Harbor Freight if you buy something else (or buy it @$5.99). I have gotten a couple of these free; cheap but work just the same. I usually use an IDEAL 61-340 model, four digit, auto-ranging that I purchased for about $40.00 from Lowes Lumber Company years ago. The Ideal 61-340 is the meter I used to measure voltages, resistances, and currents, listed in these tables.

To measure the movement's resistance:

Set the switch to Ohms, the red meter lead in the hole labeled V,O,Hz etc, and the black lead into COM. Look at the display for "OL" meaning open load or open circuit. Shorting the red and black leads together you should get 000.0 ohms. To measure the coil's resistance, start by connecting one lead to the movement's green wire (C) and depending on the movement's time, the other lead to the White wire (A), measure the resistance in ohms (a reading of 0.245K means 245 ohms). If you have a 561-2, then you have enough information to determine the voltage of the coil from table 1. If you have a type 565-2 then maybe you have enough information. I do not have a 12 volt type 565-2 to compare the resistance to so it is not as apparent. It is my guess that the resistance of a 565-2 with a 12 volt coil will be around 70 ohms. Since we are measuring direct current coil wire resistance here, and based on R=E/I (ohms law), the resistance should be 70 ohms, computed based on the published 0.170 amps and 12 volts DC applied for this model. If it is not then the best I can recommend is to use the trial and error method previously described as a plan of action.

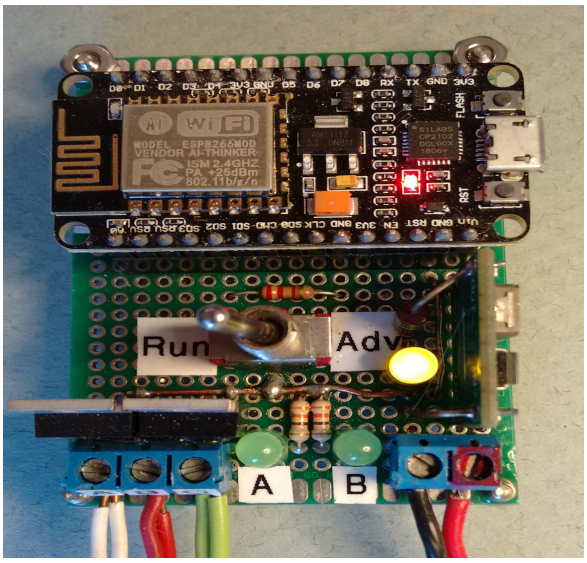


Photo 13: An ESP8266-12E NodeMCU wireless processor and all IGBT driver components mounted on a single 2 X 2.75 inch fiberglass prototyping board. Wired and running eight 24 volt secondaries.

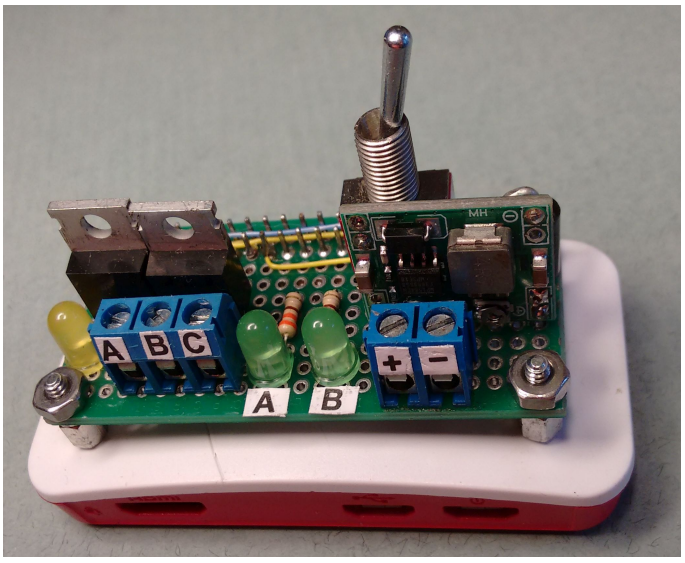


Photo 14: Version 3. Stock Raspberry Pi Zero W, inside it's protective case, is configured and running Linux OS . Driver Hat is plugged into the Zero's 40 pin GPIO connector and will run stand-a-lone with only A, B, C, and power wires attached .

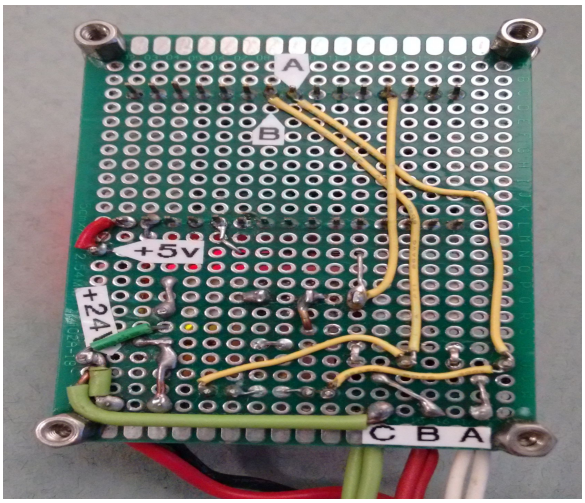


Photo 15: Bottom view of ESP8266-12E NodeMCU, FQP30N06L IGBT drivers, and the 24V DC to 5V DC converter board.

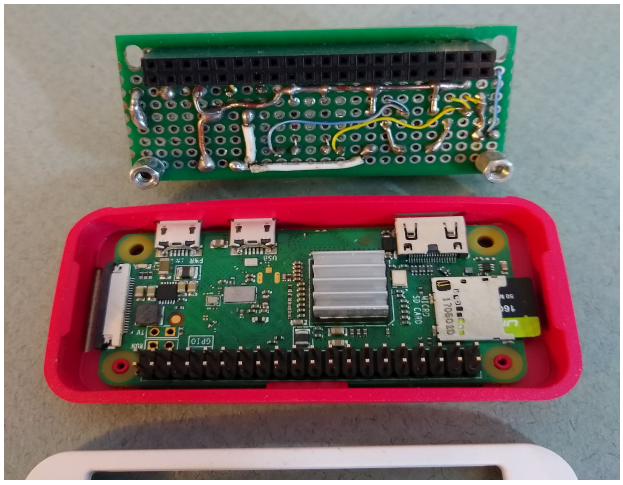
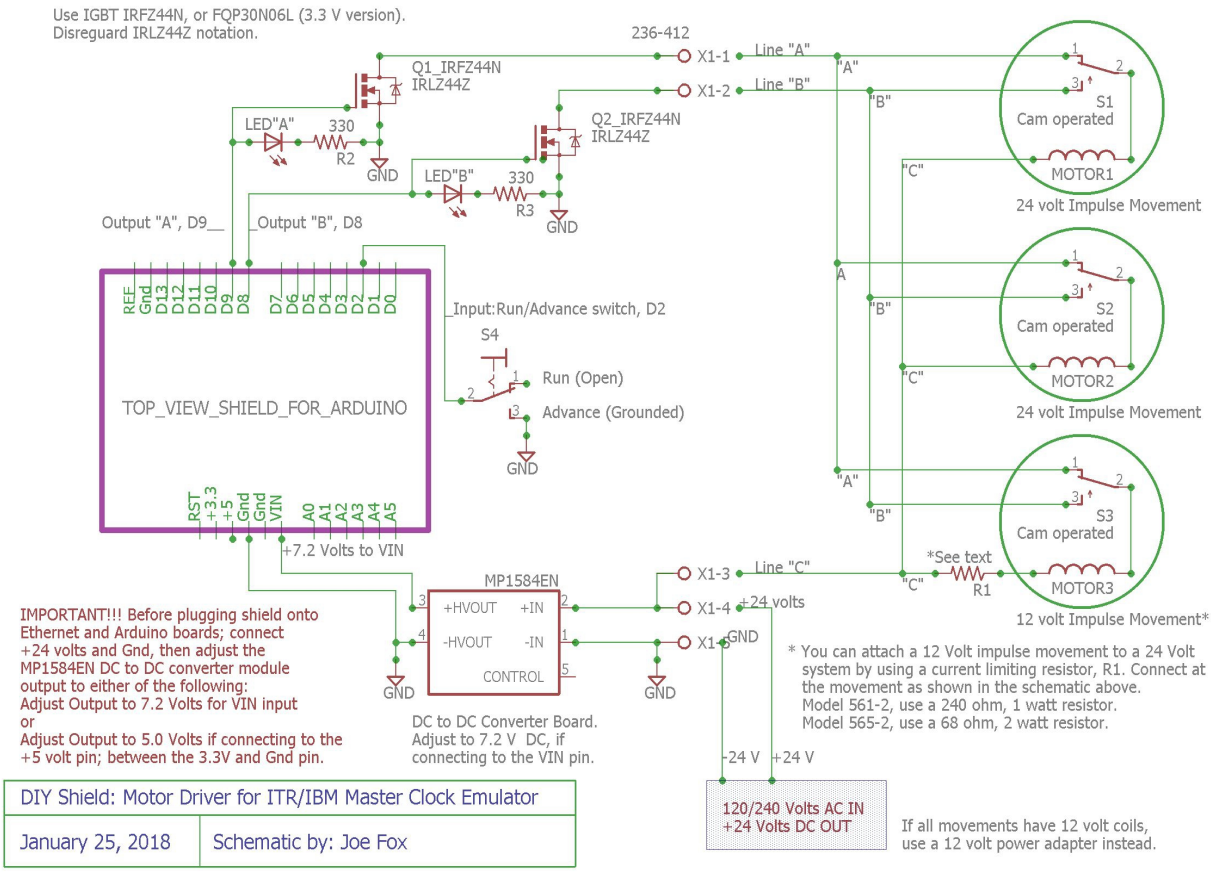


Photo 16: Top of RPi Zero W and bottom view of the Driver Hat of photo 14, showing header configuration. The heat sink, case, pre-loaded memory, and other items come with the Raspberry Pi Zero W in the CanaKit.

To measure a movement's current:

Set the meter switch to A, or amps and remove the red meter lead from the hole labeled V,O,Hz etc, and insert into the 10A hole. **Be very careful here! An Ammeter is a near short to a voltage source (very low resistance).** If you try to measure across a **voltage source** (+24 volts to red, -24 volts to black) with the red lead plugged into the 10A, mA, or uA hole, and the meter switch at (A)mperes, (or mA, or uA,) you will most likely instantly blow the fuse (if there is one) or amp meter circuit shunt resistor that is measuring current! To measure the current, connect the meter's red lead to the + 24 (or +12) volt power source. Place the meter's black lead to the movements green wire (C). This places the meter in series in the circuit.

Now connect the white wire (A) to the (-) negative terminal of the voltage source and read current in amps. Reading should be within 10 percent of the value for the model listed in my table 2. Now take the red lead out of the 10A hole and return it to the VOM hole.

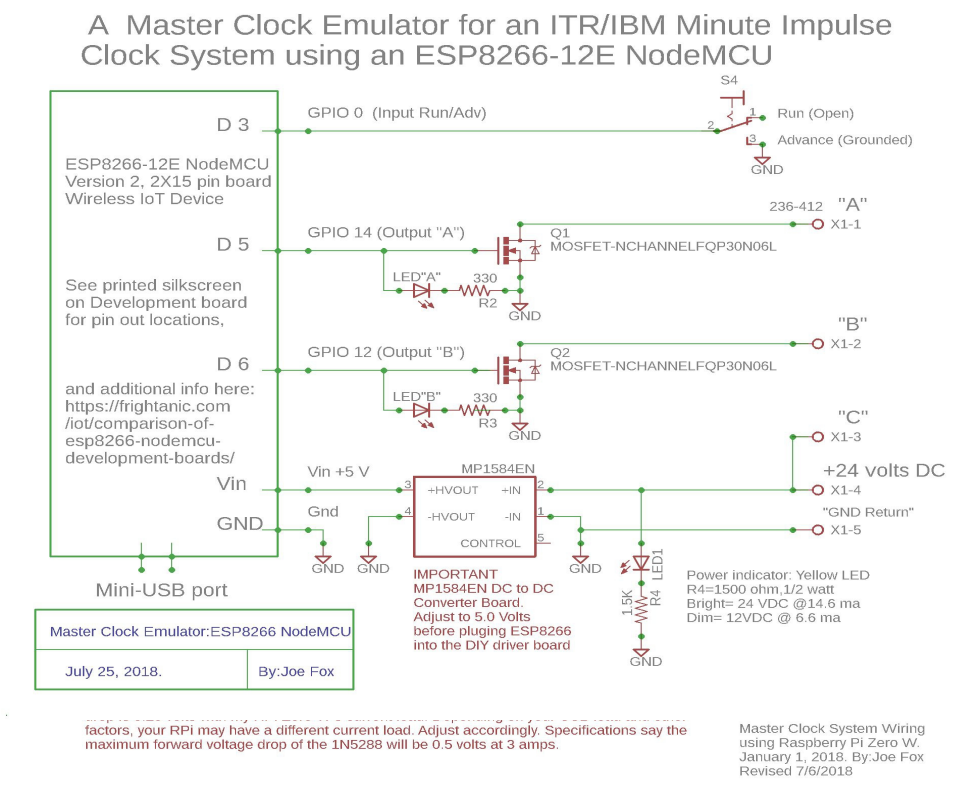


Schematic 2: The Arduino Driver Shield, top view. Note:Everything shown to the right of the terminal screws, including site wiring, 24 volt power adapter, secondary movements, and R1 are external of the shield.

All three [full page] schematics can be found in the Appendix.

Mixing 12 volt and 24 volt coil Secondaries:

If you have both 12 and 24 volt coil secondaries, you can run a 561-2 secondary with a 12 volt coil on a system using 24 volt coil secondaries by simply wiring a 240 ohm, 1 watt resistor in series, between the C wire and the C terminal on the 12 volt secondary. (See Table 2) Please label the modification so you or someone else will know what you did years later. That 12 volt secondary and current limiting resistor will not damage or effect any other attached units. If all your units have 12 volt coils, just substitute the 24 volt power supply with a 12 volt power supply with the appropriate current rating. If you have multiple 12 volt secondaries and one 24 volt secondary then you have two options. Either add a resistor to each of the 12 volt secondaries and run them all at 24 volts; or add a separate power supply and a separate IGBT transistor pair circuit, one set for 12 volts, one set at 24 volts, to their respective secondaries. Just tie each GPIO output pin to the two IGBT gates together (the two A, IGBTs and the two B, IGBTs). I would add the resistor to each secondary and use 24 volts for all secondaries as this will actually improve the armature's responses. For example, I converted my secondary with a 20 volt coil to 24 volts, as noted in table 2 by using a 150 ohm 1 watt resistor (R1) in series with the C wire and C terminal. (See photos 21 and 22)

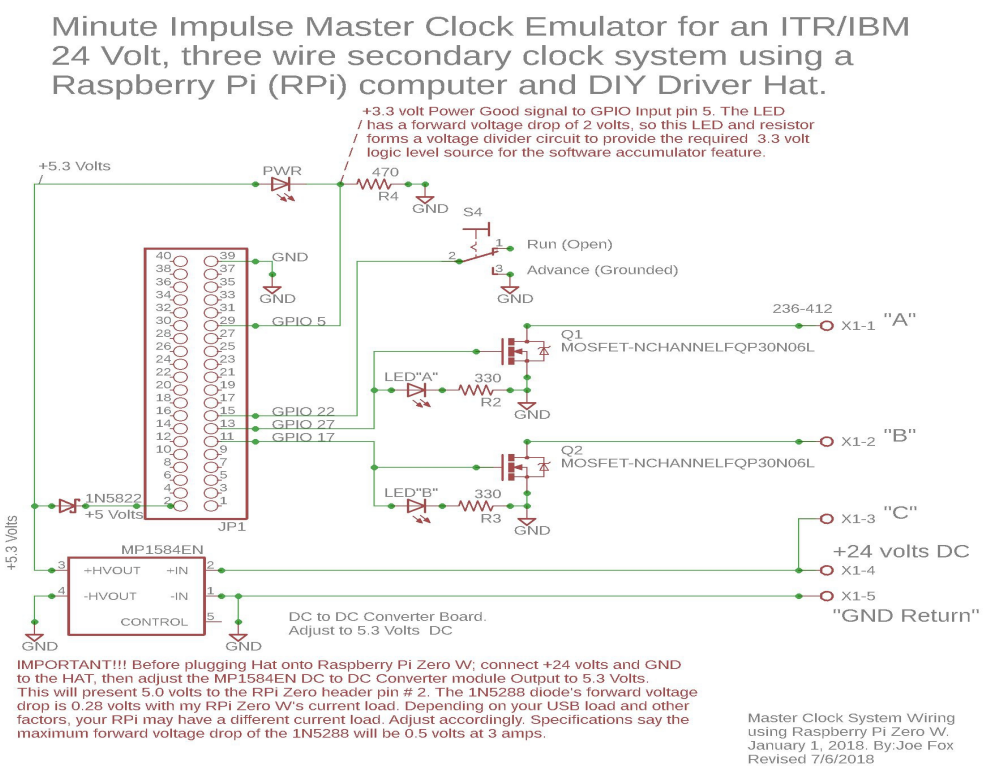


Schematic 3: Version 2, ESP8266-12E NodeMCU wireless WiFi .
Full page schematic in the Appendix

Power Supply Scheme:
A 24 volt DC power source powers the overall master clock system and I recommend a computer notebook style switching power supply, 120/240 volt AC input / 24V dc, 1.5 amperes or more output, depending on the number of secondaries you are driving. This will power the impulse secondary coils and the DC to DC buck switching power supply which supplies power to the processor. This scheme powers all three processor versions of this project. Power for the processor is from an MP1584EN, DC to DC buck voltage converter which comes already assembled except for connection header pins.
It is rated for maximum of 3 amps and only cost me $2.00ea. Nearly the cost of a 7805 linear regulator and bridge rectifier that it replaces, but no heat, no transformer, no additional rectifier, no capacitors, and a much higher allowable input voltage range (6 to 30 volts DC). I connect the input of this voltage converter to the 24 Volts DC supply, then adjust the output to 5.0 (or 5.3, or 7.2) volts, see schematics) before connecting and powering the Arduino and Ethernet shield, ESP8266-12E NodeMCU, or Raspberry Pie Zero W. Changing the input of the converter from 24 volts to 12 volts **does not require readjusting** the 5.0 volt output.

Driver Shield:building notes:
Building your own driver board is not complicated. There are only five connections needed for the Arduino's Shield and ESP8266-12E NodeMCU's development board, and a sixth (power good added) to the Raspberry Pie Zero W's Hat. Two are power and ground and the other three are data connections,

one input and two outputs. To build you own driver board, start by selecting the perforated circuit board and preparing the headers. You can easily find inexpensive perforated circuit boards, both phenolic and fiberglass, on Amazon.com. I have used both types, as shown in the photos, and of the many sizes available I purchased the 2.75 inches X 2 inches size. Both are easily trimmed by scoring with a razor knife and snapping off with a pair of pliers. I prefer “female stackable headers” with long pins as I can trim to size. It is also easier to solder more than one wire to long pins than IC socket pins.



Schematic 4: Version 3, Raspbery Pi Zero W Master Clock emulator.
Full page schematic in the Appendix

I found 20 pin header pins on Sparkfun.com, under “ESP32 Thing Stackable Header Set” @ $1.50/pair. The specifications for the headers are 2.54mm (.100 inches) hole spacing, almost a half inch long (10.5mm), female. You will need four male type header pins too, for the DC to DC voltage converter. Starting [and viewing] at the top side of the Arduino board, look for the silkscreen label for the I/O pins “D0”, through “GND”, you need two 8 pin header strips. See schematics for pin IDs on the ESP8266-12E NodeMCU and Rpi Zero W. Two eight pin headers will work fine since you are only connecting to three of the pins on this side of the shield. The gap between these two eight pin headers serves as a visual and physical key when stacking the three boards together.

For the other side of the shield, starting at I/O pin “A5”, to “A0”, you need to trim a row of six pins. None of these pins will be wired. Then starting at I/O pin “VIN” to “RESET”, you only need 6 pins. Only the “VIN” and “GND” pins are wired on this side of the shield. You can use any convenient “GND” pin to connect the GND wire and there are three “GND” pins available on the Arduino. Two are between the “VIN” pin and the “5V” pin. Using the headers detailed above offers great stability, holding the driver shield onto the Ethernet shield firmly.

Here is a wire gauge example: Solid Cat5 E cable wires are usually 24 gauge wire and is a cheap source of hookup wire. The lower the gauge number the larger the wire. By wiring the GND and power first, you can see clearly all the power supply wires without the interference of other wires. I recommend you run 18 gauge hookup wire as the GND wire from the (-) negative screw terminal to the (-) minus input side of the converter and to both source leads of the two IGBT transistors. Run an 18 gauge insulated (Red) wire from the (+24V) screw terminal to the input (+) plus side of the DC to DC converter board and to the C screw terminal. Except for the IGBT Drain connections, all other GND and connection wires can be as small as 28 gauge.
Run a GND wire to one of the Run/Adv switch's outer leads, to the Arduino's "GND" pin, and to the two LED resistors. Run an insulated wire from the DC to DC plus (+7.3V) output to the Arduino's "VIN" pin. This is the input to the Arduino's 7 to 12 Volt onboard regulator.

*Photo 17: Armature (black object) is shown in the **de-energized** position. Drive pawl on the left, view tip through the hole.*

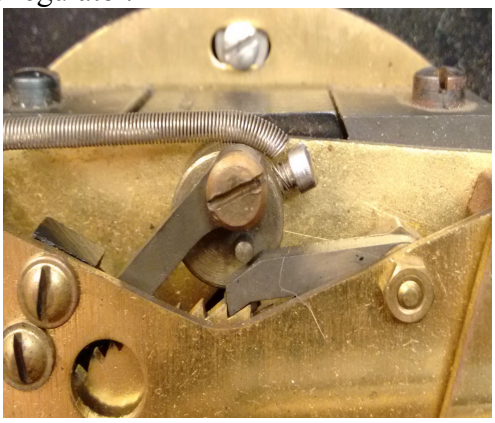


*Photo 18: Armature shown in the **energized** position. Check pawl on the right, under the drive hub's locking pin.*

Type	Year	Resistance	Coil Voltage	Measured current @ 12V	Measured current @ 24V
561-2,	1948	245 ohms	12 Volt	0.049 amps	0.097 amps (Over current!)
561-2,	1930	964 ohms	24 Volt	0.012 amps (Under current)	0.024 amps*
565-2,			12 Volt	*(I do not have one to measure.)*	
565-2,	1955	244 ohms	24 Volt	0.049 amps (Under current)	0.097 amps**

Table 1: *My 24 volt coils connected with 12 volts did not rotate the armature fully. Some may only move a little, if at all.*
** Correct reading per table 2 for this model at 24 volts.*
*** This measured 24 volt reading is 0.012 amps higher than the current listed in IBM's Published Table 2, P9.*

You can connect the +24V and (-) negative DC power supply wires to the screw terminals at this time, then adjust the output of the DC to DC converter to 7.3 volts. Then verify that 7.3 volts is present at the input "VIN" pin, **before connecting the shield** to the Ethernet and Arduino boards.
Referring to the impulse secondary driver circuit as shown in Schematic 2. The Light Emitting Diodes (LED) and their resistors let you know when an Arduino's output is active. Notice on a new plastic LED, there is a long lead and a short lead. The short lead is the cathode and this is connected to one end of a current limiting resistor. If you look at the round ridge at the base of the plastic LED, you should see a flat area. This flat also identifies the cathode. The other side of the 330 ohm resistor (Orange Orange Brown) goes to the GND wire. The long Anode lead is connected to the I/O pin "D8" or "D9", and to the gate of the A or B, IGBT transistor (left lead). Please note: In the photo and not on schematic 2, my DIY driver board shows a "Power ON", Yellow LED and a 1.5K ohm ½ watt resistor (Brown Green Red) for indicating 24 volts present. Anode to 24 volts, cathode to resistor then resistor to GND.

This LED will also illuminate at 12 volts but will be much dimmer. While this Yellow LED is not necessary, I find it very helpful to know if the shield is powered ON.The drain (center lead) of the IGBT is connected to its respective screw terminal A or B. Here, I also recommend you use 18 gauge insulated wire, shown in photo 17 as the white wires, from the A screw terminal to the drain lead of the 'A' IGBT transistor and from the B screw terminal to the drain lead of the 'B' IGBT transistor. See the datasheet on the IRFZ44N for more information on lead identifications

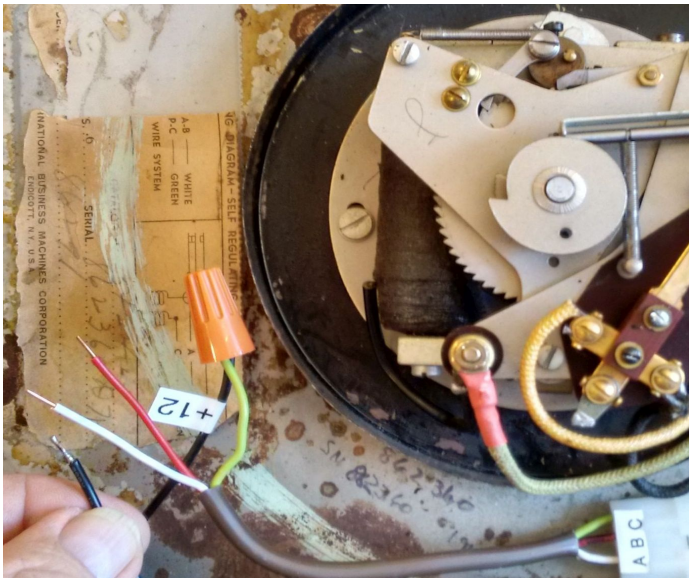


Photo 19: Setup for testing a secondary with a 12 V source. Shown de-energized, this 1948, 12 Volt, Type 561 secondary is vertical with the 12 o'clock position up. The C wire is connected to the + terminal of a 12 Volt DC power adapter and the A wire [white] is not grounded. Tip of the drive paw can be seen through the hole in the rear plate and the check pawl is contacting the gear. (B wire is red on my test adapter)

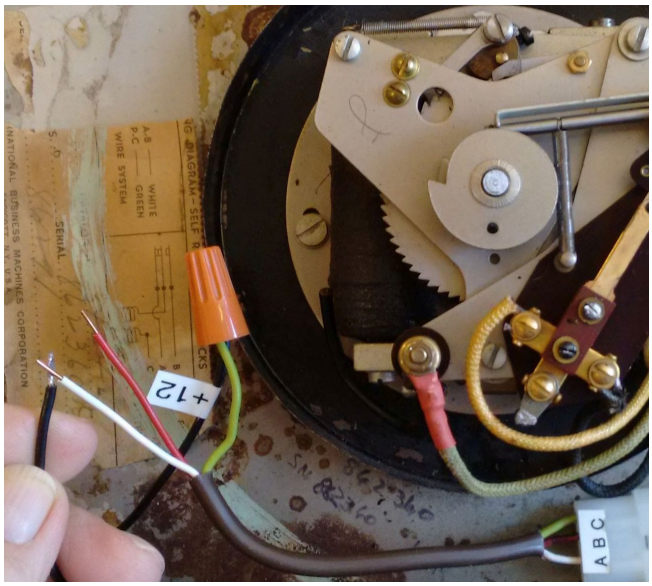


Photo 20: Here the 12 Volt, Type 561- 2 secondary is shown energized, with the C wire connected to the +12 Volt DC power adapter and the A wire [white] grounded. The drive gear is pulled forward after the adapter's GND return wire is removed from the A wire. Notice the original secondary wiring with it's standard color code: A wire is White, B wire is Black, C wire is Green.

Type	Description	12 Volt	24 volt	12V to 24 V Conversion Resistor
25	Master's Winding Magnet	.273 amps	.132 amps	43 ohm 5 watt (actual 3.36 watts)
25	Impulse Accumulator	.280 amps	.140 amps	43 ohm 5 watt (actual 3.36 watts)
805-2	Program Magnet	.545 amps	.279 amps	22 ohm 10 watt (actual 6.54 watts)
561	Indicating Clock Movement	(20V)	*.028 amps.	Use R1, 150 ohm 1 watt resistor to
	(**20 volt coil measured with 20V applied: I=28 mA)*			convert coil from 20 to 24 volts.
561-2	Indicating Clock Movement	.050 amps	.024 amps	240 ohm 1 watt (actual .6 watts)
563-2	Indicating Clock Movement	.333 amps	.171 amps	36 ohm 5 watt (actual 3.9 watts)
565-2	Indicating Clock Movement	.170 amps	.085 amps	68 ohm 2 watt (actual 2.1 watts)
569-2	Tower Clock Movement	.050 amps	.024 amps	240 ohm 1 watt (actual .6 watts)

Table 2: *Table of resistors (R1) required to convert from 12 volt (and 20 volt) coils to 24 volt operation.*

. The LEDs and their limiting resistors at each IGBT gate show me the state of the lines, ON when active. Running the program, you will see the A and B LEDs light each minute, the rapid A pulses beginning at the 59th minute:10 seconds, and the B LED not illuminate (stopped) between the 50-59th minute. The Impulse secondaries are all wired in parallel, A to A, B to B, and C to C to complete the system wiring.

NTP Time Request:
For the Arduino and NodeMCU versions, correct time is acquired from the NTP time servers at boot time and once per hour to update the Arduino's and ESP8266-12E NodeMCU's system timer (system clock). I am not sure when and how often the Rpi requests it's Network Time Protocol (NTP) time update as I believe it is coded into the Linux Operating System. The NTP time request is necessary because neither the ESP8266-12E NodeMCU or the Arduino model's on-board system clocks (crystal or ceramic resonator oscillator circuits) are accurate enough to be a clock time base, not even close. For comparison, the IBM mechanical master clock, with its low expansion coefficient Invar Steel pendulum rod and temperature compensating mercury pendulum weight, like mine, had a guaranteed accuracy of ten (10) seconds per month, and that's back in the 1930s. With today's climate controlled buildings, this tolerance can easily be held to a much tighter margin.

Photo 21: Showing a 1932 type 561-2 with a 20 volt coil, converted to 24V with R1, a 150 Ohm - 1 watt resistor, in series with coil's terminal (brass nut) and the C line. Note the 20 VDC reference on the label.

Photo 22: This 1948 type 561-2 movement with a 12 volt coil is converted to 24 volts by limiting resistor R1, 235 Ohm at 1 Watt. Shown is the resistor mounted in series with the C wire and the coil. Note the 12 V coil marking.

My Arduino's system timers gain or lose about a second per hour with current code [worst case] totaling about 720 seconds per month; that's 12 minutes! As you can see an external time base source is required. In fact, my first attempt on the Uno was 17 seconds slow per hour. So after looking at the GPS module, temperature controlled crystal oscillators, real time clocks (RTC), WWV (radio) modules, and an Internet Time Server as possible sources, I chose the cheapest and most maintenance free over time, the time server. This NTP time server request will correct any error in the processors timer at 58 minutes past the hour, for ever and ever. The Ethernet shield accesses the Wide Area Network (WAN) with an NTP time source request from the official U.S. government's time keepers, the National Institutes of Standards and Technology (NIST). Formally the National Bureau of Standards. Their Internet Time Server is calibrated by an atomic Cesium fountain clock, and is a very accurate and reliable clock time base choice for the Arduino based master. The Arduino's on board system timer (clock) is still used, but its timer is corrected every hour by the NTP time request, a minute before synchronizing all the clock movements at the top of each hour.

In other words, this master clock will never be off more than one or two seconds, regardless of temperature, voltage, crystal aging, etc., either tomorrow or ten years from now. Bear in mind the impulse secondary has a resolution of one minute, not one second. Only the master clock has a resolution of one second. The impulse secondary will only be on time for one second, then will become up to 59 seconds slow until it is brought forward at the next impulse, at zero seconds of the next minute. There is useful information at the National Institutes of Standards and Technology (NIST) at :https://tf.nist.gov/tf-cgi/servers.cgi . Be sure to heed their warning that:
"All users should ensure that their software NEVER queries a server more frequently than once every 4 seconds. Systems that exceed this rate will be refused service..."

Photo 23: Showing my "C" wire to R1 connection before installing heat-shrink tubing insulation over the terminal lugs, screw and nut.

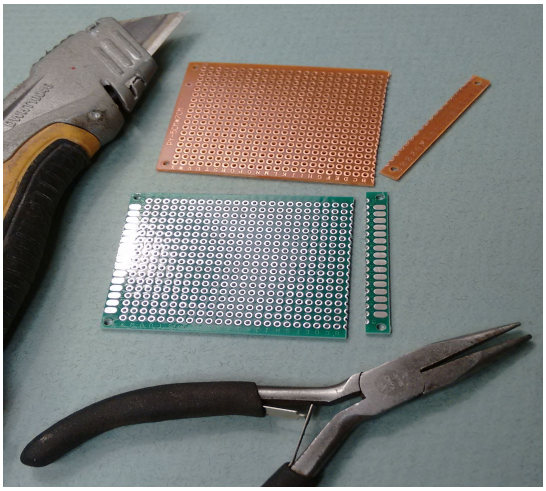


Photo 24: Tools needed to score and snap circuits boards to size.

Master Clock Emulator Version two:
Version two of the master clock emulator uses an ESP8266-12 wireless Internet of Things (IoT) processor. Phil, my son-in-law had given me an Oak that he programmed with the master clock code. The Oak, by DigiStump.com, is a wireless IoT development board that uses an ESP8266-12 wireless processor supporting 802.11 WiFi, and cost significantly less than even the open source versions of the Arduino Uno and Arduino Ethernet shield combination. Plus, not only does it provide access to the internet for the NTP time requests, the ESP8266-12 wireless internet processor, the 24Volt to 5Volt DC to DC converter, two IGBT devices, two LEDs, two limiting resistors, a Run/Adv switch, and screw terminals will all easily fit onto a single 2" X 2.75" perforated circuit board. However the Oak proved to be a great disappointment because of the ongoing issues between the IoT Particle.io cloud service and this device. I also discovered that uploading code to the Oak, Over The Air (OTA), is a nightmare, and very complicated for anyone but an experienced programmer. So after months of aggravation with lost connections and reloading code to get it back on line, I stopped using the Oak in July, 2017, in favor of the ESP8266-12E NodeMCU IoT development board. (Photos 13, 25, and 27)

ESP8266-12E NodeMCU IoT development board:
The NodeMCU that I bought is about half the cost of the Oak, and comes already programmed with code which will scan nearby wireless networks and output their signal strengths. I have not verified this but I believe the notation is in -db, so a -87db signal is twice as strong a signal as a -91 db signal. Uploading a sketch to this device at this time will wipe out this preloaded code. To view the available wireless networks, you do not need to load any code, but to view the serial port output, you will need to load the library link for ESP8266-12 support into the Arduino IDE.

See the Instructables' link and info below. Using basically the same wireless device that the Oak uses, an ESP8266-12E is mounted on a development board with supporting hardware. This NodeMCU Development board is their second version and it has its own, on-board USB to serial converter and a 5 volt input regulator, supplying the 3.3 volt operational voltage needed for the ESP8266-12E processor. So again I use the 24 volt power supply required for the secondary movements to power a DC to DC converter that powers the NodeMCU, at +5 volts. This DC to DC converter is adjusted to 5 volts output (@ 3 amp max) and is connected to the NodeMCU at the Vin header pin and will supply more power than the standard computer USB-2.0 port. The ESP8266 is a radio transceiver and consumes a lot of power. The ESP8266-12 it self uses about 213 ma (@ +19.5dbm) when transmitting, taxing the shared power of a computer's poorly regulated standard USB 2.0 port.

I found a bug while using the ESP8266-12 NodeMCU, where occasionally I would see a double pulse at 58 minutes. We believe the problem is a software race. If the processor's time (internal crystal time base/clock) is fast and advances the secondaries at it's 58 minutes, a fraction of a second before this timer is corrected by the NTP call back to 57.999... seconds (or earlier), then clocks to 58 minutes (again), the processor immediately outputs a second 58th minute pulse. This leaves the secondaries at 59 minutes, not 58 minutes, but one minute fast. The secondaries will now ignore the 59th minute pulse because of the self-correcting feature. Being at their 59th minute, they are only looking for the next “B” signal. The easy solution is: Change the NTP call to the 59th minute, so that only the first pulse of a double pulse will be executed by secondaries that are synchronized. Phil has ported the Arduino master clock sketch code to work with the NodeMCU and that code can be downloaded here as an attachment. At this time, the ESP8266-12E sketch has only been verified to work in the ESP8266-12E NodeMCU. It is possible that other available ESP8266-12 boards will also work if their boards are selected, complied, and uploaded. At $6.50 ea (& free shipping from Amazon.com), I don't have a compelling reason to try a different ESP8266-12 board. The Oak, at $11.00 ea + shipping, had it's own version master clock sketch because the clock software is loaded Over The Air (OTA) via the Particle.io cloud service, and was created a year earlier. The NodeMCU greatly simplifies loading of the sketch compared to the Oak and its complicated uploading procedures. Not to mention its obvious problems with the IoT cloud service. The NodeMCU is designed to be programmed initially via it's included USB to serial converter and using a USB cable connected to a computer. (See photo 25) Attaching a USB cable and executing the Arduino IDE, then selecting the ESP8266-12 NodeMCU in the IDE tool's board manager menu, you can monitor it's output with the Arduino's IDE's serial monitor. Then you can edit and upload your master clock sketch in about a minute. If you only have one access point, only edit the first line. The only thing you need to do differently than performing an Arduino/Ethernet shield version upload, is to load the board manager/ESP8266 library link into the Arduino IDE's add library section as per this “Esp8266 Instructable” at:
https://www.instructables.com/id/Programming-ESP8266-ESP-12E-NodeMCU-Using-Arduino-/

Arduino IDE Setup for ESP8266-12NodeMCU:
Download the Arduino IDE version 1.8.5, for free, from the Arduino's official web site at: https://www.arduino.cc/en/Main/Software. The current version as of 7/2018 is 1.8.5, and should work with the Arduino Uno and Mega as is, but you will need to load the ESP8266 supporting library in order to use it with the NodeMCU development board. For those using the ESP8266-12, I suggest you follow steps 1 through 5, from here:
https://www.instructables.com/id/Programming-ESP8266-ESP-12E-NodeMCU-Using-Arduino-/

There is also a video, that shows all the steps at the beginning of this Instructable. Step 6 and 7 have you connect an LED and select the Blink Example sketch and upload the sketch to the NodeMCU. I did this and it worked. However, if you stop after step 5, connect a new NodeMCU via a USB cable,

select the ESP8266 NodeMCU from the Tools drop-down menu:Boards Manager, and verify the com port, then open the serial monitor; you should see the wireless scan and signal strength of all in range APs. This should verify your unit is working and that you can see your wireless access point. The scan should start displaying within a few seconds. Please disregard the comments at the bottom of this Instructable's web page, of comments saying the “Vin” pin requires 3.3 volts. They are wrong. “Vin” is where you connect +5 volts DC, and is the same connection to the onboard regulator that the USB +5 volts is connected to internally on this board. This +5 volt connection is regulated down to the +3.3 volt operational voltages for the ESP8266-12E. The onboard serial converter level shifts the 5 volt logic levels of the USB data stream both ways. However, access through the GPIO pins **must be** 3.3 volt logic levels and that is why I needed to use the FQP30N06L, an IGBT with it's gate “ON” threshold of 1.8 volts. I had failures using the Oak's 3.3 volt logic levels driving the IRFZ44N, IGBT's.

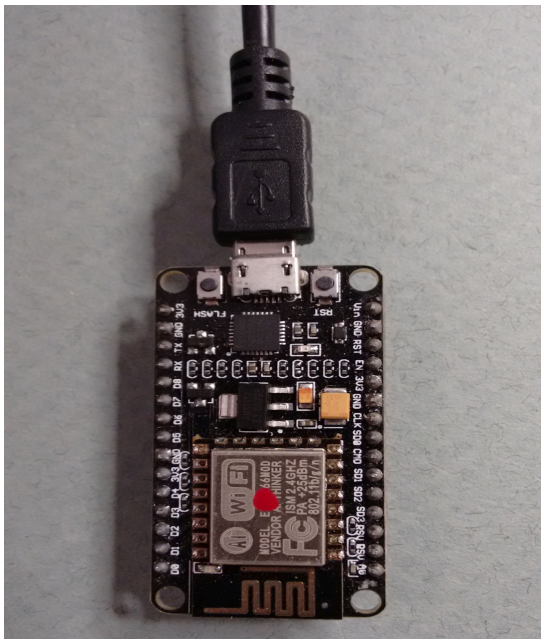


Photo 25:Just use a computer loaded with the Arduino IDE and connect a Mini-USB cable to program the NodeMCU. Program connected as above by laying the device on a non-conductive surface so the exposed pins will not be shorted.

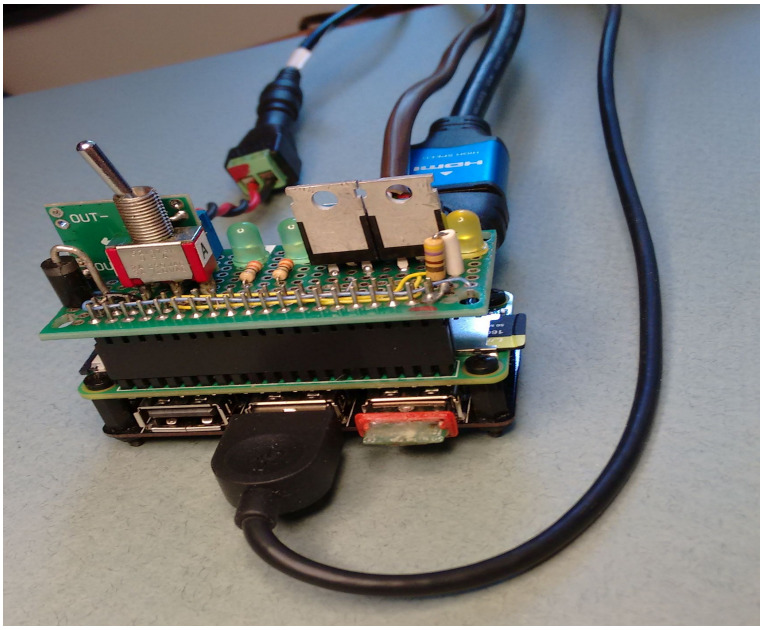


Photo 26: The RPi Zero W needs an HDMI cable and monitor, either a USB Expansion board or a combination USB Keyboard / Mouse, to configure and program. The Canakit starter kit includes cable adapters, power supply, case, memory card, and a 40 pin header. *($35 @ Amazon.com) Shown with a USB expansion board attached.*

Connecting and programming the NodeMCU needs only a USB cable. I suggest you program the unit connected as in photo 25, not plugged into anything but the USB cable and laying on a non-conducting surface to protect the exposedpins. Power will be provided via the USB port. You can now load (open) the Master Clock sketch file into the Arduino IDE, edit the clock sketch's line of code in the Network.cpp file by entering your wireless ID name and password, then compile and upload to the ESP8266. Reboot the ESP8266-12E NodeMCU, open the serial monitor and view the serial output and watch the ESP's onboard LED blink off, every minute. **Before** you plug the NodeMCU into the driver board, you should connect the 24 volt power supply, adjust the DC to DC converter to 5 volts. You are now ready to install the ESP8266-E12 NodeMCU into the DIY driver board and connect the A, B, and C wires to your secondaries. That simple.

You can configure up to three access points. The following is where you enter your Wireless Access Point's (AP) network name and password. Find and open the Network.cpp tab, Then find the following

lines of code.
This line: 26 wifiMulti.addAP("Your_Network_Name", "Your_Password"); //Enter AP Name and Password // here
This line: 27 wifiMulti.addAP("Your_Network_Name", "Your_Password"); //Enter second AP Name and //Password here
This line: 28 wifiMulti.addAP("Your_Network_Name", "Your_Password"); //Enter third AP Name and //Password here

Example :

26 wifiMulti.addAP("**NSA_Surveillance_Van_248**", "**W1ndbytheSea**"); //Enter AP Name and //Password here //Note:Sorry, "W1ndbytheSea", is not my real password.

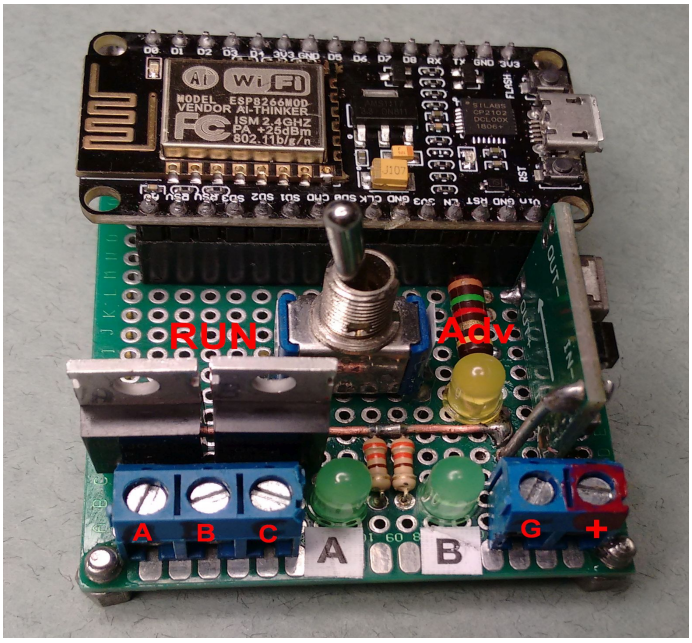


Photo 27: Master Clock Emulator Version 2, featuring the ESP8266-12E NodeMCU. Because of the 3.3 volt logic levels of the ESP8266-12, IGBT drivers with low gate voltage thresholds [like the FQP30N06L] are required.

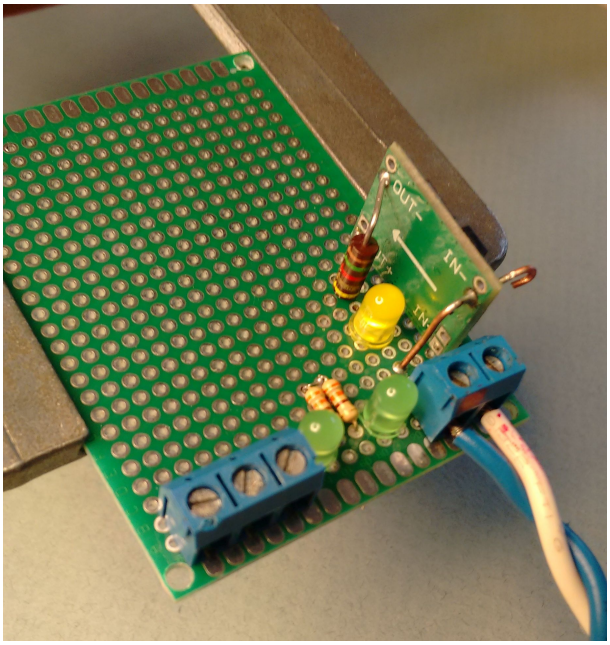


Photo 28: I start by mounting the terminals and DC to DC converter board. Then wire voltage and ground wires. Next I power the converter and adjust the converter's output to 5.0 volts DC.

Version three:Raspberry Pi Zero W:

This version features auto-advance of the secondaries from a power loss, auto-Daylight Savings Time (DST) time changes, and a battery backup scheme, and uses your wireless LAN to obtain the NTP time source for accurate time. The Raspberry Pie Zero W, is a WiFi version of the RPi series, and was released in late 2017. The nice thing about this computer is that it is compact, runs Linux booted from an SD memory card, and supports USB keyboard, mouse, and HDMI video so it is fairly simple to get running. This computer controls secondary movements by simply loading a master clock application like any other computer program and building a DIY driver circuit as described in this article.

While the RPi series offers a choice of several versions of operating systems. This version of the master clock emulator runs Linux Raspbain and loads and executes a group of clock files. The wireless network is configured from the pre-operating system load procedure and then uses the wireless internet connection to load the OS. After you copy the master clock files to the home directory, the install file should be executed from the command line to setup auto-start. This will automatically start the master clock code and will allow for stand-a-long operation; running the Rpi without the keyboard, mouse, and video monitor and requiring only 24 volts, Gnd, and the A, B, and C lines connected.

This install file is only run once and will start the clock program after a boot. The Rpi Zero does take a lot of time to setup the network, load the OS, copy files, and install the master clock program. Especially if you are new to Linux and you start from scratch; formatting a new SD card and loading the NOOBS loader files, then installing the OS, and so on. You can see that the schematic for the Rpi is just about the same as the other two versions, but with the additional input for a power good signal. The battery backup feature requires an additional barrier diode and DC to DC converter as shown in the Battery Backup schematic located in the Appendix. If battery backup is not wanted and you want to eliminate the power good diode and resistor, you must jumper 3.3 volts (header pin 17 is convenient) to header pin 29 (GPIO 5). With or without a resistor. If 3.3 volts is not present at GPIO 5, no pulses will be output. The power good LED is also used as a visual indication to show that the device is powered ON so I would recommend connecting the LED, resistor as shown.

Back-up Power:
See back-up power schematic and notes in the appendix. While backup power can be connected at the +5 volt pin on the Rpi's GPIO header pins, the same place as the barrier diode shown connected in schematic 4, I recommend using the Rpi Zero W's USB power only connector (no USB support here) for the backup battery connection to the Rpi as it may be more conveniently connected and disconnected. A second MP1584EN DC to DC converter and Barrier diode (1N5822) in series is used to provide the back-up power from batteries to the 5 volt power input. This backup power's **converter output** should be adjusted to about 5.25 volts, to accommodate the 1N5822 barrier diode that is in series, similar to the barrier diode scheme used in the primary 24 volt supply. Adjusting the backup converter's output to a slightly lower voltage then the primary converter's output should prevent the battery from supplying power while the primary power is present. I believe the Rpi will operate satisfactorily with 4.9 volts at it's power input connector. If commercial power is lost, the voltage output of the primary DC to DC converter and the power good LED voltage drops to zero, presenting a “low”, level signal at GPIO 05 (See schematic 4). When this input is“high”, (LED On) it is at a 3.3 volt logic level. When GPIO 5 is low, the Rpi, and only the Rpi continues to be powered via the second DC to DC converter and barrier diode that is powered by battery. The secondaries will not advance (no 24 volts) while primary power is out. This second DC to DC converter arrangement allows any battery pack of up to three amps, at any convenient voltage from about 6.3 volts to 30 volts, to be used. The battery pack can also be trickle charged with no effect on the DC to DC converter's output because a voltage swing of 6.3 through 30 volts will not effect the adjusted output voltage of the DC to DC converter.

Even if a battery backup is not used, the Rpi will automatically advance the secondaries after a power loss when it reboots. Be aware, it is general knowledge that you should shutdown any Linux system rather than just unplugging it's power, as it can damage or corrupt it's data should it be writing to memory at the time of a power loss. You will notice that during a reboot, the Raspbain Linux OS performs a type of check disk routine after a power loss. Secondary correction will be performed by the Rpi either with battery backup or no battery. It is just not graceful if the Rpi has to reboot, and you may need to correct the secondaries after it quits self-correcting from an unexpected reboot. You can expect the correction to continue for as long as 20 minutes without battery backup. New software updates should reduce this time. Not a problem with a backup battery attached.

Example of serial output:
(We are talking Arduino output here:)
57:58 **(57 minutes and 58 seconds past the hour)**

57:59
A 58:00 NTP Request **(means A LED ON, @ 58 minutes, and network time request sent)**
NTP: Received 48 byte packet

1C 01 0D E3 00 00 00 10 00 00 00 20 4E 49 53 54 - DD E5 CB C2 00 00 00 00 00 00 00 00 00 00 00 00 DD E5 CB E9 05 B4 D9 4C DD E5 CB E9 05 B4 E1 33

Programmed MAC address=00:AD:BE:EF:FE:EC **(Look here for programmed MAC address)**
Reference clock: Composite: DDE5CBE9
Seconds since Jan 1 1900 = 3722824681
Unix time = 1513835881
NTP time = 05:58:01 UTC
NTP Server: adjusting time by 1 seconds.
off **(off means A LED OFF)**
58:02
58:03
58:04 **(skip to 58 minutes:59 seconds)**
58:59
A 59:00 off ... **(A time off means A LED turned ON, time displayed, then OFF)**
59:09
A 59:10 off **(Then: at 59 minutes, 10 seconds, A LED ON then OFF every other second)**
59:11
A 59:12 off
59:13
A 59:14 off
59:15
A 59:16 off
59:17 ...
59:58
59:59 **(From 59:51 to 59:59 seconds, no more A pulses)**
AB 00:00 off **(Top of the hour, A and B LEDs ON, then OFF, until 50 minutes past the hour)**
00:01
00:02...

You can uncheck the auto-line feed box, at the bottom left of the screen, if you want to look at previous scrolled pages of the display. Typing on the console will enter the commands as mentioned in the console code. If you have more than one Arduino using an Ethernet shield on the same network, you will need to load a unique MAC address into each one. The serial monitor output will print the MAC address programmed during the NTP request. The MAC address is programmed into the Arduino, not the Ethernet shield.

Arduino Software Notes:
Since my preference is in hardware and I sometimes struggle with software, I call on my son-in-law Phil Hord who is something of a software magician. I am not going into detail about the logic of the master clock sketches If you are new to programming and want to learn how to create sketches, you would be better served following the Arduino tutorials on their web site or C+ and Arduino programming courses and articles published in books or on the WEB.

Those more experienced at programming (in C+) should be able to figure out what happens as there are fairly good comments in the code that Phil wrote.
Visit Phil's github repo at https://github.com/phord/master_clock ; for code updates, to get the PC simulator version, to fork the project or to contribute enhancements of your own.

The Arduino software sketch code works on both the Arduino Uno and Mega 2560 model computer boards. Generally, when the Arduino is powered up, it initializes software variables, performs several tasks, then settles into a loop. Provisions are made to control the program from the keyboard via the serial monitor; if the Arduino is plugged into the USB port, com port selected, and serial monitor is activated. But this master clock will run completely by itself if it has a wired internet connection with a DHCP server, is connected as in schematics 2, 3, 4, and is powered on. If you are activating and using the serial monitor from the Arduino IDE programming software, the screen will show the minute:seconds displayed every second and will scroll up the page. At 58 minutes, zero seconds is the A line ON event, the NTP request, the Arduino's programmed MAC address, the NTP 48 byte packet received, system clock correction if any, and the line OFF event will be displayed. Then a display of each minute:second and an A and B event every zero second as programmed, as the loop runs.

This is the code where the MAC address is assigned .The MAC address is programmed into the Arduino, not the Ethernet shield.
(under UDP tab): …

```
#include <Arduino.h>
#include <IPAddress.h>
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>
//#include "myUdp.h"
// Enter a MAC address for your controller below.
// Newer Ethernet shields have a MAC address printed on a sticker on the Ethernet shield
byte mac[] = {
0x00, 0xAD, 0xBE, 0xEF, 0xFE, 0xEC }; //Edit this line if unique MAC address needed. (LSB)
// Example, changing the LSByte from “EC” to “ED”
//will result in a MAC of, “00:AD:BE:EF:FE:ED”
```

*****IMPORTANT**** As of November 2017:

The Network Time Protocol (NTP) code is shown in this excerpt under the NTP tab. The current IP address for one of NIST's time servers is 132.163.97.1, (NIST changed it November, 2017) and its new time server name is time-a-wwv.nist.gov . NIST says: “ The global address time.nist.gov is resolved to all of the server addresses below in a round-robin sequence to equalize the load across all of the servers” . (See http://tf.nist.gov/tf-cgi/servers.cgi) **However this Adruino sketch code does not support name resolution so until then, the NTP access is via coded IP address as noted below.**

See NTP Tab:Arduino IDE

```
#include "Udp.h"
#include "console.h"
#include <ctype.h>
unsigned char timeServer[] = {132, 163, 97, 1}; // time-a-wwv.nist.gov NTP server IP coded this line.
// IPAddress timeServer(132, 163, 97, 2); // time-b-wwv.nist.gov NTP server
// IPAddress timeServer(132, 163, 97, 3); // time-c-wwv.nist.gov NTP server
IPAddress *timeServerAddress = NULL ;
```

unsigned int localPort = 8888; // local port to listen for UDP packets
unsigned int serverPort = 0 ; //...
The following is Phil's description as to how some of his code works.
Phil says;" Here is how the files are laid out:

master_clock.ino
The main Arduino code. It defines special functions that work only on the Arduino.

Clock_
generic.ino
The main clock protocol implementation. It advances the time and performs functions based on the minutes and seconds counters.

console.ino
The console code. This code talks to the serial port to show you the time and activity output. It also reads the keyboard input to allow you to advance the time, set the time, change the clock speed, and so on.

Udp.ino
A simple Udp "abstraction" for the Arduino. This code is only used on the Arduino.

ntp.ino
The Network Time Protocol code. This is separated from the UDP code so it can be used and tested on Phil's PC.
The following files are all "header" files for the real code. They define prototypes for the real functions used later on. Doing this helps the compiler ensure that all the types are correct in the end.
clock_generic.h
console.h
myUdp.h
ntp.h

The NTP time request is setup to occur at 58 minutes after the hour. (*See note about the ESP8266-`12E bug in the ESP8266 NodeMCU section) This way if the Arduino clock is running fast or slow, it will be corrected near the top of the hour, and the updated time will be reflected on the real clock using the A-B signal lines protocol in just a couple of minutes. If there is any inaccuracy in the timer, it should not accumulate much error in only two minutes. You will be interested to see how some of this works. Have a look at "generic_clock.ino". At the end of this file is the "service" routine. This is like the Arduino "loop" function that gets called over and over, for ever and ever. At the start of this function you will see this code:

```
// the service routine runs over and over again forever:
void service() {
consoleService() ;
checkNtp() ;
switch (state) {
default:
case rise:
a = checkA() ;
b = checkB() ;
```

The first two lines of the function call out to the consoleService() and checkNtp() functions. These functions are therefore also called over and over for ever and ever. Now, the interesting part is in checkNtp. This function is contained in the same file.

```
void checkNtp() {
static bool needResponse = false ;
static bool sentRequest = false ;
if ( m == 58 && !sentRequest ) {
// Read the NTP server once per hour
sendNtpRequest() ;
needResponse = true ;
sentRequest = true ;
}
// Get ready to try again on the next hour
if ( m == 2 ) sentRequest = false ;
if ( needResponse ) {
int mm, ss ;
bool success = readNtpResponse( &mm , &ss ) ;
if ( success ) {
p("\nNTP Server: adjusting time by %d seconds.\n" ,
(mm*60 + ss ) - (m*60+s) ) ;
setMinutes(mm) ;
setSeconds(ss) ;
needResponse = false ;
}
}
}
```

Console:

Serial monitor, used when testing a secondary for problems, etc

You can send any of these commands from the serial monitor. On the serial console with the Arduino IDE, you have to press ENTER to send text, but the ENTER does not get sent itself. So after entering a new time value, you have to enter something else (like a ; or . or anything) to get the time to set. The one-character commands just need the ENTER since they work in just one byte.

The Advance/Retard and Force-Pulse commands can be buffered. So you can type "+++++ ENTER" to advance 5 minutes, and you can type "ABABAAAA" to send 2 AB pulses and 4 A pulses.

+ advances the minute counter (internal only)
- decrements the minute counter (internal only)
Z Zeroes the seconds
A Forces an A pulse
B Forces a B pulse
C Forces both A and B at the same time (works just like 'AB')
12:34; Set the minutes and seconds to 12 and 34
1234; Set the minutes and seconds to 12 and 34
12:; Set just the minutes to 12; leave seconds alone
:34; Set the seconds to 34; leave the minutes alone

Changing the seconds (with Z or :34;) also syncs the clock to align with the second-start.

I added a few new console features while testing this out. If you press the left and right angle bracket keys, it will make the clock speed up or slow down by 2x each time.

This is not for running time correction; it is only for testing.

Press ">" to speed the clock up twice as fast.
Press "<" to slow the clock down 1/2 as fast.
Press "N" to trigger the NTP clock synchronization manually “ (Close Quote, program notes)

Program code libraries obtained from the web:

The basic Ethernet Shield Sketch came from: http://arduino.cc/en/Tutorial/UdpNtpClient , and has been modified.

Note: If you have two or more Arduino (SainSmart, etc) Ethernet boards on your LAN, change the MAC address of one (or more) of them [assigned in the NtpClient code] before compiling and uploading to the Arduino, so each will have an unique MAC address.

See:

```
// Enter a MAC address for your controller below.
// Newer Ethernet shields have a MAC address printed on a sticker on the shield
byte mac[] = {
0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
```

Note: the MAC address from the line of code above is: “DE:AD:BE:EF:FE:ED”. Changing the Least Significant Byte (LSB) from ED to EE, EF, or F0 is all that is needed.

Parts :

There are many sources for components for this project. Check for 24 (or 12) volt power supply listing on Amazon. I found a 3 amp power supply for $12.00 (11/2017) and a 5 amp supply even cheaper with Amazon Prime for $10.00. Although it is larger, it can deliver more current. You can search Amazon for the part name (first column below) to find very good prices too. The listings below are for convenience and I have no association with Arduino,LLC, Jameco.com, Amazon.com, Sparkfun.com, or any other supplier other than as a customer. All of Jamesco.com parts listings have a convenient link to data sheets for reference, and a fair price for the components. There is just the nasty shipping charges.

From Jameco.com:

http://www.jameco.com/Jameco/Products/ProdDS/51414.pdf
IRFZ44N P/N 669951 @ $1.25ea N CHANNEL POWER MOSFET, 55V, 49A, TO-220AB
http://www.jameco.com/Jameco/Products/ProdDS/669951IR.pdf
OSTTA024163 P/N 152347 @ $0.69ea Term Block 2 Pos, 5.08mm Solder Thru-Hole 15A
Qty of 3 needed (6 positions total, 5 used).
They offer a 3 Pos too.
https://www.jameco.com/Jameco/Products/ProdDS/152347.pdf
LED P/N 1554161 @ $3.80 pkg 20 Red LED, 15ma forward current.
LED and resistor optional. Use 330 ohm resistor to limit max current to 15ma.
https://www.jameco.com/Jameco/Products/ProdDS/790241.pdf
Resistor, LED P/N 690742 @$0.09 pkg 10 330 ohm ¼ watt resistor to limit LED current to 0.015 amps
http://www.jameco.com/Jameco/catalogs/c142/P39.pdf
Arduino EtherP/N 2152375 @ $37.95 Arduino Ethernet shield w/o Power over Ethernet (PoE)
https://www.jameco.com/Jameco/Products/ProdDS/2152374.pdf

From other vendors:

20 pin Female header pins on Sparkfun.com, “ESP32 Thing Stackable Header Set” @ 1.50/pair

by Areyourshop, IGBT Transistors 5 pack, 5PCS FQP30N06L FQP30N06 60V LOGIC N-Channel MOSFET TO-220 5/$5.95 + free shipping from Amazon.com 7/2018

by Qunqi, DC to DC Buck Voltage converter, 5pack MP1584EN ultra Small DC-DC 3A power Step-Down Adjustable Module Buck Converter adjustable 5/$8.99 (8/2018) on Amazon.com Prime

by BINZET AC to DC 24 Volt 3 Amp Power Supply Adapter Converter Regulator, 5.5mm x 2.1mm DC Plug, 24V 3A 72W , Switching power supply $11.99 with Prime, on Amazon.com 7/2018

by HiLetgo 10pcs 5x7CM FR-4 Universal Breadboard Double Sided Prototype Boards Thickness 1.6mm Price 7/2018: $5.39 Free Shipping for Prime Members @ Amazon.com

by HALJIA $9.99 Ethernet Network Shield Module W5100 Micro SD Card Slot For Arduino UNO and Mega 2560 1280 328. Amazon.com /Prime 11/2017

HiLetgo 2pcs ESP8266 NodeMCU LUA CP2102 ESP-12E Internet WIFI Development Board Open source Serial Wireless Module Works Great with Arduino IDE/Micropython (Pack of 2PCS) by HiLetgo $12.99 Prime /pair @ Amazon.com August, 2018

CanaKit Raspberry Pi Zero W (Wireless) Complete Starter Kit - 16 GB Edition
Price $33.00ea W/free shipping @ Amazon.com August 2018.

References:

Master Clock code, Arduino based and PC based, that is written by Phil Hord at Phil@phord.com , code not obtained from the web or other sources, is granted to public domain, December 25, 2013.
Arduino is a registered trademark of ARDUINO, LLC, Cambridge, MA 02141
IBM, I.T.R, are registered trademarks of International Business Machines, Armonk, NY
Page 16, I.T.R Service Instructions No 230, April 1, 1938, re-printed with permission from:
Reference Desk
IBM Corporate Archives
Route 100/CSB
Somers, NY 10589
914-766-0612
archive1@us.ibm.com

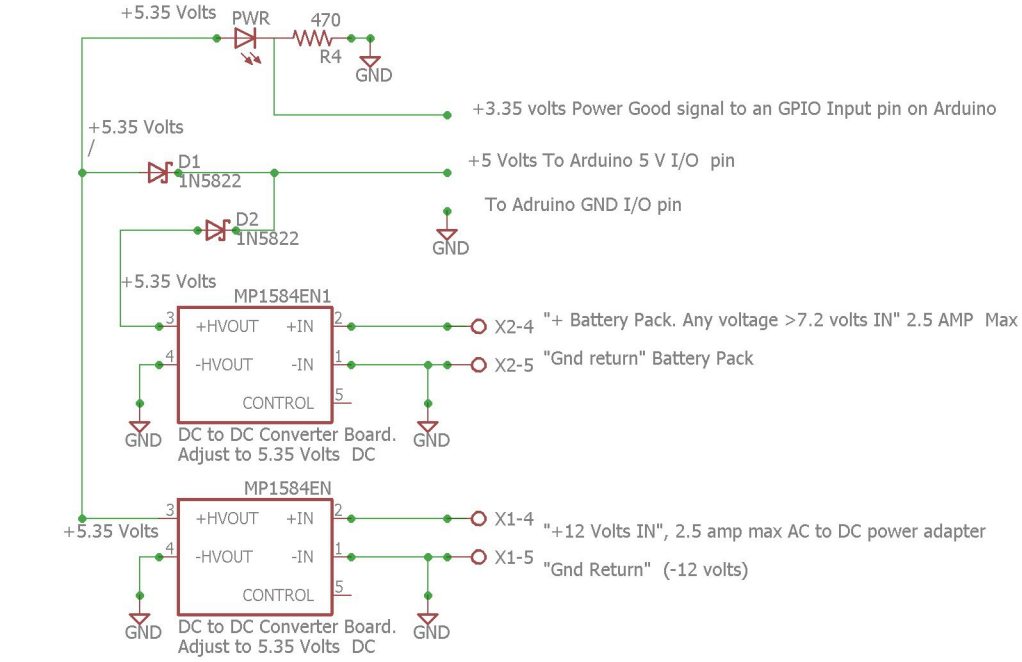
Simplex, Sainsmart, Inland are registered trademarks of their respective companies.

Link to archived first version of article as printed.
https://archive.org/stream/Nuts_and_Volts_No.1_January_2015#page/n23/mode/2up

AT: https://www.deringerney.com/products-and-capabilities/electrical-contact-manufacturing/contact-rivets/ , you will find the following manual.
For contact theory and design information, See Electrical Contact Design Manual . A great resource.

Appendix

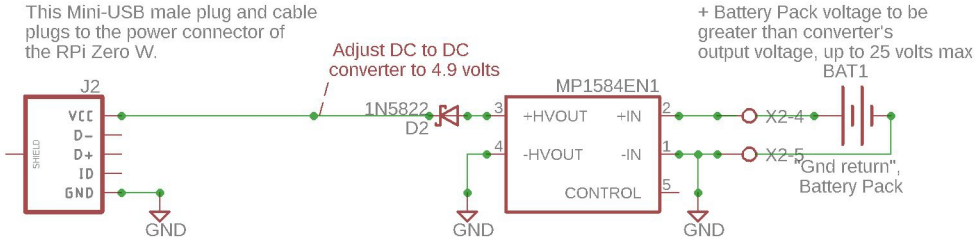
Battery backup circuit for the Arduino.



The 1N5822 barrier diode forward voltage drop depends on the current through it.
Adjust the DC to Dc converters accordingly. Forward voltage drop is .5 volts @ 3 amps (Max voltage/Max current)

Back-up power for Arduino
January 29, 2018 By Joe Fox

Battery Backup Circuit for Raspberry Pi Zero W system.

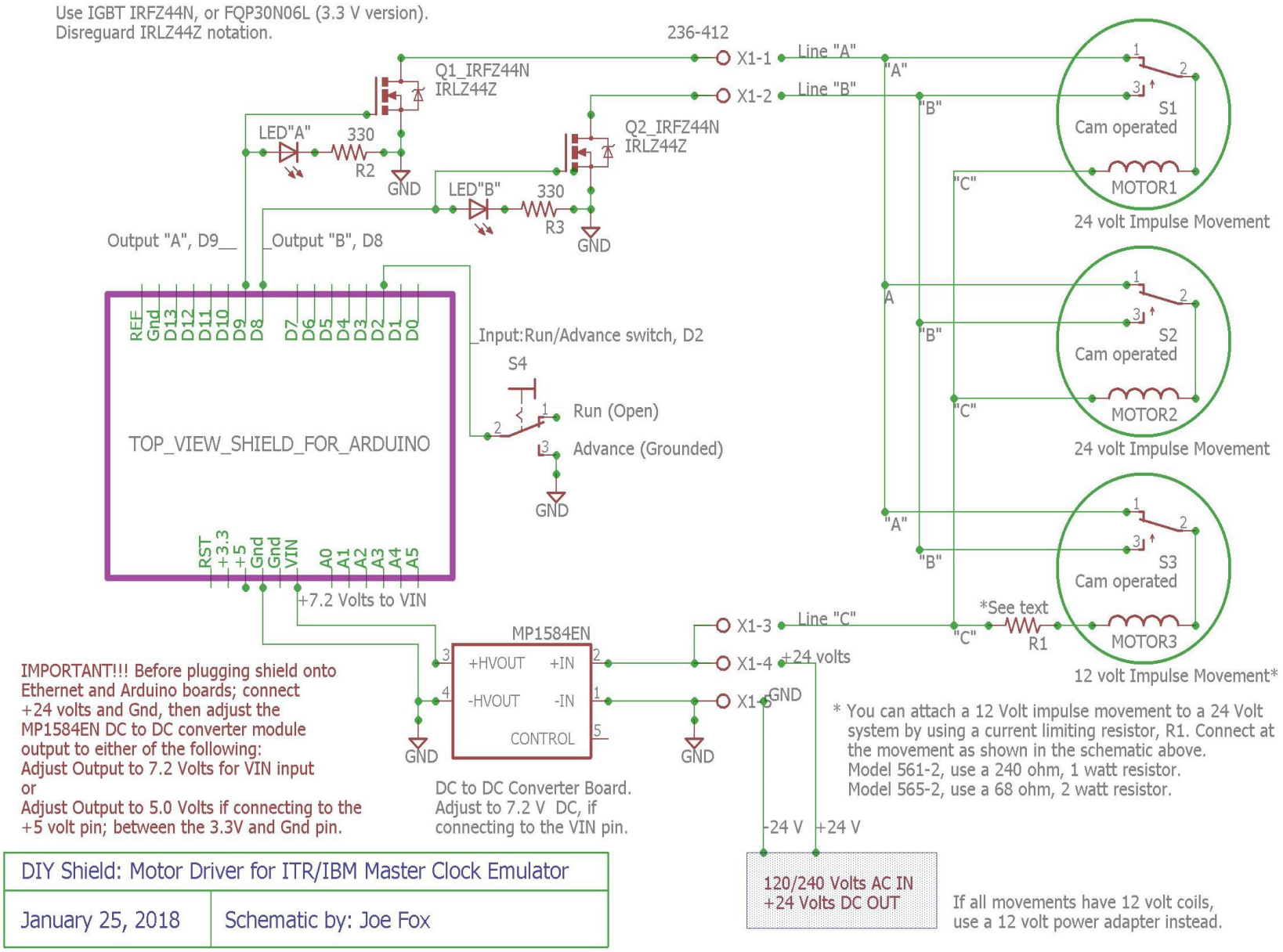


Note: Adjust DC to DC converter board to 5.2 Volts out, or 4.9 volts after the 1N5822 Barrier Diode. This voltage should be lower than the primary supply voltage so that the RPi Zero W's load current is from the primary source if active, not the battery. The 1N5822 barrier diode's forward voltage drop depends on the current through it. Forward voltage drop is .5 volts @ 3 amps (Max voltage/Max current). Adjust the DC to Dc converter accordingly.

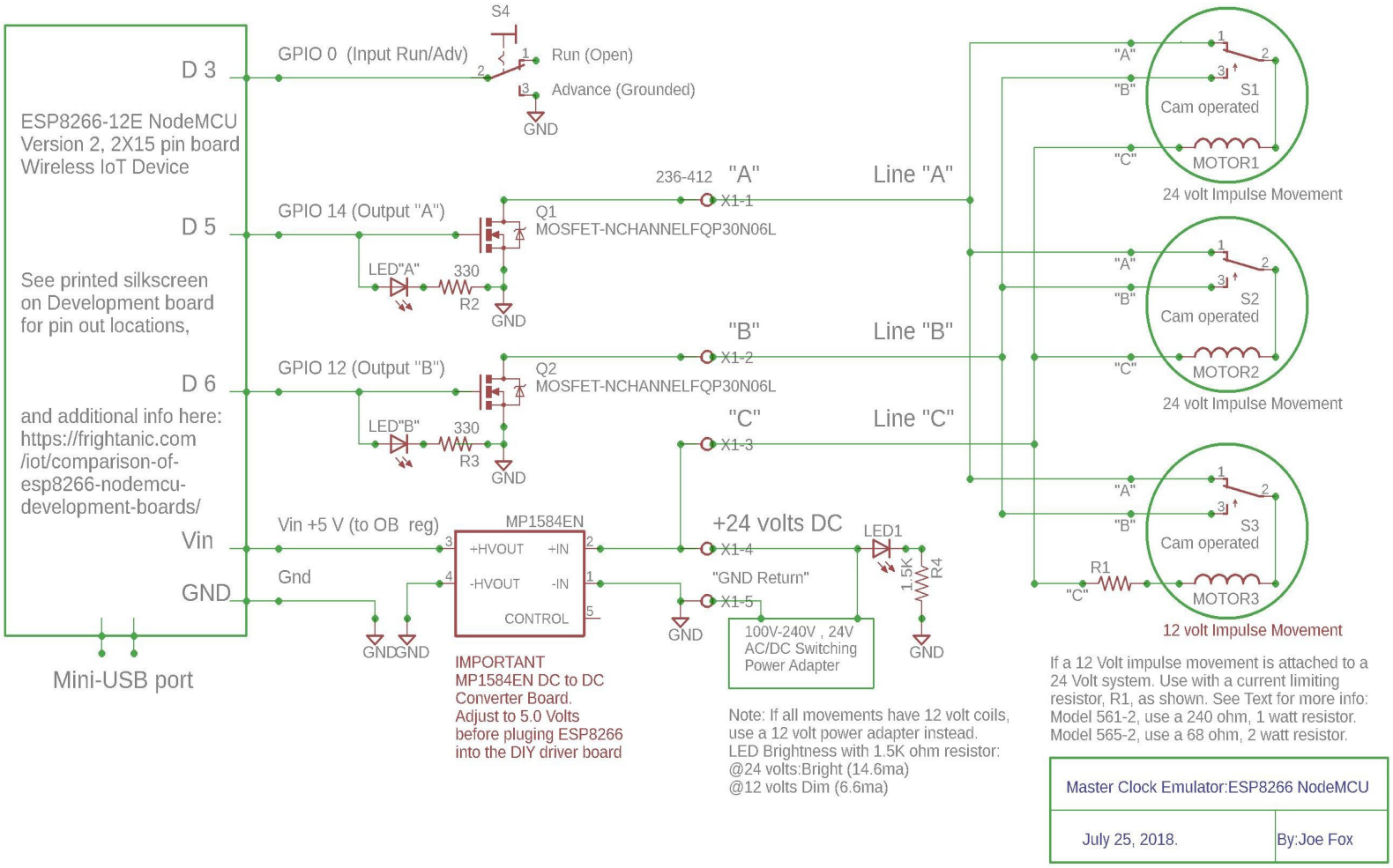
Back-up power for Raspbery Pi Zero W
July 6, 2018 By Joe Fox
Revised November 16, 2018

Battery Backup:
Here is a schematic of my battery backup scheme for the Arduino and Raspberry Pi Zero W. These schematics show attaching the backup battery pack to an Arduino and Rpi Zero W , which includes barrier diodes for steering currents. This was a response to a post question on battery backup ideas for the Arduino systems. My Arduino master clock version does not have software that supports battery backup auto-correction of secondaries except for the normal (less than 50 minute) self-correction of an IBM master clock system. There is no currently assigned pin for “power good” for the Arduino master clock version at this time.

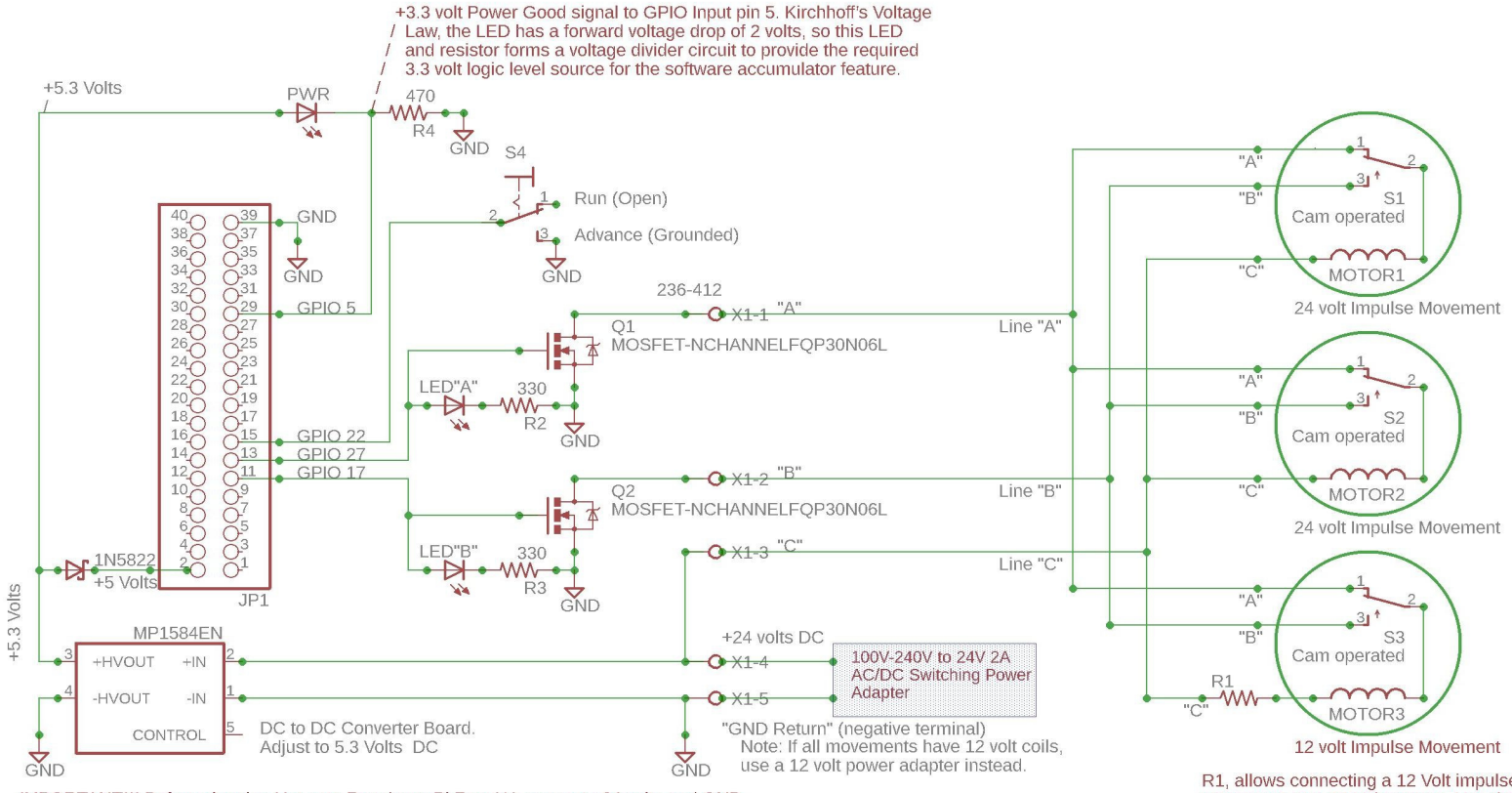
For the Raspberry Pi Zero W Master clock, The “back-up” DC to DC converter's output is connected via 1N5822 Barrier diode and mini-USB cable for connecting to the RPi's USB power connector. The converter's output should be adjusted to 5.25 volts before the diode or 4.9 volts after the diode, slightly lower then the primary DC to DC converter. This version of the Rpi zero W's master clock software does support auto correction of the secondaries, auto correction of daylight saving time, and uses a GPOI pin assigned for power loss sensing. I strongly recommend you adjust the DC to DC converter before connecting the Rpi to this output. See the Rpi schematic.


Use IGBT IRFZ44N, or FQP30N06L (3.3 V version).
Disreguard IRLZ44Z notation.
236-412
X1-1
X1-2
Line "A"
Line "B"
Q1_IRFZ44N
IRLZ44Z
Q2_IRFZ44N
IRLZ44Z
LED"A"
330
R2
GND
LED"B"
330
R3
GND
Output "A", D9__
_Output "B", D8
REF Gnd D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
TOP_VIEW_SHIELD_FOR_ARDUINO
RST +3.3 +5 Gnd Gnd VIN A0 A1 A2 A3 A4 A5
_Input:Run/Advance switch, D2
S4
Run (Open)
Advance (Grounded)
GND
+7.2 Volts to VIN
MP1584EN
+HVOUT
-HVOUT
+IN
-IN
CONTROL
GND
GND
X1-3
X1-4
X1-5
Line "C"
+24 volts
GND
"A"
"B"
"C"
S1
Cam operated
MOTOR1
24 volt Impulse Movement
A
"B"
"C"
S2
Cam operated
MOTOR2
24 volt Impulse Movement
"A"
"B"
"C"
*See text
R1
S3
Cam operated
MOTOR3
12 volt Impulse Movement*
IMPORTANT!!! Before plugging shield onto
Ethernet and Arduino boards; connect
+24 volts and Gnd, then adjust the
MP1584EN DC to DC converter module
output to either of the following:
Adjust Output to 7.2 Volts for VIN input
or
Adjust Output to 5.0 Volts if connecting to the
+5 volt pin; between the 3.3V and Gnd pin.
DC to DC Converter Board.
Adjust to 7.2 V DC, if
connecting to the VIN pin.
* You can attach a 12 Volt impulse movement to a 24 Volt system by using a current limiting resistor, R1. Connect at the movement as shown in the schematic above.
Model 561-2, use a 240 ohm, 1 watt resistor.
Model 565-2, use a 68 ohm, 2 watt resistor.
-24 V
+24 V
120/240 Volts AC IN
+24 Volts DC OUT
If all movements have 12 volt coils,
use a 12 volt power adapter instead.
DIY Shield: Motor Driver for ITR/IBM Master Clock Emulator
January 25, 2018
Schematic by: Joe Fox

A Master Clock Emulator for an ITR/IBM Minute Impulse Clock System using an ESP8266-12E NodeMCU Development board and an IGBT Driver circuit.



A Computerized Minute Impulse Master Clock System: Complete wiring for an ITR/IBM 24 Volt, three wire secondary clock system using a Raspberry Pi (RPi) computer and DIY Driver Hat.



IMPORTANT!!! Before plugging Hat onto Raspberry Pi Zero W; connect +24 volts and GND to the HAT, then adjust the MP1584EN DC to DC Converter module Output to 5.3 Volts. This will present 5.0 volts to the RPi Zero header pin # 2. The 1N5288 diode's forward voltage drop is 0.28 volts with my RPi Zero W's current load. Depending on your USB load and other factors, your RPi may have a different current load. Adjust accordingly. Specifications say the maximum forward voltage drop of the 1N5288 will be 0.5 volts at 3 amps.

R1, allows connecting a 12 Volt impulse movement to a 24 Volt system. Use the following values of R1, for the models noted. See text for more infomation.
Model 561-2, use a 240 ohm, 1 watt resistor.
Model 565-2, use a 68 ohm, 2 watt resistor.

The Screw terminals X1-1 through X1-5 and all components to the left are located on the DIY Driver Hat circuit board.
Everything to the right of the screw terminals, including site wiring, secondaries, power adapter, and R1, if used, are external.

Master Clock System Wiring
using Raspberry Pi Zero W.
January 1, 2018. By:Joe Fox
Revised 7/6/2018